Poglavje 1

Modeliranje računalniških omrežij z orodjem OMNeT++

Orodje OMNeT++ predstavlja odprtokodno simulacijsko ogrodje, ki temelji na diskretnemu proženju simulacijskih dogodkov [1, 2, 3]. Orodje je sestavljeno iz knjižnic napisanih v programskem jeziku C++, razvojnega okolja (angl. *Integrated Development Environment*) in okolja za poganjanje simulacij. OMNeT++ nudi osnovna orodja za postavljanje simulacijskih modelov, njihovo simuliranje in zbiranje ter analizo simulacijskih rezultatov. OMNeT++ torej ni klasični simulator temveč ponuja infrastrukturo za pisanje in poganjanje simulacij. Poleg osnovnega ogrodja je bilo v okviru drugih projektov kot je npr. INET [4] razvitih že veliko število paketov z že izdelanimi simulacijskimi moduli. Ker je orodje OMNeT++ zasnovano modularno, lahko že razvite simulacijske module ponovno uporabimo v svojih simulacijskih modelih. Obstoječe module lahko po eni strani uporabimo nespremenjene, po drugi strani pa je možno njihovo delovanje prilagoditi uporabnikovim potrebam. Pričujoče poglavje predstavlja osnove orodja OMNet++ in delo z njim demonstrira na preprostih zgledih s področja modeliranja računalniških omrežij.

1.1 OMNeT++ osnovni gradniki in njihovo delovanje

Najvišja komponenta simulacijskega modela je t.i. *omrežje* (angl. *network*), ki je sestavljeno iz *modulov* in povezav med njimi. Izgradnja simulacijskega modela poteka na medsebojnemu povezovanju modulov, pri čemer le-ti komunicirajo s pošiljanjem *sporočil* (angl. *messages*). Uporabnik lahko pri izgradnji svojega modela uporablja že obstoječe module ali pa ustvari svoje. Osnovni simulacijski moduli so na voljo že znotraj orodja OMNeT++, veliko število modulov pa je

dostopnih na spletu. Primer zbirke takih modulov, ki implementirajo funkcionalnosti različnih omrežnih protokolov kot so na primer UDP, TCP, IP in IPv6, je ogrodje INET (angl. *INET Framework*) [4]. V orodju OMNeT++ obstajata dve skupini modulov, in sicer *enostavni* (angl. *simple modules*) in *sestavljeni moduli* (angl. *compound modules*). Sestavljeni moduli vsebujejo enega ali več enostavnih ali drugih sestavljenih modulov, pri čemer število gnezdenj ni omejeno. Za enostavnimi moduli stoji izvedljiva C++ programska koda, ki določa njihovo funkcionalnost. Enostavni moduli izhajajo iz razreda `cSimpleModule`.

Moduli med seboj komunicirajo s pošiljanjem *sporočil* (angl. *messages*) posredno preko *vrat* (angl. *gates*) in *povezav* (angl. *connections*) ali celo neposredno med moduli(redko). Obstajajo trije tipi vrat, in sicer *vhodna* (angl. *in*), *izhodna* (angl. *out*) in *dvosmerna* (angl. *inout*). Vrata so med seboj povezana preko povezav, ki jim lahko določamo posamezne lastnosti kot so *zakasnitev* (angl. *delay*), *propustnost* (angl. *data rate*) in *delež napak* (angl. *bit error rate*). Sporočila lahko poleg *časovne oznake* (angl. *timestamp*) vsebujejo poljubne podatkovne strukture.

Modulom lahko pripadajo *parametri* (angl. *parameters*), ki se uporabljajo za določanje delovanja preprostih modulov in nastavljanje topologije modelov. Osnovni gradniki in njihovo delovanje so razloženi v nadaljevanju poglavja.

Opisovanje simulacijskega modela v orodju OMNeT++ poteka na več nivojih z različnimi jeziki. V ta namen se uporabljajo različni datotečni tipi, in sicer:

- NED datoteke: z njimi opisujemo topologijo in komponente modela.
- C++ datoteke: vsebujejo programsko kodo, ki opisuje delovanje preprostih modulov in kanalov. Ločimo dva tipa C++ datotek, in sicer t.i. *zglavne datoteke* (angl. *source files*) s končnico `h`, ki vsebujejo deklaracije lastnosti in metod in *izvorne datoteke* (angl. *source files*) s končnico `cc`, ki vsebujejo implementacijo metod in s tem določajo delovanje komponent.
- INI datoteke: predstavljajo konfiguracijske datoteke, ki določajo nastavitve za izvajanje simulacij. Ponavadi je to zgolj ena datoteka, in sicer `omnetpp.ini`.

1.2 Opisovanje v jeziku NED

Jezik NED (*Network DEscription*) služi opisovanju topologije in komponent modela. Podajamo ga v datotekah s končnico `ned`. Z njim ne določamo samega delovanja komponent, ampak zgolj njihovo strukturo in lastnosti. Jezik NED se najpogosteje uporablja za definicije omrežij, definicije sestavljenih modulov in deklaracije enostavnih modulov.

Na začetku vsake NED datoteke določimo enolično ime gradnika, ki ga datoteka predstavlja na sledeč način:

- `network ime_omrezja`: za omrežje,
- `module ime_modula`: za sestavljen modul,

- `simple ime_modula`: za enostaven modul.

Samo strukturo in lastnosti gradnika določamo znotraj različnih sekcij NED opisa. Pomembnejše sekcije, ki jih lahko uporabimo v NED datotekah so razložene v nadaljevanju.

1.2.1 Parameteri

Sekcija `parameters` služi določanju parametrov komponentam. Vsakemu parametru dodelimo enolično ime, podatkovni tip in opcijsko njegovo privzeto vrednost ali porazdelitev. Parametru lahko določimo `volatile` način delovanja, kar pomeni, da se vrednost parametra evaluira vsakič, ko simulacijsko okolje parameter zahteva. Funkcionalnost `volatile` je pomembna takrat, ko parameter nima fiksne vrednosti, ampak zanj podamo določeno porazdelitev. Primer sekcije `parameters`:

```
parameters:
 // parameter a tipa integer s privzeto vrednostjo 1
 int a = default(1);

 // volatile parameter b tipa double
 // porazdeljen uniformno na intervalu [0,3]
 volatile double b = default(uniform(0,3);

 // parameter c brez privzete vrednosti
 int c;
```

1.2.2 Podmoduli

Sekcija `submodules` služi določanju vgnezdenih modulov. Podmodule lahko določamo le v NED datotekah omrežij in sestavljenih modulov (t.j. modulov, ki niso enostavni). Pri vsakem podmodulu določimo njegovo ime (ime modula, ki že obstaja v našem projektu) in enolično labelo. Ker ima vsak podmodul svojo labelo, lahko uporabimo več instanc istega modula. Primer sekcije `submodules`:

```
submodules:
 computer1: computer {
  @display("p=50,70");
 }
 computer2: computer {
  @display("p=202,70");
 }
 switch: switch {
  @display("p=386,70");
 }
```

Pri tem lastnost `@display` določa izgled gradnika v grafično uporabniškem vmesniku (GUI) (glej poglavje 1.2.7).

1.2.3 Vrata

Sekcija `gates` služi določanju vrat enostavnih in sestavljenih modulov (NED datoteke omrežij vrat nimajo). Pri vsakih vratih podamo njihov tip (`input`, `output` ali `inout`) in enolično ime. Definiramo lahko tudi vektorski tip vrat. Primer sekcije `gates`:

```
gates:
 // izhodna vrata o1
 output o1;

 // vektorska izhodna vrata o2 velikosti 3
 output o2[3];

 // vhodna vrata i1
 input i1;

 // vektorska vhodna vrata i2
 // velikost se bo dolocila kasneje
 input i2[];
```

1.2.4 Tipi

Sekcija `types` služi deklaraciji tipov, ki jih lahko uporabljamo znotraj NED datoteke. Primer sekcije `types` znotraj katere deklariramo kanal (glej razdelek 1.2.6):

```
types:
 channel C extends ned.DatarateChannel {
  datarate = 100Mbps;
 }
```

1.2.5 Povezave

Sekcija `connections` služi določanju povezav med vgnezdenimi moduli. Povezave lahko določamo le v NED datotekah omrežij in sestavljenih modulov (t.j. modulov, ki niso enostavni). Pri vsaki povezavi podamo

- vrata, ki so preko povezave med seboj povezana: `ime_modula.ime_vrat`,
- usmerjenost povezave: `-->`, `<--` ali `<-->`,
- če želimo, da povezava vsebuje kanal (glej razdelek 1.2.6), le-tega podamo znotraj povezave: `--> ime_kanala -->`.

Primer sekcije `connections`:

```
connections:
 // lokalna vrata nimajo predpone
```

```
// vektorska vrata referenciramo z oglatimi oklepaji
i1 --> computer1.o2[0];

computer1.o2[1] <-- i2[0];

// vektorska vrata lahko avtomatsko povecamo z ++
i2++ --> computer1.o2[2];

// povezava dveh vrat, ki niso lokalna
computer2.i1 <-- computer1.o1;

// povezava dveh vrat preko kanala C
switch.o1 --> C --> computer2.i2++;
```

1.2.6 Kanali

Kanali niso sekcije same po sebi, ampak v sekcijah zgolj nastopajo. S pomočjo kanalov lahko določamo lastnosti povezav med moduli. V ta namen lahko uporabljamo tri tipe kanalov, ki so razloženi v nadaljevanju. Za primer uporabe kanalov znotraj povezave glej razdelek 1.2.5, za primer deklaracije kanala kot tip pa razdelek 1.2.4.

Kanali tipa `IdealChannel`

Kanali tipa `IdealChannel` se uporabljajo brez parametrov in služijo simuliranju idealnega prenosa paketov, t.j. brez napak, brez zakasnitev in z neskončno propustnostjo. Če v povezavi ne podamo kanala, se privzeto uporabi idealni kanal.

Kanali tipa `DelayChannel`

Kanali tipa `DelayChannel` se uporabljajo za simuliranje zakasnitev povezave in predstavljajo razširitev tipa `IdealChannel`. Modelirana zakasnitev se pri prenašanju običajnih sporočil ne upošteva. Upošteva se le pri prenašanju paketov (glej razdelek 1.3.1). Tem kanalom lahko določamo dve lastnosti, in sicer:

- `delay`: določa zakasnitev v enotah `s`, `ms`, `us`,...
- `disabled`: ali je kanal omogočen - če postavimo to lastnost na `True`, bo kanal zavrgel vsa sporočila.

Kanali tipa `DatarateChannel`

Kanali tipa `DatarateChannel` predstavljajo razširitev tipa `DelayChannel` in dodatno omogočajo simuliranje pasovne širine povezave in verjetnost napake paketov pri prenosu. Modelirana pasovna širina in verjetnost napake se pri prenašanju običajnih sporočil ne upošteva. Upošteva se le pri prenašanju paketov

(glej razdelek 1.3.1). Kanalom tipa `DelayChannel` lahko dodatno določamo tri lastnosti, in sicer:

- `datarate`: določa pasovno širino kanala v enotah `Kbps`, `Mbps`,... Pasovna širina se upošteva pri računanju časa prenosa paketa in se prišteje zakasnitvi pri prenosu (angl. *delay*). Vrednost 0 pomeni neomejeno pasovno širino.

- `ber` (*Bit Error Rate*): določa verjetnost napake na nivoju enega bita. Lahko zavzame vrednosti znotraj intervala $[0, 1]$. Če simulacijsko okolje ugotovi, da je prišlo do napake, postavi zastavico `Error flag` na nivoju paketa.

- `per` (*Packet Error Rate*): določa verjetnost napake na nivoju enega paketa. Lahko zavzame vrednosti znotraj intervala $[0, 1]$. Če simulacijsko okolje ugotovi, da je prišlo do napake, paketu postavi zastavico `Error flag`.

1.2.7 Lastnosti

Kot kanali, tudi lastnosti niso sekcije same po sebi, ampak v sekcijah zgolj nastopajo. Z njimi lahko v NED datotekah določimo vrednosti lastnostim (angl. *properties*) posameznim objektom (modulom, parametrom, vratom, povezavam itd.). Lastnosti referenciramo z oznako `@`, za katero podamo ime lastnosti, ki ji sledi njena vrednost znotraj oklepaja. Primeri pogostejše uporabljenih lastnosti so:

- `@display`: določa izgled gradnika v GUI. Primer uporabe: `@display ("i=block/browser");`

- `@unit`: določa enote parametra. Primer uporabe: `double startTime @unit(s) = default(0 s);`

- `@statistic`: za beleženje statistike preko signalov (glej poglavje 1.5).

1.3 Programiranje v jeziku C++

Delovanje preprostih modulov in kanalov določimo v jeziku C++. V tem delu bo opisano zgolj programiranje preprostih modulov, saj podrobnosti programiranja kanalov presegajo obseg tega dela. Programiranje kanalov je med drugim opisano v [3].

Programiranje v orodju OMNeT++ temelji na dogodkovnem proženju metod - t.i. dogodkovno vodeno programiranje (angl. *event-driven programming*), pri čemer so glavni dogodki vezani na sporočila, ki jih modul dobi bodisi od sebe bodisi od drugih modulov. Delovanje posameznega modula opisujemo z dvema datotekama, in sicer

- datoteka `ime_modula.h`: vsebujejo deklaracije metod in lastnosti ter reference na knjižnice. Vedno je potrebno vključiti knjižnico `<omnetpp.h>`.

- datoteka `ime_modula.cc`: vsebuje implementacijo metod. Datoteka mora vključevati datoteko `ime_modula.h`.

V nadaljevanju sledi opis osnovnih razredov v okolju OMNeT++, osnovnih metod, s katerimi lahko programiramo obnašanje enostavnih modulov in posebnosti C++ programiranja v okolju OMNeT++.

1.3.1 Osnovni razredi v okolju OMNeT++

Sledeči razdelek opisuje nekatere osnovne razrede v okolju OMNeT++, katerih deklaracije so podane v zglavni datoteki `<omnetpp.h>`.

Razred `cSimpleModule`

Enostavni moduli se vedno dedujejo iz razreda `cSimpleModule`. Modul ima poleg konstruktorja in destruktorja že deklarirane virtualne metode, ki jih lahko uporabnik implementira in se kličejo ob določenih simulacijskih dogodkih. To so `initialize()`, `handleMessage(cMessage *msg)` in `finish()` (glej razdelek 1.3.2).

Razred `cMessage`

Razred `cMessage` služi implementaciji sporočil, ki si jih moduli med seboj izmenjujejo. Z njim so povezane sledeče metode, ki jih lahko uporabljamo znotraj programske kode - predvsem v metodi `handleMessage()` (glej razdelek 1.3.2):

- `send()`: za pošiljanje sporočil drugim modulom preko specificiranih vrat; primer: `send(msg,"out")`.

- `sendDelayed()`: za zakasnjeno pošiljanje sporočil drugim modulom preko specificiranih vrat; primer `sendDelayed(msg,0.005,“out”)`.

- `scheduleAt()`: za pošiljanje sporočil samemu sebi. Primer uporabe: generator prometa pošilja sporočila samemu sebi ob intervalih, ki so definirani z medprihodnimi časi. Sprejem tega sporočila povzroči generiranje prometa; primer: `scheduleAt(simTime()+par("interArrivalTime"). doubleValue(), msg)` (glej razdelek 1.3.3);

- `cancelEvent()`: za preklic dogodka, ki je bil prožen z metodo `scheduleAt()`; primer: `cancelEvent(msg)`.

- `cancelAndDelete()`: za preklic in brisanje dogodka.

Razred `cGate`

Razred `cGate` omogoča delo z vrati modula. Instance tega razreda vedno pripadajo določenemu modulu, ki z njimi tudi upravlja (klic konstruktorja in destruktorja). Glavne metode za delo z vrati so

- `gate()`: vrne referenco na vrata z imenom, ki je podan kot argument metode; primer: `cGate *outGate = gate("g")`. Pri `inout` vratih imamo za vsako smer svoj objekt: `cGate *inGate = gate("g$i"); cGate outGate = gate("g$o")`.
- `hasGate()`: vrne `True`, če modul ima vrata s podanim imenom; primer `if (hasGate("gOut")) ...`
- `gateSize()`: vrne velikost vektorskih vrat, ki so podana kot argument; primer: `gateSize("g")`.

Razred `cChannel`

Podobno kot pri programiranju enostavnih modulov tudi pri programiranju kanalov izhajamo iz obstoječega OMNeT++ razreda, in sicer iz razreda `cChannel`. Ponavadi potrebe po pisanju lastnih kanalov ni, saj lahko izhajamo iz predefiniranih kanalov (`IdealChannel`, `DelayChannel` in `DatarateChannel`), zato se v podrobnosti programiranja kanalov ne bomo spuščali, vseeno pa je v nadaljevanju razložena osnovna uporaba kanalov v jeziku C++.

Kanali so vedno vezani na vrata. Do njih torej dostopamo zgolj preko vrat z uporabo metode `getChannel()`. Primer uporabe: `cChannel *channel = gate->getChannel()`. Lahko pristopamo tudi v obratni smeri, tako da preko kanala pridemo do reference vrat: `cGate *gate = channel->getSourceGate()`.

Kanali, ki modelirajo zakasnitev paketov pri prenosu pravimo *prenosni kanali* (angl. *Transmission Channels*). Preko podanih vrat lahko pridemo do njihovega prenosnega kanala (ni nujno, da je to ravno kanal, ki je neposredno vezan na ta vrata) na sledeč način: `cChannel channel = gate->getTransmissionChannel()`.

Pakete lahko preko prenosnega kanala prenašamo, šele ko je le-ta prost. Zasedenost prenosnega kanala lahko preverimo z metodo `isBusy()`, do časa sprostitve prenosnega kanala pa pridemo z metodo `getTransmissionFinishTime()`.

Razred `cPacket`

Objekti, ki pripadajo razredu `cPacket`, t.i. *paketi*, predstavljajo razširitev objektov razreda `cMesssage` in omogočajo simuliranje prenosa komunikacijskih paketov. Paketi omogočajo modeliranje zakasnitve prenosa, omejene pasovne širine prenosnih kanalov in napak pri prenosu preko kanalov. V ta namen se uporabljajo skupaj z objekti tipa `cChannel`, ki jih lahko vežemo na povezave med moduli (glej razdelek 1.2.6). Pogosto uporabljene metode razreda `cPacket` so

- `getBitLength(), getByteLength()`: vrne dolžino paketa v bitih oziroma bajtih, ki se uporabi za izračun trajanja prenosa paketa preko kanala,
- `setBitLength(), setByteLength()`: metodi za nastavljanje dolžine paketa, pri čemer argument podamo v bitih oziroma v bajtih,
- `setBitError(), hasBitError`: vračanje oziroma nastavljanje zastavice, ki pove, da je paket pokvarjen,
- `getDuration()`: vračanje časa prenosa paketa - vrednost je pravilna šele po opravljenem prenosu paketa.

Razred `cQueue`

Razred `cQueue` predstavlja implementacijo čakalne vrste. Privzeto delovanje čakalne vrste je *First In First Out* (FIFO), ki pa ga lahko spremenimo. Spremenimo jo lahko tudi v prioritetno čakalno vrsto. Pogosto uporabljene metode razreda `cQueue` so

- `insert()`: metoda za dodajanje elementov v čakalno vrsto,
- `pop()`: metoda za odstranjevanje elementov iz čakalne vrste (vsakič se vzame tisti element, ki je naslednji na vrsti),
- `clear()`: metoda, ki izprazni čakalno vrsto,
- `isEmpty()`: metoda, ki vrne odgovor na vprašanje - ali je čakalna vrsta prazna,
- `length()`: metoda, ki vrne dolžino čakalne vrste.

1.3.2 Osnovne metode enostavnih modulov v OMNeT++

Sledeči razdelek opisuje osnovne metode, ki jih programiramo kot odzive na določene simulacijske dogodke.

Metoda `ime_modula()`

Metoda `ime_modula()` predstavlja konstruktor, ki se kliče ob kreaciji objekta, ki modul predstavlja. Metodo kliče simulator.

Metoda `initialize()`

Metoda `initialize()` je inicializacijska metoda, ki jo simulator kliče še pred prvim simulacijskim dogodkom. Če je to potrebno, iz te metode sprožimo prvi simulacijski dogodek.

Metoda `handleMessage(cMessage *msg)`

Metoda `handleMessage` predstavlja jedro vsakega modula in s tem tudi simulacije, saj izmenjava sporočil predstavlja glavne simulacijske dogodke v okolju OMNeT++. Metoda se sproži vsakič, ko modul dobi sporočilo - bodisi od sebe bodisi od drugih modulov. Ob sprejemu sporočila se mora metoda ustrezno odzvati glede na njegovo vsebino. Odziv na sprejem sporočila ponavadi vsebuje vsaj pošiljanje novega sporočila bodisi sebi bodisi nekemu drugemu modulu.

Metoda `finish()`

Metoda se kliče, ko v čakanju ni več nobenih dogodkov - ob koncu simulacije. Ponavadi se simulacija zaključi, ko na čakanju ni nobenega sporočila več v nobenem modulu. Kliče se samo v primeru, ko se je simulacija izvedla brez napak. Iz tega razloga ta metoda ni primerna za čiščenje pomnilnika - to je potrebno narediti v destruktorju. Metoda `finish` je primerna predvsem za obdelavo pridobljenih podatkov po izvedbi simulacije.

Metoda `~ime_modula()`

Metoda `~ime_modula()` predstavlja desktruktor, ki se kliče ob uničenju objekta. V metodi je potrebno izbrisati vse objekte, ki smo jih ustvarili z `new`. Za sporočila, ki so trenutno na čakanju (angl. *pending*) uporabimo metodo `cancelAndDelete()`, za ostale objekte pa metodo `delete()`. Simulacijsko okolje nas bo ob koncu simulacije opozorilo, če za seboj nismo počistili.

1.3.3 Ostale posebnosti pri programiranju

Sledeči razdelek izpostavi nekaj posebnosti pri C++ programiranju v okolju OMNeT++.

Registracija modula

Na začetek datoteke `ime_modula.cc` moramo vedno vključiti makro `Define_Module(ime_modula)`, ki modul registrira v OMNeT++ okolju. V nasprotnem primeru nam simulacijsko okolje javi napako, da modul ne obstaja. V primeru programiranja lastnih kanalov le-te registriramo na podoben način z makrojem `Define_Channel(ime_kanala)`.

Dostopanje do parametrov modula

Znotraj NED datotek lahko definiramo parametre modula, preko katerih se lahko posredujejo določene vrednosti do najnižjega nivoja, t.j. do enostavnih modulov in kanalov. Do vrednosti parametrov lahko pridemo z uporabo metode `par()`; primer uporabe: `double delay = par("delay")`. Parametri tipa `volatile` se bodo ovrednotili vsakič, ko bomo zahtevali njihovo vrednost preko te metode, ostali parametri pa bodo cel čas simuliranja imeli enako vrednost.

Klic metod drugih modulov

Metode, ki pripadajo različnim modulom, se privzeto med seboj vidijo. V ta namen je v vsako metodo, ki jo želimo klicati od zunaj, t.j. iz nekega drugega modula, na začetek potrebno dodati bodi makro `Enter_Method()`, pri katerem se bo ob klicu metode v GUI vmesniku pojavil podani izpis, bodisi makro `Enter_Method_Silent()`. Primer uporabe makroja: `Enter_Method ("release(%ld)", amountToRelease)`.

Opazovanje vrednosti spremenljivk med simulacijo

Če želimo opazovati vrednosti posamezne spremenljivke med izvajanjem simulacije, to povemo z makrojem `WATCH()` v metodi `initialize()`. Primer uporave: `WATCH(ime_spremenljivke)`.

Izpisovanje v konzolo

Med izvajanjem simulacije lahko v konzolo izpisujemo nize na sledeč način
`EV « niz1 « niz2 « spremenljivka1 « ...;`

Izpisovanje v GUI simulatorja

Izpisovanje v GUI prožimo z metodo `getDisplayString().setTagArg()`.
Primer uporabe:

```
char buf[80];
sprintf(buf, "oznaka: %d\n",spremenljivka);
getDisplayString().setTagArg("t",0, buf);
```

1.4 Konfiguriranje eksperimentov z INI datotekami

Medtem ko na NED datoteke gledamo kot na del modelov, INI datoteke (ponavadi je na projekt vezana samo ena datoteka, in sicer `omnetpp.ini`) predstavljajo del eksperimentov. Tak način dvonivojskega opisovanja nam omogoča ločitev postavljanja modelov od eksperimentov. INI datoteke torej opisujejo eksperimente, med katerimi lahko izbiramo po zagonu simulacije. Posamezna INI datoteka lahko vsebuje več t.i. konfiguracij (angl. *configurations*) parametrov nad istim ali različnimi modeli (t.j. omrežji). Ob zagonu simulacije izberemo konfiguracijo, ki jo želimo simulirati. V nadaljevanju so navedene nastavitve, ki jih najpogosteje določamo v INI datoteki.

- Model oziroma omrežje nad katerim želimo izvajati simulacije določamo z lastnostjo `network`, ki mu dodelimo ime NED datoteke izbranega modela; primer: `network=FifoNet`.

- Vrednosti parametrov lahko v INI datotekah določamo samo tistim parametrom, katerih vrednost še ni bila določena v NED datotekah. Določamo jih tako, da na levi strani podamo ime parametra v obliki `ime_omrezja.ime_modula.ime_parametra`, na desni strani pa njegovo vrednost (če je potrebno vključno z enotami). Vrednost je lahko tako kot v NED datotekah podana kot porazdelitev. Če je modul vgnezden, je potrebno podati celo pot do njega. Pri podajanju poti lahko uporabljamo tudi t.i. *nadomestne znake* (angl. *wildcards*), npr. `*`, `**` in `?`. Primer: `**.fifo.bitsPerSec = 1000bps`. Orodje nam omogoča konfiguracijo več simulacijskih ponovitev z različnimi vrednostmi parametrov ali različnimi omejitvami.

  Primer 1:

  ```
  *.ime_parametra = ${1, 2, 5, 10..50 step 10}
  ```

  Primer 2:

  ```
  *.ime_parametra = ${N=1, 2, 5, 10..50 step 10}
  ```

  Primer 3:

  ```
  *.param1 = ${i=1..10 step 2}
  *.param2 = ${j=1..20 step 3}
  constraint = $j <= sqrt($i)
  ```

- Običajno je potrebno določiti časovne parametre simulacij, predvsem čas njihovega trajanja. Primer: `sim-time-limit = 100h`.

- Nastavljanje načinov zbiranja simulacijskih podatkov je razloženo v poglavju 1.5.

OMNeT++ INI datoteke so v grobem sestavljene iz dveh tipov sekcij, in sicer iz sekcije *General* in iz *Config* sekcij.

V sekciji *General* konfiguriramo splošne nastavitve, ki bodo pri vseh eksperimentih enake. S tem se izognemo ponovitvam istih nastavitev v vseh sekcijah. Primer sekcije *General*:

```
[General]
network = FifoNet
sim-time-limit = 100h
```

Sekcij *Config* je lahko več. V primeru, da želimo definirati le en eksperiment, sekcij *Config* ne potrebujemo, saj lahko eksperiment opišemo zgolj s sekcijo *General*. Primer dveh sekcij *Config*, ki sledita gornji sekciji *General*:

```
[Config Fifo1]
description = "low job arrival rate"
**.gen.sendIaTime = exponential(0.2s)
**.gen.msgLength = 100b
**.fifo.bitsPerSec = 1000bps
```

```
[Config Fifo2]
description = "high job arrival rate"
**.gen.sendIaTime = exponential(0.01s)
**.gen.msgLength = 10b
**.fifo.bitsPerSec = 1000bps
```

1.5 Zbiranje simulacijskih rezultatov

Simulacijske rezultate lahko po eni strani pridobivamo že med tekom simulacije, t.i. *runntime analiza* (glej razdelek 1.3.3). Tak način zbiranja podatkov pa ni primeren za obsežnejše simulacije. V ta namen moduli v okolju OMNeT++ uporabljajo t.i. *signale*, ki jih lahko uporabljamo bodisi nad enostavnimi moduli bodisi na kanali. Moduli in kanali preko signalov oddajajo določene vrednosti (npr. čas nahajanja zahteve v čakalni vrsti, dolžina čakalne vrste itd.), ki jih simulacijsko okolje zbira na podlagi podanih konfiguracij. Odvisno od konfiguracije se te vrednosti po potrebi še dodatno obdelajo v metodi `finish()` - *skalarni podatki* ali pa shranijo v vektor - *vektorski podatki*. Rezultati simuliranja se shranijo v izhodne datoteke skalarjev (angl. *output scalar files*) in izhodne datoteke vektorjev (angl. *output vector files*). Vsak tek simulacije (angl. *simulation run*) rezultate shrani v svojo datoteko.

Uporabo signalov mora uporabnik konfigurirati na treh nivojih, in sicer:

- v izvorni in zglavni datoteki enostavnega modula (C++),
- v NED datoteki in
- v INI datoteki.

V nadaljevanju je podrobneje opisana uporaba signalov.

1.5.1 Programiranje signalov (C++)

Do signalov v okolju OMNeT++ dostopamo preko njihovih ID-jev (*signalID*), ki jih shranjujejmo v spremenljivke tipa `simsignal_t`. Signale, ki jih nameravamo uporabiti je potrebno deklarirati v zglavni datoteki modula oziroma kanala

```
simsignal_t imeSignala;
```

V `initialize()` metodi modula, ki mu signal pripada, posamezen signal registriramo z metodo `registerSignal()`, ki ji kot argument podamo ime signala, preko katerega bo le-ta viden simulacijskemu okolju

```
imeSignala = registerSignal("imeSimSignala");
```

Signali objekt je tako pripravljen za oddajanje vrednosti simulacijskemu okolju, ki jih prožimo z metodo `emit()`. Oddajanje signala praviloma prožimo vsakič, ko se opazovana vrednost spremeni. Metodi `emit()` kot argument podamo ime signalnega objekta (ime spremenljivke) in vrednost, ki jo oddajamo preko signala

```
emit(imeSignala, vrednost);
```

Čeprav se ponavadi oddajajo številske vrednosti, je lahko oddana vrednost predstavljena tudi z drugimi podatkovnimi tipi.

V nadaljevanju sledi primer uporabe signala, ki oddaja dolžino čakalne vrste. Najprej deklariramo signal `queueLengthSignal`

```
simsignal_t queueLengthSignal;
```

V metodi `initialize()` signal registriramo

```
queueLengthSignal = registerSignal("queueLength");
```

Vsakič, ko se dolžina čakalne vrste spremeni (bodisi pride nova zahteva bodisi se v procesiranje vzame naslednja iz čakalne vrste), je potrebno javiti novo dolžino čakalne vrste (predpostavljamo, da za čakalno vrsto uporabljamo objekt `queue`, ki pripada razredu `cQueue`)

```
emit(queueLengthSignal,queue.length());
```

1.5.2 Konfiguracija beleženja signalov (NED)

Avtomatsko beleženje signala vklopimo z deklaracijo v NED datoteki enostavnega modula ali kanala, ki mu signal pripada. To naredimo v sekciji `parameters` z uporabo lastnosti `@statistic`, pri čemer ime signala podamo kot *indeks lastnosti* (angl. *property index*) v oglatih oklepajih, lastnosti beleženja signala pa kot *vrednosti lastnosti* (angl. *property values*) v navadnih oklepajih:

```
parameters:
   @statistic[imeSimSignala](lastnosti beleženja signala);
```

Lastnosti beleženja, ki jih ponavadi nastavljamo so:

- `title`: opis pomena signala, ki ga lahko uporabijo razna vizualizacijska orodja pri prikazovanju,
- `unit`: enote - za vizualizacijo,
- `interpolationmode`: uporabljena metoda interpolacije, če je le-ta potrebna,
- `enum`: simbolna imena za različne numerične vrednosti, npr. `"IDLE=1, BUSY=2, DOWN=3"`,
- `record`: način izračunavanja skalarnih vrednosti iz vseh pridobljenih vrednosti signala.

Pogosteje uporabljene vrednosti za lastnost `record` so:

- `vector`: vse vrednosti se shranijo v vektor,

- `count`: prešteje število oddanih signalov - vsebina signalov v tem primeru ni relevantna,
- `last`: vzame samo zadnjo oddano vrednost,
- `sum`: sešteje vse oddane vrednosti,
- `mean`: vzame srednjo oddano vrednost,
- `min`: vzame minimalno oddano vrednost,
- `max`: vzame maksimalno oddano vrednost,
- `timeavg`: s pomočjo časovnih oznak izračuna časovno povprečje oddanih vrednost,
- `stats`: vzame `count`, `sum`, `mean`, standardno deviacijo deviation, `min` in `max`,
- `histogram`: vzame `stats` in histogram vrednosti.

Primer:

```
PRIMER
```

1.5.3 Konfiguracija beleženja signalov (INI)

V INI datotekah lahko določamo način beleženja podatkov na nivoju posameznih simulacij (konfiguracij). Ponavadi nastavljamo sledeče parametre:

- `output-vector-file`: datoteke za shranjevanje vektorskih podatkov (privzeto: `$resultdir/$configname-$runnumber.vec`),
- `output-scalar-file`: datoteke za shranjevanje skalarnih podatkov (privzeto: `$resultdir/$configname-$runnumber.sca`),
- `vector-recording`: vklop/izklop beleženja vektorskih podatkov na nivoju signala, modula ali celotne simulacije; primer: `**.vector-recording = true/false`,
- `scalar-recording`: vklop/izklop beleženja skalarnih podatkov na nivoju signala, modula ali celotne simulacije; primer: `**.scalar-recording = true/false`,
- `result-recording-mode`: način beleženja podatkov na nivoju signala, modula ali celotne simulacije, pri čemer lahko izbiramo med vrednostmi parametra `record` lastnosti `@statistic` v NED datotekah (glej razdelek 1.5.2); pri tem se vrednost definirana v NED datoteki povozi; primer: `**.queueLength.result-recording-mode = timeavg,max`,

- `param-record-as-scalar`: vklop/izklop shranjevanja parametrov v datoteko skalarjev na nivoju modula ali celotne simulacije; primer: `**.networkLoad.param-record-as-scalar = true`,
- `sim-time-limit`: določitev časa simuliranja; primer: `sim-time-limit = 10000 s`,
- `warmup-period`: začetni čas simuliranja, ko naj se statistike ne beležijo; primer: `warmup-period = 100 s`,
- `record-eventlog`: vklop/izklop beleženja dogodkov (angl. *eventlog*); `record-eventlog = true/false`.

1.6 Simuliranje

Način poganjanja simulacij lahko nastavljamo preko okna *Run Configurations*, do katerega pridemo preko menija *Run*. V nadaljevanju so opisane možnosti za simuliranje in anlaizo simulacijskih rezultatov v orodju OMNeT++.

1.6.1 Okolje *Tcl/Tk*

Preden začnemo z beleženjem podatkov je priporočljivo preveriti pravilnost delovanja modela preko simulacijskega okolja *Tcl/Tk*. To naredimo tako, da v oknu *Run Configurations* kot *User Interface* izberemo *Default* oziroma *Tcl/Tk*. Pri tem je pomembno, da je polje *Run number* prazno (privzeto). Izberemo lahko tudi INI datoteko (privzeta je *omnetpp.ini* in konfiguracijo. Če konfiguracije nismo izbrali in jih imamo v INI datoteki več, nas orodje *Tcl/Tk* najprej vpraša, katero konfiguracijo želimo simulirati. Simulator zatem vzpostavi konfiguracijo in določi parametre na sledeč način: če ima parameter vrednost določeno v NED datoteki, se mu dodeli ta vrednost, sicer preveri, če je njegova vrednost določena v INI datoteki. Če je, se vzame vrednost iz te datoteke, sicer mora uporabnik vnesti vrednost preko *Tcl/Tk* okna. Orodje *Tcl/Tk* omogoča spremljanje poteka simulacije preko *Inspect Network* vmesnika, v katerega se izrisuje pretok sporočil preko modulov ali preko konzole, v katerega se izpisujejo definirani izpisi. Ob dvojnem kliku na posamezen modul v *Inspect Network* oknu, se odpre okno z lastnostmi modula, kjer lahko spremljamo vrednosti njegovih parametrov in vrednosti spremenljivk definiranih v makroju `WATCH()` (glej razdelek 1.3.3).

1.6.2 Simuliranje preko ukazne vrstice

Okolje *Tcl/Tk* je primerno le za preverjanje pravilnosti delovanja modela in njegovo razhroščevanje (angl. *debugging*). Za zbiranje simulacijskih rezultatov imamo v okolju OMNeT++ na voljo simuliranje preko ukazne vrstice (angl. *Command Line*). V oknu *Run Configurations* v ta namen kot *User Interface* izberemo *Command Line*. Nastaviti je potrebno konfiguracijo, ki jo bomo simulirali in simulacijske teke (angl. *Run number(s)*), ki jih želimo simulirati (privzeto =

0). Za izvedbo vseh simulacijskih tekov, definiranih v INI datoteki, v polje *Run number* vnesemo *.

1.6.3 Obdelava simulacijskih rezultatov

Po uspešni izvedbi simulacij se podatki, katerih beleženje smo nastavili v NED in INI datotekah shranijo v direktorij *results* v datoteke tipa *vec* za vektorske podatke, tipa *sca* za skalarne podatke in *elog* za beleženje dogodkov. Posamezna datoteka je vezana na simulacijski tek - vsak simulacijski tek ima svojo *vec*, *sca* in *elog* datoteko. Če imamo posamezno beleženje izklopljeno, se datoteka ne ustvari.

Ob dvojnem kliku na datoteke tipa *vec* ali *sca* se odpre okolje *Analysis Editor*, v katerem lahko vsebino datotek pregledujemo, filtriramo rezultate in izrisujemo različne grafe. Ob dvojnem kliku na datoteke tipa *elog* se odpro okno, v katerem lahko pregledujemo potek simulacijskih dogodkov.

Literatura

[1] "OMNeT++." `www.omnetpp.org/`, 2012.

[2] A. Varga, *Modeling and Tools for Network Simulation*, ch. OMNeT++, pp. 35–59. Springer-Verlag, 2010.

[3] A. Varga, *OMNeT++ User Manual, Version 4.1.* OpenSim Ltd., 2010.

[4] "INET." `http://inet.omnetpp.org/`, 2012.