# Ubacivanje jednostavna modula u OMNeT++

**Obadić, Vito**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Vito Obadić**

# Ubacivanje jednostavna modula u OMNeT++

**ZAVRŠNI RAD**

**Varaždin, 2024.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Vito Obadić**

**Matični broj: 35918/07-R**

**Studij: Primjena informacijske tehnologije u poslovanju**

# Ubacivanje jednostavna modula u OMNeT++

**ZAVRŠNI RAD**

Mentor: dr. sc. Luka Milić

**Varaždin, lipanj**

**2024.**

*Vito Obadić*

**Izjava o izvornosti**

Izjavljujem da je moj završni rad izvorni rezultat mojega rada te da se u izradbi rečenoga nisam koristio drugim izvorima osim onima koji su u njem navedeni. Za izradbu rada su rabljene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredaba u sustavu FOI-radovi*

_____

**Sažetak**

OMNeT++ mrežni simulator koristi se za simulaciju i modeliranje raznolikih vrsta mreža, sustava i prometnica. Njegova glavna prednost je fleksibilnost i raznolikost kod istraživanja mreža i mrežnih protokola. OMNeT++ koristi dva programska jezika: NED i C++. NED je deskripcijski jezik koji služi za definiranje, povezivanje i stvaranje veza između modula. C++ se koristi za pokretanje svih simulacija te za komunikaciju između svih povezanih modula. Cilj ovog rada je izrada jednostavnog modula koristeći NED i druge programske jezike te prikaz simulacije tog modula. Igraća konzola je napravljena isključivo korištenjem postojećih modula i NED jezika, kao i meteorološka stanica dok je jednostavna meteorološka stanica izrađena koristeći programski jezik Python. Jednostavna meteorološka stanica može se povezati s bilo kojim uređajem u simulatoru te šalje podatke o temperaturi i vlažnosti zraka. Svi moduli su zapakirana, spremni i dostupni široj javnosti za daljnju distribuciju i korištenje u OMNeT++ simulatoru i budućim simulacijama.

**Ključne riječi:** OMNeT++, NED, C++, Python, modul, podmodul, mreža

# Kazalo

# 1. Uvod

U ovom završnom radu istražit ću simulator OMNeT++ i njegov jezik modeliranja NED. Naglasak će biti na metodama izgradnje dodatnih modula bez zahtjeva za naprednim znanjem programiranja u C++-u. Ovo je od iznimne vrijednosti jer omogućuje široku spektru korisnika uključiti se u proces stvaranja simulacijskih modela, bez obzira na njihovu stručnost u programiranju.

Proučit će se proces izradbe jednostavna programskoga modula unutar simulatora OMNeT++ rabeći mehanizme koji su na raspolaganju. Stvorit će se simulacijski moduli koji će simulirati ponašanje komada sklopovlja.

Prikazat će se mogućnost kreiranja modula, podmodula i simulacija koristeći programski jezik Python.  Zatim će se istražiti načini kako zapakirati ovaj modul kako bi se distribuirao i učinio pristupačnim široj javnosti.

## 1.1. Metode i tehnike rada

Pri izradbi ove teme rabio sam navedene izvore za proučavanje aplikacije OMNeT++ koje sam pronašao s pomoću tražilice Google znalac. Izvori su dodani s pomoću aplikacije Zoteo. Praktičan dio izrađen je u samoj aplikaciji OMNeT++ gdje je uz pomoć literature, interneta, foruma i stečenoga iskustva ostvaren.

## 1.2. OMNeT++

Model OMNeT++-a sastoji se od hijerarhijski ugniježđenih modula. Dubina ugniježđenosti modula nije ograničena, što korisniku omogućuje zrcaliti logičku građu stvarna sustava u građi modela. Moduli komuniciraju preko prosljeđivanja poruka. Poruke mogu u sebi imati po volji složene podatkovne strukture. Moduli mogu slati poruke ili izravno na svoje odredište ili duž u naprijed definirana puta, kroz vrata i veze [1].

OMNeT++ se može rabiti u različne svrhe, uključujući modeliranje prometa telekomunikacijskih mreža, modeliranje protokola, modeliranje mreža čekanja, modeliranje više procesora i drugih raspodijeljenih sklopovskih sustava, pregled valjanosti sklopovskih arhitektura, procjenu vidova performanse složenih programskih sustava i modeliranje kojega bilo drugoga sustava gdje diskretni pristup događaju bude prikladan. Parametri u modulima služe u tri glavne svrhe: prilagodba ponašanja modula, stvaranje opruživih topologija modela i olakšavanje komunikacije modula (tako što ti parametri glume zajedničke varijable).

Module na najnižoj razini hijerarhije, koja sadržava algoritme u modelu, treba osigurati korisnik. Tijekom izvođenja simulacije čini se da ovi jednostavni moduli rade usporedno, ostvareni kao korutine. Simulacije OMNeT++-a mogu sadržavati različna korisnička sučelja za otklanjanje pogrješaka, demonstraciju i slijedno izvođenje. Simulator, korisnička sučelja i oruđa su prenosiva i znano je da rade na Windowsima i brojnim inačicama UNIX-a, rabeći različne kompilatore za C++.

# 2. Koncepti modeliranja

OMNeT++ korisniku pruža učinkovita oruđa za opisivanje građe stvarna sustava. Neke od glavnih značajaka su:

• hijerarhijski ugniježđeni moduli;

• moduli kao instancije tipova modula;

• moduli koji komuniciraju porukama preko kanala;

• opruživi parametri modula;

• jezik za opis topologije [1].

Model OMNeT++-a sastoji se od hijerarhijski ugniježđenih modula koji komuniciraju porukama. Modeli OMNeT++-a često se nazivaju mrežama. Modul najviše razine je sustavski modul. Sustavski modul sadržava podmodule, koji mogu sadržavati i druge podmodule. Dubina ugnježđivanja modula nije ograničena; ovo korisniku omogućuje razmišljanje o logičkoj građi stvarna sustava u građi modela [1].

Moduli koji sadržavaju podmodule nazivlju se složenim modulima, za razliku od jednostavnih modula koji su na najnižoj razini hijerarhije modula. Jednostavni moduli sadržavaju algoritme u modelu. Korisnik ostvaruje jednostavne module u C++-u, rabeći knjižnicu simulacijskih razreda OMNeT++-a [1].

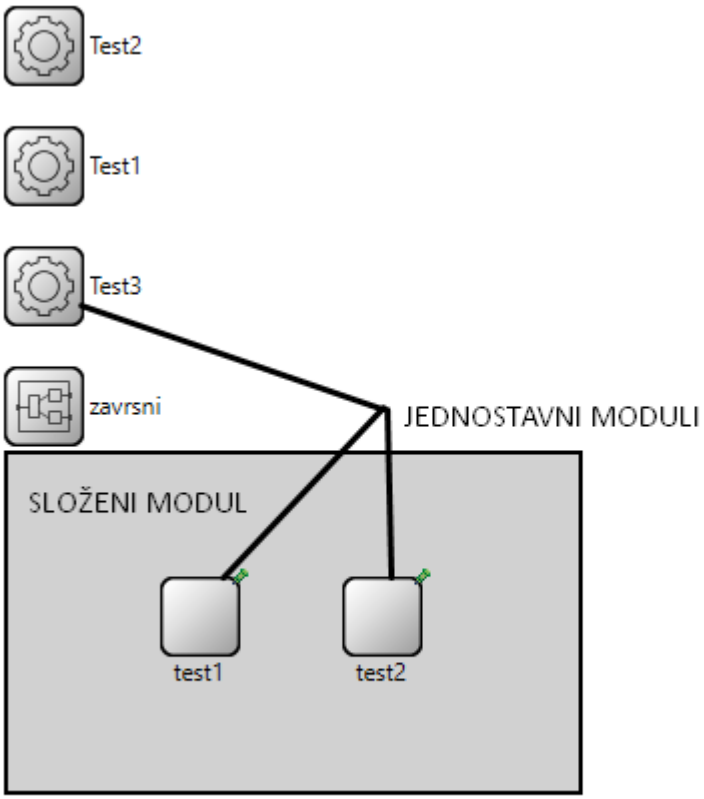

*Slika 1. OMNeT'++ (Izvor: Vlastita izrada)*

## 2.1. Tipovi modula

I jednostavni i složeni moduli smatraju se tipovima modula u OMNeT++-u. Prigodom izgradnje modela korisnik definira te tipove modula, a instancije ovih tipova služe kao sastavnice za složenije tipove modula. U konačnici korisnik stvara modul sustava instanciranjem tipa modula definirana prije, a svi mrežni moduli se zatim instanciraju kao podmoduli i „potpodmoduli“ unutar modula sustava.

U kontekstu izgradnje modela ne čini se razlika između porabe jednostavna modula ili složena modula kao građevnoga bloka. Ova opruživost omogućuje korisniku podijeliti jednostavan modul u više jednostavnih modula ugrađenih unutar složena modula ili obrnuto; ona okuplja funkcionalnost složenoga modula u jedan jednostavni modul, bez utjecaja na postojeće korisnike tipa modula.

Na dalje, vrste modula mogu se pohraniti u zasebne datoteke, neovisno o njihovoj stvarnoj lokaciji porabe. Ova značajka omogućuje korisniku grupiranje tipova modula zajedno i stvaranje knjižnica sastavnica za prikladan ustroj i ponovnu porabu.

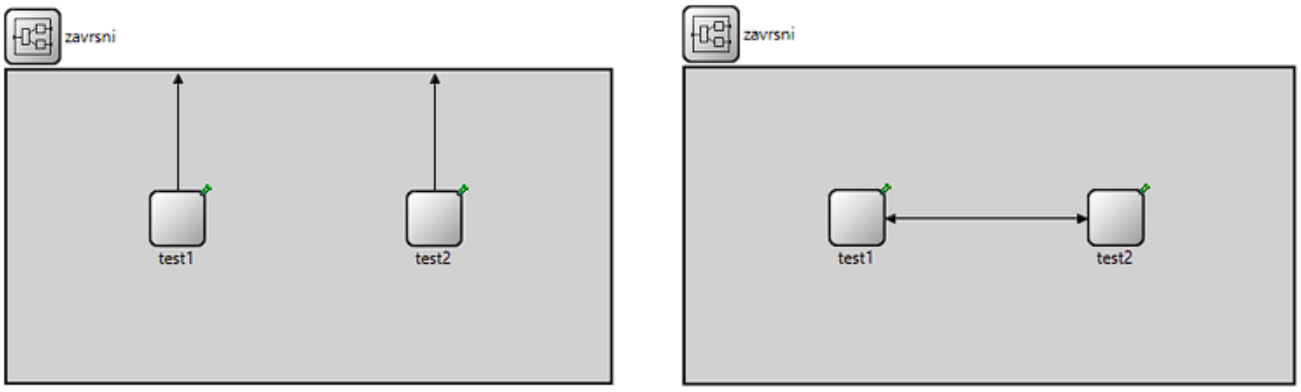



*Slika 2. Jednostavni moduli (Izvor: Vlastita izrada)*

## 2.2. Poruke, vrata, poveznice

Moduli komuniciraju razmjenom poruka. U stvarnoj simulaciji poruke mogu činiti okvire ili pakete u računalnoj mreži, poslove ili kupce u mreži čekanja ili druge vrste pokretnih entiteta. Poruke mogu sadržavati po volji složene podatkovne strukture [1].

Jednostavni moduli mogu slati poruke ili izravno na svoje odredište ili duž u naprijed definirana puta, kroz vrata i veze. „Vrijeme lokalne simulacije“ modula naprjeduje kad modul primi poruku. Ta poruka može prispjeti iz drugoga modula ili iz istoga modula („samoporuke“ se rabe za ostvaraj mjerača vremena). „Vrata“ su ulazna i izlazna sučelja modula; poruke se šalju preko izlaznih vrata i prispijevaju kroz ulazna vrata. Svaka veza (koja se naziva i poveznicom) stvorena je unutar jedne razine hijerarhije modula: unutar složenoga modula mogu se spojiti odgovarajuća vrata dvaju podmodula, ili vrata jednoga podmodula i vrata složenoga modula. Zbog hijerarhijske građe modela poruke obično putuju kroz niz veza, vezâ za odlazak i dolazak u jednostavnim modulima. Takvi nizovi veza koje idu od jednostavna modula do jednostavna modula nazivlju se rutama. Složeni moduli djeluju kao „crne kutije“ u modelu, prozirno prenoseći poruke između njihove unutrašnjosti i vanjskoga svijeta [1].




*Slika 3. Povezanost modula (Izvor: Vlastita izrada)*

## 2.3. Modeliranje prijenosa paketa

Vezama se mogu dodijeliti tri parametra koji olakšavaju modeliranje komunikacijskih mreža, ali mogu biti korisni i za druge modele: kašnjenje rasprostiranja, stopa pogrješke u bitovima i brzina prijenosa podataka; sva tri su izborna. Moguće je odrediti parametre veze pojedinačno za svaku vezu ili definirati tipove veza i rabiti ih kroz cijeli model [1].

Dakle, kašnjenje širenja, stopa pogrješke u bitovima i brzina prijenosa podataka parametri su koji se mogu dodijeliti vezama u OMNeT++-u za pomoć u modeliranju komunikacijskih mreža i drugih modela. Ponavljamo, ovi parametri su izborni i mogu se specificirati pojedinačno za svaku vezu ili definirati kao vrste veza za porabu u cijelom modelu. Kašnjenje širenja čini vremensko kašnjenje koje poruka doživljava dok putuje kroz kanal. Stopa pogrješke u bitu omogućuje modeliranje jednostavnih kanala s smetnjama određivanjem vjerojatnosti pogrješaka u prijenosu bita. Brzina prijenosa podataka, mjerena u bitovima po sekundi, rabi se za izračunavanje vremena prijenosa paketa.

U slučaju brzina prijenosa podataka slanje poruke u modelu odgovara prijenosu prvoga bita, dok dolazak poruke odgovara prijmu posljednjega bita. Međutim, vrijedno je napomenuti da ovaj model možda nije primjenjiv na sve scenarije. Na primjer, protokoli kao što su Token Ring i FDDI ne čekaju da prispije cijeli okvir prije ponavljanja prvih bitova. U tim slučajima okviri „protječu kroz postaje“ i kasne samo nekoliko bitova. Stoga, ako se modeliraju takve mreže, značajka modeliranja brzine prijenosa podataka OMNeT++-a ne može se rabiti.

## 2.4. Načini izgradnje dodatnih modula bez programiranja u C++-u

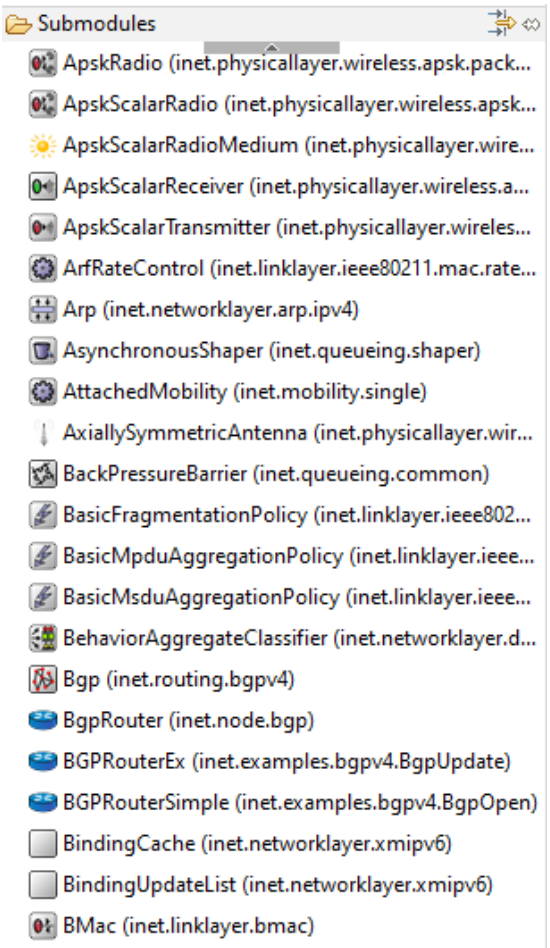
Iako je C++ glavni jezik za stvaranje simulacijskih modula, područje je simulacijskoga programiranja puno veće i opširnije. Postoje brojni alternativni putovi koji se mogu rabiti za izgradnju više modula bez potrebe za izravnim programiranjem u programskom jeziku C++. Ove opcije pružaju niz strategija i mogućnosti, od kojih je svaka prilagođena za zadovoljavanje različitih zahtjeva i preferencija unutar simulacijskoga okružja.

Neke od metoda su: INET Framework, jednostavni moduli, složeni moduli, uvezeni moduli, proširke u Pythonu i OMNeT++ GUI.

## 2.5. INET Framework

Za OMNeT++ okvir INET nudi veliku knjižnicu u naprijed izrađenih mrežnih simulacijskih modela i protokola. Mogu se rabiti brojne u naprijed izrađene mrežne sastavnice rabeći okvir INET bez potrebe za pisanjem koda u C++-u. Koncepti visoke razine koje nudi ovaj okvir omogućuju usredotočivanje na stvaranje i konfiguraciju mrežnih scenarija.

Okvir je INET podijeljen na module, od kojih svaki čini određeni mrežni protokol ili drugi član (kao što su preklopnici, domaćini ili usmjernici). Kako bi se izgradile zamršene mrežne topologije, mogu se konfigurirati i povezati moduli rabeći tzv. Network Description Language (NED). Specifikacija mrežnih sastavnica, njihovih parametara i njihovih veza omogućena je jezikom NED, koji olakšava izradbu simulacija bez potrebe porabe programskoga jezika C++.




*Slika 4. INET (Izvor: Vlastita izrada)*

## 2.6. Proširke u Pythonu

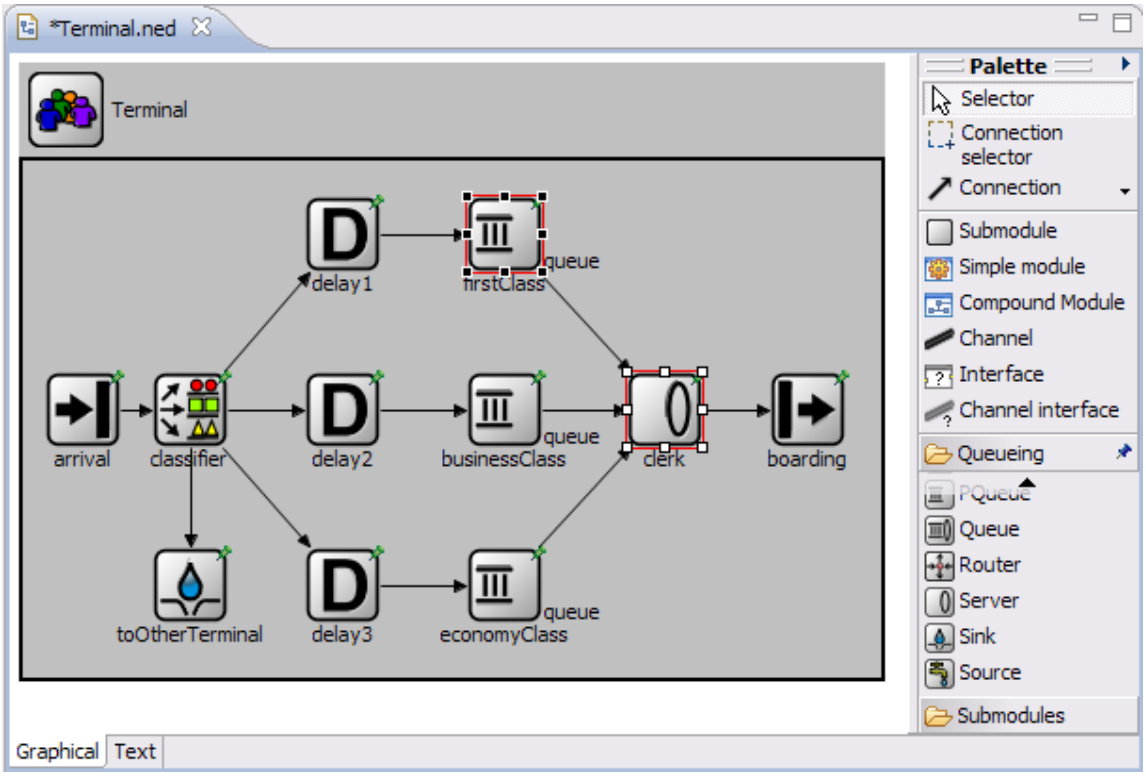
OMNeT++ podupire Python kao skriptni jezik, nudeći altenativu C++-u za neke zadatke s pomoću proširaka koje se dodaju u aplikaciju. Proširke u Pythonu korisne su za djelatnosti kao što su automatizacija pokretanja simulacije, procjena simulacijskih podataka i stvaranje posebnih vizualizacija – iako možda nemaju tako dobre performanse kao moduli u C++-u.

Kroz tzv. OMNeT++ Simulation API može se komunicirati s simulacijama u OMNeT++-u pomoću skripata u Pythonu. Kao rezultat toga provođenje pokusa i izvođenje raščlambe nakon simulacije postaje opruživije.

## 2.7. OMNeT++ GUI

OMNeT++ IDE pruža slikovno sučelje prilagođeno korisniku za stvaranje i konfiguraciju scenarija simulacije. Korisnici mogu jednostavno osmisliti simulacije porabom pristupa „povuci i ispusti" za uklapanje u naprijed izgrađenih sastavnica iz okvira INET.

OMNeT++ GUI omogućuje korisnicima slikom povezati simulacijske module, postaviti njihove parametre i rasporediti položaje u simulacijskom okružju. Ovaj pristup slikom smanjuje oslanjanje na programiranje u C++-u za jednostavne simulacije i olakšava posao čineći intuitivnijim proces razvoja simulacije.




*Slika 5. OMNeT'++ GUI (Izvor: Vlastita izrada)*

## 2.8. Osmišljenost simulacijske knjižnice

OMNeT++ pruža bogatu knjižnicu objekata za jednostavne module „implementatore“. Postoji nekoliko razlikujućih čimbenika između ove knjižnice i drugih općenitih ili simulacijskih knjižnica. Knjižnica razreda OMNeT++-a pruža refleksivne funkcionalnosti koje omogućuju ostvaraj otklanjanja pogrešaka i praćenja visoke razine, kao i automatske animacije povrh toga kao što je prikazano u korisničkom sučelju Tkenv. Curenje memorije, „natjecanje pokazivača“ i drugi problemi alokacije memorije česti su u programima u C++-u koje nisu napisali stručnjaci. OMNeT++ olakšava ovu poteškoću praćenjem vlasništva objekata i otkrivanjem pogrješaka uzrokovanih pokazivačima sličnih naziva i zlorabom zajedničkih objekata. Zahtjevi za jednostavnu porabu, modularnost, otvoreno podatkovno sučelje i potpora za uključivanje jednako su snažno utjecali na osmišljenost knjižnice razreda. Dosljedna primjena objektno orijentiranih tehnika čini jezgru simulacije skupljenom i tankom. Razmjerno je lako razumjeti njezine unutarnje mehanizme, što je korisna osobina i za otklanjanje pogrješaka i za obrazovnu porabu [2].

Postalo je uobičajeno izvoditi simulacije mreža velika razmjera s pomoću OMNeT++-a, s nekoliko desetaka tisuća ili više čvorova u mreži. Kako bi se zadovoljio ovaj zahtjev, agresivna optimizacija memorije ostvarena je u jezgri simulacije, temeljena na zajedničkim objektima i semantiki kopiranja pri pisanju.

Problemi preciznosti s izračunima s pomičnim zarezom povremeno su uzrokovali poteškoće u simulacijama. Kako bi se riješio ovaj problem, vrijeme simulacije je promijenjeno u 64-bitni predstav s fiksnom točkom na temelju cjelobrojnih brojeva. Jedna od glavnih poteškoća koju je trebalo riješiti bila je kako otkriti brojevne preljeve, budući da jezici C i C++, unatoč njihovim eksplicitnim ciljevima bivanja „bliskima sklopovlju“, nemaju potporu za otkrivanje preljeva cijelih brojeva.

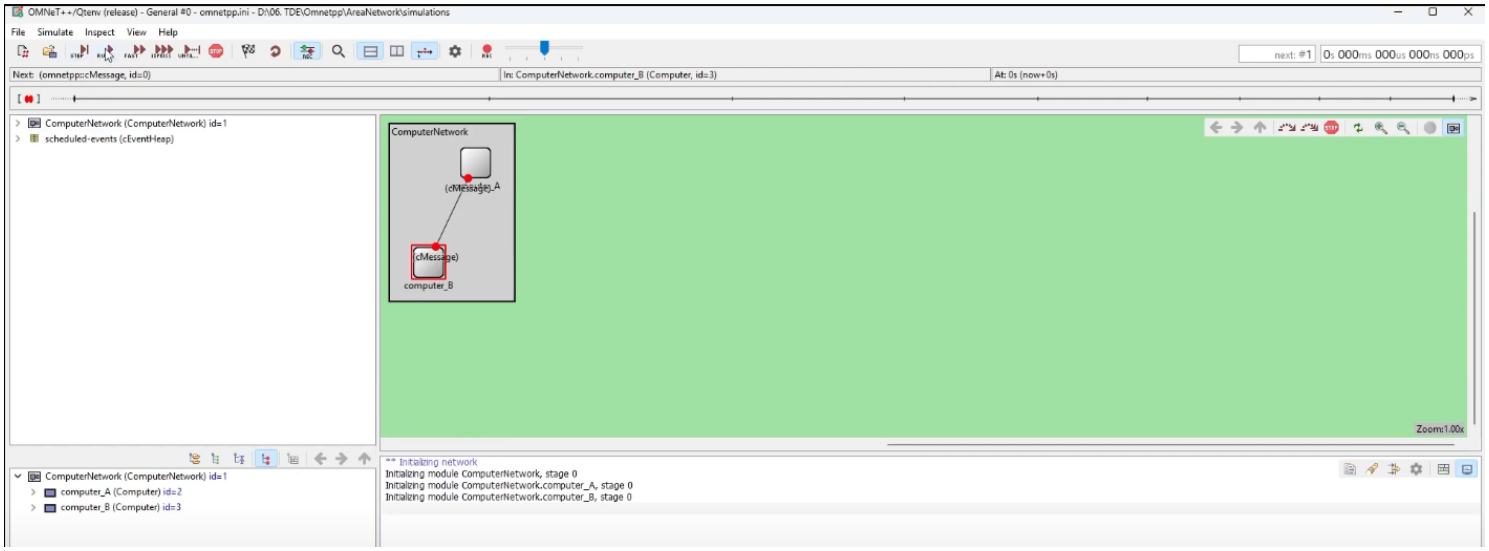
## 2.9. Tkenv: slikovno korisničko sučelje za simulacije

Tkenv je prijenosno slikovno korisničko sučelje s prozorima. Tkenv podupire inteaktivno izvođenje simulacije, njezino praćenje i otklanjanje pogrešaka. Tkenv se preporučuje u razvojnom koraku simulacije ili u prezentacijske i obrazovne svrhe, budući da omogućuje dobivanje opsežne slike stanja simulacije u bilo kojoj točki izvođenja i prati što se događa unutar mreže [1].

Najvažnije značajke su: animacija protoka poruka, slikovni prikaz brojidbe i izlaznih vektora tijekom simulacije, zasebni prozor za izlaz teksta svakoga modula, mogućnost gledanja zakazanih poruka u prozoru kako simulacija naprjeduje, događaj po događaj, prebacivanje između obične i brze izvedbe, obilježene prijekidne točke, prozori inspektora za ispitivanje i mijenjanje objekata i varijabala u modelu, to da se simulacija može ponovno pokrenuti, snimke (opsežno izvješće o modelu: objekti, varijable itd.) [1].

Tijekom izvođenja simulacije Tkenv omogućuje pregled rezultata, poput izlaznih vektora. Za prikaz rezultata mogu se rabiti histogrami i grafikoni vremenskih sljedova. Ovo čini ispitivanje pravilna rada simulacijskoga programa bržim i nudi izvrsno okružje za izvođenje pokusa s modelom dokle se izvodi. Tkenv ima mogućnost znatno ubrzati otklanjanje pogrješaka kada se rabi s uklanjivačima pogrješaka gdb ili xxgdb.

Tkenv je aplikacija za tzv. Tcl/Tk koja radi na svim sustavima na koje je Tcl/Tk prenesen, uključujući Windowse, Macintosh i Unix/X [1].




*Slika 6. Tkenv (Izvor: Vlastita izrada)*

# 3. Jezik NED

OMNeT++ uključuje Network Description Language (NED) kao temeljnu sastavnicu za definiranje mrežnih topologija i simulacijskih modela. Služeći kao deklarativni jezik, NED nudi apstrakciju visoke razine koja omogućuje programerima opisati građu i ponašanje mrežnih entiteta na sažet način.

Topologija modela specificirana je s pomoću NED-a. NED podupire modularni opis mreže. To znači da se opis mreže sastoji od nekoliko opisa sastavnica (kanala, tipova jednostavnih/složenih modula). Kanali, jednostavni moduli i složeni moduli jednoga opisa mreže mogu se ponovno rabiti u drugom opisu mreže. Kao posljedica toga jezik NED omogućuje korisnicima izgradnju vlastite knjižnice modula [1].

Datoteke koje sadržavaju opise mreže općenito imaju dometak „.ned“. Datoteke se NED-a ne rabe izravno: prevode se u kod u C++-u s pomoću kompilatora NEDC, a potom ih prevodi kompilator za C++ i povezuje u izvršnu datoteku simulacije [1].

## 3.1. Deklarativna skladnja

Rabeći jednostavnu i izravnu skladnju, korisnici NED-a mogu definirati module, njihove veze i konfiguracije. Ova metoda je u suprotnosti s imperativnim jezicima, gdje programeri moraju specificirati precizne korake koje bi simulacija trebala poduzeti da bi se dovršila. Dokle simulacijski mehanizam brine o specifičnostima „kako“ simulacija radi, NED omogućuje korisnicima odrediti „od čega“ je simulacija izgrađena i „kako“ sastavnice međusobno djeluju.

Korisnici mogu predočiti simulacijski model na organski i razumljiviji način zahvaljujući NED-ovoj deklarativnoj skladnji. Oni se mogu usredotočiti na definiranje građe i ponašanja mrežnih entiteta, a ne na proceduralne specifičnosti izvođenja simulacije.

NED ublažuje poteškoće u upravljanju izvođenjem simulacije nudeći opis simulacije više razine, čineći ju pristupačnijom i lakšom za porabu. Bez zaglavljivanja u potankostima tijeka kontrole simulacije istraživači mogu opisati međudjelovanja između mrežnih sastavnica, što dovodi do bistrijega i razumljivijega opisa simulacijskoga modela.

## 3.2. Hijerarhijska građa

Hijerarhijska građa koju podupire NED omogućuje korisnicima grupirati simulacijske modele u module i podmodule. Ova značajka poboljšava modularnost i bistrinu simulacije. Ugnježđivanjem modula korisnici mogu stvarati zamršene sustave, što model čini jednostavnijim za održavanje i razumijevanje.

Korisnici mogu poboljšati ustroj i mogućnost ponovne porabe svojih sastavnica simulacije razbijanjem simulacije na manje module kojima se lakše upravlja. Korisnici mogu učinkovito raditi s simulacijama velikih razmjera zahvaljujući hijerarhijskoj građi koja promiče bolju organizaciju koda i pojednostavljuje održavanje.

Korisnici se mogu usredotočiti na određene značajke simulacije, a da ih ne zatrpaju suvišne potankosti mogućnostî izgradnje hijerarhija. To će omogućiti istraživačima razviti preciznije i korisnije mrežno modeliranje boljim razumijevanjem cjelokupne arhitekture simulacije i međudjelovanjâ.

## 3.3. Identifikatori

Identifikatori su nazivi modula, kanala, mreža, podmodula, parametara, vrata, atributa kanala i funkcija.

Identifikatori se moraju sastojati od slovâ engleske abecede (a-z, A-Z), brojaka (0-9) i dolnje crte „_". Identifikatori mogu započeti samo slovom ili dolnjom crtom. Ako se identifikator želi započeti znamenkom, tomu nazivu, koji se želi, dodati je dolnju crtu, npr. „_3Com".

Ako ima identifikatora koji se sastoje od nekoliko riječi, po ustaljenosti se početak svake riječi piše velikim slovom. Isto tako, preporučuju se nazivi modula, kanala i mreža započeti velikim slovom, a nazivi parametara, vrata i podmodula malim slovom. Dolnje se crte rijetko rabe [1].

## 3.4. Uvoz

Deklaracije iz druge datoteke opisa mreže mogu se uvesti s pomoću naredbe za uvoz. Moguće je rabiti članove (kanale, tipove jednostavnih/složenih modula) definirane u opisu mreže, nakon uvoza. Prigodom uvoza datoteke potrebne su samo informacije o deklaraciji. Osim toga, prigodom uvoza datoteke .ned datoteka se ne prevodi automatski

kompilatorom za NED; umjesto toga sve datoteke opisa mreže moraju biti prevedene i povezane, i to ne samo one najviše razine.

Nazivi datoteka mogu se specificirati s proširkom .ned ili bez njega. Osim toga, mogu se imenovati kazala u kojima se nalaze uvezene datoteke uključivanjem staze u nazive datoteka, ili još bolje, rabiti opcija naredbenoga retka „-I path>“ kompilatora za NED.

## 3.5. Definiranje jednostavnih modula

Jednostavni moduli osnovni su građevni blokovi za druge (složene) module. Tipovi jednostavnih modula identificirani su nazivima. Po ustaljenosti nazivi modula počinju velikim slovima. Jednostavan se modul definira deklaracijom njegovih parametara i vratâ [1].

```
4  simple Test2
5  {
6      gates:
7          input in;
8          output out;
9  }
10
11 simple Test1
12 {
13     gates:
14         input in;
15         output out;
16
17 }
18
19 simple Test3
20 {
21 }
```

*Slika 7. Definicija jednostavnog modula (Izvor: Vlastita izrada)*

### 3.5.1. Parametri jednostavna modula

Varijable koje su dijelom modula nazivlju se parametrima. Jednostavni algoritmi modula mogu postavljati upite i rabiti jednostavne parametre modula. Na primjer, broj poruka koje bi trebao generirati modul pod nazivom „TrafficGen“ određen je parametrom koji se zove „numOfMessages“. Nazivi se rabe za identifikaciju parametara. Nazivi parametara često počinju malim slovima.

Dio opisa modula „parameters:“ je mjesto gdje se parametri deklariraju navođenjem njihovih naziva. Tip parametra može biti naveden kao koji bilo tip, „bool“, „bool const“, „brojevni“, „brojevni const“ (ili jednostavno „const“) ili niz.

Ako je tip parametra izostavljen, pretpostavlja se brojevni. Praktično, to znamenuje da se treba samo izrijekom navesti tip za niz, „bool“- ili „char“-vrijednost parametara. Neka se ima na umu da se stvarne vrijednosti parametara daju poslije, kad se modul rabi kao građevni blok tipa složenoga modula ili kao modul sustava [1].

```
16 simple TemperatureSensor
17 {
18     parameters:
19         double minTemperature;
20         double maxTemperature;
21     gates:
22         output out;
23 }
24
25 simple HumiditySensor
26 {
27     parameters:
28         double minHumidity;
29         double maxHumidity;
30     gates:
31         output out;
32 }
```

*Slika 8. Parametri (Izvor: Vlastita izrada)*

## 3.6. Expressions (izrazi)

Korisnici mogu provoditi izračune i procjene unutar deklarativnoga okvira simulacijskoga modela zahvaljujući NED Expressionsima, snažnoj značajki NED-a. Ovi izrazi simulacijskomu okružju daju dinamične i prilagodljive značajke bez potrebe za izravnim programiranjem u C++-u. Matematičke operacije za izračun, uvjetni izrazi za izbor i pristup parametrima modula za simulacije s međudjelovanjem vrijedne su značajke tih izraza.

„Izraze“ korisnici mogu rabiti za izradbu dinamičnih simulacijskih modela koji odgovaraju na promjenjive situacije čim se pojave. Kao rezultat potpore jezika NED za matematičke operacije kao što su sinus, kosinus i „e na potenciju“, korisnici mogu brzo izvršiti zamršene izračune. Izrazi omogućuju manipulaciju nizovima, omogućujući korisnicima mijenjati rezultate simulacije i proizvoditi zanimljive biljege u simulaciji.

## 3.7. GNED – Graphical NED Editor

GNED Editor omogućuje slikovno osmišljavanje složenih modula. GNED radi s datotekama NED – ne rabi nikakvo neugodno unutarnje obličje datoteke. Može se ukrcati koja bilo od postojećih datoteka u NED-u, onda se urede složeni moduli u njoj slikovno i za tim pohrani datoteka natrag. Ostatak stvari u datoteki u NED-u (jednostavni moduli, kanali, mreže itd.) će preživjeti tu radnju. GNED stavlja sve podatke koji se odnose na slike u tzv. prikazne nizove [1].

Kad se pohrani datoteka, GNED ponovno generira tekst u NED-u raščlanjujući datoteku u NED-u u unutarnju podatkovnu strukturu. Jedan rezultat toga je da će uvoz biti „kanoniziran“; što vrijedi da GNED osigurava da je datoteka u NED-u dobro građena i slikom dosljedna, što korisnicima olakšava čitanje i razumijevanje simulacijskoga modela. Pače i ako se promijeni slikovni prikaz do krajnjih granica uklanjanjem/dodavanjem podmodula, vrata, parametara, veza itd., komentari u izvornom NED-u se održavaju, budući da ih kompilator pridružuje članovima u NED-u kojima odgovaraju.

Vizualno oruđe GNED već je dulje vrijeme potpuno dvosmjerno. Uvijek se može prebaciti na izvorni prikaz u NED-u dok se uređuje grafika, mogu se učiniti promjene ondje i za tim se može vratiti na grafiku.

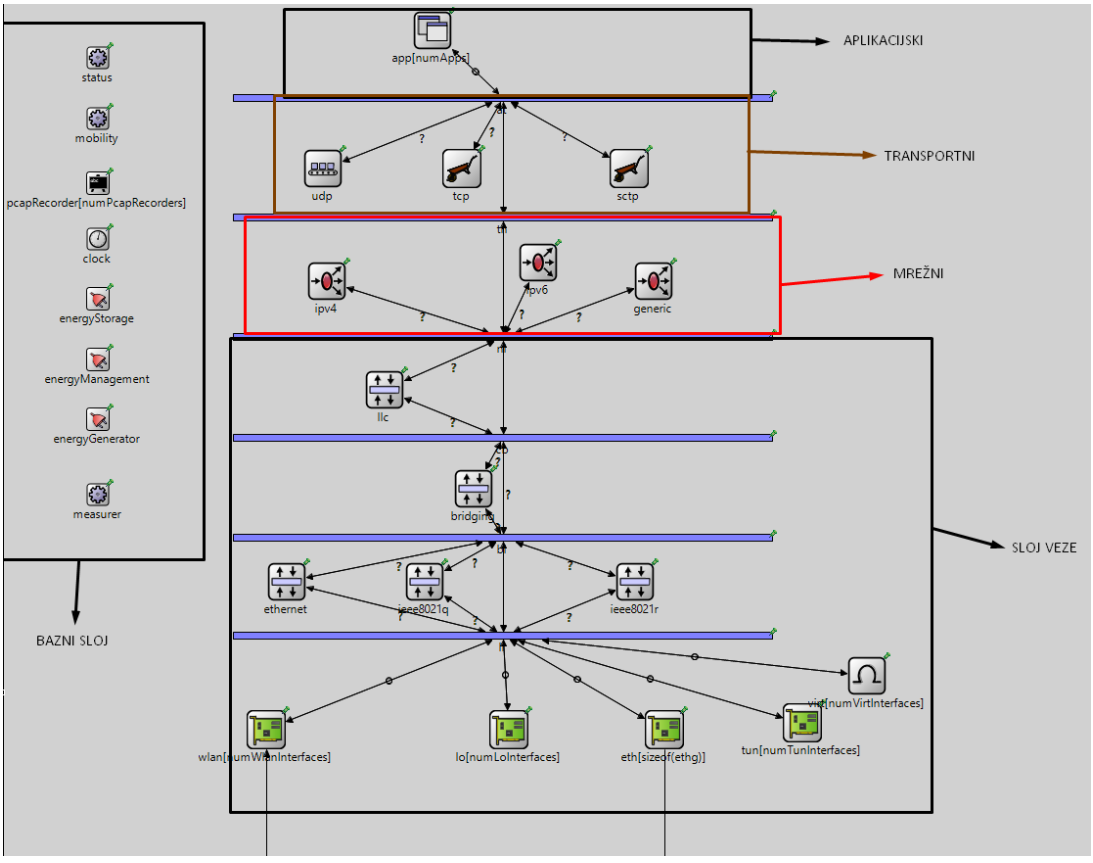
# 4. Igraća konzola

Zadatak je bio ostvariti jednostavan programski modul uz pomoć mehanizama koji ne zahtijevaju programiranje u programskom jeziku C++. Iako je za malo pa nemoguće kreirati jednostavne i složene module koji su spremni za simulaciju bez programiranja, moguće je izraditi njihovu „ljusku“ i definirati im parametre s kojima se želi raditi, ali za implementaciju logike rada potrebno je rabiti programski jezik C++.

Za izradbu igraće konzole odlučeno je za drugačiji pristup samomu zadatku. Rabeći prije stečena znanja znalo se je da igraća konzola i računalo dijele puno zajedničkih značajaka i imaju sličnu svrhu. Instaliravši INET dobiven je pristup različnim modulima koji su već definirani u jezicima NED i C++ te su pripravni za porabu.

Pretražujući INET odabrano je nekoliko modula koji su se činili prikladnim za ove potrebe te je odlučeno za modul StandardHost. StandardHost je model računala koje ima Ethernet i pristup Wi-Fiju te rabi protokole TCP, UDP i SCTP. Sve su te specifikacije su isto tako potrebne i za igraću konzolu.

Igraća konzola sastoji se od 5 slojeva, a to su: primjenski, prijenosni, mrežni, sloj veze i temeljni sloj. Temeljni sloj sadržava najosnovniju infrastrukturu za mrežne čvorove te nije kruto spregnut s komunikacijskim protokolom.




*Slika 9. Dizajn igraće konzole (Izvor: Vlastita izrada)*

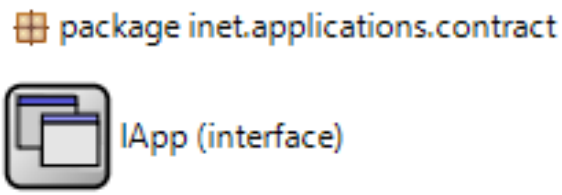
```
double cpuFrequency @unit("GHz") = default(1.02 GHz);
int cpuCores = default(4);
double gpuFrequency @unit("GHz") = default(0.92 GHz);
double ramCapacity @unit("GB") = default(4 GB);
double storageCapacity @unit("GB") = default(32 GB);


double batteryCapacity @unit("mAh") = default(4310 mAh);
double batteryChargingSpeed @unit("mAh/s") = default(1000 mAh/s);
int batteryChargeCycles = default(800);
double batteryHealth = default(0.95);
bool batteryOverchargeProtection = default(true);
double autoShutdownTimeout @unit("s") = default(1800s);
```

*Slika 10. Parametri igraće konzole (Izvor: Vlastita izrada)*

Da bih sama konzola bila što stvarnija nisu mijenjani parametri postojećeg modula već su joj dodane specifikacije već postojeće konzole u stvarnom svijetu naziva Nintendo Switch. Ova konzola ima njezine stvarne značajke. Samim tim korisnik koji želi ovu konzolu porabiti tijekom simulacije ima na raspolaganju ove specifikacije s kojima može izvoditi pokuse i simulirati stvarno ponašanje igraće konzole u virtualnom simulacijskom okružju. Iako se ovim parametrima dade puno pokusa izvesti i mogu se raditi razne simulacije, samu logiku rada nije moguće ostvariti preko NED-a nego je potrebno rabiti C++, što nije bila tema ovoga rada

## 4.1. Primjenski sloj

U primjenskom sloju nalazi se podmodul IApp. IApp je sučelje koje definira osnovne funkcionalnosti i metode koje trebaju biti nazočne u svakoj aplikaciji koja će se modelirati u simulaciji. Poraba ovoga sučelja u potpunosti olakšava stvaranje aplikacija jer omogućuje programerima i korisnicima usredotočiti se na ostvaraj potankosti rada aplikacije, a ne na samu izradbu rečene. Primjenski je sloj izravno povezan s prijenosnim slojem konzole.




*Slika 11. IApp (Izvor: Vlastita izrada)*

## 4.2. Prijenosni sloj

Prijenosni sloj ima tri komunikacijska protokola: TCP, UDP i SCTP. TCP je protokol koji se rabi za pouzdan prijenos podataka između uređaja. Osigurava ispravnost i potpunost podataka, a u slučaju da se podatci oštete ili izgube TCP ih ponovno šalje na prvobitno odredište. Kako bi se izbjegnule zagušenosti na mreži, TCP prilagođuje brzinu prijenosa podataka. Što se tiče same igraće konzole, TCP tu ima veliku ulogu za prijenos vrlo bitnih podataka tijekom igranja igara kao što su položaji i kretanja igrača, naredbe i stanja.

UDP je protokol koji omogućuje iznimno brz prijenos podataka. UDP radi na načelu da ne ostvaruje vezu prije prijenosa podataka nego samo šalje podatke na određeni izvor; takov način prijenosa može uzrokovati gubitak paketa u prijenosu. Taj način prijenosa čini ga jednostavnim za ostvaraj i koristan je za aplikacije koje ne zahtijevaju potpunu pouzdanost i redoslijed doprave paketa. U „gamingu“ ovaj protokol rabi se za prijenos video podataka kao što je prijenos uživo, i u prijenosu audio podataka  u stvarnom vremenu.

Protokol SCTP osmišljen je tako da kombinira značajke protokola UDP i TCP. Sličan je TCP-u, ali isto tako pruža prijenos podataka orijentiran na poruke poput UDP-a. Isto tako, za razliku od TCP-a, SCTP osigurava potpuni istodobni prijenos nekoliko tokova podataka između povezanih krajnjih točaka. Isto je tako učinkovitiji od TCP-a kad je u pitanju preuređivanje podataka, čim se poslužitelj oslobađa nepotrebne opterećenosti preuređivanjem. SCTP može biti koristan za složenije igre preko interneta koje zahtijevaju višestruke tokove podataka kako bi se održala pouzdana i stabilna komunikacija između igrača.

```
module TransportLayerNodeBase extends NetworkLayerNodeBase
{
    parameters:
        bool hasUdp = default(firstAvailableOrEmpty("Udp") != "");
        bool hasTcp = default(firstAvailableOrEmpty("Tcp", "TcpLwip", "TcpNsc") != "");
        bool hasSctp = default(false);
        @figure[transportLayer](type=rectangle; pos=250,158; size=1000,134; fillColor=#ff0000; lineColor=#808080; cornerRadius=5; fillOpacity=0.1);
        @figure[transportLayer.title](type=text; pos=1245,163; anchor=ne; text="transport layer");
    submodules:
        udp: <default(firstAvailableOrEmpty("Udp"))> like IUdp if hasUdp {
            @display("p=375,225");
        }
        tcp: <default(firstAvailableOrEmpty("Tcp", "TcpLwip", "TcpNsc"))> like ITcp if hasTcp {
            @display("p=525,225");
        }
        sctp: <default(firstAvailableOrEmpty("Sctp"))> like ISctp if hasSctp {
            @display("p=675,225");
        }
        tn: MessageDispatcher {
            @display("p=750,300;b=1000,5,,,,1");
        }
    connections allowunconnected:
        udp.ipOut --> tn.in++ if hasUdp;
        udp.ipIn <-- tn.out++ if hasUdp;

        tcp.ipOut --> tn.in++ if hasTcp;
        tcp.ipIn <-- tn.out++ if hasTcp;

        sctp.ipOut --> tn.in++ if hasSctp;
        tn.out++ --> sctp.ipIn if hasSctp;

        tn.out++ --> ipv4.transportIn if hasIpv4;
        tn.in++ <-- ipv4.transportOut if hasIpv4;

        tn.out++ --> ipv6.transportIn if hasIpv6;
        tn.in++ <-- ipv6.transportOut if hasIpv6;

        tn.out++ --> generic.transportIn if hasGn;
        tn.in++ <-- generic.transportOut if hasGn;

        tn.out++ --> nl.in++;
        tn.in++ <-- nl.out++;
}
```

*Slika 12. Prijenosni sloj (Izvor: Vlastita izrada)*

## 4.3. Mrežni sloj i sloj veze

Ovaj modul definira ponašanje i međudjelovanje između različnih slojeva mrežnoga protokola (kao što su IPv4, IPv6 i generični modul) unutar danoga čvora u mreži. IPv4, IPv6 i generični modul omogućuju komunikaciju između gornjega i donjega sloja mrežnoga protokola. Ova modularna struktura omogućuje razdvajanje odgovornosti i komunikaciju između različnih slojeva mrežnoga protokola, što je ključno za pravilno funkcioniranje složenih mrežnih sustava.

U sloju veze definirana su sva sučelja koja ova konzola posjeduje. Konzola posjeduje Ethernetovo sučelje koje joj omogućuje povezivanje na mrežu preko kabela te omogućuje slanje i primanje Ethernetovih okvira i upravljanje adresama MAC. ITunnelInteface je sučelje za simulaciju tunelskih sučelja. Porabom prividnoga tunela promet iz jedne mreže može se

prenositi preko druge mreže rabeći sučelja tunela. U ovo sučelje uključene su metode za upravljanje tunelima i prijenos podataka preko njih. ILoopbackInteface je sučelje za oponašanje mrežnoga povratnoga sučelja. Bez porabe stvarne mreže konzola može komunicirati sama sobom s pomoću mrežnoga sučelja tzv. povratne petlje. IVirtualInteface: osobine i procedure za simulaciju prividnih mrežnih sučelja definirane su ovim sučeljem. Prividna sučelja su apstraktne konceptualne točke za komunikaciju sustava. Ova sučelja omogućuju prividno modeliranje različnih mrežnih članova koji su neovisni o fizičkoj infrastrukturi.

## 4.4. QSS

Svaki od prije navedenih slojeva povezan je s modulom koji je nazvan QSS. Ovaj modul međusobno povezuje više aplikacija, protokola i sučelja te automatski šalje poruke i pakete između njih. Omogućuje mnogo različnih konfiguracija od slojevitih arhitektura gdje dispečeri poruka odvajaju različite komunikacijske slojeve do centraliziranih arhitektura gdje je jedan dispečer poruka povezan s svim sastavnicama. Ovaj modul je jedan od najbitnijih kod igraće konzole jer je za nju iznimno bitno brzo i pouzdano slanje i primanje podataka uz što manje kašnjenje i uz što manje smetnje i gubitke. Uz to još je i bitan prioritet samoga primanja i slanja u konzoli i na mreži. Modificirajući ovaj modul dodavši mu parametre pobrinulo se je da je ponašanje konzole što stvarnije tijekom simulacije.

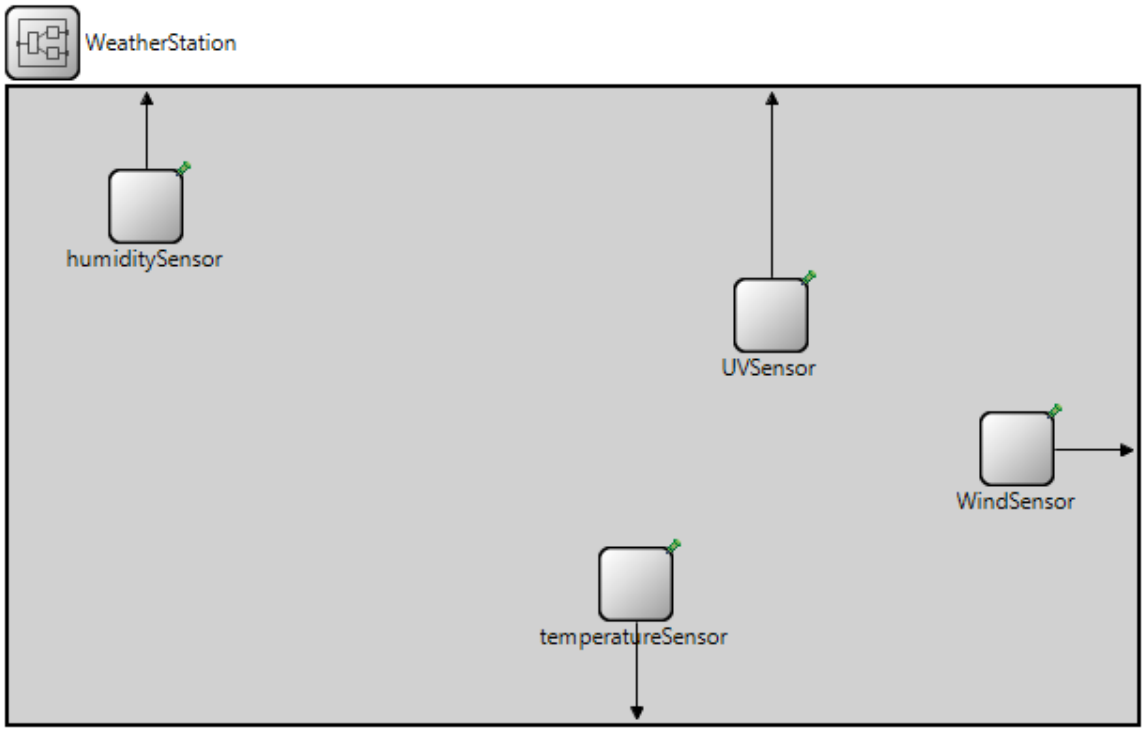
```
parameters:

    double latencyTolerance = default(20);
    double packetLossTolerance = default(0.01);
    double requiredBandwidth = default(1000000);
    double gamingTrafficRate = default(100000000);

    double connectionTimeout = default(5);
    int maxRetries = default(3);
    double pingInterval = default(1);
    double gameUpdateInterval = default(0.1);
    double maxInputDelay = default(0.05);

    bool useQSS = default(true);
    int maxQSSIterations = default(100);
    double qssTolerance = default(1e-6);
```

*Slika 13. QSS (Izvor: Vlastita izrada)*

## 4.5. Meteorološka postaja

Dodatno glavnomu zadatku ovoga završnoga rada ostvarena je meteorološka postaja rabeći samo jezik NED. Ova meteorološka postaja osmišljena je tako da sadržava različne vrste osjetnika (očutnika, senzora), od kojih je svaki odgovoran za mjerenje specifičnih parametara povezanih s vremenom. Ovi osjetnici uključuju temperaturni osjetnik, osjetnik vlažnosti zraka, osjetnik UV-zračenja i osjetnik brzine vjetra.

Senzor temperature ima u naprijed postavljen najmanji i najveći temperaturni raspon i programiran je za precizno mjerenje temperature. Ovaj raspon jamči da senzor proizvodi vrijednosti temperature koje su unutar u naprijed određenih granica. Senzor vlažnosti zraka dobro radi unutar postavljenoga raspona vrijednosti vlažnosti i slično se usredotočuje na procjenu razine vlažnosti. Dok daje mjerenja unutar korisnički definiranoga raspona UV-indeksa, senzor UV-zračenja se usredotočuje na raščlambu UV-indeksa. Konačno, senzor brzine vjetra pruža mjerenja koja su unutar određenoga najmanjega i najvećega raspona brzine i specijaliziran je za kvantificiranje brzine vjetra. Svi ovi senzori, bez obzira na njihove različne funkcije, dijele zajedničku značajku: „out gate“. Ova vrata služe kao kanal kroz koji senzori prenose svoje prikupljene podatke središnjoj jedinici za obradbu.




*Slika 14. Meteorološka postaja (Izvor: Vlastita izrada)*

Modul WeatherStation je srce simulacije, uključujući sve podmodule senzora i njihove povezane funkcije. Prihvaća nekoliko korisnički definiranih vrijednosti, kao što su najmanje i najveće granice za temperaturu, vlažnost, UV-indeks i brzinu vjetra.

Svaki od navedenih senzora stvoren je kao jednostavni modul naredbom „simple“ u jeziku NED. Da bi se senzori i sama meteorološka postaja mogli rabiti za simulacije, potrebno je zasebno s pomoću naredbe „define“ definirati svaki od tih senzora rabeći programski jezik C++. Tema ovoga rada je stvaranje jednostavnoga modula bez porabe programskoga jezika C++. Zbog teme ovoga rada nisu se mogli definirati moduli, implementirati sama logika za slanje podataka iz senzora u modul te se nije mogla definirati logika meteorološke postaje. Bez tih definicija nije moguće prikazati rad postaje preko simulacija.

```
simple TemperatureSensor
{
    parameters:
        double minTemperature;
        double maxTemperature;
    gates:
        output out;
}

simple HumiditySensor
{
    parameters:
        double minHumidity;
        double maxHumidity;
    gates:
        output out;
}

simple UVSensor
{
    parameters:
        double minUVIndex;
        double maxUVIndex;
    gates:
        output out;
}
simple WindSpeedSensor
{
    parameters:
        double minSpeed;
        double maxSpeed;
    gates:
        output out;

}
```

```
module WeatherStation
{
    parameters:
        double minTemperature;
        double maxTemperature;
        double minHumidity;
        double maxHumidity;
        double minUVIndex;
        double maxUVIndex;
        double minSpeed;
        double maxSpeed;
    gates:
        output out @loose;
        inout port @allowUnconnected;
    submodules:
        temperatureSensor: TemperatureSensor {
            minTemperature = minTemperature;
            maxTemperature = maxTemperature;
            @display("p=334,264");
        }
        humiditySensor: HumiditySensor {
            minHumidity = minHumidity;
            maxHumidity = maxHumidity;
            @display("p=73,63");
        }
        UVSensor: UVSensor {
            minUVIndex = minUVIndex;
            maxUVIndex = maxUVIndex;
            @display("p=406,121");
        }
        WindSensor: WindSpeedSensor {
            minSpeed = minSpeed;
            maxSpeed = maxSpeed;

            @display("p=537,192");
        }

    connections allowunconnected:
        temperatureSensor.out --> out;
        humiditySensor.out --> out;
        UVSensor.out --> out;
        WindSensor.out --> out;
}
```

*Slika 15. Kod meteorološke postaje (Izvor: Vlastita izrada)*

## 4.6. Izradba jednostavne meteorološke postaje s pomoću programskoga jezika Python i pokretanje skripte u OMNeT++-a

Koristeći programski jezik Python, razvijen je programski kod koji se može izvršavati unutar ugrađenoga teminala OMNeT++ simulatora. Ovaj kod, tijekom svoje izvršne faze, proizvodi različite datoteke dodajući im specifične parametre koji su ključni za uspostavu meteorološke postaje te omogućuju pokretanje simulacija povezanih s tom postajom.

Za pravilno izvršavanje ovog programa neophodno je unutar direktorija glavnoga projekta stvoriti mapu imenovanu „scripts“. Upravo u ovu mapu treba dodati pripravljenu Python-skriptu.

Za pokretanje Python-skripte smještene u mapi scripts unutar OMNeT++ okružja prvo je potrebno otvoriti „Git Bash" terminal. Terminal se može otvoriti rabeći oruđnu vrpcu OMNeT++-sučelja ili preko brza tipkovničkoga prečaca Ctrl+Alt+Shift+T. Nakon što je terminal aktiviran, sljedeći korak uključuje navigaciju do željenoga direktorija s pomoću promjene staze. Precizna staza do direktorija varira ovisno o specifičnoj instalaciji i konfiguraciji simulatora. Za potrebe ovoga završnoga rada primjer naredbe za promjenu staze glasi: cd C:/Users/x/Desktop/omnetpp-6.0/samples/zavrsni/script. Nakon promjene staze potrebno je pokrenuti Python skriptu unosom sljedeće naredbe u terminal: python WeatherStationSimulation.py .

Ovim postupkom se inicira izvođenje skripte, koja zatim „intrigira“ s OMNeT++-simulatorom po definiranim uputama i parametrima unutar skripte. Kad se ova skripta pokrene preko simulatora, automatski se generira nova mapa pod nazivom „WeatherStationSimulation". Datoteke koje skripta kreira u novonastaloj mapi potrebno je premjestiti u src-mapu projekta u kojem se želi rabiti. Ovaj postupak omogućuje učinkovito stvaranje simulacijskoga okružja za meteorološku postaju, omogućujući korisniku da potanko prilagodi parametre simulacije sukladno specifičnim zahtjevima ili scenarijima ispitivanja.

Pravi programski kod Python-skripte nahodi se priložen uz završni rad.

```
import os

# Definiranje strukture direktorija
project_name = "WeatherStationSimulation"
base_dir = os.path.join(os.getcwd(), project_name)
os.makedirs(base_dir, exist_ok=True)
src_dir = os.path.join(base_dir, "src")
os.makedirs(src_dir, exist_ok=True)
```

*Slika 16. Python-kod za definiranje građe direktorija meteorološke postaje (Izvor: Vlastita izrada)*

```
#Kreiranje datoteka potrebnih za postaju i simulaciju
files = {
    "WeatherStation.ned": """
simple WeatherStation {
    parameters:
        double initialTemperature = default(20);
        double initialHumidity = default(50);
    gates:
        output out;
}
""",
    "WeatherData.msg": """
message WeatherData {
    double temperature;
    double humidity;
}
""",
    "WeatherStation.h": """
#ifndef WEATHERSTATION_H_
#define WEATHERSTATION_H_

#include <omnetpp.h>
#include "WeatherData_m.h"

using namespace omnetpp;

class WeatherStation : public cSimpleModule {
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

#endif /* WEATHERSTATION_H_ */
""",
    "WeatherStation.cc": """
#include "WeatherStation.h"
#include "WeatherData_m.h"

Define_Module(WeatherStation);

void WeatherStation::initialize() {
    // Zakaži prvu poruku za pokretanje periodičnog slanja
    cMessage *timerMsg = new cMessage("timer");
    scheduleAt(simTime() + 1.0, timerMsg); // Sends data every second
}

void WeatherStation::handleMessage(cMessage *msg) {
    if (msg->isSelfMessage()) {
        // Stvori novu WeatherData poruku
        WeatherData *weatherData = new WeatherData();
        weatherData->setTemperature(20.0); // Fixed temperature
        weatherData->setHumidity(50.0); // Fixed humidity

        // Pošalji poruku
        send(weatherData, "out");

        // Zakažiti sljedeće slanje
        scheduleAt(simTime() + 1.0, msg);
    }
}
"""
```

*Slika 17. 1. dio Python-koda meteorološke postaje (Izvor: Vlastita izrada)*

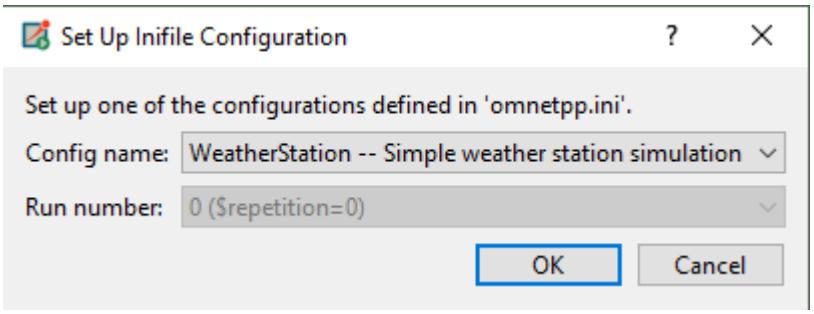
```
    """,
    "Receiver.h": """
#ifndef RECEIVER_H_
#define RECEIVER_H_

#include <omnetpp.h>
#include "WeatherData_m.h"

using namespace omnetpp;

class Receiver : public cSimpleModule {
  protected:
    virtual void handleMessage(cMessage *msg) override;
};

#endif /* RECEIVER_H_ */
""",
    "Receiver.cc": """
#include "Receiver.h"
#include "WeatherData_m.h"

Define_Module(Receiver);

void Receiver::handleMessage(cMessage *msg) {
    WeatherData *weatherData = dynamic_cast<WeatherData *>(msg);
    if (weatherData != nullptr) {
        // Zabilježi primljenu temperaturu i vlažnost
        EV << "Received weather data - Temperature: " << weatherData->getTemperature()
           << "°C, Humidity: " << weatherData->getHumidity() << "%" << endl;
    }
    delete msg;
}
""",
    "omnetpp.ini": """
[General]
network = WeatherNetwork
**.initialTemperature = 22
**.initialHumidity = 60
sim-time-limit = 5s

[Config WeatherStation]
description = "Simple weather station simulation"
""",
    "WeatherNetwork.ned": """
simple Receiver
{
    parameters:
        @display("i=device/laptop");
    gates:
        input in; // Ulazni gate za primanje podataka
}
network WeatherNetwork {
    submodules:
        weatherStation: WeatherStation {
            parameters:
                initialTemperature = default(22);
                initialHumidity = default(60);
            @display("p=100,100");**.initialTemperature = 22
        };
        receiver: Receiver {
            @display("p=100,300");
        };
    connections:
        weatherStation.out --> receiver.in;
}
"""
}
#Kreiranje datoteka na disku
for filename, content in files.items():
    with open(os.path.join(src_dir, filename), 'w') as f:
        f.write(content)

print(f"Project {project_name} has been generated at {base_dir}")
```

*Slika 18. 2. dio Python-koda meteorološke postaje (Izvor: Vlastita izrada)*

## 4.7. Simulacija meteorološke postaje

Kako bi sama simulacija radila važno je napomenuti da je potrebno izbrisati sve datoteke koje su predefinirane od strane OMNeT++ da bi se simulacija mogla pokrenuti bez ikakvih grašaka te je potrebno omogućiti INET u projektu.

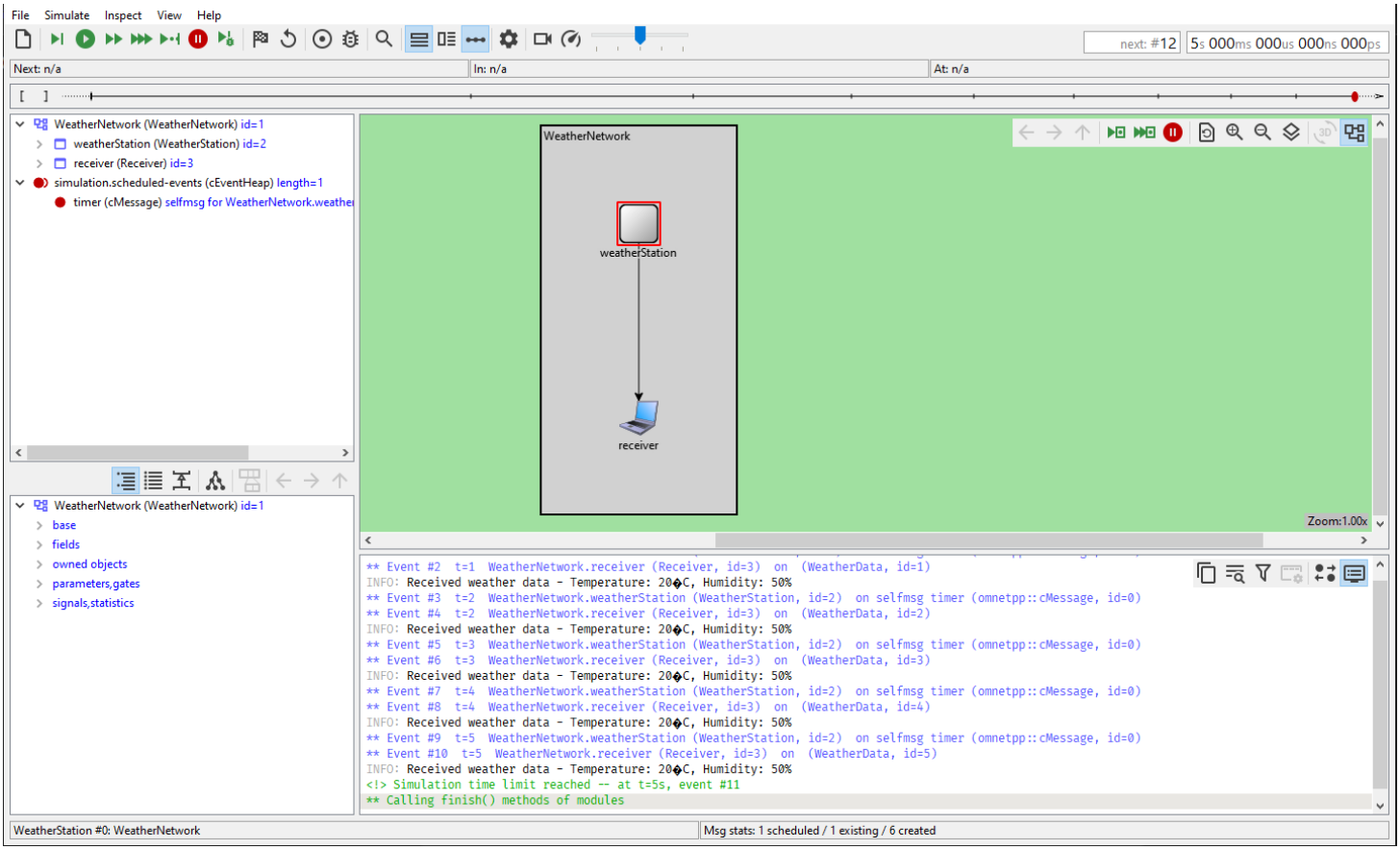
Pokretanjem simulacije preko tipke „Run“ i odabirom omnetpp.ini-datoteke simulacija počinje s inicijalizacijom modula. WeatherStation-modul se inicijalizira i odmah planira slanje prve poruke (WeatherData), koja se planira za slanje u simulacijskom vremenu 0. Također, postavljene su inicijalne vrijednosti temperature i vlažnosti zraka koje su definirane u omnetpp.ini-datoteki.




*Slika 19. Pokretanje simulacije (Izvor: Vlastita izrada)*

Kad simulacija dosegne vrijeme slanja prve poruke, WeatherStation modul stvara i šalje WeatherData-poruku prema Receiver-modulu. Poruka sadržava podatke o temperaturi i vlažnosti koje su definirane ili kao inicijalne vrijednosti u WeatherStation-modulu ili preko parametara u omnetpp.ini-datoteki. Receiver-modul prima poruku i obrađuje ju. U ovom slučaju obradba se svodi na ispis informacija o primljenoj poruci u konzolu (ili Event Log u OMNeT++ GUI-u). Nakon obradbe poruka se briše kako bi se oslobodio memorijski prostor.

Simulacija se automatski zaustavlja nakon 5 simulacijskih sekunda zbog parametra „sim-time-limit = 5s“ postavljenoga u omnetpp.ini-datoteci. To dođe da će simulacija prestati s radom i bez dodatnih radnjâ nakon što simulacijsko vrijeme dosegne 5 sekunda.




*Slika 20. Simulacija meteorološke postaje (Izvor: Vlastita izrada)*

# 5. Zapakiravanje modula i objava široj javnosti

Obavljeno je istraživanje pretražujući Internet te se je došlo do zaključka da za OMNeT++ ne postoji službeni repozitorij gdje ljudi mogu objavljivati svoje radove koje su izradili u samom simulatoru. Naišlo se je na nekolicinu radova koje su kreatori objavili na GitHub-u.

GitHub je platforma za razvoj programske potpore koja omogućuje programerima pohranu i upravljanje kodom, a temeljna je na tehnologiji oblaka. GitHub nudi mogućnost da rad može biti javno dostupan. Kad je rad javno dostupan vlasnik rada može primati kritike i zamisli drugih korisnika platforme što pomaže u daljem razvoju i napredovanju projekta ili rada.

S pomoću postojećega privatnoga računa na platformi GitHub stvoren je vlastiti privatni repozitorij pod nazivom „omnetpp“ jer nije moguće rabiti posebne znakove kao npr. ++ prigodom dodjeljivanja naziva repozitoriju. Repozitoriju je dodan opis na engleskom jeziku da bi projekti i njihov opis bili razumljivi ljudima širom svijeta.

Projekti su objavljeni i zapakirani na način kako su i drugi korisnici objavili. Pronašao sam mapu gdje su projekti iz simulatora pohranjeni te sam potrebne dokumente projekta objavio u svojem repozitoriju. To znamenuje da drugi korisnici mogu preuzeti dokumente projekata te ih pohraniti u mapu gdje njihov simulator pohranjuje i pripravni su za porabu i dalji razvoj projekata

## 5.1. Uvoz modula u OMNeT++

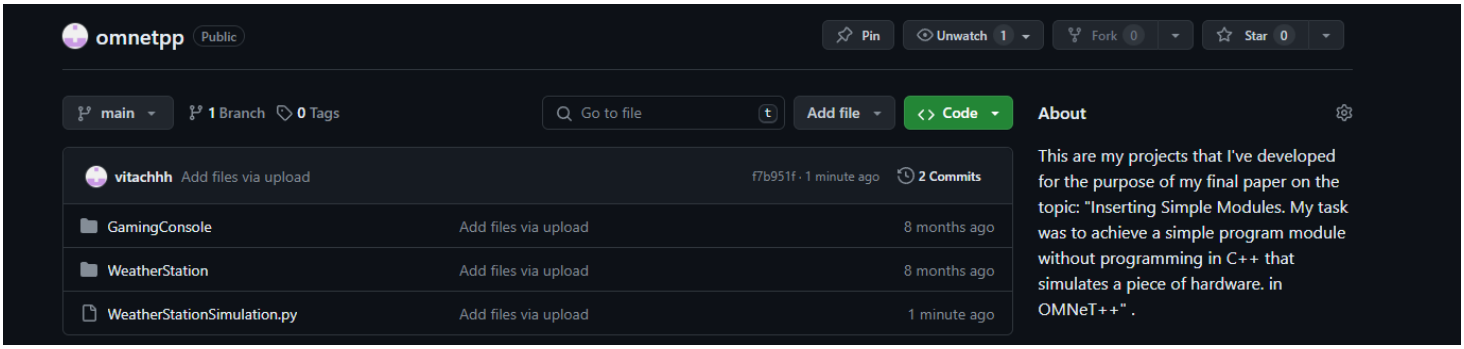
Nakon preuzimanja modula potrebno je modul ubaciti u OMNeT++-simulator. Ako se modul nalazi u .zipu ili .rar datoteci potrebno ga je prvo raspakirati. Nakon raspakiravanja mapu modula potrebno je ubaciti u OMNeT++-simulator. Postoje 2 načina ubacivanja modula u simulator.

Prvi način je ubacivanje mape modula izravno u korijensko (root) kazalo trenutačnoga workspacea. To se može postignuti tako da se mapa modula premjesti u kazalo „omnetpp-X\samples“ ter će se prikazati u simulatoru u „Project Exploreru“.

Drugi način je uvoz modula preko samoga simulatora. Kad se pokrene simulator tada je u oruđnoj vrpci potrebno odabrati: „File > Import, General > Existing Projects into Workspace“ i pritisnuti tipku „Next.“ U dijelu „Select root directory“, rabeći „Browse“ treba

pronaći direktorij gdje se nahodi projekt ili modul koji želimo uvesti te pritiskom na tipku „Finish“ uvažamo modul u trenutačni workspace.

## 5.2. Poraba modula u drugim projektima u OMNeT++-u




*Slika 21. Repozitorij*

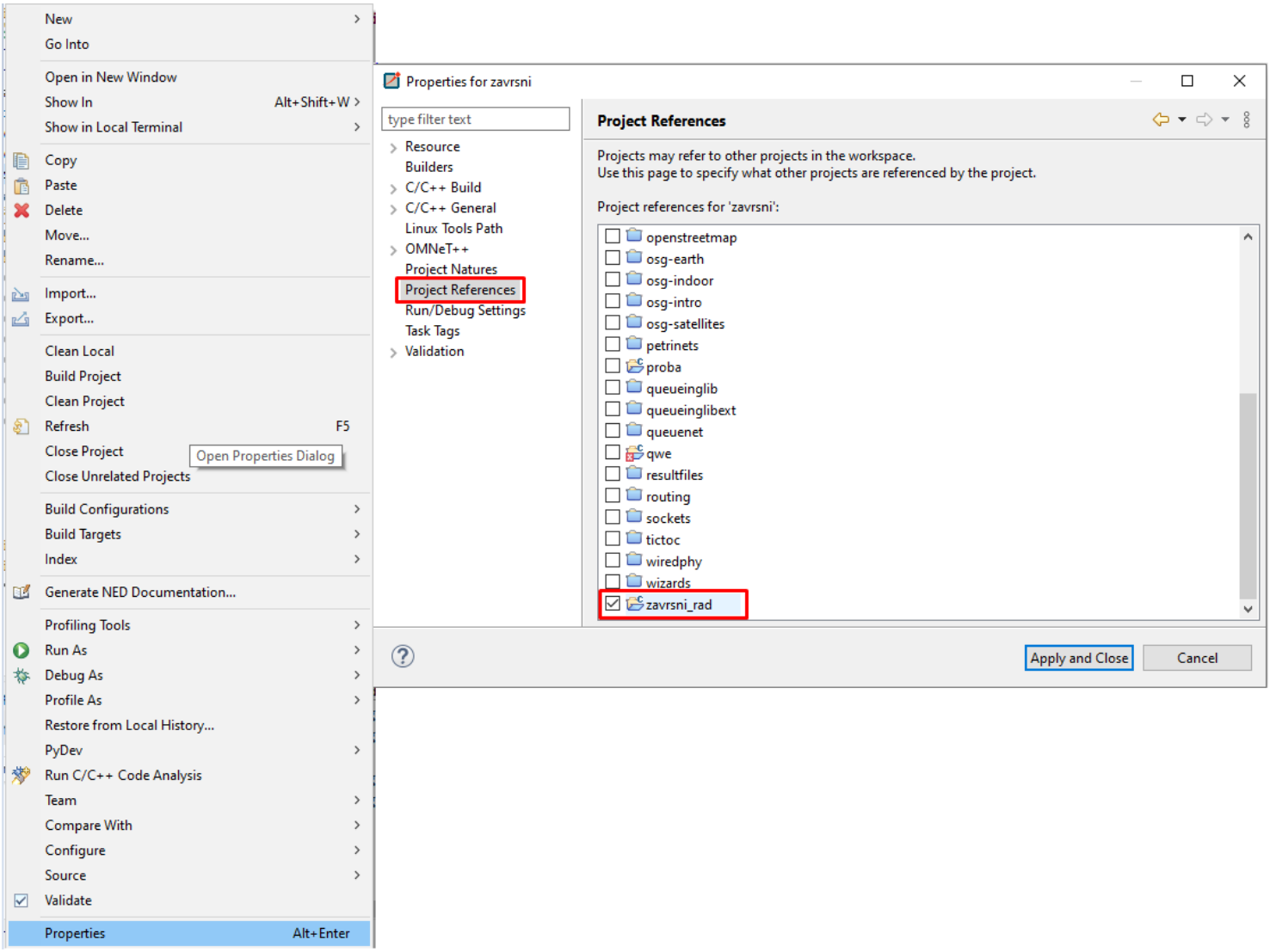
*Slika 22. Postavke projekta (Izvor: Vlastita izrada)Slika 23. Repozitorij*

Nakon što su provedeni prethodno istumačeni koraci za uvoz projekta, sljedeći korak je omogućivanje njegove porabe u željenom projektu.

Za to je potrebno, unutar Project Explorera, desnim pritiskom miša na projekt odabrati opciju „Properties". Ovim postupkom otvara se novi prozor s različitim konfiguracijskim izborima. Iz izbornika na lijevoj strani potrebno je odabrati „Project References". Ovdje se nalazi popis svih projekata raspoloživih unutar radnoga prostora OMNeT++-a.

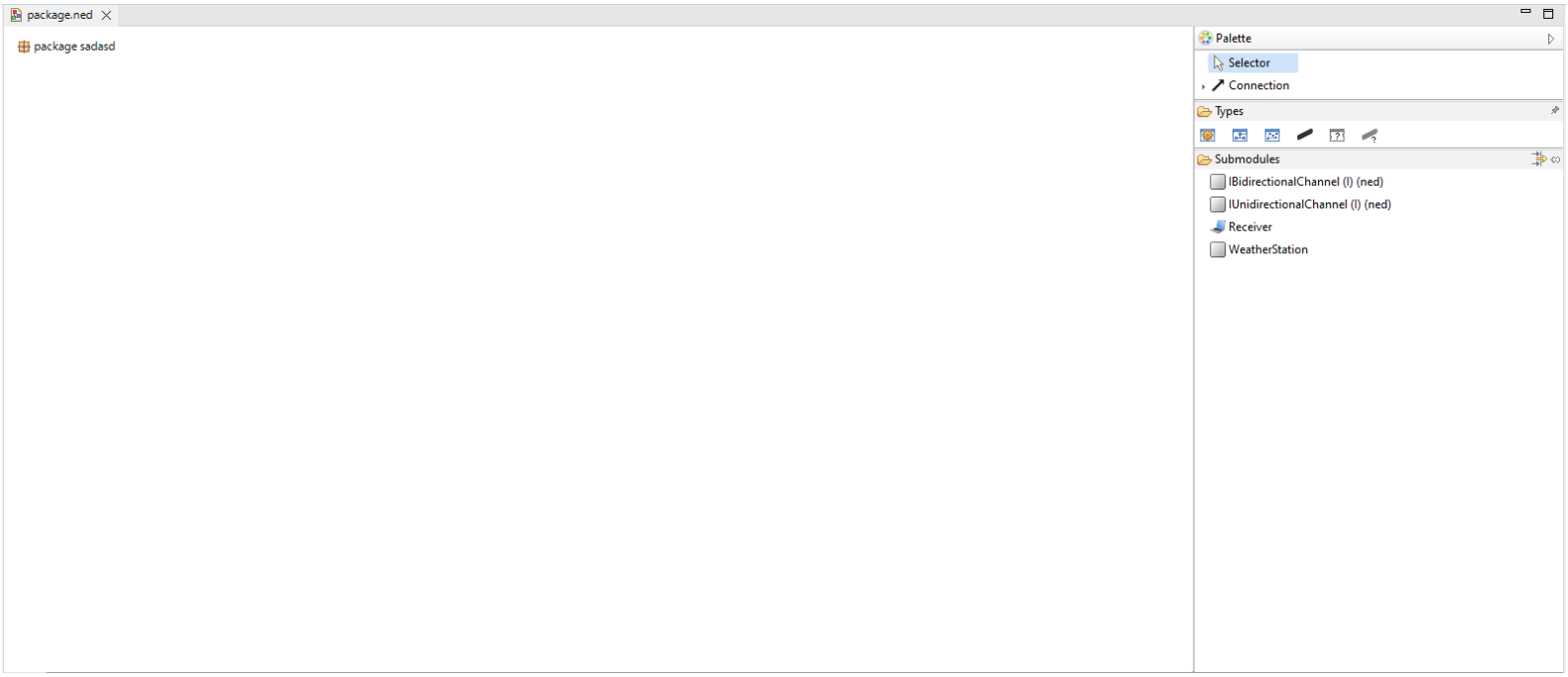
U ovom odjeljku „Project References" obilježava se kvačicom projekt koji ima modul meteorološke postaje, čim se omogućuje njegova poraba u trenutačno djelatnom projektu. Nakon obilježavanja željenoga projekta promjene se prihvaćaju, čim se prozor automatski zatvara.

Ovaj postupak omogućuje fleksibilnu porabu različitih modula i sastavnica unutar OMNeT++-projekata, olakšavajući tako razvoj složenih simulacijskih scenarija.




*Slika 24. Postavke projekta (Izvor: Vlastita izrada)*

Kad je projekt aktiviran unutar drugoga projekta, otvaranjem .ned-datoteke iz projekta na kojem se trenutno radi na desnoj strani prozora bit će prikazani svi podmoduli raspoloživi za porabu. U ovom konkretnom slučaju to uključuje podmodule „Receiver" i „WeatherStation". Ovaj prikaz omogućuje jednostavnu identifikaciju i pristup sastavnicama koje su na raspolaganju za integraciju unutar simulacijskoga modela, čim se olakšava proces konstrukcije i raščlambe složenih simulacijskih scenarija.




*Slika 25. Podmoduli u projektu (Izvor: Vlastita izrada)*

# 6. Zaključak

Kroz ovaj rad imao sam prigodu istražiti moćan mrežni simulator OMNeT++, koji nudi širok spektar primjena. Iako je znan po mrežnoj simulaciji, rabi se i za simulaciju raskrižja i električnih mreža.

Moj zadatak je bio stvoriti simulacijski model komada sklopovlja bez programiranja u programskom jeziku C++. Zahvaljujući jeziku modeliranja NED unutar OMNeT++-a i programskom jeziku Python, uspješno sam stvorio tri projekta. Ti modeli su jednostavno zapakirani i podijeljeni s javnošću, čim su postali sredstva za dalju porabu i prilagodbu.

Kroz ovaj rad prepoznao sam ne samo vrijednost oruđa poput OMNeT++-a, nego i moć jezika NED koji omogućuje stvaranje modela bez dubokoga programerskoga znanja. Porabom programskoga jezika Python uspio sam dokazati da je moguće stvoriti module bez porabe C++-a, iako je potrebno znanje o njem. Objavljivanjem svojih rezultata dao sam mogućnost korisnicima simulatora rabiti moje module i dalje ih razvijati po vlastitim potrebama.

# 7. Literatura

[1] A. Varga, OMNeT++ Discrete Event Simulation System, Version 2.3, June 15, 2003.

[2] H. A.Varga, AN OVERVIEW OF THE OMNeT++ SIMULATION ENVIRONMENT, Marseille, France: ICST, 2008.

[3] C. Thomas, Learning OMNeT++, Birmingham, UK.: Packt Publishing Ltd., 2013.

# 8. Popis slika