Primjena OMNeT++ alata za modeliranje računalom upravljanih okolina

Majdandžić, Ivana

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

PRIMJENA OMNET++ ALATA ZA MODELIRANJE RAČUNALOM UPRAVLJANIH OKOLINA

Diplomski rad

Ivana Majdandžić

Osijek, 2016.

SADRŽAJ

1. UVOD 1
2. RAČUNALOM UPRAVLJANI SUSTAVI 2
2.1. Raspodijeljeni proračuni i upravljanje mrežama 2
2.2. Izazovi i prilike u industriji 4
3. SIMULACIJSKI ALAT OMNeT++ 6
3.1. Struktura alata OMNeT++ 7
3.2. Struktura modela 9
3.3. Struktura modula 10
3.4. Programski jezik NED 12
3.5. OMNeT++ IDE sučelje 15
3.6. Mrežna simulacija 16
3.7. Korisničko sučelje Tkenv 17
4. SIMULACIJSKI MODEL BESPILOTNIH LETJELICA 20
4.1. Simulacijski model 20
4.2. Simulacijski moduli 22
4.3. Izvedba simulacijskog modela 26
4.3.1. Scenarij 1 – nadzor grada Osijeka 28
4.3.2. Scenarij 2 – potraga za ljubimcem 29
4.3.3. Scenarij 3 – potraga za izgubljenim djetetom 31
4.3.4. Scenarij 4 – prometna nesreća 33
4.3.5. Scenarij 5 – navijački izgredi na stadionu 35
4.4. Analiza scenarija 36
5. ZAKLJUČAK 37
LITERATURA 38
SAŽETAK 39
ABSTRACT 40
ŽIVOTOPIS 41
PRILOZI 42

1. UVOD

Pojam računalom upravljane okoline (engl. *Cyber-physical systems*, CPS) predstavlja novu generaciju sustava s integriranim računalnim i fizičkim sposobnostima kojima je omogućena interakcija s ljudima na više načina. Sposobnost interakcije, te širenje mogućnosti preko računalnih operacija, komunikacije i upravljanja, ključ je za razvitak buduće tehnologije.

Cilj ovog diplomskog rada je prikazati upotrebu simulacijskog alata OMNeT++ preko više simulacijskih modela bespilotnih letjelica, te njihovu stvarnu primjenu. Svaki simulacijski model je predstavljen vlastitim scenarijem, koji se događa na određenim koordinatama u gradu Osijeku. Time je u radu pokazano da se alatom OMNeT++ i priručnim alatima (Google Earth) moguće prikazati računalom upravljanu okolinu u simulaciji, te eventualnim dorađivanjem modela samu simulaciju iskoristiti u stvarnom svijetu (Scenarij 4 – prometna nesreća).

U drugom poglavlju objašnjeno je i opisano što su to računalom upravljane okoline, te kakva je njihova primjena u industriji i znanosti. U trećem poglavlju je objašnjen simulacijski alat OMNeT++, za što se koristi, te u koju svrhu i kako će biti korišten u ovome radu. Zadnje, četvrto poglavlje prikazuje krajnji cilj rada, a to je simulacija modela bespilotnih letjelica u alatu OMNeT++. Prikazat će se i opisati nekoliko scenarija njihove upotrebe, kako bi se približio način primjene simulacijskog alata OMNeT++, te simulacija leta bespilotnih letjelica na području grada Osijeka. Također, bit će prikazani rezultati i koraci simulacije u programskom alatu Google Earth.

2. RAČUNALOM UPRAVLJANI SUSTAVI

Napretkom znanosti došlo je do razvoja moćnih znanstvenih sustava, inženjerskih metoda i alata. Istovremeno, razvijeni su novi programski jezici, proračunske tehnike u stvarnom vremenu, metode vizualizacije, konstrukcija sastavnika (engl. *compiler*) u ugradljivim sustavskim arhitekturama i programskim alatima, te novi inovativni pristupi za osiguranje pouzdanosti, računalne sigurnosti i tolerancije na greške u računalnim sustavima kao i velik broj moćnih alata za modeliranje i provjeravanje. Istraživanje računalnih sustava kao cilj ima integraciju znanja i inženjerskih principa u proračunskim i inženjerskim disciplinama kao što su mrežna sučelja, upravljanje, programi, interakcija s ljudima, teorija učenja, kao i elektroničke, mehaničke, kemijske, medicinske i druge inženjerske discipline [3].

U industriji, mnogi su inženjerski sustavi nastali razdvajanjem konstrukcije upravljačkog sustava iz implementacijskih dijelova samih uređaja i/ili programskih alata. Nakon što je konstruiran upravljački sustav, te potvrđen iscrpnim simulacijama, metodama ugađanja pokušavaju se riješiti slabe točke u modeliranju i nasumične greške. Međutim, integracija raznih podsustava, uz zadržavanje samog sustava funkcionalnim i operativnim, je skupa i dugotrajna. Na primjer, u automobilskoj industriji, sustav upravljanja vozilom se oslanja na dijelove koje su napravili razni proizvođači koji imaju vlastite programe i uređaje. Najveći izazov za proizvođače originalne opreme koji daju dijelove je smanjiti troškove razvijanjem dijelova koji mogu biti integrirani u različita vozila. Povećana složenost dijelova i upotreba naprednijih tehnologija u senzorima, bežičnoj komunikaciji, te višejezgrenim procesorima, pruža velik izazov za izgradnju nove generacije sustava za upravljanje.

Dobavljač i integrator trebaju nove sustave koji omogućuju pouzdanu i isplativu integraciju nezavisnih dijelova sustava. U pravilu, to su:

- nacrt, analiza i provjera dijelova na razne načine, uključujući sam sustav i arhitekturu, koji mogu biti podložni drugim sustavima
- analiza i razumijevanje interakcije među upravljačkim sustavima, te raznim podsustavima
- osiguravanje sigurnosti, stabilnosti i učinka, istovremeno zadržavajući isplativost

2.1. Raspodijeljeni proračuni i upravljanje mrežama

Razvijanje sustava koji mogu upravljati okolinom, te komunicirati s vanjskim svijetom još je u začecima. Ograničenja u profesionalnom svijetu rezultirala su usko definiranom istraživanju ovisnom o području znanosti. Istraživanje je podijeljeno u izolirane poddiscipline kao što su senzori, komunikacije i mrežni sustavi, teorija upravljanja, matematika, programsko

inženjerstvo, te računalni sustavi. Primjerice, sustavi se kreiraju i analiziraju raznim alatima za modeliranje. Svaki prikaz sustava ističe određene karakteristike, dok druge zanemaruje, sve kako bi se analiza prikazala na prigodan način. Obično, određeni način formalizacije predstavlja ili računalni ili fizički proces, ali ne oba istovremeno.

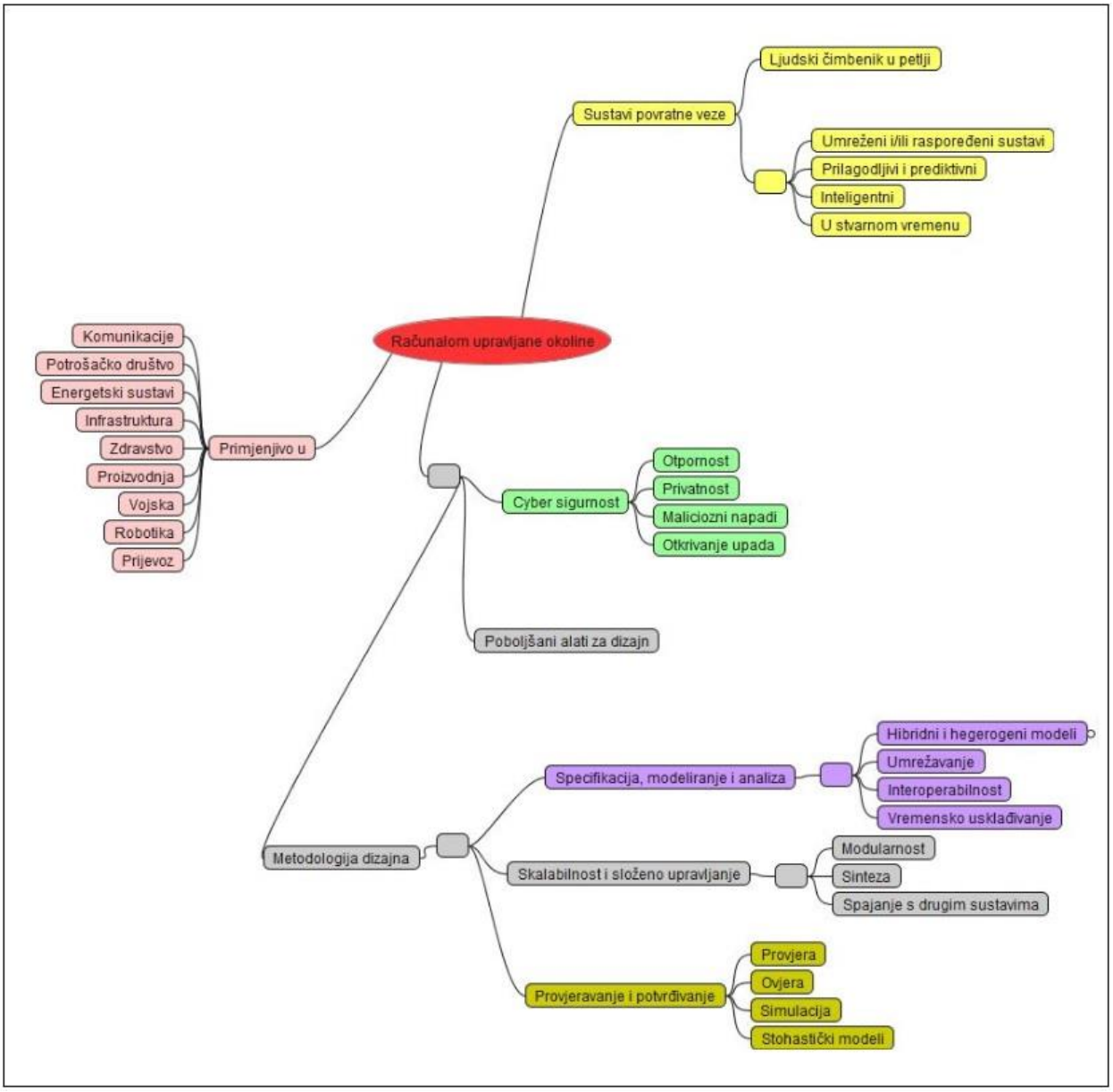
Inovativni pristupi apstrakciji i arhitekturi omogućavaju urednu implementaciju upravljanja, komunikacije i proračuna u ubrzani razvoj i konstrukciju CPS-a, odnosno računalno upravljanih okolina [3].

Neki od izazova u istraživanju računalom upravljanih okolina su:

- konstrukcija kvalitetnih protokola za rad u stvarnom vremenu u bežičnim mrežama
- smanjivanje razlika između kontinuiranih i diskretnih sustava
- veličina koju sustavi mogu poprimiti, te smanjivanje iste na najmanju moguću mjeru, uz zadržavanje ili poboljšanje korisnosti

Kako bi se pomicale granice postojeće tehnologije, potrebno je razvijati nove programske alate i operacijske sustave. Moraju biti vrlo pouzdani, rekonfigurabilni, te, gdje je to potrebno, i certificirani, od samih dijelova pa do cijelih sustava. Takvi složeni sustavi moraju imati veću pouzdanost i iskoristivost nego što imaju sustavi u današnjim računalnim infrastrukturama. Certificiranje obično uzima više od 50% resursa potrebnih za razvijanje novih, sigurnosnih sustava. Iz tog razloga jedini način za sigurno i uspješno certificiranje i razvoj sustava jest ulaganje resursa u smisleno planiranje. Međutim, takav način rada je gotovo nemoguće primijeniti u složenim nacrtima, pogotovo u sustavima koji ovise o interoperabilnosti. Tako da strategija testirati dok ima novca nije pouzdana, te su potrebne znanstvene i činjenične metode za pomoću u rasuđivanju pouzdanosti nekog sustava. Potrebni su novi modeli, algoritmi, metode i alati koji će pripojiti utvrđivanje i potvrđivanje u same programske alate i sustave na razini nacrta, radi samog upravljanja.

Razne računalne okoline, algoritmi, metode i alati su potrebni za osiguravanje visoke pouzdanosti i sigurnosti raznorodnih (engl. *heterogeneous*) surađujućih dijelova sustava koji komuniciraju u složenoj fizičkoj okolini na više lokacija i vremenskih perioda. Slika 2.1 prikazuje osnovnu strukturu računalom upravljane okoline, njene značajke, te mogućnost primjene [15].



Slika 2.1. Osnovna struktura računalom upravljane okoline

2.2. Izazovi i prilike u industriji

Napredak u razvoju računalom upravljanih okolina može biti ubrzan utvrđivanjem potreba, prepreka i prilika u nekoliko industrijskih grana, te radom na istraživanjima u kojima sudjeluju stručnjaci akademske i inženjerske struke iz raznih disciplina. Cilj je razvijanje novih sustava novim metodama za izradu visoko pouzdanih sustava koji su kompatibilni i integrabilni na svim razinama. Dosad je ulaganje u CPS tehnologije bilo poveće, ali usredotočeno na kratkoročne tehnologije koje donose brzu zaradu. Odnedavno, vlade i neke grane industrije ulažu u dugoročne, konkurentne i inovativne tehnologije. Na primjer, Europska Unija je inicirala veliku združenu tehnološku inicijativu koju financiraju članice EU i industrija poznata pod imenom

ARTEMIS (engl. *Advanced Research and Technology for Embedded Intelligence Systems*), prema [3]. Slične inicijative postoje i u SAD-u, Japanu, Kini, Južnoj Koreji i Njemačkoj. Nažalost, velike prepreke koje koče razvoj računalom upravljanih sustava nalaze se u velikom broju industrijskih grana.

Kao što je već navedeno, računalom upravljane okoline (CPS) su fizički i inženjerski sustavi koji su nadzirani, koordinirani, upravljani i integrirani od strane računalne i komunikacijske jezgre. Ta poveznica između računalnog i fizičkog svijeta vidljiva je i na manjim i na velikih sustavima.

CPS je spoj integriranih sustava, sustava u stvarnom vremenu, senzorskih sustava i upravljanja. S obzirom da su sve dostupnija kvalitetnija računala i senzori, bežične mreže su gotovo posvuda, kao i alternativni izvori energije, očito je da će u budućnosti računalom upravljane okoline biti sve traženije i potrebnije.

Računalom upravljane okoline su sustavi od kojih se očekuje velika uloga u konstrukciji i razvoju budući inženjerskih sustava s novim sposobnostima koji uvelike nadmašuju današnje razine autonomije, funkcionalnosti, iskoristivosti, pouzdanosti i računalne sigurnosti. Napreci mogu biti ubrzani bliskom suradnjom akademskih disciplina u komunikaciji, upravljanju, računarstvu, i ostalim inženjerskim i računalnim disciplinama.

Jedna od takvih okolina jest upravljanje bespilotnim letjelicama, čiji će rad i upravljanje, biti prikazano u idućim poglavljima kroz simulacijski alat OMNeT++.

3. SIMULACIJSKI ALAT OMNeT++

OMNeT++ (engl. *Objective Modular Network Testbed in C++*) kao diskretno simulacijsko okruženje je dostupan javnosti još od 1997. [6]. Napravljen je s namjerom da se može simulirati rad komunikacijskih mreža, višeprocesorskih i drugih sustava s primjenom na više različitih područja. Umjesto specijaliziranog simulatora, OMNeT++ je napravljen da bude što općenitiji. Otada se pokazalo da je ideja bila uspješna, te se danas OMNeT++ koristi u raznim područjima, od mreža za upravljanje redovima i redoslijedima do bežičnih i ad-hoc mrežnih simulacija, od poslovnih procesa do P2P (engl. *Peer-to-peer*) mreže.

OMNeT++ je zasnovan na programskom jeziku C++, koji služi za simulaciju i modeliranje komunikacijskih mreža, višeprocesorskih i drugih distribucijskih ili paralelnih sustava. Besplatan je i otvorenog koda, te može biti korišten pod APL (engl. *Academic Public License*) udrugom, koja omogućava korištenje besplatnih programskih alata u neprofitne svrhe. Cilj razvoja samog alata OMNeT++ jest izrada konačnog proizvoda koji je snažan i besplatan simulacijski alat. Simulacijski alat koji može biti korišten u akademskim, edukacijskim i istraživačkim institucijama, za simulaciju računalnih mreža i distribucijskih illi paralelnih sustava. OMNeT++ je dostupan na svim uobičajenim operacijskim sustavima uključujući Linux, Mac OS/X i Windows, uz korištenje GCC alata (engl. *GNU Compiler Collection*) ili prevoditelja Microsoft Visual C++.

Svrha alata OMNeT++ jest dati simulacijski pristup korisniku sa strane same razvojne okoline. To znači, kako umjesto izravnog pružanja simulacijskih dijelova za računalne mreže, on pruža osnovne alate za pisanje takvih simulacija. Specifična područja primjene su podržana od strane raznih simulacijskih modela i razvojnih cjelina kao što su *Mobility Framework* ili *INET Framework* (engl. *Internet networking framework*). Ovi su modeli razvijeni posve neovisno od alata OMNeT++, te imaju vlastite cikluse izdavanja.

Još od prvih izdanja, simulacijski modeli su bili razvijani od strane raznih pojedinaca i istraživačkih skupina u nekoliko područja: bežične i *ad-hoc* mreže, senzorske mreže, IP i IPv6 mreže, MPLS (engl. *Multiprotocol Label Switching*), bežični kanali, P2P mreže, SAN (engl. *Storage Area Networks*) mreže, optičke mreže, mreže za upravljanje redovima, datotečni sustavi, međuspojevi velikih brzina (*InfiniBand*), i drugi [1]. Uz istraživačke skupine i neprofitne istraživačke institucije, OMNeT++ za komercijalne projekte i vlastita istraživanja koriste i tvrtke kao što su IBM, Intel i Cisco.

Razumijevanje potrebe za simulacijom je veliki čimbenik u odlučivanju. Prilikom izrade mreže, potrebno je uzeti u obzir nekoliko glavnih točaka koje bitno određuju smjer, i način izvedbe neke mreže (prema [1]):

- trošak potrebnog sklopovlja, poslužitelja, preklopnika koji služe da mreža što bolje i kvalitetnije funkcionira – kvalitetniji uređaji su u pravilu i skuplji
- veća količina vremena je potrebna kako bi se ispravno postavila velika specijalizirana mreža, koje će se koristiti u poslovne ili akademske svrhe
- preinake na postojećoj mreži traži dodatno planiranje, te ako dođe do greške u planiranju, a samim time i u izvedbi, postoji mogućnost da mreža neće ispravno raditi

Uzimanjem u obzir pozitivne i negativne čimbenike kod izrade stvarne mreže, često dolazi do potrebe da se prije upuštanja u takav projekt, prvo napravi simulacijski model iste mreže u alatu OMNeT++. Neke prednosti (iako ne bez ponekih privremenih nedostataka) su prema [1]:

- što se tiče uređaja i njihovog troška, on se svodi na kupnju računala koje može pokrenuti OMNeT++ (koji je besplatan)
- potrebno je vremena za učenje izrade simulacija. Međutim, jednom kada se shvate osnove, izrada svake sljedeće simulacije je sve lakša.
- nema rizika kod preinaka na simuliranoj mreži, a upravo to je i glavni razlog korištenja simuliranih mreža; rezultati simulacije se mogu analizirati te na temelju njih je moguće odrediti kakav će biti utjecaj na stvarnu mrežu
- ako postoji greška u kodu simulacije, postoji mogućnost da se simulacija neće izvoditi ispravno

Simulacije omogućavaju da se moguće poteškoće predvide, i na taj način moguće je izbjeći poteškoće u stvarnom sustavu(otpornost, ograničenja, oštećenja).

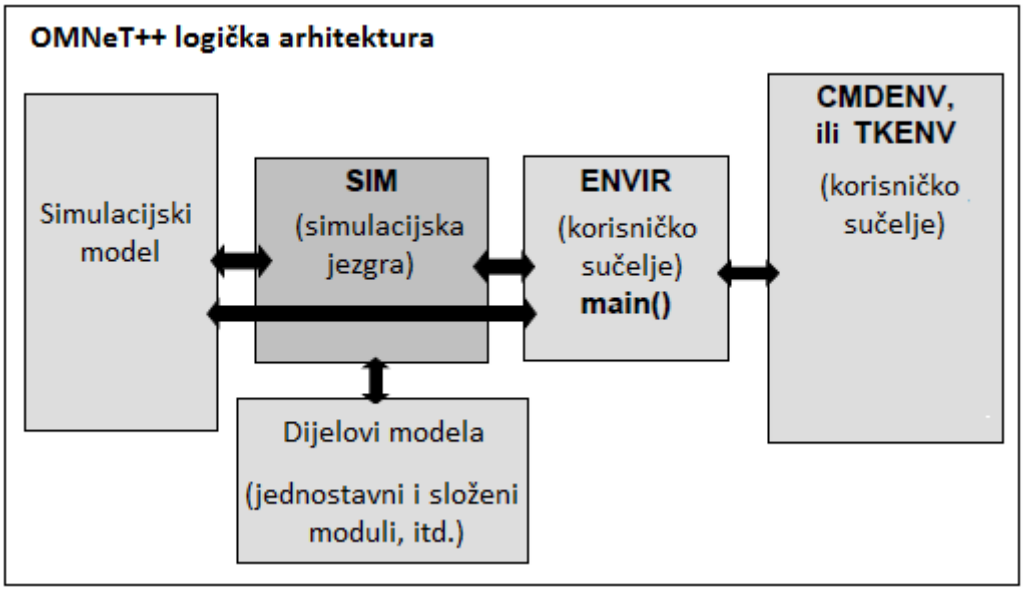
3.1. Struktura alata OMNeT++

OMNeT++ je od početka zamišljen kao alat za podršku mrežnih simulacija u velikim razmjerima. Za cilj je imao nekoliko glavnih uvjeta [1]:

- kako bi se omogućile simulacije u velikim razmjerima, simulacijski modeli moraju biti posloženi hijerarhijski, te izrađeni od materijala koji mogu biti iskorišteni što je više moguće puta
- simulacijski alati trebaju olakšati vizualizaciju i otklanjanje grešaka u simulacijskim modelima kako bi se smanjilo vrijeme traženja i ispravljanje grešaka, koje obično uzima velik postotak vremena u simulacijskim ili edukacijskim projektima

- simulacijski alati moraju biti modularni i prilagodljivi, te trebaju moći spajati simulacije u veće aplikacije kao što su alati za planiranje mreža, što donosi dodatne zahtjeve u pogledu upravljanja memorijom i sl.
- podatkovno sučelje treba biti otvorenog tipa kako bi bilo moguće izazvati i obraditi ulazne i izlazne podatke pomoću uobičajenih i opće prihvaćenih programskih alata
- potrebno je omogućiti integriranu razvojnu okolinu IDE (engl. *Integrated Development Environment*) koja uvelike olakšava razvijanje modela i analiziranje rezultata

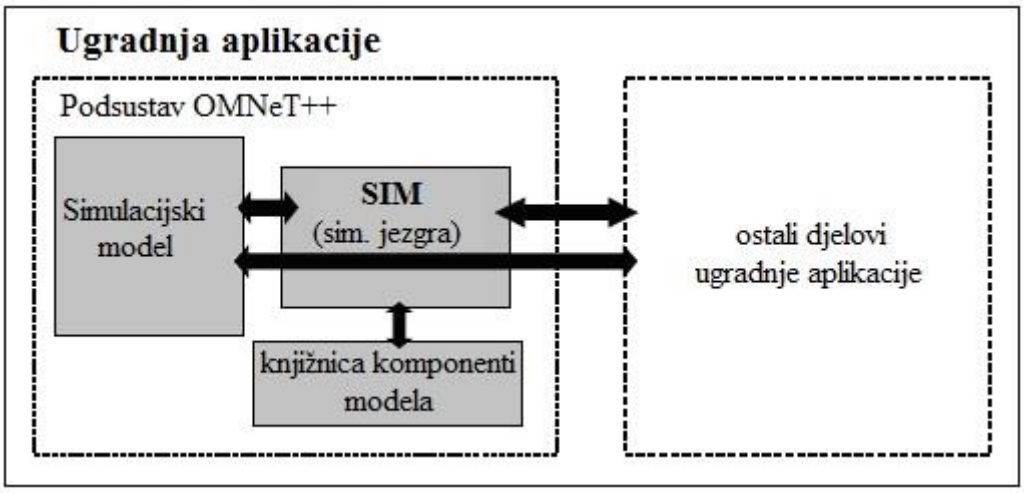
Simulacijski program OMNeT++ ima modularnu strukturu. Prema [1] logička arhitektura takve modularne strukture prikazana je na slici 3.1:



Slika 3.1. Logička arhitektura simulacijskog alata OMNeT++

Baza dijelova modela sastoji se od koda jednostavnih i složenih modula. Moduli su instancirani, te je konkretan simulacijski model izgrađen od strane same simulacijske jezgre i baze klasa na početku svakog izvršavanje simulacije. Simulacija se izvršava u okolini dobivenoj iz baze sučelja (*Envir*, *Cmdenv* i *Tkenv*). Ta okolina određuje odakle dolaze ulazni podaci, gdje se šalju rezultati simulacije, te što se događa s otklanjanjem grešaka koje se uočavaju iz izlaznih podataka. Isto tako, okolina upravlja izvršavanjem same simulacije, i određuje na koji način je prikazan simulacijski model.

Na slici 3.2 prikazano je kako je zamjenom korisničkog sučelja moguće prilagoditi cijelu okolinu u kojoj se pokreće simulacija, prema [1]. Čak je moguće i OMNeT++ simulaciju ugraditi u veću aplikaciju. To je moguće zato što postoji općenito sučelje između *Sim* i baze korisničkih sučelja, kao i činjenica da su sve *Sim*, *Envir*, *Cmdenv* i *Tkenv* baze fizički odvojene. Isto tako, moguće je usred izgradnje simulacije u ugrađenu aplikaciju iz postojećih modula slagati dodatne modele.



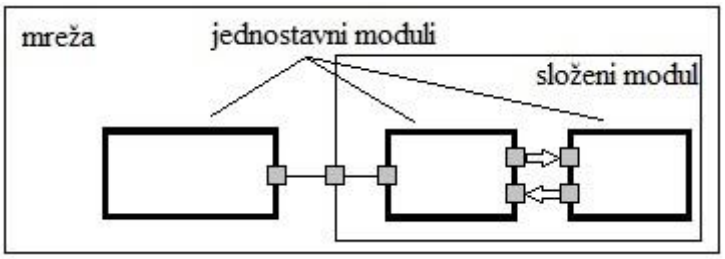
Slika 3.2. Ugradnja OMNeT++ simulacije u vanjsku aplikaciju

3.2. Struktura modela

Model OMNeT++ sastoji se od modula koji komuniciraju primopredajom poruka. Aktivni moduli nazivaju se jednostavnim modulima, a napisani su u programskom jeziku C++, koristeći klase koje se nalaze u simulacijskom alatu. Jednostavni moduli mogu biti grupirani u složene module, a količina hijerarhijskih razina nema ograničenja. Poruke mogu biti poslane ili preko spoja koji se protežu između pojedinih modula ili izravno do odredišnog modula. Koncept jednostavnih i složenih modula sličan je DEVS (engl. *Discrete Event System Specification*) nedjeljivim i spojenim modulima.

Spojeni i složeni moduli su primjeri tipova modula. Opisujući model, korisnik definira tipove modula. Primjeri tih tipova modula služe kao dijelovi za složene tipove modula [6]. U konačnici, korisnik izrađuje sustav modula kao mrežni modul koji je poseban složeni tip modula, ali bez izlaza u stvarni svijet. Kada se tip modula koristi kao blok za izradu, nema razlike u tome radi li se o jednostavnom ili složenom modulu. To omogućuje korisniku da transparentno podijeli modul u nekoliko jednostavnih modula unutar složenog modula, ili obratno, da ugradi funkcionalnost složenog modula u jedan jednostavan modul, bez utjecaja na korisnike istog tipa modula. Izvodljivost modula je dokazana u razvojnim cjelinama kao što su INET Framework i Mobility Framework, i njihovim aplikacijama nastalima kao njihov produžetak.

Prema [1], na slici 3.3 prikazana je struktura modula u OMNeT++ simulacijskom alatu gdje blokovi podebljanog okvira predstavljaju jednostavne module, dok preostali blokovi predstavljaju složene module. Strjelice koje spajaju blokove predstavljaju spojeve i prolaze.



Slika 3.3. OMNeT++ struktura modula

Moduli komuniciraju porukama koje, uz uobičajene atribute kao što je vremenska oznaka, mogu sadržavati proizvoljne podatke. Jednostavni moduli obično šalju poruke preko prolaza, ali moguće je slati ih i izravno prema odredišnim modulima. Prolazi su ulazna i izlazna sučelja modula. Poruke se šalju preko izlaznih prolaza (izlazi), te pristižu preko ulaznih prolaza (ulazi). Ulazi i izlazi mogu biti povezani spojem. Spojevi su napravljeni na istoj razini modulske hijerarhije. Unutar složenog modula, odgovarajući prolazi dva podmodula, ili prolaz jednog podmodula ili prolaz složenog modula mogu biti povezani. Spojevi koji se protežu duž hijerarhijskih razina nisu dopušteni, jer bi se time spriječila mogućnost ponovne upotrebe modela. Zbog hijerarhijske strukture modela, poruke obično putuju preko lanca spojeva, kako bi imale i početak i kraj u jednostavnim modulima. Svojstva kao što je kašnjenje u propagaciji, brzina prijenosa podataka, te učestalost pogrešnih bitova, mogu biti pridodane spojevima. Spojevi se mogu definirati i kao kanali, te mogu biti korišteni na više mjesta.

Moduli mogu imati parametre. Parametri se većinom koriste za prosljeđivanje konfiguracijskih podataka jednostavnim modulima, te za pomoć u definiranju topologije modela. Parametri mogu imati više vrsta vrijednosti: *string*, *numeric*, *boolean*. Budući da su predstavljeni kao objekti u programskom alatu, uz to što sadržavaju konstante, parametri se mogu ponašati kao izvori nasumičnih brojeva što se može implementirati u samom modelu. Moduli mogu interaktivno tražiti korisnika neku vrijednost, te mogu sadržavati izraze koji se pozivaju na druge parametre.

3.3. Struktura modula

Već je ranije spomenuto kako složeni moduli mogu prosljeđivati parametre ili izraze svojim podmodulima, koji su počesto zapravo jednostavni moduli. Budući da se sama struktura modula, kako složenih, tako i jednostavnih, u konačnici sastoji od jednostavnih modula, potrebno je objasniti što su to jednostavni moduli.

Jednostavni moduli su aktivni elementi u modulu. Oni su nedjeljivi elementi u hijerarhiji modula. Jednostavni moduli napisani su u programskom jeziku C++, koristeći OMNeT++ kao bazu klasa. OMNeT++ omogućuje IDE C++, tako da je moguće napisati, pokrenuti i otklanjati

neispravnosti u kodu bez napuštanja OMNeT++ IDE-a. Simulacijska jezgra ne pravi razliku između poruka i događaja, događaji su također prikazani kao poruke, prema [6].

Jednostavni moduli napisani su pomoću procesne interakcije. Korisnik implementira funkcionalnost jednostavnog modula podklasiranjem cSimpleModule klase. Funkcionalnost se dodaje preko dva, alternativna, programska modela:

- model rutiniranog kodiranja
- model funkcije obrade događaja

Programski kod modula pokreće vlastiti niz, kojim upravlja simulacijska jezgra svaki puta kada moduli primi neki događaj, odnosno poruku. Funkcija koja sadrži kod za rutinu obično sadrži i beskonačnu petlju s poslanim i primljenim pozivima. Koristeći funkciju obrade događaja, simulacijska jezgra jednostavno poziva danu funkciju objekta modula s porukom kao argumentom. Funkcija se vraća odmah nakon što je poruka obrađena. Glavna razlika između ova dva programska modela jest što kod potonjeg svaki jednostavni modul treba vlastiti CPU stog, što pak znači da je za njega potrebno više memorije kako bi ga simulacijski program pokretao, prema [6]. Ovo dolazi do izražaja kada model ima velik broj modula (više desetaka tisuća).

Moguće je napisati kod koji se izvršava pri inicijalizaciji ili kod koji se izvršava kada modul prestaje s radom i potom sprema skalarne rezultate u datoteku. OMNeT++ također podržava inicijalizaciju na više razina, gdje je inicijalizaciju modula potrebno napraviti u više navrata. Podršku za ovu vrstu inicijalizacije često se ne može naći u simulacijskim paketima, te se stoga obično imitira razašiljanjem događaja planiranima u nultom vremenu simulacije, što nije elegantno rješenje.

Slanje i primanje poruka su najčešći zadaci jednostavnih modula. Poruke mogu biti poslane ili preko izlaza ili izravno preko nekog drugog modula. Moduli primaju poruke ili preko jedne od nekoliko vrsta dolaznih poziva (model rutina), ili su poruke dostavljene modulu zapravo prizvane od strane same simulacijske jezgre (model događaja). Poruke mogu biti definirane određivanjem njihovog sadržaja u MSG datoteku. OMNeT++ na sebe preuzima izradu potrebnih C++ klasa. MSG datoteke omogućavaju OMNeT++ jezgri generiranje reflektivnog koda koji daje uvid u poruke i istražuje njihov sadržaj za vrijeme trajanja izvođenja.

Moguće je dinamički mijenjati topologiju mreže. Mogu se stvarati i brisati moduli, te preuređivati spojeve za vrijeme izvođenja simulacije. Isto vrijedi i za slaganje složenih modula s unutarnjom parametarskom topologijom.

OMNeT++ pruža bogatu objektnu bazu za implementaciju jednostavnih modula. Postoji nekoliko razlikovnih faktora između ove baze i ostalih općenitih ili simulacijskih baza.

OMNeT++ baza klasa pruža funkcionalnost koja omogućava implementaciju otklanjanja grešaka i praćenja na visokoj razini, kao i automatiziranu animaciju. Gubici u memoriji i ostale poteškoće kod raspodjele memorije uobičajeni su u C++ programima. OMNeT++ ublažava ove poteškoće praćenjem vlasništva objekta i otkrivanjem grešaka nastalima zloupotrebom dijeljenih objekata. Zahtjevi za lakoćom korištenja, modularnošću, sučeljima otvorenog koda, te podrškom za ugrađivanje ovise ponajviše o konstrukciji same baze klasa. Dosljedno korištenje objektno-orijentiranih tehnika čini samu simulaciju kompaktnom i optimiziranom. Tako je lakše razumjeti interne strukture, što pogotovo dolazi do izražaja kada se ovaj alat koristi u edukacijske svrhe i pri traženju grešaka u kodu.

U zadnje vrijeme je postalo uobičajeno raditi simulacije velikih mreža pomoću alata OMNeT++, u kojima se može nalaziti i nekoliko desetaka tisuća mrežnih čvorova. Kako bi se ispunili takvi zahtjevi, u simulacijsku jezgru je implementirana agresivna optimizacija memorije, zasnivana na dijeljenim objektima i semantici ispisivanja kopija.

3.4. Programski jezik NED

Korisnik definira strukturu modela (module i njihovu međusobnu povezanost) preko OMNeT++ topologijskog opisnog jezika, NED (engl. *Network description*). Tipični sastavni dijelovi NED opisa su deklaracije jednostavnih modula, definicije složenih modula i definicije same mreže. Deklaracije jednostavnih modula opisuju sučelje modula: prolaze i parametre. Definicije složenih modula sastoje se od deklaracije vanjskog sučelja modula, definicije podmodula i njihovih međuspojeva. Mrežne definicije su složeni moduli okarakterizirani kao samostalni simulacijski modeli.

Jezik NED je konstruiran s planom za buduće implementacije i moguće razvoje alata OMNeT++. Međutim, zbog nedavnog rasta u količini i složenosti OMNeT++ simulacijskih modela i razvojnih cjelina potreban je dodatan razvoj NED jezika, prema [1]. U neka manja poboljšanja, uvedeno je nekoliko većih glavnih svojstava:

- Nasljednost. Moduli i kanali sada mogu imati podklase. Izvedenim modulima i kanalima mogu se dodavati novi parametri, prolazi i (vrijedi za složene module) novi podmoduli i spojevi. Postojećim parametrima se može postaviti točno određena vrijednost, te se može definirati veličina prolaznog vektora.
- Sučelja. Sučelja modula i kanala mogu biti iskorištena kao oznake na mjestu gdje bi inače bio iskorišten neki tip modula ili kanala, gdje je konkretan tip modula ili kanala određen u konfiguraciji mreže nekim parametrom. Konkretni moduli moraju implementirati sučelje koje može biti zamijenjeno.

- Paketi. Kako bi se izbjegla redundancija i dupliciranje imena među različitim modulima, te kako bi se pojednostavilo određivanje koje su NED datoteke potrebne simulacijskom modelu, u NED jezik je uvedena struktura paketa istovjetna onoj kakvu nalazimo u *Java* programskom jeziku.
- Unutarnji tipovi. Tipovi kanala i modula korišteni lokalno od strane složenog modula mogu biti definirani unutar samog složenog modula, kako bi imenovanje modula i klasa bilo pojednostavljeno i urednije.
- Označavanje metapodataka. Module, kanale, parametre, prolaze i podmodule je moguće označiti dodavanjem svojstava. Simulacijska jezgra sama ne koristi izravno metapodatke, već su oni nosioci dodatnih informacija za razne alate, vremensko trajanje izvođenja nekog dijela simulacije, ili čak informacija o modulima u modelu.

NED datoteke mogu biti pretvorene u *XML* i natrag bez gubitka podataka, uključujući komentare.

NED datoteku se može objasniti po Fifo komunikacijskom modelu (engl. *First in first out*), što je prema [5] prikazano na slici 3.4:

```
network FifoNet
{
  submodules:
    gen: Source {
      parameters:
        @display("p=89,100");
    }
    fifo: Fifo {
      parameters:
        @display("p=209,100");
    }
    sink: Sink {
      parameters:
        @display("p=329,100");
    }
  connections:
    gen.out --> fifo.in;
    fifo.out --> sink.in;
}
```

Slika 3.4. Programski kod Fifo simulacijske mreže

Vidljivo je da se simulacija sastoji od tri međusobno povezana modula pri čemu je svaki modul objekt (gen, fifo, sink) postojeće klase (Source, Fifo, Sink) koji se instancira u trenutku pokretanja simulacije. Svaki složeni modul sastoji se od podmodula, definiranih spojeva, ulazno-izlaznih sučelja i parametara koji se inicijaliziraju pokretanjem simulacije. Svaki jednostavni modul definiran je parametrima i sučeljima za međusobno povezivanje. Primjer definicije jednostavnog modula Fifo nalazi se na slici 3.5, prema [5]:

```
simple Fifo
{
  parameters:
    volatile double serviceTime @unit(s);
    @display("i=block/queue;q=queue");
    @signal[qlen](type="long");
    @signal[busy](type="bool");
    @signal[queueingTime](type="simtime_t");
    @statistic[qlen](title="queue length";record=vector,timeavg,max;
            interpolationmode=sample-hold);
    @statistic[busy](title="server busy state";record=vector?,timeavg;
            interpolationmode=sample-hold);
    @statistic[queueingTime](title="queueing time at
dequeue";unit=s;record=vector,mean,max;interpolationmode=none);
  gates:
    input in;
    output out;
}
```

Slika 3.5. Fifo NED modul

Jednostavni modul Fifo definiran je parametrom koji određuje vrijeme obrade poruke u redu čekanja te ulaznim, odnosno izlaznim sučeljima za prihvat poruke od generatora poruka i slanje poruke prema modulu sink klase Sink. Razmjena poruka vrši se kroz ulazno-izlazna sučelja. Svaka poruka je objekt instanciran od klase cMessage koja je dio simulacijske jezgre ili od klase izvedene nasljeđivanjem osnovne klase cMessage. Svaki jednostavni modul je objekt instanciran iz osnovne klase cModule ili njene izvedene klase.

Spojevi između modula su objekti izvedeni iz klase Channel s osnovnim karakteristikama prijenosnih medija poput kašnjenja, kapaciteta, udaljenosti. Po kreiranju NED datoteke potrebno je definirati parametre koji se učitavaju prilikom inicijalizacije simulacije. Parametri se mogu definirati izravno kroz NED datoteku, ali se najčešće definiraju u **.ini* datoteci.

Primjer definicije *.*ini* datoteke, prikazan je na slici 3.6:

```
[General]
network = FifoNet
record-eventlog = true
sim-time-limit = 150s
cpu-time-limit = 400s
#debug-on-errors = true
#record-eventlog = true

[Config Fifo1]
description = "low job arrival rate"
**.gen.sendIaTime = exponential(0.3s)
**.fifo.serviceTime = 0.02s

[Config Fifo2]
description = "high job arrival rate"
**.gen.sendIaTime = exponential(0.03s)
**.fifo.serviceTime = 0.02s
```

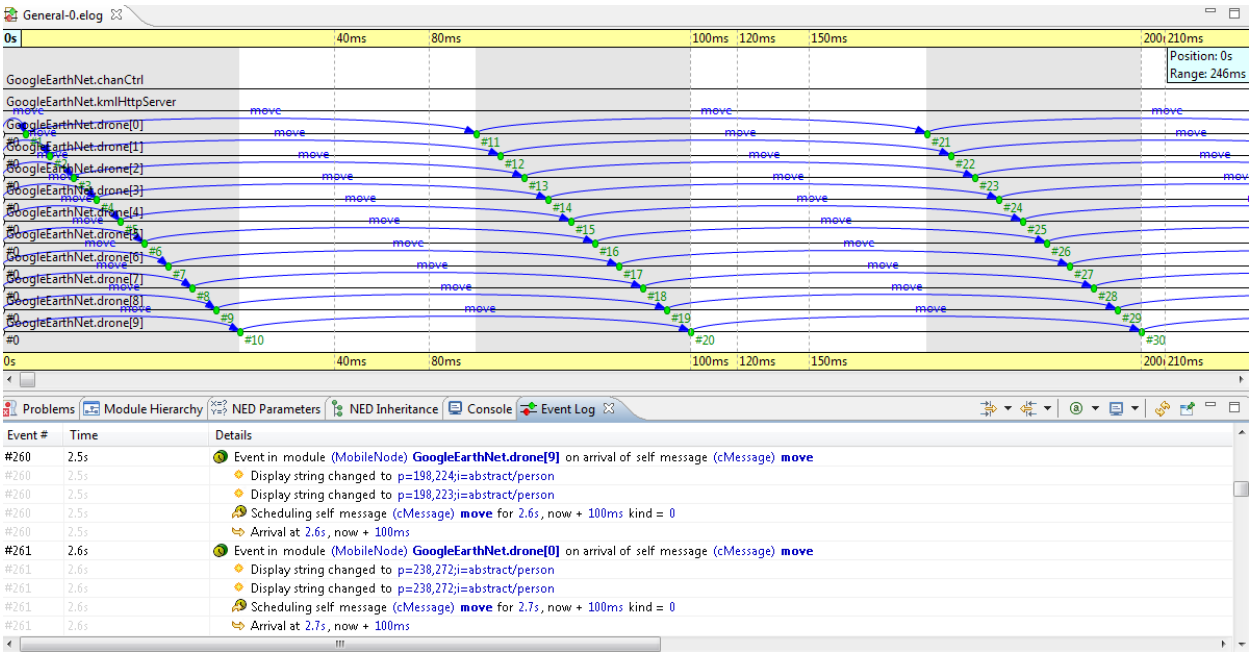
Slika 3.6. Fifo *.*ini* datoteka

3.5. OMNeT++ IDE sučelje

OMNeT++ uključuje *IDE*, koje pak sadrži grafički urednik koji koristi NED kao urođeni format datoteka. Štoviše, grafički urednik može funkcionirati na proizvoljno napisanom NED kodu. Grafički urednik je potpuno funkcionalan dvosmjerni alat, što znači da korisnik može uređivati topologiju mreže ili grafički ili u NED pregledniku, te prebacivati se između dva preglednika u bilo kojem trenutku. To je moguće zbog dobro isplaniranog i konstruiranog NED jezika.

Ponašanje velikih i složenih modela obično je teško razumjeti zbog složenih interakcija među različitim modulima. OMNeT++ umanjuje složenost tako što daje prioritet unaprijed definiranim spojem između određenih modula. Grafički prikaz izvršavanja omogućava korisniku praćenje interakcije među modulima od određene razine: moguće je animirati, usporiti simulaciju, ili je čak pratiti korak po korak, no ponekad je i dalje teško pratiti točan redoslijed ili vrijeme izvršavanja događaja. Zato zbog praktičnih razloga, simulacijsko vrijeme nije proporcionalno stvarnom vremenu. To znači da se, kod npr. praćenja simulacije iz korak u korak, događaji koji imaju istu vremensku oznaku ne animiraju istovremeno, već jedan za drugim.

OMNeT++ pomaže korisniku vizualizirati interakciju tako što bilježi interakcije među modulima u datoteku. Ta datoteka može biti obrađena nakon izvršenja simulacije (ili za vrijeme njenog izvođenja), te može biti iskorištena za crtanje interakcijskih dijagrama. OMNeT++ IDE ima slijedni dijagramski alat koji pruža uvid u događaje. Moguće je fokusirati se na sve ili na samo određene module i prikazati interakciju među njima, prema [1]. Na slici 3.7 prikazan je slijedni dijagram kretanja bespilotnih letjelica koji je dobiven izvršavanjem simulacijskog modela u poglavlju četiri.



Slika 3.7. Slijedni dijagram OMNeT++ IDE

3.6. Mrežna simulacija

U praksi je uvijek dobro kada se različiti dijelovi simulacije što je više moguće razdvajaju. Ponašanje modela definirano je u C++ kodu, dok je topologija modela (i parametri koji definiraju tu topologiju) definirana NED jezikom. Ovaj pristup omogućava korisniku da smjesti različite dijelove modela na različita mjesta, čime se u konačnici dobije uredniji model i olakšan rad. Obično se u općenitom simulacijskom scenariju želi znati kako će se simulacija ponašati s različitim ulaznim parametrima. Te varijable ne pripadaju ni ponašanju (kod) niti topologiji (NED datoteke) jer se mogu mijenjati iz simulacije u simulaciju. Kako bi se spremile te vrijednosti, koriste se *INI* datoteke, koje pružaju jednostavan način za određivanje promjene parametara, te omogućavaju simuliranje svake potrebne kombinacije parametara. Dobiveni rezultati simulacije mogu lako biti dostupni i obrađeni postojećim alatom za analizu.

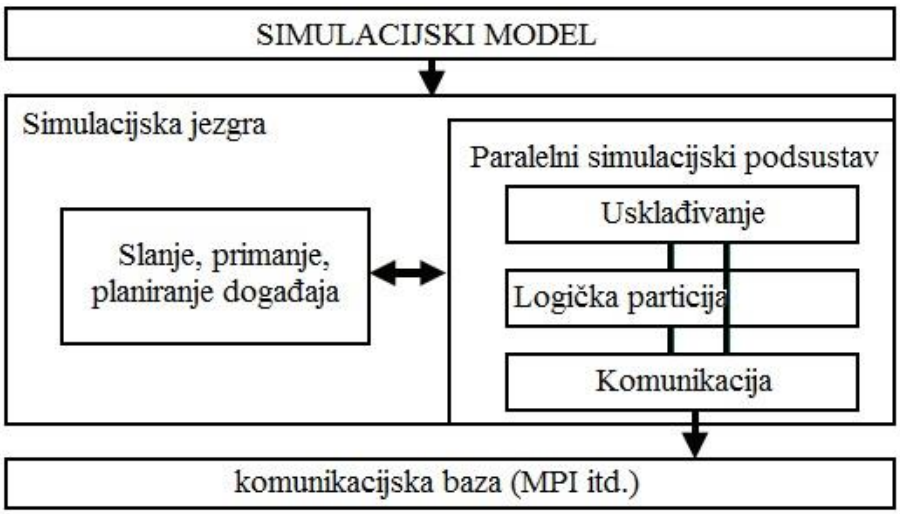
Klase obično sadrže klasne redove i spremnike, gdje redovi mogu funkcionirati kao prioritetni redovi. Same poruke su objekti koji mogu sadržavati proizvoljne podatke i druge objekte (njihovim sakupljanjem ili nasljeđivanjem), te mogu biti ugrađene u druge poruke.

OMNeT++ podržava usmjeravanje mrežnog prometa. Time je omogućeno istraživanje stvarne mrežne topologije, prikazivanje iste u obliku grafova, te njihovim upravljanjem ili primjenom algoritama za pronalaženje najkraćeg puta.

Postoji nekoliko statističkih klasa, od onih jednostavnijih koje rade standardne devijacije prikupljenih uzoraka, do onih složenijih koje služe za distribucijske procjene. Složenije statističke klase uključuju tri vrlo konfigurabilne histogramske klase, te implementacije P^2 i *k*-

split algoritama. Također je moguće ispisivati dobivene rezultate u vremenskim periodima u izlaznu datoteku za vrijeme trajanja same simulacije, dok ujedno postoje i alati za analizu rezultata nakon njenog završetka.

Moguće je izvršavati paralelne simulacije. Vrlo velike simulacije mogu imati velike koristi od paralelno raspodijeljenih simulacija, PDES (engl. *Parallel Distributed Simulation*), čime će se ubrzati izvođenje simulacije, ili bolje i ravnomjernije rasporediti memorijski resursi. Ako simulacija zahtjeva nekoliko GB memorije, grupiranje simulacije može biti ujedno i jedini uspješan način izvedbe iste. Kako bi se izvođenje ubrzalo, mrežni uređaji ili nakupine (engl. *cluster*) trebaju imati malo vrijeme odziva, te model treba imati sposobnost paralelnog nasljeđivanja. Komunikacijski sloj je MPI (engl. *Message Passaging Interface*), ali u slučaju da korisnik nema MPI moguće je izvršavati neke osnovne testove putem postojećih kanala. Slika 3.8, prema [1], objašnjava logičku arhitekturu paralelne simulacijske jezgre.



Slika 3.8. Logička arhitektura paralelne simulacijske jezgre

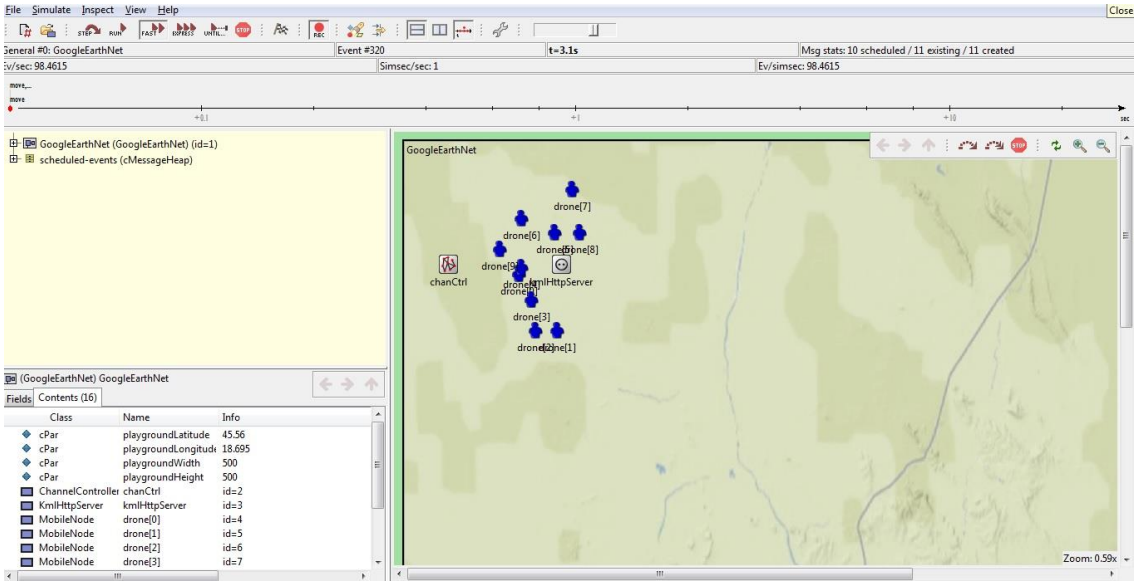
Mrežna emulacija, uz simulaciju u stvarnom vremenu, je moguća zahvaljujući jezgri koja upravlja raspoređivanjem događaja. Raspoređivanje u alatu OMNeT++ sadrži testnu inačicu stvarne simulacije i pojednostavljeni primjer mrežne imitacije. Spajanjem alata OMNeT++ s drugim simulatorima (hibridna operacija), iliti HLA (engl. *High level architecture*), u velikom dijelu ovisi o implementiranju vlastitog načina raspoređivanja, prema [1].

3.7. Korisničko sučelje Tkenv

Korisničko sučelje Tkenv je GUI prozor koji je povezan sa simulacijom objekta. Tkenv koristi tri metode: automatiziranu animaciju, prozore za prikaz modulskih izlaznih podataka i objektne ispitivače. Automatizirana animacija (animacija bez programiranja) može animirati tok poruka u mrežnim dijagramima i prikazivati promjene stanja na čvorovima. Budući da aplikacije mrežnih

simulacija rijetko kada trebaju biti potpuno prilagodljive promjenama, automatska animacija se često pokaže kao najbolji način prikazivanja simulacije. Može biti iskorišten za prikaza stanja ili sadržaja nekog objekta na najbolji mogući način (npr. histogramski objekt je prikazan grafički, histogramskim dijagramom). Također ga je moguće koristiti za ručno modificiranje objekta. U alatu OMNeT++, moguće je ispitati svaki simulacijski objekt. Stoga nije potrebno dodatno kodiranje jednostavnih modula kako bi se ispitivači mogli koristiti.

Potrebno je napomenuti da je moguće cijelo korisničko grafičko sučelje isključiti, te samu simulaciju pokretati kao čisti program preko komandne linije, što može biti posebno korisno kod izvršavanja većeg broja simulacija gdje potreban grafički prikaz za svaku od njih, što je prikazano na slici 3.9.



Slika 3.9. Korisničko sučelje Tkenv

Konačni cilj pokretanja neke simulacije jest prikupljanje rezultata na temelju kojih se može dobiti uvid u rad sustava. Detaljnije simulacije često daju velike količine podataka, koje je potrebno organizirati na razumljiv način. OMNet++ opisuje simulacijska izvršavanja (i rezultate dobivene iz njih) sljedećim pojmovima, prema [1]:

- model opisan izvršnim i NED datotekama. Datoteke modela smatraju se nepromjenjivima u svrhu pokusa, što znači da se modificiranjem C++ ili NED datoteka zapravo dobije novi, drugačiji model.
- Studija. Slijed pokusa u svrhu proučavanja nekog fenomena na jednom ili više modula. U studiji se obično radi veći broj pokusa iz kojih mogu biti izvučeni određeni zaključci. Jedna studija može sadržavati pokuse na različitim modelima, s tim što se jedan pokus uvijek izvršava na jednom točnom određenom modelu.

- pokus tj. istraživanje parametara na modulu
- mjerenje. Skup simulacijskih izvršavanja na istom modelu, s istim parametrima, ali s moguće različitim početnim točkama. Mjerenje može biti okarakterizirano postavkama parametara i simulacijske jezgre u INI datoteci, bez početnih točaka
- repliciranje ili ponavljanje mjerenja. Vrlo često, potrebno je napraviti nekoliko repliciranja, gdje svaka od njih imaju svoju početnu točku mjerenja. Određena je početnim točkama koje koristi.
- Izvršavanje. Jedno pokretanje simulacije, koje je određeno točnim vremenom izvršavanja i računalom na kojem se pokretanje simulacije vrši.

OMNeT++ podržava izvršavanje cijelog (ili dijela) pokusa u jednom slijedu izvršavanja. Nakon određivanja modela (izvršne i NED datoteke) i parametara (u INI datoteci) moguće je točno odrediti koja će se mjerenja raditi. Može se izvršavati slijed simulacija i pratiti njen napredak preko IDE. Više procesora i/ili procesorskih jezgri korisni su za izvršavanje više simulacija odjednom. Važnost više procesora i jezgri se često zanemaruje, ali njima se uz smanjivanje sveukupnog trajanja pokusa, isti pokusi rade i puno efikasnije.

Analiziranje rezultata simulacije je poduži proces koji oduzima puno vremena. U većini je slučajeva cilj doći do toga da se svakim izvršenjem simulacije dobiju isti podaci ili prikažu isti grafovi za različite module u modelu. Budući da se želi izbjeći ponavljanje koraka pri svakoj novoj simulaciji (redundancije), vrlo bitan čimbenik je automatizacija procesa. Zbog manjka podrške za automatizacijom u postojećim GUI alatima za analiziranje, mnogi korisnici su prisiljeni pisati skripte.

4. SIMULACIJSKI MODEL BESPILOTNIH LETJELICA

U ovome poglavlju prikazat će se simulacija leta bespilotnih letjelica (engl. *drone*) u simulacijskom alatu OMNeT++ na području grada Osijeka. Svaka od letjelica ima vlastitu nasumičnu putanju leta, te svaka od njih odašilje bežični signal u određenom radijusu. Cilj simulacije je prikazati njihova kretanja i preklapanje njihovih signala, preko kojih letjelice mogu razmjenjivati informacije. U ovom slučaju OMNeT++ ima funkciju računalne mreže, preko koje su u simulaciji programirani letovi i bežični signali bespilotnih letjelica.

Simulacija se sastoji od grafičkog prikaza leta bespilotnih letjelica na dva načina:

- korisničko sučelje Tkenv
- satelitska karta dobivena preko aplikacije Google Earth

Dakle, programski alati i aplikacije korišteni u ovoj simulaciji su OMNeT++ i Google Earth, koji su instalirani na 64-bitnom operacijskom sustavu Windows 7.

4.1. Simulacijski model

Kao osnova svake simulacije u alatu OMNeT++, koriste se konfiguracijske .ini datoteke. Tako je i za potrebe ove simulacije konfigurirana datoteka *omnetpp.ini* (slika 4.1):

```
[General]
record-eventlog = true
scheduler-class = "cSocketRTScheduler"

cmdenv-module-messages = true
cmdenv-event-banners = true

network = GoogleEarthNet
**.playgroundLatitude = 45.55
**.playgroundLongitude = 18.69
**.playgroundWidth = 3km
**.playgroundHeight = 3km
**.drone[*].trailLength = 450
**.drone[*].modelURL = "./~./omnetpp-4.6/samples2/Osijek Drones/dae/drone.dae"
**.drone[*].txRange = 650m
**.drone[*].modelScale = 0.5
**.drone[*].speed = 35km/h
**.drone[*].startX = uniform(0.1km, 1.9km)
**.drone[*].startY = uniform(0.1km, 1.9km)
```

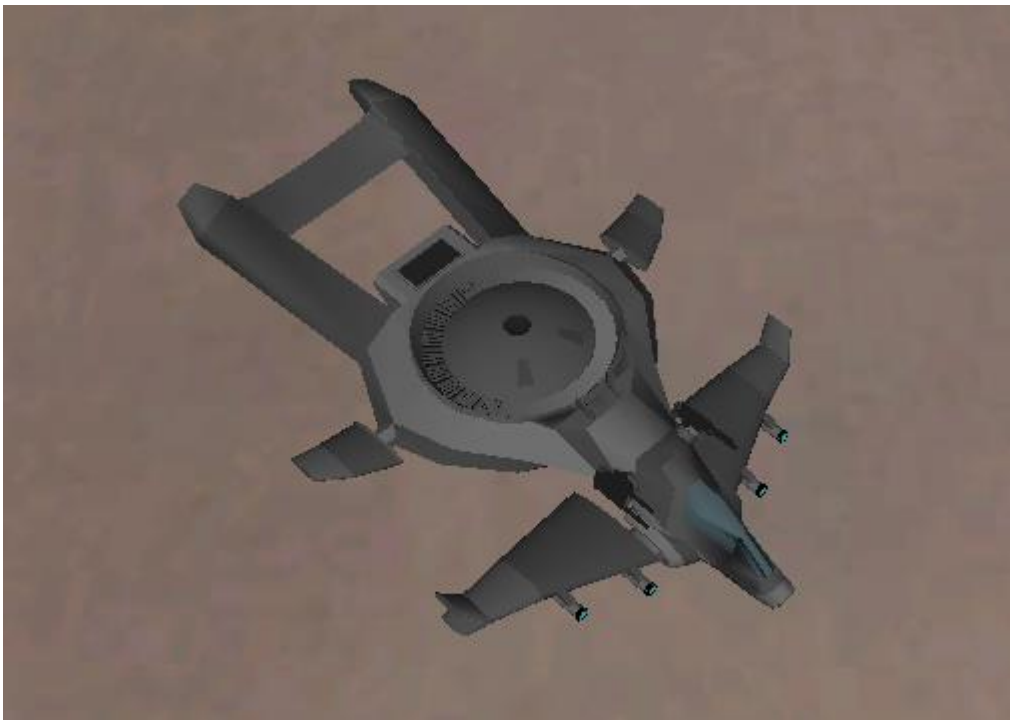
Slika 4.1. Datoteka omnetpp.ini

U datoteci .ini nalaze se parametri simulacijskog modela, podijeljeni u tri dijela:

- *record-eventlog*, *cheduler-class*, *cmdenv-module-messages*, *cmdenv-event-banners*, *network* – parametri koji sadrže ime klase modula korištenog u simulaciji i ime same simulacijske mreže, te parametri koji uključuju bilježenje događaja (engl. *event log*)

- ***.*playground* – parametri koji definiraju uvjete simulacije, odnosno lokaciju gdje će se simulacije izvršavati – zemljopisna dužina i širina, područje na kojem će letjelice letjeti
- ***.*drone* – parametri koji definiraju karakteristike bespilotnih letjelica, kao što su sam izgleda modela letjelica na karti, dužina prijeđenog puta, radius bežične mreže, brzina letjelice, te startna pozicija

Dakle, u datoteci *omnetpp.ini* za osnovne postavke simulacijskog modela zadano je da se bespilotne letjelice kreću nasumično na području od 9km^2 iznad grada Osijeka. Sve letjelice imaju identične odašiljače, te mogu odašiljati vlastite bežične signale u krugu od 650 metara. Radi jednostavnosti, svaki od tih signala je prikazan kao savršeni krug. Vizualizacija simulacije prikazuje same bespilotne letjelice kao 3D (trodimenzionalne) modele i njihove tragove prijeđenog puta. Isto tako kao poluprozirni krugovi u raznim bojama prikazani su i bežični signali tih letjelica, odnosno područje koje ti signali pokrivaju. U slučaju da dođe do međusobnog povezivanja dviju ili više mreža odjednom, letjelice čije su mreže povezane, bit će spojene bijelom linijom. Na slici 4.2 prikazan je trodimenzionalni model bespilotne letjelice, korišten u simulacijskom modelu.



Slika 4.2. 3D model bespilotne letjelice

Za dodatnu i detaljniju vizualizaciju, koristit će se aplikacija Google Earth. Naime, OMNeT++ podržava prikaz simulacijskih modela pomoću datoteka .kml. KML (engl. *Keyhole Markup Language*) je zasnivana na XML notaciji (struktura slična XML datotekama) te služi za prikaz

geografskih točaka u geografskim Internet preglednicima i specijaliziranim alatima za prikaz dvodimenzionalnih i trodimenzionalnih mapa. Jedan od takvih alata je Google Earth.

Tako je za ovaj simulacijski model napravljena datoteka *DroneFlight.kml*, koja ima sljedeći sadržaj (slika 4.3):

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
 <NetworkLink>
  <name>OMNeT++ Simulation</name>
  <visibility>0</visibility>
  <open>0</open>
  <refreshVisibility>0</refreshVisibility>
  <flyToView>0</flyToView>
  <Link>
   <href>http://localhost:4242/snapshot.kml</href>
   <refreshMode>onInterval</refreshMode>
   <refreshInterval>1</refreshInterval>
  </Link>
 </NetworkLink>
</kml>
```

Slika 4.3. Sadržaj datoteke DroneFlight.kml

U datoteci .kml nalazi se veza (engl. *link*) do simulacije, koja dolazi iz simulacijskog modela, čime preko datoteke .kml dolazi do vizualizacije trenutnog stanja simulacije. Stanje se osvježava periodički, svake sekunde. Aplikacija Google Earth čita te podatke i prikazuje ih.

4.2. Simulacijski moduli

Za potrebe ove izvedbe ove simulacije korišteno je više različitih modula. Kao što je već ranije spomenuto, vizualni prikaz simulacije u aplikaciji Google Earth ostvaruje se preko datoteke *DroneFlight.kml*.

Veza (u datoteci .kml navedena pod *href*) do simulacije se poziva preko HTTP GET zahtjeva (engl. *request*), koji je obrađen od strane *KmlHttpServer* modula. Modul sastavlja datoteku .kml iz dijelova dobivenih iz *IkmlFragmentProvider* objekta. Sve bespilotne letjelice u simulaciji su interpretirane kao mobilni čvorovi i definirane su u modulu *MobileNode*, gdje zajedno s modulom *ChannelController* čine *IkmlFragmentProvider* objekt u *KmlHttpServer* modulu. Prvi daje prikaz mobilnih čvorova, dok drugi daje prikaz spoja u .kml datoteci.

Kao što već spomenuto, moduli se sastoji od C++ kodiranih datoteka, s .cc ekstenzijom. Poneki moduli imaju definiranu i datoteku zaglavlja s ekstenzijom .h u kojoj su definirani pozivatelji zaglavlja, koji zavisi od drugih modula. Pozivanje takvih datoteka zaglavlja omogućava svim modulima koji je koriste, pristup objektima drugih modula, gdje se datoteke zaglavlja ponašaju kao zajednički, međudjeljivi objekti. Time je olakšano planiranje i izvedba simulacijskih mreža

jer kod većih mreža izravni međuspoj među samim modulima je teško pratiti, te datoteke zaglavlja olakšavaju cijeli proces, Jedan od modula koji koristi i .cc i .h datoteke je *KmlHttpServer* modul koji se sastoji od *KmlHttpServer.h* i *KmlHttpServer.cc* datoteka. Kod *KmlHttpServer.cc* datoteke se nalazi u prilozima na kraju rada, dok je *KmlHttpServer.h* definiran na slici 4.4:

```
#ifndef __KMLHTTPSERVER_H__
#define __KMLHTTPSERVER_H__
#include <omnetpp.h>
class cSocketRTScheduler;
class IKmlFragmentProvider
{
 public:
   virtual std::string getKmlFragment() = 0;
};
class KmlHttpServer : public cSimpleModule
{
 protected:
   static KmlHttpServer *instance;
   cMessage *rtEvent;
   cSocketRTScheduler *rtScheduler;
   char recvBuffer[4000];
   int numRecvBytes;
   std::vector<IKmlFragmentProvider*> providerList;
 protected:
   int findKmlFragmentProvider(IKmlFragmentProvider* p);
 public:
   KmlHttpServer();
   virtual ~KmlHttpServer();
   static KmlHttpServer *getInstance();
   virtual void addKmlFragmentProvider(IKmlFragmentProvider* p);
   virtual void removeKmlFragmentProvider(IKmlFragmentProvider* p);
 protected:
   virtual void initialize();
   virtual void handleMessage(cMessage *msg);
   virtual void handleSocketEvent();
   virtual std::string processHttpRequest(const char *request);
   virtual std::string getReplyFor(const char *uri);
   virtual std::string addHttpHeader(const char *content, const char *mimetype);
};
#endif
```

Slika 4.4. Sadržaj datoteke KmlHttpServer.h

Vidi se kako su u zaglavlju definirane klase, odnosno objekti *cSocketRTScheduler* i *IkmlFragmentProvider*.

Modul *KmlUtil* paralelno s *KmlHttpServer* modulom radi na generiranju *DroneFlight.kml* datoteke. U njemu se nalaze funkcije za generiranje oznaka koje će biti interpretirane u vizualnom prikazu. Neke od tih oznaka služe za prikaz 3D modela bespilotne letjelice, linije spajanja, radijus bežične mreže, i dr. Sastoji se od *KmlUtil.cc* (u prilozima) i *KmlUtil.h*, slika 4.5:

```
#ifndef __KMLUTIL_H__
#define __KMLUTIL_H__
#include <omnetpp.h>
#ifndef M_PI
#define M_PI 3.14159265
#endif
class KmlUtil
{
 public:
   struct Pt2D {float lon, lat;  Pt2D() {lon=lat=0;}  Pt2D(float lo, float la) {lon=lo; lat=la;}};
   struct Pt3D {float lon, lat, alt;   Pt3D() {lon=lat=alt=0;}   Pt3D(float lo, float la, float al)
{lon=lo; lat=la; alt=al;}};
   static std::string folderHeader(const char *id=NULL, const char *name=NULL, const char
*description=NULL);
   static std::string placemark(const char *id, float lon, float lat, float alt, const char
*name=NULL, const char *description=NULL);
   static std::string lineString(const char *id, const std::vector<Pt2D>& pts, const char
*name=NULL, const char *description=NULL, const char *color=NULL);
   static std::string lines(const char *id, const std::vector<Pt2D>& pts, const char *name=NULL,
const char *description=NULL, const char *color=NULL);
   static std::string disk(const char *id, float lon, float lat, float r, const char *name=NULL, const
char *description=NULL, const char *color=NULL);
   static std::string model(const char *id, float lon, float lat, float heading, float scale, const char
*link, const char *name=NULL, const char *description=NULL);
   static std::string track(const char *id, const std::vector<Pt2D>& pts, float timeStep, float
modelScale, const char *modelLink, const char *name=NULL, const char *description=NULL,
const char *color=NULL);
   static double y2lat(double lat, double yoff) { return lat - yoff / 111111; }
   static double x2lon(double lat, double lon, double xoff) { return lon +  xoff / 111111 /
cos(fabs(lat/180*M_PI)); }
   static void hsbToRgb(double hue, double saturation, double brightness, double& red, double&
green, double &blue);
};
#endif
```

Slika 4.5. Sadržaj datoteke KmlUtil.h

Idući modul koji je konfiguriran već je ranije spomenut. Radi se o *MobileNode* modulu, koji prikazuje mobilne čvorove (bespilotne letjelice) stalnom brzinom, sa smjerom kretanja koji se mijenja prilikom svake promjene stanja. Mobilnim čvorovima omogućuje i prikaz vlastite pozicije, traga kretanja, doseg odašiljanja, i sl. preko datoteke .kml. Isto kao i kod prethodnog modula, datoteka *MobileNode.cc* se nalazi u prilozima na kraju rada.

Modul *ChannelController* služi za vizualizaciju povezanosti među mobilnim čvorovima preko dattoeke .kml, te je odgovoran za praćenje međusobne udaljenosti mobilnih čvorova. Kao modul *KmlHttpServer,* sastoji se od vlastitih datoteka *ChannelController.cc* i *ChannelController.h,* od koji je prvospomenuta prikazana u prilozima, dok se na slici 4.6 ispod nalazi prikaz potonje:

```
#ifndef __CHANNELCONTROLLER_H_
#define __CHANNELCONTROLLER_H_
#include <omnetpp.h>
#include "KmlHttpServer.h"
#include "KmlUtil.h"
class IMobileNode
{
 public:
   virtual double getX() = 0;
   virtual double getY() = 0;
   virtual double getTxRange() = 0;
};
class ChannelController : public cSimpleModule, public IKmlFragmentProvider
{
 protected:
   static ChannelController *instance;
   std::vector<IMobileNode*> nodeList;
   double playgroundLat;
   double playgroundLon;
   virtual void initialize();
   virtual void handleMessage(cMessage *msg);
   virtual std::string getKmlFragment();
   int findMobileNode(IMobileNode* p);
 public:
   ChannelController();
   virtual ~ChannelController();
   static ChannelController *getInstance();
   virtual void addMobileNode(IMobileNode* p);
   virtual void removeMobileNode(IMobileNode* p);
};
#endif
```

Slika 4.6. Sadržaj datoteke ChannelController.h

Od ostalih modula koji se koriste u simulaciji, posebno je potrebno navesti modul *SocketRTScheduler*. To je modul koji vrlo bitan za izvođenje simulacije, jer se opetovano izvodi u stvarnom vremenu (engl. *Real-time scheduler*). Ovaj modul je zadužen za okidanje drugih modula u stvarnom vremenu i jednakim vremenskih razmacima. Također, omogućuje i vanjsku komunikaciju, što znači da se njegovo opetovano okidanje (a samim time i pokretanje ostalih modula) može prikazati nekim drugim alatom. U ovom slučaju ovdje se radi o aplikaciji Google Earth, koja to radi preko datoteke *DroneFlight.kml*. *SocketRTScheduler.cc* se nalazi u prilozima, a *SocketRTScheduler.h* je definiran na slici 4.7.

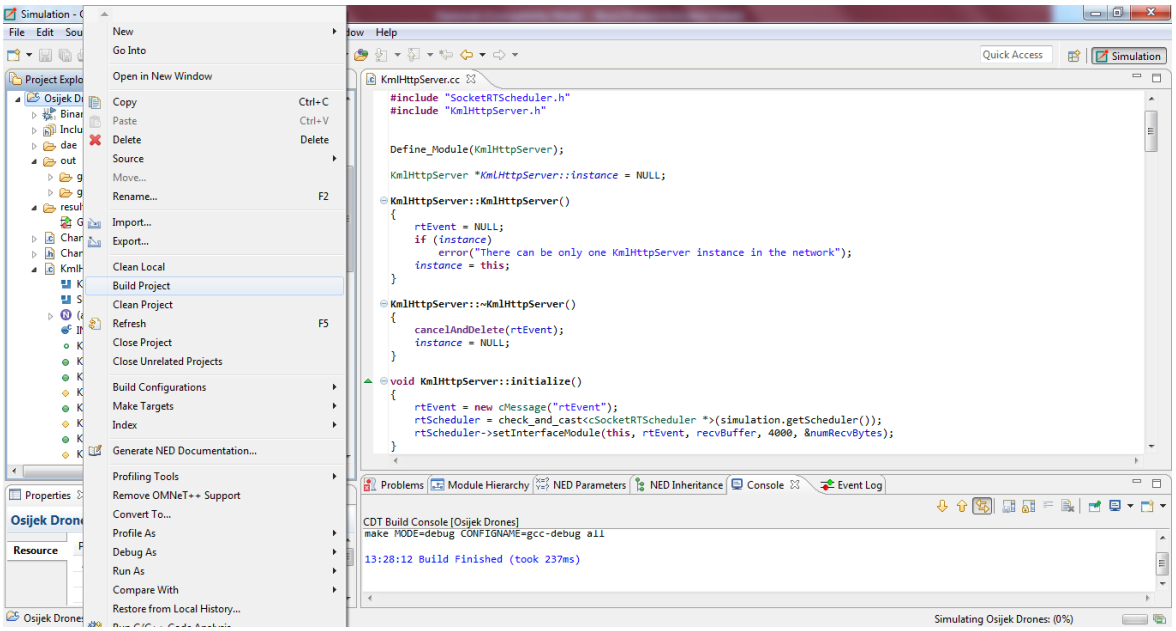
```
#ifndef __CSOCKETRTSCHEDULER_H__
#define __CSOCKETRTSCHEDULER_H__
#include <platdep/sockets.h>
#include <platdep/timeutil.h>
#include <omnetpp.h>
class cSocketRTScheduler : public cScheduler
{
 protected:
   int port;
   cModule *module;
   cMessage *notificationMsg;
   char *recvBuffer;
   int recvBufferSize;
   int *numBytesPtr;
   timeval baseTime;
   SOCKET listenerSocket;
   SOCKET connSocket;
   virtual void setupListener();
   virtual bool receiveWithTimeout(long usec);
   virtual int receiveUntil(const timeval& targetTime);
 public:
   cSocketRTScheduler();
   virtual ~cSocketRTScheduler();
   virtual void startRun();
   virtual void endRun();
   virtual void executionResumed();
   virtual void setInterfaceModule(cModule *module, cMessage *notificationMsg,
                                   char *recvBuffer, int recvBufferSize, int *numBytesPtr);
   virtual cMessage *getNextEvent();
   virtual void sendBytes(const char *buf, size_t numBytes);
   virtual void close();
};

#endif
```

Slika 4.7. Sadržaj datoteke SocketRTScheduler.h

4.3. Izvedba simulacijskog modela

Da bi se simulacijski model pokrenuo, prvo je potrebno izgraditi projekt (engl. *build project*). Tom akcijom se izgrađuju izvršne datoteke unutar samog projekta, koje su nužne za neometano pokretanje i rad same simulacije. To je moguće napraviti preko komandne linije (najčešće korišten na Linux platformama) ili preko IDE OMNeT++ sučelja. Izgled IDE sučelja je prikazan na slici 4.8.

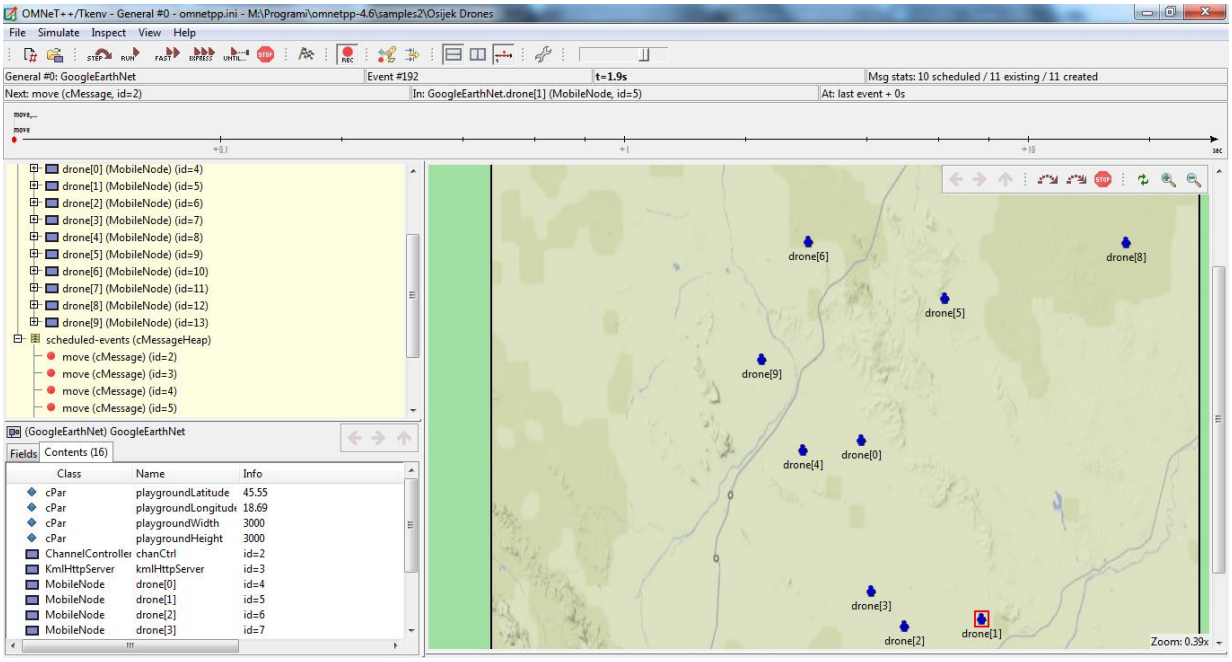
Kao što se vidi na slici 4.8, u IDE korisničkom sučelju je moguće pisati i raditi revizije modula i njihovog koda. Također, moguće je odrediti prioritete i svojstva svim dijelovima simulacijskog modela, te na kraju provjeriti i rezultate simulacije. Iz IDE korisničkog sučelja simulacijski model se pokreće kao OMNeT++ simulacija.



Slika 4.8. IDE korisničko sučelje

Iduće korisničko sučelje koje se otvara je korisničko sučelje Tkenv. Ono kao grafičko sučelje prikazuje kretanje bespilotnih letjelica na karti, na nekom određenom području. Svaka letjelica ima ime *drone* (engl.) i vlastito numeriranje od 0 do 9.

Na slici 4.9 je vidljiv sadržaj svakog mobilnog čvora (bespilotne letjelice) i događaji (engl. *event*) do kojih dolazi tijekom simulacije. Također, mogu se detaljnije ispitati dijelovi simulacije, prije pokretanja iste. Simulacija može se izvršavati u više brzina.

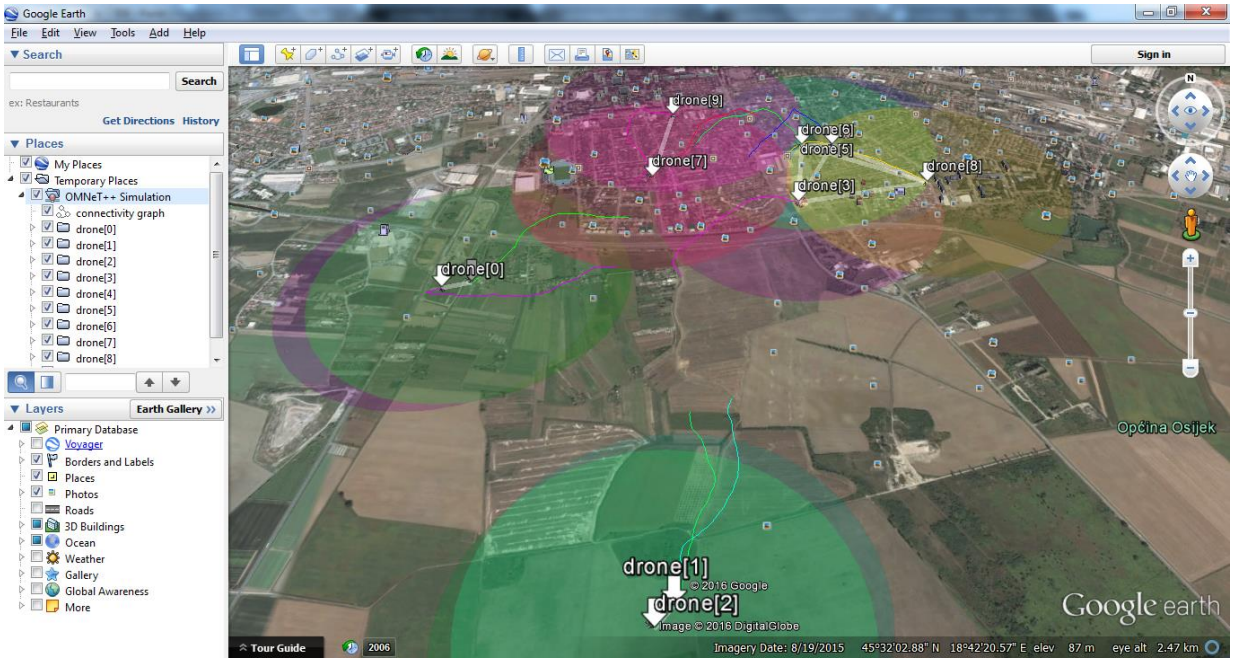


Slika 4.9. Tkenv korisničko sučelje

Budući da se simulacija događaja i njihovo bilježenje vrše u mikrookruženju, mali pomaci u kretnji bespilotnih letjelica u aplikaciji Google Earth ne mogu biti prikazani, odnosno nisu uočljivi. Stoga je ovaj simulacijski model nužno pokrenuti u brzom ili ekspresnom (najbržem) modu rada, kako bi se kretanje letjelica moglo prikazati na 3D karti u aplikaciji Google Earth.

4.3.1. Scenarij 1 – nadzor grada Osijeka

U prvom scenariju simulacijskog modela na slici 4.10 prikazano je svih 10 bespilotnih letjelica, u letu na području iznad grada Osijeka. Svaka od letjelica ima vlastiti radijus od 650 metara unutar kojeg se odašilje bežični signal. Da bi se uspostavila komunikacija i razmjena informacija među pojedinim letjelicama, potrebno je da barem jedna od njih bude unutar navedenog radijusa neke druge letjelice. Datoteka *omnetpp.ini* ovog scenarija nalazi se u prilozima pod P.4.3.1.



Slika 4.10. Google Earth simulacija

Budući da se simulacija izvodi u stvarnom vremenu, međusobni spoj i položaji letjelica se stalno mijenjaju. To znači da u nekim vremenskim periodima, letjelice ne mogu komunicirati međusobno, jer nijedna nije unutar bežičnog pojasa neke druge letjelice.

Slika 4.10 prikazuje zbroj svih modula korištenih u simulacijskom modelu (navedeni u poglavlju 4.2.). U ovom uhvaćenom trenutku vidi se kako su letjelice *drone1* i *drone2* u međusobnoj komunikaciji, te su u mogućosti razmjenjivati informacije. Štoviše, njihovi bežični pojasevi se gotovo preklapaju (zelena i plava boja), a po njihovom tragu kretanja vidi da se kreću sličnom putanjom.

Letjelice *drone0* i *drone4* (maslinasto zelena i ljubičasta boja) su u sličnom položaju kao i letjelice gore navedene.

Letjelice *drone7* i *drone9* su također u međusobnoj komunikaciji. Međutim, po njihovim kretnjama može se zaključiti kako će *drone7* ubrzo izaći van dometa *drone9*, čime će prestati njihova razmjena informacija.

Najzanimljivija situacija je kod letjelica *drone3*, *drone5*, *drone6* i *drone8*. Naime, *drone3*, *drone5* i *drone6* su u međusobnoj komunikaciji, dok su ujedno i sve tri zajedno u komunikaciji s *drone8* jer se nalaze u njenom dometu. Time je pokazano kako je moguća komunikacija između više bespilotnih letjelica na zadanome području. Sigurno je za pretpostaviti da je tijekom vremena u nekom trenutku moguće da se sve letjelice nađu u uskom području, gdje će sve jedna drugoj biti u dometu. Time sve mogu simultano sinkronizirati sve podatke. Ovakve slučajeve je posebno moguće ostvariti i iskoristiti za simulaciju ponašanja bespilotnih letjelica koje će biti korištene u stvarnom vremenu i prostoru. Time se može istražiti ponašanje letjelica na raznim terenima i vremenskim uvjetima. Informacije koje sadrže i međusobno razmjenjuju, mogu sadržavati slike i snimke iz zraka, i sl.

4.3.2. Scenarij 2 – potraga za ljubimcem

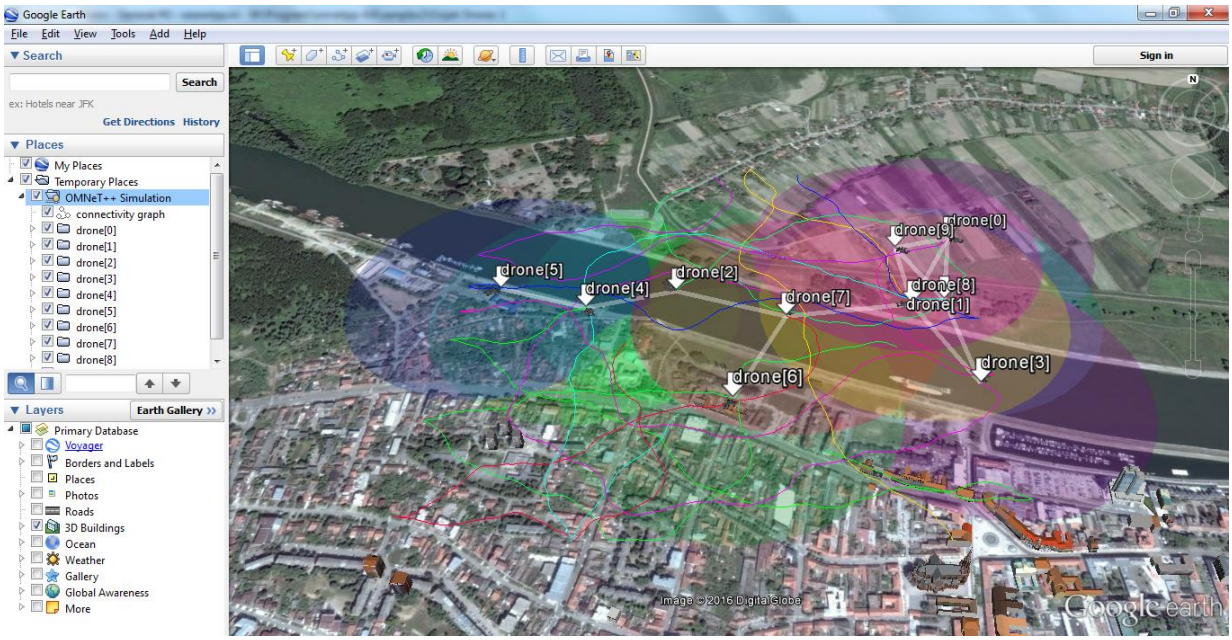
U ovome scenariju zamislimo da se bespilotne letjelice koriste u potrazi za izgubljenim psom na obali Drave, u gradu Osijeku. Naime, vlasnik je izgubio psa iz vida. GPS lokator na ogrlici psa je odaslao zadnje koordinate gdje se nalazi ogrlica, a time vjerojatno i sam pas.

Koordinate koje je GPS lokator posljednji puta poslao su odmah definirane u *omnetpp.ini* datoteci kao:

****.playgroundLatitude** = 45.551611*
****.playgroundLongitude** = 18.696703*

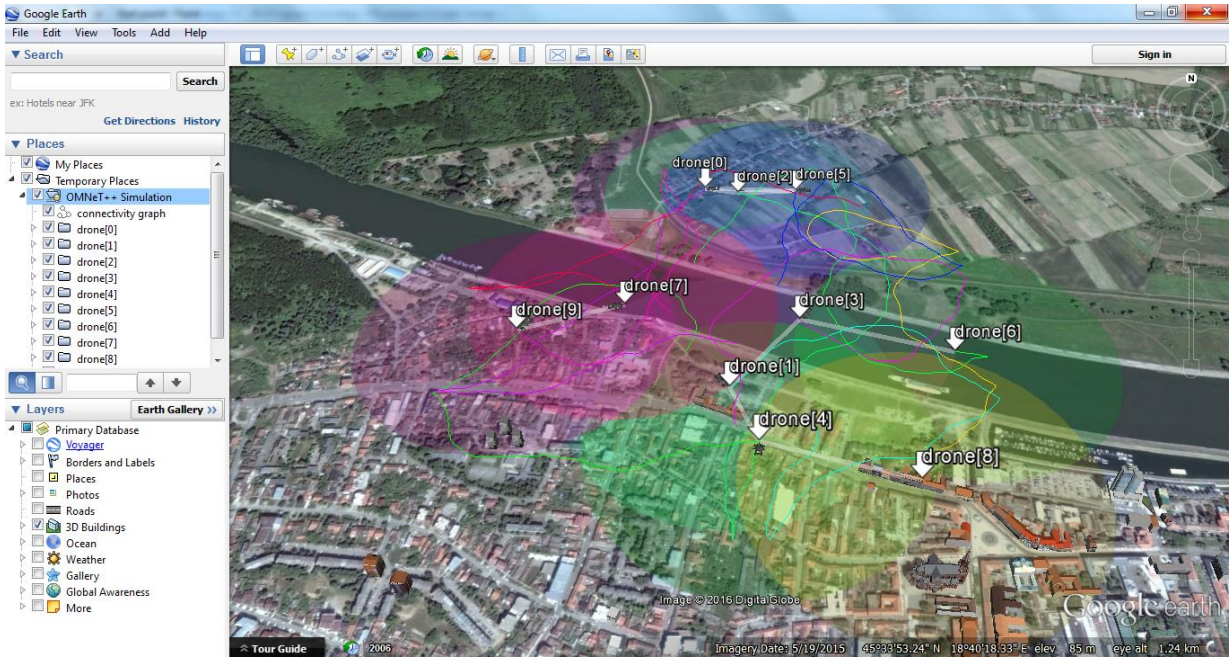
Te koordinate su zadane kao centar potrage, te će se potraga vršiti u radiusu od 500 metara od te točke (GPS lokator je oštećen). Maksimalna brzina leta bespilotnih letjelica iznosi 12 m/s, odnosno 43 km/h, jer je brzina pronalaska ljubimca od velike važnosti. Cijela *omnetpp.ini* datoteka ovoga scenarija se nalazi u prilogu P.4.3.2.

Na slici 4.11 je vidljivo kako je većina bespilotnih letjelica započela potragu izvan kruga od 500 metara od polazišne točke. Jedina letjelica koja je krenula od nje, jest *drone7*. Ostale letjelice se kreću vlastitim putanjama oko te točke te polako zatvaraju krug potrage, kako bi se izbjegla mogućnost da im nešto promakne. Isto tako, vidi se kako svaka od bespilotnih letjelica ima vlastiti maksimalan domet bežičnog signala, koji iznosi 300 metara (vidljivo u raznim bojama na slici 4.5), pomoću kojeg pokušavaju uhvatiti signal koji ima kratki domet lokatora s ogrlice.



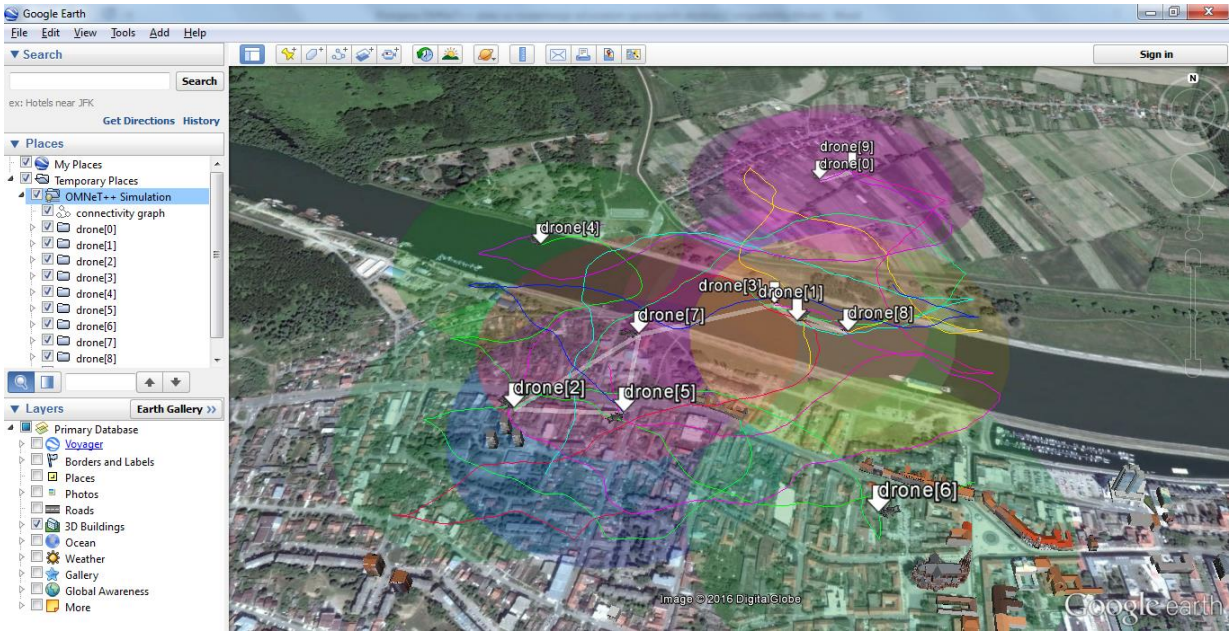
Slika 4.11. Početak potrage

Na slici 4.12 se vidi kako su bespilotne letjelice napravile prsten oko područja pretrage, najviše se koncentrirajući na samo obalu Drave, u potrazi za psom. Najbolje se vidi manji prsten koji su napravile letjelice *drone1*, *drone3*, *drone4*, *drone6* i *drone8*, gdje bijela linija koja ih međusobno spaja, jest zapravo linija komunikacije među njima, preko koje koordiniraju potragu i uspoređuju rezultate potrage.



Slika 4.12. Stvaranje prstena

Na slici 4.13 nalazi se prikaz završetka potrage, gdje se sve bespilotne letjelice usmjeravaju prema *drone1*, *drone3* i *drone8*, koje su pronašle lokator, te se potraga može dovršiti izravno na terenu.

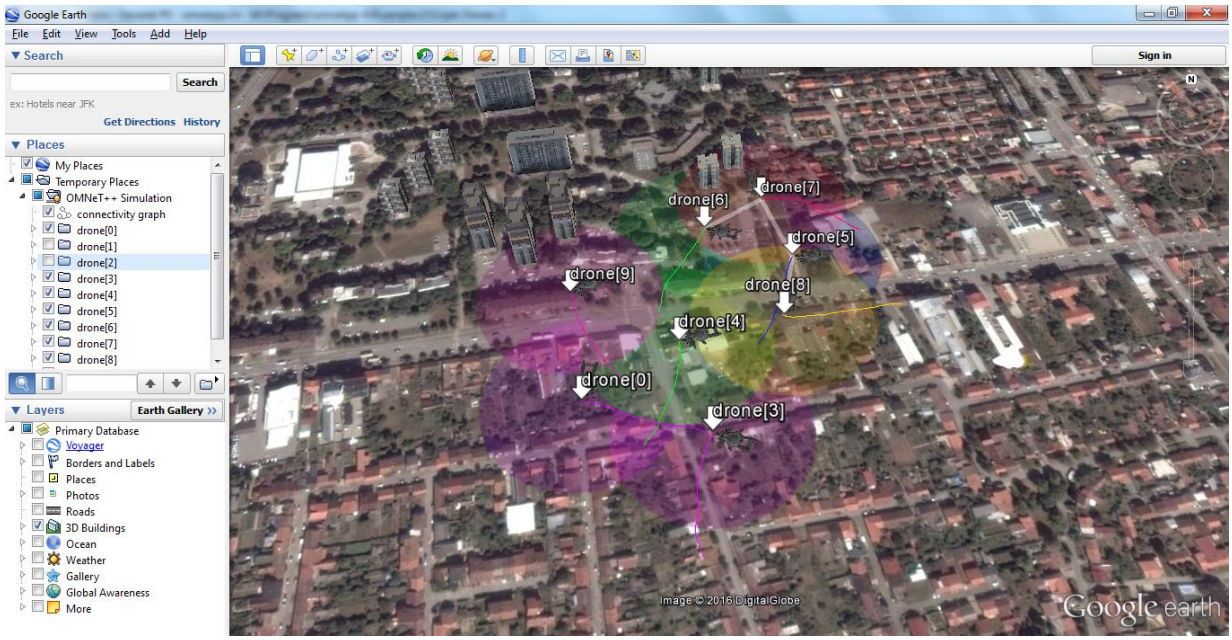


Slika 4.13. Sužavanje područja potrage

4.3.3. Scenarij 3 – potraga za izgubljenim djetetom

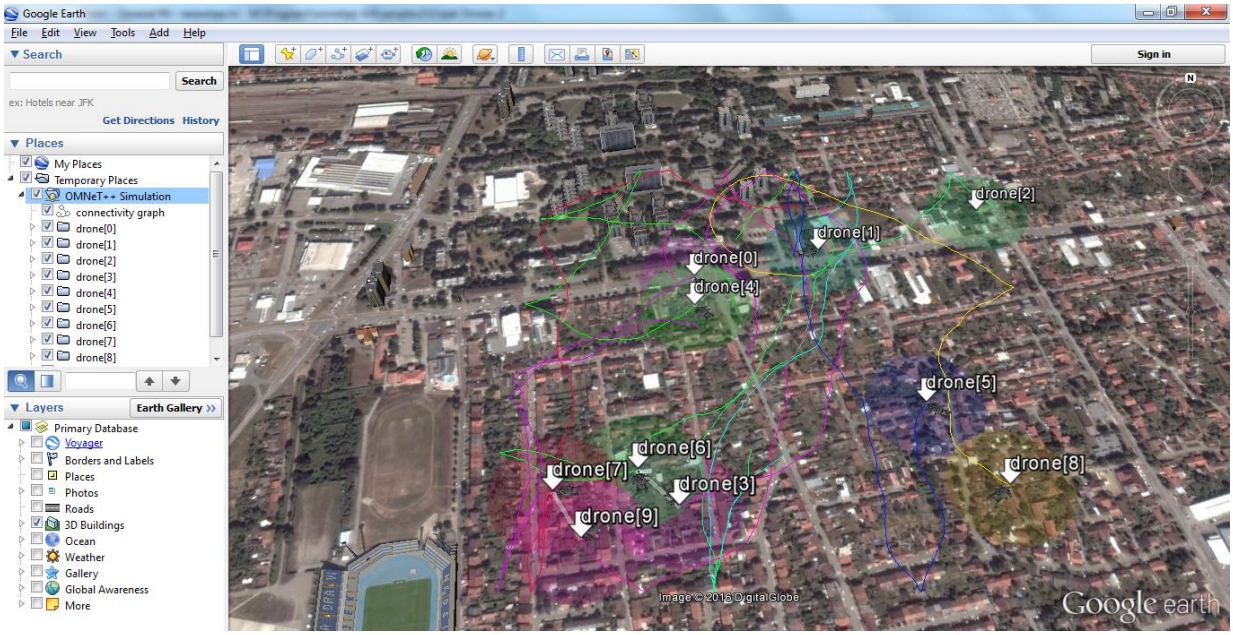
U trećem scenariju bespilotne letjelice će koristiti za potragu za izgubljenim djetetom na području oko ulice Martina Divalta, Sjenjaka i Vatrogasnog naselja, u gradu Osijeku. Datoteka *Omnetpp.ini* ovog scenarija može se naći u prilogu P.4.3.3.

Na slici 4.14 prikazan je početni raspored bespilotnih letjelica, oko mjesta gdje je izgubljeno dijete zadnji puta viđeno. Pretpostavimo da su na letjelicama instalirane kamere pomoću kojih snimaju okolinu, te izravno šalju signal do centralnog računala, gdje se nalazi program za prepoznavanje lica. Domet kamera iznosi 80 metara, te je prikazan kružnica u raznim bojama, oko letjelica.



Slika 4.14. Početak potrage

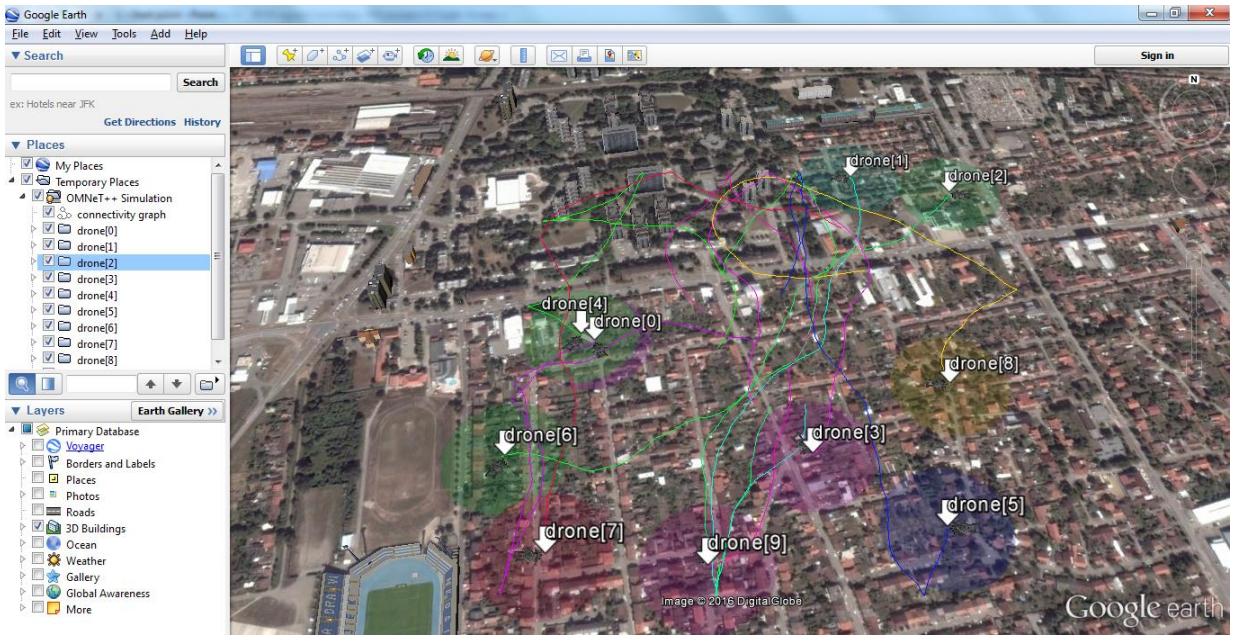
Budući da letjelice nisu pronašle nikoga tko odgovara izgledu izgubljenog djeteta, potrebno je proširiti radijus potrage, što je prikazano na slici 4.15. Budući da se ne zna općeniti smjer niti navike djeteta, letjelice vrše potragu u svim smjerovima, s time da većina letjelica pretražuje područje Vatrogasnog naselja, jer se u njemu nalaze obiteljske kuće, te je s visine moguće snimiti veći dio terena, za razliku od naselja Sjenjak, gdje se pretežito nalaze zgrade.



Slika 4.15. Širenje potrage

Na slici 4.16 vidi se kako su letjelice *drone3*, *drone6*, *drone7* i *drone9* koncentrirale potragu na područje od dva ulična bloka jer je na tom području uočeno nekoliko djece koja po izgledu sliče

izgubljenom djetetu, što znači da je to područje od posebnog interesa, te ga je potrebno dodatno istražiti.

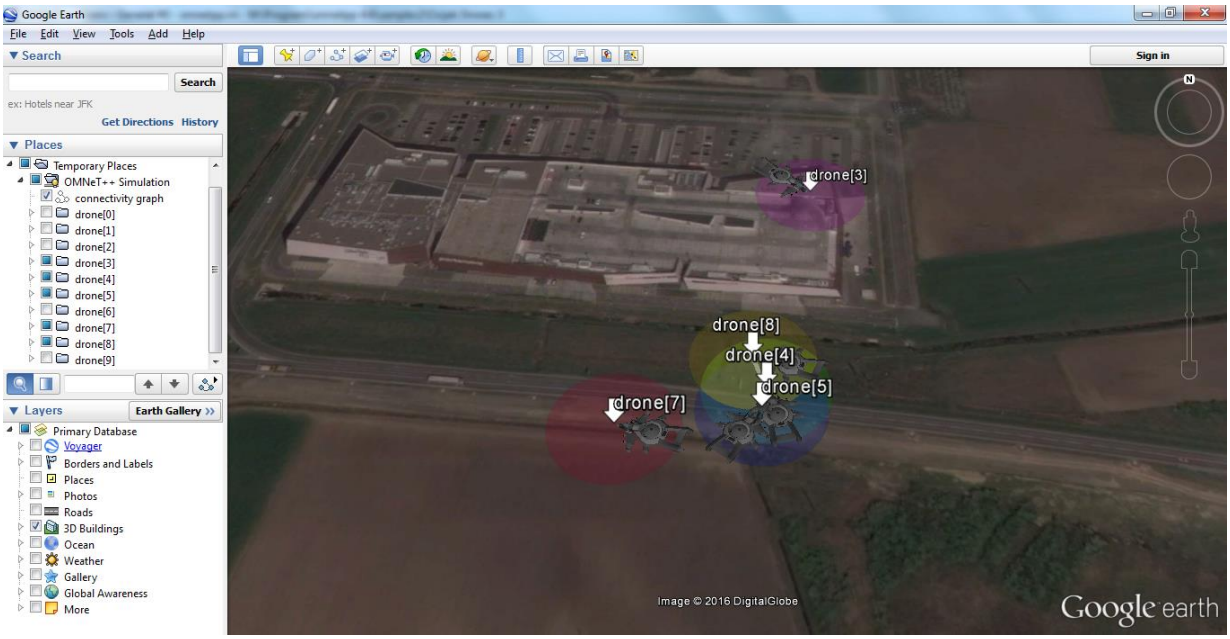


Slika 4.16. Daljnje širenje potrage

4.3.4. Scenarij 4 – prometna nesreća

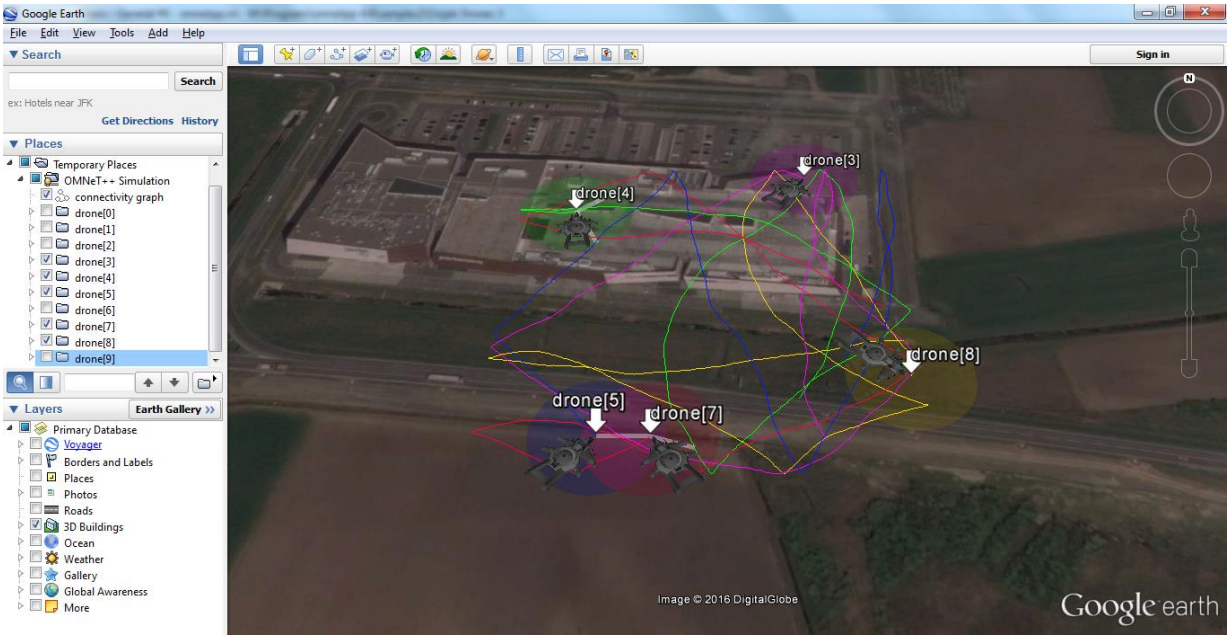
Scenarij 4 prikazuje upotrebu bespilotnih letjelica nakon teška prometne nesreće koja se dogodila na osječkoj obilaznici kod trgovačkog centra. Letjelice pomažu u očevidu (2D i 3D snimanje radi lakše rekonstrukcije nesreće), te se snimke u stvarnom vremenu izravno šalju nadzornicima letjelica, gdje se u slučaju pronalaska točke od većeg značaja, nadzornik može preuzeti upravljanje određene letjelice, te je izravno usmjeriti na bliže i detaljnije snimanje te točke. Datoteka *omnetpp.ini* ovog scenarija nalazi se u prilogu P.4.3.4.

Slika 4.17 prikazuje početak očevida bespilotnih letjelica. Budući da se radi o lančanom sudaru nekoliko vozila pri velikim brzinama, vozila i njihovi dijelovi su razasuti po većem području. Letjelice započinju očevid nad područjem gdje se nalazi veći broj vozila, a ujedno i najveći broj materijala nad kojima je potrebno napraviti očevid. Domet kamera je prikazan obojanim kružnicama oko letjelica, te iznosi 30 metara.



Slika 4.17. Početak očevida

Na slici 4.18 prikazan je očevid koji je u tijeku a u njemu sudjeluju bespilotne letjelice, tako što pokrivaju veliko područje oko samog centra prometne nesreće. Za svaku letjelicu je prikazan prijeđen put (obojane linije), te se vidi kako se radi o zaista velikom području za koji je potrebno napraviti očevid. Letjelice ovdje imaju lakši i brži pristup terenu, za razliku od ljudstva, te samim time nije ni potrebno velik broj ljudi za sam očevid. Letjelice ujedno služe i za osiguranje područja očevida, jer svaki prijestup mogu gotovo odmah uočiti, te alarmirati nadzornika, kako bi se moglo pravodobno reagirati na moguću kontaminaciju područja na kojem se dogodila prometna nesreća.



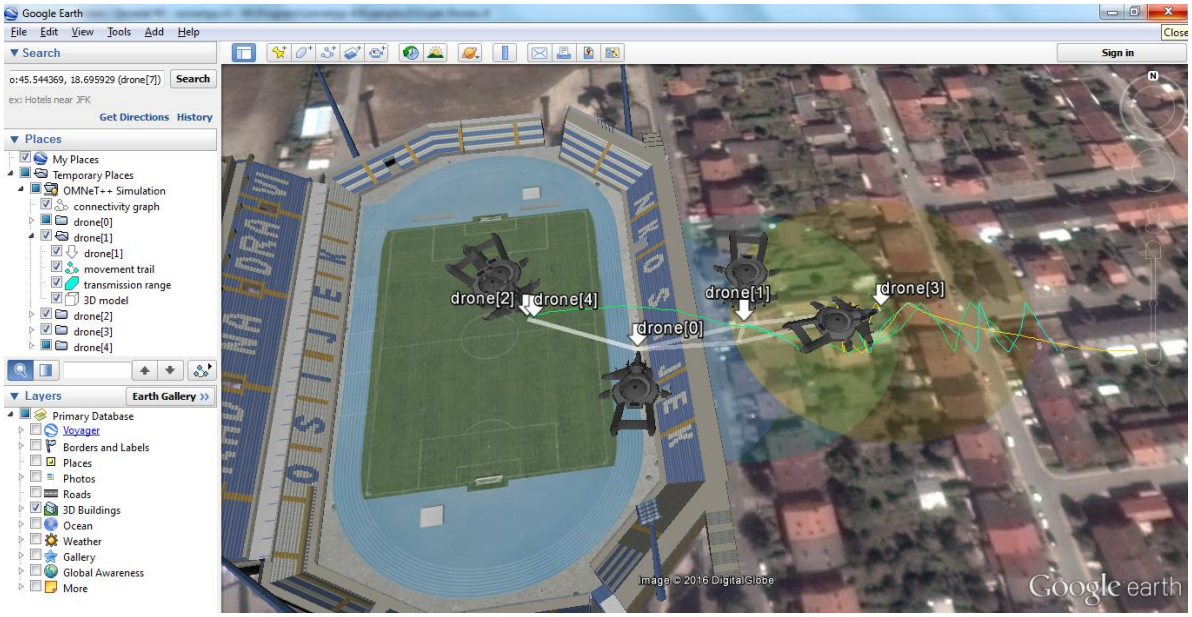
Slika 4.18. Očevid u tijeku

Bespilotne letjelice u ovakvim scenarijima mogu biti vrlo važan čimbenik jer često vremenski uvjeti i/ili prohodnost terena utječu na kvalitetu i brzinu provedbe očevida, što su dvije najvažnije stavke kod vršenja očevida nad prometnim nesrećama.

4.3.5. Scenarij 5 – navijački izgredi na stadionu

Posljedni scenarij prikazuje korištenje bespilotnih letjelica za snimanje i nadzor navijača i huligana na nogometnoj utakmici na stadionu Gradski vrt, u gradu Osijeku. Snimke se poslije mogu pregledati u slučaju da dođe do navijačkih izgreda na utakmici, kako bi se lakše i brže pronašli počinitelji. Budući da se počinitelji često maskiraju tijekom izgreda ili ih je vrlo teško raspoznati tijekom počinjenja samih nereda, zbog velikog broja ljudi na stadionu, bespilotne letjelice mogu prije, poslije i za vrijeme izgreda pružiti na raspolaganje velik broj snimaka iz više kutova. *Omnetpp.ini* datoteka ovog scenarija nalazi se u prilozima pod P.4.3.5.

Slika 4.19 prikazuje kretanje bespilotnih letjelica tijekom nadzora nogometne utamice. Obojane kružnice prikazuju domet kamera (50 metara), dok obojane linije pokazuju put koje su letjelice prešle. Time je uočljivo kako letjelice *drone1*, *drone3* i *drone4* imaju isto polazište, izvan stadiona, dok su letjelice *drone0* i *drone2* poletjele izravno na stadionu. Bijela linija koja međusobno spaja letjelice prikazuje komunikacijski kanal među istima, preko kojeg se letjelice međusobno sinkroniziraju. Naime, osobi koja nadzire let svih letjelica nije potrebno da se spaja posebno sa svakom letjelicom radi koordinacije, već preko jedne letjelice koordinira let svih ostalih letjelica, jer su međusobno povezane preko komunkacijskog kanala. To znači da ako se na kameri neke letjelice uoči neko mjesto koje je potrebno odmah istražiti, letjelice će se međusobno optimalno rasporediti za nadzor tog područja, i ostatka stadiona.



Slika 4.19. Nadzor nogometnog stadiona

Budući da se prikaz simulacija izvodi u aplikaciji Google Earth, za svaku letjelicu je moguće uključiti ili isključiti prikaz nekog elementa, kao što su na primjeu *drone1*:

- **drone[1]** – *ime letjelice*
- **movement trail** – *prijeđeni put letjelice* drone1
- **transmission range** – *domet letjelice (kružnica); u ovom scenariju korišten kao domet kamere*
- **3D model** – *prikaz trodimenzionalnog modela bespilotne letjelice*
- **connectivity graph** – *prikaz linije komunikacije među letjelicama*

4.4. Analiza scenarija

Može se reći da je kroz scenarije u ovome radu (nadzor grada Osijeka, potraga za ljubimcem, potraga za izgubljenim djetetom, prometna nesreća, navijački izgredi na stadionu) vidljiva raznolika upotreba simulacijskog alata OMNeT++.

U prvom scenariju (nadzor grada Osijeka), prikazan je simulacijski model općenite primjene, gdje se bespilotne letjelice mogu upotrebljavati na više načina. Uz pomoć aplikacije Google Earth, vizualizirana je simulacija leta bespilotnih letjelica, te područje koje iste mogu pokrivati.

Drugi i treći scenarij (potraga za ljubimcem, potraga za izgubljenim djetetom) prikazuju ponašanje računalom upravljane okoline, odnosno bespilotnih letjelica, gdje je u malom vremenskom roku i na relativno uskom području grada Osijeka potrebno iskoristiti sve mogućnosti simulirane okoline, kako bi se u stvarnom vremenu postigli optimalni razultati.

Prometna nesreća je četvrti simulacijski model, gdje je prikazano kako računalom upravljana okolina omogućuje ljudima lakšu obradu podataka i rekonstrukciju same nesreće. Usklađivanjem bespilotnih letjelica računalom, vrijeme i kvaliteta obrade višestruko su poboljšani.

Zadnji, peti scenarij (navijački izgredi na stadionu), prikazuje kako bi izgledao nadzor nad navijačima za vrijeme nogometne utakmice, u stvarnome vremenu. Ovakvom simulacijom, moguće je predvidjeti kretanja navijača i bespilotnih letjelica, čime se na kraju mogu dobiti optimalna mjerenja i podaci, te maksimalno iskoristiti mogućnosti računalom upravljanih okolina u ovakvim sustavima.

5. ZAKLJUČAK

Cilj ovog rada je bio predstaviti i objasniti što je simulacijski alat OMNeT++, te kakva je njegova primjena. U simulacijama u radu korišteni su trodimenzionalni modeli bespilotnih letjelica, koji uspostavljaju međusobnu komunikaciju kada im se preklapaju radijusi dometa bežičnih mreža. Scenariji koji su obrađeni u radu, samo su neki od mogućih primjena simulacijskog alata OMNeT++, pomoću kojeg je moguće simulirati stvarne uvjete i prikazati kako će se računalom upravljani sustavi ponašati u vanjskoj okolini. Na temelju simulacija koje su izvedene u pet različitih scenarija, vidljivo je kako je alat OMNeT++ uspješno rješenje za simulaciju modela čiji se rezultati mogu pokazati u stvarnome vremenu i ostvaren je prikaz simulacijskog modela na stvarnim lokacijama (više točaka u gradu Osijeku), unaprijed snabdijevajući letjelice početnim koordinatama i povezivanjem simulacije s aplikacijom Google Earth.

LITERATURA

[1] A. Varga, R. Hornig, An overview of the OMNeT++ simulation environment, Simutools '08, sv. 60, str.170-180., Marseille, Francuska, 2008.

[2] A. Varga, The OMNeT++ discrete event simulation system, IEEE Transactions on Education, br. 4, sv. 42, studeni 1999.

[3] R. Baheti, H. Gill, Cyber-physical systems, The Impact of Control Technology, 2011.

[4] T. Chamberlain, Learning OMNeT++, Packt Publishing, Birmingham, 2013.

[5] D. Pehar, Programski sustav za razvoj i verifikaciju algoritama usmjeravanja za komunikacijske mreže s višenamjenskim protokolom s komutacijom oznaka, Zagreb, 2011.

[6] OMNeT++ Discrete Event Simulator, https://omnetpp.org/, pristupljeno 20.04.2016

[7] N. Sakur, R. Membrath, Modeling and simulation of IEEE 802.11g using OMNeT++, IGI Global, str. 590-608, siječanj, 2010.

[8] F. Bause, P. Buchholz, J. Kriege, S. Vastag, Simulating Process Chain Models with OMNeT++, Simutools '08, sv. 19, str.196-206, Marseille, Francuska, 2008.

[9] T. Steinbach, H.D. Kenfack, F. Korf, T.C. Schmidt, An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy, Simutools '11, str. 375-382, 2011. , Barcelona, Španjolska

[10] E.A. Lee, S.A. Seshia, Introduction to Embedded Systems - A Cyber-Physical Systems Approach, str. 227-294, LeeSeshia.org, 2011.

[11] S. Khaitan, Design Techniques and Applications of Cyber Physical Systems: A Survey, IEEE Systems Journal, str. 350-365, 2014.

[12] S. Karnouskos, Cyber-Physical Systems in the Smart Grid“, „2011 9th IEEE International Conference, str.20-23, Lisabon, Portugal, srpanj 2011.

[13] Omnet simulation, http://omnetsimulation.com/, pristupljeno 13.05.2016.

[14] A. Varga, OMNeT++ User Manual 4.6, OpenSim Ltd., 2014.

[15] Cyber-Physical Systems, http://cyberphysicalsystems.org/ ,pristupljeno 28.08.2016.

SAŽETAK

Računalom upravljane okoline predstavljaju novu generaciju sustava s integriranim računalnim i fizičkim sposobnostima kojima je omogućena interakcija s ljudima na više načina. OMNeT++ je diskretni simulator zasnovan na programskom jeziku C++, koji služi za simulaciju i modeliranje komunikacijskih mreža, višeprocesorskih, te raspodijeljenih ili paralelnih sustava. Besplatan je i otvorenog koda. Cilj razvoja samog alata OMNeT++ jest izrada konačnog proizvoda koji je snažan i besplatan simulacijski alat koji može biti korišten u akademskim, edukacijskim i istraživačkim institucijama, za simulaciju računalnih mreža i raspodijeljenih ili paralelnih sustava. U ovome radu prikazana je simulacija leta bespilotnih letjelica u simulacijskom alatu OMNeT++ na području grada Osijeka. Pokazana je njihova primjena u pet različitih scenarija. Cilj simulacije bio je prikazati njihova kretanja i preklapanje njihovih signala, preko kojih letjelice mogu razmjenjivati informacije.

Ključne riječi: bespilotna letjelica, OMNeT++, računalom upravljane okoline (CPS), simulacija

Cyber-physical systems modelling by OMNET++ tool

ABSTRACT

Cyber-physical systems (CPS) is a new generation of systems with integrated computational and physical capabilities that can interact with humans through many new modalities. OMNeT++ is a C++-based discrete event simulator for modeling communication networks, multiprocessors and distributed or parallel systems. OMNeT++ is public-source. The goal for developing OMNeT++ was to produce a powerful open-source discrete event simulation tool that can be used by academic, educational and research-oriented commercial institutions for the simulation of computer networks and distributed or parallel systems. In this paper a simulation of drones is shown in OMNeT++ in the area of city of Osijek. Their application is presented in five scenarios. Goal of simulation is to present their movements and overlapping of their wireless signal, through which the drones can exchange the information.

Keywords: drone, OMNeT++, cyber-physical systems (CPS), simulation

ŽIVOTOPIS

Ivana Majdandžić (djevojački Marinić) je rođena 21. srpnja 1987. godine u Osijeku. Godine 1994. upisuje Osnovnu školu Jagode Truhelka u Osijeku, te završava Osnovnu školu Franje Krežme s odličnim uspjehom. Godine 2002. upisuje III. gimnaziju (prirodoslovno-matematička) koju završava 2006. godine s vrlo dobrom uspjehom. Nakon mature 2006. godine, upisuje preddiplomski studij računarstva na elektrotehničkom fakultetu u Osijeku polaganjem razredbenog ispita te je u 10 posto najuspješnijih. Godine 2009. završava preddiplomski studij računarstva, i upisuje diplomski studij procesnog računarstva. Udala se 2013. godine, i 2015. godine rodila curicu Doru.

PRILOZI

P.4.2. KmlHttpServer.cc

```
#include "SocketRTScheduler.h"
#include "KmlHttpServer.h"

Define_Module(KmlHttpServer);

KmlHttpServer *KmlHttpServer::instance = NULL;

KmlHttpServer::KmlHttpServer()
{
   rtEvent = NULL;
   if (instance)
      error("There can be only one KmlHttpServer instance in the network");
   instance = this;
}

KmlHttpServer::~KmlHttpServer()
{
   cancelAndDelete(rtEvent);
   instance = NULL;
}

void KmlHttpServer::initialize()
{
   rtEvent = new cMessage("rtEvent");
   rtScheduler = check_and_cast<cSocketRTScheduler *>(simulation.getScheduler());
   rtScheduler->setInterfaceModule(this, rtEvent, recvBuffer, 4000, &numRecvBytes);
}

KmlHttpServer *KmlHttpServer::getInstance()
{
   if (!instance)
      throw cRuntimeError("KmlHttpServer::getInstance(): there is no KmlHttpServer module in
the network");
   return instance;
}

int KmlHttpServer::findKmlFragmentProvider(IKmlFragmentProvider* p)
{
   for (int i=0; i<(int)providerList.size(); i++)
      if (providerList[i] == p)
         return i;
   return -1;
}

void KmlHttpServer::addKmlFragmentProvider(IKmlFragmentProvider* p)
{
   if (findKmlFragmentProvider(p) == -1)
      providerList.push_back(p);
}

void KmlHttpServer::removeKmlFragmentProvider(IKmlFragmentProvider* p)
{
   int k = findKmlFragmentProvider(p);
```

```
    if (k != -1)
        providerList.erase(providerList.begin()+k);
}

void KmlHttpServer::handleMessage(cMessage *msg)
{
    if (msg != rtEvent)
        error("This module does not handle messages"); // only those from the SocketRTScheduler

    handleSocketEvent();
}

void KmlHttpServer::handleSocketEvent()
{
    char *endHeader = NULL;
    for (char *s=recvBuffer; s<=recvBuffer+numRecvBytes-4; s++)
        if (*s=='\r' && *(s+1)=='\n' && *(s+2)=='\r' && *(s+3)=='\n')
            {endHeader = s+4; break;}

    if (!endHeader)
        return;
    std::string header = std::string(recvBuffer, endHeader-recvBuffer);
    //EV << header;

    if (endHeader == recvBuffer+numRecvBytes)
        numRecvBytes = 0;
    else {
        int bytesLeft = recvBuffer+numRecvBytes-endHeader;
        memmove(endHeader, recvBuffer, bytesLeft);
        numRecvBytes = bytesLeft;
    }

    std::string reply = processHttpRequest(header.c_str());

    rtScheduler->sendBytes(reply.c_str(), reply.size());
    rtScheduler->close();
}

std::string KmlHttpServer::processHttpRequest(const char *httpReqHeader)
{
    std::string header(httpReqHeader);
    std::string::size_type pos = header.find("\r\n");
    if (pos==std::string::npos)
    {
        EV << "Bad HTTP request\n";
        return std::string("HTTP/1.0 400 Bad Request\r\n");
    }

    std::string cmd(header,0,pos);
    EV << "Received: " << cmd << "\n";

    if (cmd.length()<4 || cmd.compare(0,4,"GET "))
    {
        EV << "Wrong HTTP verb, only GET is supported\n";
        return std::string("HTTP/1.0 501 Not Implemented\r\n");
    }

    pos = cmd.find(" ",4);
    std::string uri(cmd,4,pos-4);
```

```
    std::string reply = getReplyFor(uri.c_str());
    return reply;
}

const char *INDEX_HTML =
    "<html>\n"
    "<head>\n"
    "   <title>OMNeT++ Google Earth Demo</title>\n"
    "                                                                                                    <script
src=\"http://www.google.com/jsapi?key=ABQIAAAAhaIVVNK7shon8s61HoJP3hTWQ61sd-
CgFDCq8tRqXSWpKVHs8RQiqzV8RfPWPm7pTTJyU_gk6LVxAg\"></script>\n"
    "   <script type=\"text/javascript\">\n"
    "     var ge;\n"
    "     var networkLink;  // for setTimeout()\n"
    "\n"
    "     if (typeof google == \"undefined\") {\n"
    "       alert(\"Error: Google API not available, connection to google.com failed?\");\n"
    "       throw \"we're done\";\n"
    "     }\n"
    "\n"
    "     try {\n"
    "       google.load(\"earth\", \"1\");\n"
    "     } catch (e) {\n"
    "       alert(\"Error loading Google Earth: \" + e.message);\n"
    "     }\n"
    "\n"
    "     function init() {\n"
    "       google.earth.createInstance('map3d', initCB, failureCB);\n"
    "     }\n"
    "\n"
    "     function initCB(instance) {\n"
    "       ge = instance;\n"
    "       ge.getWindow().setVisibility(true);\n"
    "       ge.getNavigationControl().setVisibility(true);\n"
    "\n"
    "       // add NetworkLink to snapshot.kml\n"
    "       var link = ge.createLink('');\n"
    "       var href = window.location.href.replace(/[^/]*$/, \"snapshot.kml\");\n"
    "       link.setHref(href);\n"
    "       link.setRefreshMode(ge.REFRESH_ON_INTERVAL);\n"
    "       link.setRefreshInterval(0.1);  // reload every 0.1s, or as frequently as possible\n"
    "       networkLink = ge.createNetworkLink('');\n"
    "       networkLink.set(link, false, true); // sets link, refreshVisibility, and flyToView\n"
    "       ge.getFeatures().appendChild(networkLink);\n"
    "\n"
    "       setTimeout(\"networkLink.setFlyToView(false)\", 2000);\n"
    "     }\n"
    "\n"
    "     function failureCB(errorCode) {\n"
    "       alert(\"Google Earth reported an error: \" + errorCode);\n"
    "     }\n"
    "\n"
    "     google.setOnLoadCallback(init);\n"
    "   </script>\n"
    "\n"
    "</head>\n"
    "<body>\n"
    "   <div id=\"map3d\" style=\"height: 660px; width: 1000px;\"></div>\n"
    "</body>\n"
    "</html>\n";
```

```
std::string KmlHttpServer::getReplyFor(const char *uri)
{
   if (strcmp(uri, "/")==0 || strcmp(uri, "/index.html")==0)
   {
      return addHttpHeader(INDEX_HTML, "text/html");
   }
   if (strcmp(uri, "/snapshot.kml")==0)
   {
      std::stringstream out;
      out << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
      out << "<kml xmlns=\"http://www.opengis.net/kml/2.2\">\n";
      out << "<Document>\n";
      for (int i=0; i<(int)providerList.size(); i++)
         out << providerList[i]->getKmlFragment();
      out << "</Document>\n";
      out << "</kml>\n";

      return addHttpHeader(out.str().c_str(), "application/vnd.google-earth.kml+xml");
   }
   return "HTTP/1.0 404 Not Found\r\n\r\n";
}

std::string KmlHttpServer::addHttpHeader(const char *content, const char *mimetype)
{
   // assemble reply
   std::stringstream out;
   out << "HTTP/1.0 200 OK\r\n";
   out << "Content-Type: " << mimetype << "\r\n";
   //out << "Content-Length: " << strlen(content) << "\r\n";
   out << "\r\n";
   out << content;
   return out.str();
}
```

P.4.2. MobileNode.cc

```
#include <omnetpp.h>
#include "KmlHttpServer.h"
#include "KmlUtil.h"
#include "ChannelController.h"

class MobileNode : public cSimpleModule, public IKmlFragmentProvider, public IMobileNode
{
  protected:
    double playgroundLat,playgroundLon;
   double playgroundHeight,playgroundWidth;
   double timeStep;
   unsigned int trailLength;
   std::string color;
   std::string modelURL;
   double modelScale;
   bool showTxRange;
   double txRange;
   double speed;

      double heading;
   double x, y;
```

```
    std::vector<KmlUtil::Pt2D> path;

    double y2lat(double y) { return KmlUtil::y2lat(playgroundLat, y); }

    double x2lon(double x) { return KmlUtil::x2lon(playgroundLat, playgroundLon, x); }

  public:
    MobileNode();
    virtual ~MobileNode();

    double getX() { return x; }
    double getY() { return y; }
    double getTxRange() { return txRange; }

  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual std::string getKmlFragment();
};

Define_Module(MobileNode);

MobileNode::MobileNode()
{
}

MobileNode::~MobileNode()
{
}

void MobileNode::initialize()
{
    x = par("startX");
    y = par("startY");
    heading = 360*dblrand();
    timeStep = par("timeStep");

    getDisplayString().setTagArg("p", 0, x);
    getDisplayString().setTagArg("p", 1, y);

    playgroundLat = simulation.getSystemModule()->par("playgroundLatitude");
    playgroundLon = simulation.getSystemModule()->par("playgroundLongitude");
    playgroundHeight = simulation.getSystemModule()->par("playgroundHeight");
    playgroundWidth = simulation.getSystemModule()->par("playgroundWidth");

    trailLength = par("trailLength");
    modelURL = par("modelURL").stringValue();
    modelScale = par("modelScale");
    showTxRange = par("showTxRange");
    speed = par("speed");
    txRange = par("txRange");

    color = par("color").stringValue();
    if (color.empty())
    {
        char buf[16];
        double red,green,blue;
        KmlUtil::hsbToRgb(dblrand(), 1.0, 1.0, red, green, blue);
        sprintf(buf, "%2.2x%2.2x%2.2x", int(blue*255), int(green*255), int(red*255));
```

```
        color = buf;
    }

    KmlHttpServer::getInstance()->addKmlFragmentProvider(this);
    ChannelController::getInstance()->addMobileNode(this);

    cMessage *timer = new cMessage("move");
    scheduleAt(simTime(), timer);
}

void MobileNode::handleMessage(cMessage *msg)
{
    heading += 120*(dblrand()-0.5) * timeStep;
    heading = fmod(heading + 360, 360);

    double vx = sin(heading*M_PI/180) * speed;
    double vy = -cos(heading*M_PI/180) * speed;
    x += vx*timeStep;
    y += vy*timeStep;

    if (x < 0) {x=0; heading = 360 - heading; }
    if (x > playgroundWidth) {x=playgroundWidth; heading = 360-heading;}
    if (y < 0) {y=0; heading = 180 - heading;}
    if (y > playgroundHeight) {y=playgroundHeight; heading = 180 - heading;}

    if (trailLength > 0)
        path.push_back(KmlUtil::Pt2D(x2lon(x),y2lat(y)));

    if (path.size () > trailLength)
        path.erase(path.begin());

    getDisplayString().setTagArg("p", 0, x);
    getDisplayString().setTagArg("p", 1, y);

    scheduleAt(simTime()+timeStep, msg);
}

std::string MobileNode::getKmlFragment()
{
    double longitude = x2lon(x);
    double latitude = y2lat(y);
    char buf[16];
    sprintf(buf, "%d", getIndex());
    std::string fragment;
    fragment += KmlUtil::folderHeader((std::string("folder_")+buf).c_str(), getFullName());

#ifdef USE_TRACK
    fragment += KmlUtil::track((std::string("track_")+buf).c_str(), path, timeStep, modelScale,
modelURL.c_str(), "movement trail", NULL, (std::string("ff")+color).c_str());
#else
    fragment += KmlUtil::placemark((std::string("placemark_")+buf).c_str(), longitude, latitude,
2*modelScale, getFullName(), NULL);
    if (trailLength > 0)
        fragment += KmlUtil::lineString((std::string("trail_")+buf).c_str(), path, "movement trail",
NULL, (std::string("ff")+color).c_str());
    if (showTxRange)
        fragment += KmlUtil::disk((std::string("disk_")+buf).c_str(), longitude, latitude, txRange,
"transmission range", NULL, (std::string("40")+color).c_str());
    if (!modelURL.empty()) {
        double modelheading = fmod((360 + 90 + heading), 360);
```

```
        fragment += KmlUtil::model((std::string("model_")+buf).c_str(), longitude, latitude,
modelheading, modelScale, modelURL.c_str(), "3D model", NULL);
    }
#endif

    fragment += "</Folder>\n";
    return fragment;
}
```

P.4.2. ChannelController.cc

```
#include "ChannelController.h"

Define_Module(ChannelController);

ChannelController *ChannelController::instance = NULL;

ChannelController::ChannelController()
{
    if (instance)
        error("There can be only one ChannelController instance in the network");
    instance = this;
}

ChannelController::~ChannelController()
{
    instance = NULL;
}

ChannelController *ChannelController::getInstance()
{
    if (!instance)
        throw cRuntimeError("ChannelController::getInstance(): there is no ChannelController
module in the network");
    return instance;
}

int ChannelController::findMobileNode(IMobileNode* p)
{
    for (int i=0; i<(int)nodeList.size(); i++)
        if (nodeList[i] == p)
            return i;
    return -1;
}

void ChannelController::addMobileNode(IMobileNode* p)
{
    if (findMobileNode(p) == -1)
        nodeList.push_back(p);
}

void ChannelController::removeMobileNode(IMobileNode* p)
{
    int k = findMobileNode(p);
    if (k != -1)
        nodeList.erase(nodeList.begin()+k);
}
```

```
void ChannelController::initialize()
{
   playgroundLat = simulation.getSystemModule()->par("playgroundLatitude");
   playgroundLon = simulation.getSystemModule()->par("playgroundLongitude");
   KmlHttpServer::getInstance()->addKmlFragmentProvider(this);
}

void ChannelController::handleMessage(cMessage *msg)
{
   error("This module does not process messages");
}

std::string ChannelController::getKmlFragment()
{
   std::vector<KmlUtil::Pt2D> connections;

   std::string fragment;

   for (int i=0; i<(int)nodeList.size(); ++i)
   {
      for (int j=i+1; j<(int)nodeList.size(); ++j)
      {
         IMobileNode *pi = nodeList[i];
         IMobileNode *pj = nodeList[j];
         double ix = pi->getX(), iy = pi->getY(), jx = pj->getX(), jy = pj->getY();
         if (pi->getTxRange()*pi->getTxRange() > (ix-jx)*(ix-jx)+(iy-jy)*(iy-jy)) {
            double ilat = KmlUtil::y2lat(playgroundLat, iy);
            double ilon = KmlUtil::x2lon(ilat, playgroundLon, ix);
            connections.push_back(KmlUtil::Pt2D(ilon, ilat));

            double jlat = KmlUtil::y2lat(playgroundLat, jy);
            double jlon = KmlUtil::x2lon(jlat, playgroundLon, jx);
            connections.push_back(KmlUtil::Pt2D(jlon, jlat));
         }
      }
   }
   fragment += KmlUtil::lines("connectivity_1", connections, "connectivity graph", NULL,
"60FFFFFF");

   return fragment;
}
```

P.4.2. SocketRTScheduler.cc

```
#include "SocketRTScheduler.h"

#ifndef _WIN32
#define closesocket(x) ::close(x)
#endif


Register_Class(cSocketRTScheduler);

Register_GlobalConfigOption(CFGID_SOCKETRTSCHEDULER_PORT, "socketrtscheduler-
port", CFG_INT, "4242", "When cSocketRTScheduler is selected as scheduler class: the port
number cSocketRTScheduler listens on.");

inline std::ostream& operator<<(std::ostream& out, const timeval& tv)
```

```
{
    return out << (unsigned long)tv.tv_sec << "s" << tv.tv_usec << "us";
}


cSocketRTScheduler::cSocketRTScheduler() : cScheduler()
{
    listenerSocket = INVALID_SOCKET;
    connSocket = INVALID_SOCKET;
}

cSocketRTScheduler::~cSocketRTScheduler()
{
}

void cSocketRTScheduler::startRun()
{
    if (initsocketlibonce()!=0)
        throw cRuntimeError("cSocketRTScheduler: Cannot initialize socket library");

    gettimeofday(&baseTime, NULL);

    module = NULL;
    notificationMsg = NULL;
    recvBuffer = NULL;
    recvBufferSize = 0;
    numBytesPtr = NULL;

    port = ev.getConfig()->getAsInt(CFGID_SOCKETRTSCHEDULER_PORT);
    setupListener();
}

void cSocketRTScheduler::setupListener()
{
    listenerSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (listenerSocket==INVALID_SOCKET)
        throw cRuntimeError("cSocketRTScheduler: cannot create socket");

    sockaddr_in sinInterface;
    sinInterface.sin_family = AF_INET;
    sinInterface.sin_addr.s_addr = INADDR_ANY;
    sinInterface.sin_port = htons(port);
    if (bind(listenerSocket, (sockaddr*)&sinInterface, sizeof(sockaddr_in))==SOCKET_ERROR)
        throw cRuntimeError("cSocketRTScheduler: socket bind() failed");

    listen(listenerSocket, SOMAXCONN);
}

void cSocketRTScheduler::endRun()
{
}

void cSocketRTScheduler::executionResumed()
{
    gettimeofday(&baseTime, NULL);
    baseTime = timeval_substract(baseTime, SIMTIME_DBL(simTime()));
}

void cSocketRTScheduler::setInterfaceModule(cModule *mod, cMessage *notifMsg, char *buf, int
bufSize, int *nBytesPtr)
```

```
{
  if (module)
    throw cRuntimeError("cSocketRTScheduler: setInterfaceModule() already called");
  if (!mod || !notifMsg || !buf || !bufSize || !nBytesPtr)
    throw cRuntimeError("cSocketRTScheduler: setInterfaceModule(): arguments must be non-
NULL");

  module = mod;
  notificationMsg = notifMsg;
  recvBuffer = buf;
  recvBufferSize = bufSize;
  numBytesPtr = nBytesPtr;
  *numBytesPtr = 0;
}

bool cSocketRTScheduler::receiveWithTimeout(long usec)
{
  fd_set readFDs, writeFDs, exceptFDs;
  FD_ZERO(&readFDs);
  FD_ZERO(&writeFDs);
  FD_ZERO(&exceptFDs);

  if (connSocket!=INVALID_SOCKET)
    FD_SET(connSocket, &readFDs);
  else
    FD_SET(listenerSocket, &readFDs);

  timeval timeout;
  timeout.tv_sec = 0;
  timeout.tv_usec = usec;

  if (select(FD_SETSIZE, &readFDs, &writeFDs, &exceptFDs, &timeout) > 0)
  {
    if (connSocket!=INVALID_SOCKET && FD_ISSET(connSocket, &readFDs))
    {
      char *bufPtr = recvBuffer + (*numBytesPtr);
      int bufLeft = recvBufferSize - (*numBytesPtr);
      if (bufLeft<=0)
        throw cRuntimeError("cSocketRTScheduler: interface module's recvBuffer is full");
      int nBytes = recv(connSocket, bufPtr, bufLeft, 0);
      if (nBytes==SOCKET_ERROR)
      {
        EV << "cSocketRTScheduler: socket error " << sock_errno() << "\n";
        closesocket(connSocket);
        connSocket = INVALID_SOCKET;
      }
      else if (nBytes == 0)
      {
        EV << "cSocketRTScheduler: socket closed by the client\n";
        closesocket(connSocket);
        connSocket = INVALID_SOCKET;
      }
      else
      {
        EV << "cSocketRTScheduler: received " << nBytes << " bytes\n";
        (*numBytesPtr) += nBytes;

        timeval curTime;
        gettimeofday(&curTime, NULL);
        curTime = timeval_substract(curTime, baseTime);
```

```
            simtime_t t = curTime.tv_sec + curTime.tv_usec*1e-6;
            notificationMsg->setArrival(module,-1,t);
            simulation.msgQueue.insert(notificationMsg);
            return true;
          }
        }
        else if (FD_ISSET(listenerSocket, &readFDs))
        {
          sockaddr_in sinRemote;
          int addrSize = sizeof(sinRemote);
          connSocket = accept(listenerSocket, (sockaddr*)&sinRemote, (socklen_t*)&addrSize);
          if (connSocket==INVALID_SOCKET)
            throw cRuntimeError("cSocketRTScheduler: accept() failed");
          EV << "cSocketRTScheduler: connected!\n";
        }
    }
    return false;
}

int cSocketRTScheduler::receiveUntil(const timeval& targetTime)
{
    timeval curTime;
    gettimeofday(&curTime, NULL);
    while (targetTime.tv_sec-curTime.tv_sec >=2 ||
           timeval_diff_usec(targetTime, curTime) >= 200000)
    {
        if (receiveWithTimeout(100000))
            return 1;
        if (ev.idle())
            return -1;
        gettimeofday(&curTime, NULL);
    }

    long usec = timeval_diff_usec(targetTime, curTime);
    if (usec>0)
        if (receiveWithTimeout(usec))
            return 1;
    return 0;
}

cMessage *cSocketRTScheduler::getNextEvent()
{
    if (!module)
        throw cRuntimeError("cSocketRTScheduler: setInterfaceModule() not called: it must be called from a module's initialize() function");

    timeval targetTime;
    cMessage *msg = sim->msgQueue.peekFirst();
    if (!msg)
    {
        targetTime.tv_sec = LONG_MAX;
        targetTime.tv_usec = 0;
    }
    else
    {
        simtime_t eventSimtime = msg->getArrivalTime();
        targetTime = timeval_add(baseTime, SIMTIME_DBL(eventSimtime));
    }

    timeval curTime;
```

```
    gettimeofday(&curTime, NULL);
    if (timeval_greater(targetTime, curTime))
    {
        int status = receiveUntil(targetTime);
        if (status == -1)
            return NULL;
        if (status == 1)
            msg = sim->msgQueue.peekFirst();
    }
    else
    {
    }

    return msg;
}

void cSocketRTScheduler::sendBytes(const char *buf, size_t numBytes)
{
    if (connSocket==INVALID_SOCKET)
        throw cRuntimeError("cSocketRTScheduler: sendBytes(): no connection");

    send(connSocket, buf, numBytes, 0);
}

void cSocketRTScheduler::close()
{
    if (connSocket!=INVALID_SOCKET)
    {
        closesocket(connSocket);
        connSocket = INVALID_SOCKET;
    }
}
```

P.4.2. KmlUtil.cc

```
#include "KmlUtil.h"


std::string KmlUtil::folderHeader(const char *id, const char *name, const char *description)
{
    std::stringstream out;
    out.precision(8);
    out << "<Folder";
    if (id)
        out << " id='" << id << "' ";
    out << ">\n";

    if (name)
        out << "  <name>" << name << "</name>\n";
    if (description)
        out << "  <description>" << description << "</description>\n";
    return out.str();
}

std::string KmlUtil::placemark(const char *id, float lon, float lat,float alt, const char *name,
const char *description)
{
    std::stringstream out;
```

```
    out.precision(8);
    out << "<Placemark";
    if (id)
        out << " id='" << id << "' ";
    out << ">\n";
    if (name)
        out << "  <name>" << name << "</name>\n";
    if (description)
        out << "  <description>" << description << "</description>\n";
    out << "  <Style>\n";
    out << "    <IconStyle>\n";
    out << "      <Icon>\n";
    out << "        <href>http://maps.google.com/mapfiles/kml/pal4/icon28.png</href>\n";
    out << "      </Icon>\n";
    out << "    </IconStyle>\n";
    out << "  </Style>\n";
    out << "  <Point>\n";
    out << "    <altitudeMode>relativeToGround</altitudeMode>\n";
    out << "    <coordinates>" << lon << "," << lat << "," << alt << "</coordinates>\n";
    out << "  </Point>\n";
    out << "</Placemark>\n";
    return out.str();
}

std::string KmlUtil::lines(const char *id, const std::vector<KmlUtil::Pt2D>& pts, const char
*name, const char *description, const char *color)
{
    std::stringstream out;
    out.precision(8);
    out << "<Placemark";
    if (id)
        out << " id='" << id << "' ";
    out << ">\n";
    if (color) {
        out << "  <Style>\n";
        out << "    <LineStyle>\n";
        out << "      <color>" << color << "</color>\n";
        out << "      <width>" << 5 << "</width>\n";
        out << "    </LineStyle>\n";
        out << "  </Style>\n";
    }
    if (name)
        out << "  <name>" << name << "</name>\n";
    if (description)
        out << "  <description>" << description << "</description>\n";
    out << "  <MultiGeometry>\n";
    for (int i=0; i<(int)pts.size()-1; i+=2)
    {
        out << "    <LineString>\n";
        out << "      <extrude>false</extrude>\n";
        out << "      <tesselate>true</tesselate>\n";
        out << "      <coordinates>";
        out << pts[i].lon << "," << pts[i].lat << " " << pts[i+1].lon << "," << pts[i+1].lat;
        out << "</coordinates>\n";
        out << "    </LineString>\n";
    }
    out << "  </MultiGeometry>\n";
    out << "</Placemark>\n";
    return out.str();
}
```

```
std::string KmlUtil::lineString(const char *id, const std::vector<KmlUtil::Pt2D>& pts, const char *name, const char *description, const char *color)
{
   std::stringstream out;
   out.precision(8);
   out << "<Placemark";
   if (id)
      out << " id='" << id << "' ";
   out << ">\n";
   if (color) {
      out << "  <Style>\n";
      out << "    <LineStyle>\n";
      out << "      <color>" << color << "</color>\n";
      out << "    </LineStyle>\n";
      out << "  </Style>\n";
   }
   if (name)
      out << "  <name>" << name << "</name>\n";
   if (description)
      out << "  <description>" << description << "</description>\n";
   out << "  <LineString>\n";
   out << "    <extrude>false</extrude>\n";
   out << "    <tesselate>true</tesselate>\n";
   out << "    <coordinates>";
   for (int i=0; i<(int)pts.size(); i++)
      out << pts[i].lon << "," << pts[i].lat << " ";
   out << "</coordinates>\n";
   out << "  </LineString>\n";
   out << "</Placemark>\n";
   return out.str();
}

std::string KmlUtil::disk(const char *id, float lon, float lat, float r, const char *name, const char *description, const char *color)
{
   static const int N = 40;
   static float x[N], y[N];
   static bool initialized = false;
   if (!initialized) {
      for (int i=0; i<N; i++) {
         x[i] = cos(i*M_PI*2/N);
         y[i] = sin(i*M_PI*2/N);
      }
      initialized = true;
   }

   std::stringstream out;
   out.precision(8);
   out << "<Placemark";
   if (id)
      out << " id='" << id << "' ";
   out << ">\n";
   if (color) {
      out << "  <Style>\n";
      out << "    <LineStyle>\n";
      out << "      <width>0</width>\n";
      out << "    </LineStyle>\n";
      out << "    <PolyStyle>\n";
      out << "      <color>" << color << "</color>\n";
```

```
    out << "    </PolyStyle>\n";
    out << "  </Style>\n";
  }
  if (name)
    out << "  <name>" << name << "</name>\n";
  if (description)
    out << "  <description>" << description << "</description>\n";
  out << "  <Polygon>\n";
  out << "    <extrude>false</extrude>\n";
  out << "    <outerBoundaryIs>\n";
  out << "      <LinearRing>\n";
  out << "        <coordinates>\n";
  for (int i=0; i<N; i++) {
    double lati = y2lat(lat, y[i]*r);
    double loni = x2lon(lati, lon, x[i]*r);
    out << loni << "," << lati << " ";
  }
  out << "        </coordinates>\n";
  out << "      </LinearRing>\n";
  out << "    </outerBoundaryIs>\n";
  out << "  </Polygon>\n";
  out << "</Placemark>\n";
  return out.str();
}

std::string KmlUtil::model(const char *id, float lon, float lat, float heading, float scale, const char
*link, const char *name, const char *description)
{
  std::stringstream out;
  out.precision(8);
  out << "<Placemark";
  if (id)
    out << " id='" << id << "' ";
  out << ">\n";
  if (name)
    out << "  <name>" << name << "</name>\n";
  if (description)
    out << "  <description>" << description << "</description>\n";

  out << "  <Model>\n";
  out << "    <altitudeMode>relativeToGround</altitudeMode>\n";
  out << "    <Location>\n";
  out << "      <longitude>" << lon << "</longitude>\n";
  out << "      <latitude>" << lat << "</latitude>\n";
  out << "    </Location>\n";
  out << "    <Orientation>\n";
  out << "      <heading>" << heading << "</heading>\n";
  out << "    </Orientation>\n";
  out << "    <Scale>\n";
  out << "      <x>" << scale << "</x>\n";
  out << "      <y>" << scale << "</y>\n";
  out << "      <z>" << scale << "</z>\n";
  out << "    </Scale>\n";
  out << "    <Link>\n";
  out << "      <href>" << link << "</href>\n";
  out << "    </Link>\n";
  out << "  </Model>\n";

  out << "</Placemark>\n";
  return out.str();
```

```
}

std::string KmlUtil::track(const char *id, const std::vector<Pt2D>& pts, float timeStep, float modelScale, const char *modelLink, const char *name, const char *description, const char *color)
{
   std::stringstream out;
   out.precision(8);

   out << "<Placemark";
   if (id)
      out << " id='" << id << "' ";
   out << ">\n";
   if (color) {
      out << "  <Style>\n";
      out << "    <LineStyle>\n";
      out << "      <color>" << color << "</color>\n";
      out << "      <width>5</width>\n";
      out << "    </LineStyle>\n";
      out << "  </Style>\n";
   }
   if (name)
      out << "  <name>" << name << "</name>\n";
   if (description)
      out << "  <description>" << description << "</description>\n";

   out << "  <gx:Track>\n";
   double t = SIMTIME_DBL(simTime() - (pts.size()-1)*timeStep);
   for (int i=0; i<(int)pts.size(); i++, t+= timeStep) {
      int x = (int)t;
      int ss = x % 60; x-= ss; x/=60;
      int mm = x % 60; x-= mm; x/=60;
      int hh = x;
      char when[100];
      sprintf(when, "2010-05-28T%2.2d:%2.2d:%2.2dZ", hh, mm, ss);
      out << "    <when>" << when << "</when>\n";
   }
   for (int i=0; i<(int)pts.size(); i++)
      out << "    <gx:coord>" << pts[i].lon << " " << pts[i].lat << "</gx:coord>\n";

   if (modelLink) {
      out << "    <Model>\n";
      out << "    <Orientation>\n";
      out << "      <heading>" << 0 << "</heading>\n";
      out << "    </Orientation>\n";
      out << "    <Scale>\n";
      out << "      <x>" << modelScale << "</x>\n";
      out << "      <y>" << modelScale << "</y>\n";
      out << "      <z>" << modelScale << "</z>\n";
      out << "    </Scale>\n";
      out << "    <Link>\n";
      out << "      <href>" << modelLink << "</href>\n";
      out << "    </Link>\n";
      out << "  </Model>\n";
   }
   out << "  </gx:Track>\n";

   out << "</Placemark>\n";

   return out.str();
```

```
}

void KmlUtil::hsbToRgb(double hue, double saturation, double brightness,
                       double& red, double& green, double &blue)
{
  if (brightness == 0.0)
  {
     red   = 0.0;
     green = 0.0;
     blue  = 0.0;
     return;
  }

  if (saturation == 0.0)
  {
     red   = brightness;
     green = brightness;
     blue  = brightness;
     return;
  }

  float domainOffset;
  if (hue < 1.0/6)
  {
     domainOffset = hue;
     red   = brightness;
     blue  = brightness * (1.0 - saturation);
     green = blue + (brightness - blue) * domainOffset * 6;
  }
   else if (hue < 2.0/6)
   {
     domainOffset = hue - 1.0/6;
     green = brightness;
     blue  = brightness * (1.0 - saturation);
     red   = green - (brightness - blue) * domainOffset * 6;
   }
   else if (hue < 3.0/6)
   {
     domainOffset = hue - 2.0/6;
     green = brightness;
     red   = brightness * (1.0 - saturation);
     blue  = red + (brightness - red) * domainOffset * 6;
   }
   else if (hue < 4.0/6)
   {
     domainOffset = hue - 3.0/6;
     blue  = brightness;
     red   = brightness * (1.0 - saturation);
     green = blue - (brightness - red) * domainOffset * 6;
   }
   else if (hue < 5.0/6)
   {
     domainOffset = hue - 4.0/6;
     blue  = brightness;
     green = brightness * (1.0 - saturation);
     red   = green + (brightness - green) * domainOffset * 6;
   }
   else
   {
```

```
    domainOffset = hue - 5.0/6;
    red   = brightness;
    green = brightness * (1.0 - saturation);
    blue  = red - (brightness - green) * domainOffset * 6;
  }
}
```

P.4.3.1. Scenarij 1

```
[General]
cmdenv-event-banner-details = true
cmdenv-message-trace = true
debug-statistics-recording = true
parallel-simulation = false
record-eventlog = true
scheduler-class = "cSocketRTScheduler"
#debug-on-errors = true

#cmdenv-express-mode = false
cmdenv-module-messages = true
cmdenv-event-banners = true

network = GoogleEarthNet
**.param-record-as-scalar = true
**.playgroundLatitude = 45.560
**.playgroundLongitude = 18.695
**.playgroundWidth = 0.5km
**.playgroundHeight = 0.5km
**.drone[*].trailLength = 600
**.drone[*].modelURL = "../~/Osijek Drones/dae/drone.dae"
**.drone[*].txRange = 100m
**.drone[*].modelScale = 0.5
**.drone[*].speed = 10mps
**.drone[*].startX = uniform(0.1km, 0.4km)
**.drone[*].startY = uniform(0.1km, 0.4km)
```

P.4.3.2. Scenarij 2

```
[General]
cmdenv-event-banner-details = true
cmdenv-message-trace = true
debug-statistics-recording = true
parallel-simulation = false
record-eventlog = true
scheduler-class = "cSocketRTScheduler"
#debug-on-errors = true

#cmdenv-express-mode = false
cmdenv-module-messages = true
cmdenv-event-banners = true

network = GoogleEarthNet
**.param-record-as-scalar = true
**.playgroundLatitude = 45.570407
**.playgroundLongitude = 18.665024
**.playgroundWidth = 1.0km
**.playgroundHeight = 0.02km
**.drone[*].trailLength = 1500
```

```
**.drone[*].modelURL = "../~/Osijek Drones 1/dae/drone.dae"
**.drone[*].txRange = 300m
**.drone[*].modelScale = 0.5
**.drone[*].speed = 12mps
**.drone[*].startX = uniform(0.1km, 0.9km)
**.drone[*].startY = uniform(0.1km, 0.9km)
```

P.4.3.3. Scenarij 3

```
[General]
cmdenv-event-banner-details = true
cmdenv-message-trace = true
debug-statistics-recording = true
parallel-simulation = false
record-eventlog = true
scheduler-class = "cSocketRTScheduler"
#debug-on-errors = true

#cmdenv-express-mode = false
cmdenv-module-messages = true
cmdenv-event-banners = true

network = GoogleEarthNet
**.param-record-as-scalar = true
**.playgroundLatitude = 45.551611
**.playgroundLongitude = 18.696703
**.playgroundWidth = 0.7km
**.playgroundHeight = 0.04km
**.drone[*].trailLength = 800
**.drone[*].modelURL = "../~/Osijek Drones 2/dae/drone.dae"
**.drone[*].txRange = 80m
**.drone[*].modelScale = 0.5
**.drone[*].speed = 15mps
**.drone[*].startX = uniform(0.1km, 0.5km)
**.drone[*].startY = uniform(0.1km, 0.5km)
```

P.4.3.4. Scenarij 4

```
[General]
cmdenv-event-banner-details = true
cmdenv-message-trace = true
debug-statistics-recording = true
parallel-simulation = false
record-eventlog = true
scheduler-class = "cSocketRTScheduler"
#debug-on-errors = true

#cmdenv-express-mode = false
cmdenv-module-messages = true
cmdenv-event-banners = true

network = GoogleEarthNet
**.param-record-as-scalar = true
**.playgroundLatitude = 45.555165
**.playgroundLongitude = 18.637661
**.playgroundWidth = 0.2km
```

```
**.playgroundHeight = 0.03km
**.drone[*].trailLength = 600
**.drone[*].modelURL = "../~/Osijek Drones 3/dae/drone.dae"
**.drone[*].txRange = 30m
**.drone[*].modelScale = 0.5
**.drone[*].speed = 20mps
**.drone[*].startX = uniform(0.1km, 0.2km)
**.drone[*].startY = uniform(0.1km, 0.2km)
```

P.4.3.5. Scenarij 5

```
[General]
cmdenv-event-banner-details = true
cmdenv-message-trace = true
debug-statistics-recording = true
parallel-simulation = false
record-eventlog = true
scheduler-class = "cSocketRTScheduler"
#debug-on-errors = true

#cmdenv-express-mode = false
cmdenv-module-messages = true
cmdenv-event-banners = true

network = GoogleEarthNet
**.param-record-as-scalar = true
**.playgroundLatitude = 45.545078
**.playgroundLongitude = 18.695777
**.playgroundWidth = 0.3km
**.playgroundHeight = 0.02km
**.drone[*].trailLength = 800
**.drone[*].modelURL = "../~/Osijek Drones 4/dae/drone.dae"
**.drone[*].txRange = 50m
**.drone[*].modelScale = 0.5
**.drone[*].speed = 10mps
**.drone[*].startX = uniform(0.1km, 0.2km)
**.drone[*].startY = uniform(0.1km, 0.2km)
```