Integrasjon av nettverkssimulatoren OMNeT++ med HLA

Martin Normann Nielsen

Forsvarets forskningsinstitutt

31. mars 2011

FFI-rapport 2010/01290

1140

P: ISBN 978-82-464-1933-6
E: ISBN 978-82-464-1934-3

Emneord

Distribuert simulering

Modellering og simulering

Nettverkssimulering

Godkjent av

Karsten Bråthen	Prosjektleder
Eli Winjum	Forskningssjef
Vidar S. Andersen	Avdelingssjef

Sammendrag

Denne rapporten beskriver arbeidet med å integrere en diskret hendelsessimulator, OMNeT++, i distribuerte simuleringsføderasjoner basert på High Level Architecture (HLA). Arbeidet har hatt som mål å samle simuleringsansvar og simuleringsfunksjonalitet for simulering av datanettverk i en dedikert simulator (en nettverkssimulator). Andre simuleringer deltakende i simuleringsføderasjoner er ment å benytte seg av nettverkssimulatorens tjenester for simulering av overføring av all tale og meldingstrafikk mellom simulerte entiteter.

English summary

This report describes the work of integrating a discrete event simulator (OMNeT++) into distributed simulations based on the High Level Architecture. The aim of the work has been to gather the responsibility and functionality of simulating the effects of communication systems on the message passing between entities into a dedicated simulator (a network simulator). Simulations partaking in federated simulations are meant to utilize the services of the network simulator for simulation of all message and voice communications transfer between simulated entities.

Innhold

1	**Innledning**	**7**
2	**Bakgrunn**	**7**
2.1	Målsetning	7
2.2	Distribuert simulering	9
2.3	Objective Modular discrete event Network simulator in C++ (OMNeT++)	10
3	**Modeller av datakommunikasjon**	**10**
3.1	OSI-modellen	11
3.2	Kommunikasjonsmodellen i RPR FOM	12
4	**Kommunikasjonsmodell for å benytte OMNeT++ som tjeneste i distribuerte simuleringer**	**19**
4.1	Grensesnittet mellom kommunikasjonsmodellene til entitetsbaserte simuleringer og OMNeT++	19
4.2	Simuleringsmodellen i OMNeT++	26
4.3	Modellers påvirkning av ende-til-ende-kommunikasjon	28
5	**Programvaredokumentasjon**	**30**
5.1	Grensesnitt for distribuert simulering	30
5.1.1	Integrasjon av OMNeT++ (C++) med HLA (Java)	30
5.1.2	Eksekvering av modellimplementasjon	33
5.1.3	Grensesnitt mot OMNeT++-modeller	34
5.1.4	Simulering av radioer	35
5.1.5	Objektmodell og støttede RTI-tjenester	36
6	**Testføderasjon og resultater**	**39**
6.1	Tilpasning av eksisterende modell	40
6.2	Erfaringer	41
6.3	Begrensninger	42
7	**Diskusjon**	**42**
8	**Videre arbeid**	**43**

1 Innledning

Denne rapporten beskriver arbeidet med å integrere en diskret hendelsessimulator, Objective Modular discrete event Network simulator in C++ (OMNeT++) [1], i distribuerte simuleringsføderasjoner basert på High Level Architecture (HLA) [2]. Arbeidet har som mål å samle simuleringsansvar og simuleringsfunksjonalitet for simulering av datanettverk i distribuerte simuleringer i en dedikert simulator (en nettverkssimulator). Andre simuleringer deltakende i simuleringsføderasjoner er ment å benytte seg av nettverkssimulatorens tjenester for simulering av overføring av all tale og meldingstrafikk mellom simulerte entiteter. Arbeidet har bestått i å implementere et HLA-grensesnitt for OMNeT++, samt å designe og å implementere et hensiktsmessig grensesnitt mellom entitetsbaserte simuleringer og nettverkssimulatoren.

Denne rapporten gir en nærmere beskrivelse av bakgrunnen for arbeidet, måten andre simuleringer er tenkt å benytte seg av nettverkssimulatoren og nytteverdien av dette. Rapporten inneholder også dokumentasjon for de HLA-baserte utvidelsene til OMNeT++. Deler av arbeidet er også dokumentert i [3].

I kapittel to beskrives bakgrunnen og målsetningen for arbeidet og OMNeT++ introduseres. Kapittel tre beskriver eksisterende kommunikasjonsmodeller. Kapittel fire beskriver kommunikasjonsmodellen utviklet for å benytte OMNeT++ som tjeneste i distribuerte simuleringer, kapittel fem beskriver implementasjon av kommunikasjonsmodellen, mens kapittel seks, syv og åtte omhandler erfaringer med bruk av kommunikasjonsmodellen og forslag til videre arbeid.

2 Bakgrunn

Kommunikasjonsteknologi muliggjør nye operative konsepter, som f.eks. nettverksbasert forsvar (NbF), men teknologien har også begrensninger som det er viktig å ta hensyn til. I hvilken grad en kan øke deling av informasjon og kunnskap, oppnå bedre og enklere samarbeid, samt i større grad nyttegjøre seg av distribuerte og virtuelle organisasjoner, som er noen av mulighetene som kan oppnås med NbF, avhenger av både organisatoriske og tekniske aspekter. En virtuell teknologidemonstrator (VTD) skal kunne støtte konseptutvikling og eksperimentering (CD&E) med både teknologi og organisasjon, men en god VTD avhenger i stor grad av en hensiktsmessig representasjon av kommunikasjonsteknologiene som benyttes. Eksempler på relevante kommunikasjonsteknologier er taktiske datalinker, andre taktiske nett, våpen-datalinker eller deler av Forsvarets digitale nett (FDN).

2.1 Målsetning

Målsetningen med arbeidet, som beskrives i denne rapporten, var å starte etablering av et rammeverk for simulering av nettverk basert på nettverkssimulatoren OMNeT++ med grensesnitt mot HLA. Dette rammeverket var ment å fungere som en tjeneste i distribuerte simuleringer for å

simulere all overføring av tale og data. OMNeT++ er et felles verktøy for kommunikasjonssimulering på FFI og verktøyet kan bidra til gjenbruk av modeller og utnyttelse av felles ressurser. Fire hovedanvendelser kan tenkes for et felles rammeverk for kommunikasjonssimulering. Disse er:

- design og evaluering av nettverksteknologier, f.eks. nettverksprotokoller
- testing og evaluering av dataprogrammer som benytter datanettverk for å kommunisere, f.eks. SOA-tjenester
- beslutningsstøtte, f.eks. i simulering av operasjoner der kommunikasjonsteknologi er en faktor
- syntetisk trening og øving på samhandling via realistiske datanettverk

Arbeid knyttet til de to første punktene gjøres i dag hovedsakelig i to trinn. Det første trinnet er å gjennomføre syntetiske tester av teknologier i en nettverkssimulator som i stor del er basert på tilfeldige inndata, deretter implementeres teknologien og tas ut i felt for å gjøre operative tester. VTD-er med kommunikasjonssimuleringsrammeverket vil også kunne benyttes mellom disse fasene og belyse problemstillinger knyttet til operativ bruk av teknologier før de tas ut i felt. Operative tester er ofte forbundet med kostnader og med manglende kontroll på faktorer som kan ha avgjørende betydning for forsøkene. Ved å gjennomføre felttestene først i en VTD vil

- alle parametere i forsøk være under kontroll og en kan gjennomføre flere tester med samme parametersett,
- inndata (enheters bevegelse og når data sendes mellom enheter) til modellen er simulerte operasjoner slik at realismen til inndataene øker,
- sannsynligheten for å lykkes i feltforsøk øker,
- antall feltforsøk kan reduseres og
- feltforsøk kan være mer avanserte på et tidligere stadium fordi teknologien er gjort mer robust ved bruk av VTD-er.

Ett delmål er å støtte simuleringer som kjører i sann tid, dvs. hovedsaklig punkt to og fire. Etablering av rammeverk for kommunikasjonssimulering ble påbegynt i løpet av FFI prosjekt 1038 – Virtuelle teknologidemonstratorer og videreført i FFI-prosjekt 1140 – Syntetisk miljø for framtidige taktiske kapasiteter. Arbeidet ble delt inn i fire områder:

1. Utvikling av en HLA-bro for OMNeT++ for å kunne ta imot og sende HLA-objekter og interaksjoner.
2. Utvikling av HLA-radiobibliotek for å lette utvikling av kommuniserende føderater og for å opparbeide erfaring med bruk av radiodelen av Real-time Platform Reference Federation Object Model (RPR-FOM).
3. Utvikling av testføderater som kan sende og motta meldinger for å teste HLA-broen og HLA-radiobiblioteket.
4. Etablering av et modellbibliotek for å kunne gjennomføre enkle simuleringer med OMNeT++.

Ambisjonen med arbeidet er tredelt. For det første er det et ønske å skape en komponent som sikrer rettferdig strid innen datakommunikasjon på tvers av simuleringer. Med rettferdig strid menes at en simulering ikke oppnår fordeler ved å modellere verden slik at dennes simulerte enheter oppnår fordeler. Eksempler på slike fordeler er radarer som ikke tar hensynt til terreng, våpen med ubegrenset rekkevidde og feilfrie kommunikasjonslinjer med uendelig kapasitet. For det andre er det en ambisjon å samle utvikling av kommunikasjonsmodeller i ett verktøy. Dette gir fordeler med hensyn på muligheter for å utvikle komplekse kommunikasjonsmodeller ved hjelp av egnet verktøystøtte, på gjenbruk av modeller på tvers av simuleringer og FFI-prosjekter og på samarbeid om utvikling av modeller mellom prosjekter på FFI. For det tredje er det en ambisjon at et kommunikasjonsrammeverk skal være med på å redusere kompleksiteten ved utvikling av simuleringer som krever simulering av datakommunikasjon. Ved å tilby utviklere av for eksempel plattformsimuleringer et enkelt grensesnitt mot kommunikasjonsrammeverket trenger ikke disse utvikle modeller for kommunikasjon og kan dermed benytte mer av tiden på å utvikle hovedsimuleringen som benytter seg av kommunikasjonssimuleringsrammeverket.

2.2 Distribuert simulering

For å kunne simulere store og komplekse systemer er det ofte nødvendig å distribuere deler av simuleringer over flere maskiner. Disse kan være spredt på forskjellige geografiske steder, likevel vil resultatet av simuleringer være det samme som om de hadde foregått på en datamaskin. Det finnes forskjellige teknologier for å knytte sammen distribuerte simuleringer. Innen militær modellering og simulering (M&S) benyttes ofte HLA.

HLA kan beskrives som en komponentbasert mellomvarearkitektur for simulering. En slik arkitektur legger til rette for oppdeling av simuleringer i mindre enheter (komponenter), og fordeling av komponenter slik at delsimuleringer kan distribueres på separate datamaskiner. En slik oppdeling legger også til rette for at simuleringer kan endres ved å variere sammensetningen av komponenter. Oppdelingen gjør også at komponentene kan modelleres og implementeres av eksperter med kompetanse innen den enkelte komponents ansvarsområde. For å forhindre problemer med kommunikasjon mellom komponenter er det behov for en definisjon av hvordan grensesnittene mellom komponentene skal se ut (HLA) og en felles forståelse av de data som utveksles.

Forståelsen av data defineres av en felles Federation Object Model (FOM). En FOM beskriver hva som simuleres og hvilke data simuleringer utveksler. For militære simuleringer vil dette ofte være enheter som fly, skip og kjøretøy, sensorer som f. eks. radar, radioutstyr, radiomeldinger osv. En FOM beskriver også hvordan data som utveksles mellom simuleringer skal se ut (i bits og bytes). M&S-prosjektene på FFI har valgt å standardisere på Real-time Platform Reference (RPR) FOM versjon 2.17d [4]. I simuleringer basert på HLA kalles komponentene for føderater. En føderasjon samler komponentene som skal inngå i en kjøring av en simulering. Hver simulering i en føderasjon definerer en Simulation Object Model (SOM) som beskriver hvilket subset av FOM-en den publiserer og/eller ønsker å motta oppdateringer for fra andre simuleringer.

Føderater kan ha sammenfallende simuleringsbehov. For eksempel vil føderater som simulerer kommuniserende enheter ha liknende funksjonalitet for simulering av datakommunikasjon. Føderater kan implementere denne funksjonaliteten selv eller benytte felles simuleringstjenester. Den sistnevnte fremgangsmåten er gjenstand for arbeidet som beskrives i denne rapporten og som implementeres ved hjelp av OMNeT++.

2.3 Objective Modular discrete event Network simulator in C++ (OMNeT++)

OMNeT++ er en objektorientert og modulær diskret hendelsessimulator. Den er bygget opp som et generelt rammeverk som støtter opp under utvikling og eksekvering av for eksempel modeller av trafikk i telekommunikasjonsnett, kommunikasjonsprotokoller, kø-nettverk, multiprosessorsystemer og andre områder som egner seg for diskret hendelsessimulering. OMNeT++ benyttes hovedsaklig for nettverkssimulering og dette gjenspeiles i flere av programvaregrensesnittene og i annen funksjonalitet.

Modeller i OMNeT++ lages ved å definere små enkle moduler med begrenset simuleringsfunksjonalitet (engelsk term ”simple module”). Disse små modulene komponeres så hierarkisk sammen og innkapsles i større moduler (engelsk term ”compound module”). Det er ingen grense for hvor mange moduler som kan komponeres sammen. Alle moduler (små og store) kommuniserer med andre moduler via klart definerte inn- og utporter med meldinger av vilkårlig kompleksitet. Dvs. en kan sende egendefinerte komplekse datastrukturer via portene. Ved hjelp av et eget modelleringsspråk kobles modulene sammen og gis startbetingelser. På denne måten skilles det klart mellom implementasjonen av modulene (skrevet i C++) og simuleringsmodellene som defineres i et eget modelleringsspråk[1].

Simulatoren er monolittisk og har ingen grensesnitt for å delta i heterogene distribuerte simuleringer med bruk av for eksempel HLA. Den kan imidlertid kjøre parallelle distribuerte simuleringer der flere instanser av OMNeT++ replikeres for Monte Carlo-simuleringer. Det er også mulig å dele opp problemer og modellere disse på egnet måte ved hjelp av modellspråket slik at deler av modellen effektivt kan spres på flere prosessorer enten internt i en datamaskin eller over flere datamaskiner. Simuleringskjernen forbinder moduler som eksekverer på separate datamaskiner. Modeller som eksekverer på forskjellige prosessorer bør ikke være sterkt avhengige av hverandre.

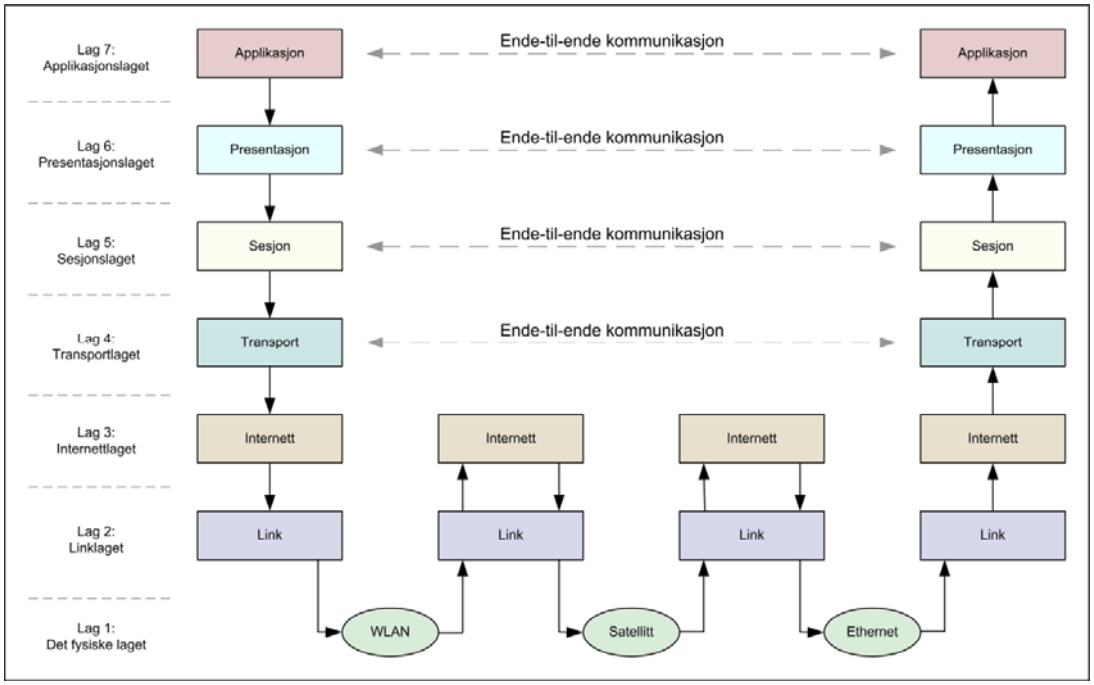
3 Modeller av datakommunikasjon

I dette kapittelet beskrives to forskjellige modeller av datakommunikasjon. Først gjennomgås Open Systems Interconnection (OSI) referansemodell for datakommunikasjon [5]. Deretter beskrives RPR FOM sin kommunikasjonsmodell og hvordan denne kan tolkes med henblikk på OSI-modellen.

[1] Modeller defineres ved hjelp av Network Description (NED) filer. Det finnes også et grafisk brukergrensesnittgrensesnitt i OMNeT++ som kan generere NED-filer.

3.1 OSI-modellen

Datakommunikasjon beskrives ofte ved hjelp av en lagdelt modell. Se Figur 3.1. En slik modell samler konseptuelt lik funksjonalitet i ett lag. Vertikalt i modellen vil et lag tilby tjenester til laget over og benytte tjenester fra laget under. Horisontalt vil lagene, fordelt på forskjellige nettverksenheter, være sammenkoblet ved hjelp av protokoller og vil på den måten kunne kommunisere og samarbeide.



Figur 3.1 ISO sin OSI referansemodell for datakommunikasjon.

OSI-modellen er den mest kjente modellen innen datakommunikasjon. OSI-modellen er lagdelt og definerer syv distinkte lag. Disse er:

- applikasjonslaget
- presentasjonslaget
- sesjonslaget
- transportlaget
- nettverkslaget
- datalinklaget
- det fysiske laget

Figur 3.1 gjengir modellen som kort beskrives i det følgende. Lag en, det fysiske laget, er ansvarlig for bl.a. etablering av forbindelse f.eks. til et radiobasert eller kobberbasert overføringsmedium og for å modulere signalet ut på mediet. Lag to, linklaget, er ansvarlig for funksjonalitet som adressering, feilkontroll og retting, og flytkontroll mellom to direkte kommuniserende noder i et nettverk. For at to noder skal kunne kommunisere indirekte på tvers

av forskjellige typer nettverk implementeres funksjonalitet for dette på lag tre. På dette laget implementeres gjerne hierarkisk adressering og gruppeadressering, fragmentering av meldinger, tjenestekvalitet for å differensiere hvilken ressurstilgang forskjellige meldinger får og rutingprotokoller for å knytte sammen nettverk av nettverk. Over dette laget ligger transportlaget som håndterer ende-til-ende pålitelighet. Sesjonslaget (lag fem) etablerer, håndterer og avslutter dialoger mellom applikasjoner, mens presentasjonslaget (lag seks) oversetter mellom syntaktiske og semantiske forskjeller i dialoger for applikasjonslaget på lag syv.

Det er viktig å merke seg at noen lag har liknende funksjonalitet. For eksempel har lag to og tre adresseringsfunksjonalitet og kan ha tjenestekvalitetsmekanismer, og lag to og fire kan ha mekanismer for retransmisjon av meldinger. Det som hovedsaklig skiller funksjonaliteten fra hverandre er rekkevidden. Fra lag fire og oppover håndteres ende-til-ende funksjonalitet, lag tre håndterer logisk adressering og dermed ruting av pakker på tvers av nettverk, mens lag en og to håndterer det fysiske mediet og adressering lokalt. OSI-modellen er en referansemodell og kan benyttes for å modellere mange forskjellige typer kommunikasjonsteknologier. Mange modeller benytter ikke alle lagene i OSI-modellen for å beskrive teknologier. Modellen for internettbasert datakommunikasjon er et eksempel på dette. Den slår sammen lag en og to til et felles linklag, og lag fem til syv beskrives som ett applikasjonslag.

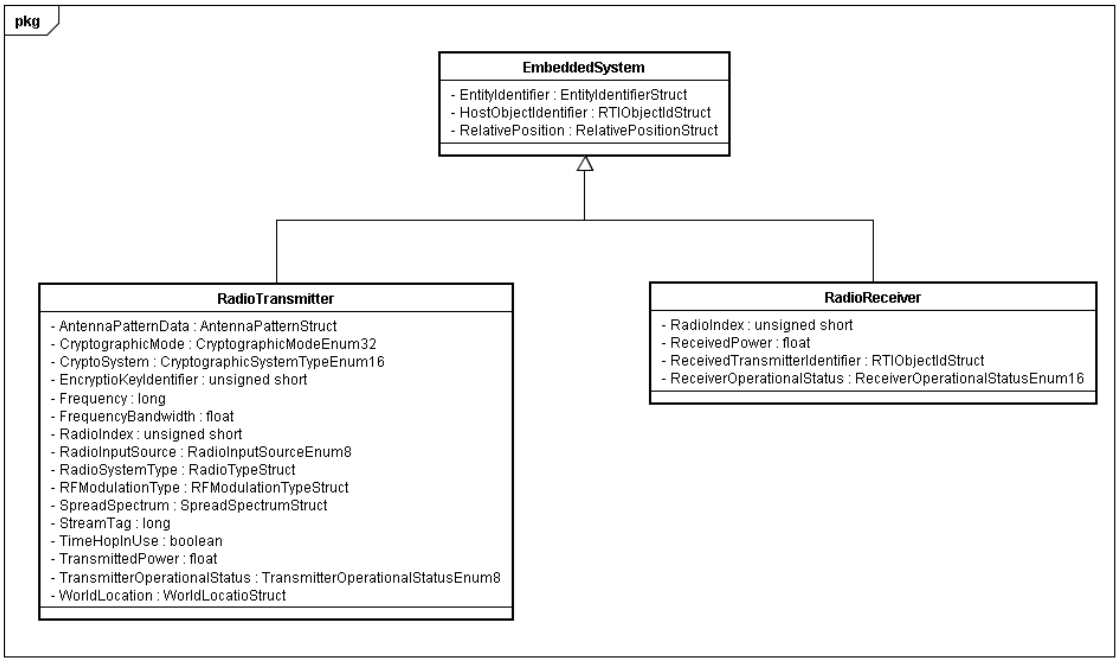
Den viktigste funksjonaliteten lag to til fire tilbyr er imidlertid adressering. Som eksempel benytter internettmodellen en tre-trinns adresseringsmodell. På det laveste nivået trenger radiosendere en måte å adressere hvilke mottakere blant de innenfor transmisjonsavstand som er tenkt å være mottakere av signaler. Til dette benyttes medium aksesskontroll (MAC-adresser) og finnes på linklaget. Dersom det er behov for å sende meldinger over flere hopp, dvs. at noen enheter videresender meldinger i retning endelig destinasjon er det behov for en globalt unik adresse i tillegg til MAC-adressen. På Internett benyttes Internett Protocol (IP-adresser) (internettlaget). Det tredje steget innen adressering er å identifisere hvilket dataprogram på en enhet meldinger er tiltenkt for. Hver enkelt enhet kan ha flere dataprogrammer eksekverende i parallell som benytter samme IP-adresse. To kjente transportlagsprotokoller, Transmission Control Protocol (TCP) og User Datagram Protocol (UDP), benytter portnummere for å skille mellom dataprogrammer. Dataprogrammer på applikasjonslaget reserverer ett eller flere portnummere og benytter disse som del av egen adresse.

Ut ifra denne lagdelte beskrivelsen tolkes og beskrives kommunikasjonsmodellen i RPR FOM i neste underkapitel.

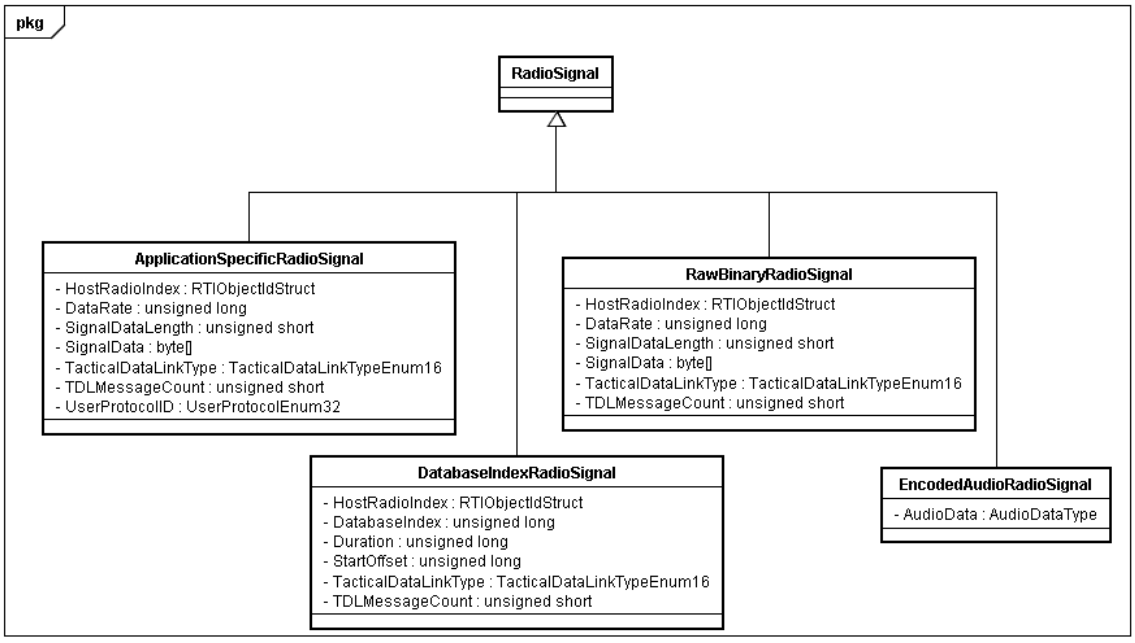
3.2 Kommunikasjonsmodellen i RPR FOM

RPR FOM modellerer entiteter og interaksjoner mellom entiteter. RPR FOM sin kommunikasjonsmodell modellerer derfor et domene fra et annet perspektiv enn OSI-modellen som modellerer datakommunikasjon spesifikt. Det er likevel mulig å beskrive mesteparten av kommunikasjonsmodellen til RPR FOM ved hjelp av OSI-modellen. RPR FOM sin modell av datakommunikasjon baserer seg på tre informasjonsklasser som representerer ulike sider ved et kommunikasjonsnettverk, henholdsvis objektklassene som representerer radiosendere og

radiomottakere (Figur 3.2) og interaksjonsklassene som representerer radiosignaler (Figur 3.3). RPR FOM definerer ingen objektmodell for transmisjon over fastnett.



Figur 3.2 UML-modell av radiosender og radiomottaker som definert i RPR FOM v2.17d.



Figur 3.3 UML-modell av radiosignaler som definert i RPR FOM v2.17d.

RadioSignal-klassen inneholder en peker som relaterer radiosignaler til radiosendere ved hjelp av *HostRadioIndex* (se Figur 3.3). Informasjon for simulering av transmisjon av signaler må derfor ekstraheres fra radiosenderobjektene. *RadioSignal*-klassen og dennes subklasser inneholder ingen

informasjon om intendert mottaker eller annen protokollinformasjon. Unntaket er klassen *ApplicationSpecificRadioSignal* som inneholder et attributt som kan henvise til en brukerdefinert protokoll slik at det er mulig å tolke datainnholdet i et signal på forskjellige egendefinerte måter.

Det finnes to måter å tolke RPR FOM sin kommunikasjonsmodell ut ifra OSI-modellen. Den første måten, som RPR FOM er designet for, er å la radiosender- og radiomottakerklassen representere lag en og deler av lag to. Lag en representeres ved attributter om f.eks. frekvensbruk og antennedata, mens lag to representeres ved attributter om kryptering og metode for kanaltilgang. *RadioSignal*-klassen representerer et signal på det fysiske laget siden ingen protokollinformasjon er definert i RPR FOM. Denne måten egner seg derfor best for kringkasting (dvs. alle-til-alle-kommunikasjon), men også for punkt-til-punkt kommunikasjon dersom det finnes kun to kommuniserende noder innenfor transmisjonsavstand i en simulering eller der det på forhånd er definert hvilke par av noder som kommuniserer. Den sistnevnte metoden benyttes i føderasjoner der VR-Forces [6] benytter QualNet [7] for simulering av datakommunikasjon.

RPR FOM foreskriver fem steg for å simulere datakommunikasjon mellom simulerte enheter. Disse er:

1. Opprette *RadioTransmitter*-objekt
2. Opprette *RadioReceiver*-objekt
3. Opprette interaksjon som er subklasse av *RadioSignal*-klassen
4. Sende radiosignalinteraksjon
5. Motta radiosignalinteraksjon

Stegene beskrives nærmere i det følgende.

❖ **Opprette RadioTransmitter-objekt**

I dette steget opprettes et *RadioTransmitter*-objekt. Objektet arver av *EmbeddedSystem* som vist i Figur 3.2. Attributtene som er arvet fra *EmbeddedSystem* (Tabell 3.1) skaper en relasjon til enheten radioen er tilknyttet, for eksempel en stridsvogn. Dette muliggjør blant annet at to forskjellige og uavhengige simuleringer kan simulere henholdsvis stridsvogner og radioer. Denne metoden benyttes bl.a. ved simulering av Identify Friend Foe-transpondere [8].

Arvede attributter	Definisjon
EntityIdentityIdentifier	Identifikator satt sammen av lokasjons ID-en, applikasjons ID-en, og entitetsnummeret til entiteten radioen er tilknyttet.
HostObjectIdentifier	Objektinstans-ID-en til entiteten radioen er tilknyttet.
RelativePosition	Den relative posisjonen til radioen i forhold til den tilknyttede enheten.

Tabell 3.1 Arvede attributter fra klassen EmbeddedSystem.

RadioTransmitter-klassen inneholder feltene i Tabell 3.1 og

Tabell 3.2. Attributter skrevet med fet skrift må fylles ut, resterende er valgfrie. Ved å tillate valgfrie attributter gir dette fleksibilitet som gjør det mulig å komponere simuleringsføderasjoner med komponenter som i varierende grad simulerer de forskjellige aspektene av radiokommunikasjon. Samtidig skaper dette utfordringer med hensyn på rettferdig strid.

Klassens attributter	Definisjon
AntennaPatternData	Angir mønsteret på strålingen fra antennen.
CryptoSystem	Identifiserer typen kryptoutstyr som benyttes.
CryptographicMode	Indikerer om basebånd eller tofase benyttes.
EncryptionKeyIdentifier	En tallverdi som identifiserer kryptonøkkel. Dersom en radiosender og –mottaker publiserer samme tallverdi benytter de også samme kryptonøkkel.
Frequency	Den midtre frekvens for radiotransmisjoner.
FrequencyBandwidth	Båndpass for transmisjoner.
SpreadSpectrum	Verdien er sann hvis det benyttes en frekvenshoppende algoritme for transmisjoner.
StreamTag	En globalt unik identifikator for å assosiere lydstrømmer med en kanal. Ref. *EncodedAudioRadioSignal* i RPR FOM vist i Figur 3.3.
RadioIndex	Dette er et identifikasjonsnummer for hver radio på en gitt enhet. Dette nummeret kan ikke endres under simulering.
RadioInputSource	Angir hvilken posisjon eller dataport som radiosenderen mottar data fra. Eksempler på posisjon er pilot, vognkommandør, skytter osv.
RadioSystemType	Datastrukturen består av følgende felter som ikke kan endres under simulering. Verdiene feltene kan ha er definert i [9]. **Type**: Angir type utstyr, f.eks. ammunisjon, sensor eller radio. Tallverdi 7 for radio. **Land**: F.eks. tallverdi 163 for Norge. **Domene**: F.eks. land (1), luft (2), overflate (3), osv. **Kategori**: Type radio, f.eks. tale, taktisk datalink eller landingssystem for luftfartøyer. **Versjon av nomenklatur**: Angir om det benyttes nasjonal, produsentspesifikk eller standardisert militær nomenklatur. **Nomenklatur**: Angir spesifikk nomenklatur.
RFModulationSystemType	Angir hvordan modulasjonsparametere skal tolkes.
RFModulationType	Klassifisering av modulasjonstype.
TimeHopInUse	Sann hvis en tidsdelt algoritme benyttes for kanaltilgang.
TransmittedPower	Gjennomsnittlig sendereffekt.
TransmitterOperationalStatus	Angir om radiosenderen er skrudd på.
WorldLocation	Angir radioantennens posisjon.

Tabell 3.2 Attributter i klassen RadioTransmitter.

❖ **Opprette radiomottaker**

I dette steget opprettes et *RadioReceiver*-objekt. Objektet er som *RadioTransmitter*-klassen en underklasse av *EmbeddedSystem* som vist i Figur 3.2. *RadioReceiver*-klassen inneholder feltene i tabellen under.

Klassens attributter	Definisjon
RadioIndex	Dette er et identifikasjonsnummer for hver radio på en gitt enhet. Dette nummeret kan ikke endres under simulering.
ReceivedPower	Gjennomsnittlig styrke på mottatt signal.
ReceivedTransmitterIdentifier	Objektinstans-ID-en settes lik radiosenderen hvis melding mottas.
ReceiverOperationalStatus	Angir om radiomottakeren er skrudd på.

Tabell 3.3 Attributter i klassen RadioReceiver.

❖ **Opprette radiomelding**

I dette steget opprettes en av fire typer objekter. Henholdvis *RawBinaryRadioSignal*, *ApplicationSpecificRadioSignal*, *DatabaseIndexRadioSignal* eller *EncodedAudioRadioSignal*. Disse er underklasser av klassen *RadioSignal* som selv ikke inneholder egne attributter (se Figur 3.3). I tabellen under gjengis klassen *ApplicationSpecificRadioSignal*. Denne dekker mesteparten av radiofunksjonaliteten som finnes i RPR FOM. De andre er beskrevet i [4].

Som del av prosessen med å opprette et *RadioSignal*-objekt må følgende gjennomføres:

1. Opprette meldingsobjekt.
2. Sette dataraten meldingen skal sendes med.
3. Bestemme hvilken lokal radio som skal benyttes for transmisjon og merke meldingen med objektinstans-ID-en til radioen.
4. Fylle ut innholdet i meldingen. Dersom dette er et *DatabaseSpecificRadioSignal* skal meldingsinnholdet peke til en database som inneholder meldingene som skal sendes.
5. Beregne lengden på meldingen i antall bit og merke meldingen med riktig lengde.
6. Dersom meldingen sendes med taktisk datalink (TDL) må meldingen merkes med type TDL og med antall meldinger innholdet i meldingsobjektet består av.
7. Dersom dette er et *ApplicationSpecificRadioSignal* skal også protokoll-ID-feltet merkes slik at det er mulig for mottaker å tolke meldingen på korrekt måte.

❖ **Sende radiomelding**

Det finnes ingen adresseringsmekanisme for å nå et spesifikt sett med mottakere slik *RadioSignal*-klassen er definert i RPR FOM. Derfor består dette steget av å forsikre seg om at attributtene til radioen er fylt ut korrekt, f.eks. at kryptoparameretere, frekvens og sendereffekt er satt riktig og at radioen er slått på. Deretter publiseres radiomeldingen ut på HLA.

Klassens attributter	Definisjon
DataRate	Dataraten meldingen sendes med.
HostRadioIndex	Objektinstans-ID-en til radioen som sender meldingen. Består av til sammen fire identifikatorer (lokasjons-, applikasjons-, entitets- og radionummer).
SignalDataLength	Lengden på signalet angitt i antall bits.
SignalData	Inneholder innholdet i en melding
TacticalDataLinkType	En tallverdi som representerer en spesiell taktisk datalink.
TDLMessageCount	Angir antall TDL-meldinger som meldingen inneholder.
UserProtocolID	Angir hvordan hva slags koding/protokoll som er benyttet for meldingsinnholdet.

Tabell 3.4 Attributter i klassen ApplicationSpecificRadioSignal.

Mangelen på adresseringsinformasjon har gjort at det har oppstått ad hoc måter å visualisere at to enheter kommuniserer med hverandre. MÄK Technologies [10] visualiserer i sine applikasjoner [6;11] at to enheter kommuniserer ved å undersøke tiden mellom to radiomeldinger. Dersom to enheter sender meldinger innenfor et kort tidsrom så antar applikasjonene at enhetene kommuniserer og visualiserer dette.

❖ **Motta radiomelding**

Ved mottak av radiosignaler skal *RadioReceiver*-klassens atributter gjenspeile at den simulerte radiomottakeren er skrudd på, hvilken styrke den mottar signalet med og fra hvilken radiosender den mottar meldinger fra. Det er den simulerte mottakerens ansvar å simulere hvorvidt det er mulig å ta i mot og dekode signalet for videre prosessering.

RPR FOM legger med andre ord til rette for simulering av radiopropagasjon og alle-til-alle kommunikasjon. Attributtet *StreamTag* i *RadioTransmitter*-klassen kan benyttes for å skille talenett fra hverandre. En kan også tenke seg å benytte samme mekanisme for å lage multicastgrupper for annen meldingstrafikk.

En annen måte å benytte RPR FOM er som en grunnstein i modeller av det fysiske laget og legge til modeller av alle de høyereliggende kommunikasjonslagene i modellen til enhver kommuniserende enhet. Signaleringsinformasjon mellom og internt i protokoller følger da med de simulerte meldingene. Attributtet *SignalData* i underklassene av *RadioSignal* vil da inneholde informasjon som går horisontalt mellom lagene i OSI-modellen og som etterlikner kommunikasjonsprotokollagenes signalering mellom enheter. Eksempler er MAC- og IP-adresser, samt TCP-informasjon. Dette er i praksis en måte å konstruere en distribuert nettverksimulator basert på HLA og er en krevende måte å konstruere simuleringer på. Det øker mengden meldinger som utveksles mellom simuleringer dramatisk og kan ha konsekvenser for hvorvidt det er mulig å gjennomføre plattformsimuleringer i sann tid pga. kapasitetsproblemer i det fysiske nettverket og i kjøretidsinfrastrukturen som knytter simuleringer sammen. Implementasjon av simulerte kommunikasjonsprotokoller i hver simulerte enhet skaper også unødvendig høy kompleksitet. Hver simulator må implementere samme modell av

kommunikasjonsprotokoller som benyttes. Full kompatibilitet mellom entitetsbaserte simuleringsmodeller fra forskjellige leverandører synes i dag vanskelig å oppnå. Kompatibilitetsproblemene vil ikke bli mindre ved å legge til simulering av protokoller.

I det neste kapitelet beskrives modellen som ble utviklet for å benytte OMNeT++ i distribuerte simuleringer basert på HLA og som gjør det lettere å simulere nettverk.

4 Kommunikasjonsmodell for å benytte OMNeT++ som tjeneste i distribuerte simuleringer

Kommunikasjonssimuleringsrammeverket med OMNeT++ skal dekke modellering og simulering av alle lagene i OSI-modellen. I tillegg skal entitetsbaserte simuleringer kunne inneha modeller for kommunikasjon seg i mellom, det vil si mot lag syv i OSI-modellen, og kunne benytte OMNeT++ for modellering og simulering av underliggende lag. Det er derfor behov for et grensesnitt mellom modellene til entitetsbaserte simuleringer og modellene til OMNeT++. I tillegg er det behov for å kunne adressere avsendere og mottakere i kommunikasjonsmodellen, å forenkle simulering av radioer og for et enkelt programvaregrensesnitt for sending og mottak av simulerte meldinger og datastrømmer.

4.1 Grensesnittet mellom kommunikasjonsmodellene til entitetsbaserte simuleringer og OMNeT++

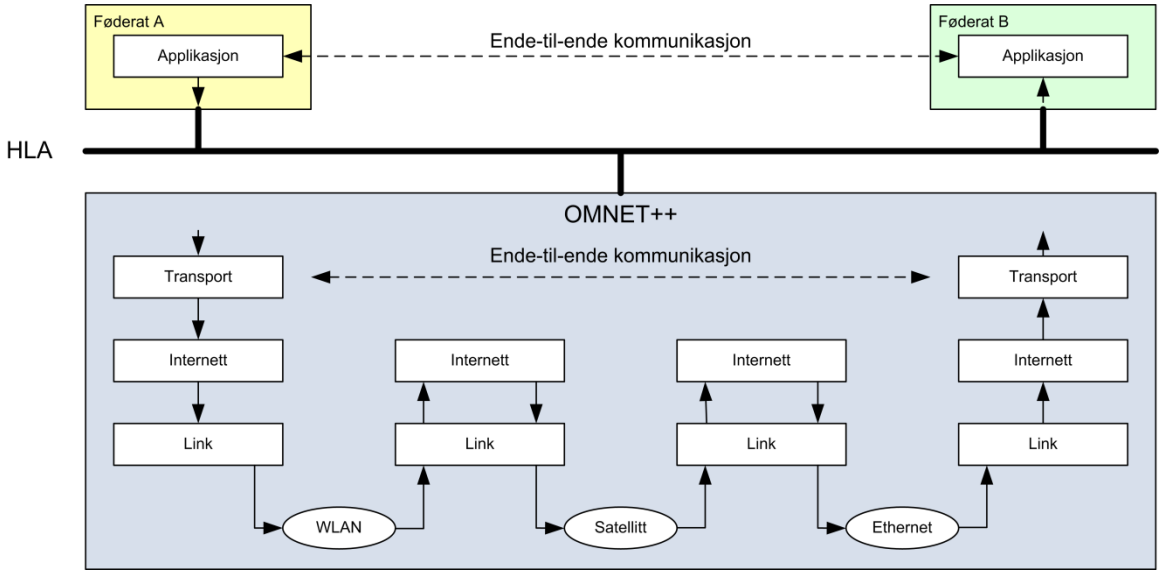
Det er utviklet et grensesnitt mellom kommunikasjonsmodellene til entitetsbaserte simuleringer og OMNeT++ som endrer konseptet i RPR FOM for simulering av datakommunikasjon på en grunnleggende måte. I den nyutviklede modellen utveksles data mellom entiteter ved lag syv, det vil si at kommunikasjonen foregår ende-til-ende, det opereres med datastrømmer som lyd og bilde, og med meldinger, ikke med radiosignaler. Kommunikasjonsmodellen er i større grad teknologiuavhengig, og en utnytter eksisterende adresserings-, pålitelighets-, krypterings-, tjenestekvalitets- og rutingmekanismer osv. i lagene under. Lagene under blir simulert av OMNeT++. Figur 4.1 illustrerer dette.

For å få til dette er det nødvendig med en modell av grensesnittet mot lag syv. Det er laget en objektmodell der endepunkter (les applikasjoner) er definert som henholdsvis produsenter og konsumenter av data. Ved å representere endepunktene som egne klasser i en HLA-basert objektmodell er det mulig å adressere spesifikke konsumenter direkte. Denne måten å modellere endepunkter på er teknologiuavhengig i forhold til kommunikasjonslagene under. For å representere gruppekommunikasjon er det definert konsumentgrupper i objektmodellen.

Klassen *DataServiceAccessPoint* representerer applikasjoners roller. Produsenter og konsumenter knyttes til *BaseEntity*-klassene i RPR FOM. Dermed er det mulig å ha flere konsumenter (applikasjoner) per entitet og per radio. Det er valgt ikke å endre radiomodellen i RPR FOM fordi den er tilstrekkelig for å modellere det fysiske laget. En beholder kompatibilitet med eksisterende simuleringer og visualiseringer og det fremmer gjenkjennbarheten i modellen med tanke på

utbredelsen til RPR FOM. Modellen av relasjonen mellom den nye objektklassen og radioobjektklassene er gjengitt i Figur 4.2.

I denne modellen er kommunikasjon modellert på lag syv og simulering og publisering av tilstandsinformasjon for radiosendere og -mottakere er ikke nødvendig for å simulere kommunikasjon mellom radiobaserte noder. Det kan likevel være hensiktsmessig å publisere tilstandsinformasjon om radiosendere og -mottakere av to grunner. Den første er simulering av elektronisk krigføring der deteksjon av radiosendere er ett element av interesse. I et slikt tilfelle kan det også være av interesse å publisere radiosignaler for hvert hopp i et multi-hopp nettverk. Til dette benyttes RPR FOM sitt *RawBinaryRadioSignal*.



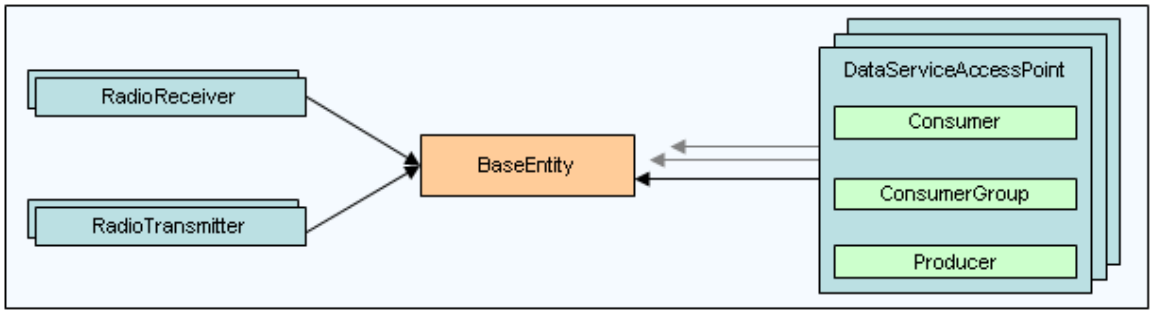
Figur 4.1 Fordeling av simuleringsansvar i henhold til Internett-modellen.

Den andre grunnen er dersom entiteten radioen er tilknyttet ønsker å endre på radioparameretere, for eksempel å slå radioen av for å redusere sannsynligheten for å bli oppdaget i forbindelse med simulering av elektronisk krigføring. HLA har en mekanisme for å la to simuleringer samarbeide om å simulere objekter kalt ”ownership transfer management”. Med denne kan OMNeT++ publisere radioobjekter og la andre simuleringer overta eierskap over attributter de ønsker kontroll over. På denne måten kan andre simuleringer for eksempel styre om radioer er slått på uten å benytte mekanismer (interaksjonstyper) for dette mellom de entitetsbaserte simuleringene og OMNeT++. Et viktig poeng i dette tilfellet er at RPR FOM ikke støtter overføring av eierskap til attributter, kun hele objektklasser. RPR FOM garanterer dermed ikke at simuleringer vil være konsistente dersom mekanismen i HLA likevel benyttes. Derfor benyttes en annen mekanisme i RPR FOM for å oppnå dette. Med interaksjonsklassen *AttributeChangeRequest* kan enititetsbaserte simuleringer be OMNeT++ om å endre radioparametere. OMNeT++ velger selv om og når den ønsker å rette seg etter forespørselen og svarer med interaksjonsklassen *AttributeChangeResult*.

Figur 4.3 gjengir et klassediagram av *DataServiceAccessPoint*. Objektklassen er modellert som en underklasse av *EmbeddedSystem*. *EmbeddedSystem* har en peker til objektklassen som applikasjonen som genererer meldingen er del av. *DataServiceAccessPoint* har et attributt som ved hjelp av en enumerert verdi gjenspeiler objektklassens rolle. De enumererte verdiene er:

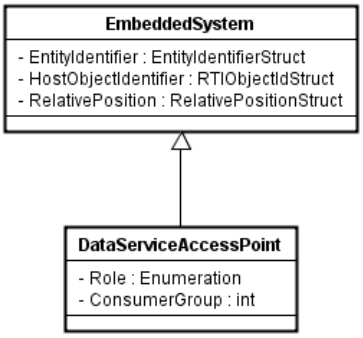
- Produser
- Consumer
- ConsumerGroupMember
- ProduserAndConsumer
- ProduserAndConsumerGroupMember

Den har også et attributt som gjenspeiler tilhørighet i en konsumentgruppe dersom objektklassen skal benyttes for gruppekommunikasjon.



Figur 4.2 Relasjonen ("pekerstrukturen") mellom klassene.

For å utnytte at grensesnittet mot lag syv er adresserbart, er det nødvendig å innføre nye modeller av meldinger og datastrømmer som benytter seg av adresseringsmekanismen. Dette er gjort ved hjelp av tre nye interaksjonsklasser. Den første klassen modellerer overføring av data. De to siste modellerer samhandling mellom entitetsbaserte simuleringer og OMNeT++. Henholdsvis forespørsel til OMNeT++ om å simulere overføring av data og respons fra OMNeT++ om status på forespørselen. Modellen er laget slik at det er mulig å simulere overføring av data med adressering av sender og mottaker uten å benytte tjenestene OMNeT++ tilbyr. Entitetsbaserte simuleringer må i slike tilfeller selv simulere effektene av bruk av kommunikasjonsteknologier. Figur 4.7 gjengir modellen av de nye interaksjonene. Disse arver fra klassen *Communications* på samme måte som klassen *RadioSignal*. Det er opprettet to nye subklasser, *ProtocolDataUnits* og *CESMessages*.



Figur 4.3 Klassediagram av klassen som representerer produsenter og konsumenter av data.

ProtocolDataUnits samler klasser som modellerer data som utveksles mellom lagene i OSI-modellen. Det er modellert kun en underklasse av *ProtocolDataUnit*. Interaksjonsklassen *DataServicePDU* modellerer informasjon som flyter mellom applikasjonen og lag syv. Dersom det er behov for å modellere flere kommunikasjonslag i en entitetsbasert simulering vil det være nødvendig å legge til interaksjonsklasser som modellerer grensesnittet mellom det nederste laget den entitetsbaserte simuleringen dekker og det øverste OMNeT++ simulerer. Det har ikke vært behov for å modellere flere grensesnitt under arbeidet beskrevet i denne rapporten.

DataServicePDU inneholder syv felter.

- **avsender**; inneholder en peker til *DataServiceAccessPoint*
- **mottaker**; inneholder en peker til *DataServiceAccessPoint*
- **prioritet**; en tallverdi som angir prioritet som ønskes for datapakken i det simulerte nettet
- **serienummer**; en tallverdi som identifiserer datapakken unikt for hvert *DataServiceAccessPoint* sender- og mottakerpar
- **innhold**; et buffer som inneholder dataene som skal overføres mellom to kommuniserende noder i et nett
- **lengde**; en tallverdi som angir lengden på innholdet i datapakken i bit
- **metadataidentifikator**; en enumerert identifikator som angir en metamodell av meldingsinnholdet.

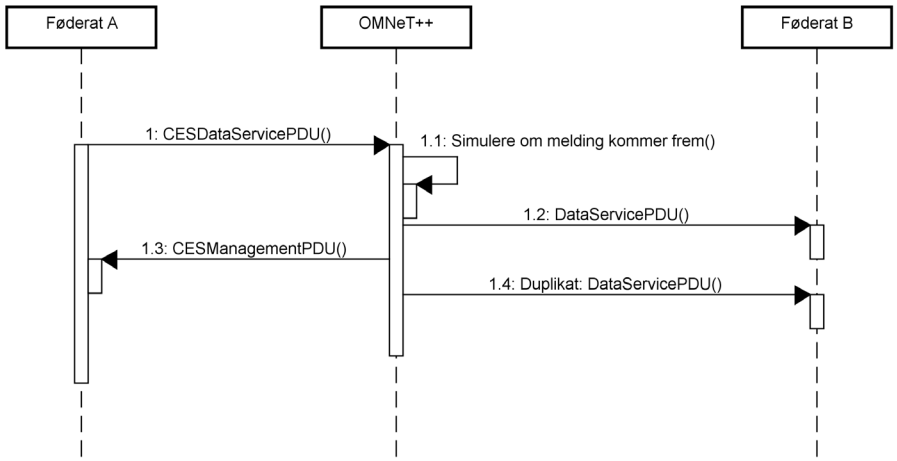
CESMessages samler interaksjoner som modellerer kommunikasjon mellom entitetsbaserte simuleringer og OMNeT++. Navnet *CESMessages* henspeiler på den engelske betegnelsen Communication Effects Server. Det er definert to underklasser av *CESMessages*.

CESDataServicePDU er helt lik *DataServicePDU*, men har en annen semantikk. Det er en forespørsel til OMNeT++ om å simulere overføring av data mellom to applikasjoner. Avsender og mottakeradressene henspeiler på mellom hvilke applikasjoner datatrafikk skal simuleres. Kommuniserende applikasjoner kan være lokalisert på samme enhet eller til og med på to forskjellige kontinenter der data må traversere bærere som satellittkommunikasjon, Ethernet og HF-radio før de når endelig destinasjon.

CESManagement interaksjonsklassen er definert for å gi OMNeT++ mulighet til å kommunisere til avsender og mottaker om status på forespørselen om å simulere overføring av data mellom to applikasjoner. Klassen inneholder fire attributter. Avsender- og destinasjonsadresse, sekvensnummer som identifiserer dataene unikt og et attributt som angir status for overføringen. Med denne modellen kan avsendere og mottakere ikke forespørre OMNeT++ om status, OMNeT++ gir denne på eget initiativ. Hittil er det definert følgende statusbeskjeder:

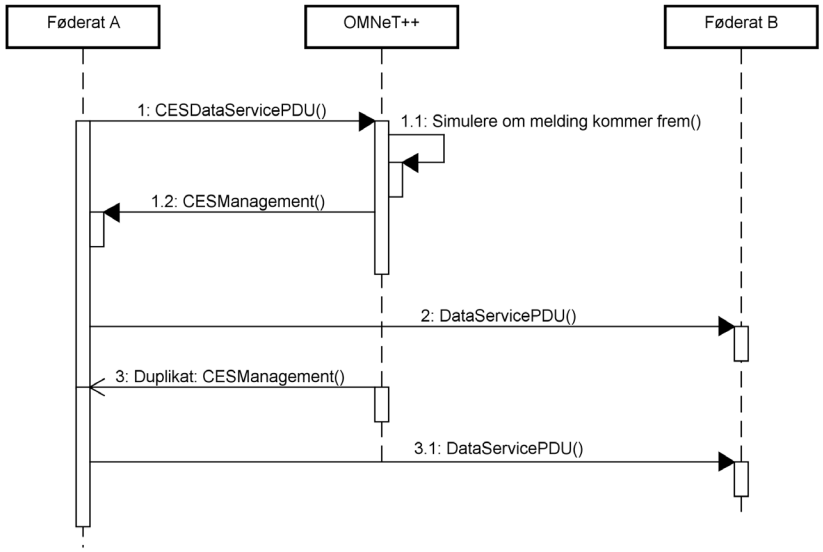
- MsgDelivered
- MsgDeliveryFailed
- ResourcesNotAvailable
- QoSLevelNotAvailable
- NoPowerAvailable
- RadioJammed
- NoRouteToHost
- HostUnknown
- CESOverloaded

Simulering av datakommunikasjon ved hjelp av OMNeT++ kan gjøres på tre forskjellige måter ved hjelp av de definerte objekt- og interaksjonsklassene. Den første måten er å sende alle data inklusive meldingsinnhold til OMNeT++ ved hjelp av *CESDataServicePDU* for simulering. Dette er illustrert i Figur 4.4. OMNeT++ simulerer ende-til-ende kommunikasjon. Dersom simuleringen viser at pakken kommer frem, sender OMNeT++ denne til destinasjonen ved hjelp av *DataServicePDU* til riktig tid. Dersom dataene tapes, gir OMNeT++ beskjed til avsender om resultatet av forsøket ved hjelp av en *CESManagement*-interaksjon. Dette er en ressurskrevende måte å simulere på. Trafikken på nettverket blir tredoblet i antall pakker som sendes mellom simuleringer og mellom to- og tredoblet i antall bytes som sendes. I tillegg må kommunikasjonssimulatoren håndtere datamengden (innholdet i dataene) som til enhver tid er i transitt mellom simulerte entiteter. Dette krever mye prosesseringsressurser hos nettverkssimulatoren. Men denne metoden har også fordeler. Den gir mulighet for å manipulere innhold i datakommunikasjon ved hjelp av OMNeT++. En kan for eksempel simulere degradering av taledata, simulere angrep på og manipulasjon av trafikk som går mellom to entiteter og en kan simulere duplisering av meldinger (interaksjon 1.4 i Figur 4.4).



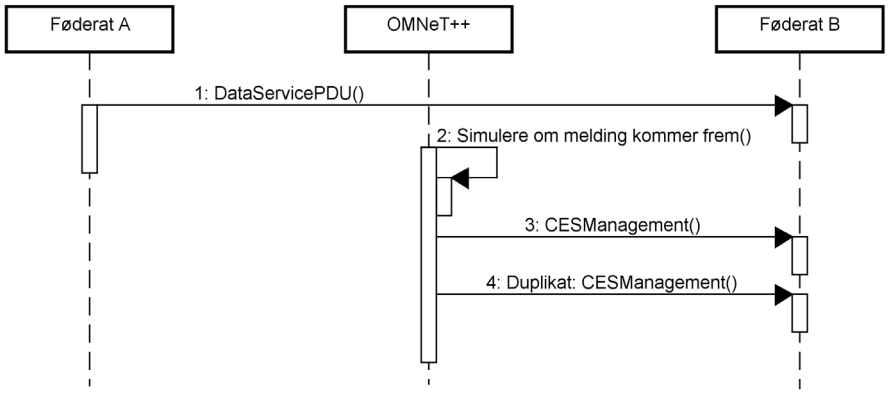
Figur 4.4 Sekvensdiagram for flyten av interaksjoner ved bruk av den første metoden.

Den andre måten å benytte OMNeT++ på er å sende *CESDataServicePDU* til OMNeT++, men uten meldingsinnhold. Dette er illustrert i Figur 4.5. OMNeT++ simulerer så om dataene kommer frem til destinasjonen og sender beskjed til avsender ved hjelp av en *CESManagement*-interaksjon om status. Dersom simuleringen viser at dataene kommer frem, sender avsenderen en *DataServicePDU* til mottaker. Dette reduserer ikke antall meldinger i nettverket mellom simuleringene. Mengden data som nettverket og OMNeT++ må håndtere reduseres imidlertid betraktelig (avhengig av størrelsen på meldingene som utveksles mellom simulerte entiteter). Med denne metoden kan ikke OMNeT++ modellere endringer i dataene som overføres. Duplisering krever utvidelse av *CESManagement*-interaksjonen med flere statusverdier og at avsendere bufrer meldinger lenge nok til at forespørsler fra OMNeT++ til avsender om retransmisjon av duplikater kan behandles. Duplikater dobler antall meldinger som behøves mellom simuleringer sammenliknet med den første metoden fordi det kreves to interaksjoner for å simulere hvert duplikat.

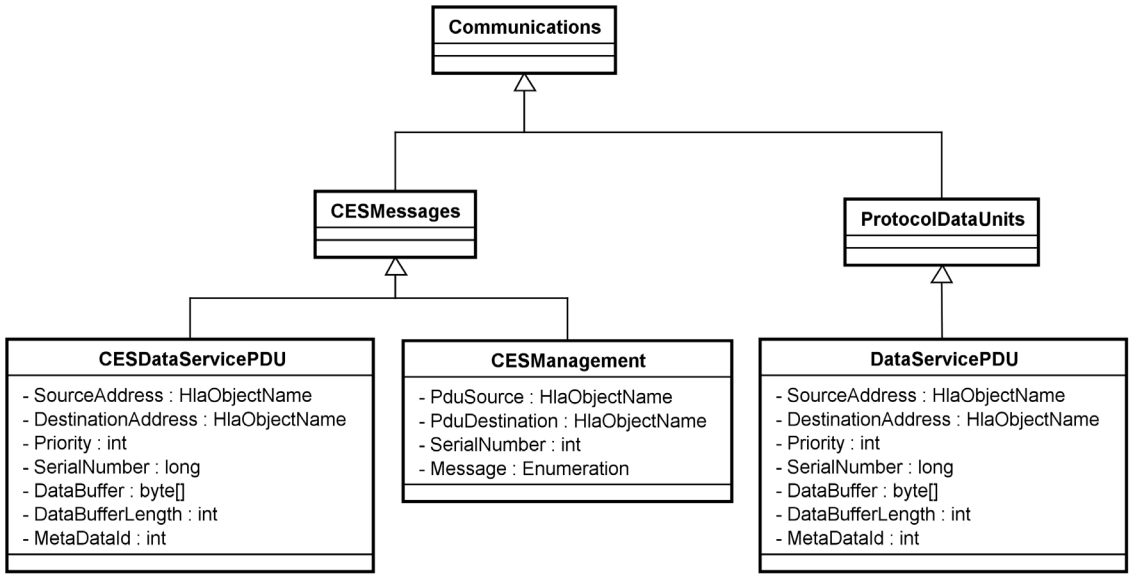


Figur 4.5 Sekvensdiagram for flyten av interaksjoner ved bruk av den andre metoden.

Den tredje måten å benytte OMNeT++ på er å sende en full datapakke til destinasjon ved hjelp av *DataServicePDU*. Dette er illustrert i Figur 4.6. Dersom mottakeren er registrert i databasen til OMNeT++ som en simulering som ønsker kommunikasjonssimuleringstjenester, vil OMNeT++ prosessere interaksjonen og simulere om dataene kommer frem. Mottaker bufrer meldingen helt til den får en *CESManagement*-interaksjon som forteller om status for deretter å avgjøre om den kan konsumere dataene. Avsender mottar også statusbeskjeden dersom den har behov for å vite om meldingen kom frem. Dette gir det minste forbruket av båndbredde og kan benyttes av føderater som selv implementerer kommunikasjonssimulering og av føderater som ikke støtter OMNeT++ (selv om de må implementere *DataServicePDU*). Dette krever at mottaker har nok bufferkapasitet til mellomlagring av meldinger som ikke er avdømt av OMNeT++ eller som kan komme som duplikater.



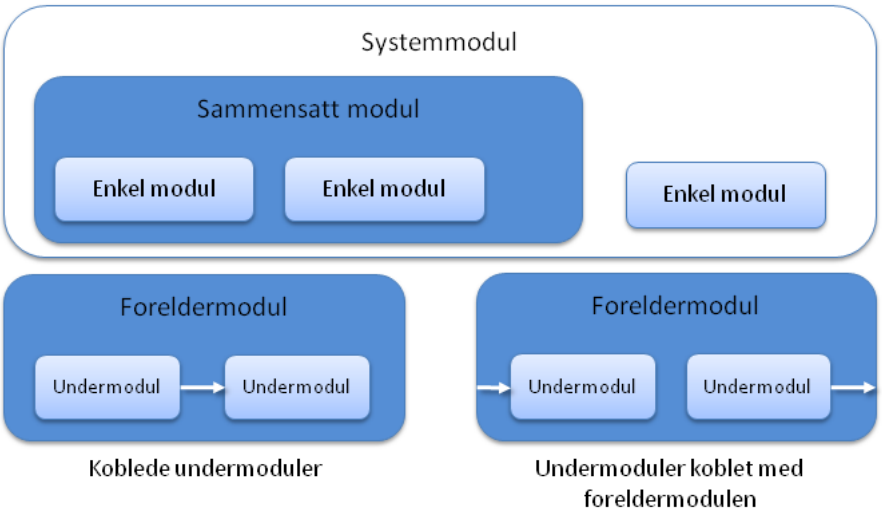
Figur 4.6 Sekvensdiagram for flyten av interaksjoner ved bruk av den tredje metoden.



Figur 4.7 Klassediagram av de nye interaksjonsklassene.

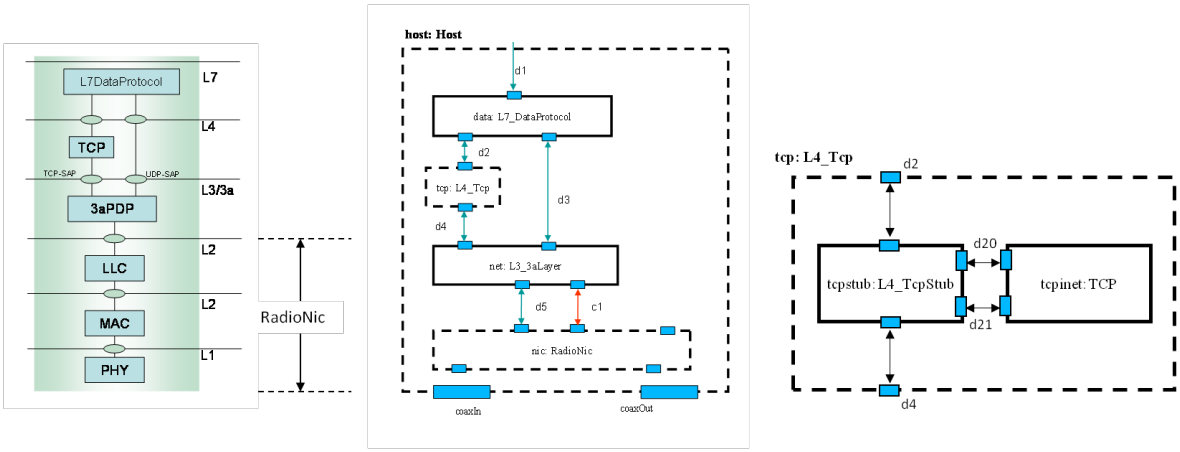
4.2 Simuleringsmodellen i OMNeT++

Som nevnt i kapittel 2.3 er OMNeT++ sine modeller basert på enkle moduler som komponeres sammen til større og mer funksjonelle moduler. Større moduler komponeres hierarkisk og kobles sammen med andre moduler. Figur 4.8 illustrerer dette. Moduler kommuniserer ved hjelp av meldinger. Disse meldingene er også modellert hierarkisk som betyr at de kan innkapsles i hverandre. I simulering av datakommunikasjon vil en typisk innkapsle lag på lag med header-informasjon/protokoll-informasjon.



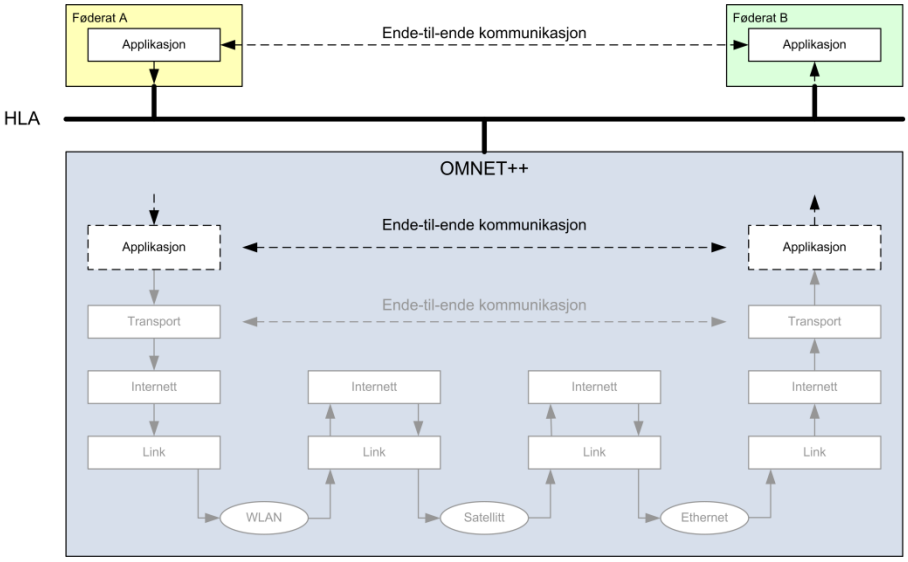
Figur 4.8 Illustrasjon av generell oppbygning av modeller i OMNeT++.

En god måte å modellere nettverk på vil være å lage enkle moduler som modellerer delfunksjonalitet på hvert av OSI-lagene, samle funksjonalitet for hvert lag i større koblede moduler, og så koble lagene sammen. Figur 4.9 illustrerer dette. I figuren er det laget en modell av en node. Modellen er basert på Internett-modellen. I det viste utsnittet vises den hierarkiske oppdelingen der ende-noden er toppmodul og hvert av kommunikasjonslagene er modellert som koblede undermoduler. Kommunikasjonslagene består av enkle moduler som er programmert i C++. De behandler meldinger som kommer inn og sender videre på utporter.



Figur 4.9 *I [12] illustreres dette på en god måte. Til venstre vises den lagdelte modellen basert på Internett-modellen, i midten vises den koblede modellen av en node i nettverket, og til høyre er lag fire fra den midterste figuren nærmere illustrert.*

I OMNeT++ sine modeller vil objektklassen *DataServiceAccessPoint* være representert ved en stedfortredermodul illustrert i Figur 4.10 med stiplet boks. Denne modulen vil typisk være en enkel modul skrevet i C++ som innkapsler HLA-interaksjonen *CESDataServicePDU* med nødvendige metadata (f.eks. IP-adresser) i en meldingstype som kommunikasjonslaget under kan håndtere. I figuren er kommunikasjonslaget under applikasjonslaget på lag fire. Hvilke lag som er modellert i OMNeT++ og hvor mange lag en modell består av er irrelevant for interoperabilitet med HLA så lenge modellen har et veldefinert grensesnitt mot applikasjonslaget.



Figur 4.10 *Illustrasjon av stedfortredermodulen som er nødvendig for å modellere eksterne HLA-baserte applikasjoner som kommuniserer.*

Applikasjonslaget vil i de fleste tilfeller være spesialskrevet for bruk med HLA. Lagene under krever endringer for å oppdateres med ny informasjon ved for eksempel mobilitet eller dersom det er ønskelig å publisere informasjon fra lavereliggende lag på HLA om for eksempel

objektklassen *RadioTransmitter*. Bevegelsen til noder tilgjengeliggjøres for OMNeT++-modeller ved hjelp av et ”black board” der data av generell interesse for flere moduler tilgjengeliggjøres. Det kan også være nødvendig å lage en modul med sanntidsdata fra HLA som erstatter moduler som modellerer bevegelse. Dette må spesiallages for hver modell som skal integreres. Modeller må også endres slik at de kan gi beskjed dersom for eksempel data tapes via en *CESManagement*-interaksjon.

4.3 Modellers påvirkning av ende-til-ende-kommunikasjon

Modeller som implementeres i OMNeT++ vil påvirke ende-til-ende kommunikasjon på mange måter. I det følgende vil de viktigste effektene som ligger i å benytte gode kommunikasjonsmodeller i HLA-baserte simuleringer beskrives. Effektene karakteriseres ved å beskrive resultatet opplevd av ende-nodene og ut ifra hvordan modellene påvirker nettverkstrafikk som passerer hvert av lagene.

Effekten ved ende-nodene oppleves ved økt realisme med hensyn på hvor ofte en taper meldinger og hvilken forsinkelse en opplever mellom ende-noder i forskjellige terreng og i forskjellige faser av operasjoner ved bruk av spesifikke kommunikasjonsteknologier. Terreng spiller ofte en avgjørende rolle ved radiokommunikasjon. Det er også viktig å ta hensyn til at forskjellige faser av operasjoner har ulike krav til koordinering og overføring av data mellom enheter. Nettverkets evne og kapasitet til å støtte faser med stor belastning vil materialisere seg ved størrelsen på pakketapet og lengden på forsinkelsen i nettverket. En annen viktig komponent er variasjon i forsinkelse også kalt ”jitter”. Spesielt for video- og talesamband kan variasjon i forsinkelse vanskeliggjøre kommunikasjon. Duplisering av data som sendes kan også påvirke kommunikasjon dersom kommunikasjonslagene under applikasjonslaget ikke filtrerer dette. Et siste poeng er ende-noders opplevde kvalitet med de overnevnte metrikker når en også tar hensyn til prioritering av trafikk. I operasjoner vil mengden trafikk med ulike prioriteter variere som funksjon av tid, dermed vil også den opplevde kvaliteten for de forskjellige prioritetsklassene også variere etter hvor mye kapasitet som til enhver tid er tilgjengelig for hver av klassene.

Den andre måten er å beskrive effektene av hvordan hvert kommunikasjonslag for hvert hopp mellom to kommuniserende ende-noder inklusive ende-nodene påvirker opplevd tjenestekvalitet hos ende-nodene. Summen av påvirkningen fra de forskjellige lagene oppleves i ende-nodene som beskrevet over. De viktigste aspektene per lag oppsummeres i det følgende:

Det fysiske laget: På det fysiske laget bestemmer valg av frekvens, modulering, koding, sendereffekt, antennenes egenskaper, elektronikkens evner til å dekode signaler, tilstedeværelse av andre signaler i samme frekvensbånd og terreng- og atmosfæriske forhold hvorvidt det er mulig å overføre signaler mellom to noder. Dette påvirker i stor grad bitfeil, pakketap og overføringskapasitet.

Linklaget: På linklaget påvirker valg av strategi for:

- aksess til delte radiokanaler,
- feildeteksjon og retting,
- retransmisjon,
- fragmentering,
- kryptering,
- mobilitet mellom nettverk,
- prioritering og køing av trafikklasser og
- flytkontroll

størrelsen på overføringskapasitet, bitfeil, pakketap, forsinkelse og jitter i et nettverk.

Nettverkslaget: På nettverkslaget bygges rutetabeller som gjør at pakker kan sendes på tvers av nettverk, enheter er mobile mellom nettverk, trafikklasser prioriteres og køes, pakker fremsendes, det finnes mekanismer for signalisering av metning i nettverk, for gruppekommunikasjon og for sikkerhet. Hvordan ruter planlegges, pakker køes og hvor mye prosesseringsressurser som finnes tilgjengelig påvirker pakketap, forsinkelse og jitter.

Transportlaget: På transportlaget fragmenteres pakker for å passe størrelsesmessig til nettverkslagets begrensninger, det finnes mekanismer for ende-til-ende-pålitelighet, og flytkontroll implementeres. Valg av strategi for ende-til-ende-pålitelighet og flytkontroll påvirker i stor grad forsinkelse, jitter og pakketap.

Sesjonslaget: Sesjonslaget definerer endepunkter for kommunikasjon og har ingen direkte påvirkning på opplevd tjenestekvalitet.

Presentasjonslaget: På dette laget implementeres ofte sikkerhetsmekanismer som ende-til-ende-kryptering per applikasjon. Prosseringskapasistet i ende-nodene, mengden ekstra informasjon som sendes mellom ende-noder, og andre nettverkstjenester som kreves for å etablere en sikker kanal påvirker overføringskapasitet, forsinkelse og pakketap.

Applikasjonslaget: Trafikk som modelleres og genereres på applikasjonslaget påvirker det simulerte nettverket i like stor grad som det simulerte nettverket påvirker trafikken. Størrelsen på datapakker og hvor ofte disse sendes påvirker hvordan lagene under er i stand til å overføre data. OMNeT++ kan ha modeller av annen nettverkstrafikk som ikke kommer fra HLA-baserte entiteter. Denne trafikken modellerer ”bakgrunnsstøy”. Det er også viktig å legge merke til at de forskjellige protokollagene lager en del ”bakgrunnsstøy” som er nødvendig for at de skal fungere. Applikasjonslaget har dermed stor innvirkning på pakketap, forsinkelse og jitter.

Generelle aspekter som kan påvirke alle kommunikasjonslagene er prosesseringskapasitet, lagringskapasitet og for eksempel batterikapasitet. Ressurstilgang påvirker i stor grad

kommunikasjonslagene til enhetene som er opphav til, formidler og er endepunkter for nettverkstrafikk.

5 Programvaredokumentasjon

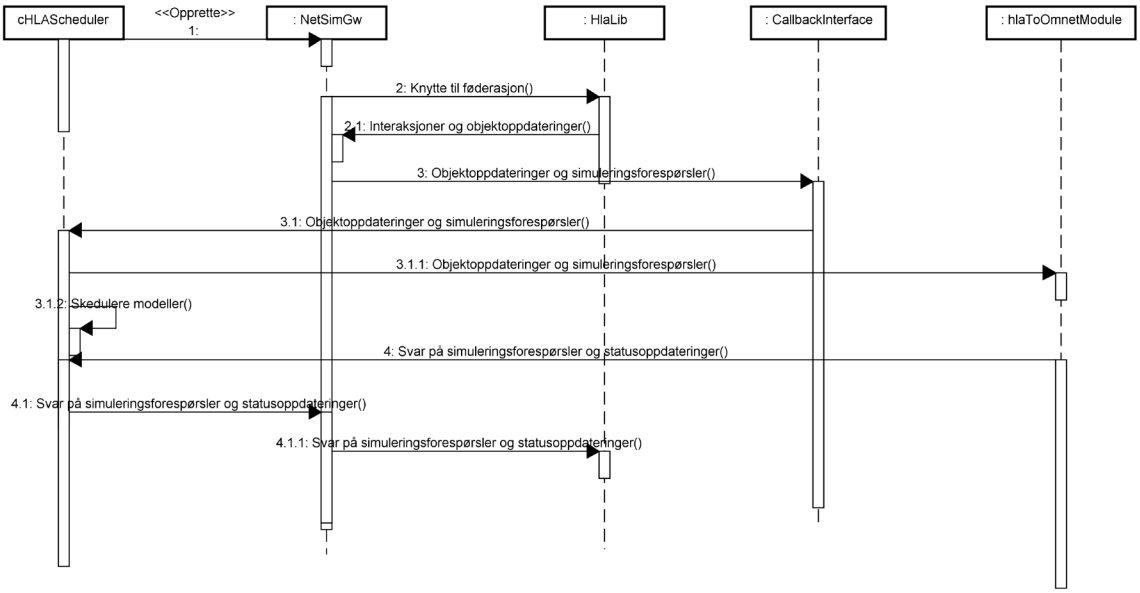
Dette kapittelet dokumenterer programvareimplementasjonen av OMNeT++ sitt grensesnitt mot HLA, tilpasning av en eksisterende modell i OMNeT++ for bruk i distribuerte simuleringer, samt noen spesielle innstillinger som er nødvendig for å kjøre OMNeT++ med endringene.

5.1 Grensesnitt for distribuert simulering

OMNeT++ sitt grensesnitt mot HLA baserer seg på programvarebiblioteket HlaLib som er utviklet ved FFI [13]. HlaLib er utviklet for å lette utvikling og integrasjon av simuleringer med HLA. Det tilbyr et høynivå abstraksjonslag mot HLA som gjør det naturlig å arbeide med FOM-en direkte uten å måtte ta hensyn til tekniske detaljer for å sende og motta data fra RTI. HlaLib er skrevet i Java. Dette påvirker hvordan integrasjonen er gjort på grunn av at OMNeT++ er skrevet i C++. I tillegg er det laget et sanntidsgrensesnitt (engelsk term ”real-time HLA scheduler”) for å la eksterne hendelser påvirke modeller eksekvert i OMNeT++. Design og implementasjon av disse beskrives i dette underkapittelet i tillegg til simulering av radiosendere og radiomottakere.

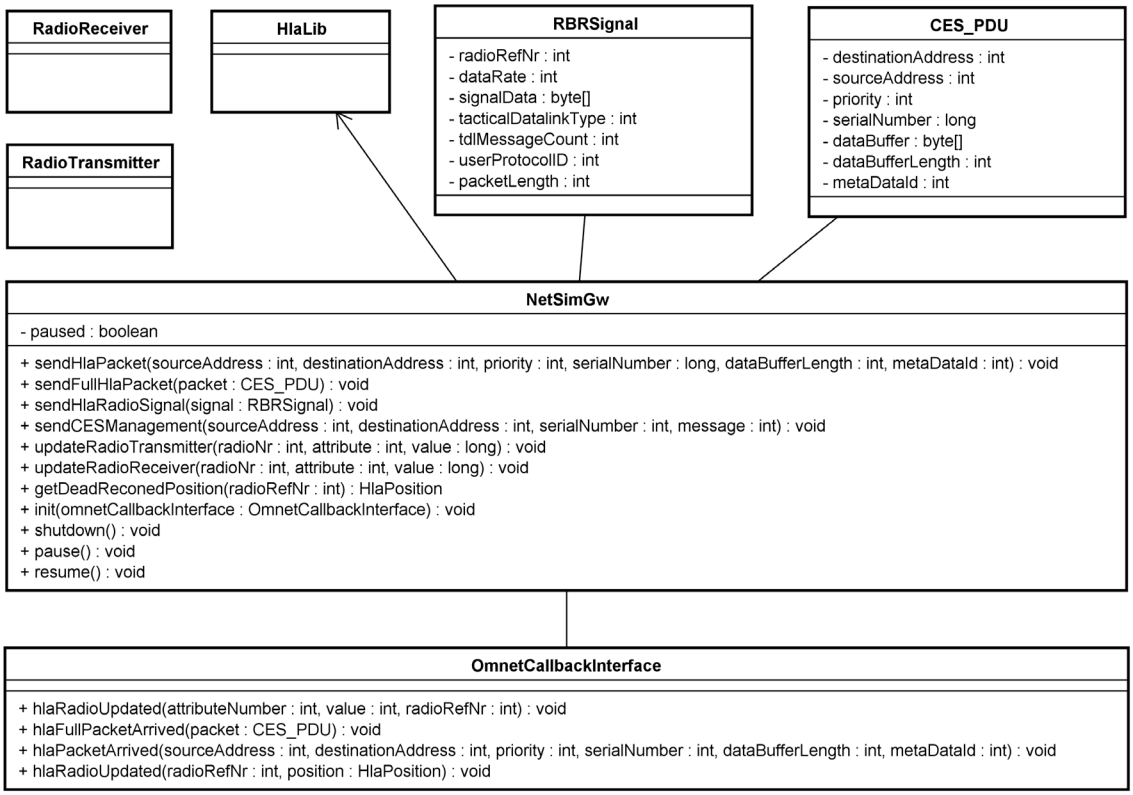
5.1.1 Integrasjon av OMNeT++ (C++) med HLA (Java)

HLA-grensesnittet beskrives under ved hjelp av klasse- og aktivitetsdiagrammer. Disse beskriver de viktigste klassene og den generelle programflyten (Figur 5.1) samtidig som de utelater noen implementasjonsdetaljer for å redusere kompleksiteten ved beskrivelsene. Figur 5.2 og Figur 5.3 gjengir klassediagram av henholdsvis Java- og C++-delen av HLA-grensesnittet. Klassediagrammet av C++-delen gjengir også sanntidsgrensesnittet.



Figur 5.1 Sekvensdiagram av meldingsflyten.

I Java-delen inneholder klassen NetSimGw[2] den viktigste funksjonaliteten. Klassen oppretter forbindelse til RTI ved hjelp av HlaLib og slutter seg til en føderasjon. Metoder i denne klassen blir kalt fra C++ for å sende interaksjoner ut på RTI-en, for å oppdatere publiserte radioer, forespørre om informasjon om HLA-objekter og for å gi klassen beskjed om hvilken tilstand nettverkssimulatoren er i (pause eller operativ). I tillegg er det definert et Java-grensesnitt som beskriver funksjonskall mot C++ kalt OmnetCallbackInterface. Dette grensesnittet implementeres av C++-klassen CallbackInterface.



Figur 5.2 Klassediagram av Java-delen av OMNeT++ HLA-gateway.

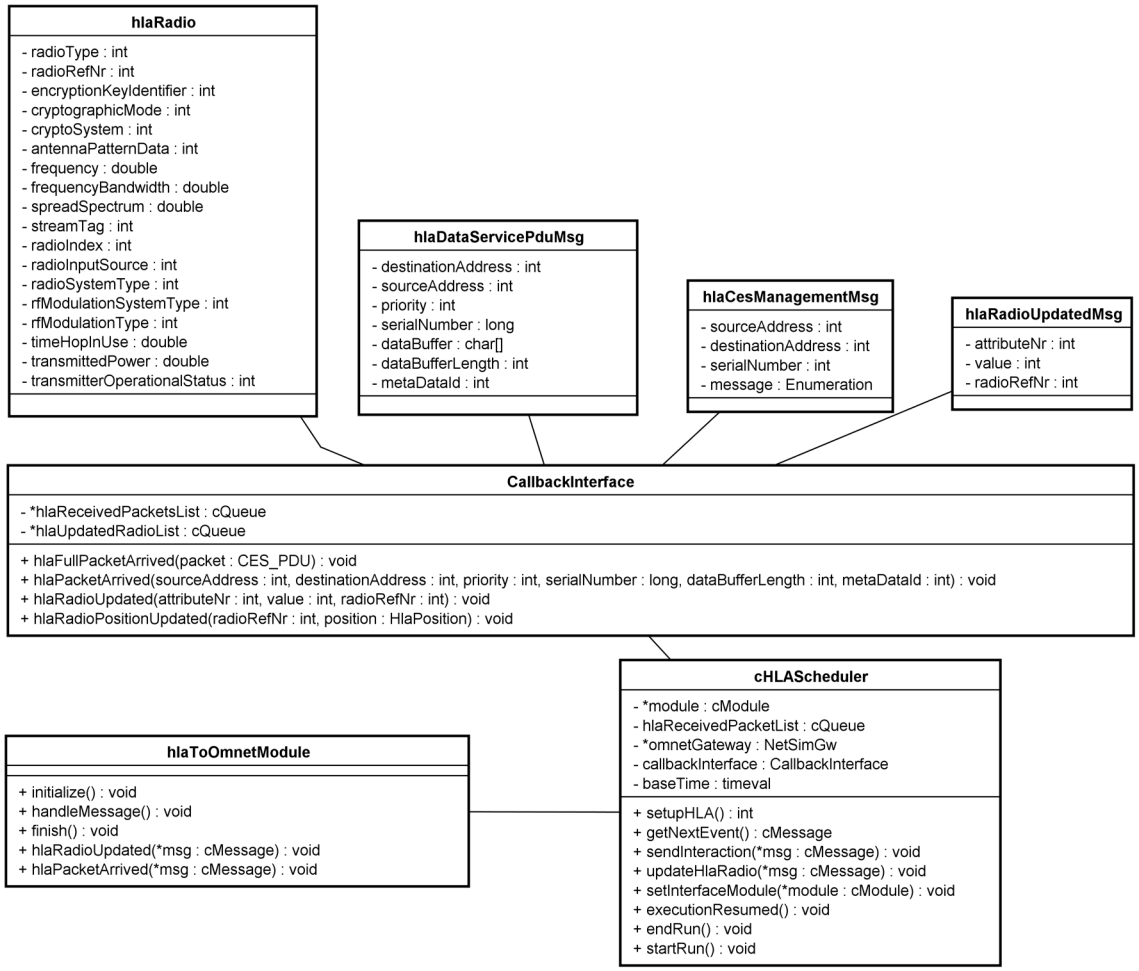
CallbackInterface lager OMNeT++-meldinger (underklasser av cMessage), som kan sendes mellom OMNeT++-moduler gjennom inn- og utporter, av forespørsler om å simulere kommunikasjon. Meldingene sendes til sanntidsgrensesnittet cHlaScheduler som innimellom eksekvering av diskrete hendelser videresender forespørslene til hlaToOmnetModule. Denne enkle modulen (eng. simple module) fordeler forespørslene til stedfortredermodulene som representerer applikasjonslagene simulert i klientføderatene. Først når dette har skjedd vil sanntidsgrensesnittet på nytt eksekvere diskrete hendelser. På denne måten unngås det at OMNeT++-simuleringer blir ustabile på grunn av introduksjon av parallellitet i en sekvensiell simulator. I tillegg vil det ikke forløpe simuleringstid fra en melding ble publisert på RTI-en til den er representert i en stedfortredermodul for applikasjonslaget. Det vil riktignok forløpe noe virkelig tid for å overføre meldingen mellom simuleringer og å prosessere den. Ved å basere seg

[2] Navn på klasser, metoder og funksjoner fra implementasjonen av grensesnittet mot RTI er skrevet med denne skrifttypen. Klasser fra FOM-en er skrevet i kursiv.

på denne fremgangsmåten kan også tidsstyrte simuleringer enkelt implementeres ved hjelp av samme metode.

Først når meldinger overføres fra stedfortredermodulen til kommunikasjonslaget under simuleres meldingers ferd gjennom en kommunikasjonsinfrastruktur utelukkende ved hjelp av OMNeT++ sine kommunikasjonsmodeller. Når meldinger kommer frem til stedfortredermoduler for applikasjonslaget ved destinasjonene, sendes disse til hlaToOmnetModule som via cHlaScheduler sender disse til Java-siden representert ved NetSimGw. Java-klassen publiserer en *DataServicePDU*-interaksjon på RTI-en dersom pakken kom frem. Den publiserer også en *CESManagement*-interaksjon for å redegjøre for status på simuleringsforespørselen.

Alle de tre metodene for simulering av datakommunikasjon med OMNeT++ nevnt i kapittel 4.1 er implementert. Det er dermed mulig å eksperimentere med hvilken som er mest hensiktsmessig for en gitt simulering. Metodene hlaFullPacketArrived og hlaPacketArrived i *OmnetCallbackInterface* benyttes for å skille mellom simulering av overføring av meldinger med og uten innhold.



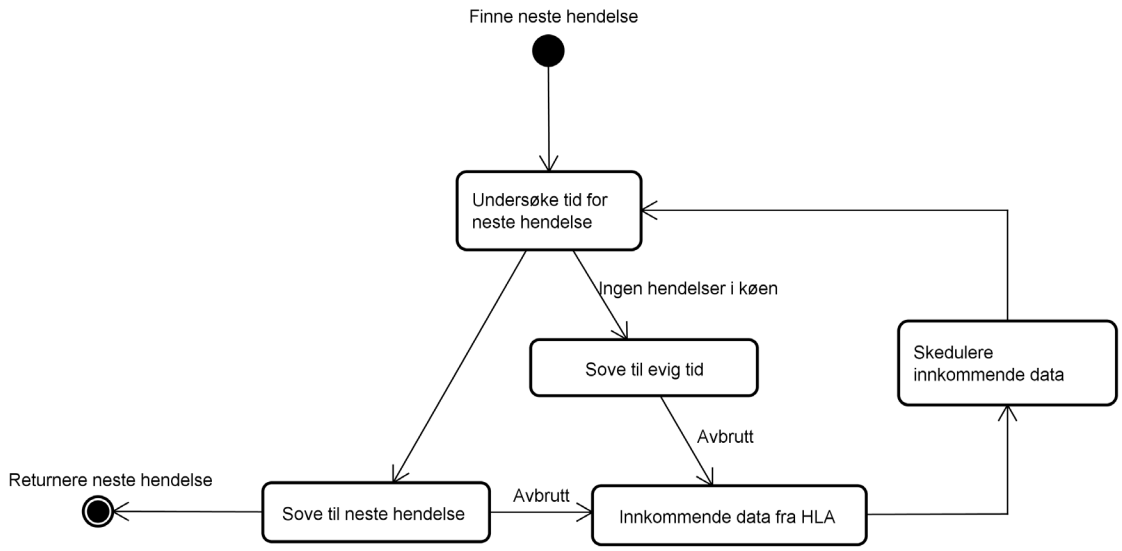
Figur 5.3 Klassediagram av C++-delen av OMNeT++ HLA-gateway.

5.1.2 Eksekvering av modellimplementasjon

Simuleringskjernen til OMNeT++ er modulær. De enkelte delene av kjernen kan ofte erstattes med små endringer i kode eller ved hjelp av konfigurasjonsfiler. Simuleringskjernen er bygget opp slik at valg av eksekveringskomponent[3] skjer i en konfigurasjonsfil (*omnet.ini*) som alltid følger med en simuleringsmodell. Dette skjer sømløst siden grensesnittet til eksekveringskomponenter er veldefinert. Fire funksjoner må implementeres og disse kalles av de andre delene av simuleringskjernen. Den egenutviklede klassen cHlaScheduler implementerer funksjonene og registrerer klassen hos OMNeT++ slik at den kan velges. Implementasjonen er gjort på følgende måte:

- **Klassens konstruktør:** Når klassen instansieres av OMNeT++ starter den JavaVM og instansierer Java-delen av HLA-grensesnittet. Instansiering av Java-delen tilknytter også OMNeT++ til en HLA-føderasjon.
- **Klassens destruktør:** Ved destruksjon avsluttes Java-delens tilknytning til RTI-en og alle ressurser frigis.
- **startRun()** kalles når OMNeT++ er ferdig med å laste simuleringsmodeller og er klar til å simulere. Her nullstilles alle tellere og tid. Ved Monte Carlo-simuleringer kalles denne funksjonen for hver kjøring.
- **endRun()** kalles når nye modeller skal lastes, OMNeT++ avsluttes eller etter hver kjøring i en Monte Carlo-simulering.
- **executionResumed()** kalles når simuleringer fortsetter etter en pause.
- **getNextEvent()** er hovedfunksjonen som velger hvilken diskret hendelse som skal eksekveres i neste tidssteg. I denne funksjonen sluses data fra RTI-en til stedfortredermodulen for applikasjonslaget mellom diskrete hendelser/tidssteg i OMNeT++. Tidsstegene synkroniseres til veggklokketid for å kjøre OMNeT++ modeller i sann tid. Aktivitetsdiagrammet i Figur 5.4 under illustrerer dette.

[3] På engelsk ”scheduler”.



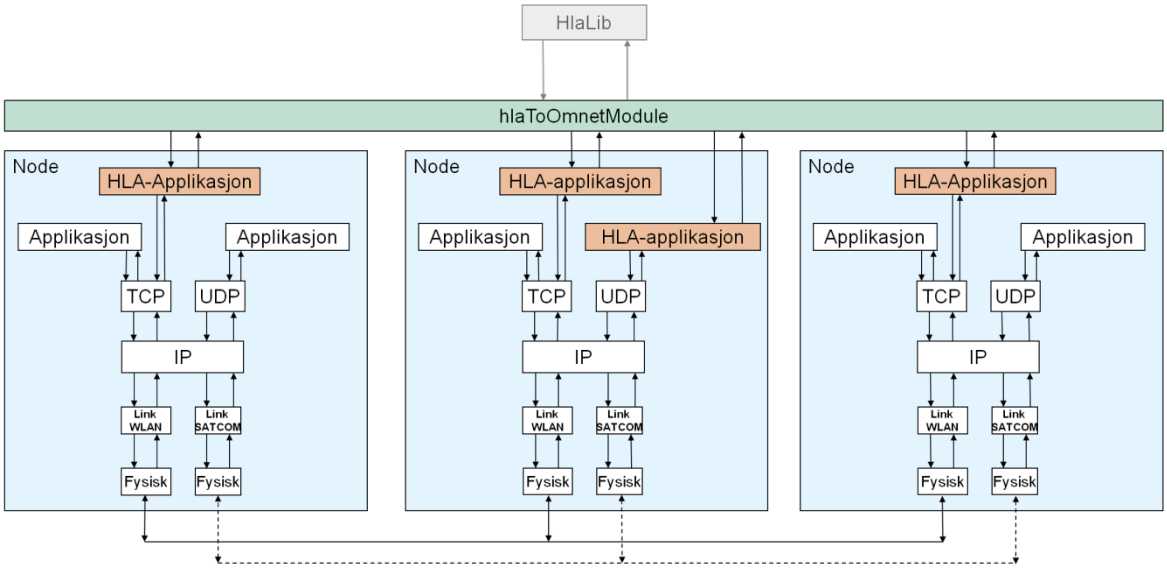
Figur 5.4 Aktivitetsdiagram for cHlaScheduler.

For å benytte OMNeT++ i tidsstyrte simuleringer må funksjonen getNextEvent() synkronisere sin egen tid til HLA i stedet for veggklokketid. Dette krever små endringer i kildekoden.

5.1.3 Grensesnitt mot OMNeT++-modeller

HLA-grensesnittet tilgjengeliggjøres for OMNeT++-modeller ved modulen hlaToOmnetModule. Denne må inkluderes i kommunikasjonsmodeller ved hjelp av modellspråket til OMNeT++. Modulen er litt annerledes enn andre moduler da den kommuniserer direkte med simuleringskjernen, men behandles av andre moduler og i modelleringsspråket på samme måte. hlaToOmnetModule registrer seg hos cHlaScheduler når den lastes av OMNeT++ sammen med resten av simuleringsmodellen. Hvis hlaToOmnetModule ikke benyttes, kan cHlaScheduler benyttes som en vanlig sanntidseksekveringsmodul i OMNeT++.

Figur 5.5 gir et eksempel på hvordan en koblet modell kan se ut. Oversettelse mellom *DataServicePDU*-interaksjoner og OMNeT++-modellens egen datapakketype skjer i stedfortredermodulen. Oversettelsen er modellspesifikk og må tilpasses for hver modell som skal benyttes i distribuert simulering basert på HLA.



Figur 5.5 Illustrasjon av hvordan en koblet modell i OMNeT++ med hlaToOmnetModule kan se ut.

Figuren illustrerer også godt hvorfor *DataServiceAccessPoint* peker på entiteten den er en del av og ikke til en radio. Det er nettverkslaget, her illustrert som IP, som bestemmer hvilken radio som skal benyttes.

5.1.4 Simulering av radioer

Simulering og publisering av radioer skjer i henhold til RPR FOM beskrevet i kapittel 3.2. Klassen NetSimGw publiserer og oppdaterer alle radiosendere og –mottakere. NetSimGw abonnerer på *BaseEntity*-objekter og oppdaterer posisjonene til radioene i OMNeT++ via OmnetCallbackInterface. Det bør på bakgrunn av scenario og simuleringsmodell settes egne grenseverdier for når oppdatering av posisjon bør sendes til OMNeT++. Avhengig av hvordan simuleringsmodeller er bygget opp kan oppdateringer utløse mange beregninger for både nettverkslaget (nettverkstopologi) og det fysiske laget (signal/støy-forhold). HLA-grensesnittet støtter dødregning av posisjon via programvarebiblioteket til HlaLib. På denne måten kan modeller i OMNeT++ selv be om posisjonsoppdateringer ved behov. Dette kan være en ressursbesparende løsning for simulerte noder med særdeles lite trafikk eller som har lite behov for oppdatering av posisjonsdata. Noder som ønsker de mest oppdaterte data innimellom oppdateringer fra HLA kan også be om oppdateringer via denne mekanismen, men kall mellom Java og C++ krever en del ressurser og bør holdes til et minimum. Dersom simulatoren benyttes i tidsstyrte simuleringer uten sanntidskrav kan det være en fordel alltid å be om dødregnede posisjoner.

NetSimGw abonnerer også på interaksjonen *AttributeChangeRequest* og endrer tilsvarende parametere i radiomodellene til OMNeT++ ved hjelp av OmnetCallbackInterface. Interaksjonen *AttributeChangeResult* gir tilbakemelding på hvorvidt endringen lyktes. Modellene av radioer i OMNeT++ oppdaterer publiserte radioparametere ved hjelp av hlaToOmnetModule. Det er mulig å sende ut interaksjonen *RawBinaryRadioSignal* for hvert hopp i f.eks. et ad hoc nettverk,

men det anbefales ikke pga. høyt ressursforbruk. Dette må konfigureres separat og kun benyttes ved behov.

5.1.5 Objektmodell og støttede RTI-tjenester

RPR FOM med tilleggene laget for OMNeT++ definerer hele føderasjonens objektmodell. OMNeT++ benytter imidlertid ikke hele RPR FOM når den inngår i en distribuert simulering. Den inngår en avtale med RTI-en om hvilke objekt- og interaksjonsklasser den skal motta og publisere. Denne avtalen spesifiseres i en Simulation Object Model (SOM). SOM-en er beskrevet i Tabell 5.1 og Tabell 5.2. Tjenestene OMNeT++ benytter er beskrevet i
Tabell 5.2 Interaksjonsklasser og attributter HLA-grensesnittet publiserer og abonnerer.

Objekter	Attributt	Publiserer/Abonnerer
RadioTransmitter	EntityIdentityIdentifyer	Publiserer
	HostObjectIdentifier	Publiserer
	RelativePosition	Publiserer
	AntennaPatternData	Publiserer
	CryptoSystem	Publiserer
	CryptographicMode	Publiserer
	EncryptionKeyIdentifier	Publiserer
	Frequency	Publiserer
	FrequencyBandwidth	Publiserer
	SpreadSpectrum	Publiserer
	StreamTag	Publiserer
	RadioIndex	Publiserer
	RadioInputSource	Publiserer
	RadioSystemType	Publiserer
	RFModulationSystemType	Publiserer
	RFModulationType	Publiserer
	TimeHopInUse	Publiserer
	TransmittedPower	Publiserer
	TransmitterOperationalStatus	Publiserer
	WorldLocation	Publiserer
RadioReceiver	EntityIdentityIdentifyer	Publiserer
	HostObjectIdentifier	Publiserer
	RelativePosition	Publiserer
	RadioIndex	Publiserer
	ReceivedPower	Publiserer
	ReceivedTransmitterIdentifier	Publiserer
	ReceiverOperationalStatus	Publiserer
BaseEntity	EntityIdentifyer	Abonnerer
	Spatial	Abonnerer
DataServiceAccessPoint	EntityIdentifyer	Abonnerer
	HostObjectIdentifier	Abonnerer
	RelativePosition	Abonnerer
	Role	Abonnerer
	ConsumerGroup	Abonnerer

Tabell 5.1 Klasser og attributter HLA-grensesnittet publiserer og abonnerer.

Interaksjoner	Attributt	Publiserer/Abonnerer
DataServicePDU	SourceAddress	Abonnerer og publiserer
	DestinationAddress	Abonnerer og publiserer
	Priority	Abonnerer og publiserer
	SerialNumber	Abonnerer og publiserer
	DataBuffer	Abonnerer og publiserer
	DataBufferLength	Abonnerer og publiserer
	MetaDataId	Abonnerer og publiserer
RawBinaryRadioSignal	DataRate	Publiserer
	HostRadioIndex	Publiserer
	SignalDataLength	Publiserer
	SignalData	Publiserer
	TacticalDataLinkType	Publiserer
	TDLMessageCount	Publiserer
CESDataServicePDU	SourceAddress	Abonnerer
	DestinationAddress	Abonnerer
	Priority	Abonnerer
	SerialNumber	Abonnerer
	DataBuffer	Abonnerer
	DataBufferLength	Abonnerer
	MetaDataId	Abonnerer
CESManagement	PduSource	Publiserer
	PduDestination	Publiserer
	SerialNumber	Publiserer
	Message	Publiserer
AttributeChangeRequest	ObjectIdentifiers	Abonnerer
	AttributeValueSet	Abonnerer
AttributeChangeResult	ObjectIdentifier	Publiserer
	AttributeChangeResult	Publiserer
	AttributeValueSet	Publiserer

Tabell 5.2 Interaksjonsklasser og attributter HLA-grensesnittet publiserer og abonnerer.

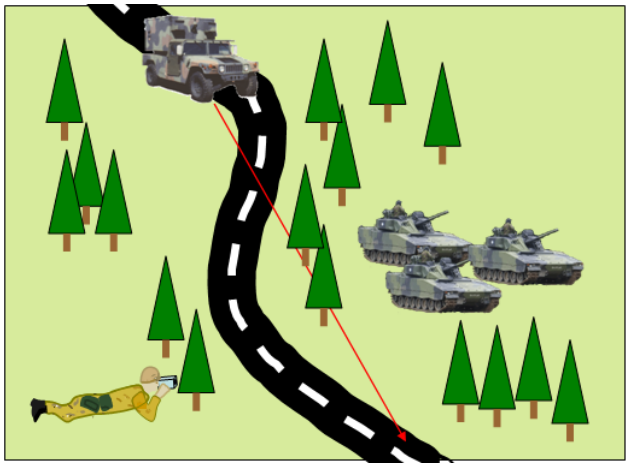
HLA-tjenestetype	Tjeneste
Federation Management	Create Federation Execution
	Destroy Federation Execution
	Join Federation Execution
	Resign Federation Execution
Declaration Management	Publish Object Class Attributes
	Subscribe Object Class Attributes
Object Management	Register Object Instance
	Discover Object Instance
	Delete Object Instance
	Remove Object Instance
	Update Attribute Values
	Reflect Attribute Values
	Request Attribute Value Update
	Provide Attribute Value Update

Tabell 5.3 HLA-tjenester benyttet av føderaten.

6 Testføderasjon og resultater

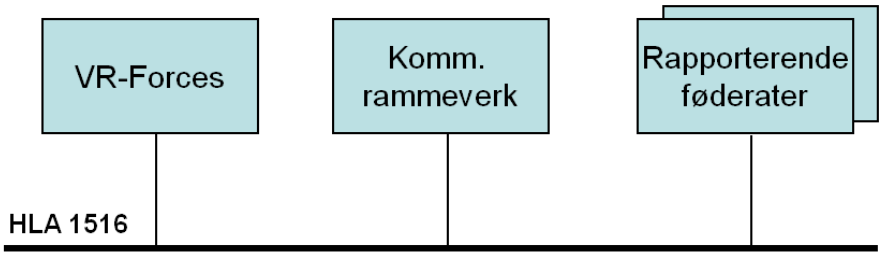
Dette kapittelet beskriver føderasjonen som ble satt sammen for å teste kommunikasjonssimuleringsrammeverket, scenarioet som ble benyttet og erfaringer fra forsøket. Et enkelt scenario ble definert for å opparbeide erfaringer med å benytte den nye objektmodellen for kommunikasjonssimulering, med det utviklede HLA-grensesnittet og med å klargjøre en eksisterende kommunikasjonsmodell for HLA-baserte simuleringer. Scenarioet, gjengitt i Figur 6.1, besto av ett kjøretøy som beveget seg langs en akse og rapporterte egen posisjon til egne styrker i området gjennom et mobilt ad hoc nett basert på IEEE 802.11. Så fort egne enheter kom innenfor rekkevidden til kjøretøyets radio ble posisjonsinformasjon sendt og de andres posisjoner ble lagt ved i bekreftelser på at posisjonsoppdateringene ble mottatt.

Scenarioet ble implementert ved å benytte komponenter hentet fra simuleringsinfrastrukturen utviklet i bl.a. FFI-prosjekt 1038 – Virtuelle teknologidemonstratorer, kommunikasjonsrammeverket og en egenutviklet rapporteringsføderat. Styrkenes bevegelse ble simulert av VR-Forces [6]. Kommunikasjonsrammeverket oppdaterte sin interne modell basert på posisjonsoppdateringer fra VR-Forces. Rapporteringsføderaten mottok også posisjonsdataene og satt disse inn i en *DataServicePDU*/*CESDataServicePDU* som ble sendt via kommunikasjonsrammeverket til intendert rapporteringsføderat. Kommunikasjonsrammeverket simulerte om posisjonsrapportene kom frem.



Figur 6.1 Testscenario.

Kommunikasjonsrammeverket benyttet en eksisterende pakke med kommunikasjonsmodeller basert på åpen kildekode som finnes tilgjengelig for OMNeT++ kalt INET [14]. INET består av mange modeller og dekker alle protokollagene i internettmodellen. Tre INET-modeller ble benyttet under forsøkene, IEEE 802.11 i ad hoc modus for Medium Access Control (MAC)-laget, Internet Protocol (IP) for nettverkslaget og User Datagram Protocol (UDP) for transportlaget.



Figur 6.2 Føderasjon benyttet for test.

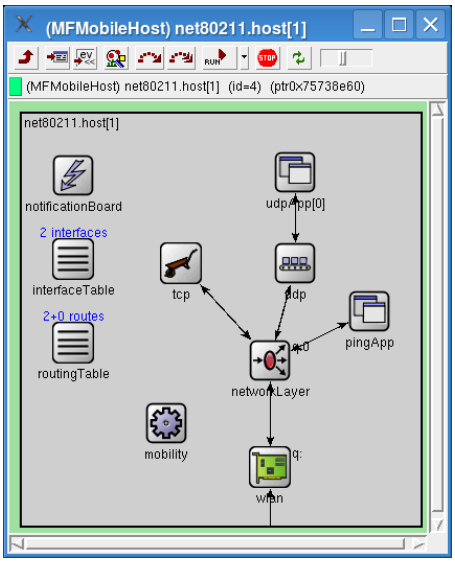
6.1 Tilpasning av eksisterende modell

Modellene i INET kan ikke uten videre benyttes i simuleringer basert på HLA. I hovedsak må det gjøres fire endringer.

- Attributtene til *DataServicePDU*-klassen må konverteres til og fra den interne representasjonen i INET. Dette gjøres ved å implementere en stedfortrederklasse for klientføderatene. I dette tilfellet i form av en UDP-applikasjon modellert som en enkel modul i OMNeT++ kalt *udpApp*. Modulen tar også hånd om å sende pakker inn i og ta imot fra INET-modellen.
- Den andre endringen er å bytte ut mobilitetsmodellen som benyttes av MAC-modellen. Denne erstattes med en stedfortredermodul som representerer Spatial-atributtet til den HLA-baserte entiteten radioen er tilknyttet og som konverterer posisjonene til det interne koordinatsystemet.

– Den tredje endringen besto i å gjøre MAC-, IP- og UDP-modellene i stand til å sende ut *CESManagement* interaksjoner til klientføderaten med beskjed om status på simuleringsforespørselen.
– Den fjerde endringen besto i å publisere *RadioTransmitter*- og *RadioReceiver*- objekter fra MAC-modellen. MAC-laget er kalt ”wlan” i Figur 6.3.

I skrivende stund har MAC-modellen ikke blitt endret slik at den kan publisere *RawBinaryRadioSignal*-interaksjoner for hvert simulerte radiosignal i OMNeT++. Det er heller ikke implementert støtte for *AttributeChangeRequest* og *AttributeChangeReply* interaksjoner.



Figur 6.3 Komponentdiagram av modellene benyttet for testen. Visualiseringen er gjort i OMNeT++.

6.2 Erfaringer

Erfaringer opparbeidet under integrasjonsarbeidet viste at å gjøre oversettelse mellom de HLA-baserte adressene til de interne IP- og UDP-baserte adressene i *udpApp* var meget enkelt så lenge det finnes et en-til-en-forhold mellom applikasjoner simulert i HLA og stedfortrederklasser simulert ved hjelp av *udpApp*-moduler. Hvor vanskelig det er å endre hvilke protokoller som benyttes til å simulere overføring av data mellom applikasjoner, avhenger hovedsakelig av kompleksiteten med å endre den interne koblingen mellom moduler i OMNeT++, ikke av forhold knyttet til objektmodellen. Å endre den interne koblingen er relativt enkelt. Dersom andre eksisterende OMNeT++-modeller skal benyttes, må i tillegg de fire stegene beskrevet over utføres for å endre disse slik at de kan benyttes med HLA-baserte simuleringer. Dette kan være ressurskrevende. Av de fire stegene med å integrere eksisterende modeller er det tredje det mest tidkrevende. All kildekode til modellene som skal integreres må leses gjennom for å identifisere hvor pakker kan bli kastet. Kildekoden må så skrives om slik at *CESManagement*-interaksjoner kan sendes til klientføderat.

Selve utviklingen av føderaten som rapporterer egenposisjon var enkel. Fokuset var på informasjonen som skulle utveksles, ikke på protokollagene som skulle få informasjonen frem til mottaker.

6.3 Begrensninger

HLA-grensesnittet til OMNeT++ implementerer ikke tidsstyringsmekanismene i HLA. Det vil si det er ment for sanntidssimulering og vil ikke nødvendigvis gi samme resultat for hver kjøring. For at simulering av ende-til-ende-forsinkelse skal gi mening, må minste ende-til-ende-forsinkelse være betydelig større enn summen av tiden det tar å sende forespørselen om å simulere, tiden det tar å simulere og tiden det tar å sende resultatet av simuleringen til mottaker. Typiske anvendelser vil være å simulere taktisk trådløs kommunikasjon som multirolleradio (MRR), Link-16 og liknende. Det vil si nettverk med lav båndbredde og relativt stor forsinkelse. Det krever ikke mye arbeid å implementere tidsstyring i HLA-grensesnittet og kan gjøres i fremtiden. Da vil alle typer nettverk kunne simuleres og resultatene vil være reproduserbare.

7 Diskusjon

Det er tidligere gjort liknende arbeid med å implementere kommunikasjonsrammeverk basert HLA og det finnes også kommersielle produkter. Disse beskrives i det følgende.

Kommersielle produkter som QualNet [7] og OPNET [15] har HLA-grensesnitt, men implementerer andre objektmodeller. QualNet for eksempel er del av Battle Lab Collaborative Simulation Environment [16] og benyttes også i Boeing Australia sitt Systems Analysis Laboratory [17]. OPNET har tidligere blitt benyttet i Simulation and Evaluation of Adaptive Mobile Large-Scale Network Systems (SEAMLSS) [18]. Georgia Tech Network Simulator (GTNetS) som er gitt ut som et åpen-kildekodeprosjekt [19] benyttes i Joint Virtual Network Centric Warfare Project [20] med HLA-utvidelser, og OMNeT++ med HLA-utvidelser benyttes ved Computational Research Center for Complex Systems [21] og i C2 Wind Tunnel prosjektet [22].

Artikler fra de ovenfornevnte prosjektene, samt andre omhandler i stor grad tekniske aspekter ved å integrere nettverkssimulatorer med HLA, tilgjengelighet av modeller og bruksområder for kommunikasjonsrammeverk i HLA-baserte føderasjoner. Det er lite informasjon tilgjengelig om hva slags objektmodeller som har blitt benyttet foruten noen få unntak. QualNet benytter RPR FOM 0.5 og 1.0 på en spesiell måte der nettverkssimulatoren er avhengig av at ikke-standardisert tilleggsinformasjon legges ved i *RadioSignal*-interaksjoner [23]. I [24] beskrives en spesiell FOM og en egen klient-tjener-modell, i [25] beskrives en implementasjon av en distribuert nettverkssimulator som benytter en egen objektmodell for å knytte sammen de ulike delene av nettverkssimulatoren, og i [26] beskrives en utvidelse av RPR FOM 2.0 draft 17 slik at det blir mulig å vite hvilke radiomottakere som mottar hvilke signaler.

Ingen av artiklene tar for seg hvordan en simuleringsmodell best deles opp mellom HLA-baserte klienter og kommunikasjonstjeneren på en måte som er uavhengig av nettverksteknologiene som skal simuleres, og som primært omhandler dataene som skal utveksles mellom klientene.

Implementasjonen av nettverkssimulatoren er basert på erfaringer fra en forsøksimplementasjon gjort på bakgrunn av [27]. I denne implementasjonen ble HLA-grensesnittet lagt til en enkel modul som kjørte periodisk. Våre erfaringer med denne løsningen er at det gir liten kontroll med når simuleringsforespørsler behandles og at det belaster simulatoren unødvendig da den hele tiden må sjekke om det har kommet nye forespørsler. Simulatoren har en en-trådet eksekveringsmekanisme som gjør at man heller ikke kan skille ut modulen som en egen prosess. Simulatoren vil risikere å havne i en ukjent tilstand hvis moduler interagerer med andre moduler på vilkårlige tidspunkter. Ved å plassere HLA-grensesnittet i simuleringskjernen har man mulighet til å kontrollere når simuleringsforespørsler behandles og en er i stand til å håndtere simuleringsforespørsler fortløpende.

Underveis i utviklingsarbeidet ble det publisert en artikkel [21] som beskriver implementasjon av en eksekveringsmekanisme på samme måte som ble valgt for den som er beskrevet i denne rapporten. Forskjellen ligger hovedsakelig i hvordan OMNeT++ er bundet sammen med HLA. I begge tilfeller benyttes egne biblioteker for å knytte simuleringer til HLA ved hjelp av et Java bibliotek. I artikkelen beskrives bindinger til deres bibliotek med XML-baserte fjernprosedyrekall. På FFI har det tidligere blitt utviklet et bibliotek kalt HlaLib [13]. Bindinger mot dette er implementert med Java Native Interface (JNI). Begge metodene har sine fordeler. XML-baserte fjernprosedyrekall er enklere å implementere, mens JNI gir en betydelig ytelsesforbedring som er viktig når simulatoren utsettes for høy belastning.

8 Videre arbeid

Arbeidet med å implementere HLA-grensesnittet og den nye objektmodellen for kommunikasjonssimulering har vært en del av det initielle arbeidet med å etablere en infrastruktur for å simulere kommunikasjon i HLA-baserte føderasjoner. I løpet av arbeidet har det kommet en del ideer til videre arbeid. Disse ideene kan deles inn i tre områder. Utvide den eksisterende simuleringsinfrastrukturen til å kunne interagere med kommunikasjonstjeneren, videre arbeid med kommunikasjonstjeneren og å gjennomføre en pilotcase der bruk av kommunikasjonstjeneren står sentralt. Disse områdene beskrives nærmere i det følgende.

På grunn av at det i løpet av arbeidet med å etablere kommunikasjonsrammeverket ble laget en ny objektmodell, har ingen av de eksisterende verktøyene støtte for denne. Videre arbeid vil kunne inkludere oppgradering av 3D-visualiseringskomponentene våre som for eksempel VR-Vantage til også å visualisere kommunikasjon eller mangel på kommunikasjon. I prosjektet har det også blitt utviklet en voice-over-IP (VOIP) applikasjon som benytter HLA for gruppekommunikasjon. Det kunne vært interessant å videreutvikle denne til å benytte nettverkssimulatoren for å degradere digital taletrafikk.

Kommunikasjonstjeneren kan også videreutvikles. Den finnes programvarebiblioteker utviklet med tanke på parallellsimulering med OMNeT++. Dersom ytelse blir et problem kan man utforske mulighetene for parallellsimulering, spesielt for det fysiske laget. Det er også interessant å utforske muligheter for å benytte flere kommunikasjonstjenere som hver simulerer forskjellige typer nett slik at ikke en tjener er ansvarlig for all kommunikasjonssimulering. I den forbindelse kan det være interessant og også opparbeide seg erfaring med HLA-tjenesten ”data distribution management” for å redusere belastningen på det fysiske nettverket.

Det bør gjøres grundigere undersøkelser av hvor stor last OMNeT++ tåler med de forskjellige måtene å interagere med simulatoren på og hvor stort påtrykk OMNeT++ tåler når for eksempel modellen av IEEE 802.11 skal eksekveres og simuleringen skal klare å kjøre i sanntid.

I arkitekturen som er benyttet for å utvikle HLA-grensenittet til OMNeT++ er lagt vekt på å lage et enkelt grensesnitt. Det ville vært interessant å se hvor generisk arkitekturen er ved å integrere nettverksimulatoren ns-3 [28], etterfølgeren til ns-2 [29] som er simulatoren som benyttes i brorparten av akademiske miljøer. Dette vil gi erfaring med å undersøke hvor generisk fremgangsmåten og objektmodellen er og hvilke hindringer som står i veien for å kunne skifte ut kommunikasjonstjenere som ”plug-and-play”-komponenter. Dersom man enkelt kan skifte kommunikasjonstjenere er det mulig å velge den med best modeller og best implementasjon for et gitt tilfelle og også bytte etter hvert som teknologi for simulering av kommunikasjon utvikler seg.

Testene som er gjort med kommunikasjonstjeneren er gjort i forbindelse med utvikling og er hovedsaklig utført for å verifisere grensesnittet og objektmodellen. Det ville vært interessant å gjennomføre en pilot der bruk av kommunikasjonstjeneren er sentralt. En slik studie bør gjennomføres i samarbeid med andre prosjekter på FFI som har behov for å gjennomføre en nærmere undersøkelse av nettverkskomponenter i et syntetisk miljø.

Referanser

[1] A. Varga, "The OMNeT++ Discrete Event Simulation System", in *Proceedings of the European Simulation Multiconference*, 2001, pp. 319-324.

[2] NATO NSA, "STANAG 4603 (Edition 1) - Modelling and Simulation Architecture Standards for Technical Interoperability: High Level Architecture (HLA)", July 2008.

[3] M. N. Nielsen and K. Brathen, "A Base Object Model for Federating a Communications Effects Server", in *Proceedings of the 2010 Spring Simulation Interoperability Workshop (SIW)*, 2010, 10S-SIW-020.

[4] SISO, "Real-time Platform Reference Federation Object Model (RPR FOM) 2.0 draft 17", SISO, 2003.

[5] ITU-T, "Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model", ITU-T, X.200, July 1994.

[6] MÄK Technologies, "VR-Forces 3.10.1", (2007). [Online]. Available: http://www.mak.com/products/vrforces.php.

[7] Scalable Network Technologies, "QualNet", (2010). [Online]. Available: http://www.scalable-networks.com/products/qualnet.

[8] M. N. Nielsen, "Modellering og implementasjon av IFF transponderføderat for bruk i simuleringer basert på High Level Architecture", Forsvarets forskningsinstitutt, FFI/NOTAT-2007/00040, 2007.

[9] SISO, "Enumeration and Bit Encoded Values for use with Protocols for Distributed Interactive Simulation Applications", SISO, SISO-REF-010-2006, 2006.

[10] MÄK Technologies, "MÄK Technologies website", (2006). [Online]. Available: http://www.mak.com/

[11] MÄK Technologies, "Plan View Display 2.10", (2007). [Online]. Available: http://www.mak.com/products/pvd.php.

[12] T. J. Berg, "oTWLAN - a tool to simulate tactical ad-hoc networks", Forsvarets forskningsinstitutt, FFI/RAPPORT-2009/00911, 2009.

[13] A. Alstad, "HlaLib 3.0 - User Guide", Forsvarets forskningsinstitutt, FFI-rapport 2010/00871, 2010.

[14] "INET Framework for OMNET++ 3.x", (20-10-2006). [Online]. Available: http://www.omnetpp.org/pmwiki/index.php?n=Main.INETFramework.

[15] OPNET Technologies, Inc, "OPNET", (2010). [Online]. Available: http://www.opnet.com.

[16] J. M. Moore, "Communication Effects Services for Distributed Simulations", in *Proceedings of the 2008 Spring Simulation Interoperability Workshop (SIW)*, 2008, 08S-SIW-055.

[17] R. Aplin and C. Kluge, "Communications In The Simulated Battlespace", in *Proceedings of the 2008 Simulation Technology and Training Conference (SimTecT)*, 2008.

[18] A. J. Steigerwald and L. P. Longtin, "Using the SEAMLSS Environment to Realistically Emulate Communications", in *Proceedings of the 1999 Spring Simulation Interoperability Workshop (SIW)*, 1999, 99S-SIW-177.

[19] G. F. Riley, "The Georgia Tech Network Simulator", in *ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research (MoMeTools)*, 2003.

[20] R. A. Pritchard, "Real-time Scalable Network Simulation", in *Proceedings of the 2009 Spring Simulation Interoperability Workshop (SIW)*, 2009, 09S-SIW-027.

[21] G. Emanuele, C. Gaetano, and T. Salvatore, "HLA-OMNET++: An HLA Compliant Network Simulator", in *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, IEEE Computer Society, 2008, pp. 319-321.

[22] H. Neema, G. Hemingway, J. Green, J. Sztipanovits, and G. Karsai, "Rapid Synthesis HLA-Based Heterogeneous Simulation: A Model-Based Integration Approach", ISIS Technical Report, T-08-0077, May 2009.

[23] Scalable Network Technologies, "Network-enabled Battlefield Simulation," 2009.

[24] A. J. Steigerwald, W. Silva, M. Cosby, and M. Hieb, "Real-World Communications Emulation in the HLA", in *Proceedings of the 1997 Spring Simulation Interoperability Workshop (SIW)*, 1997, 97S-SIW-199.

[25] S. M. Johnson, G. Wolff, and D. Evans, "A Distributed Communications Framework: Integrating Distributed Communications Models Into a High-Level Architecture Environment", in *Proceedings of the 2004 Fall Simulation Interoperability Workshop (Fall SIW 2004)*, 2004, 04F-SIW-009.

[26] G. Warren, "Simulating the Communications Network in Net Centric Operations", in *Proceedings of the 2005 Spring Simulation Interoperability Workshop (SIW)*, 2005, 05S-SIW-010.

[27] A. W. Malik, A. Basit, and S. A. Khan, "HLA Compliant Network Enabled Distributed Modeling and Simulation Infrastructure Design", in *Proceedings of the 4th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, Salzburg, Austria: World Scientific and Engineering Academy and Society (WSEAS), 2005.

[28] "The ns-3 network simulator", (2011). [Online]. Available: http://www.nsnam.org.

[29] "The Network Simulator - ns-2", (17-6-2008). [Online]. Available: http://www.isi.edu/nsnam/ns.

Forkortelser

CES	Communications Effects Server
FDN	Forsvarets digitale nett
FOM	Federation Object Model
GTNetS	Georgia Tech Network Simulator
HLA	High Level Architecture
IEEE	Institute of Electrical and Electronics Engineers
IFF	Identify Friend or Foe
IP	Internet Protocol
ISO	International Standards Organization
JNI	Java Native Interface
NS	Network Simulator
M&S	Modeling and Simulation
MAC	Medium Access Control
OMNeT++	Objective Modular Testbed in C++
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
RPR FOM	Real-time Platform Reference Federation Object Model
RTI	Run-time Infrastructure
SEAMLSS	Simulation and Evaluation of Adaptive Mobile Large-Scale Network Systems
SOM	Simulation Object Model
SOA	Service Oriented Architecture
TCP	Transmission Control Protocol
TDL	Taktisk datalink
UDP	User Datagram Protocol
VOIP	Voice-Over-IP
VTD	Virtuell teknologidemonstrator
XML	Extensible Markup Language