FFI RAPPORT

DEMONSTRATOR FOR BILDEOPPBYGGING

MEVASSVIK Ole Martin, HAFNOR Hilde, HANSEN Bjørn Jervell, LEERE Anton B, ROSE Kjell, SANDEN Helge, URDAHL Morten, VIKEN Kjell O, BRÅTHEN Karsten

FFI/RAPPORT-2003/02748

FFIE/730/134

Godkjent
Kjeller 1 august 2003

Vidar S Andersen
Forskningssjef

DEMONSTRATOR FOR BILDEOPPBYGGING

MEVASSVIK Ole Martin, HAFNOR Hilde, HANSEN Bjørn Jervell, LEERE Anton B, ROSE Kjell, SANDEN Helge, URDAHL Morten, VIKEN Kjell O, BRÅTHEN Karsten

FFI/RAPPORT-2003/02748

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25, 2027 Kjeller, Norge

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment

P O BOX 25
NO-2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-2003/02748	UNCLASSIFIED	105
1a) PROJECT REFERENCE	**2a) DECLASSIFICATION/DOWNGRADING SCHEDULE**	
FFIE/730/134	-	

4) TITLE

DEMONSTRATOR FOR BILDEOPPBYGGING

PICTURE PRODUCTION DEMONSTRATOR

5) NAMES OF AUTHOR(S) IN FULL (surname first)

MEVASSVIK Ole Martin, HAFNOR Hilde, HANSEN Bjørn Jervell, LEERE Anton B, ROSE Kjell, SANDEN Helge, URDAHL Morten, VIKEN Kjell O, BRÅTHEN KarstenHelge, URDAHL Morten, VIKEN Kjell O, BRÅTHEN KarstenHelge, URDAHL Morten,VIKEN Kjell, BRÅTHEN Karsten

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:	IN NORWEGIAN:
a) Command and control systems	a) Kommando og kontroll systemer
b) Information systems	b) Informasjonssystemer
c) Military communications	c) Militær kommunikasjon
d) Distributed processing	d) Distribuert databehandling
e)	e)

THESAURUS REFERENCE: **INSPEC**

8) ABSTRACT

The report describes a technology demonstrator for maritime command and control, more specifically maritime picture production. The demonstrator includes legacy command and control information systems and new decision support components using state of the art middleware technology for integration. The network utilizes IP and includes both satellite and HF radio communication links. An HLA simulation environment stimulates the demonstrator. Several types of large screen display present the maritime situation picture, including a 3D stereoscopic table display.

9) DATE	AUTHORIZED BY	POSITION
	This page only	
1. August 2003	Vidar S Andersen	Director of Research

ISBN-82-464-0741-4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

INNHOLD

DEMONSTRATOR FOR BILDEOPPBYGGING

1 INNLEDNING

Prosjekt 730 – KKI-Sjø hadde som målsetning å gi anbefalinger om fremtidig utvikling av ledelsessystemer for maritime operasjoner. Det skulle komme fram til anbefalingene gjennom analyse av fremtidige teknologiske muligheter og sammenligning av levetidskostnader og operativ effekt av ulike alternativer. Anbefalingene ble gitt på konseptuelt nivå som hovedideer som skulle være styrende for fremtidig utvikling og en konkret systemplan med tilhørende levetidkostnader (31) basert på det anbefalt konseptet. Fra disse ble det også utarbeidet i alt 14 anbefalinger (35) som hadde en form som Forsvaret lettere kunne ta stilling, utformet som beslutninger som var nødvendig for å gå i den ønskede retningen. Disse overordnede anbefalingene ble også benyttet når resultatene av prosjektet skulle formidles og gjøres kjent i Forsvaret.

I tillegg til analysene som la grunnlaget for anbefalingene var det også en målsetting for prosjektet å kunne vise hvordan noen av de viktigste anbefalingene kunne realiseres med bruk av tilgjengelige teknologi. Prosjektet skulle derfor utvikle en demonstrator i et laboratoriemiljø. Det området som demonstratoren skulle dekke innenfor ledelse av maritime operasjoner, var bildeproduksjon. Bildeproduksjon ble valgt fordi det legger grunnlaget for all videre ledelse og er dermed et svært viktig funksjonsområde.

Rapporten gir en forholdsvis komplett beskrivelse av den demonstratoren for maritim bildeoppbygging som ble utviklet i prosjekt 730 – KKI-Sjø. Den dekker både bruksforhold og tekniske sider med vekt på det siste. Kapittel 2 beskriver mer konkret målsettingene med demonstratoren. Kapittel 3 gir en kort oversikt over de viktigste delene av demonstratoren. Kapittel 4 gir bakgrunn for valg av arkitektur for demonstratoren og beskriver den valgte arkitekturen, mens kapittel 5 beskiver mellomvareteknologiene som ble benyttet for å realisere arkitekturen. I kapittel 6 beskrives brukerfunksjonene mens kapittel 7 gir en detaljert beskrivelse av tjenestekomponentene som demonstratoren består av. Kapittel 8 beskriver hvordan de Maritime Command and Control Information System (MCCIS) og Maria kartsystem er integrert i demonstratoren. Kommunikasjonssystemet i demonstratoren blir presentert i kapittel 9 med vekt på HF- og satellittkommunikasjon. I kapittel beskrives simuleringsomgivelsen. Rapporten avsluttes med en beskrivelse av laboratoriet som ble etablert for demonstratoren, en beskrivelse av hvordan demonstratoren kan kjøres og en konklusjon.

2 MÅLSETTING

Målsettingen med demonstratoren var altså konkret å vise (i stedenfor å beskrive) hvordan det var mulig å realisere noen av de viktigste anbefalingene som var fremkommet gjennom

analyser. På denne måten var demonstratoren et supplement til analysene. Mer konkret skulle følgende demonstreres:

- Ny funksjonalitet for å støtte de som bygger situasjonsbildet
- Hvordan moderne objekt-orientert programvarearkitektur kunne integrere både fremtidige moduler og funksjonalitet i dagens kommando og kontroll informasjonssystemer.
- Bruk av kommersielle kommunikasjonsprotokoller, spesielt Internet Protocol (IP) kunne benyttes om en gjennomgående protokoll også for smalbåndet mobilt samband
- Bruk av nye typer brukergrensesnittutstyr og presentasjonsformater for situasjonsbildet.

Demonstratoren omfatter derfor i hovedsak anbefalinger innenfor områdene kommunikasjons- og informasjonssystemer. Innenfor ny funksjonalitet for bildeoppbygging ble det utviklet støtte for klassifikasjonsfusjon og en funksjon for fusjon av planer, rapporter og radardata. Elementer av et nettverksbasert kommunikasjonssystem ble også demonstrert, også for HF-kommunikasjon. Innenfor presentasjon av situasjonsbildet ble det lagt vekt på bruk av ulike storskjermer. Rapporten gir mer informasjon om disse områdene og hvordan de ble realisert i demonstratoren.

Et eget laboratorium ble etablert ved FFI for demonstratoren. For å kunne stimulere demonstratoren med data var det behov for å utvikle en simuleringsomgivelse. Den tok utgangspunkt i et tidligere utviklet simuleringsprogram for analyse av sensorsystemer og bildeoppbyggingsarkitektur kalt Sensorsim som ble videreutviklet til å omfatte distribuert bildeoppbygging. Simuleringsomgivelsen er også beskrevet senere i rapporten.

3 OVERSIKT

Demonstratoren avgrenser seg til produksjon av Recognized Maritime Picture (RMP) på ett sted, det være seg på et hovedkvarter eller for en styrke. Alle sensordata og delbilder som er grunnlaget for å produsere RMP og defineres som en del av simuleringsomgivelsen. Demonstratoren har fokus på funksjonalitet for å støtte RMP-operatøren i å sette sammen informasjon fra ulike kilder til ett konsistent bilde samt presentere dette bildet for operativ bruker. For å utvikle denne funksjonaliteten valgte vi å benytte en komponentbasert framgangsmåte i form av løst koblede og forholdsvis uavhengige komponenter (tilpasset en overordnet systemarkitektur). Disse komponentene ”samarbeider” så for å tilby den ønskede funksjonaliteten i bildeproduksjonssystemet. Demonstratorens komponenter kan grovt deles inn i tre kategorier: Arven, komponenter utviklet for ny (komponentbasert) arkitektur og tjenester som er grensesnittet mot simuleringsomgivelsen.

Arven i demonstratoren består av Maritime Command and Control Information System (MCCIS) og Maria. Formålet med å benytte disse var å utnytte allerede eksisterende funksjonalitet (tids- og kostnadsbesparelse i utvikling) og for å få erfaring med hvor krevende det var å gjenbruke eksisterende systemer i et komponentbasert informasjonssystem. I MCCIS

er deler av trackhåndteringsfunksjonaliteten gjort tilgjengelig for eksterne komponenter gjennom mellomvareteknologi. Det er også sett på hvordan en kan synkronisere trackdatabasen i MCCIS med eksterne databaser, noe som er et viktig moment hvis en på sikt skal gjøre seg uavhengig av MCCIS og gå gradvis over til nye løsninger. For å kunne utnytte eksisterende kartfunksjonalitet ble deler av Maria gjort eksternt tilgjengelig ved at den produserer rasterbilder av de geografiske områdene operatøren ønsker å se på skjermen.

Komponentene som er utviklet av prosjektet består i hovedsak av et operatørgrensesnitt, en RMP trackdatabase, en komponent for mottak av data fra eksterne enheter samt to komponenter for å støtte klassifikasjon av overflatetrack. Operatørgrensesnittet inkluderer kartfunksjonalitet, enkel trackhåndtering og grensesnitt mot klassifikasjonskomponentene. Støtte til track-klassifikasjon inkluderer funksjonalitet som ikke finnes i eksisterende RMP produksjonssystem (MCCIS). Funksjonaliteten som er utviklet består av en komponent som utfører fusjon av klassifikasjonsrapporter fra ulike sensorer til ett entydig klassifikasjonsestimat, og en komponent som klassifiserer radartrack ved å fusjonere forhåndsdefinerte ruteplaner med egenrapporter og radartrack. For mottak av data fra eksterne kilder er det utviklet en rapporteringstjeneste. Det ble også utviklet en variant av rapporteringstjenesten for overføring av trackdata med bruk av internettprotokoll over HF-modemer.

Simuleringsomgivelsens funksjon er å simulere et på forhånd oppsatt hendelsesforløp med tilhørende plattformer, sensorer, bildeproduksjonsnoder og sende trackdata til demonstratoren. Simuleringsomgivelsen består grovt sett av et simuleringsprogram som er utviklet av prosjektet (SensorSim), et program for å styre simuleringsomgivelsen og komponenter som opptrer som sensortjenester i demonstratoren. SensorSim simulerer hele bildeproduksjonsprosessen, d v s deteksjon, tracking, enkel datafusjon og informasjonsutveksling. SensorSim er satt opp slik at det simulerer sensorer og bildeoppbyggingsprosessene som pågår i de enhetene som er definert til å ligge utenfor selve demonstratoren.

For å kunne få erfaring med bruk av internettløsninger for samband mot mobile enheter over store avstander inngår to HF-modemer og en simulator for satellittsamband. Demonstratoren består også av utstyr for presentasjon og interaksjon med situasjonsbildet. Det er utviklet to operatørposisjoner som har en tre-skjerm løsning bestående av tre 20” LCD-skjermer som gir ett stort sømløst skjermbilde for operatøren. Presentasjonsutstyret omfatter også en interaktiv 50” storskjermsløsning (SmartBoard), et 60” interaktivt bord samt tre LCD-prosjektører. Totalt sett består demonstratoren av til sammen 14 datamaskiner i et heterogent miljø med både PC og UNIX-løsninger.

3.1 Komponenter

Dette kapitlet inneholder en oversikt over komponentene som demonstratoren er bygget opp av, se Figur 3-1 for en oversikt.

RMP-komponenten er den sentrale trackdatabasen som holder RMP-et. Databasen i RMP-komponenten er synkronisert og konsistent med trackdatabasen i MCCIS. RMP-komponenten

er kun en fasade som andre komponenter benytter for å hente ut eller skrive trackdata. RMP-komponenten benytter en annen Jini-tjeneste, *Outrigger,* for lagring av dataene.

Rapporteringstjenesten (*Reporter*) gjør data fra en enkelt sensor eller fusjonsnode tilgjengelig i Control of Agent Based Systems (CoABS) Grid. En Reporter kan settes opp til å sende ut data for en enkelt plattform (f eks UAV), fra en styrke (f eks en fregatt) eller et operasjonsrom (f eks Sjøops). En kan ha et vilkårlig antall rapporteringstjenester som hver sender data fra ulike kilder.

HF Sender er en spesialkomponent som henter data fra en Reporter og komprimerer disse for å sende dem over HF. Enheter som rapporterer over simulert satellittkommunikasjon benytter Reporter direkte.

Mottakstjenesten (*Receiver*) abonnerer på RMP-data fra alle tilgjengelige Reporter og HF-Sender tjenester og oppdaterer RMP-komponenten.

Klassifikasjonskomponenten (*Classifier*) lytter på alle endringer i RMP modellen relatert til klassifikasjonsinformasjon og produserer et klassifikasjonsestimat av alle trackene i RMP modellen. Classifier har også et grensesnitt mot brukeren slik at denne kan se på hvilke data som ligger til grunn for klassifikasjonsestimatet, samt se på sannsynlighetsfordelingen for klassifikasjonen.

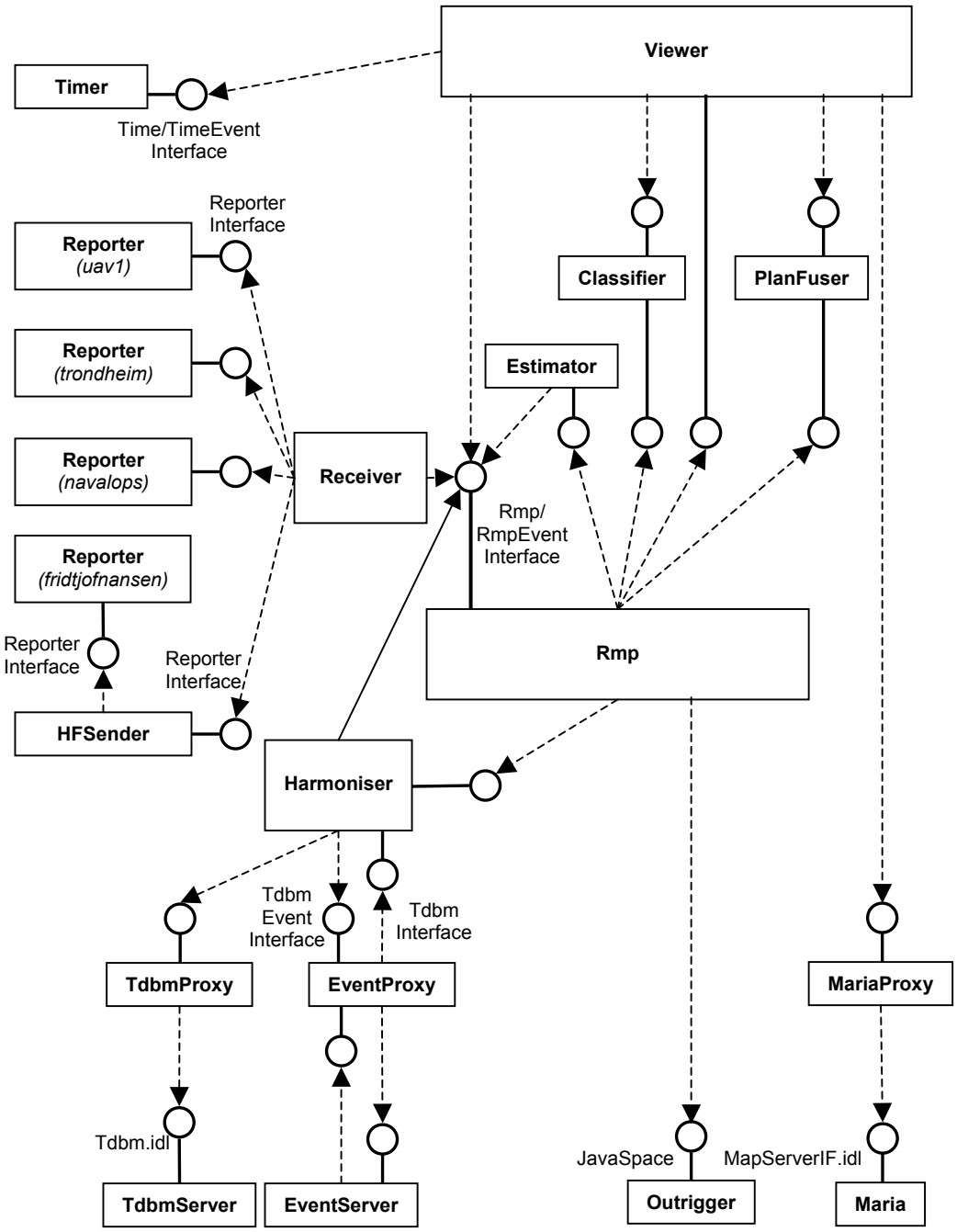
Ruteplankomponenten (*PlanFuser)* har som oppgave å klassifisere radar-track på grunnlag av informasjon fra ruteplaner og egenrapporter mottatt fra fartøy. Ruteplankomponenten har også et brukergrensesnitt slik at brukeren kan få presentert ruteinformasjon og se hvilke radar-track som er assosiert med aktive ruteplaner.

*Time*r er en tidstjeneste som holder rede på ”demonstratortiden”, og sørger samtidig for at tiden i demonstratoren og simuleringsomgivelsen er den samme. En komponent kan spørre tidstjenesten hva klokka er, eller be om å få periodiske tidsoppdateringer.

Harmoniser sørger for at RMP-komponentens og MCCIS’s trackdatabase er konsistente. Hvis en endring skjer i RMP-komponenten sørger den for at MCCIS blir oppdatert og vice-versa.

Estimator vedlikeholder et oppdatert posisjonsestimat for radar-track ut fra de rapportene som er tilgjengelige. Versjonen som er implementert benytter kun siste mottatte rapport som posisjonsestimat.

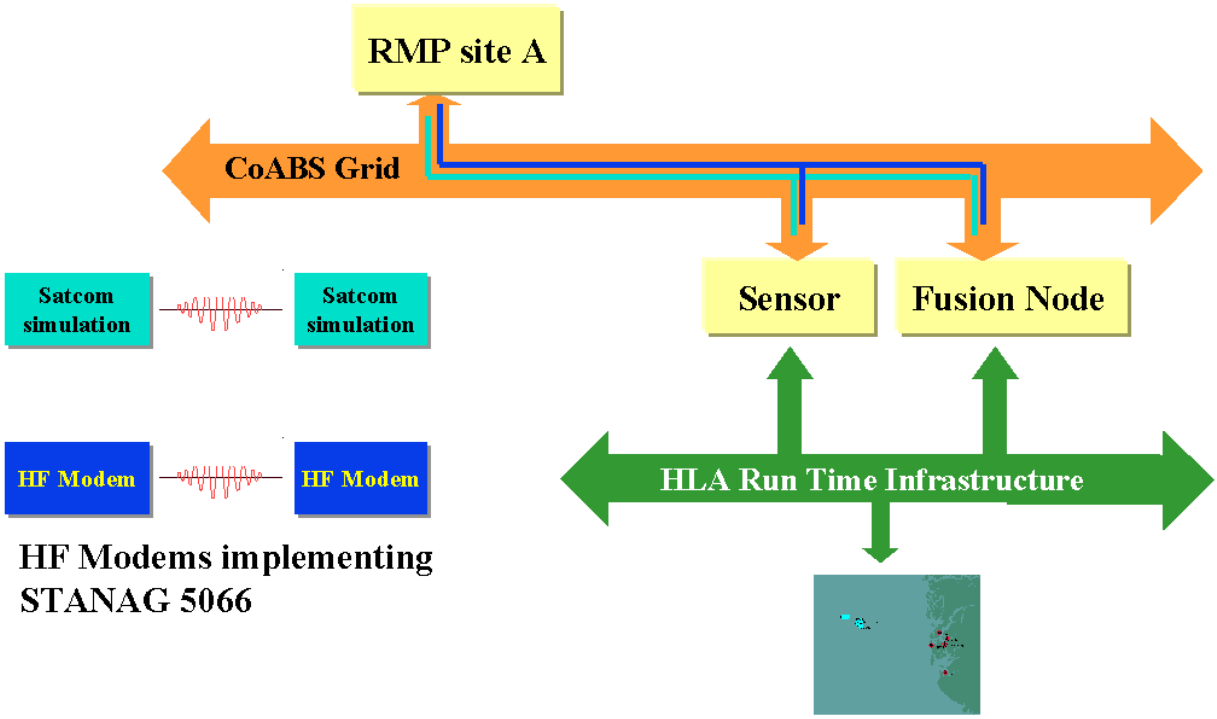
Viewer er brukergrensesnittet for RMP-operatøren. Dette er ikke en komponent i den forstand at den ikke publiserer noen tjenester for andre komponenter. Den er en applikasjon som benytter seg av ulike komponenter for å tilby brukeren et sett med tjenester.



Figur 3-1 Komponentdiagram for demonstratoren

3.2 Mottak av data fra simuleringsomgivelsen

Figur 3-2 viser skisse som illustrerer dataflyten ved mottak av data over HF eller simulert satellittkommunikasjon. Sensorer og fusjonsnoder kan registrere seg som rapporteringstjenester i CoABS Grid. Disse tjenestene videreformidler data produsert i simuleringsomgivelsen (SensorSim), og gjør disse tilgjengelig for mottakstjenesten som videre oppdaterer RMP-komponenten. Et vilkårlig antall sensorer og fusjonsnoder kan settes opp til å rapportere sine data med en egen rapporteringstjeneste for å simulere et enkelt "Sensor Grid". I figuren vises også omgivelsessimulatoren og dens forbindelse med resten av demonstratoren.



Figur 3-2 Simuleringsomgivelsens grensesnitt mot demonstratoren

4 DEMONSTRATOR ARKITEKTUR

Ett av målene med demonstratoren var å demonstrere deler av den fleksibilitet kommersielle objektorienterte programvarearkitekturer gir for produksjon og distribusjon av et bilde, og hvordan funksjonalitet i eksisterende system kan integreres i en slik arkitektur.

4.1 Valg av arkitekturløsning

Utformingen av demonstratorarkitekturen er et resultat av prosjektets innledende konsept-arbeider og arkitekturanalyser (10). Basert på krav utledet fra konseptarbeidene for fremtidig ledelsessystem (31) ble det i konseptarbeidet for informasjonssystemer utledet ønskede egenskaper for et fremtidig K2IS (32). Eksempler på slike egenskaper er tilpasningsevne,

åpenhet, modifiserbarhet, gjenbrukbarhet, feiltoleranse og skalerbarhet for å håndtere krav til f eks kosteffektivitet, interoperabilitet, fleksibilitet, robusthet og heterogenitet i et fremtidig K2IS. Et konsept for et informasjonssystem beskriver i hovedsak hvordan et informasjons-system tenkes realisert for å oppnå ønskede egenskaper. I demonstratoren har vi ikke tatt mål av oss å realisere alle ønskede K2IS-egenskaper, men like fullt skulle demonstratoren være et "proof of concept" for noen av disse. I prosjektets arkitekturstudie var en del av aktiviteten å vurdere og skissere valg av arkitekturer, metodikker og teknologier for utvikling og vedlikehold av et fremtidig K2IS. I arbeidet med bildedemonstratoren valgte man tre-lags arkitekturmodell for distribuerte løsninger. Det ble også benyttet en komponentorientert tilnærming som passet naturlig inn i en arkitekturtankegang bl a ved å utvikle veldefinerte grensesnittspesifikasjoner som tilrettelegger for iterativ og inkrementell utvikling (utviklingsstrategi for systembygging).

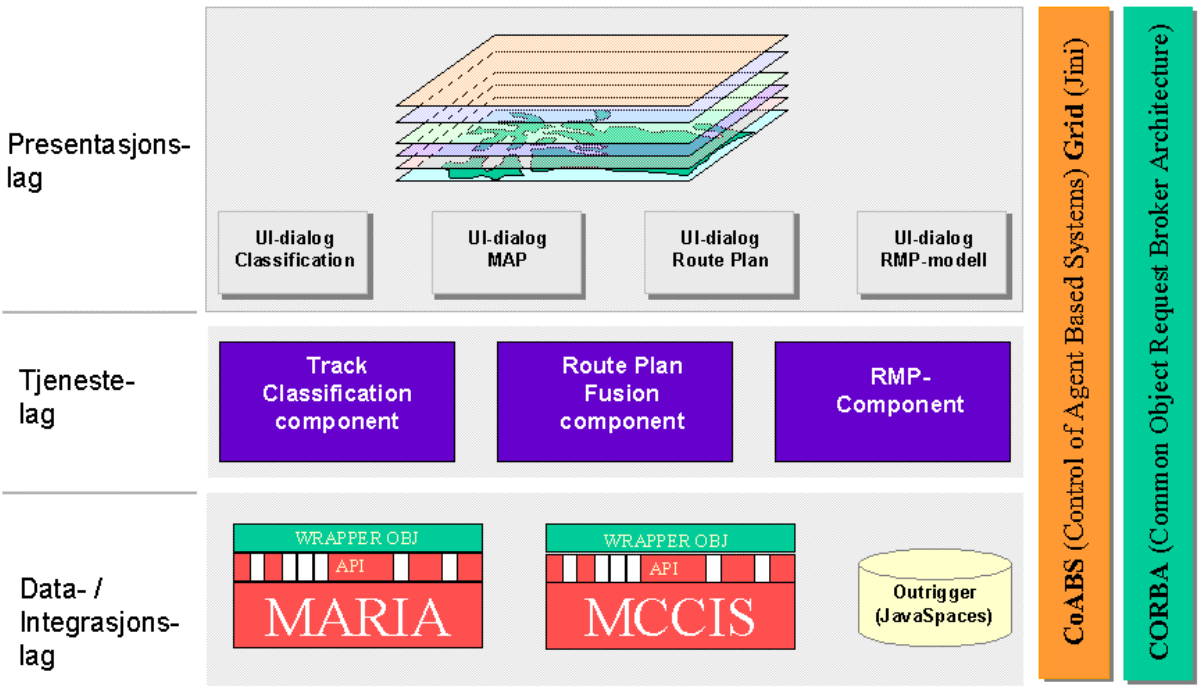
4.2 Overordnet arkitektur

Arkitekturen beskrevet i Figur 4-1 gir en overordnet beskrivelse av hvordan demonstratoren er satt sammen (struktur). Demonstratoren er bygget over en distribuert og komponentbasert programvarearkitektur som skiller demonstratoren inn i tre logisk atskilte lag: presentasjons-, tjeneste- og data-/integrasjonslag (tilsvarende objektmodellen i J2EE (33)). Generelt sett omfatter presentasjonslaget i en tre-lags modell enkle applikasjoner og klienter som kjører på maskiner knyttet til en arbeidsposisjon og benyttes av sluttbrukeren for å hente ut informasjon og interaksjon med systemet. En applikasjon på presentasjonslaget benytter seg av tjenester fra tjenestelaget gjennom veldefinerte grensesnitt. Med en applikasjon mener vi den samlede funksjonaliteten som tilbys en bruker, og denne vil typisk benytte seg av flere komponenter. Tjenestelaget omfatter gjenbrukbare tjenester som kan benyttes av flere applikasjoner. Data- og integrasjonslaget omfatter persistent datalagring (f eks databaser) og integrasjon mot eksisterende systemer (systemarven).

I demonstratoren er noen av elementene i disse lagene fysisk distribuert, og innenfor hvert av lagene (med unntak av presentasjonslaget) forekommer det distribusjon gjennom infrastrukturtjenester som tilbys av to COTS mellomvareteknologier: CORBA og Jini (kjernen i CoABS Grid). I demonstratoren bruker vi CORBA for integrering av funksjonalitet fra eksisterende systemer, og CoABS Grid for dynamisk integrasjon av tjenester i en bildeoppbyggingsnode samt informasjonsutveksling mellom geografisk distribuerte bildeoppbyggingsnoder. En beskrivelse av CORBA og CoABS Grid gis i kapittel 4.3.

Ved bruk av mellomvareteknologi støtter demonstratorarkitekturen:

- integrasjon av utvalgt funksjonalitet fra eksisterende systemer,
- distribusjon av programvarekomponenter på heterogene plattformer lokalisert i en bildeoppbyggingsnode,
- informasjonsutveksling mellom geografisk distribuerte bildeoppbyggingsnoder,
- samt dynamisk oppkobling av tjenester.



Figur 4-1 Demonstratoren er bygget over en 3-lags komponentbasert arkitekturmodell

De mest sentrale tjenestene for bildeoppbygging er innkapslet som programvarekomponenter på tjenestelaget, hver med et veldefinert grensesnitt som til sammen utgjør grensesnitt-spesifikasjonen for demonstratoren. Det er verdt å merke seg at komponentinndelingen ikke har vært gjenstand for en utførlig analyse, men har kommet delvis som er resultat av arven og de komponenter som allerede var tilgjengelige, d v s det er ikke gitt at vår inndeling er den mest optimale. Demonstratoren er bygget opp ved hjelp av en kombinasjon av egenutviklede og eksisterende gjenbrukbare komponenter. I figuren befinner de egenutviklete komponentene seg på tjenestelaget mens de eksisterende komponentene (MARIA og MCCIS) befinner seg på data - og integrasjonslaget. Nedenfor beskrives kort hver av lagene i demonstratoren.

Presentasjonslaget i demonstratoren omfatter en enkel Java-applikasjon. Applikasjonen aksesserer komponentene på tjenestelaget og benyttes av operatør for å få presentert situasjonsbildet og for å ha interaksjon med systemet. Presentasjonsapplikasjonen i demonstratoren omfatter spesifikk funksjonsløsning for en type brukerrolle (bildeoppbygger) i systemet. Operatørgrensesnittet er objektorientert og dialogbasert og blir nærmere beskrevet i kapittel 6.

Tjenestelaget i demonstratoren omfatter de sentrale tjenestekomponentene for bildeproduksjon. I demonstratoren er det utviklet tre slike tjenestekomponenter: En klassifikasjonskomponent, en ruteplankomponent og en RMP-komponent. Klassifikasjonskomponenten utfører fusjon av klassifikasjonsdeklarasjoner, ruteplankomponenten fusjonerer egenrapporter og ruteplaner, mens RMP-komponenten er den komponenten som til enhver tid holder det oppdaterte situasjonsbildet. Klassifikasjon- og ruteplankomponenten holder ingen data. De aktiveres kun ved hendelser fra RMP-komponenten eller ved interaksjon fra bruker (altså ikke persistente).

RMP-komponenten trenger persistent datalagring for de data som til enhver tid inngår i situasjonsbildet. Disse dataene lagres ikke i selve RMP-komponenten men i Outrigger (som avbildes på data- og integrasjonslaget i figuren). Outrigger er en implementasjon av JavaSpaces, som er en tjeneste definert i Jini for (persistent) lagring og utveksling av data mellom tjenester. RMP-komponenten og JavaSpaces, klassifikasjon- og ruteplankomponenten blir nærmere beskrevet i kapittel 7.

De to eksisterende systemene som er integrert i demonstratoren er MCCIS og MARIA. Funksjonaliteten fra MCCIS omfatter deler av "Track Management"-funksjonen . "Track Management"-funksjonen er gjort tilgjengelig gjennom et CORBA-grensesnitt utviklet i prosjektet med støtte og API fra INRI UK (nå Northrop Grumman IT Europe). MARIA er kartdelen av SjøTAS, og er gjort tilgjengelig gjennom et CORBA-grensesnitt spesifisert av prosjektet og utviklet av Teleplan. Gjennom CORBA-grensesnittet bruker demonstratoren MARIA som en karttjener. Ved å benytte CORBA er denne funksjonaliteten ikke bare gjort tilgjengelig for demonstratoren men også for hvilken som helst annen ekstern komponent, fra en hvilken som helst plattform og språk. En nærmere beskrivelse av integrasjonen av MCCIS er gitt i kapittel 8.3 og en beskrivelse av MARIA karttjener er gitt i kapittel 8.1.

Ved å benytte en tre-lagsarkitektur har demonstratoren fått en åpen arkitektur for integrasjon av eksisterende systemfunksjonalitet samt ny funksjonalitet. I en tre-lagsmodell, som vist i Figur 4-1, kan f eks presentasjonslaget utvides til å omfatte flere applikasjoner som kan utføre andre typer prosesseringer i systemet. Et eksempel på dette kan være dersom det skal introduseres en ny brukerrolle med andre behov enn det bildebyggerrollen har, eller gjenbruke en eller flere komponenter på tjenestelaget til andre funksjoner. Da kan man f eks utvikle og legge til en ny applikasjon som tilbyr de nye brukerne spesifikke løsninger til støtte for de nye behovene. Ved slike typer utvidelser av systemet ville det være naturlig å dele presentasjonslaget i to logiske deler: et applikasjonslag som tilbyr en samling funksjonalitet og et tynt klientlag (f eks en web-leser). Videre vil en utvidelse av tjenestetilbudet kunne la seg gjøre ved å legge til nye tjenestekomponenter som f eks representerer planleggingsfunksjonen i virksomheten, eller et annet sentralt virksomhetsområde. I tilfelle man ønsket å utvide demonstratoren til også å omfatte funksjonalitet fra f eks NORCCIS II eller andre systemer kan det f eks gjøres på tilsvarende måte som for MCCIS og MARIA. Komponenter fra forskjellige leverandører kan på denne måten flettes sammen til større funksjonelle enheter, der hver enhet isolert sett kan endres og påbygges.
Det er i prinsippet ingen begrensning på hvor mange eksisterende systemer, databaser, komponenter og applikasjoner som kan inkluderes inn i en slik distribuert arkitektur. Men til større en distribuert løsning blir, til mer øker kompleksiteten og dermed krav til drift og vedlikehold.

4.3 Infrastrukturtjenester

Kravet om et distribuert og komponentbasert system krever en rekke forskjellige system- og infrastrukturtjenester for å oppnå distribusjon og kommunikasjon mellom komponentene i hvert av lagene og mellom komponentene mellom lagene i arkitekturen (som vist i Figur 4-1). Derfor

er demonstratorarkitekturen bygget over to mellomvareteknologier: CORBA og CoABS Grid. I demonstratoren har vi brukt CORBA som infrastruktur for integrering av funksjonalitet fra eksisterende systemer, og CoABS Grid som infrastruktur for dynamisk integrasjon av tjenester i en bildeoppbyggingsnode samt informasjonsutveksling mellom geografisk distribuerte bildeoppbyggingsnoder.

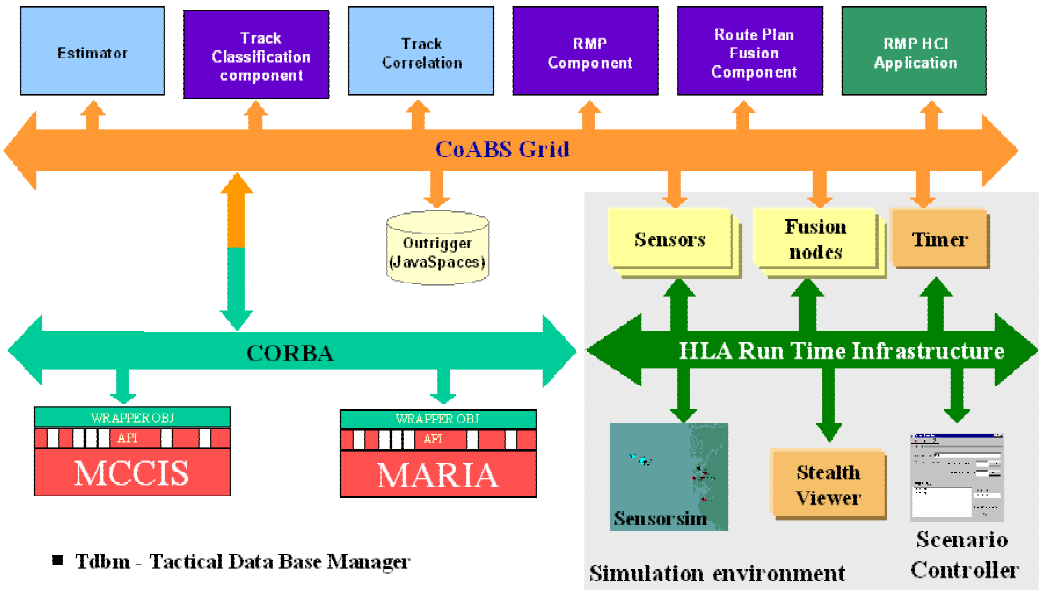
CoABS Grid (12) er en infrastruktur som kan integrere agentsystemer, objektorienterte systemer og eksisterende systemer på en enkel måte. CoABS Grid er utviklet i DARPA-programmet Control of Agent Based Systems (CoABS), og er bygget over Jini Connection Technology (Sun) (13). Jini er et rammeverk for bygging av robuste, skalerbare distribuerte systemer og har attraktive egenskaper for utvikling av framtidige K2IS-er som må være mer tilpasningsdyktige og fleksible m h t variasjon i oppgaver og styrkesammensetning. CoABS Grid tillater dynamisk integrasjon av tjenester fra komponentbaserte og eksisterende systemer. Den tillater også fjernaksess til tjenester (f eks bruk av Java RMI), meldingsbasert informasjonsutveksling (f eks XML-meldinger) og bygging av stabile distribuerte systemer over ustabile nettverk. CoABS Grid støtter dynamisk oppkobling av tjenester med ulike transportprotokoller og media, transparent for komponenter og applikasjoner.

CORBA (34) er en åpen industristandard som tillater integrasjon av komponenter fra heterogene språk, operativsystemer, datamaskiner og nettverksprotokoller. CORBA tilbyr en rekke generiske tjenester som f eks transaksjon-, database- og feilhåndteringstjenester. CORBA brukes i demonstratoren for å gjøre intern funksjonalitet i eksisterende systemer tilgjengelig for eksterne systemer og komponenter, uavhengig av plattform og språk.

4.4 Teknisk arkitektur

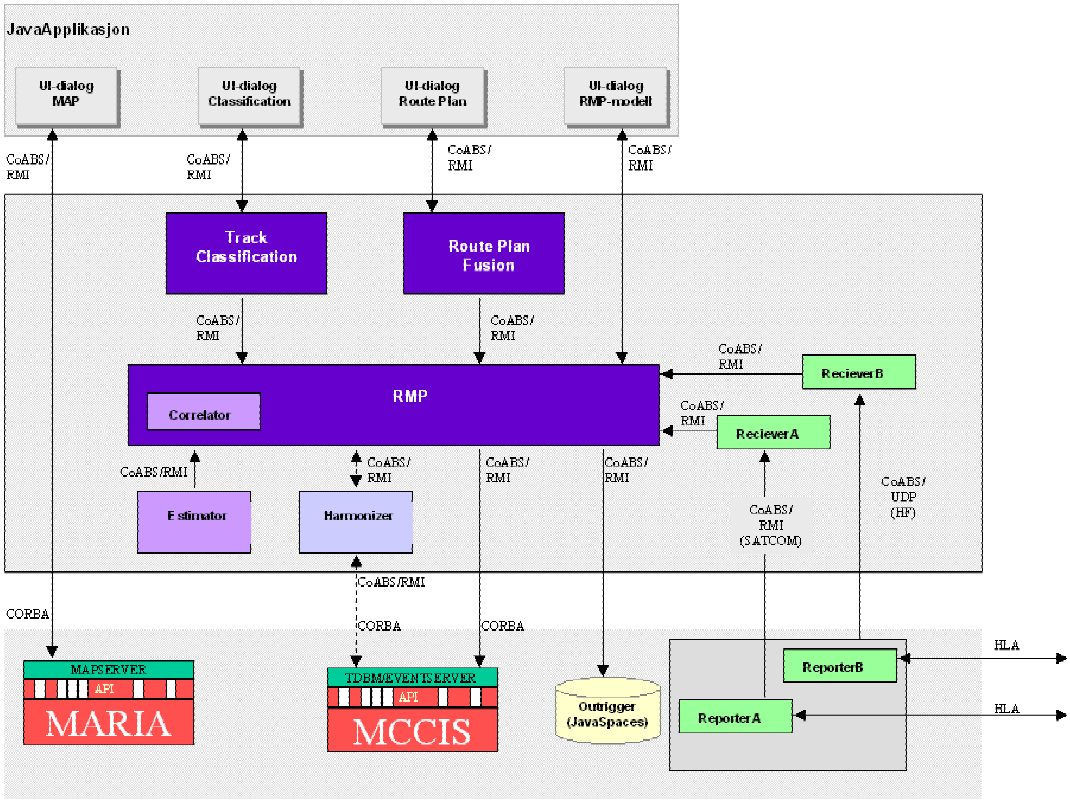
Figur 4-2 og Figur 4-3 gir en oversikt over teknisk arkitektur for demonstratoren, inklusive simuleringsomgivelsen. Hensikten med Figur 4-2 er å vise hvordan de to infrastrukturene binder de ulike elementene i demonstratoren sammen. Øverst i figuren gjenfinnes de tre tjeneste-komponentene (som omtalt tidligere) sammen med to "internkomponenter" (i lyseblått) og presentasjonsapplikasjonen (i grønt). Internkomponentene er ikke tjenestekomponenter slik de er definert i en tre-lagsmodell, men tilbyr intern funksjonalitet for tjenestekomponentene (d v s de benyttes kun av en komponent). Simuleringsomgivelsen er bygget over High Level Architecture (HLA), som er en arkitektur for distribuert simulering, og er nærmere beskrevet i kapittel 10.

Figur 4-3 viser sammenheng og kommunikasjon mellom komponentene, både i og mellom lagene i tre-lagsarkitekturen.



Figur 4-2 Teknisk arkitektur

I Jini er konnektorteknologien (protokollen klienten benytter for å snakke med tjenesten) usynlig for klienten. Denne egenskapen har vi utnyttet ved at rapporteringstjenesten kan benytte to konnektorteknologier. Figur 4-3 viser hvilken infrastruktur og konnektorteknologi som er benyttet mellom de ulike delene av demonstratoren.



Figur 4-3 Sammenheng og kommunikasjon mellom komponenter

5 BRUK AV MELLOMVARETEKNOLOGI

5.1 CORBA

Common Object Request Broker Architecture (CORBA) er en spesifikasjon utviklet av Object Management Group (OMG) og adresserer interoperabilitet mellom forskjellige maskinvare-platformer og programmeringsspråk i distribuerte objektsystemer. CORBA har også mekanismer for støtte til interoperabilitet mellom forskjellige mellomvareteknologier (10).

CORBA spesifiserer mekanismer som gjør at objekter kan sende anrop og motta svar på en transparent måte. Hovedkonseptene til CORBA omfatter:

- Spesifikasjon av mellomvaretjenester som brukes av applikasjoner.
- Enhver applikasjon kan være en klient, tjener eller begge deler.
- Enhver interaksjon mellom objekter skjer gjennom forespørsler.
- Støtte av statisk og dynamisk binding mellom objekter.
- Et grensesnittdefinisjonsspråk: Et grensesnitt representerer kontrakten mellom klient- og tjenerapplikasjonen. Grensesnittdefinisjonen viser parametrene som overføres og en unik grensesnittidentifikator. Grensesnitt spesifiseres ved hjelp av Interface Description Language (IDL) definert av OMG.
- Et objektadapter som styrer aktivering og deaktivering av objekter, og generering av objektreferanser.

CORBA har stor støtte i industrien og en rekke leverandører leverer CORBA-løsninger. Teknologien benyttes i større og større grad i f m oppgradering og utvikling av kampsystemer. OMG har etablert en Command, Control, Communications, Computers and Information (C4I) "Domain Special Interest Group" (C4I DSIG) for å etablere et grensesnitt og rammeverk for C4I-domenet. OMGs C4I DSIG har startet en prosess for å spesifisere domenespesifikke grensesnitt for systemer som inngår i C4I-domenet med utgangspunkt i krav, standarder, produkter og pågående arbeider innen dette området.

Målsettingen for C4I DSIG er å:

- arbeide innad i OMG for å utvide CORBA-teknologien der det er nødvendig for å oppfylle kravene fra C4I-miljøet
- arbeide for anvendelse av CORBA-teknologien i C4I-miljøet
- arbeide sammen med andre OMG-grupper for å sikre at OMG-definerte teknologier dekker C4I-kravene
- gi retningslinjer og støtte til C4I-miljøene m h p bruk av OMG-teknologier

- definere et CORBA-basert rammeverk for C4I-domene.

5.2 Jini og CoABS Grid

Hensikten med Jini (13) er å sette grupper av utstyrs- og programvare-komponenter sammen til et enkelt (ukomplisert), dynamisk distribuert system. Det meste sentrale konseptet i Jini er en *tjeneste* (service). En tjeneste er en entitet som kan benyttes av en person, et program eller en annen tjeneste. Et Jini-system består av tjenester som settes sammen for utførelsen av en bestemt oppgave. Jini's dynamiske natur gjør det mulig å legge til nye tjenester etter behov og fjerne tjenester som det ikke lenger er behov for, og gjør det også mulig å håndtere en situasjon hvor tjenester kommer og går av andre årsaker. Dette gjør f eks at Jini tillater at tjenester og brukerne av tjenestene flytter seg i nettverket. Jini har heller ikke som forutsetning at tjenestene er knyttet sammen via et perfekt sambandsnettverk. Disse egenskapene gjør Jini til en meget interessant teknologi for militære anvendelser.

En av målsetningene til Jini er å gjøre det enkelt å knytte sammen tjenester og få disse til å operere sammen. Jini-teknologien er utviklet med interoperabilitet i distribuerte systemer som en målsetning, og vil kunne bidra til interoperabilitet i militære systemer.

I demonstratoren har vi ikke benyttet Jini direkte, men en infrastruktur som heter CoABS (Control of Agent Based Systems) Grid (12) som er en overbygning over Jini. CoABS Grid er utviklet i et DARPA-prosjekt og har som formål å muliggjøre interoperabilitet av agentsystemer (agenter er spesielle programvarekomponenter).

CoABS Grid har en Grid Manager hvor en kan kontrollere infrastrukturtjenestene Jini benytter:

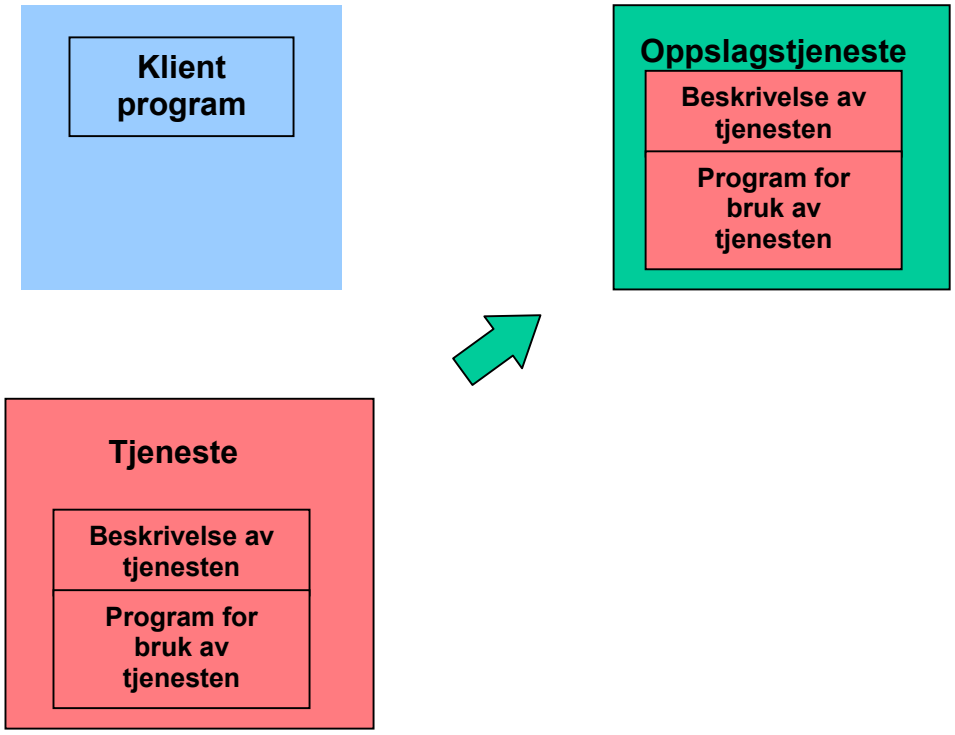
- web-server,
- RMI-server og
- og tjeneste for registrering og oppslag av Jini-tjenester (LookUp Service – LUS).

I Grid Manager kan en også se hvilke tjenester som er registrert. I tillegg er det definert en standard måte å beskrive tjenestene på samt hjelpeklasser for registrering, søk og oppkobling mot Jini-tjenestene. Det finnes også en "postboks" for meldingsbasert informasjonsoverføring. Vi har ikke benyttet all funksjonalitet i CoABS Grid, men primært benyttet den for å forenkle bruken av Jini.

5.2.1 Hovedprinsippet for Jini

Alle som har plundret med å installere utstyr og programvare på en PC, vet at det må finnes en enklere måte å gjøre dette på. Jini's løsning er en dynamisk installasjon av tjenester. Som et eksempel kan en benytte installasjon av en printer i et nettverk. Når en kobler en Jini-basert printer til et nettverk, vil den først lokalisere en oppslagstjeneste og registrere seg i den. Den vil registrere seg med en beskrivelse av printeren som for eksempel sort-hvitt eller farge,

papirformat og papirtyper og andre egenskaper som hvor printeren er plassert. I tillegg vil den registrere et program for bruk av tjenesten (dette programmet blir heretter kalt tjenestens proxy-objekt). Når en bruker skal skrive ut et dokument, søkes det i oppslagstjenesten etter en printer med de ønskede egenskaper.

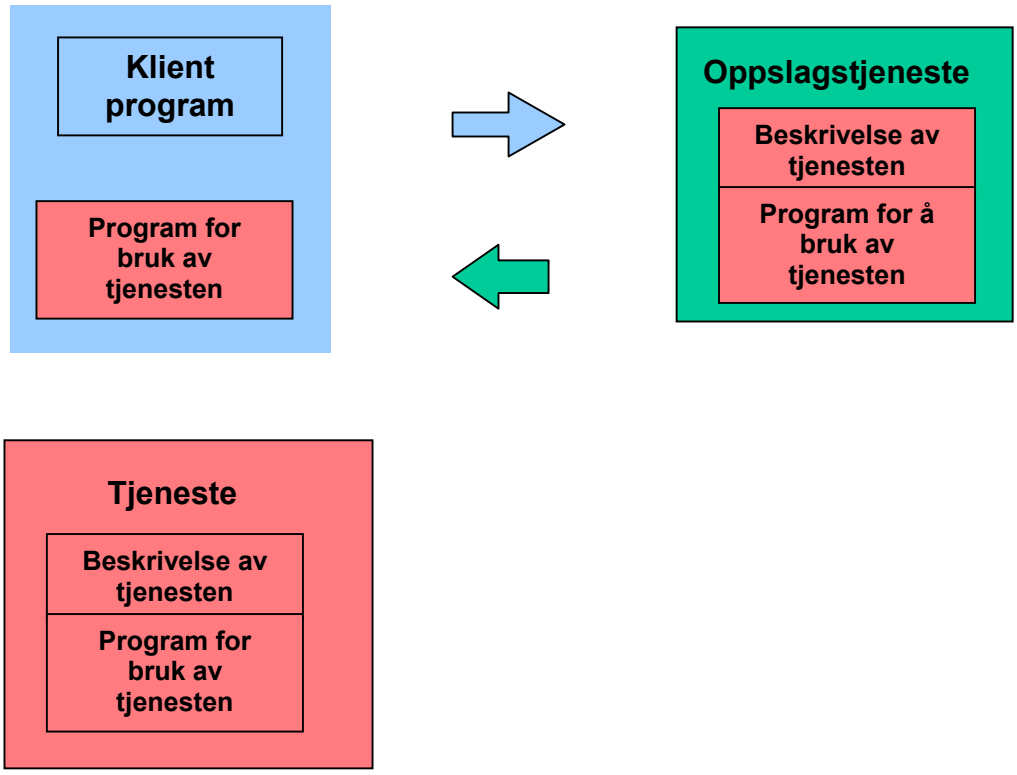


Figur 5-1 *Tjenesten registrerer seg i en oppslagstjeneste med en beskrivelse og et program for bruk av tjenesten (proxy-objekt)*

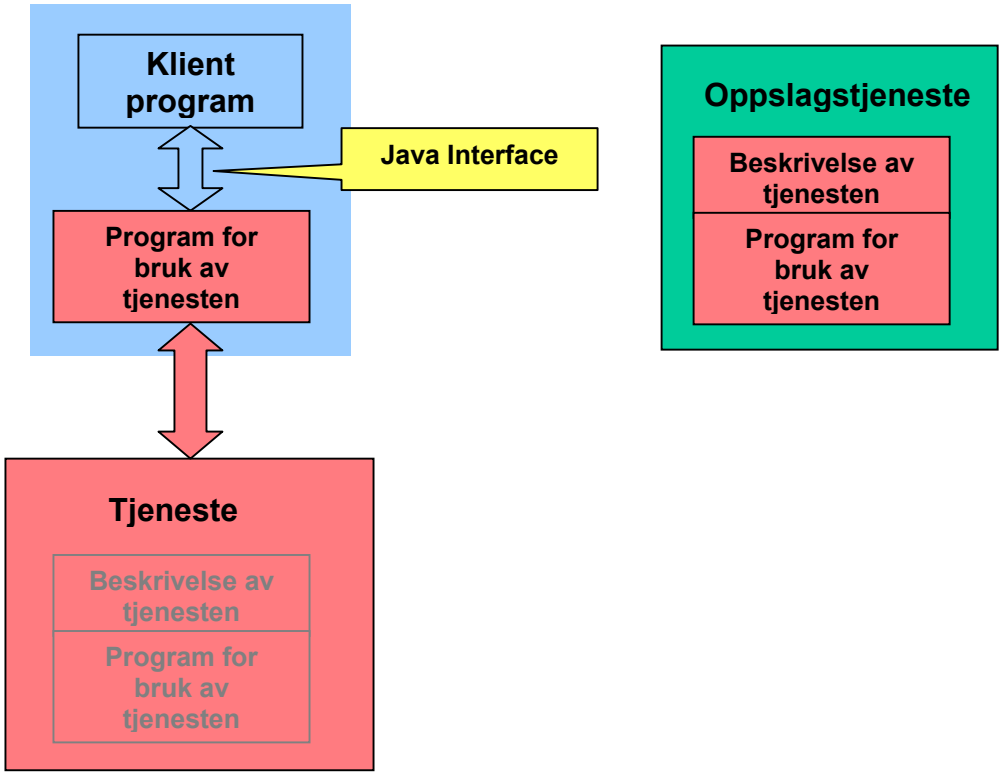
Deretter lastes programmet for bruk av tjenesten (proxy-objektet) ned, og utskriften kan begynne. Med Jini er det ikke nødvendig med en forhåndsinstallasjon av printeren, den installeres dynamisk når det er behov for tjenesten. Det eneste som må være bestemt på forhånd er grensesnittet mellom brukerprogrammet og programmet for bruk av tjenesten. Dette grensesnittet er definert som et Java-grensesnitt (Java Interface).

Rekkefølgen når man tar i bruk en tjeneste blir da:

1. Tjenesten registrerer seg i en oppslagstjeneste med en beskrivelse og et program for bruk av tjenesten (proxy-objekt).
2. En bruker søker i oppslagstjenesten etter tjenester med ønsket beskrivelse og grensesnitt.
3. Proxy-objektet for bruk av den valgte tjenesten lastes ned.
4. Proxy-objektet benyttes ved bruk av tjenesten.



Figur 5-2 *Klientprogrammet søker i oppslagstjenesten etter tjenester med ønsket beskrivelse og grensesnitt. Programmet for bruk av av tjenesten (proxy-objekt) lastes ned*

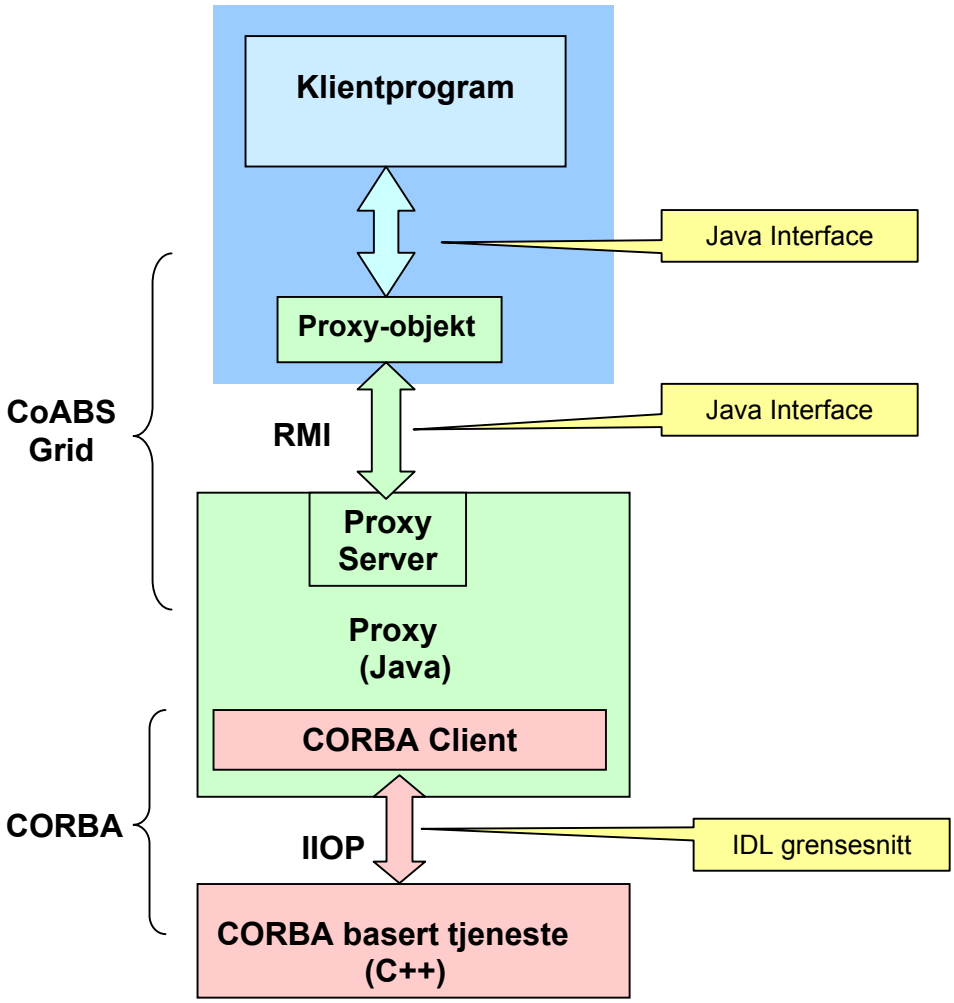


Figur 5-3 *Det nedlastede programmet benyttes ved bruk av tjenesten*

En bruker av en tjeneste aksesserer proxy-objektet som et lokalt objekt. Transporten mellom brukeren og tjenesten er implementert i proxy-objektet og er derved skjult for brukeren. Java Remote Method Invocation (RMI) er mye brukt mellom proxy-objektet og tjenesten, men en hvilken som helst transport-protokoll kan benyttes. I demonstratoren benytter de fleste proxy-objektene RMI for aksess av tjenestene. Unntaket er HF-Sender som benytter UDP (User Datagram Protocol) for effektiv transport over HF.

5.2.2 Aksess av CORBA-baserte tjenester fra CoABSGrid

Mangelfull interoperabilitet mellom Java RMI over CORBA (RMI-IIOP) og Visibroker, gjorde det nødvendig å utvikle proxy-tjenester for aksess av CORBA-baserte tjenester som vist i Figur 5-4. Fordelen med denne løsningen er at den gir et klart skille mellom CoABS Grid og CORBA, mens ulempen er at det innføres et forsinkende mellom-lag.



Figur 5-4 Demonstratoren benytter proxy-tjenester for aksess av CORBA-baserte tjenester

5.2.3 Hjelpeklasser for oppkobling mot tjenestene

For å gjøre det enklere å koble seg opp mot tjenestene er det laget hjelpeklasser som ligger i de samme programpakkene som tjenesten. F eks for å koble seg opp mot RMP-komponenten kan man benytte hjelpeklassen mil.ffi.rmp.RmpClient og koble seg opp med følgende kode:

```
RmpClient    rmp_client    = new RmpClient();
RmpInterface rmp           = rmp_client.lookup_rmp_service();
```

Andre hjelpeklasser er mil.ffi.rmp.RmpEventClient, mil.ffi.tdbmproxy.TdbmClient, mil.ffi.eventproxy.TdbmEventClient og mil.ffi.mariaproxy.MapClient.

6 BRUKERGRENSESNITT

Operatørgrensesnittet (Viewer) er implementert som en ren Java-applikasjon basert på Java Swing biblioteket. Grensesnittet er objektorientert og dialogbasert og utformet i h t "Java Look and Feel" - Metal style.

Alle brukerdialoger og menyer er objektorientert, som betyr at operatøren kan klikke på objekter i situasjonsbildet for å få opp menyer eller dialoger som tilbyr operatøren informasjon eller funksjonalitet avhengig av valgt objekt. I dette kapittelet skisseres de viktigste momentene i f m brukergrensesnittutviklingen for demonstratoren.

6.1 Brukergrensesnitt og operatørfunksjonalitet

Det var opprinnelig definert to brukerroller i demonstratoren: Personer som bygger bildet (bildeoppbygger) og beslutningstakere (team) som benytter bilde som beslutningsgrunnlag. Disse to brukerrollene vil ha en god del felles funksjonalitet, men også funksjonalitet tilpasset de to rollene. Demonstratoren omfatter i hovedsak felles funksjonalitet og funksjonalitet for bildeoppbygger.

6.1.1 Hovedvindu

Hovedvinduet i Viewer (Figur 6-1) består av en "Menu bar" (hovedmeny), en "Tool bar" (verktøymeny) og et kartvindu. Nederst i vinduet er det lagt inn en "Status bar" (statuslinje) som viser dato og klokkeslett for simuleringen og posisjonen til muspekeren.

Hovedmenyen inneholder standard menyer som "File", "Edit", "View" og "Help". I tillegg er det lagt inn følgende menyer:

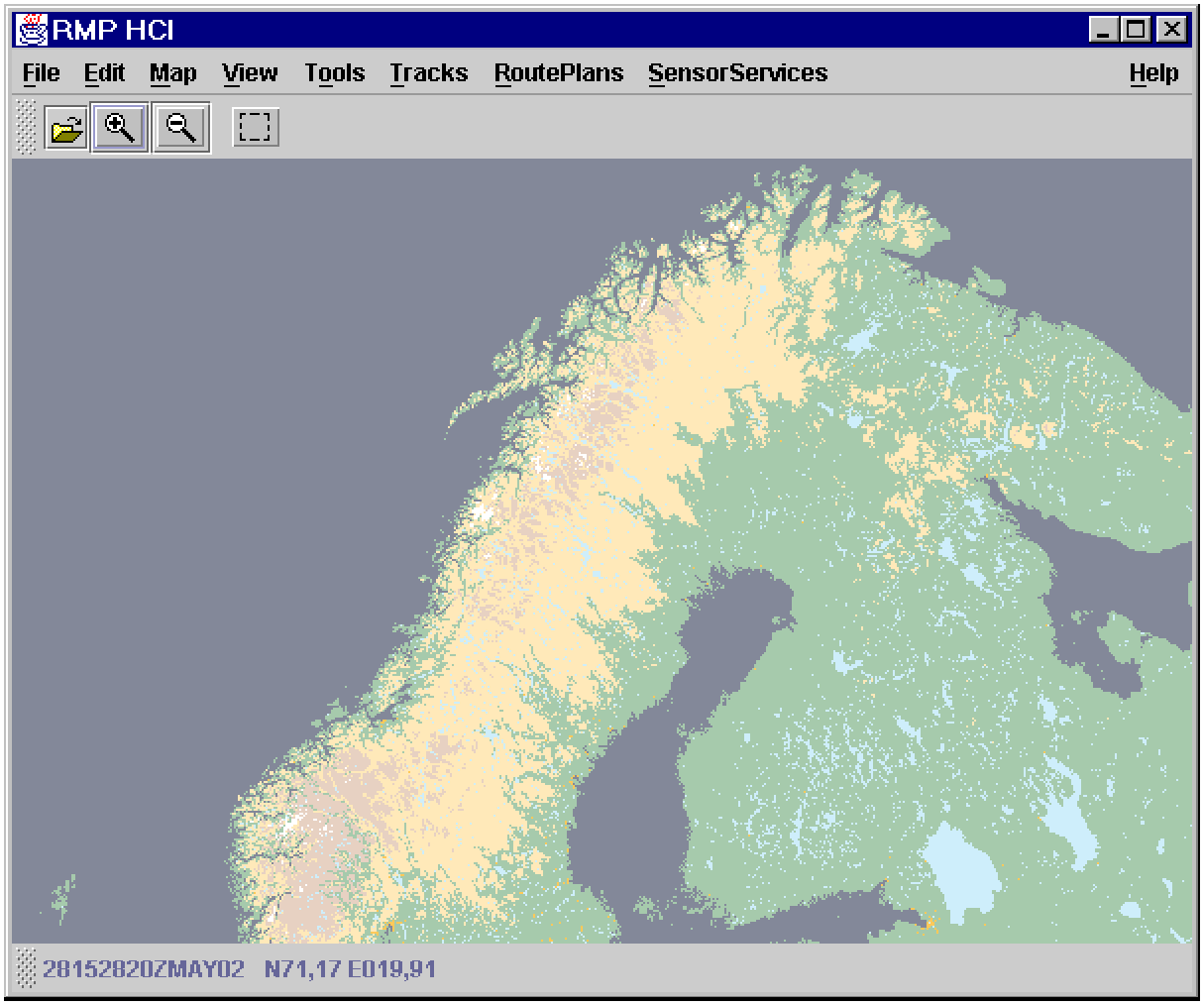
- "Tools": Inneholder foreløpig kun en "EventLog" med liste over hendelser fra RMP-komponenten (f eks oppdaterte tracks).
- "Tracks" inneholder funksjoner for å operere på track (f eks "new track", "delete track ",

”merge track”). Se kapittel 6.1.4.

- ”RoutePlans”: Her kan brukeren få en liste over alle tilgjengelige ruteplaner ("Display Plans"), og velge hvilke ruteplaner som skal visualiseres i kartet (se kapittel 6.1.6).
- ”SensorServices”: Her var det opprinnelig tenkt å legge inn funksjoner for at bruker skulle kunne søke etter tilgjengelige sensorer og å velge de han ønsket å motta data fra. Dette ble ikke implementert.

Verktøymenyen gir brukeren tilgang til ofte brukte funksjoner, som f eks:

- ”Open Area”: Velger blant predefinerte geografiske områder (”Open Area” ligger også under ”File” på hovedmenyen).
- ”Zoom”: Forstørrer/forminsker kartutsnitt (”Zoom In/Out” ligger under ”View” på hovedmenyen).
- ”Select Target”: Velge track i et område.



Figur 6-1 Hovedvindu i brukergrensesnittet

6.1.2 Kart

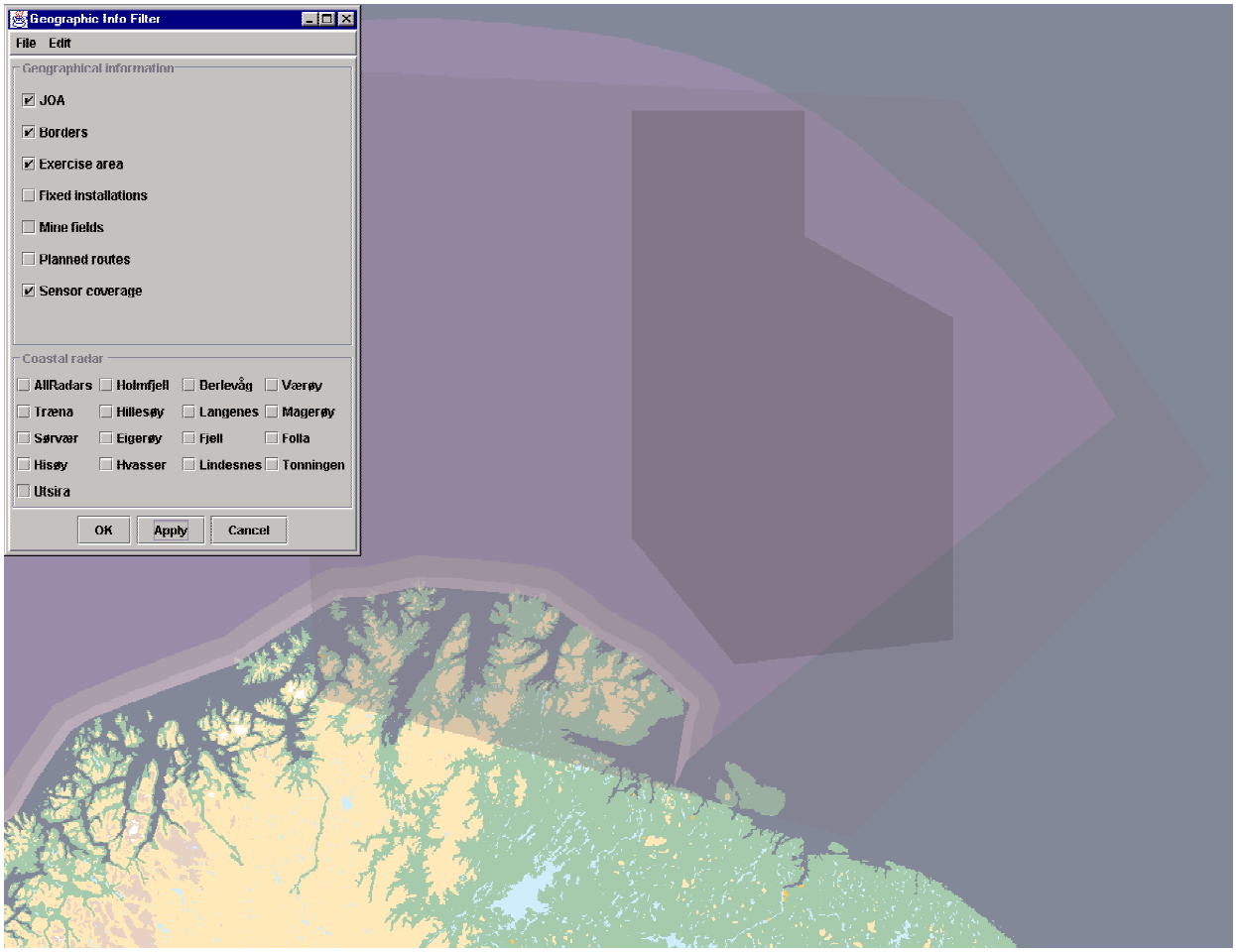
Demonstratoren benytter Maria som karttjener. Dette vil si at Maria produserer et rasterbilde av det kartområdet brukeren ønsker presenterert på skjermen, samt at Maria regner ut den

geografiske posisjonen til et pixel i rasterbildet. I tillegg er det er implementert en del standard kartfunksjonalitet for å:

- skalere bildet (ZoomIn, ZoomOut)
- velge hvilket område som skal vises ved å
    - dra kartet ved hjelp av musa,
    - benytte et lagret geografisk område,
    - eller angi geografisk posisjon og målestokk for å flytte seg i kartet.
- velge hva som skal vises av geografisk stedfestet informasjon
- opprette flere vinduer og lagre kartoppsett.

6.1.3 Annen stedfestet informasjon

I Viewer er det implementert lagdeling av geografisk stedfestet informasjon (f eks grenser og kystradardekning), samt taktisk informasjon ("Joint Operational Area" (JOA), ruteplaner og øvelsesområder). Brukeren kan slå av og på lagene etter ønske ved hjelp av dialogboksen vist i Figur 6-2.



Figur 6-2 Dialog for valg av overlays

Kartinformasjonen i lagene over kartet ligger lagret som SOSI-filer og når informasjonen skal presenteres leses den inn i Viewer av en SOSI-tolker. SOSI står for ”Samordnet Opplegg for

Stedfestet Informasjon" som er et standardformat for digitale geografiske data i Norge, og definert av Statens kartverk. SOSI-tolkeren som er implementert i demonstratoren er ikke komplett. Vi har kun implementert en del av det som ligger i SOSI-versjon 1.4 (14). Vi har også gått utover standarden ved at vi har lagt inn ekstra linjetemaer som gir oss andre farger enn det som er definert i standarden.

Fargene brukt i kartet er basert på arbeider beskrevet i (18) (19). Vi har brukt delvis transparente farger for å kunne vise overlappende områder og annen informasjon i kartet. Figur 6-2 illustrerer dette med et utsnitt av Barentshavet med norske grenser: grunnlinje, 4-, 12- og 200 nautiske mil, Gråsonen og et Joint Operation Area (JOA). Ved at fargene er transparent kan det legges flere lag i kartvinduet uten at kartet (eller annen informasjon) blir skjult.

De lagene som er implementert er organisert i en dialog (GeoFilter) hvor operatøren kan velge hvilke lag som skal vises. I GeoFilter-dialogen er det implementert et fast utvalg av geografisk informasjon som kan velges. Dette gjelder:

- JOA
- Grenser
- Øvelsesområde
- Faste Installasjoner (ikke implementert)
- Minefelt (ikke implementert)
- Ruter/leder
- Radardekning

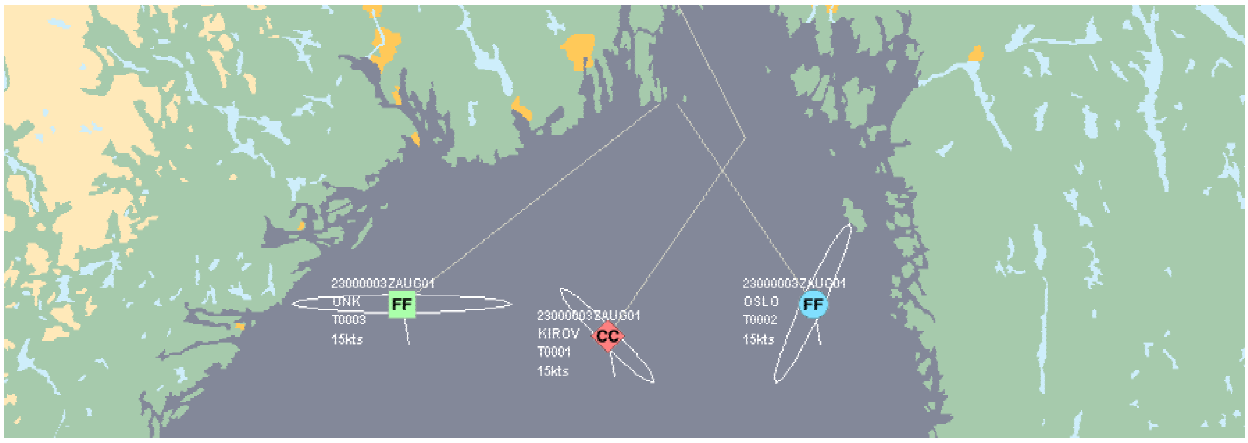
Dekningen til kystradarkjeden er lagt inn og operatøren kan velge å presentere alle sammen eller bare et utvalg av radarene. Valgene i dialogboksen er statiske, det betyr at skal man endre på informasjonsutvalget i selve grensesnittet må man inn i koden gjøre endringene der. Derfor er det lagt til støtte for at operatøren (fra File-menyen i dialogboksen) kan hente opp egne SOSI-filer som kan tegnes inn som et lag i kartet.

JOA vist i Figur 6-2 er digitalisert gjennom DIMAS og grensene ligger på originalformat fra Statens Kartverk.

6.1.4 Track presentasjon og Track Management

Fartøyer (tracks) angis med symboler, farge og tekst basert på Mil-Std 2525A (6). Symbolene er skalerbare med valg for fire symbolstørrelser. I tillegg tegnes det tilleggsinformasjon som hastighetsvektor, usikkerhet i posisjon (ellipse) og track-historie. Track-nummer, navn, dato og fart angis med tekst i henhold til (6).

Figur 6-3 viser tre fartøyer i Oslofjorden med medium symbolstørrelse, hastighetsvektor, track-historie og usikkerhetsangivelse. Velges minste symbolstørrelse fjernes all tekst og symbolet vises som et punkt med en retningsvektor og usikkerhetsangivelse. Symbolstørrelsene "Large", "Medium", "Small" og "Point" tilsvarer henholdsvis 50, 25, 15 og 5 punkter på skjermen.



Figur 6-3 Presentasjon av track

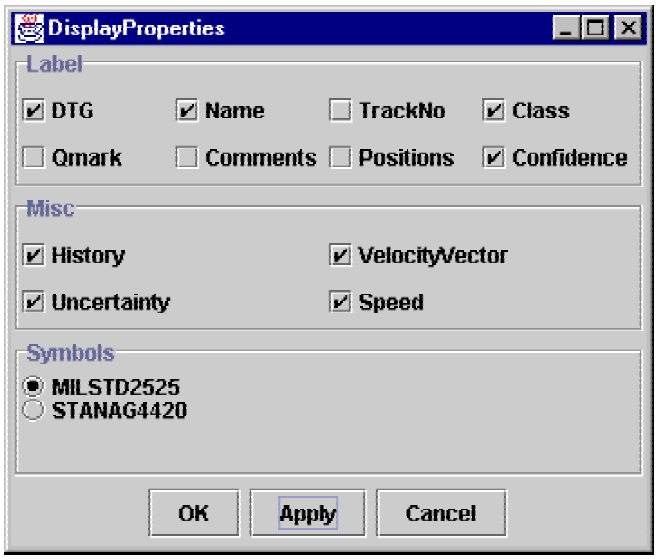
Operatøren kan velge *hvilke* typer track som skal vises og hva som skal vises *om* et track. Figur 6-4 viser ”Filter”-dialogen for å filtrere hvilke type tracks som skal vises. Standardvalg er satt til "alle typer track", og man kan filtrere på identitet (venn, nøytral eller ukjent) og kategori. Denne dialogen er tilgjengelig fra hovedmenyen og gjelder alle track i bildet.

Figur 6-4 Filter dialogen

DisplayProperties (Figur 6-5) er en liste over mulige egenskaper ved track hvor operatør etter fritt valg kan velge det han vil vise. Egenskapene er i henhold til (6). DisplayProperties er tilgjengelig både fra hovedmenyen (for alle track) og for ett valgt track eller for en valgt gruppe av track.

Det er også implementert funksjoner for å legge inn et nytt track, slette track og oppdatere track (trackhåndteringsfunksjonalitet). Ut fra *Edit*-menyen kan operatøren velge track på to ulike måter:

1. Velge track fra en liste
2. Velge track ved å markere en region på skjermen.

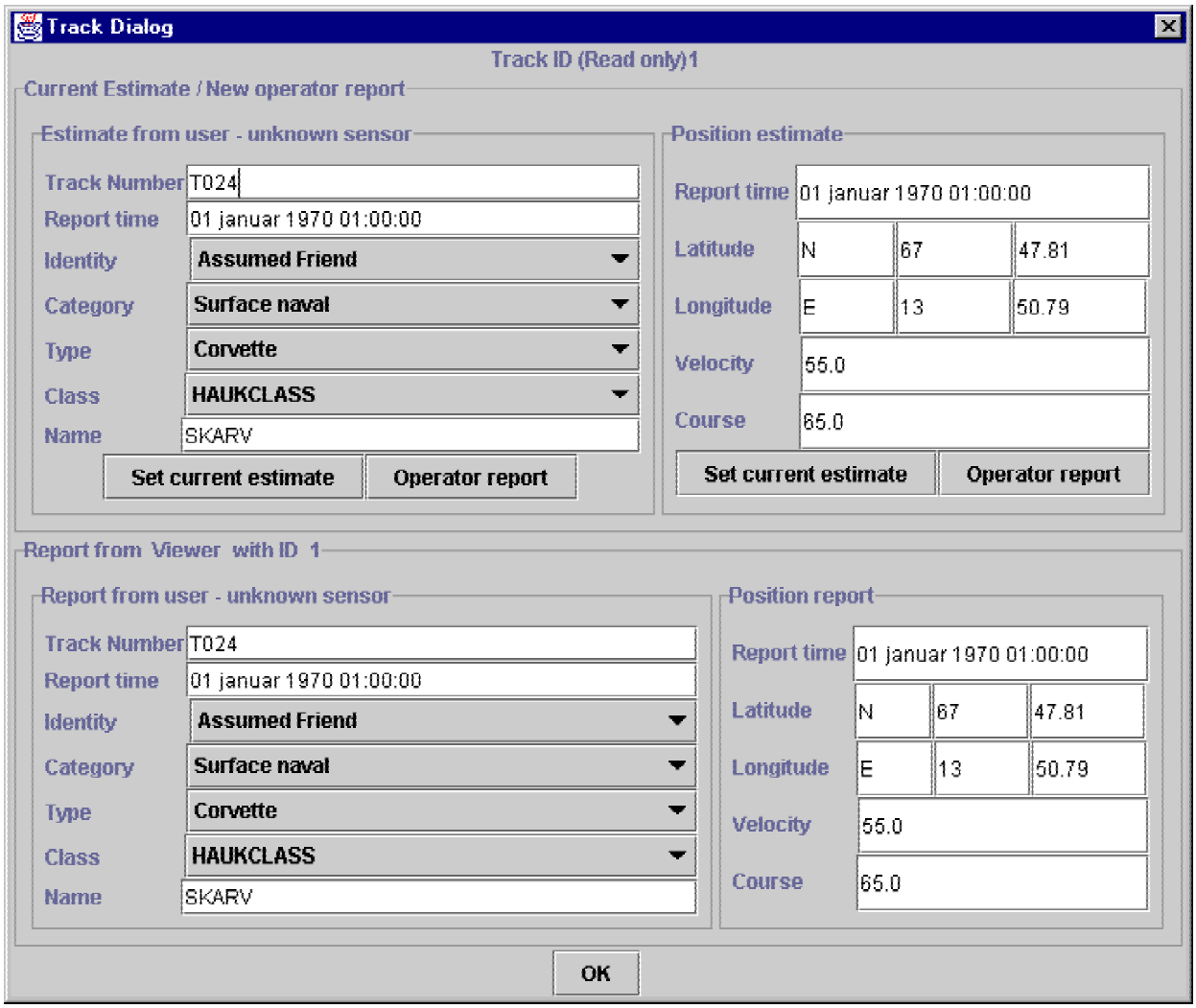


Figur 6-5 Operatør skal kunne spesifisere hva som skal vises om et track

Grensesnittet er objektorientert slik at menyer og funksjonalitet endres etter som ett eller flere track er valgt. F eks er det er fire forskjellige menyer som vises avhengig av hvor mange track som er valgt:

1. Hvis ingen track er valgt vises ”New Track” og generelle kartfunksjoner
2. Når ett track er valgt fremkommer "Classification", "TrackDialog" og ”Delete Track”
3. Når to track er valgt vises ”Merge Track”, "TrackDialog", "Classification" og ”Delete Track”.
4. Hvis flere enn to track er valgt fremkommer kun "TrackDialog" og ”Delete Track”.

"Merge Track" slår sammen to track til ett. Dette menyvalget fremkommer kun når to track er valgt. TrackDialogen (Figur 6-6) er sentral når operatøren ønsker å vise all informasjon om et track i en liste. Denne dialogen brukes også ved ”New Track” og ved oppdatering av track. Klassifikasjonsdialogen blir nærmere beskrevet i neste kapittel.



Figur 6-6 "TrackDialog"

6.1.5 Klassifikasjon

Klassifikasjonsdialogen startes ved at brukeren velger et track og velger 'Classification', se kapittel 6.1.4. Da vises dialogboksen i Figur 6-7. I denne dialogboksen kan brukeren

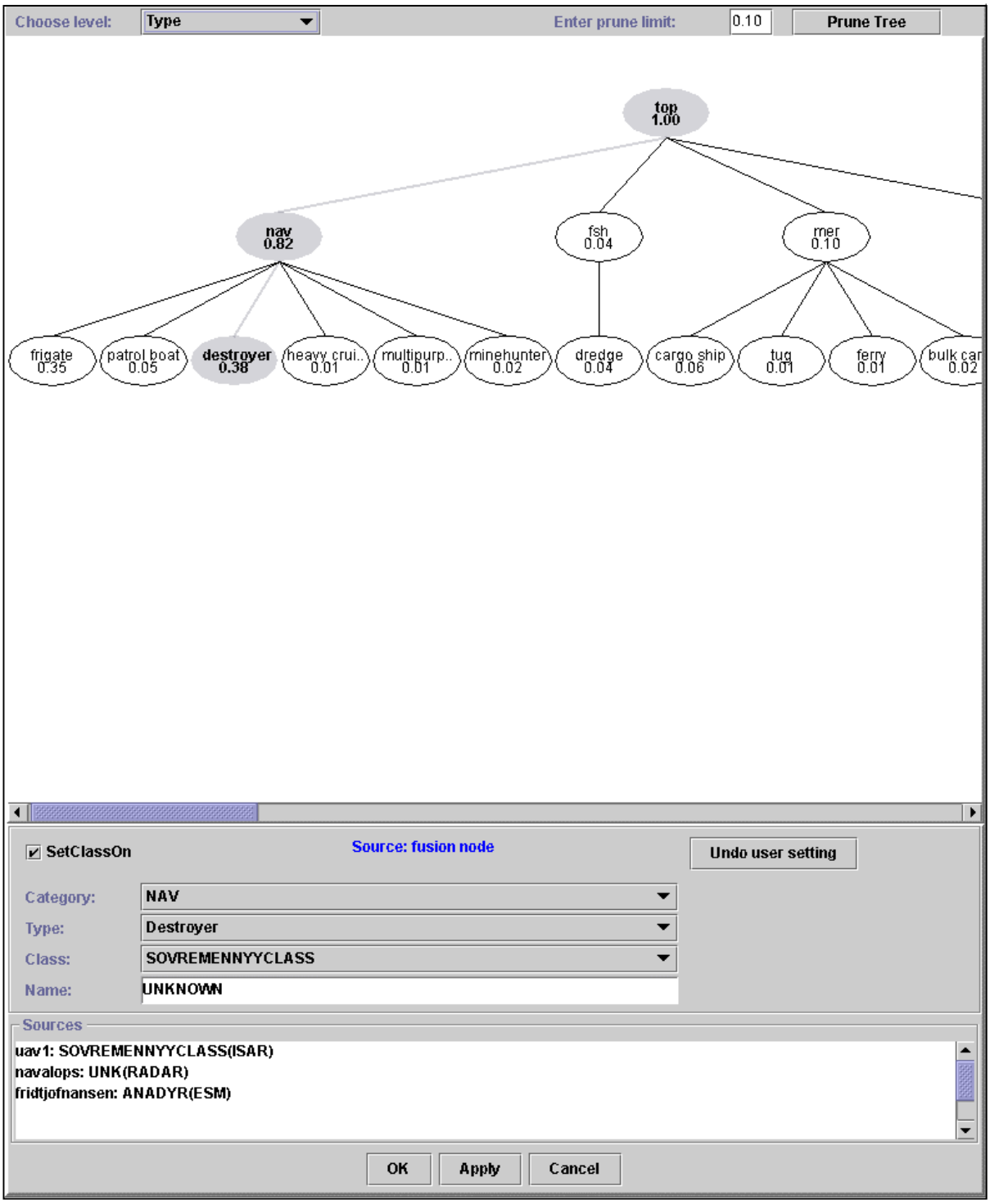
- se all klassifikasjonsinformasjonen for et valgt track
- og angi en klassifikasjon av tracket.

Klassifikasjonsinformasjonen vises i tre deler. I den øverste delen vises et klassifikasjonstre hvor klassifikasjonsestimatet til tracket er markert med grått. I dette treet er alle klassifikasjonsmuligheter for tracket vist i en hierarkisk struktur sammen med en konfidensangivelse beregnet av klassifikasjonskomponenten (se kapittel 7.3). Den hierarkiske strukturen har fire nivåer og er basert på OTG (1) og MIL-STD 4420 (5):

- kategori (f eks marinefartøy, fiskefartøy, handelsfartøy, ukjent),
- type (f eks fregatt, destroyer),
- klasse (f eks Oslo-klassen, Hauk-klassen),

- navn (entydig klassifikasjon).

I Figur 6-7 er klassifikasjonstreet vist til nivå 'type' som angitt i boksen merket 'Choose level:' øverst i venstre hjørne.



Figur 6-7 Dialogboksen for klassifikasjonsinformasjon

Under klassifikasjonstreet vises klassifikasjonsestimatet på hver av de fire nivåene. Her angis også kilden til estimatet (Source: fusion node). De mulige verdiene her er:

1. ’fusion node’ (klassifikasjonsestimatet er basert på klassifikasjonsrapporter fra fusjonsnoder),
2. ’organic sensor’ (klassifikasjonsestimatet er basert på klassifikasjonsrapporter fra organiske sensorer),
3. ’merchant report’ (klassifikasjonsestimatet er basert på egenrapporter fra handelsfartøy)
4. ’own report’ (klassifikasjonsestimatet er basert på egenrapporter fra fartøy som ikke er handelsfartøy)
5. og ’user!’ (klassifikasjonen er satt av en bruker).

Den nederste delen av klassifikasjonsdialogen viser hvilke kilder som har gitt klassifikasjonsdeklarasjoner om tracket, hvilke deklarasjoner de har gitt og hva slags sensorer som er grunnlaget for deklarasjonene. Dette vises på formen:

uav1: SOVREMENNYYCLASS (ISAR)

som betyr at uav1 har sendt klassifikasjonsdeklarasjonen SOVREMENNYYCLASS og at dette er observert med en ISAR (Invers Syntetisk Aperture Radar).

I tillegg til framvisning av klassifikasjonsinformasjon gir klassifikasjonsdialogboksen brukeren flere muligheter til å endre på hvordan informasjonen i dialogboksen vises. Brukeren kan også sette klassifikasjonen av det valgte tracket. Dette er beskrevet i det følgende.

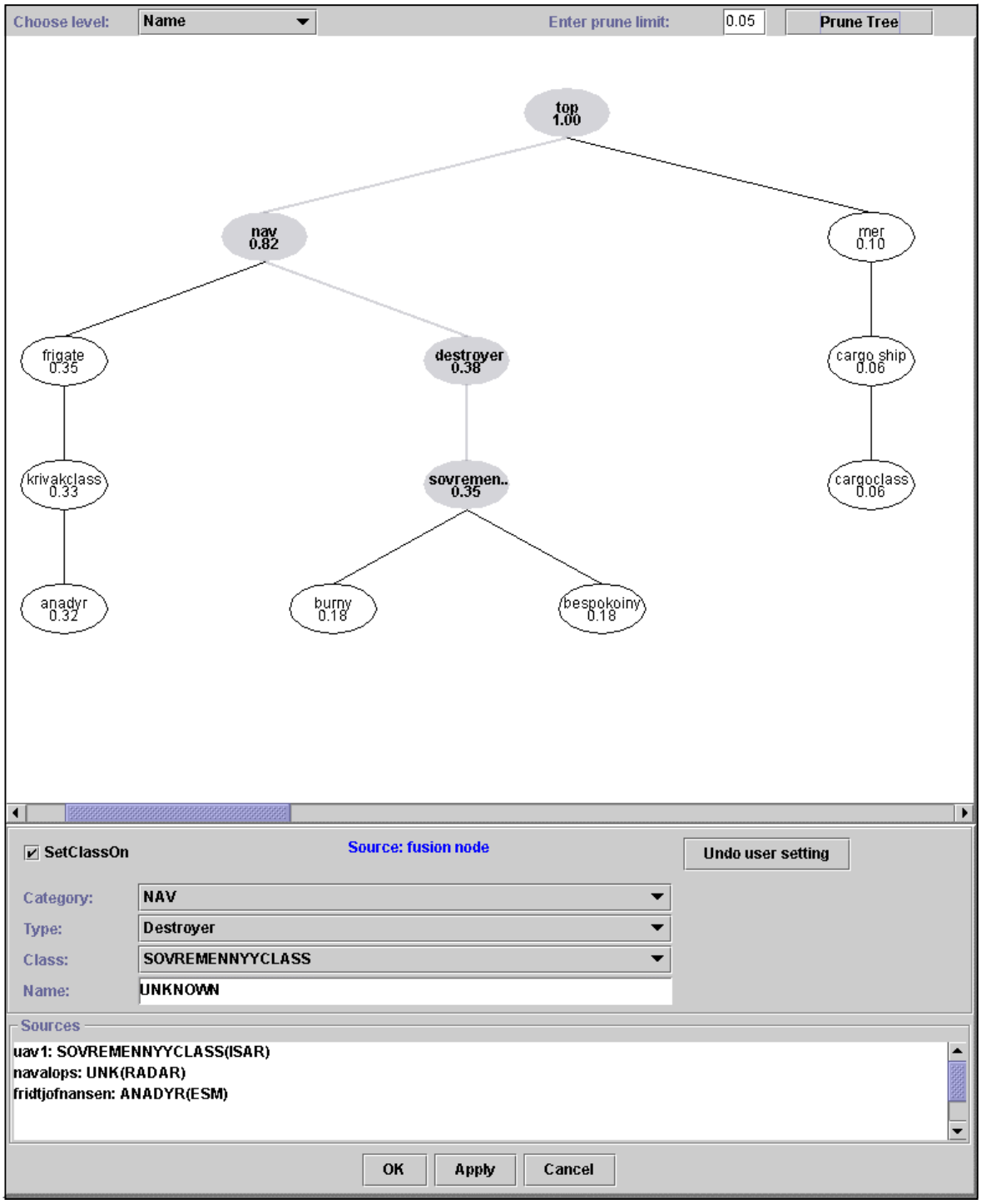
Brukeren kan velge hvor mange nivåer som skal vises i klassifikasjonstreet med ’Choose level’ øverst i venstre hjørne av dialogboksen. Valgene er ’top’, ’category’, ’type’, ’class’ og ’name’. Ved valg av f eks ’type’ vil alle nivåene ned til og med type vises i treet slik som i Figur 6-7.

Når klassifikasjonstreet blir like stort som i Figur 6-7 blir det hele nokså uoversiktlig fordi det blir forsøkt vist flere noder enn det er plass til i dialogboksen. For å bedre på dette kan brukeren velge at kun noder med konfidens større enn en gitt verdi vises. Dette gjøres ved at verdien skrives inn etter ’Enter prune limit’ øverst i høyre hjørne av dialogboksen og at knappen ’Prune Tree’ så trykkes. Hvis dette gjøres med verdi 0,05 og nivået i klassifikasjonstreet velges til ’name’, vil det se ut som i Figur 6-8.

Ved å krysse av i boksen ’SetClassOn’ til venstre under klassifikasjonstreet, kan brukeren sette klassifikasjonen av tracket ved å velge i boksene merket ’Category’, ’Type’ og ’Class’ eller skrive inn et navn i boksen merket ’Name’. Når så knappen ’Apply’ eller ’OK’ trykkes, blir klassifikasjonen satt. Når brukeren har satt en klassifikasjon, står denne helt til brukeren gir en ny klassifikasjon eller til brukeren sletter klassifikasjonen han satte. En brukersatt klassifikasjon kan slettes ved å trykke på knappen ’Undo user setting’ som finnes til høyre under klassifikasjonstreet. Ytterligere klassifikasjonsdeklarasjoner fra sensorer vil ikke påvirke klassifikasjonsestimatet så lenge det er satt av en bruker.

Helt nederst i klassifikasjonsdialogboksen finnes knappene ’OK’, ’Apply’ og ’Cancel’. ’OK’-knappen utfører en eventuell klassifikasjonsendring og lukker dialogboksen, ’Apply’ utfører en eventuell klassifikasjonsendring uten å lukke dialogboksen, mens ’Cancel’ lukker dialogboksen

uten å utføre en eventuell klassifikasjonsendring.



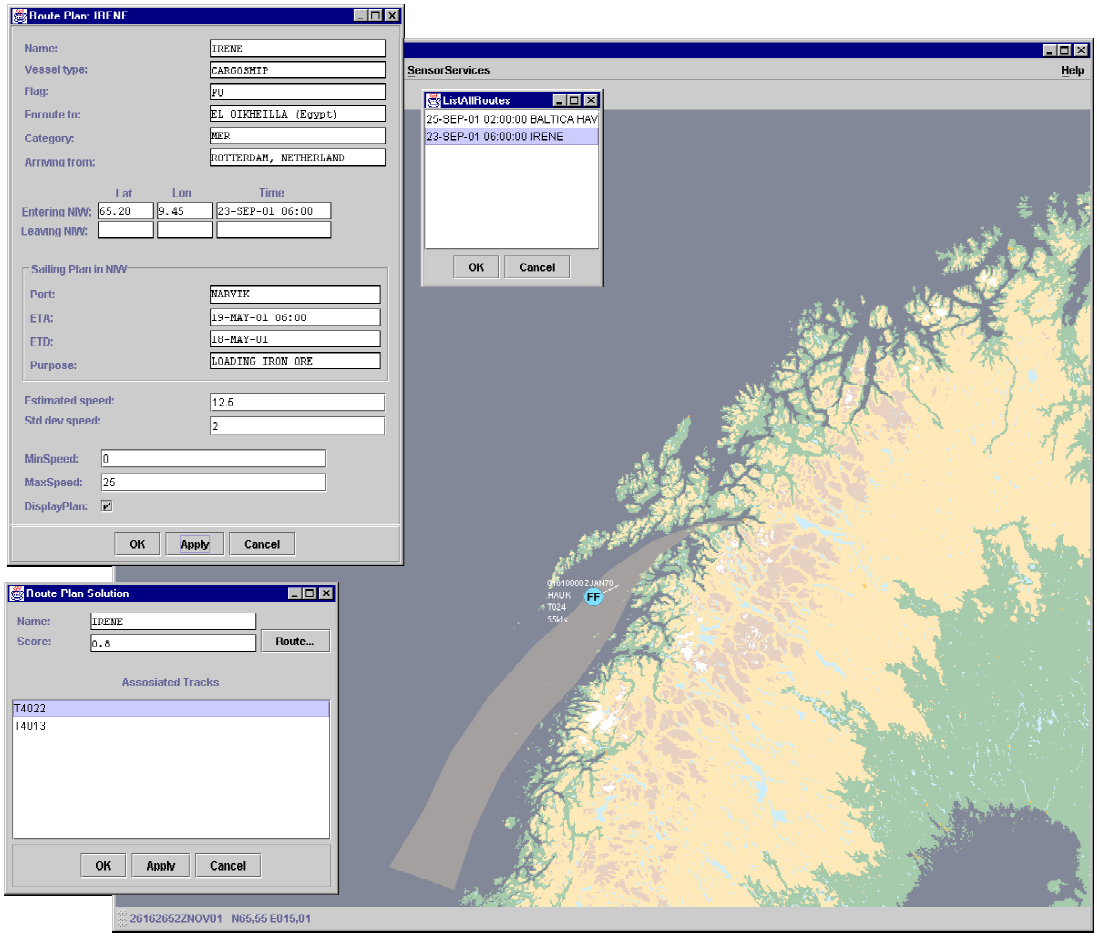
Figur 6-8 Dialogboksen for klassifikasjon når hele treet vises og man filtrerer bort noder med konfidens mindre enn 0,05.

6.1.6 Ruteplaner

Dette kapitlet inneholder kun en beskrivelse av brukergrensesnittet som er utviklet mot ruteplankomponenten (se kapittel 7.2 for beskrivelse).

I hovedmenyen kan operatøren få listet alle ruteplaner ("ListAllRoutes"). Ved å dobbeltklikke på en ruteplan fremkommer et vindu som inneholder all informasjon om den valgte planen (se Figur 6-9):

- Navnet på fartøyet som skal følge ruta
- Fartøystype
- Flagg/nasjonalitet
- Fartøyets neste destinasjon
- Fartøyskategori
- Siste anløpte havn
- Tidspunkt for kryssing av grunnlinjen for seilas inn og ut av norsk farvann.
- Seilingsplan i norsk farvann (anløpssted, ankomsttidspunkt og formål med besøket).



Figur 6-9 Ruteplandialogene m/visualisering av estimert ruteplan

Informasjonen nevnt ovenfor er tilsvarende det som rapporteres i henhold til Anløpsreglementet. I tillegg må operatøren legge inn fartsprofilen til fartøyet gjennom å spesifisere antatt gjennomsnittsfart, maksimal fart og gjennomsnittsfartens standardavvik. Denne informasjonen benyttes for å konfigurere ruteplanalgoritmen.

Når man velger en rute kommer det også opp et vindu som viser løsningen til

ruteplanalgoritmen for denne ruta (RoutePlanSolution) i form av en liste av radartrack som har blitt assosiert til ruta. Disse radartrackene er de kandidatene som mest sannsynlig representerer fartøyet som er tilknyttet ruteplanen. Fra ruteplandialogen er det også mulig for operatør å få tegnet ruteplanen i kartet. Ruteplanen tegnes da ut i ett eget lag i kartvinduet. Den geografiske beskrivelsen av ruta må legges inn av brukeren manuelt. Dette har vi gjort ved å digitalisere ruta i DIMAS og lagret den på et SOSI-format (14) som leses inn av Viewer. I et operativt system ville man ha hatt en database av ruter å velge mellom, samt mulighet for å tegne opp en ny rute i kartet.

7 TJENESTEKOMPONENTER

Dette kapitlet inneholder en beskrivelse av komponentene i tjenestelaget. Først beskrives RMP-komponenten og datamodellen den er basert på, så ruteplankomponenten og klassifikasjonskomponenten. Deretter følger en beskrivelse av rapporteringstjenesten, posisjonsestimatoren samt komponenten som harmoniserer RMP-komponenten og MCCIS sin trackdatabase.

7.1 Modell av det maritime situasjonsbildet

Dette kapitlet gir en oversikt over objektmodellen av det maritime situasjonsbildet som er brukt i demonstratoren. Med et maritimt situasjonsbilde menes her situasjonsbeskrivelsen som presenteres for en beslutningstager på operativt nivå eller for styrkeledelsen på taktisk nivå. Situasjonsbeskrivelsen er en beskrivelse av situasjonen slik den er for øyeblikket i tillegg til en beskrivelse av historien som har ledet til øyeblikkssituasjonen.

7.1.1 Generelt om modellen

I arbeidet med å lage en modell av innholdet i det maritime situasjonsbildet ble følgende utvekslingsformater og presentasjonsstandarder gjennomgått for å identifisere hva slags informasjon som måtte kunne representeres i modellen: Utvalgte meldinger i OTG (1), Link 11 (2), Link 22 (3) og AdatP-3 (4) samt informasjonshierarkiene i presentasjonsstandardene STANAG 4420 (5) og Mil-Std 2525A (6).

Det ble videre tatt utgangspunkt i GRACE Common Model (8), som er basert på ATCCIS (Army Tactical Command Control Information System) Generic Hub (7). Gjennom den nevnte gjennomgangen av utvekslingsformater og presentasjonsstandarder ble følgende mangler ved ATCCIS sett i forhold til demonstratorens behov identifisert:

- det er ingen mulighet for å angi usikkerhet i forbindelse med angivelse av en enhets posisjon,
- det er ingen mulighet til å angi kvaliteten til en enhets klassifikasjon,
- og begrepet ’track’ er ikke representert.

Disse manglene ble utbedret ved at:

- det ble gitt mulighet for å legge til en usikkerhetsellipse gjennom klassen Ellipsis i forbindelse med en enhets posisjon,
- det ble gitt mulighet til å angi klassifikasjonskvalitet gjennom klassen UnitPerception,
- klassen Context ble omdøpt til Track siden et track i ATCCIS kan representeres ved å gruppere flere Perceptions sammen med en Context.

Siden modellen kun skulle brukes internt i prosjektet, var det ikke behov for å foreta utvidelser eller endringer i forhold til ATCCIS som ikke kunne relateres til oppfyllelse av direkte krav til demonstratoren. Spesielt ble det kun sett på modellering av overflateobjekter siden det i demonstratoren kun skulle utarbeides funksjonalitet for etablering av overflatebildet.

7.1.2 Implementasjon

RMP-komponenten lagrer det maritime situasjonsbildet i form av et antall track, og gjør disse tilgjengelig for andre komponenter. Trackene holder rapportene fra de forskjellige bidragsyterne (Reporting Unit) separert, i motsetning til MCCIS som i alle fall sett fra utsiden, blander disse sammen. Datamodellen er nærmere beskrevet i kapittel 7.1.2.1.

RMP-komponenten er implementert som en tjeneste som registrerer seg i CoABS Grid, eller rettere sagt to tjenester. Den ene tjenesten benyttes for å legge inn og hente ut data av modellen, mens den andre tjenesten er en hendelsestjeneste som benyttes for å registrere interesse for endringer i modellen f eks at noen har lagt inn nye rapporter. Man får da varsel (mottar en hendelse) ved endringer i trackdatabasen. Sett fra utsiden er RMP-komponenten beskrevet av datamodellen, tjenestenes grensesnitt og noen enkle regler for bruk av tjenesten. Dette er f eks at man må opprette et track før man kan begynne å legge til rapporter og hente ut informasjon om tracket. RMP-komponentens grensesnittet er beskrevet i kapittel 7.1.2.2.

RMP-komponenten benytter en Jini-tjeneste ved navn *Outrigger* for lagring av trackene. RMP-komponenten inneholder ikke noen data selv, alle data er lagret i *Outrigger*. *Outrigger* er en referanseimplementasjon av *JavaSpaces* som er en spesifikasjon av en Jini-tjeneste for (persistent) lagring og utveksling av data. Et hovedmoment var at *Outrigger* er en del av Jini-teknologien som CoABS Grid er basert på, og at vi derved kunne implementere Rmp-komponenten uten å innføre enda en teknologi-familie i demonstratoren (f eks en SQL-database). Samtidig ville vi bli kjent med en ny teknologi for lagring av data. Implementasjonen av RMP-komponenten er beskrevet i kapittel 7.1.3.

Vi var noe i tvil om JavaSpaces var velegnet for lagring av RMP-modellen, og er nok fortsatt i tvil om dette. Den tiltenkte bruken av JavaSpaces er nok noe forskjellig fra vår anvendelse. Vi har inntrykk av at demonstratoren ble noe treg i vårt meget omfattende scenario når vi kjører 10 ganger sann tid. Det gjenstår å finne ut om dette skyldes *Outrigger* eller andre forhold som f eks for lite hukommelse i arbeidsstasjonene. Det gjenstår å gjøre målinger av effektiviteten av Rmp-komponenten når antallet rapporter som lagres blir stort. Vi er med andre ord i tvil om hvor godt JavaSpaces-teknologien skalerer.

Den første implementasjonen av RMP-komponenten var i et testprogram som benyttet klassen

Vector, som er en av Javas standard ”container”-klasser. Dette ble gjort samtidig som testprogrammet ble tilpasset datamodellen. Denne implementeringen ble i første omgang benyttet til å teste ut grensesnittet og klienter som benytter RMP-komponenten og i neste omgang til å teste ut RMP-komponenten.

7.1.2.1 Datamodellen

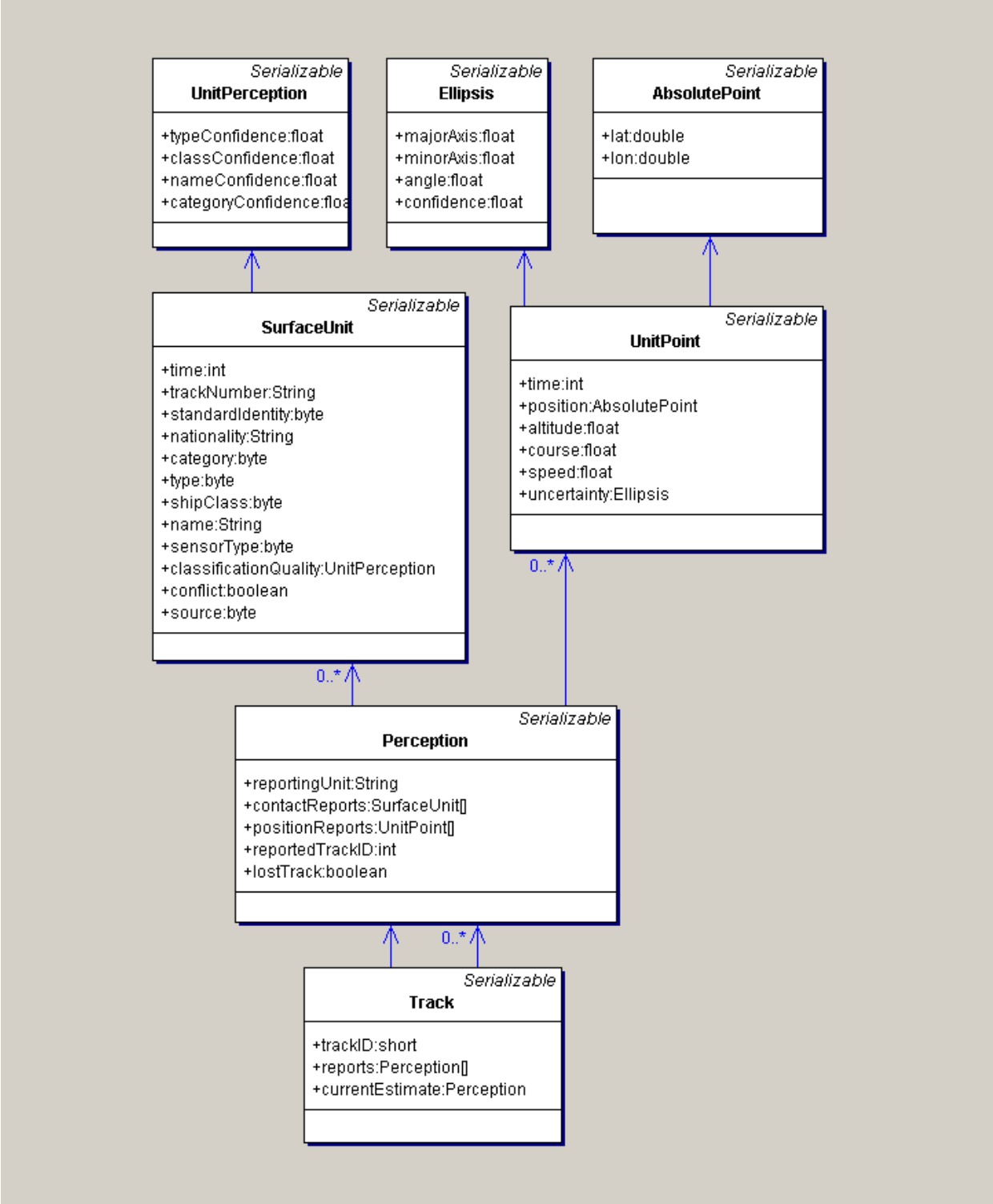
Datamodellen som benyttes for RMP er vist i Figur 7-1 og er beskrevet vha UML. Figuren viser hvordan et track er bygd opp eller beskrevet v h a datatypene Track, Perception, SurfaceUnit, UnitPoint, UnitPerception, Ellipsis og AbsolutePoint. Tegnene 0..* ved pilene betegner at man kan ha fra null til mange objekter av den typen det pekes på.

Et Track er assosiert med en identifikator, som benyttes i datautvekslingen mellom klientene og RMP-komponenten. Denne vil være konstant gjennom trackets levetid, i motsetning til tracknummeret som kan endres av operatørene. I et operativt system vil denne identifikatoren normalt være skjult for operatøren, men under utvikling av demonstratoren var det nyttig å ha den synlig. MCCIS har en tilsvarende identifikator, *track record number* (trkrec).

Hver enkelt ReportingUnit’s oppfattelse av et track kalles en persepsjon. Perception inneholder alle rapportene fra en enkelt reportingUnit. Et Track kan inneholde flere Perception (en for hver raporterende enhet), og en Perception kan inneholde et vilkårlig antall kontaktrapporter av typen SurfaceUnit og et vilkårlig antall posisjonsrapporter av typen UnitPoint. Merk at rapportene fra hver enkelt enhet (reportingUnit) som rapporterer et track holdes atskilt. I tillegg inneholder Perception identifikatoren (tilsvarende track nummer) som reportingUnit benytter for rapportering av tracket, og informasjon om man fremdeles har kontakt med tracket (”lost track”).

I tillegg til at kontaktrapportene inneholder kodete klassifikasjonsdata for kategori, type, klasse og navn, kan rapporten inneholde en UnitPerception med informasjon om kvaliteten på klassifikasjonen. Posisjonsrapportene kan ha en Ellipsis som beskriver posisjonsusikkerheten.

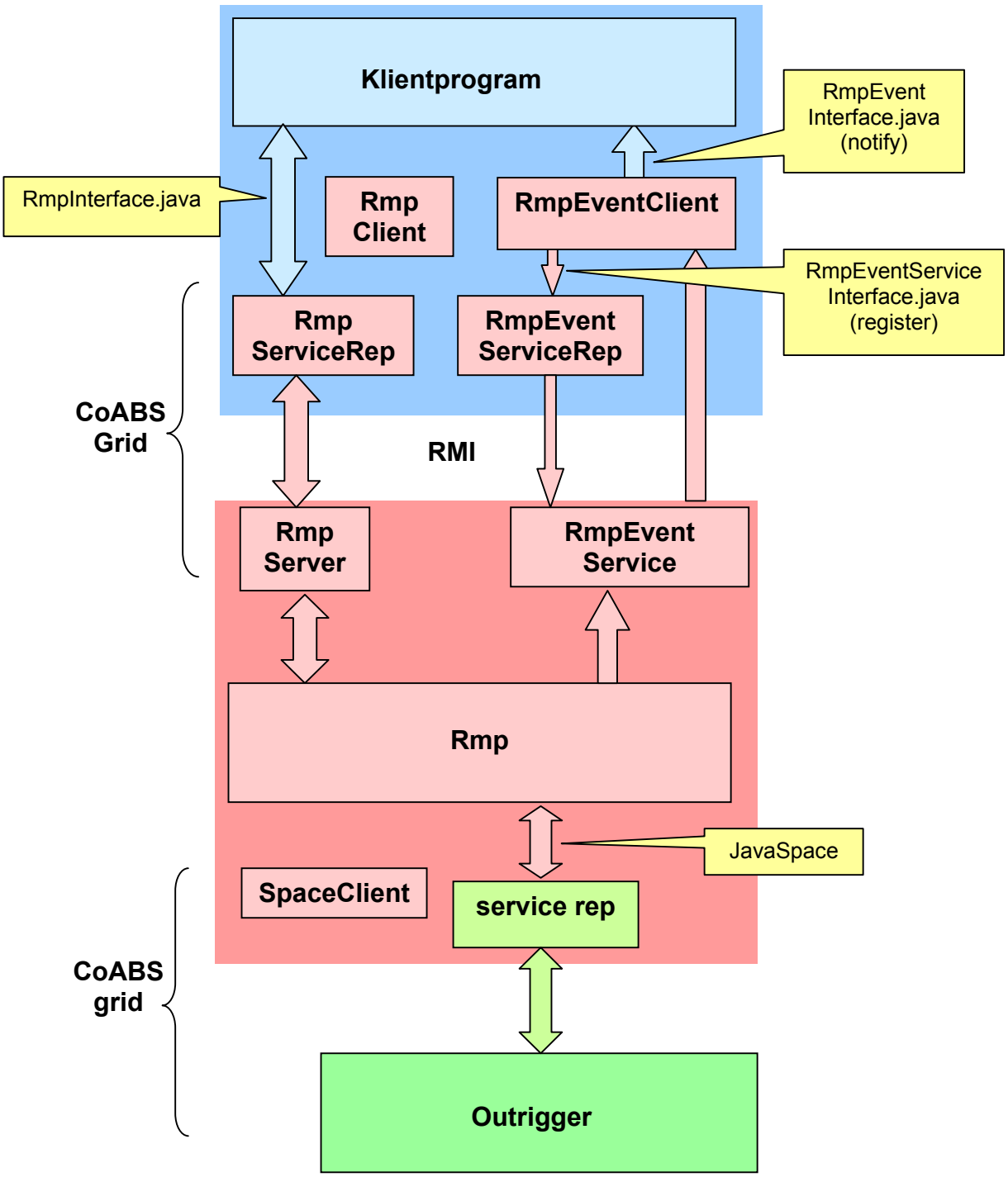
Det beste estimatet om et track finnes i currentEstimate som inneholder den beste kontakt- og posisjonsinformasjonen. Basert på kontaktrapportene kan klassifikasjonskomponenten estimere kategori, type, klasse og navn, og basert på posisjonsrapportene kan posisjonsestimatoren estimere posisjonsdata. Klassifikasjonskomponenten og posisjonsestimatoren setter dette inn i currentEstimate. I tillegg kan også operatøren sette data inn i currentEstimate.



Figur 7-1 Datamodellen for RMP

7.1.2.2 Tjenester

RMP-komponenten registererer seg som to tjenester i CoABS-Grid. ”RmpServer” benyttes for å sette data inn i og hente data ut av RMP-komponenten, mens ”RmpEventService” gir varsel (hendelser) ved endringer av innholdet i modellen.



Figur 7-2 Rmp-tjenestene

```
public interface RmpInterface extends Remote {

short[]   getTrackIDs      ();
short     newTrack         (Track track,   long eventTime);
void      mergeTracks      (short source,  short target, long eventTime);
void      deleteTrack      (short trackID, long eventTime)
Track     getTrack         (short trackID, int oldestReportTime)
Track     getLastReports (short trackID)

void movePerception(short fromTrackID,short toTrackID,String reportingUnit,long eventTime)

UnitPoint     getCurrentUnitPointEstimate   (short trackID)
void setCurrentUnitPointEstimate(short trackID,UnitPoint estimate,long eventTime)
SurfaceUnit getCurrentSurfaceUnitEstimate (short trackID)
void setCurrentSurfaceUnitEstimate (short trackID, SurfaceUnit estimate, long eventTime)

String[]              getReportingUnits (short trackID)
Perception        getPerception      (short trackID, String reportingUnit)
void                  deletePerception  (short trackID, String reportingUnit, long eventTime)

UnitPoint[] getUnitPointReports(short trackID, String reportingUnit, int oldestReportTime)
void addUnitPointReport(short trackID,String reportingUnit,UnitPoint report,long eventTime)
UnitPoint   getLastUnitPointReport(short trackID, String reportingUnit)
UnitPoint   getUnitPointReport    (short trackID, String reportingUnit, int reportTime)

SurfaceUnit[] getSurfaceUnitReports(short trackID,String reportingUnit,int oldestReportTime)
void addSurfaceUnitReport(short trackID,String reportingUnit,
                                  SurfaceUnit report,long eventTime)
SurfaceUnit   getLastSurfaceUnitReport(short trackID, String reportingUnit)
SurfaceUnit   getSurfaceUnitReport    (short trackID, String reportingUnit, int reportTime)

// For receiver
short newTrack(String reportingUnit,int reportedTrackID,UnitPoint up,
                    SurfaceUnit su,long eventTime)
void  setLostTrack(String reportingUnit,int reportedTrackID,long eventTime)
void addUnitPointReport(String reportingUnit,int reportedTrackID,
                               UnitPoint report,long eventTime)
void addSurfaceUnitReport(String reportingUnit,int reportedTrackID,
                                  SurfaceUnit report,long eventTime)
int   getReportedTrackID  (short trackID, String reportingUnit)
short getTrackID              (String reportingUnit, int reportedTrackID)

// For exchange of data with MCCIS

short getRemoteTrackID(short trackID)
void  setRemoteTrackID(short trackID, short remoteTrackID)
short getLocalTrackID   (short remoteTrackID)
short newTrack            (Track track, long eventTime, short remoteTrackID)
}
```

Figur 7-3 Rmp-tjenestens grensesnitt

7.1.2.3 RmpServer-tjenesten

RmpServer-tjenesten gir tilgang til innholdet i RMP-komponenten. Tjenesten er beskrevet v h a Java-grensesnittet RmpInterface som er vist i Figur 7-3. Ved å spørre oppslagstjenesten (Jini's

Lookup Service) etter tjenester som implementerer RmpInterface, vil man dersom tjenesten er registrert i CoABS Grid, få returnert et proxy-objekt som implementerer RmpInterface. Dette objektet kan benyttes for å aksessere tjenesten.

For å gjøre det enkelt å knytte seg til tjenesten er det laget en klasse mil.ffi.rmp.RmpClient som klientprogrammene kan benytte som vist nedenfor:

```
RmpClient     rmp_client     =  new RmpClient();
RmpInterface  remote_rmp     =  rmp_client.lookup_rmp_service();
```

Objektet remote_rmp er proxy-objektet som er lastet ned fra oppslagstjenesten. Dette er et lokalt objekt som gir tilgang til tjenesten. Alt klienten trenger å vite er at proxy-objektet (remote_rmp) har implementert RmpInterface, d v s at objektet implementerer metodene som klienten trenger for å benytte tjenesten. For f eks å få tak i posisjonsdata for et track med trackID lik 42, kan følgende kode benyttes:

```
UnitPoint     position_estimate     = remote_rmp.getCurrentUnitPointEstimate(42);
Double        latitude              = position_estimate.position.lat;
Double        longitude             = position_estimate.position.lon;
```

De fleste metodene i RmpInterface setter inn og henter ut data i henhold til datamodellen, og burde være selvforklarende i henhold til modellen. Noen av parametrene finnes ikke igjen i datamodellen:

- long eventTime benyttes i metoder som endrer innholdet i trackdatabasen. Disse metodene gir opphavet til hendelser som brukerne av RMP-komponenten kan abonnere på v h a RmpEventService-tjenesten. Parameteren long eventTime sendes med hendelsen og kan benyttes av klientprogrammene til å kjenne igjen hendelser de selv er opphav til.
- int oldestReportTime benyttes ved uthenting av rapporter. Alle rapporter som er yngre eller like gamle som oldestReportTime blir tatt med. Ved å sette parameteren lik 0, blir alle rapporter tatt med.
- short remoteTrackID benyttes for datautveksling med MCCIS, og er det samme som track-reckord-nummer (trkrec) i MCCIS. I Rmp-komponenten lagres denne sammen med trackID.

Alle metodene som returnerer datatypen short returnerer trackID, med unntak av getRemoteTrackID(short trackID) som returnerer remoteTrackID.

For å få tak i alle track som er lagret i RMP-komponenten må man benytte metoden getTrackIDs() som returnerer trackID for alle trackene, og deretter kan man benytte getTrack(trackID …) metoden for å hente et track av gangen.

7.1.2.4 RmpEventService-tjenesten

Endringer i RMP-komponenten gir opphav til hendelser som brukerne av RMP-komponenten kan få beskjed om via RmpEventService-tjenesten. Tjenesten er beskrevet av Java-grensesnittet RmpEventServiceInterface, som er vist i Figur 7-4. Alt denne tjenesten gjør er å registrere en RemoteEventListener som abonnent eller lytter til hendelser med identitet lik ev_id.

```
public interface RmpEventServiceInterface extends Remote
{
  long register(long ev_id, RemoteEventListener listener);
}
```

Figur 7-4 Grensesnitt til RmpEventService

For å gjøre det enkelt å knytte seg til tjenesten er det laget en klasse mil.ffi.rmp.RmpEventClient som klientprogrammene kan benytte for å knytte seg til tjenesten, f eks som vist i nedenfor:

```
RmpEventClient rmp_event_client    = new RmpEventClient(this);

RmpEventServiceInterface rmp_event_service =
            rmp_event_client.lookup_rmp_event_service();
```

Merk at metoden lookup_rmp_event_service() i tillegg til å finne tjenesten også benytter tjenesten til å registrere RmpEventClient som lytter (listener) eller abonnent på hendelsene. Hendelsene sendes videre til klientprogrammet (this).

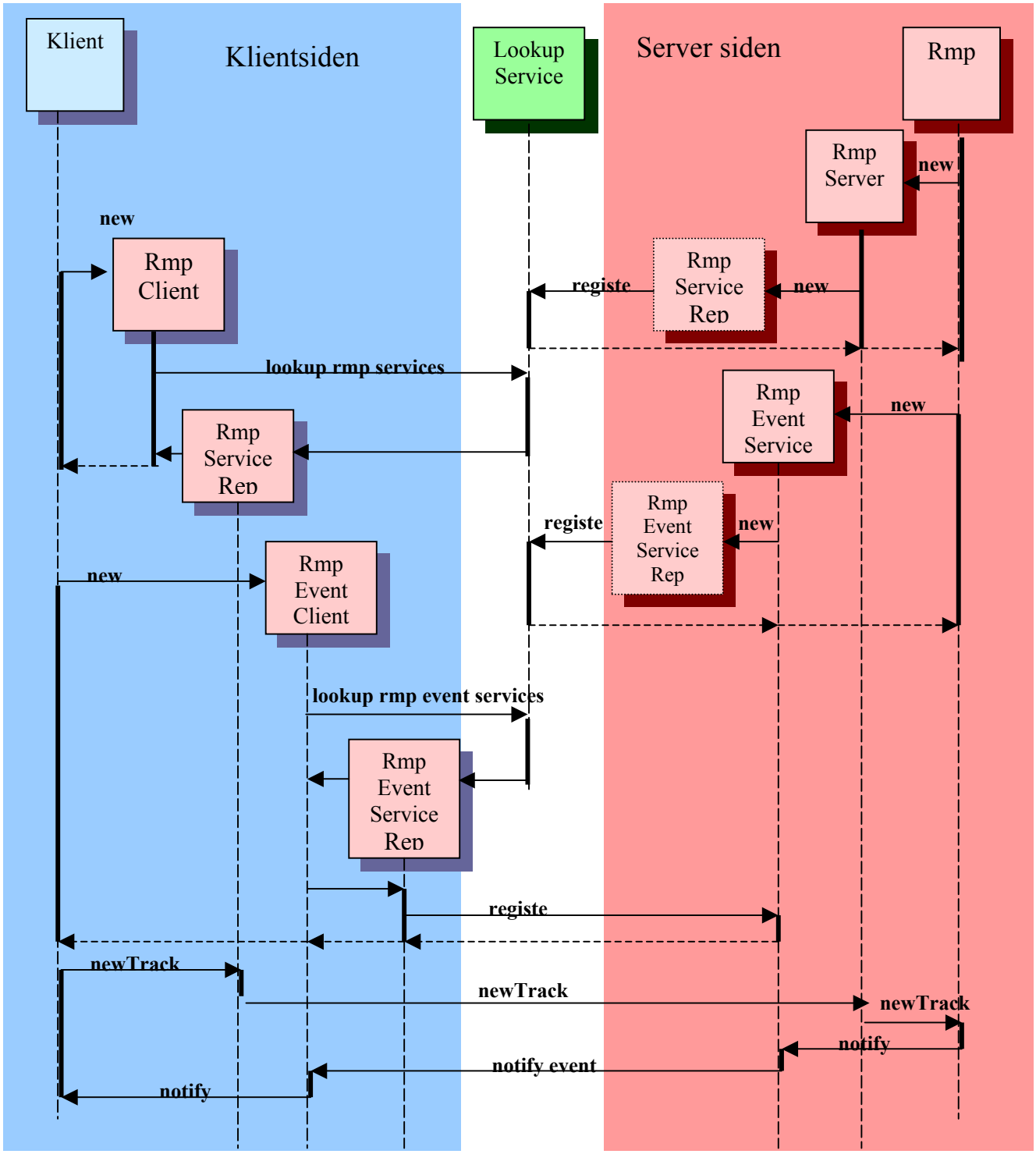
For at klientprogrammet skal kunne motta event via RmpEventClient må det implementere grensesnittet RmpEventInterface som vist i Figur 7-5, d v s at klientprogrammet må ha en notify() metode for mottak av hendelsene.

```
public interface RmpEventInterface
{
   void notify (RmpEvent event);
}
```

Figur 7-5 Klientprogrammet må implementere RmpEventInterface for å kunne motta RMP-event

7.1.2.5 Sekvensen ved oppkobling og bruk av tjenestene

Sekvensen ved oppkobling og bruk av tjenestene er vist som et UML sekvensdiagram i Figur 7-6. Ved oppstart av RMP-komponenten vil denne etter å ha koblet seg opp mot Outrigger (dette er ikke vist i sekvensdiagrammet), registrere RmpServer og RmpEventService-tjenestene i CoABS Grid. Deretter vil klient-programmene kunne koble seg opp mot tjenesten.



Figur 7-6 Sekvensen ved oppkobling mot Rmp-modellen og bruk av tjenesten

Når data settes inn i RMP-komponenten, f eks et nytt track, sendes dataene lokalt til proxy-objektet som sørger for fjerntransporten av data til tjenesten. Informasjon om at et nytt track er satt inn sendes i form av hendelser til klienter som har registrert seg. Klienter som er interessert kan deretter benytte informasjonen i hendelsen til å hente tracket.

7.1.3 JavaSpaces

For spesifikasjonen av JavaSpaces henvises det til JavaSpaces™ Service Specification som er en del av Jini (13). Tjenesten er spesifisert som en Java-grensesnitt med fire metoder:

- write: for å skrive et dataobjekt inn i JavaSpaces
- read: for å lese ut et dataobjekt som passer en gitt mal (”template”)
- take: for å ta ut et dataobjekt
- notify: for å få en hendelse tilsendt når dataobjekter som passer malen blir skrevet inn i JavaSpaces.

Datatypen Entry er en sentral datatype i Jini. Alle offentlige felter i en Entry må være objekter, d v s de kan ikke være av primitive typer som f eks float og int.

7.1.3.1 Lagring av posisjonsrapporter

Som et eksempel på hvordan data lagres i JavaSpaces er det nedenfor vist hvordan posisjonsrapporter skrives inn i og leses ut av JavaSpaces. Datamodellen definerer klassen UnitPoint for posisjons-rapporter. Den har int time, float course og float speed som primitive datatyper, og kan derfor ikke være en Entry. Det er derfor innført en egen klasse SpaceUnitPoint for innpakking av posisjonsrapportene.

```
public final class SpaceUnitPoint implements Entry
{
   public SpacePerceptionID     perceptionID;
   public SpaceUnitPointID      unitPointID;
   public Time                  time;
   public UnitPoint             unitPoint;

public SpaceUnitPoint()
{
}

public SpaceUnitPoint( SpacePerceptionID pID, SpaceUnitPointID sID, Time t, UnitPoint up)
{
   perceptionID          = pID;
   unitPointID           = sID;
   time                  = t;
   unitPoint             = up;
}
public UnitPoint getUnitPoint()
{
   return unitPoint;
}
}
```

Figur 7-7 Posisjonsrapportene av klassen UnitPoint pakkes inn i klassen SpaceUnitPoint ved lagring i JavaSpaces

SpaceUnitPoint inneholder fire felter:

- perceptionID: identifiserer persepsjonen (se datamodellen) som rapporten er knyttet til.
- unitPointID: rapportens sekvensnummer
- time: kopi og innpakking av rapporterings-tidspunktet

- unitPoint: posisjonsrapporten

For å sette en rapport inn i JavaSpaces kan man benytte følgende kode (når man kjenner perceptionID og unitPointID):

```
Time            time    = new Time(report.time);
SpaceUnitPoint  sup     = new SpaceUnitPoint(perceptionID,unitPointID,time,report);
space.write(sup,null,lease);
```

Tidsrommet rapporten skal holdes angis v h a en ”lease”. Etter utløpet av denne tidsperioden blir rapporten automatisk fjernet.

For å lese ut en rapport med kjent perceptionID og tidspunkt benyttes en mal med identifikator og tid satt til de ønskede verdier og de andre feltene satt til null. JavaSpaces vil da søke etter et lagret objekt (Entry) som er i henhold til malen. For at en Entry skal passe malen må alle feltene i Entry passe malen. Dersom et av feltene som sammenlignes er lik null passer de. For å lese ut en rapport kan man f eks benytte følgende kode:

```
UnitPoint       report      = null;
SpaceUnitPoint  template    = new SpaceUnitPoint(perceptionID,null,time,null);
SpaceUnitPoint  sup         = (SpaceUnitPoint) space.readIfExists(template,null,timeout);
If(sup != null) {
        report              = sup.unitPoint;
}
```

7.2 Ruteplankomponent

Ruteplankomponentens oppgave er å assosiere radartrack med kjente ruteplaner. En ruteplan er en beskrivelse av seilasen til et fartøy inkludert hvor det skal seile, når og formålet med seilasen. En nærmere beskrivelse av innholdet i ruteplanen er forklart i kapittel 6.1.6.

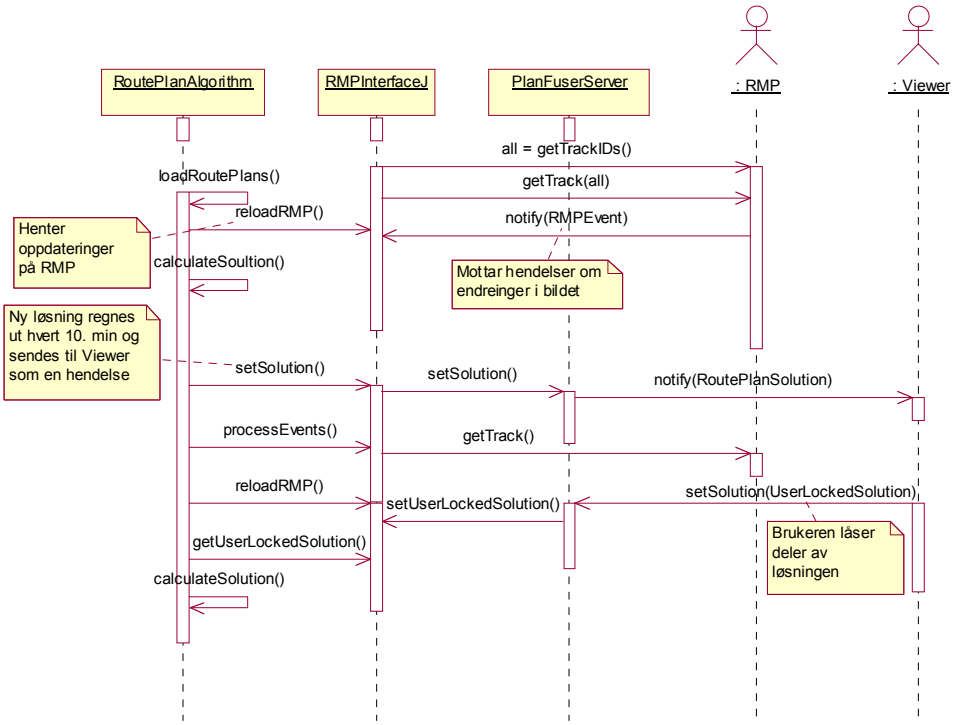
Ruteplankomponenten er basert på gjenbruk og videreutvikling av en eksisterende algoritme som er utviklet i prosjektet (29). Implementasjonen av denne er dokumentert i et eget notat (30), og for en mer detaljert beskrivelse av ruteplanalgoritmen henvises det til dette notatet. Algoritmen er implementert i C++ og må gå på UNIX-operativsystem. Denne delen av ruteplankomponenten blir heretter kalt *ruteplanalgoritmen*.

For å få tilgang til ruteplanalgoritmen i CoABS Grid må den ha et Java-grensesnitt. Det er flere måter å gjøre dette på, for MCCIS og Maria ble f eks CORBA valgt. For å få en så enkel og effektiv løsning som mulig valgte vi imidlertid å benytte Java Native Interface (JNI) som muliggjør direkte kall fra Java til C++ og omvendt uten bruk av mellomvareteknologi. Dette gjør også at Ruteplankomponenten bare går som en enkelt prosess på en maskin.

Som en del av arbeidet med demonstratoren har ruteplanalgoritmen blitt utvidet i forhold til det som er beskrevet i (30). Dette omfatter utvidelser av hvilke kriterier som gjelder for assosiasjon mellom ruteplaner og track:

- Hvis tracket er klassifisert til en annen type, kategori eller navn enn ruteplanen blir det *ikke* assosiert med ruteplanen
- Hvis tracket er navnebestemt til å ha samme navn som ruteplanen *blir* det assosiert med ruteplanen.

7.2.1 Grensesnitt og kommunikasjon mot andre komponenter



Figur 7-8 Sekvensdiagram for ruteplankomponenten

Java-delen av Ruteplankomponenten er ansvarlig for all kommunikasjon mot CoABS Grid og andre komponenter. Den videreformidler også løsningene fra ruteplanalgoritmen til klientene via en hendelsestjeneste og mottar konfigurasjonsdata fra klientene andre veien. Det er også implementert Java-klasser som lytter på endringer i RMP-komponenten og videreformidler disse til ruteplanalgoritmen slik at denne til enhver tid har et oppdatert RMP.

```
public interface PlanFuserInterface extends Remote {

   void  setSolution(UserLockedSolution solution) throws RemoteException;
   long register(long ev_id, RemoteEventListener listener) throws RemoteException;
}
```

Figur 7-9 Grensesnittet til ruteplankomponenten

Grensesnittet til Ruteplankomponenten er vist i Figur 7-9 og er implementert av klassen PlanFuserServer. Denne har både rollen som hendelsestjeneste samtidig som den representerer

tjenesten i CoABS Grid. Dette betyr at klientene registrerer seg hos den samme tjenesten som de benytter for å interaktere med ruteplankomponenten. En egen klasse, RMPInterfaceJ, har som oppgave å oversette mellom Java og C++ samt holde på hendelser mottatt fra RMP-komponenten. De innkommende hendelsene hentes av ruteplanalgoritmen etter hver gang ny løsning er beregnet.

Figur 7-8 viser et UML sekvensdiagram som illustrerer kommunikasjon mellom Ruteplan-komponenten og andre komponenter. Merk at sekvensdiagrammet forutsetter at klientene har koblet seg opp og at RMPInterfaceJ har registrert seg hos hendelsestjenesten til RMP-komponenten. Rollefordelingen mellom objektene er slik at PlanFuserServer tar for seg all kommunikasjon mot klientene, mens RMPInterfaceJ tar vare på alle data som skal tilflyte ruteplanalgoritmen. RMPInterfaceJ går derfor ikke i en egen tråd slik som PlanFuserServer, men aktiviseres av ruteplanalgoritmen. Dette for å unngå å måtte bruke semaforer og synkronisering av data og metoder.

7.2.2 Inn- og utgangsinformasjon

Ruteplanene som leses inn ved oppstart av algoritmen må legges inn manuelt. Dette betyr at ruteplandataene må skrives inn i en tekstfil i henhold til et gitt format, og at den geografiske utstrekningen til ruta må digitaliseres. Vi har gjort sistnevnte med bruk av DIMAS og lagret ruta på SOSI-format. Man må deretter benytte et separat program (Sosi2led) for å konvertere ruta til ruteplanalgoritmens internformat (dette er det samme formatet som benyttes i SensorSim).

```
public class RoutePlanSolution {

  public String KnownVesselName;
  public int[] trackIDs;
  public UnitPoint estimatedPosition;
  public UnitPoint[] uncertaintyArea;
  public double confidence;
  public int time;
}
```

Figur 7-10 Løsningen som sendes til Viewer

Løsningen som ruteplanalgoritmen sender til Viewer, *RoutePlanSolution,* er navnet på fartøyet (benyttes for å identifisere ruteplanen), tidspunktet for løsningen, identifikatoren til trackene ruteplanen er assosiert med, posisjonsestimatet til fartøyet gitt som et usikkerhetsområde og konfidensintervallet til posisjonsusikkerheten. Se Figur 7-10.

Brukeren kan velge å låse forholdet mellom et gitt track og en ruteplan og dermed tvinge ruteplanalgoritmen til å lage en løsning som er konsistent med dette valget. Dette gjøres ved å kalle SetSolution–metoden hos RoutePlanServer. Denne løsningen kan inneholde instrukser om at et gitt track *ikke* skal være assosiert med ruteplanen eller at den *skal* være det.

7.3 Klassifikasjonskomponent

Klassifikasjon av overflatetrack i demonstratoren blir utført av en egen klassifikasjons-komponent. Denne beregner et klassifikasjonsestimat basert på klassifikasjonsinformasjon fra

sensorer og brukere slått sammen med forhåndsoppsatte feiltabeller for de forskjellige sensortypene. En feiltabell angir sensortypens klassifikasjonskvalitet, og et eksempel på en slik feiltabell kan sees i Tabell 7.1.

Komponenten samhandler i demonstratoren med RMP-komponenten, se kapittel 7.1, og Viewer, se kapittel 6. Fra RMP-komponenten fås klassifikasjonsinformasjonen om et track, mens brukeren kan interaktere med klassifikasjonskomponenten gjennom en egen brukergrensesnittkomponent. Brukerinteraksjonen består i å sette klassifikasjon av track og hente fram informasjonen som ligger til grunn for et klassifikasjonsestimat.

7.3.1 Informasjonshierarkiet

I prosesseringen av klassifikasjonsinformasjon slår klassifikasjonskomponenten sammen informasjon fra flere klassifikasjonsnivåer:

- kategori: marinefartøy, handelsfartøy, fiskefartøy, undervannsfartøy, luftfartøy, landfartøy og ukjent. Siden demonstratoren er avgrenset til produksjon av overflatebildet, vil klassifikasjonsnivået 'kategori' kun ha verdiene marinefartøy, handelsfartøy, fiskefartøy og ukjent
- type: f eks fregatt eller MTB
- klasse: f eks Oslo-klassen eller Hauk-klassen
- navn: fartøyet kan klassifiseres entydig

Disse nivåene er basert på OTG (1), STANAG 4420 (5) og Mil-Std 2525A (6) og er organisert slik at hver kategori inneholder en eller flere typer, hver type inneholder en eller flere klasser og hver klasse inneholder en eller flere enkeltfartøyer. Disse klassifikasjonsnivåene finnes også i RMP-modellen (se kapittel 7.1).

7.3.2 Grensesnitt

Klassifikasjonskomponentens grensesnitt er vist i Figur 7-11. Tjenestene brukes av Viewer for henholdsvis å få bakgrunnsinformasjonen til et track, la brukeren få sette klassifikasjonen av et track og til å beregne klassifikasjonsestimatet på nytt når en brukersatt verdi slettes (se kapittel 6.1.5).

```
public interface ClassifierInterface extends Remote
{
  Info  listInfo                (short trackID, boolean resetUserInput)
  void  userInput               (short trackID, String declaration, byte level, boolean info,
                                long time)
  void  removeUserSetting       (short trackID, long time)
}
```

Figur 7-11 Klassifikasjonskomponentens grensesnitt

7.3.3 Konfigurasjon

Når klassifikasjonskomponenten startes trenger den informasjon om hvilke fartøy den kan forvente å få informasjon om (kalt Order of Battle, OoB) og klassifikasjonskvaliteten til alle sensortypene den kan forvente å få informasjon fra. Disse opplysningene hentes fra forhåndsoppsatte tekstfiler.

7.3.4 Inngangs- og utgangsinformasjon

Inngangsinformasjonen til komponenten er klassifikasjonsrapporter. En slik rapport består av:

- en klassifikasjonsdeklarasjon
- avsenderen av deklarasjonen
- hvilken type sensor som er grunnlaget for deklarasjonen
- tidspunktet for deklarasjonen

Med en klassifikasjonsdeklarasjon menes her utsagn av typen 'fregatt', som uttrykker at en sensor sier at tracket er en fregatt, 'Oslo-klassen', som uttrykker at sensoren sier tracket er av Oslo-klassen, og lignende utsagn. I tillegg til brukeren vil klassifikasjonskomponentens informasjonskilder være ESM-sensorer (Electronic Support Measures), IFF-sensorer (Identification-Friend-Foe), EO/IR-sensorer (Elektrooptisk/infrarød) og ISAR (Invers syntetisk aperture radar). Det antas her at klassifikasjonsrapportene er betinget uavhengige, noe modellen i klassifikasjonskomponenten krever (se kapittel 7.3.7).

I tillegg til klassifikasjonsrapportene trenger klassifikasjonskomponenten en kvantifisering av usikkerheten i klassifikasjonsrapportene samt en á priori-sannsynlighet for alle klassifikasjonene komponenten kan gi. Dette er beskrevet i kapittel 7.3.7.

Utgangsinformasjonen, resultatet, fra komponenten er et klassifikasjonsestimat som er utformet som den mest sannsynlige stien gjennom et klassifikasjonstre bygget opp av de forskjellige klassifikasjonsnivåene (se Figur 7-16). En slik sti starter i den mest sannsynlige noden på klassifikasjonsnivået kategori. Det neste steget på stien er denne nodens mest sannsynlige barn og så videre helt ned til klassifikasjonsnivået navn. Sammen med klassifikasjonsestimatet følger en konfidensangivelse på hvert av klassifikasjonsnivåene. Denne konfidensangivelsen er et tall mellom 0 og 1 der høye tall angir stor sikkerhet. Når brukeren setter klassifikasjon vil konfidensangivelsen være 1, se kapittel 7.3.6.

7.3.5 Samhandling med andre komponenter

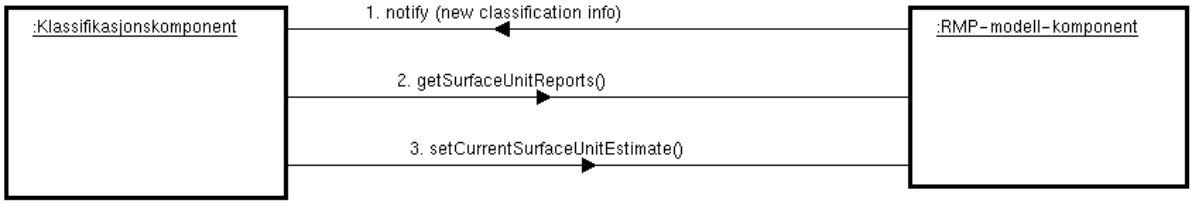
Klassifikasjonskomponenten kommuniserer med RMP-komponenten og klassifikasjonskomponentens brukergrensenittkomponent.

Samhandlingen med RMP-komponenten består i at klassifikasjonskomponenten abonnerer på hendelser gjennom RMP-komponentens hendelsestjeneste (se kapittel 7.1.2.4). Disse hendelsene er:

- NEW_TRACK_EVENT
- MOVE_PERCEPTION_EVENT

- NEW_PERCEPTION_EVENT
- PERCEPTION_DELETE_EVENT
- ADD_SURFACEUNIT_REPORT_EVENT
- DELETE_SURFACEUNIT_REPORT_EVENT

Når en av disse hendelsene er mottatt, henvender klassifikasjonskomponenten seg til RMP-komponenten for å få alle klassifikasjonsrapportene som finnes for dette tracket. Da kan den, på grunnlag av disse rapportene, beregne en á posteriori-fordeling for klassifikasjonen av det aktuelle tracket. Á posteriori-fordelingen vil være klassifikasjonskomponentens konfidensfordeling over alle mulige klassifikasjoner av tracket når den nye informasjonen er tatt hensyn til. Klassifikasjonsrapportene hentes i form av SurfaceUnit-objekter ved hjelp av RMP-komponentens getSurfaceUnitReports-metode. Når det er gjort, melder den resultatet, se kapittel 7.3.3, tilbake til RMP-komponenten som et SurfaceUnit-objekt ved hjelp av RMP-komponentens tjeneste setCurrentSurfaceUnitEstimate. Da kan RMP- komponenten oppdatere dette trackets CurrentEstimate. Denne samhandlingen er vist i Figur 7-12 som et "collaboration diagram" i UML.



Figur 7-12 Samhandlinga med RMP-modell-komponenten

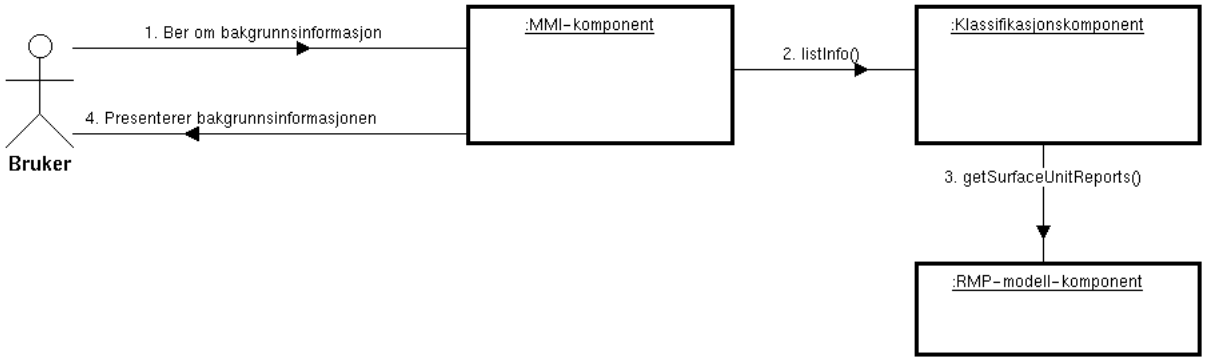
Samtidig som klassifikasjonskomponenten oppdaterer det aktuelle trackets CurrentEstimate, sjekker den om det nye estimatet er i konflikt med estimatet som lå i CurrentEstimate før den siste oppdateringen. Dersom dette er tilfelle, settes den boolske attributten conflict i CurrentEstimate til sann verdi.

I tillegg kommuniserer klassifikasjonskomponenten med brukeren gjennom en komponent (se kapittel 6.1.5). Brukeren kan gjennom denne komponenten overstyre klassifikasjonen av et track og be om bakgrunnsinformasjon om et tracks klassifikasjonsestimat. Dette gjøres ved at komponenten bruker henholdsvis klassifikasjonskomponentens tjenester userInput og listInfo.

7.3.6 Interaksjon med brukeren

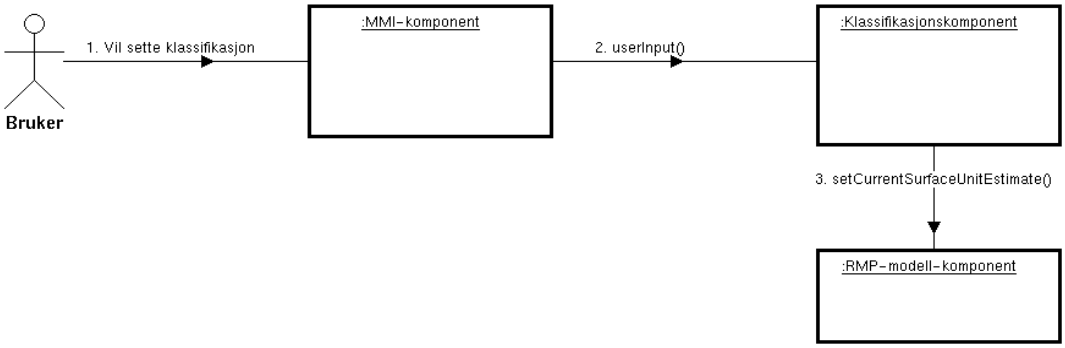
I enkelte tilfeller kan brukeren være uenig eller være i tvil om hva som ligger bak klassifikasjonsestimatet. Derfor kan informasjonen som ligger til grunn for klassifikasjonsestimatet presenteres for brukeren på forespørsel. På grunnlag av denne informasjonen kan brukeren vurdere estimatet som er gitt og eventuelt gå inn og overstyre den automatiske klassifiseringen. Informasjonen som vil bli presentert for brukeren i en slik situasjon er hele á posteriori-fordelinga, selve klassifikasjonsestimatet og en liste over hvilke sensorer som har bidratt, hva slags type sensorer de er og hva slags deklarasjoner de er kommet

med, se kapittel 6.1.5. Denne interaksjonen er vist i Figur 7-13.



Figur 7-13 Interaksjon med brukeren: brukeren ønsker å se bakgrunnsinformasjonen for klassifikasjonen av et track

Brukeren kan også sette klassifikasjonen av et track direkte. Når brukeren gjør dette, låses klassifikasjonen til verdien brukeren har satt den til og brukeren må aktivt be om at dette omgjøres for at senere deklarasjoner skal tas hensyn til. Denne interaksjonen er vist i Figur 7-14.



Figur 7-14 Interaksjon med brukeren: brukeren ønsker å sette klassifikasjonen av et track

7.3.7 Modell for klassifikasjonsfusjon

Klassifikasjonskomponenten bruker bayesiansk datafusjon for å slå sammen (fusjonere) klassifikasjonsdeklarasjoner (se (11)). Denne metoden er regneeffektiv, men krever at á priori-fordelinga av OoB og alle sensormodeller er satt opp på forhånd. I tillegg krever den at alle klassifikasjonsdeklarasjonene som skal fusjoneres er betinget uavhengige.

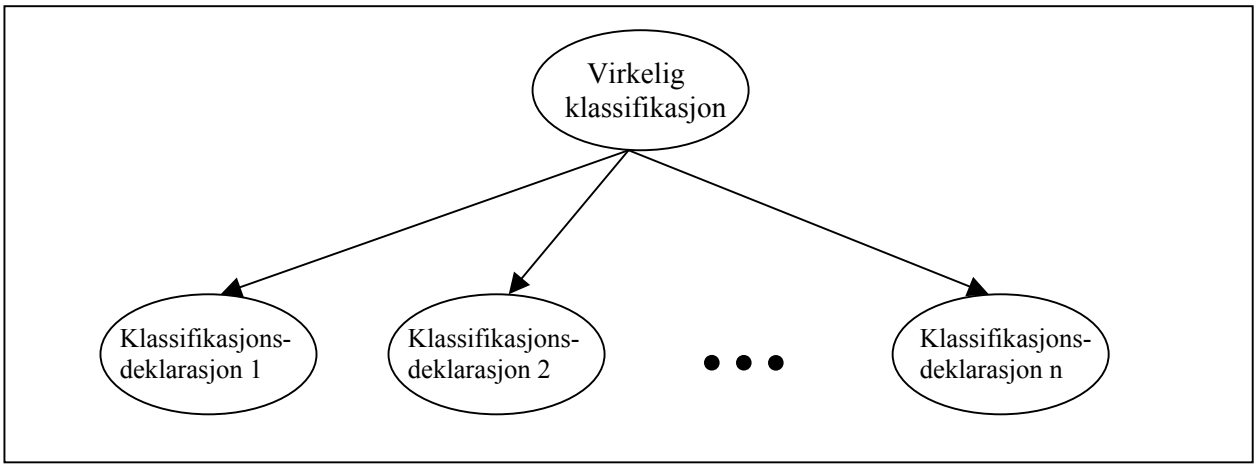
Når det gjelder á priori-fordelingen, er den i denne klassifikasjonskomponenten valgt til å være uniform. Dette betyr at alle klassifikasjoner på klassifikasjonsnivå 'navn' antas å være like sannsynlige før noen deklarasjoner er mottatt. Komponenten beregner selv á priori-sannsynlighetene basert på OoB (se kapittel 7.3.3).

$P(ESM \mid Virkelig)$		Virkelig Fregatt	Virkelig MTB	Virkelig Minerydder
ESM	Fregatt	0,8	0,05	0,15
	MTB	0,1	0,7	0,15
	Minerydder	0,05	0,05	0,5
	Ukjent	0,05	0,2	0,2

Tabell 7.1 Et eksempel på en likelihoodfunksjon for en sensor på et gitt klassifikasjonsnivå, her en ESM-sensor på klassifikasjonsnivå 'type'

Sensormodellene tar sikte på å modellere sensorens klassifikasjonskvalitet, og er satt opp i tabeller lik Tabell 7.1. Disse tabellene er i det videre kalt likelihoodfunksjoner. Hver sensortype har fått satt opp en likelihoodfunksjon for hvert klassifikasjonsnivå der den kan gi en deklarasjon. Dermed blir det mange og til dels store likelihoodfunksjoner, noe som gjør det til en stor og tidkrevende jobb å holde dem oppdatert. Innholdet i slike likelihoodfunksjoner kan baseres på f eks etterretningsinformasjon, ekspertvurderinger eller tekniske tester av sensorene, og er satt opp i filer i forkant av demonstratorkjøringene, se kapittel 7.3.3. Selve likelihood-funksjonen kan tolkes slik: horisontalt finnes alle klassifikasjonsmuligheter for det aktuelle tracket på det aktuelle klassifikasjonsnivået, mens alle deklarasjonsmuligheter fra sensoren på klassifikasjonsnivået finnes vertikalt. Tallene i likelihoodfunksjonen er da sannsynligheten for en bestemt deklarasjon gitt trackets virkelige klassifikasjon. Tallet 0,8 i ruta 'Fregatt'/'Fregatt' i Tabell 7.1 betyr dermed at ESM-sensoren i dette scenariet har en sannsynlighet på 0,8 for at den måler 'Fregatt' når det virkelig er en fregatt det er snakk om.

Den bayesianske datafusjonen i klassifikasjonskomponenten er implementert i et Bayesnettverk (se (9) og (11)). Det overordnede Bayesnettverket som brukes i klassifikasjonskomponenten er vist i Figur 7-15. For forklaring av et slikt Bayesnettverk henvises leseren til (11).

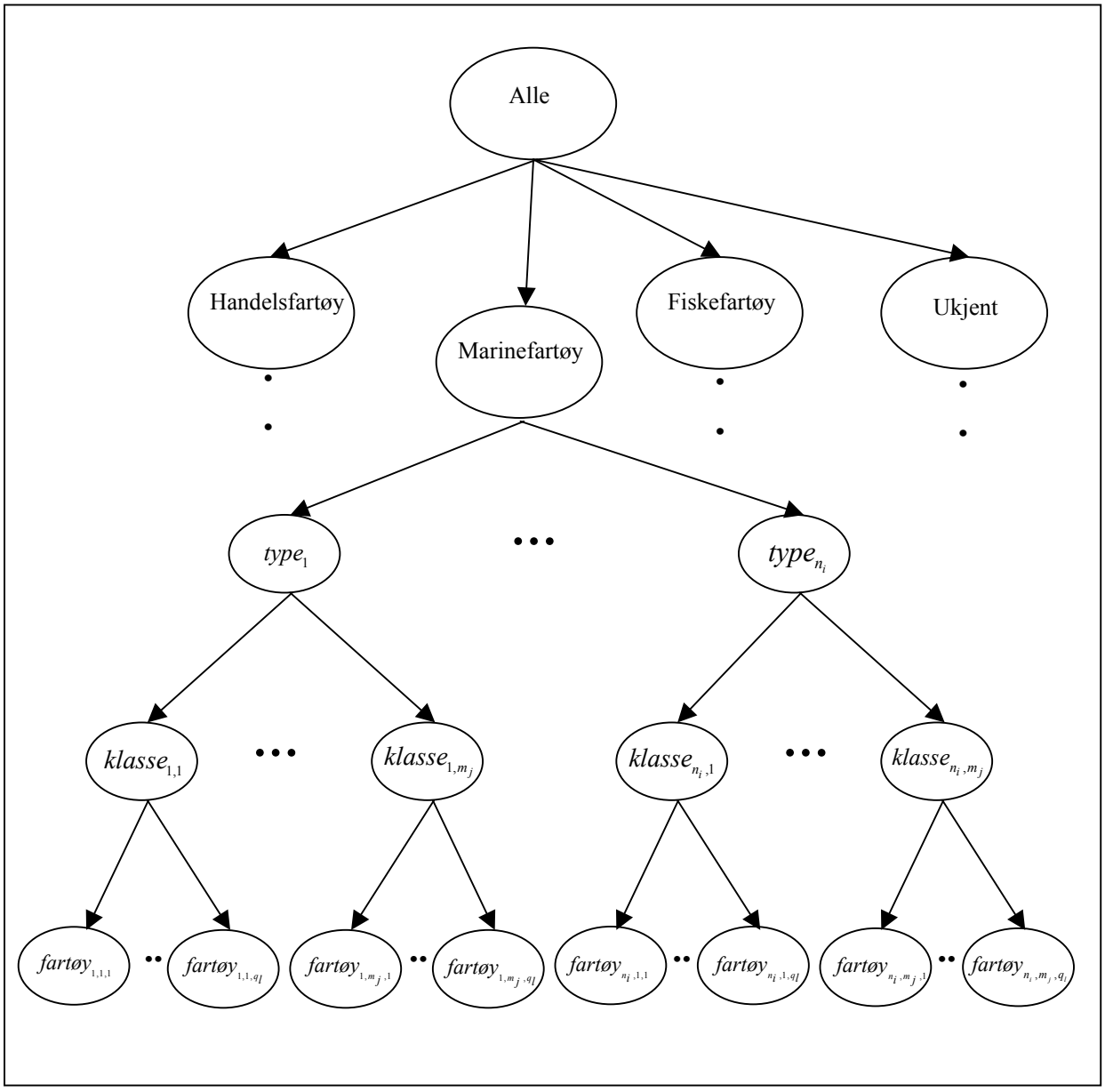


Figur 7-15 Det overordnede Bayesnettverket

Variablene i Bayesnettverket er:

- ’Virkelig klassifikasjon’
- ’Klassifikasjonsdeklarasjon 1’
- ’Klassifikasjonsdeklarasjon 2’

og så videre.



Figur 7-16 Tilstanden til 'Virkelig klassifikasjon' satt opp i et Bayesnettverk. Det er her 4 kategorier som hver har n_i typer, $i = 1,...,4$, som hver har m_j klasser, $j = 1,...,n_i$. Hver klasse har q_l fartøyer, $l = 1,...,m_j$. Indekseringa i figuren er slik at den første indeksen står for type, den andre for klasse mens den tredje står for enkeltfartøy

De mulige tilstandene til variablene ‘Klassifikasjonsdeklarasjon 1’, ‘Klassifikasjonsdeklarasjon 2’ o s v. vil være alle deklarasjonene den bestemte sensoren kan gi. Dersom en sensor i en gitt

situasjon kun kan klassifisere på nivå 'kategori' (se kapittel 7.3.1) vil tilstandsrommet f eks være: {marinefartøy, handelsfartøy, fiskefartøy, ukjent}.

De mulige tilstandene til variabelen 'Virkelig klassifikasjon' kan gis en trestruktur og settes opp i et Bayesnettverk som vist i Figur 7-16. Dette Bayesnettverket kalles i det videre et klassifikasjonstre, og det har variable med tilstandene 'Ja' og 'Nei'. M.a.o. er objektet som skal klassifiseres klassifisert som '$type_2$' dersom variabelen '$type_2$' har tilstand 'Ja'.

For å få sensordeklarasjonene inn i Bayesnettverket i Figur 7-16, må Bayesnettverket i Figur 7-15 slås sammen med det i Figur 7-16. Det gjøres her ved at sensordeklarasjonene, som er såkalt 'hard evidence' (én bestemt klassifikasjon: *fregatt*), omgjøres til såkalt 'soft evidence' (en fordeling over flere klassifikasjoner, f eks: {fregatt: 0,8, mtb: 0,05, minerydder: 0,15} ved hjelp av likelihoodfunksjoner lik den i Tabell 7.1, og gis til hver sin node på det aktuelle klassifikasjonsnivået. Resultatet blir da et Bayesnettverk som er forskjellig etter hvilken sensor som gir klassifikasjonsdeklarasjonen og hvilket klassifikasjonsnivå deklarasjonen gis på. Disse Bayesnettverkene vil se ut som Bayesnettverket vist i Figur 7-16. For en mer utdypende forklaring av begrepene 'hard evidence' og 'soft evidence' henvises leseren til (9).

Bayesnettverket skal beregne hvordan en deklarasjon påvirker á posteriori-sannsynlighetene også på andre klassifikasjonsnivåer enn nivået deklarasjonen ble gitt på. Dette kalles propagering av á posteriori-sannsynligheter i klassifikasjonstreet, og gjøres ved hjelp av overgangssannsynlighetsmatriser som forteller hvilken sammenheng det er mellom de forskjellige nivåene i Bayesnettverket. Et eksempel på en slik overgangssannsynlighetsmatrise er vist i Tabell 7.2. I klassifikasjonskomponenten er overgangssannsynlighetsmatrisene basert på hvor mange enkeltfartøy de forskjellige klassifikasjonene på de forskjellige nivåene inneholder. Overgangssannsynligheten i Tabell 7.2 vil derfor si at 40% av fregattene i scenariet er av Oslo-klassen og 60% er av Fridtjof Nansen-klassen, mens 90% av MTBene er av Hauk-klassen og 10% er av Skjold-klassen.

$P(klasse \mid type)$		Type	
		Fregatt	MTB
Klasse	Oslo-klassen	0.4	0.0
	Fridtjof Nansen-klassen	0.6	0.0
	Hauk-klassen	0.0	0.9
	Skjold-klassen	0.0	0.1

Tabell 7.2 En overgangssannsynlighetsmatrise fra nivå 'type' til nivå 'klasse'

7.3.8 Implementasjon

Klassifikasjonskomponenten er implementert som en tjeneste som registrerer seg i CoABS Grid (se kapittel 5.2). Der venter den på kall fra brukergrensesnittkomponenten (se kapittel 6.1.5) og hendelser fra RMP-komponenten (se kapittel 7.1.2.4).

Selve fusjonen foregår i et objekt av klassen Classifier, mens hendelsen fra RMP-komponenten

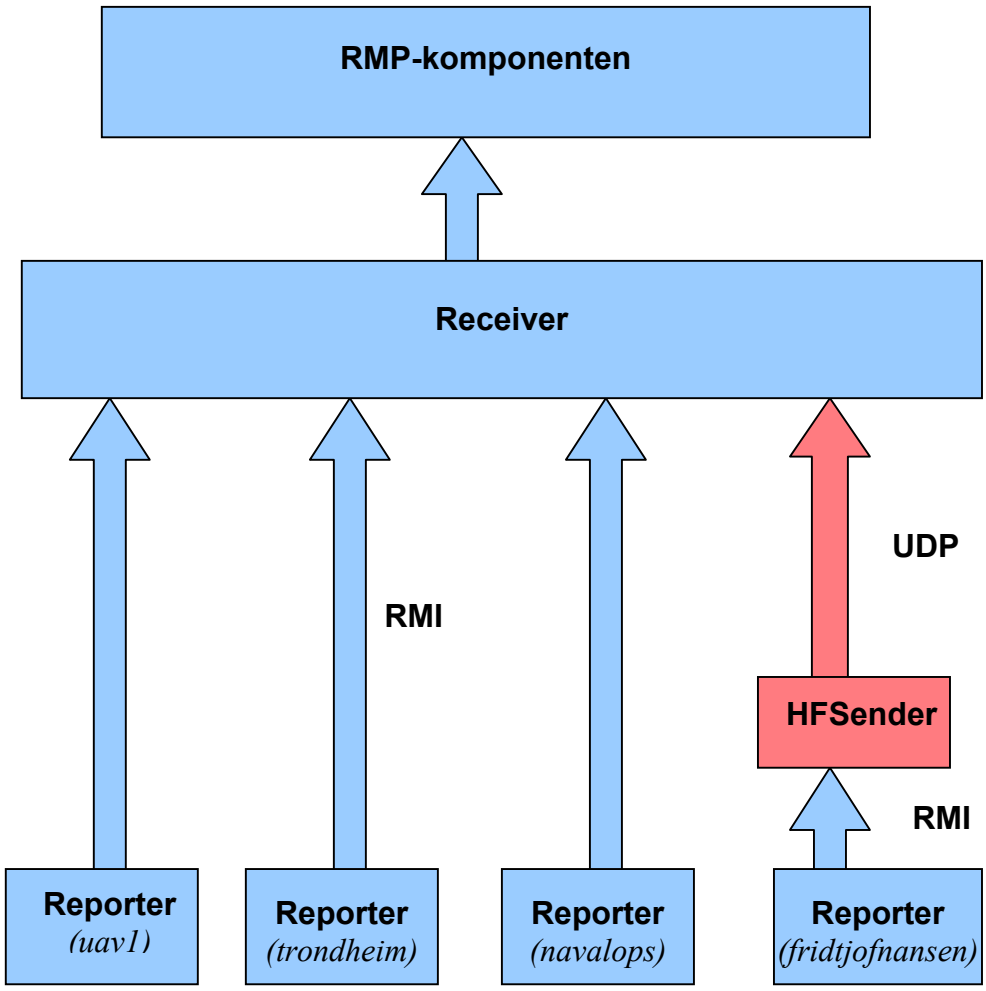
mottas i tjenesten notify() i et objekt av klassen TrackEventsReceiver. TrackEventsReceiver har en kobling til Classifier og kaller i sin notify-metode newClassificationInfo hos Classifier for å starte klassifikasjonsfusjonen.

Klassifikasjonskomponenten holder orden på hvilke sensorer som har rapportert det aktuelle tracket, hvilke deklarasjoner sensorene har kommet med og hvordan likelihoodmatrisene til sensorene ser ut gjennom at Classifier har koblinger til objekter av klassene Sensor, Declaration og LikelihoodMatrix.

Komponenten lagrer ingen informasjon om trackene i situasjonsbildet. Hver gang RMP-komponenten eller brukergrensesnittkomponenten ønsker oppdatert klassifikasjonsinformasjon om et track, lastes all informasjon og klassifikasjonskomponenten beregner klassifikasjonsestimatet på nytt.

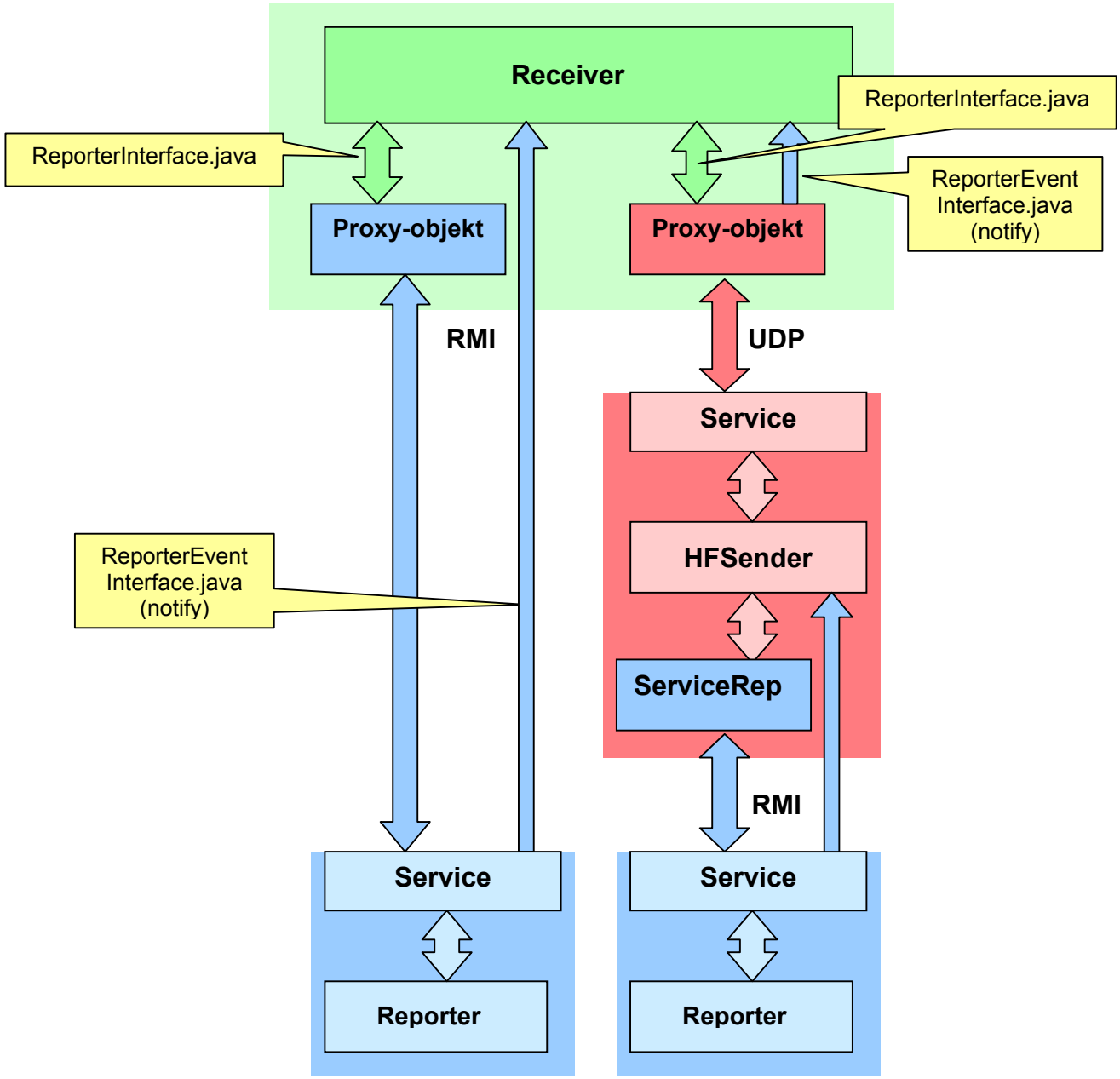
7.4 Rapporterings- og mottakstjenestene

Dette kapittelet beskriver koblingen mellom sensorer og informasjonskilder og (de sentrale delene av) demonstratoren. Demonstratorens sensorer og informasjonskilder er simulerte, og



Figur 7-17 Rapportering og innhenting av data

for en beskrivelse av simuleringsomgivelsen henvises det til kapittel 10. De simulerte sensorene og informasjonskildene registrerer seg i CoABS Grid som om de var virkelige sensorer og informasjonskilder, og for denne beskrivelsen er det uvesentlig om dataene kommer fra simuleringsomgivelsen eller fra den virkelige verden.



Figur 7-18 Koblingen mellom sensorer og informasjonskilder og (de sentrale deler av) demonstratoren

Det er tre komponenttyper som inngår i overføring av data fra sensorer og informasjonskilder til (de sentrale deler av) demonstratoren:

- Reporter som representerer sensorer og informasjonskilder
- HF-sender som rapporterer data med UDP (for transmisjon på HF)

- Receiver som mottar (henter) data og lagrer dem i RMP-komponenten

Forholdet mellom komponentene er skissert i Figur 7-17, og litt mer detaljert i Figur 7-18. Receiver knytter seg opp til de aktuelle Reporter-ene som har registrert seg i CoABS Grid. For å demonstrere HF-samband mellom sensorer og (sentrale deler av) demonstratoren, ble det laget en egen rapporteringstjeneste, HF-Sender, som benytter UDP for transport av data mellom tjenesten og klienten.

En Reporter registrerer sin tjeneste i CoABS Grid med en beskrivelse og et proxy-objekt. En Receiver knytter seg til tjenesten ved å laste ned tjenestens registrerte proxy-objekt som benyttes for datautveksling med tjenesten. Merk at transportprotokollen mellom proxy-objektet og tjenesten er skjult for klienten (Receiver). Proxy-objektet til Reporter benytter Java's standard protokoll, RMI (Remote Method invocation) for datautveksling mens HF-Sender sitt proxy-objekt benytter UDP datagram for datautvekslingen.

7.4.1 Reporter

En Reporter representerer en sensor eller en informasjonskilde i CoABS Grid. Ved oppstart knytter Reporter seg til en HLA-føderasjon og registrerer seg som en tjeneste i CoABS Grid med et proxy-objekt, navn og beskrivelse. Navnet blir gitt som et argument ved oppstart av Reporter. Beskrivelsen starter med RMI for å indikere at proxy-objektet benytter Java RMI som protokoll mot tjenesten.

Reporter sitt grensesnitt er definert i Figur 7-19. Proxy-objektet implementerer dette grensesnittet. Grensesnittet har metoder for å hente ut henholdsvis posisjons- og kontakt-rapporter og en metode for å registrere seg som mottaker av hendelser.

```
public interface ReporterInterface extends Remote {

   UnitPoint        getUnitPointReport    (int reportedTrackID, int reportTime)
   SurfaceUnit      getSurfaceUnitReport(int reportedTrackID, int reportTime)
   long register(long ev_id, RemoteEventListener listener)
   void deregister(RemoteEventListener listener)
}
```

Figur 7-19 Reporter-tjenesten sitt grensesnitt

Når nye data blir tilgjengelige fra simuleringsomgivelsen, sender sensoren en hendelse til de Receiver-instansene som har registrert seg som mottakere av hendelser. Receiver-ene kan deretter hente data fra Reporter vha proxy-objektets metoder.

For en nærmere beskrivelse av Reporter og simuleringen av sensorer og informasjonskilder henvises det til kapittel 10.

7.4.2 HF-sender

HF-sender mottar data fra en Reporter og sender dataene videre til en Receiver. Ved oppstart

knytter HF-senderen seg til en Reporter som ligger i argumentet fra kommandolinja, og registrerer seg som rapporteringstjeneste i CoABS Grid med det samme navnet som Reporter-en, men med en tjenestebeskrivelse som begynner med *UDP* (i stedet for *RMI* som benyttes av reporterene). Under demonstrasjonene har vi latt HF-senderen knytte seg opp mot Reporteren som videreformidler data fra *fridtjofnansen*.

Når HF-Senderen mottar en hendelse fra den Reporter den har koblet seg opp mot, henter den både siste kontakt- og posisjons-rapport fra Reporter, genererer en track-melding som inneholder rapportene og sender denne til proxy-objektet (som Receiver-en har lastet ned) i form av et UDP-datagram på en port som er assosiert med hendelsen. Proxy-objektet (re-)genererer hendelsen, sender dette (lokalt) til Receiver-en og gjør kontakt- og posisjonsrapportene klare til avhenting. Receiver-en kan da hente posisjons- og kontaktrapportene fra proxy-objektet på samme måte som om de ble hentet fra proxy-objektet til Reporter.

En UDP track-melding inneholder både en kontakt- og en posisjonsrapport. Fordelen med å benytte UDP er at denne protokollen ikke benytter kvitteringer som gjør det nødvendig å reversere sendingen med de forsinkelser som er forbundet med dette. Track-meldingen er kompakt kodet for sending på HF og utgjør 134 bytes data.

7.4.3 Receiver

Receiver mottar data fra Reporter-ene som er registrert i CoABS Grid og sender dataene videre til Rmp-komponenten.

Ved oppstart knytter Receiver-en seg opp mot RMP-komponenten. Dersom denne ikke er tilgjengelig, venter Receiver-en på den. Deretter ser Receiver-en etter rapporteringstjenester som har registrert seg i CoABS Grid, og kobler seg opp mot dem for mottak av data. Receiver sjekker jevnlig om nye rapporteringstjenester har registrert seg, og knytter seg opp mot disse.

Ved oppstart benyttes argumentene til å ekskludere tilkobling til en Reporter. Det første argumentet er enten *RMI* eller *UDP*. Dersom argumentet er lik *RMI*, så vil Receiver ikke knytte seg til en rapporteringstjeneste (proxy-objekt av type ReporterInterface) med tjenestebeskrivelse som starter med *RMI* og med navn lik det andre argumentet på kommandolinja.

Under demonstrasjonene har vi latt en HF-sender knytte seg opp mot rapporteringstjenesten til *fridtjofnansen* og Receiver-en har da knyttet seg opp mot HF-senderen og mottatt dataene fra *fridtjofnansen* via denne. For å unngå at Receiver-en også knytter seg direkte opp mot *fridtjofnansen* slik at vi får dobbeltrapportering, har vi kjørt Receiver-en med *RMI* som det første argumentet og *fridtjofnansen* som det andre argumentet.

7.5 Posisjonsestimator

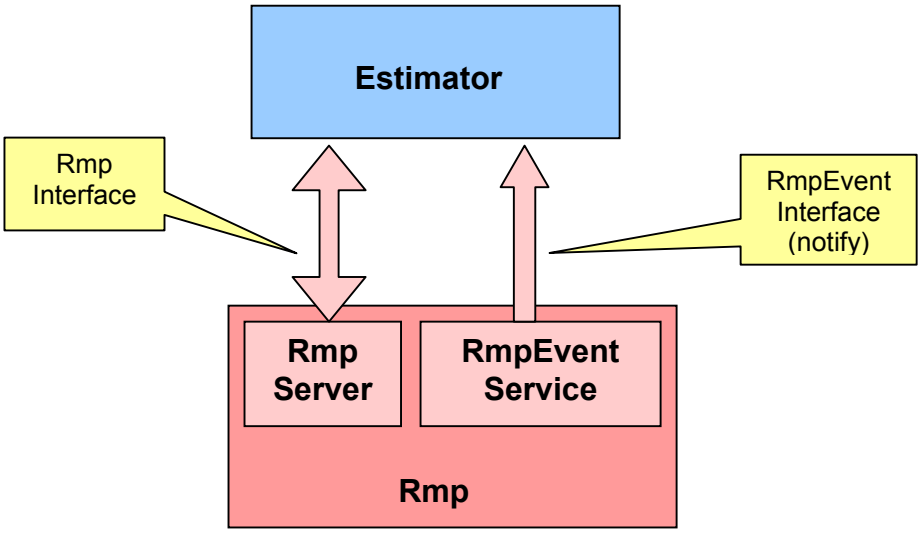
Posisjonsestimatoren fusjonerer posisjonsrapportene fra de rapporterende enhetene til et posisjonsestimat, currentEstimate, som lagres i RMP-komponenten.

Basert på posisjonsrapportene fra alle rapporterende enheter med tilhørende posisjons-usikkerheter i form av feilellipser, estimerer denne komponenten posisjonen til et track referert til tidspunktet for mottak av siste posisjonsrapport. Posisjonsrapportene fra de rapporterende enhetene vil være basert på lokal tracking. De vil med andre ord være resultatet fra en lokal trackingprosess, og ikke basert på rådata fra sensorene. Dette gjør at posisjonsestimatoren blir forskjellig fra et typisk målfølgingsfilter tilknyttet et radarsystem.

Rapporteringen av et track kan baseres på flere rapporteringskriterier. Et av disse kan være at tracket rapporteres på nytt av en rapporterende enhet når en fremskriving (prediksjon) av posisjonen basert på siste rapport gir en posisjon som er for unøyaktig i henhold til et rapporteringsdirektiv. I dette tilfellet vil en optimal estimator stort sett basere seg på siste måling. Ved hyppig rapportering vil det være en sterk korrelasjon mellom rapportene fra samme rapporterende enhet og det vil være lite ekstra informasjon å utnytte fra eldre rapporter, slik at i dette tilfellet vil også en optimal estimator stort sett basere seg på den siste rapporten.

I et operativt system vil posisjonsestimatoren kunne basere seg på siste rapport fra hver av de rapporterende enhetene, og beregne et posisjonsestimat basert på angitte posisjonsnøyaktigheter og rapporteringstidspunkt. Estimatoren vil med andre ord beregne en veid middelverdi (av siste posisjonsrapport fra hver rapporterende enhet) som tar hensyn til posisjonsnøyaktighetene og rapporteringstidspunktet.

Demonstratorens posisjonsestimator, Estimator, er omtalt nedenfor.



Figur 7-20 Estimator henter posisjonrapporter fra Rmp-modellen og beregner et nytt posisjonsestimat (current estimate) som lagres i Rmp-modellen

7.5.1 Estimator

Estimatoren henter posisjonsrapporter fra RMP-komponenten, beregner posisjonsestimater og setter disse inn i RMP-komponenten som currentEstimate. For å kommunisere med RMP-

komponenten kobler Estimatoren seg opp mot RMP-komponentens to tjenester, RmpServer og RmpEventService.

Når en ny posisjonsrapport blir lagt inn i RMP-komponenten, får estimatoren et varsel om dette i form av en hendelse. Estimatoren henter da siste posisjonsrapport fra hver av enhetene som rapporterer tracket fra RMP-komponenten, men i stedet for å beregne et optimalt posisjonsestimat, tar estimatoren den nyeste posisjonsrapporten og setter den inn i RMP-komponenten som currentEstimate. Det vil være forholdsvis enkelt å implementere full funksjonalitet i et operativt system.

8 ARVEN

8.1 Karttjenesten

Maria MapServer er en eksperimentell kartserver utviklet og levert av Teleplan AS. Kartserveren inngår som kartkomponent i demonstratoren og har som oppgave å levere nødvendig kart til Viewer som bakgrunn for presentasjon av situasjonsbildet. Kartserveren er en enkel teknologidemonstrator og gjenspeiler, ifølge Teleplan AS, ikke kapasitetene til en framtidig CORBA løsning mot MARIA versjon 4.

Maria MapServer nås fra omverdenen vha CORBA definert ved et IDL grensesnitt (se appendiks A) og kan levere kart på rasterformat samt foreta transformasjoner mellom geografiske koordinater og skjermkoordinater. Kartserveren benytter MARIAs kartbiblioteker og MARIAs template-filer for å definere kartinnholdet.

Maria MapServer tilbyr en funksjon for å generere kart og to funksjoner for koordinattrans-formasjoner. Funksjonen GenerateMap returnerer et rasterkart med utgangspunkt i gitte parametere. Selve kartinnholdet er definert i en template som er generert med MARIA (versjon 3.5 eller høyere). Funksjonen GetPixelCoordinates returnerer kartets x,y-posisjon beregnet ut fra en gitt bredde- og lengdegrad i kartet. Funksjonen GetDecimalGrades beregner geografiske koordinater ut fra en gitt x,y-posisjon i kartet. Maria MapServer returnerer kartdata i fire formater: PNG, JPG, BMP og PPM. Parametervalgene er som følger: M_PNG_FAST, M_PNG_BEST, M_JPG_75, M_JPG_90, M_BMP24, M_PPM. Hvilke variasjoner som skjuler seg bak spesifikasjoner som FAST, BEST, 75, 90 og 24 er ikke spesifisert. I demonstratoren benyttes formatet PNG og kartserveren kalles med parameteren M_PNG_FAST.

Mercator er eneste tilgjengelige projeksjon. Dette er en projeksjon som er lite egnet for kart i nordlige deler av Barentshavet og kan ikke brukes i polområdene. Eneste tilgjengelige datum er WGS84.

8.1.1 Template-filer

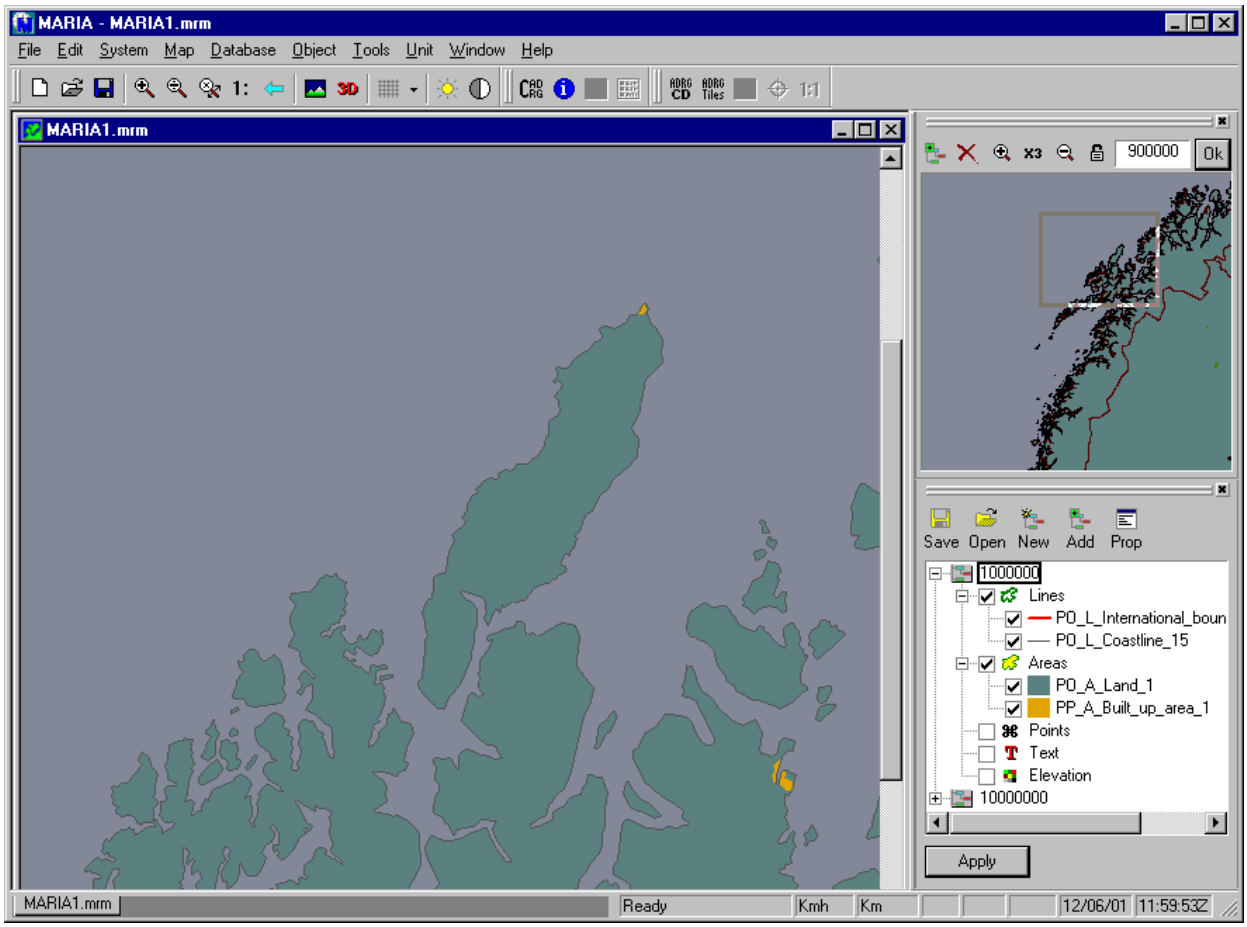
For bruk i demonstratoren er det generert tre template-filer: DemoL, DemoM og DemoH.

Bokstavene L, M og H står for henholdsvis Low, Medium og High, og betegner mengden karttemaer som benyttes for å definere kartet.

DemoL er den template-filen som normalt benyttes som bakgrunn i Viewer. Kartet som denne beskriver, er bygget opp av fire karttemaer fra kartdatabasen Digital Chart of the World (DCW). Disse er to linjetemaer som beskriver internasjonale grenser og kystlinjer, og to flatetemaer som beskriver landområder og tettbebyggelse. I tillegg er det spesifisert en bakgrunnsfarge som benyttes der flatetemaene ikke dekker, d v s havområdene. Figur 8-1 viser et eksempel på kart generert vha DemoL.

DemoM er bygget opp av 8 til 24 forskjellige karttemaer fra DCW, avhengig av hvilke av de fire tilgjengelige målestokkgruppene som er valgt.

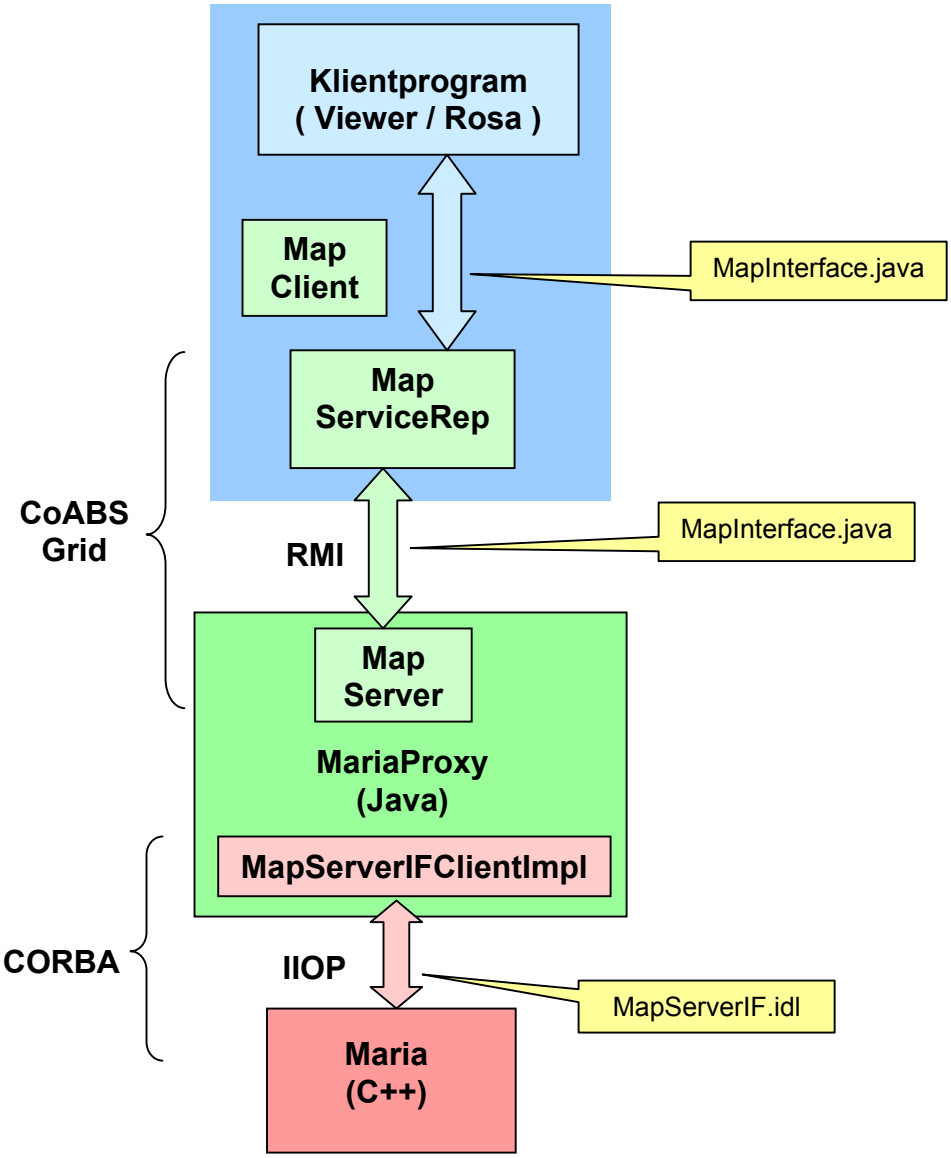
DemoH er en kombinasjon av DCW og den norske kartdatabasen N250. DCW ligger underst og over norske områder legges N250. DemoH er bygget opp av 8 til 64 forskjellige karttemaer, avhengig av hvilke av de 6 tilgjengelige målestokkgruppene som er valgt.



Figur 8-1 *Utsnitt fra Vesterålen basert på template-filen DemoL. Aktive karttemaer er angitt nederst til høyre.*

8.2 Karttjenesten MariaProxy

I CoABS Grid representeres karttjenesten av MariaProxy. MariaProxy er en tjeneste i CoABS Grid og en klient til Maria MapServer. Dette er vist i Figur 8-2.



Figur 8-2 MariaProxy er en tjeneste i CoABS Grid og en CORBA-klient til karttjenesten i Maria

Grensesnittet mellom klientprogrammet i demonstratoren og MariaProxy defineres av et Java-grensesnitt, MapInterface, som er vist i Figur 8-3. MapInterface har tre metoder som gjenspeiler Maria MapServer-grensesnittet omtalt i forrige kapittel. En for å hente et kart i form av et bilde som dekker et geografisk område, en for å finne den geografiske posisjonen til et punkt i kartbildet, og en for å finne et punkt i kartbildet som representerer en gitt geografisk posisjon. Kartbildet kan overføres på et av flere standard bildeformat. Vi benytter PNG som er innebygd i Java, og kartbildet kan derfor tegnes på skjermen uten omformatering.

```
public interface MapInterface extends Remote
{
  MapData generateMap(int format, String template, int width, int height, double west,
     double south, double north) throws RemoteException;

  JPosition getDecimalGrades( int width, int height, double west, double south,
                        double north, Point p) throws RemoteException;

  Point getPixelCoordinates(int width, int height, double west, double south,
                        double north, JPosition p) throws RemoteException;
}
```

Figur 8-3 Karttjenesten MariaProxy sitt grensesnitt

MapServerIFClientImpl er en CORBA-klient som er skrevet i Java med bruk av JBuilder 4 Professional, som genererte en stor del av koden automatisk fra Maria MapServers IDL-grensesnitt. De to grensesnittene er like men ikke identiske og MariaProxy omformatterer dataene fra dataformat som benyttes av CORBA til demonstratorens dataformat før de videresendes. Selve kartbildet omformateres ikke.

For å gjøre det enklere å implementere klientprogrammene (Viewer og testprogrammet Rosa) ble det laget en hjelpe-klasse, MapClient, for oppkobling mot tjenesten. MapClient sørger for oppkobling mot tjenesten ved at den søker i CoABS Grids oppslagtjeneste etter en tjeneste med MapInterface grensesnitt samt laster ned tjenestens proxy-objekt og returnerer dette til klientprogrammet.

8.3 Integrasjon av MCCIS

MCCIS er det mest benyttede systemet for produksjon av RMP innen NATO. MCCIS er et modent system med velprøvd funksjonalitet, men med en gammeldags arkitektur. Et viktig aspekt med demonstratoren var å forsøke å gjøre gjenbruk av utvalgte deler av funksjonaliteten i MCCIS.

MCCIS har mange funksjonsområder slik at vi var avhengige av å gjøre en avgrensning. Vi valgte å benytte trackhåndteringsfunksjonaliteten, som er den funksjonaliteten som benyttes for å manipulere trackdatabasen i MCCIS. Ved å benytte denne slapp vi å implementere egen track-korrelasjonsfunksjonalitet i demonstratoren, samt at vi fikk muligheter for å synkronisere demonstratorens trackdatabase (”RMP Modell”) mot MCCIS trackdatabase.

Trackhåndteringsfunksjonaliteten og trackdatabasen er selve kjernen i MCCIS og er en del av INRI C4I Software (ICS) (17). Trackhåndteringsfunksjonaliteten er tilgjengelig gjennom et ”Application Programmers Interface” (API) som heter ”Track Database Manager – Tdbm”. Vår strategi var derfor å gjøre relevante deler av denne API-en tilgjengelig for eksterne komponenter ved hjelp av CORBA. Valget av CORBA medfører at grensesnittet er språknøytralt noe som er en fordel hvis en skal integrere MCCIS mot komponenter/systemer som ikke er skrevet i Java. En annen grunn til valget av CORBA var at Tdbm API er implementert i programmerings-

språket C samtidig som at operativsystemet som MCCIS benytter (hp-ux) ikke fullt ut støtter Java.

For å gjøre trackhåndteringsfunksjonaliteten eksternt tilgjengelig ble det implementert en CORBA tjener (TdbmServer) i tillegg til at endringer i trackdatabasen blir gjort tilgjengelig over CORBAs ”Event”-tjeneste (EventServer). Disse grensesnittene er forklart i det påfølgende kapitlet.

8.3.1 Grensesnitt mot intern funksjonalitet i MCCIS

Informasjonen i MCCIS trackdatabase kan hentes ut og manipuleres gjennom eksisterende biblioteksrutiner (Tdbm API). Deler av dette API-et er gjort tilgjengelig gjennom en CORBA tjener heretter kalt ”TdbmServer”. Hendelser generert av endringer i databasen rapporteres over en socket og er også tilgjengelig gjennom dette API-et. Disse endringene sendes over CORBA hendelsestjeneste som heretter kalles ”EventServer”.

8.3.1.1 TdbmServer

TdbmServer eksponerer utvalgte deler av trackhåndteringsfunksjonaliteten i MCCIS. TdbmServer tilbyr metoder for å:

- Hente en liste over alle track i databasen. Metoden returnerer en liste av trackreferanser.
- Hente informasjon om klassifikasjonen til fartøyet (plattformdata).
- Hente historien (alle posisjoner) til et track. Historien består av en sekvens av datastrukturer av typen IcReport.
- Hente siste posisjon til et track.
- Hente posisjonen til et track ved et gitt tidspunkt.
- Oppdatere data om en plattform (klassifikasjonsdata).
- Opprette nye track i databasen.
- Legge til en ny posisjonsrapport for et track.
- Slette et track.
- Slette en posisjonsrapport tilhørende et track.
- Hente nåværende dato/tid i MCCIS.

Metodene til TdbmServer er vist i Figur 8-6. Grensesnittet mot TdbmServer definerer også tre datastrukturer:

- IcPlatformData (Figur 8-4) inneholder informasjon om klassifikasjonen for et fartøy.
- IcReport (Figur 8-5) inneholder en posisjonsrapport.
- IcAou (Figur 8-4) inneholder usikkerhet til en posisjonsangivelsen i en rapport.

Hver datastruktur inneholder en maske (field_mask) som angir hvilke felter i strukturen som er benyttet. Innholdet i de øvrige feltene er ikke definert.

Referansen til et track i MCCIS database er et heltall (trkrec) som angir et internt sekvensnummer (i MCCIS). Dette sekvensnummeret er den primære referansen som benyttes for å knytte informasjon opp mot et gitt track (heretter kalt trackreferanse).

```
const long IcPlatformDataFM_callsign  = 1;
const long IcPlatformDataFM_category  = 2;
const long IcPlatformDataFM_flag      = 4;
const long IcPlatformDataFM_ltn       = 8;
const long IcPlatformDataFM_hull      = 16;
const long IcPlatformDataFM_name      = 32;
const long IcPlatformDataFM_pif       = 64;
const long IcPlatformDataFM_shipclass = 128;
const long IcPlatformDataFM_threat    = 256;
const long IcPlatformDataFM_type      = 512;
const long IcPlatformDataFM_all       = 1+2+4+8+16+32+64+128+256+512;

struct IcPlatformData {
  string callsign;
  string category;
  string flag;
  string ltn;
  string hull;
  string name;
  string pif;
  string shipclass;
  string threat;
  string type;
  long field_mask;
};
```

Figur 8-4 IDL grensesnitt for TdbmServers datastruktur for plattformdata

```
struct IcAou { // area of uncertainty
  float a1; // major axis
  float a2; // minor axis
  float brg; // bearing
  long typ;// AOU type ellipse=1, bbox=2, lob=3
};

const long IcReportFM_containment    = 1;
const long IcReportFM_cse            = 2;
const long IcReportFM_dtg            = 4;
const long IcReportFM_lat            = 8;
const long IcReportFM_lng            = 16;
const long IcReportFM_spd            = 32;
const long IcReportFM_trkrec         = 64;
const long IcReportFM_aou            = 128;
const long IcReportFM_callsign       = 256;
const long IcReportFM_classification = 512;
const long IcReportFM_sensor         = 1024;
const long IcReportFM_all            = 1+2+4+8+16+32+64+128+256+512+1024;

struct IcReport {
  short containment;//signifikansnivå
  float cse; // Course
  long dtg; // date time group
  double lat; // latitude
  double lng; // longitude
  float spd; // speed
  short trkrec; // key to track record
  IcAou aou; // area of uncertainty
  string callsign;
  char classification;
  string sensor;
  long field_mask;
};
```

Figur 8-5 IDL grensesnitt for TdbmServers datastruktur for rapport og usikkerhet

8.3.1.2 Tilkobling mot TdbmServer

TdbmServer er en persistent CORBA-tjener som registrerer seg i CORBA-navnetjener med ID=”Tdbm” og KIND=”Tdbm”. Det at tjeneren er persistent medfører at den må startes av en ekstern hendelse (ikke et CORBA-kall) før første kall fra en klient. Oppstart av tjeneren kan for

eksempel skje ved oppstart av datamaskinen. Alle klienter benytter samme tjener, og de forholder seg til en tjener som ”alltid” er tilgjengelig. Klientene finner tjeneren ved å spørre CORBA navnetjener etter et objekt med ID=”Tdbm” og KIND=”Tdbm”.

TdbmServer er tilstandsløs, d v s den lagrer ingen data eller tilstander men formidler kall mellom klientene og databasen. I demonstratoren har denne serveren kun en klient, proxy-en mot CoABS Grid.

```
typedef sequence<IcReport> IcReportSeq;
typedef sequence<short> IcTrkrecSeq;

interface Tdbm {
  // new track returns trkrec
  short new_track(in IcPlatformData  plfdata, in IcReportSeq rpts);

  // update track returns trkrec
  short update_track(in IcPlatformData  plfdata, in IcReportSeq rpts);

  void delete_track(in short trkrec );
  IcReportSeq get_history(in short trkrec );
  IcReport get_last_report(in short trkrec);
  IcPlatformData get_platform_data(in short trkrec);
  IcReport get_report(in short trkrec, in long dtg);
  void add_report(in short trkrec, in IcReport rpt);
  void delete_report(in short trkrec, in long dtg);
  long get_current_time ();
  IcTrkrecSeq get_track_ids();
};
```

Figur 8-6 IDL grensesnitt for TdbmServers metoder

8.3.1.3 EventServer

EventServer har som oppgave å lytte på hendelser fra MCCIS trackdatabase, og formidle dem videre til klienter over CORBA ”event”-tjeneste.

EventServer søker opp CORBA ”event”-tjeneste gjennom navnetjeneren og oppretter et ”PushModel” objekt. som registrerer seg hos en CORBA ”event”-kanal. ”PushModel” er definert i CORBA og betyr til at klientene blir kalt asynkront etter hvert som nye hendelser oppstår.

En klient som ønsker å motta hendelser fra EventServer må finne CORBA ”event”-tjeneste og ”event”-kanal gjennom CORBA navnetjeneste eller IOR (”Interoperable Object Reference”), finne kanalens ”PushSupplier” og opprette et EventClient objekt. PushSupplier er i dette tilfellet demonstratorens EventServer. EventClient er et objekt som har implementert metoden ”Push”. Det er denne metoden hos klienten som kalles når EventServer sender en event. IOR er

en tekststreng som unikt definerer et CORBA-objekt.

Hendelser i MCCIS som mottas av EventServer og videresendes (17) er:

- VtTrackUpdateEvent
- VtTrackChangeEvent
- VtTrackMergeEvent
- VtTrackImplicitMergeEvent
- VtTrackDeleteEvent
- VtTrackAddReportEvent
- VtTrackDeleteReportEvent

Disse hendelsene fra MCCIS mappes i EventServer over til hendelser definert i IDL spesifikasjonen IcsEvents.idl (se Figur 8-7).

```
const short IcTrackUpdateEvent       = 123;
const short IcTrackChangeEvent       = 124;
const short IcTrackMergeEvent        = 126;
const short IcTrackDeleteEvent       = 125;
const short IcTrackAddReportEvent    = 127;
const short IcTrackDeleteReportEvent = 128;

module IcsEvents {

   struct EventData {
     short mtype;   //event type
     long time; // time of event  Seconds since 1.1.1970
     short trkrec; // key to tracks

     // Merge does not use trkrec
     short src_trkrec; // merge track from src
     short dest_trkrec;// merge track into dest

     // Add Report and Delete Report
     long dtg; // time of report Seconds since 1.1.1970
   };
};
```

Figur 8-7 IDL grensesnitt til EventServer

Ved hver hendelse som rapporteres over CORBA sendes det en datastruktur som inneholder et minimum av informasjon. Denne datastrukturen er definert i IDL og inneholder:

- Type hendelse
- Tidspunkt når hendelsen blir sendt.
- Referanse til et track.
- Referanse til kilde og destinasjon ved sammenslåing av track. For andre hendelser brukes ikke disse feltene.
- Tidspunktet for hendelsen. For eksempel tidsstempelet til en ny posisjonsrapport.

Dersom klienten er interessert i denne type hendelse så må den hente de oppdaterte data gjennom forespørsler til TdbmServer. I demonstratoren er proxyen mot CoABS Grid eneste klient.

8.3.1.4 Kobling mot JINI og CoABS Grid

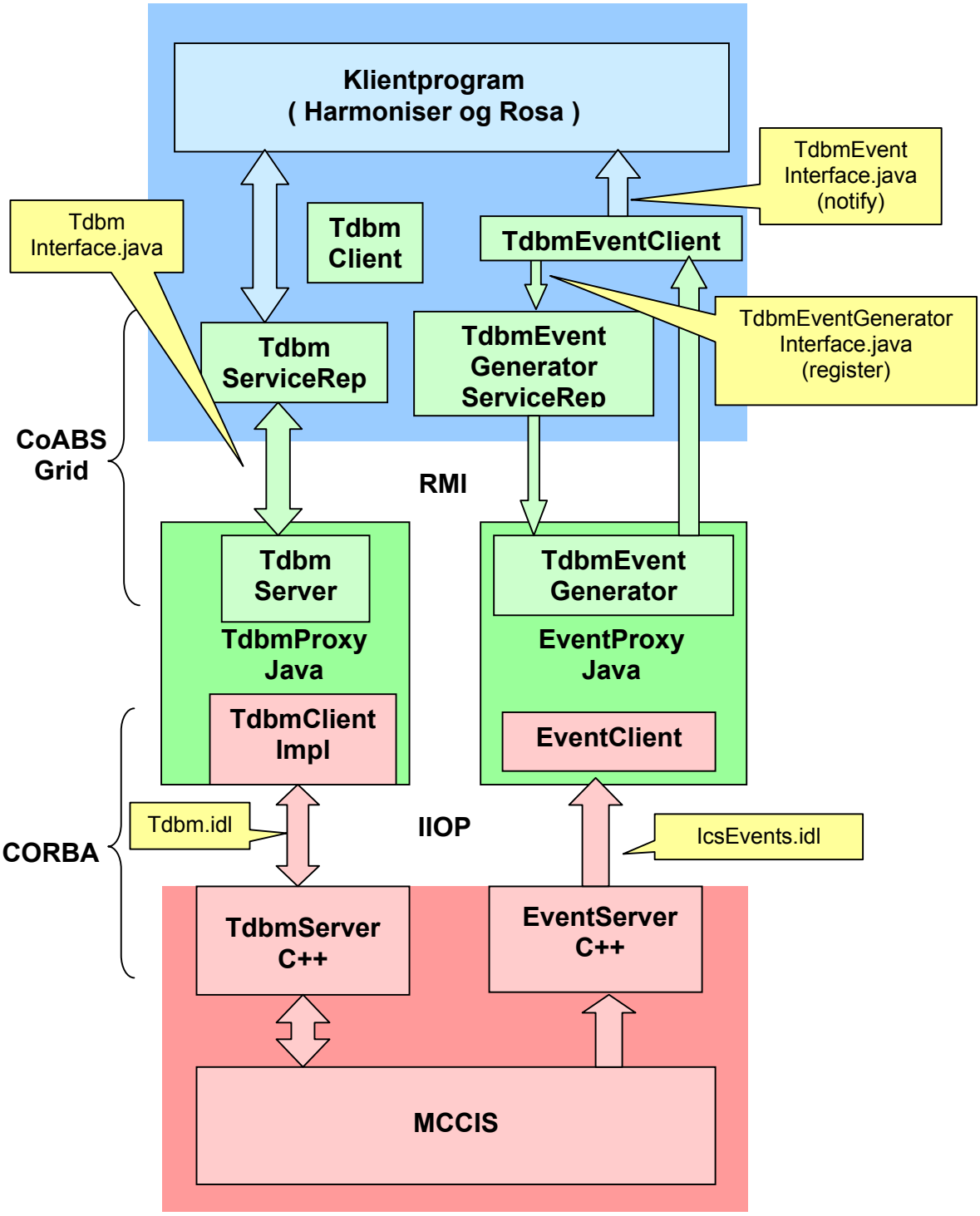
For å gjøre funksjonalitet i TdbmServer og EventServer tilgjengelig som tjenester på CoABS Grid ble det laget proxy-tjenester som oversetter alle forespørsler og hendelser mellom CORBA og Java/Jini. Disse Proxy-tjenestene finner TdbmServer og EventServer gjennom CORBA navnetjener og registrerer seg som tjenester i CoABS Grid. Se kapittel 8.3 for nærmere beskrivelse.

8.3.2 Valg av CORBA ORB

Bruk av CORBA krever tredjeparts programvare. I demonstratoren har vi brukt produktet Visibroker fra Inprise Borland. Dette inkluderer en Object Request Broker (ORB), navnetjener og verktøy for generering av programkode ut fra IDL spesifikasjon (språkbinding). Visibroker støtter både C++ og Java. Tilsvarende programvare (av varierende kvalitet) er også tilgjengelig som ”freeware” eller ”shareware” på internett. For eksempel benytter Maria kartserver OmniORB (http://omniorb.sourceforge.net/) som er ”freeware”, men denne har ingen språkbinding mot Java.

8.4 Tjenester som representerer MCCIS i CoABSGrid

For utviklingen av grensesnittet mellom MCCIS og demonstratoren ble det valgt en fremgangsmåte med proxy- (stedfortreder-) tjenestene TdbmProxy og EventProxy som representerer TdbmServer og Eventserver i CoABS Grid. Dette er vist i Figur 8-8.



Figur 8-8 *TdbmProxy representerer TdbmServeren og i EventProxy representerer EventServer i CoABSGrid*

8.4.1 TdbmProxy

TdbmProxy representerer TdbmServer i CoABS Grid. Grensesnittet mellom klienten og TdbmProxy defineres av et Java-grensesnitt, TdbmInterface, som er vist i Figur 8-9.

```
public interface TdbmInterface extends Remote {
   short[]          get_track_IDs()
   short            new_track   ( PlatformData plfdata, Report[] rpts);
   short            update_track(PlatformData plfdata, Report[] rpts)
   void             delete_track        (short trkrec)
   Report[]         get_history         (short trkrec)
   Report           get_last_report     (short trkrec)
   PlatformData     get_platform_data   (short trkrec)
   Report           get_report          (short trkrec, int      dtg)
   void             add_report          (short trkrec, Report rpt)
   void             delete_report       (short trkrec, int      dtg)
}
```

Figur 8-9 TdbmProxy tjenesten sitt grensesnitt

Kommuikasjonen mellom TdbmProxy og TdbmServer foregår gjennom TdbmServer sitt CORBA-grensesnitt. TdbmClientImpl er en CORBA-klient som er skrevet i Java med bruk av JBuilder 4 Professional, som genererte en stor del av koden automatisk fra Tdbm.idl. De to grensesnittene er like men ikke identiske og TdbmProxy omformaterer dataene før de videresendes.

For å gjøre det enklere å implementere klientprogrammene (Harmoniser-komponenten og testprogrammet Rosa) ble det laget en hjelpe-klasse, TdbmClient, for oppkobling mot tjenesten. TdbmClient sørger for oppkobling mot tjenesten ved at den søker i LUS etter TdbmServer, laster ned tjenestens service-representant, og returnerer en referanse til tjenesten til klientprogrammet. TdbmClient bidrar også til å isolere CoABS Grid-teknologien fra klient-programmet, og dersom man senere skulle ønske å erstatte CoABS Grid med en annen teknologi for aksess av TdbmServeren, som f eks direkte bruk av CORBA, så kan man lage en ny implementasjon av TdbmClient og kun innføre minimale endringer i klientprogrammet.

8.4.2 EventProxy

For event-tjenesten ble det på tilsvarende måte som for TdbmProxy valgt en egen proxy-tjeneste, EventProxy, som representerer EventServeren i CoABS Grid. Dette er vist i Figur 8-8. Grensesnittet mellom klientprogrammet og event-tjenesten defineres av to Java-grensesnitt. TdbmEventGeneratorInterface benyttes for å registrere seg hos tjenesten slik at man får beskjed om endringer i MCCIS sin trackdatabase. Dette grensesnittet er vist i Figur 8-10.

```
public interface TdbmEventGeneratorInterface extends Remote
{
   long register(long ev_id, RemoteEventListener listener);
}
```

Figur 8-10 Event tjenesten sitt grensesnitt

Det andre grensesnittet, TdbmEventInterface, er vist i Figur 8-11 og benyttes av klientprogrammet for å motta hendelsene.

```
public interface TdbmEventInterface
{
   void notify (TdbmEvent event);
}
```

Figur 8-11 Et klientprogram må implementereTdbmEventInterface for å kunne motta hendelser

EventProxy videreformidler hendelser mottatt fra EventServeren. Det er laget en egen klasse, EventClient (se Figur 8-8), som mottar hendelsene gjennom CORBA sin event-tjeneste. TdbmEventInterface og grensesnittet generert fra EventServer sitt IDL-grensesnitt er ikke identiske slik at EventProxy omformaterer hendelsene før de videresendes.

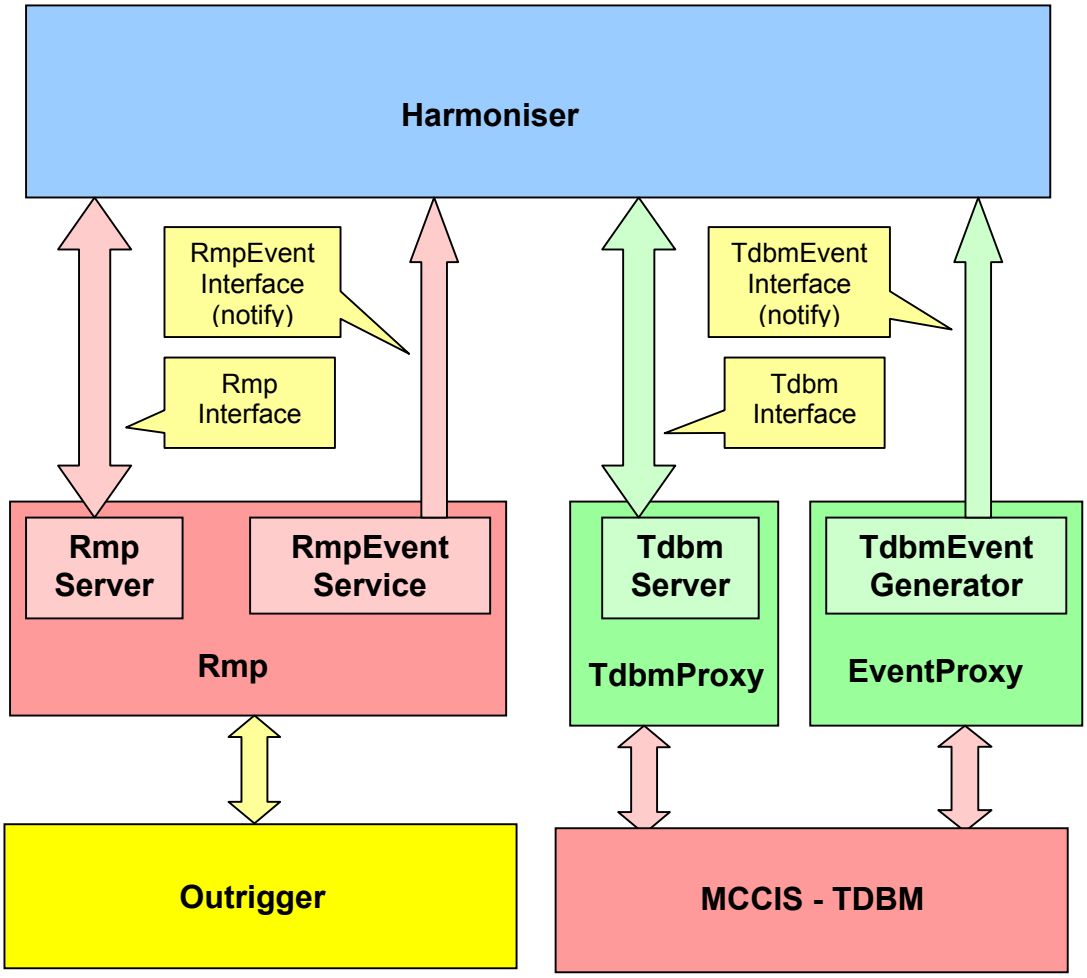
På tilsvarende måte som for TdbmProxy er det laget en egen klasse TdbmEventClient som kan benyttes av klientprogrammene for å koble seg opp mot event-tjenesten. I tillegg til å koble seg opp mot event-tjenesten registrerer denne klassen seg også som mottaker av hendelsene, slik at disse går innom denne klassen på veien til klientprogrammet. Dette er vist i Figur 8-8. Bruk av TdbmEventClient bidrar til å isolere klientprogrammet fra detaljene i CoABS Grid.

8.5 Harmonisering av databasene

Det er et mål at innholdet i RMP-komponenten og MCCIS sin track-database (Tdbm) blir så like som mulig. De to databasene inneholder ulike data og dataene representeres forskjellig slik at databasene følgelig ikke kan bli like. Vi har derfor valgt å kalle denne prosessen for harmonisering.

8.5.1 Harmoniser

Harmoniseringen av databasene utføres av Harmoniser. Denne komponenten kobler seg opp både mot RMP-komponenten og Tdbm-tjenestene (TdbmProxy og EventProxy), slik som vist i Figur 8-12. Når Harmoniser mottar hendelser som indikerer at en av databasene (enten RMP-komponenten eller TdbmProxy) er endret, henter den de nye dataene fra denne og sender dem til den andre databasen.



Figur 8-12 Komponenten Harmoniser sørger for harmonisering av Rmp-modellen og trackdatabasen i MCCIS

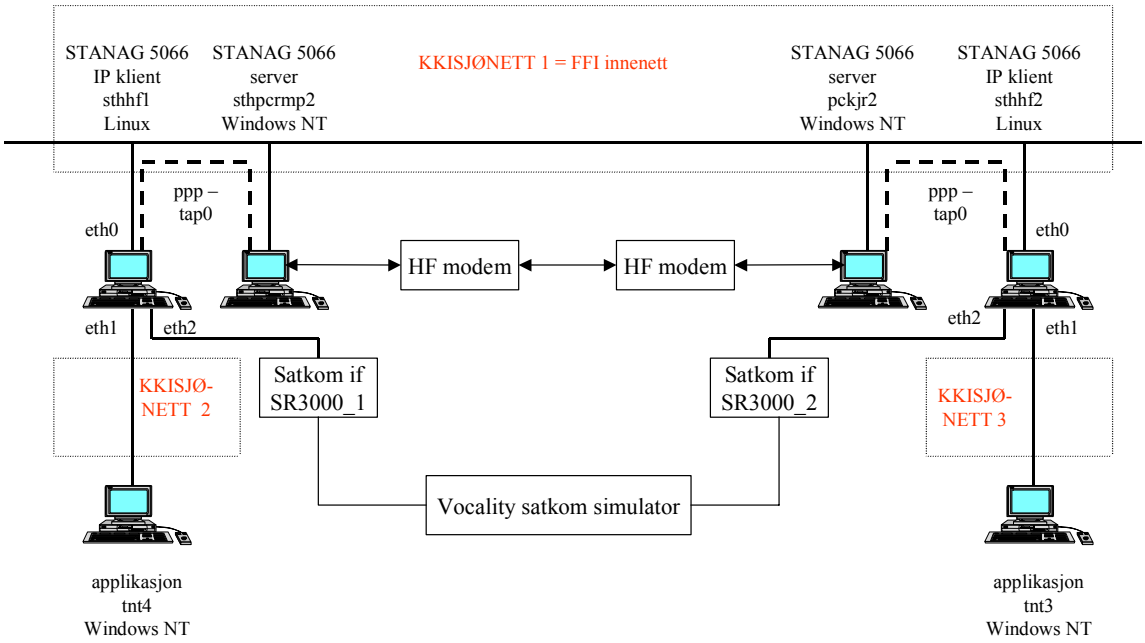
For at Harmoniser skal kunne kjenne igjen hendelser fra RMP-komponenten som den selv er opphav til, sendes det tidsinformasjon (eventTime) sammen med nye data til RmpServer. Tidsinformasjonen blir sendt med hendelsene slik at hendelser generert på grunnlag av Harmoniser sine egne oppdateringer kan filtreres bort.

9 KOMMUNIKASJON

Hovedhensikten med kommunikasjonsoppsettet i demonstratoren var å vise at standard internettprotokoll, IP, kunne benyttes i et taktisk miljø over tilgjengelige båndbredder.

For å kunne simulere et realistisk taktisk miljø, ble det implementert en lavhastighets HF-radioforbindelse og en middels til høyhastighets simulert satellittkommunikasjonsforbindelse. Som referanse kan alle enheter kommunisere over høyhastighets LAN (100Mbit/s) direkte. Denne konfigurasjonen ble oppnådd ved å sette to stk pc med Linux operativsystem og 3 fysiske nettverkskort som rutere mellom de to maskinene tnt3 og tnt4 hvor applikasjonene kjøres. Det

totale kommunikasjonsoppsettet er vist i Figur 9-1.



Figur 9-1 Oversikt over oppsett av ulike kommunikasjonsbærere

Som vist i Figur 9-1 kan trafikk mellom maskinene KKISJØNETT 2 og KKISJØNETT 3 rutes tre forskjellige veier. Disse er:

1. Over KKISJØNETT 1 som er 100 Mbit/s LAN
2. Over en HF forbindelse 75 bit/s til 2400 bit/s
3. Over satkom simulator 9600 bit/s til 2Mbit/s

Over alle de tre mulige rutene benyttes standard IP-protokoll, og bortsett fra hastigheten ser ikke applikasjonene på henholdsvis tnt3 og tnt4 noen forskjell på de forskjellige alternativene.

9.1 LAN

I dette oppsettet inngår kun standard LAN-teknologi og Linux-maskinene ruter all trafikk til/fra KKISJØNETT 1 til/fra henholdsvis KKISJØNETT 2 eller KKISJØNETT 3 avhengig av mottakers IP-adresse. Linux-maskinene er i denne konfigurasjon satt opp til å rute både vanlig IP-trafikk og IP-multicast.

9.2 HF

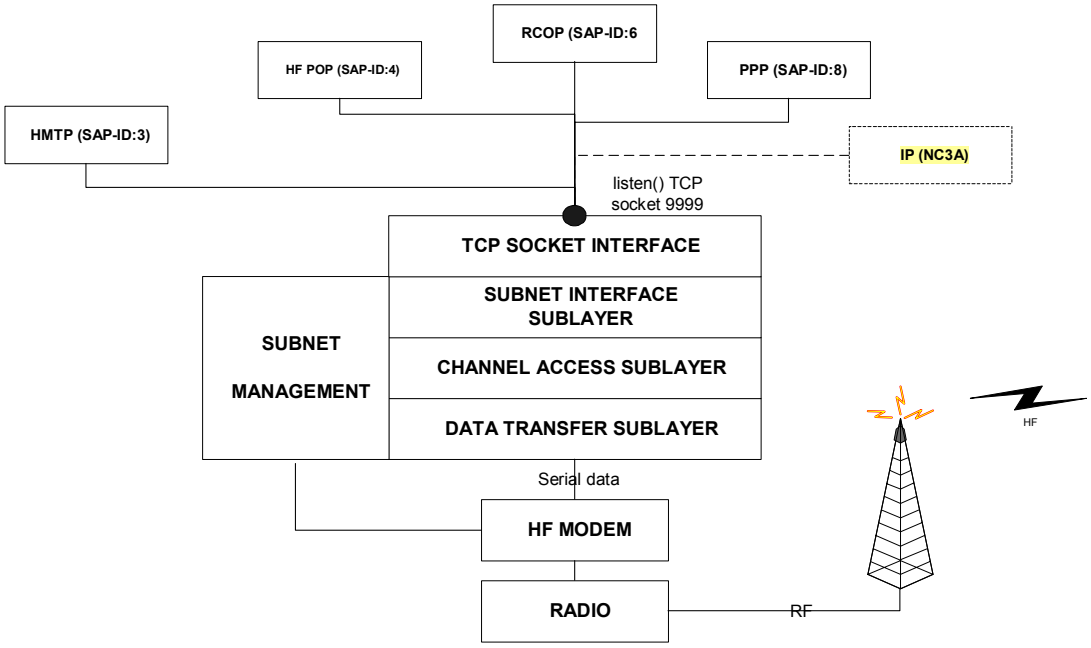
HF-linken er den mest kompliserte forbindelsen i demonstratoren. Denne er implementert ved hjelp av to HF-modemer fra Harris, RF-5710A, utlånt fra Forsvarets logistikkorganisasjon Sjø, FLO/Sjø, i Bergen. Disse modemene er koblet rygg mot rygg, og kanalen mellom dem er av den grunn feilfri. Det er ikke gjort noe forsøk på å introdusere støy på denne forbindelsen. Modemene er av samme type som FLO/Sjø har anskaffet til de norske marinefartøyer.

Modemene har innbygget en rekke forskjellige bølgeformer. I demonstratoren har vi kun benyttet STANAG 4285C, da dette er den samme modulasjon som benyttes på kringkasterne og på fartøy – land sambandet i Sjøforsvaret i dag. Protokollen som kjøres på linklaget over modemene var NATO STANAG 5066: Profile for High Frequency Radio Data Communications (1), levert av Marconi Mobil i UK. Dette er en kommersielt tilgjengelig programvarepakke som kjører under Windows NT. STANAG 5066 implementerer link-laget i OSI-modellen, og har derfor ikke implementert IP direkte. Den kan benyttes både i automatisk retransmisjonsmodus og i ren kringkastingsmodus. NC3A i den Haag har implementert en programvarepakke som kjører under Linux operativsystem og som er en generell IP-klient beregnet på å overføre IP-pakker over STANAG 5066. Denne er lastet inn på Linux-maskinene som igjen kommuniserer med STANAG 5066 serveren på Windows NT maskinene over en virtuell punkt-til-punkt forbindelse, ppp-tap0, til en TCP-socket interface. I tillegg endres rutetabellene på begge Linux-maskinene slik at trafikk mellom KKISJØNETT2 og KKISJØNETT 3 rutes over tap0 i stedet for over KKISJØNETT 1.

Figur 9-1 viser den fysiske oppkoblingen av de seks maskinene som er involvert i kjøringen av IP over STANAG 5066. Merk at i denne figuren er ingen av de andre maskinene som benyttes i demonstratoren inntegnet. Alle disse befinner seg på KKISJØNETT 1 som er FFI's innenett.

STANAG 5066 server kjøres på maskinene sthpcrmp2 og pckjr2. På hver av disse maskinene er det montert et synkront RS 232 kort for sending og mottak av data. Styring av modemene, som setting av bitrate, type modulasjon etc, er også lagt inn i Marconi's programvare og benytter en vanlig serieport på hver av maskinene. I STANAG 5066 er det definert at hastigheten på linken mellom modemene skal kunne settes opp eller ned avhengig av kanalkvaliteten. Da det i demonstratoren, som før nevnt, er en feilfri link mellom modemene, er denne tjenesten ikke i bruk i demonstratoren.

Figur 9-2 viser hvordan Marconi har implementert STANAG 5066 via et TCP socket-grensesnitt. Dette socket-grensesnittet går på de to STANAG 5066 serverne som er hhv sthpcrmp2 og pckjr2. IP-klientene, som kjøres på hhv sthhf1 og sthhf2, knytter seg til HF-nettverket via dette socket-grensesnittet som Marconi har implementert som port 9999 på sin STANAG 5066 server.



Figur 9-2 Marconis STANAG 5066 implementasjon

9.3 Satellitt kommunikasjon

Satellittkommunikasjon er i demonstratoren implementert ved hjelp av sivile komponenter: to SR3000 Internet ThinRouter fra Moxa Technologies Co. Ltd og en Simulator 2 fra Vocality International Ltd. Disse er koblet opp som vist i Figur 9-1. På Linux-maskinene settes ruting mellom KKISJØNETT 2 og KKISJØNETT 3 til å gå via nettverksforbindelse eth2, slik at i tilfellet satellittkommunikasjon er det ikke behov for noen ekstra programvare.

SR3000 Internet ThinRouter er en IP-ruter med en 10/100 Mbit/s LAN-port og en synkron V.35 WAN-port. Dette gjør at det mellom to slike rutere opprettes en synkron kanal med hastighet enten bestemt av SR3000 eller leverandøren av forbindelsen mellom disse. Da SR3000 har 56 Kbit/s som laveste bitrate og kun et fåtall muligheter opp til 2 Mbit/s har vi i vårt oppsett valgt å la nettet mellom de to SR3000 bestemme bitraten. De to SR3000 kjører en synkron ppp-forbindelse seg i mellom.

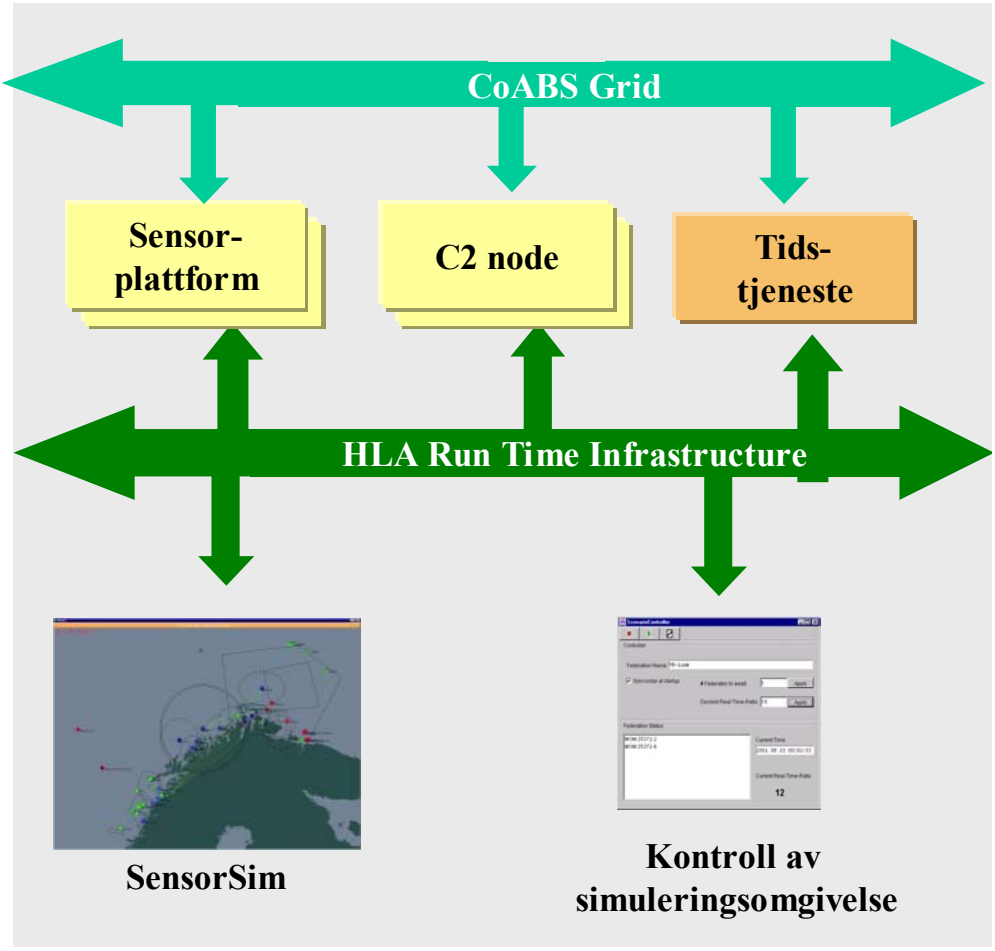
Nettet her er definert ved Simulator 2 som har 2 V.35-porter som data overføres transparent mellom. Dermed ser Simulator 2 ut som et helt normalt WAN. Simulator 2 har innebygde funksjoner som gjør at bitraten kan varieres trinnløst fra 9600 bit/s til 2 Mbit/s, noe som gir mulighet for å teste overføring både på høye og på relativt lave bitrater. Simulator 2 har også innbygget lagerkapasitet til å muliggjøre forsinkelser i nettet fra 0 til 2000 ms slik at kanalen mellom de to SR3000 har egenskaper som er tilsvarende de som oppleves på en virkelig satellittkanal.

På http://www.moxa.com finnes en nærmere beskrivelse av SR3000 Internet ThinRouter og på http://www.vocality.com finnes en nærmere beskrivelse av Simulator 2.

10 SIMULERINGSOMGIVELSEN

Simuleringsomgivelsens oppgave er å simulere et på forhånd oppsatt hendelsesforløp med tilhørende plattformer, sensorer, bildeproduksjonsnoder og produsere track-data som sendes inn i demonstratoren. Simuleringsomgivelsen består av følgende komponenter:

- Et simuleringsprogram, SensorSim, som er utviklet av prosjektet.
- Et program for å kontrollere simuleringsomgivelsen – ”Scenario Controller”
- En tidstjeneste som koordinerer og regulerer felles tid i demonstratoren – ”Timer”
- Komponenter som opptrer som sensortjenester i CoABS grid – ”Reporter”.



Figur 10-1 Oversikt over komponentene i simuleringsomgivelsen

SensorSim kan simulere hele bildeproduksjonsprosessen, d v s deteksjon, tracking, enkel datafusjon og informasjonsutveksling, og er satt opp slik at det simulerer sensorer og bildeoppbygging for de enhetene som er definert til å ligge utenfor demonstratoren. Figur 10-1 viser en oversikt over hovedelementene i simuleringsomgivelsen.

10.1 Kort beskrivelse av High Level Architecture

Simuleringsomgivelsen er bygget over High Level Architecture (HLA). HLA er en standard introdusert av US Defence Modeling and Simulation Office (DMSO) for å øke graden av gjenbruk av simuleringsmodeller. HLA finnes i to versjoner: HLA v1.3 utviklet av US DoD (22)-(24) som et utgangspunktet for IEEE standardisering, samt den resulterende IEEE standarden, IEEE 1516 (26)-(28). Disse to versjonene er svært like, men er ikke interoperable på "lufta" eller på API-nivå. Det har hittil vært svært få kommersielle produkter som støtter IEEE 1516 slik at demonstratoren benytter HLA v1.3. HLA er tenkt å dekke integrasjon av rene simuleringsmodeller, bemannede simulatorer og operativt utstyr. Den skal således forene tidligere standarder som ALSP (Aggregate Level Simulation Protocol) og DIS (Distributed Interactive Simulation).

Et sett med simuleringsmodeller som benytter HLA for informasjonsutveksling seg i mellom kalles en *føderasjon*, og om hver enkelt simuleringsmodell benyttes betegnelsen *federat*. HLA standarden består av tre deler:

- Et sett med regler for hvordan federatene skal samvirke
- En objektmodell som beskriver hva som skal utveksles av data – Federation Object Modell (FOM).
- En grensesnittspesifikasjon definert ved et sett med tjenester. HLA standarden definerer tjenester på seks ulike områder: Federation Management, Declaration Management, Object Management, Ownership Management, Time Management samt Data Distribution Management.

Valget av HLA som infrastruktur i simuleringsomgivelsen er begrunnet med behovet for å kunne gjenbruke eksisterende simuleringsprogrammer samt integrere disse med nye simuleringsprogrammer med supplerende funksjonalitet. I tillegg finnes det en del kommersielt tilgjengelige verktøy som støtter opp om denne arkitekturen, f eks visualiseringsprogramvare, dataloggere og programvare for monitorering og kontroll av føderasjoner.

10.2 Rapporteringstjeneste

Rapporteringstjenesten (Reporter) sin oppgave er å ta data fra utvalgte sensorer og bilde-produksjonsnoder og gjøre dem tilgjengelig for andre komponenter i CoABS Grid. Dette betyr at en kan benytte et antall slike tjenester, som hver konfigureres til å representere en eller flere sensorer og/eller fusjonsnoder, for å representere et enkelt "Sensor Grid". Rapporterings-tjenesten mottar data fra SensorSim over HLA RTI (Run-Time Infrastructure), og gjør disse tilgjengelig for komponenter i CoABS Grid via en enkel klient-tjener modell. Kommunikasjon mellom klient og tjener skjer ved at klienter, som har registrert seg hos tjenesten, mottar en hendelse når nye data er ankommet, hvorpå klienten så kan hente dataene ved å kalle metoder på rapporteringstjenestens grensesnitt (se nedenfor). Tre typer hendelser er definert:

1. "Track Update" – sendes ved regulære oppdateringer av et track (posisjon og hastighet) og ved nytt track.

2. ”Classification Update” – sendes når et track har endret klassifikasjon, og ved første gangs rapportering av et track.
3. ”Lost Track”- sendes når rapporterende enhet ikke lenger har noen sensorer eller andre enheter som rapporterer tracket.

Tjenesten lagrer alle data som er ankommet i en kjøring (siden oppstart) slik at en klient kan hente gamle data ved å oppgi track-id og tidspunkt for rapporten.

```
public interface ReporterInterface extends Remote {

UnitPoint getUnitPointReport  (int reportedTrackID, int reportTime) throws RemoteException;
SurfaceUnit getSurfaceUnitReport(int reportedTrackID, int reportTime) throws
RemoteException;
long register(long ev_id, RemoteEventListener listener) throws RemoteException;
void deregister(RemoteEventListener listener) throws RemoteException;
}
```

Figur 10-2 Grensesnittet til rapporteringstjenesten

Dataoverføringen fra SensorSim til rapporteringsjenesten skjer med utsendelse av en egendefinert HLA-interaksjon ved navn ”TrackMessage”. Hver enkelt federat har en Simulation Object Model (SOM) som sier hvilke data den skal skal sende og motta. For SensorSim sin del er denne vist i Tabell 10.1. Interaksjonen ”TrackMessage” inneholder tre parametere: meldingshode, kontaktdata og posisjonsdata. Dataene som sendes med i de ulike parametrene er pakket inn i et OTG (Over-The-Horizon Targeting Gold) lignende format (1) tilpasset demonstratorens datamodell. Dataene er pakket inn i en formattert tekststreng som starter med et nøkkelord (POS, CTC eller REPORT), se Figur 10.3. Bruk av et tekstformat ble valgt fordi vi på det tidspunktet dette ble implementert ikke hadde løst problemet med ulik ”endian”-koding av tallrepresentasjon på UNIX og PC-plattformen, samt at et tekstformat er enkelt ved feilsøking. I tillegg til den egendefinerte interaksjonen baserer HLA-føderasjonens objektmodell seg på ”Real-time Platform Reference - Federation Object Model” (RPR-FOM) (25). RPR-FOM ble valgt bl a for å muliggjøre bruk av en ”Stealth Viewer”.

Interaksjon	Parameter	Send	Motta	Datatype
TrackMessage	**MessageHeader**		√	Char array
	SurfaceUnit		√	Char array
	UnitPoint		√	Char array

Tabell 10.1 Simulation Object Model til Reporter

Reglene for overføring fra SensorSim til rapporteringstjenesten settes opp på samme måte som rapportering mellom noder internt i SensorSim, med det unntak at man sender til et spesielt nettverk ved navn RTI. Setter man opp periodisk utsendelse sendes rapporter om alle track ut satsvis, f eks hvert tredje minutt. Dette medfører at lasten i demonstratoren blir ujevn, noe som er en ulempe ved overføring over HF. Dette er ikke en ideell løsning, men ble valgt fordi en slapp å gjøre endringer i SensorSim.

Format for “MessageHeader”: REPORT/MESSAGETYPE/TRACKID/TIME/SOURCENAME

Format for “SurfaceUnit”:CTC/TIME_CTC/STDID/TRKNUM/NATIONALITY/
CATEGORY/TYPE/CLASS/NAME/SENSORTYPE/SOURCE

Format for “UnitPoint”: POS/TIME/LATITUDE/LONGITUDE/HEIGHT/
COURSE/SPEED/MAJOR_AXIS/MINOR_AXIS/ANGLE/CONFIDENCE

Figur 10-3 Format for oversendelse av trackdata

Tabell 10.2 viser hvilke HLA-tjenester som benyttes av rapporteringstjenesten († betyr at tjenesten initieres fra RTI):

Service Type	Service
Federation Management	**Create Federation Execution**
	Destroy Federation Execution
	Join Federation Execution
	Resign Federation Execution
	Announce Synchronization Point †
	Federation Synchronized †
	Synchronization Point Achieved
Declaration Management	**Subscribe Interaction Class**
Object Management	**Receive Interaction †**
Time Management	**Enable Time Regulation**
	Time Regulation Enabled †
	Enable Time Constrained
	Time Constrained Enabled †
	Time Advance Request

Tabell 10.2 HLA-tjenester benyttet av Reporter

10.3 Tidstjeneste

Tidstjenesten (Timer) sin oppgave er å sørge for en koordinert tid i demonstratoren, d v s den benyttes av alle komponenter i CoABS Grid som trenger å vite hva klokka er. Tiden i demonstratoren kontrolleres av simuleringsomgivelsen og har visse egenskaper:

- Den har en variabel rate i forhold til vegg-klokka, d v s den kan gå raskere, saktere eller stoppe helt opp (midlertidig stoppes av brukeren).
- Den er i fortiden, dette er fordi MCCIS ikke aksepterer meldinger tidsstemplet i framtida.
- Den er monotont økende.

Tidstjenesten er et enkelt program en kan spørre hva klokka er. En kan også be om å få tilsendt en hendelse med et periodisk tidsintervall, f eks kan en komponent registrere seg med informasjon om at den ønsker å få en hendelse hvert 10. s i demonstratortid. Denne hendelsen inneholder i tillegg til verdien på demonstratortiden, assosiert ”veggklokketid” samt et estimat

på hvor raskt demonstratortiden går i forhold til vegg-klokka. Tidstjenesten er delt inn i en tids-server og en event-tjeneste. Grensesnittene til disse er vist i Figur 10-4.

```
public interface TimeInterface extends Remote {

    int getTime() throws RemoteException;
}

public interface TimeEventServiceInterface extends Remote {

  // if you register with wakeup_interval = 0 you will never be bothered
    long register(long ev_id, int wakeup_interval, RemoteEventListener listener) throws
RemoteException;
    void deregister(RemoteEventListener listener) throws RemoteException;
}
```

Figur 10-4 Grensesnittene til Tidstjenesten

Styringen av tiden i demonstratoren baserer seg på bruk av HLA "Time Management". Det vil si at man utpeker et sett med simuleringsprogrammer som er med på å bestemme tidsutviklingen i føderasjonen. Dette skjer ved at simuleringsprogrammene fortløpende sender informasjon til HLA RTI om det laveste mulige tidsstempel på data de vil sende ut på RTI-en i framtiden. I simuleringsomgivelsen er det tre simuleringsprogrammer som regulerer tiden: Sensorsim, tidstjenesten og programmet for kontroll av simuleringsomgivelsen.

Tidstjenesten publiserer eller abonnerer ikke på noen data fra andre simuleringer (klasser eller interaksjoner). HLA-tjenestene som benyttes av tidstjenesten er listet opp i Tabell 10.3.

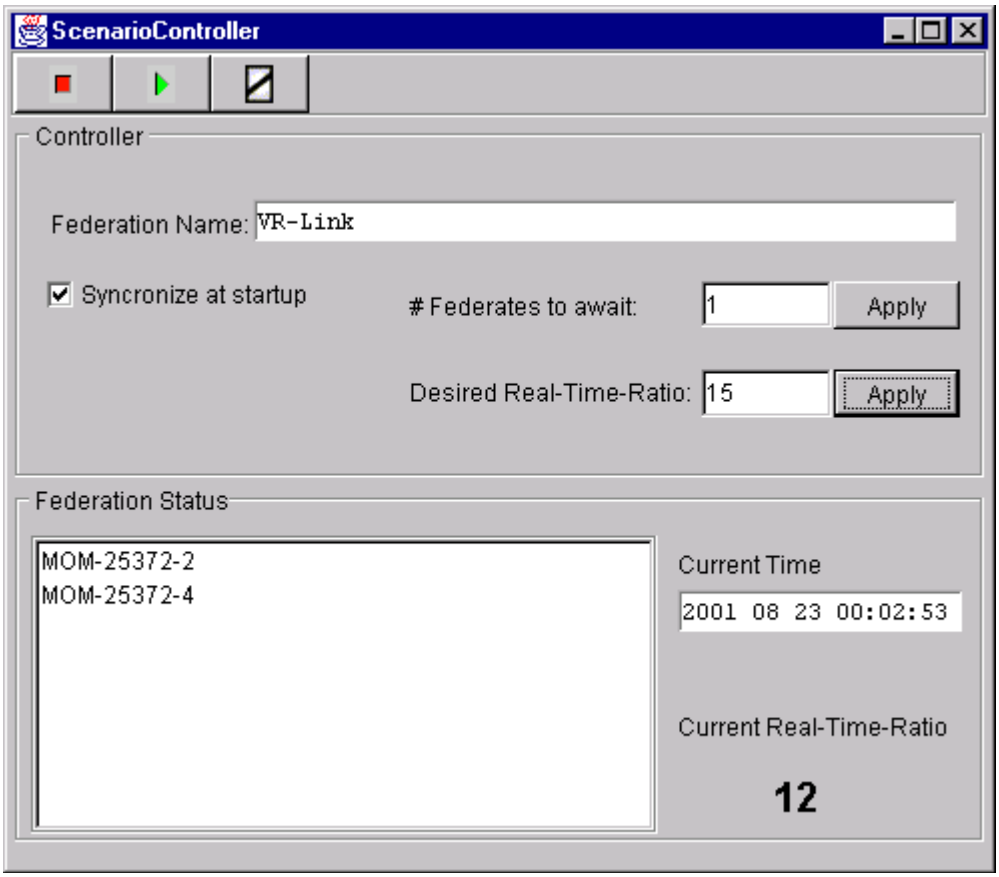
Service Type	Service
Federation Management	**Create Federation Execution**
	Destroy Federation Execution
	Join Federation Execution
	Resign Federation Execution
	Announce Synchronization Point †
	Federation Synchronized †
	Synchronization Point Achieved
Time Management	**Enable Time Regulation**
	Time Regulation Enabled †
	Enable Time Constrained
	Time Constrained Enabled †
	Time Advance Request

Tabell 10.3 HLA-tjenester som benyttes av Tidstjenesten.

10.4 Program for kontroll av simuleringsomgivelse

Scenario Controller benyttes for å kunne monitorere og kontrollere tidsutviklingen i

demonstratoren. Den har i tillegg funksjonalitet for synkronisert oppstart av simuleringsomgivelsen. Brukergrensesnittet til Scenario Controller er vist i Figur 10-5.



Figur 10-5 Program for kontroll av simuleringsomgivelse

Ved oppkjøring av simuleringsomgivelsen kan brukeren velge om en ønsker synkronisert oppstart av føderasjonen. Synkronisert oppstart betyr at alle federatene går gjennom visse faser:

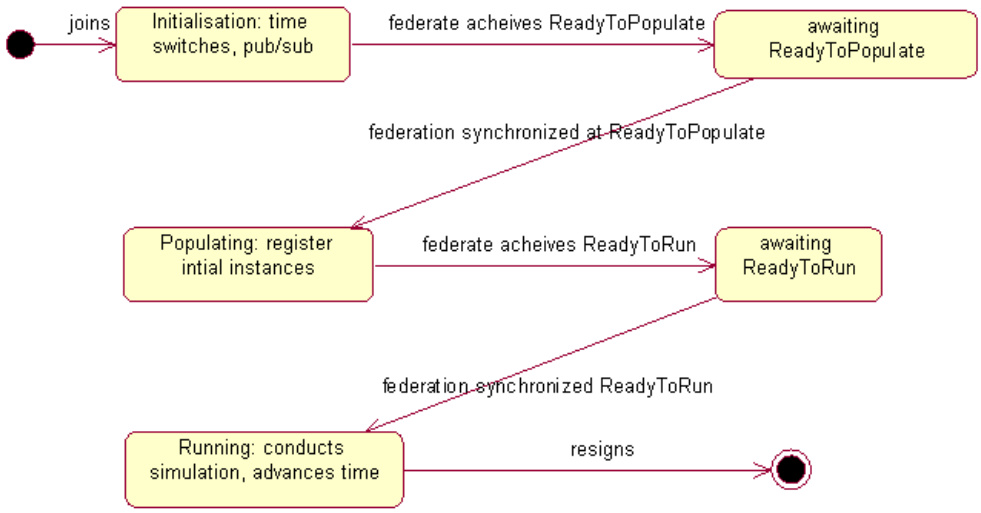
1. En fase hvor man kobler seg opp mot føderasjonen (”join”) samt initialiserer ”Time Management” tjenesten og sier hvilke data en ønsker å publisere og abonnere på.
2. En fase hvor man registrerer objekter hos RTI-en slik at de kan oppdages av andre federater.
3. En kjørefase.

Klasse	Attributt	Send	Motta
Manager.Federate	**FederateHandle**		√

Tabell 10.4 Simulation Object Model for Scenario Controller

Denne framgangsmåten for å sikre synkronisert oppstart av føderasjonen er en modifikasjon av en tilstandsmodell anbefalt i (20). Synkronisering fungerer slik at alle federatene får melding fra RTI om at føderasjonen er synkronisert når alle deltagende federater har fortalt RTI at de er klare for å gå inn i neste fase. Figur 10-6 inneholder et tilstandsdiagram for federater som deltar

i synkroniseringsprosedyren. Det er definert to synkroniseringpunkter (”ReadyToPopulate” og ”ReadyToRun”). Scenario Controller venter med å sende ”ReadyToPopulate” til RTI før alle federater er kommet med i føderasjonen. For å kunne gjøre dette må Scenario Controller vite hvor mange federater/simuleringer den skal vente på, noe som settes i brukergrensesnittet, samt vite hvor mange (eventuelt hvem) som deltar i føderasjonen. Scenario Controller abonnerer på deler av HLA ”Management Object Model” (MOM) for å få opplysning om når nye simuleringer har knyttet seg til eller forlatt federasjonen, se Tabell 10.5.



Figur 10-6 Tilstandsmodellen benyttet i simuleringsomgivelsen

Etter at simuleringen er startet har brukeren muligheten for å kontrollere hvor raskt en ønsker at tiden i demonstratoren skal gå samt mulighet for å stoppe tiden midlertidig (pause). Et estimat av hvor raskt simulatortiden går i forhold til vegg-klokka blir også presentert for brukeren. Dette er nødvendig fordi selv om brukeren ønsker at simulatortiden skal gå f eks 20 ganger raskere enn vegg-klokka vil hastighetsutviklingen i praksis begrenses av SensorSim i scenarier over en viss størrelse.

Scenario Controller er et generelt program som kan gjenbrukes i andre føderasjoner. Begrensningene er at de andre federatene må benytte synkroniseringssekvensen illustrert Figur 10-6, hvis en ønsker å benytte synkronisert oppstart. En annen begrensning, som det er enkelt å endre på, er at tidsintervallet er satt fast til ett sekund.

Scenario Controller implementerer følgende HLA tjenester (Tabell 10.5):

Service Type	Service
Federation Management	**Create Federation Execution**
	Destroy Federation Execution
	Join Federation Execution
	Resign Federation Execution
	Register Federation Synchronization Point
	Announce Synchronization Point †
	Federation Synchronized †
	Synchronization Point Achieved
Declaration Management	**Subscribe Object Class Attributes**
Object Management	**Discover Object Instance †**
	Remove Object Instance †
Time Management	**Enable Time Regulation**
	Time Regulation Enabled †
	Enable Time Constrained
	Time Constrained Enabled †
	Time Advance Request

Tabell 10.5 HLA-tjenester benyttet av Scenario Controller

10.5 SensorSim

SensorSim er motoren i simuleringsomgivelsen og inneholder simulering av fartøysbevegelser, sensormodeller, kommunikasjon mellom plattformer og bildeoppbyggingsprosesser. Det er SensorSim som generer alle track-data som er input til demonstratoren. For en detaljert beskrivelse av mulighetene i SensorSim se (21), dette kapitlet inneholder kun modifikasjoner gjort i f m utvikling av demonstratoren.

SensorSim har et forholdsvis omfattende HLA-grensesnitt, se Tabell 10.6. I dette kapitlet er alle tjenestene dokumentert, men for beskrivelse av SOM er kun den delen som er relevant for demonstratoren dokumentert.

Service Type	Service
Federation Management	**Create Federation Execution**
	Destroy Federation Execution
	Join Federation Execution
	Resign Federation Execution
	Announce Synchronization Point †
	Federation Synchronized †
	Synchronization Point Achieved
Time Management	**Enable Time Regulation**
	Time Regulation Enabled †
	Enable Time Constrained
	Time Constrained Enabled †
	Time Advance Request
Declaration Management	**Publish Object Class**

Service Type	Service
	Publish Interaction Class
	Subscribe Interaction Class
	Subscribe Object Class Attributes
	Start Registration For Object Class †
Object Management	**Register object Instance**
	Discover Object Instance †
	Update Attribute Values
	Reflect Attribute Values
	Send Interaction
	Receive Interaction †
	Delete Object Instance
	Remove Object Instance †
	Provide Attribute Value Update †
	Turn Updates On For Object Instance †
	Turn Updates Off For Object Instance †
Ownership Management	**Attribute Ownership Acquisition Notification †**
	Request Attribute Ownership Release †
	Attribute Ownership Acquisition
	Attribute Ownership Release Response

Tabell 10.6 HLA-tjenester implementert av SensorSim

Tabell 10.7 viser de deler av SOM-en som benyttes i demonstratoren. Den komplette SOM-en som er implementert inneholder i tillegg utsendelse av plattformdata (fartøystype etc) samt kinematiske data i henhold til RPR-FOM.

Interaksjon	Parameter	Send	Motta	Datatype
TrackMessage	**MessageHeader**	√		Char array
	SurfaceUnit	√		Char array
	UnitPoint	√		Char array

Tabell 10.7 Simulation Object Model til SensorSim

11 LABORATORIUM

Demonstratoren er bygget opp i laboratoriet som vist i Figur 11-1. Sentralt i demonstratoren er to operatørposisjoner for bildeoppbygging hvor den ene kan tenkes lokalisert til et hovedkvarter og den andre i en sjøgående stab (CTG). For beslutningstakerne er det installert et interaktivt visualiseringsbord med plass til fire operatører. I tillegg er det fire storskjermer i synsfeltet foran operatørene, hvorav en av skjermene er interaktiv.

Datamaskinene som benyttes i demonstratoren er tilkoblet FFIs konfidensielle innenett og alle maskinene unntatt FTS 2 hvor Maria-kartserveren og SATweb-serveren går, er lokalisert i laboratoriet. Se Figur 9-1.

11.1 Fysiske enheter

Listen nedenfor inneholder hvilke type datamaskiner og presentasjonsutstyr som inngår i demonstratoren og Figur 11-1 viser hvor enhetene fysisk er plassert i laboratoriet.

Datamaskiner:

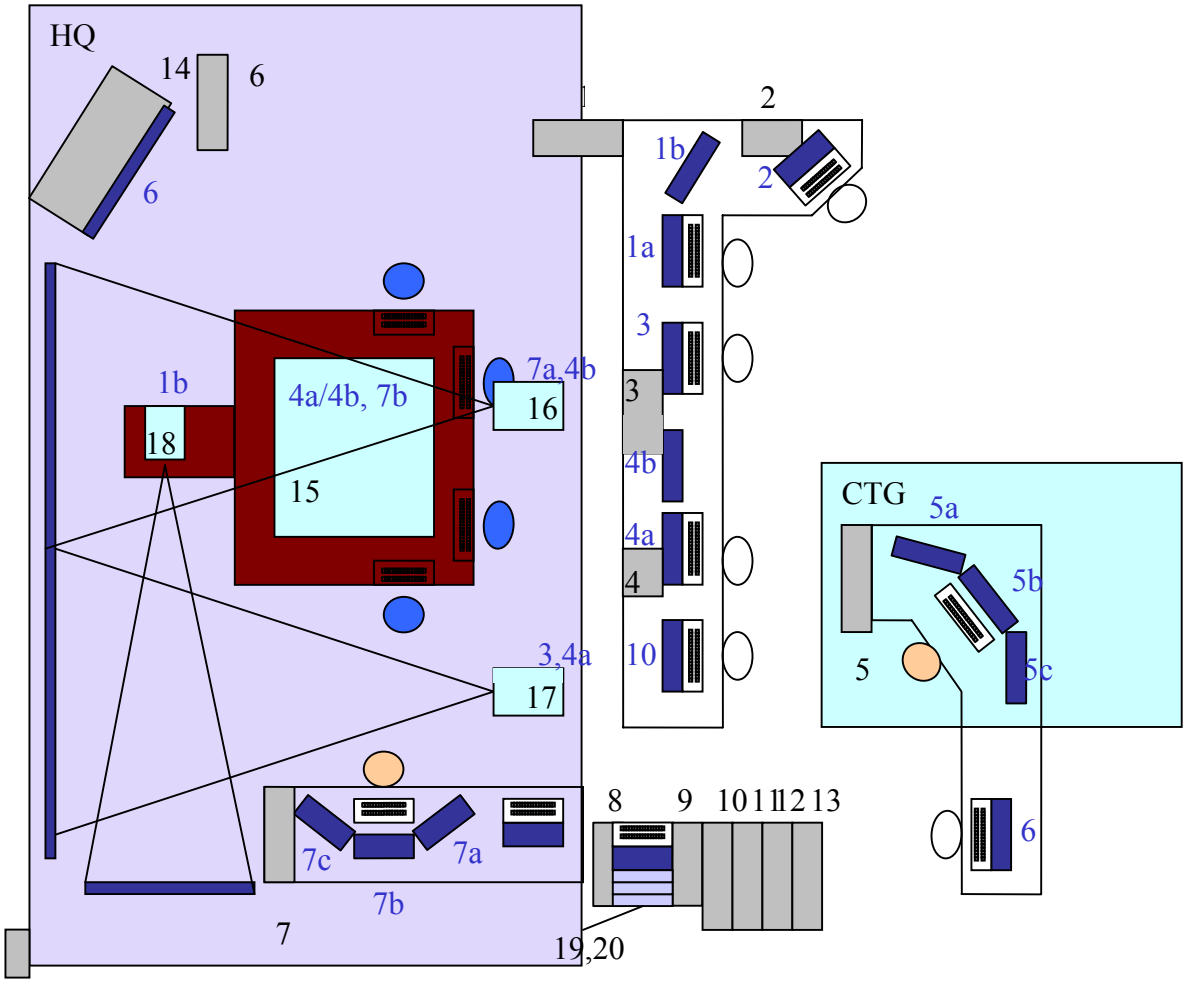
Nr	Maskin navn	Type	Operativsystem
1	STHRMP2	SUN SPARC10 med 2 videoutg. a og b	Solaris
2	STHRMP3	SUNBLADE 1000 2XCPU	Solaris
3	STHMCCIS	HP C3000	HP-UX
4	STHVIS	SGI OCTANE 2 med 2 videoutg. a og b	IRIX
5	STHOP2	SGI-ZX10 med 3 videoutg. a,b og c	NT
6	STHPCSJOTAS	CINET PPI-800	NT
7	STHOP1	SGI-ZX10 med 3 videoutganger a,b og c	NT
8	STHHF1	DELL	LINUX
9	STHHF2	CINET 556	LINUX
10	STHPCRMP2	HP VECTRA VL400	NT
11	PCKJR2	CINET Dimension XPS Pro 200	NT
12	TNT3	CINET PPI-506	NT
13	TNT4	CINET PPI-556	NT
**	FTS 2	CINET PPI-700	NT

Presentasjonsutstyr :

14 SMARTBOARD 1602 med JVC DLA 2 projektor, oppløsning 1280x1024
15 Visualiseringsbord med 2 stk. BARCO SIM6 projektorer, oppløsning 1280x1024
16 Takmontert JVC projektor DLA-G3010ZG, oppløsning 1280x1024
17 Takmontert JVC projektor DLA-G3010ZG, oppløsning 1280x1024
18 ASK projektor A10, oppløsning 1024x768

Kommunikasjonsutstyr:

19 SATCOM simulator fra Vocality
20 2 stk. HF modem Harris 5710
21 2 stk SR3000 Internet ThinRouter fra Moxa Technologies Co. Ltd



Operatørkonsoll

Terminal

Ekstra skjerm

Operatør for bildeoppbygging

Beslutningstaker

Lab-personell for kjøring av nødvendig SW (Sensorsim, MCCIS, HF ....)

Figur 11-1 Viser demonstratoren i laboratoriet, sorte tall referer til liste over fysisk utstyr, blå tall referer til hvilken video utgang på datamaskinen (jfr liste over fysisk utstyr) display utstyret er tilkoblet, maskiner med flere videoutganger er i tilegg merket med bokstavkode.

11.2 Presentasjonsutstyr

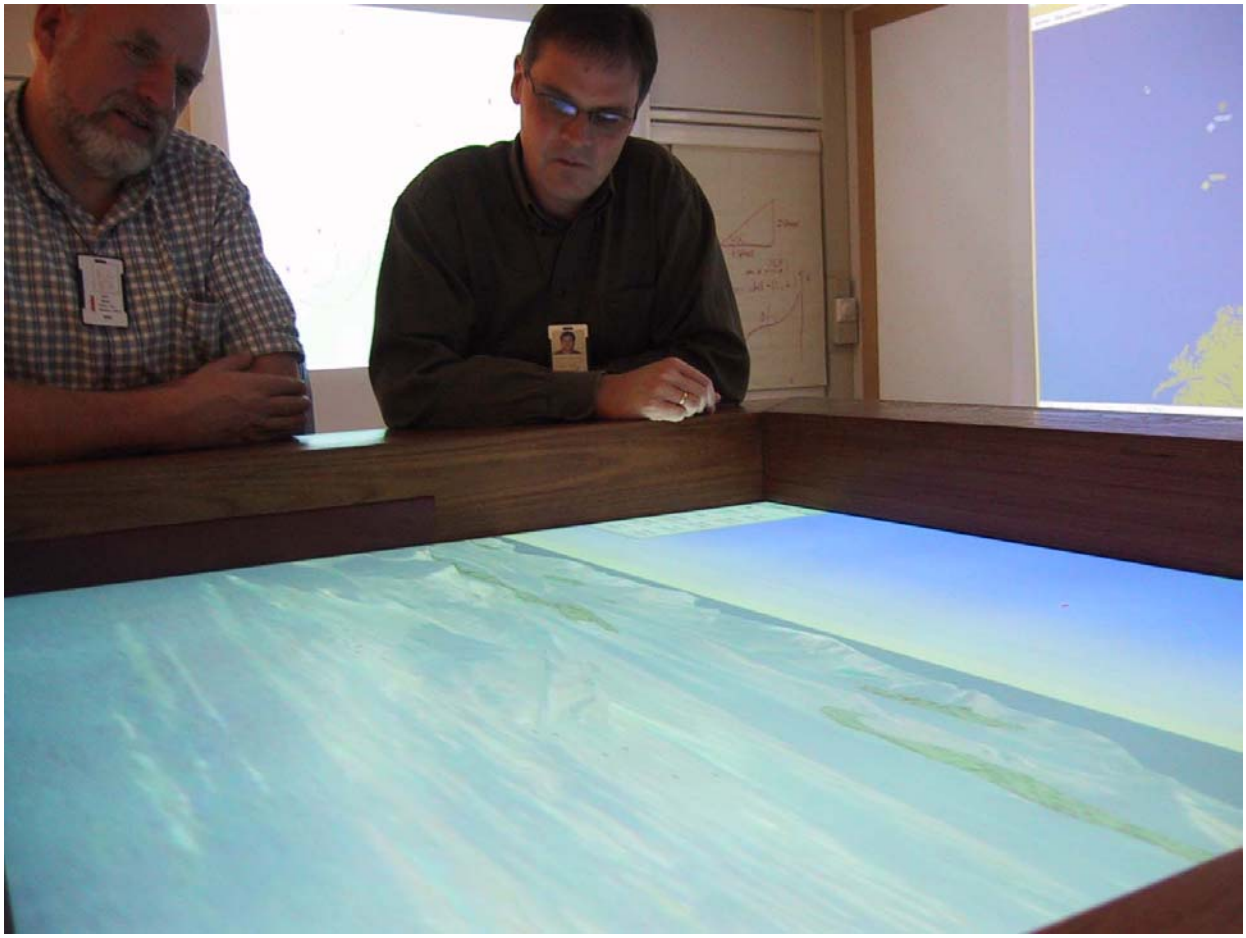
Presentasjonsutstyret som benyttes i demonstratoren er forsøkt tilpasset de fysiske målene på laboratoriet, se Figur 11-1 og Figur 11-2. Det er således ikke gjort noe dyptgående arbeid med hensyn til ergonomi og utforming, men heller ut i fra en praktisk tilnærming.

Figur 11-2 Laboratoriet

11.2.1 Visualiseringsbord

Operatørkonsollene og 3D visualiseringsbordet er spesiallaget for demonstratoren. Visualiseringsbordet har en skjermflate på ca 60" med fire operatørposisjoner og er laget av Video Systemer AS. Bordet kan i tillegg til vanlige to-dimensjonale bilder også vise stereoskopiske bilder. Prosjektørene som benyttes er 2 stk Barco sim 6 LCD (passiv stereo visning med sirkulærpolarisasjon) med en oppløsning på 1280x1024. Kravet at bordet skal kunne brukes både for 2D og 3D visualisering, stiller store krav til diffusjonsplaten (d v s der hvor bilde blir avbildet), den må ikke depolarisere lyset for da mistes bildeinformasjon for den ene av de to kanalene ved stereoskopisk visualisering, videre må den kunne gjengi bilde skarpt over hele flaten fra forskjellige betraktningsvinkler.

Vi har testet ut plater som støtter 2D og 3D visning fra forskjellige leverandører uten at vi har kommet til en fullgod løsning med hensyn til skarphet i bildet. Løsningen ser ut til å være å benytte to forskjellige plater, en for 2D og en for 3D. Maskinen som benyttes ved stereoskopisk visning er en SGI Octane2, den har en videoutgang for aktiv stereo, mens bordet krever passiv stereo, slik at vi bruker en aktiv/passiv stereo omformer for tilpasning av videosignalet.

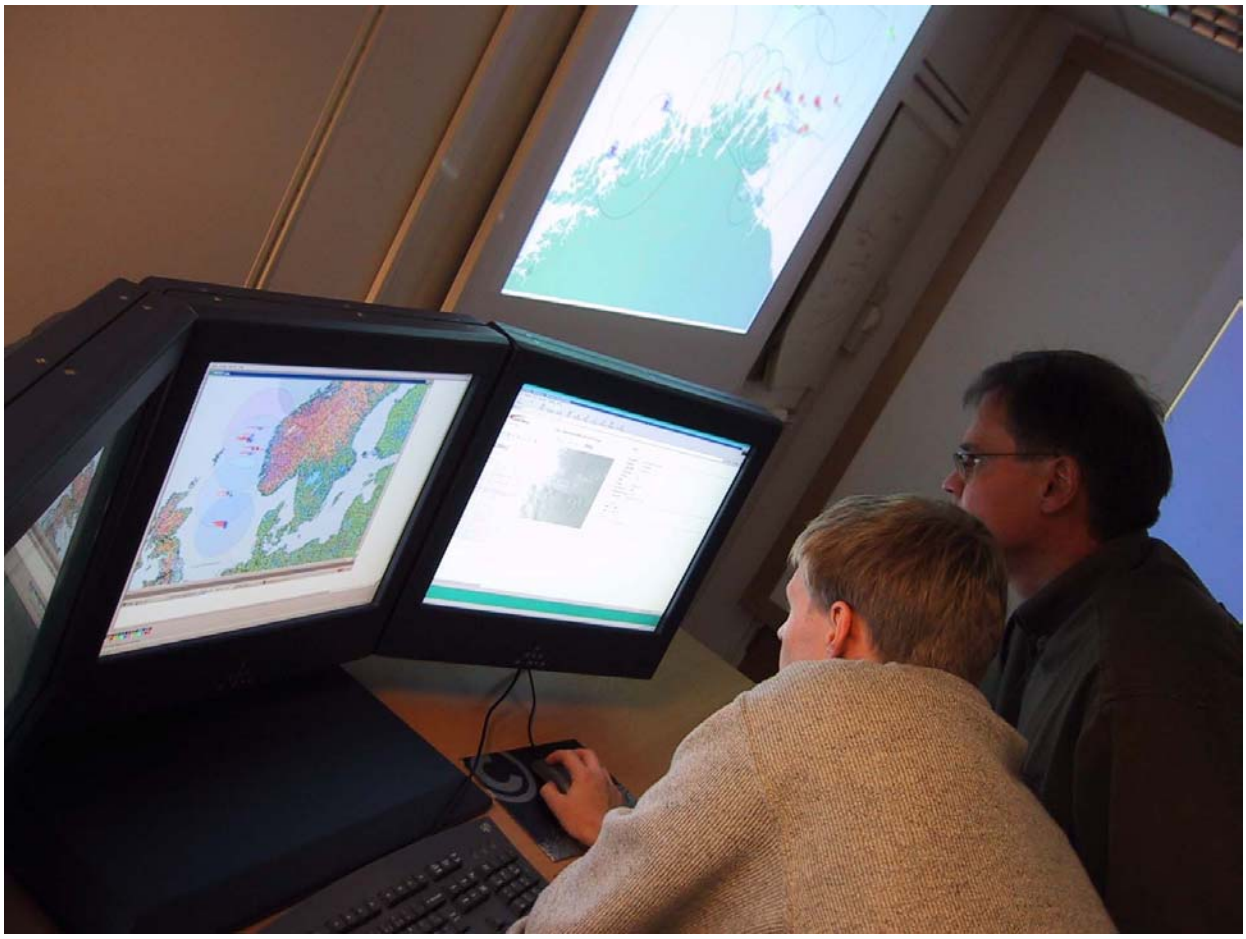


Figur 11-3 Visualiseringsbord for beslutningstakere

11.2.2 Operatørkonsoll for bildeoppbygging

Bildeoppbyggingsoperatøren vil i tilegg til selve kartbildet med tracksymboler etc ha behov for informasjon som støtter opp under selve bildeoppbyggingsprosessen. Denne tilleggsinformasjonen bør vises på separate skjermer for ikke å forkludre selve kartbildet for operatøren. Det ble derfor konstruert og produsert 2 terminaler som hver innholder 3 LCD-skjermer hos MJS (Moss Jern og Stanseindustri) som har lang erfaring innen design og produksjon av konsoller til maritimt bruk. Hvert panel eller skjerm er av type: NEC NL 160120AC27-01A med en billedflate på 21.3", et synsfelt 170 grader og en oppløsning på 1600 x 1200 piksler.

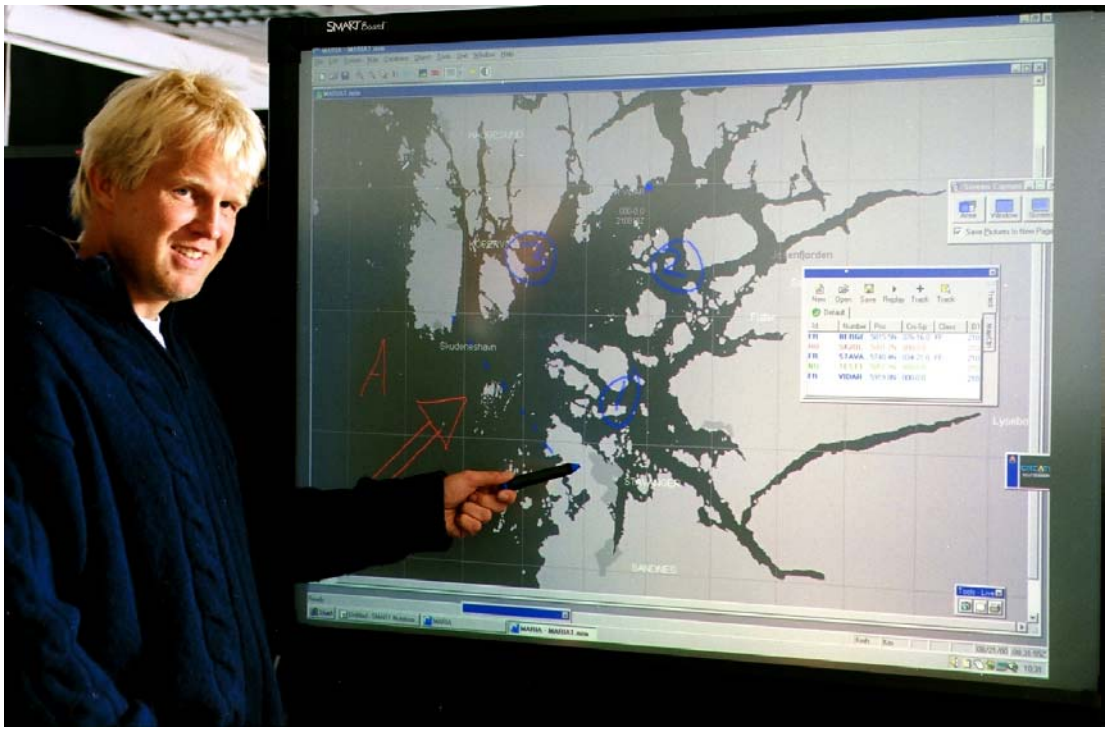
Maskinen som operatørterminalen er tilkoblet er en SGI ZX10VE som kjører NT operativsystem. For å drive 3 skjermer har vi montert inn 3 skjermkort av type Oxygen GVX1, ett på AGP-bussen og to på PCI-bussen.



Figur 11-4 Operatørterminal for bildeoppbygging

11.2.3 Smartboard 1602

Smartboard 1602 er en trykkfølsom skjerm på ca 60”, hvor en projektor belyser skjermen fra baksiden, såkalt bakprojeksjon. Projektoren som er benyttet er en JVC D-ILA med oppløsning på 1280x1024 pixler. Skjermen har en transparent trykkfølsom plate slik at muspekeren styres ved å bevege fingeren på skjermen, menyer aktiveres ved å dobbeltklikke i menyfeltet (tilsvarer venstre knapp på vanlig mus). Skjermen kan også settes i tegnemodus ved å benytte en av de fire ”fargepennene” (rød, grønn, blå og sort), og en kan da tegne på skjermen som vist i fig 11-6. I tillegg til fargepennene finnes også et ”elektronisk viskelær”. Skjermen kan således sees på som en ”white board” med skjermbildet som ”backdrop”. Informasjonen fra den trykkfølsomme skjermen overføres via en serielinje til en COM port på maskinen. Skjermbildene kan lagres i SmartNotebook.



Figur 11-5 Smartboard

Figur 11-6 Kommunikasjonsenhetene i laboratoriet. HF-modemene og satelittkommunikasjonsimulatoren kan sees i kabinettet under terminalen.

12 OPPSETT FOR KJØRING AV DEMONSTRATOR

Det er opprettet en felles brukerkonto på Unix for demonstratoren: kki. Denne brukeren fins kun på maskinene sthrmp1, sthrmp3 og sthmccis. All programkode og alle kjørbare filer ligger under KKISJØs Unix fellesområde, og det er der opprettet scriptfiler for hver komponent eller prosess som skal startes. På Windows skal demonstrator-området monteres som Y:.

Unix: /net/pilot/user/kkiss/demonstrator eller
/user/kki/kkiss/demonstrator

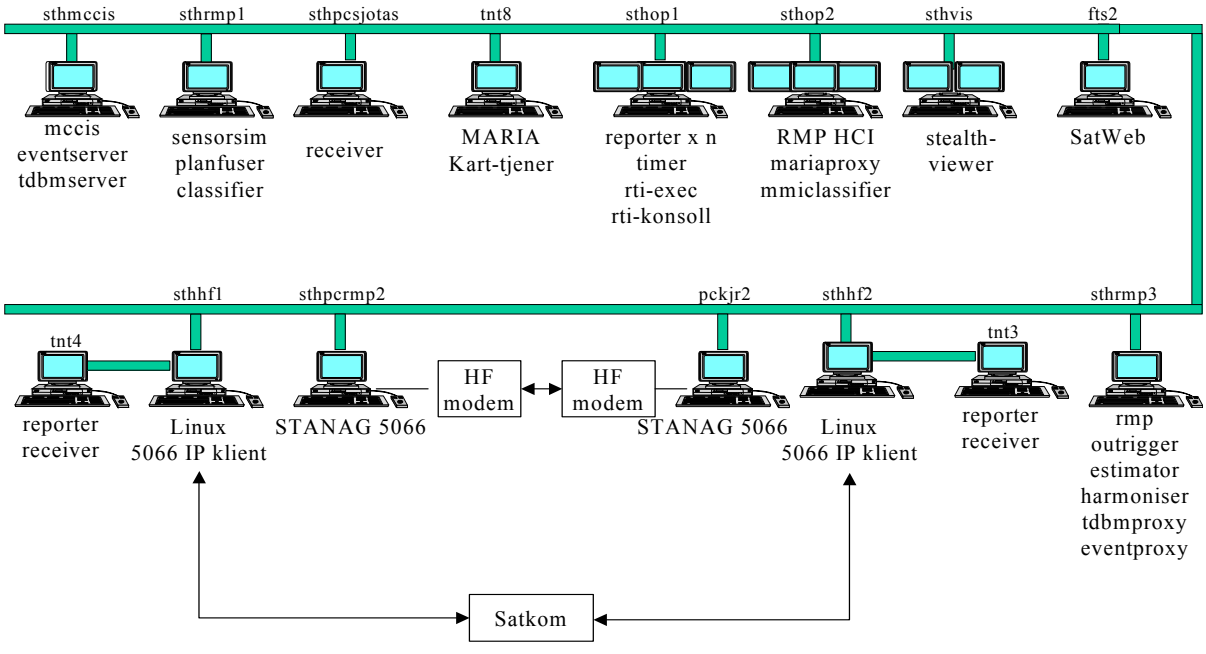
Windows: Y:\

Scriptfilene ligger på tre kataloger:

- bin
- coabsgrid.v2.0.0beta/scripts/unix/demonstrator
- coabsgrid.v2.0.0beta/scripts/win/demonstrator

12.1 Deployering av komponenter

Figur 12-1 viser en oversikt over hvilke datamaskiner de ulike komponentene kjøre på.



Figur 12-1 Deployeringen av de enkelte komponentene i demonstratoren

12.2 Prosedyre for å starte en demonstrasjon

I de følgende kapitlene beskrives prosedyrene for oppstart av demonstratoren. For alle kommandoer som må skrives av brukeren er det benyttet en monospace-font. Shell-promptet på Linux/Unix er vist som % mens det på Windows er vist som >.

12.2.1 Kommunikasjon

Det er implementert tre kommunikasjonsscenarier i demonstratoren. Disse er:

1. LAN
2. WAN over simulert HF-radio
3. WAN en simulert satellittkanal

Disse er nærmere beskrevet nedenfor.

12.2.1.1 LAN

Dette er normaltilstanden på kommunikasjonsoppsettet, og er status ved oppstart. I dette tilfellet kjøres trafikken mellom applikasjonene på tnt3 og tnt4 over KKI-SJØNETT 1, som er FFI innenett. Dette er et fiber LAN og vil ikke bli omtalt nærmere.

12.2.1.2 Simulert HF-radio

For oppsettet her forutsettes det at sthhf1, sthhf2, pckjr2 og sthpcrmp2 alle er oppe og kjører samt at modemene er slått på.

På pckjr og sthpcrmp2 gjør følgende:

1. Kjør programmet rf5710.exe. Snarvei til dette ligger på skrivebordet. Klikk på search og når modemet er funnet, avslutt programmet.
2. Kjør programmet MDH Control Centre. Snarvei til dette programmet ligger også på skrivebordet.
3. Sjekk at send og receive bitrate er lik ved å trykke på knappen merket 'S5066 GUI'. Hvis bitratene ikke er like, stopp MDH Control Centre og start på pkt 1 med å kjøre rf5710.exe igjen.

På sthhf1og sthhf2 gjør følgende som bruker root i et terminalvindu:

1. # hfmcast -b
2. Gå til et annet terminalvindu og skriv: # ./routeadd_mcast

Rute til motstående tnt(x)-maskin kommer da opp over nettporten tap0 og trafikk mellom tnt3 og tnt4 rutes nå over modemene som simulerer HF-radio. I tillegg tas eventuell multicast-trafikk hånd om.

Fila routeadd på sthhf1 er som følger:

```
# route add –host 131.1.12.0 tap0
# route add –net 131.1.12.0 netmask 255.255.255.0 gw 172.16.0.2
# /etc/rc.d/init.d/smcroute start
# smcroute –a eth1 131.1.12.2       224.0.1.85 eth0
# smcroute –a eth0 131.1.133.44   224.0.1.85 eth1
# smcroute –a eth0 131.1.132.172 224.0.1.85 eth1
```

```
# smcroute –a eth1 131.1.12.2       224.0.1.84 eth0
# smcroute –a eth0 131.1.133.44   224.0.1.84 eth1
# smcroute –a eth0 131.1.132.172 224.0.1.84 eth1
```

Fila routeadd på sthhf2 er som følger:

```
# route add –host 131.1.8.0 tap0
# route add –net 131.1.8.0 netmask 255.255.255.0 gw 172.16.0.1
# /etc/rc.d/init.d/smcroute start
# smcroute –a eth1 131.1.8.2        224.0.1.85 eth0
# smcroute –a eth0 131.1.133.44   224.0.1.85 eth1
# smcroute –a eth0 131.1.132.172 224.0.1.85 eth1
# smcroute –a eth1 131.1.8.2        224.0.1.84 eth0
# smcroute –a eth0 131.1.133.44   224.0.1.84 eth1
# smcroute –a eth0 131.1.132.172 224.0.1.84 eth1
```

Stoppes hfmcast-prosessen så vil nettporten tap0 forsvinne og trafikk mellom KKISJØNETT 2 og KKISJØNETT 3 settes tilbake til å benytte KKISJØNETT 1.

12.2.1.3 Simulert satellittkanal

For å få rutet trafikk over den simulerte satellittkanalen kjøres kommandoen # satcomup på henholdsvis sthhf1 og sthhf2. Det forutsettes at HF ikke er aktivert når dette gjøres samt at begge SR3000 ruterene og Simulator 2 er slått på. Simulert satellittforbindelse må manuelt stoppes, og dette gjøres ved å kjøre kommandoen # satcomdown på henholdsvis sthhf1 og sthhf2.

På sthhf1 inneholder fila satcomup følgende kommandoer:

```
# route del –net 131.1.12.0 netmask 255.255.255.0
# ifconfig eth2 131.1.9.1 netmask 255.255.255.0 up
# route add –net 131.1.10.0 gw 131.1.9.2 netmask 255.255.255.0 eth2
# route add –net 131.1.11.0 gw 131.1.9.2 netmask 255.255.255.0 eth2
# route add –net 131.1.12.0 gw 131.1.9.2 netmask 255.255.255.0 eth2
```

På sthhf1 inneholder fila satcomdown følgende kommandoer:

```
# ifconfig eth2 down
# route add –net 131.1.12.0 gw 131.1.134.139 netmask 255.255.255.0 eth0
```

På sthhf2 inneholder fila satcomup følgende kommandoer:

```
# route del –net 131.1.8.0 netmask 255.255.255.0
# ifconfig eth2 131.1.11.1 netmask 255.255.255.0 up
# route add –net 131.1.10.0 gw 131.1.11.2 netmask 255.255.255.0 eth2
# route add –net 131.1.9.0 gw 131.1.11.2 netmask 255.255.255.0 eth2
# route add –net 131.1.8.0 gw 131.1.11.2 netmask 255.255.255.0 eth2
```

På sthhf2 inneholder fila satcomdown følgende kommandoer:

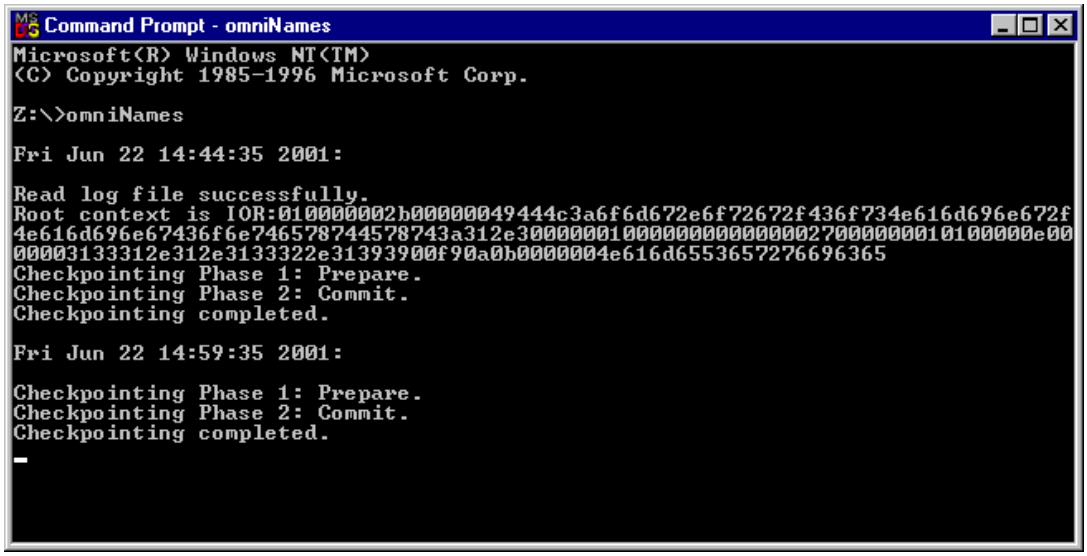
```
# ifconfig eth2 down
```

```
# route add –net 131.1.8.0 gw 131.1.134.138 netmask 255.255.255.0 eth0
```

12.2.2 MariaMapServer

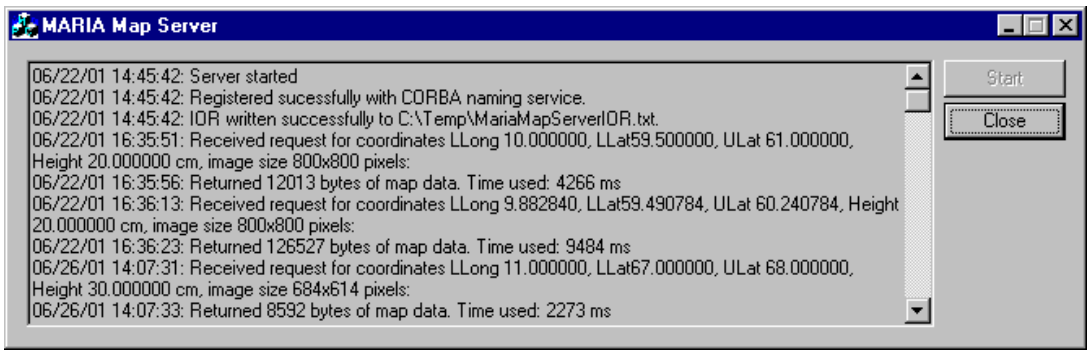
Nåværende versjon av MariaMapServer benytter OmniORB som ORB. OminORB må derfor være installert på samme server som MariaMapServer skal kjøres på. OmniORB er en CORBA 2 ORB fra AT&T og er sertifisert som CORBA 2.1 compliant.

Før oppstart av MariaMapServer, må OmniORBs Naming Service startes. Dette gjøres v h a kommandoen "> omniNames", se Figur 12-2.



Figur 12-2 CORBA Naming Service startes fra Command Prompt

MariaMapServer startes ved å kjøre programmet C:\Program Files\MARIAMapServer\ MariaMapServer.exe. Når MariaMapServer startes, vil et dialogvindu komme opp (Figur 12-3). Dialogvinduet inneholder et loggvindu, en START og en CLOSE knapp. Loggvinduet vil til enhver tid vise alle kartoperasjonene som kartserveren har utført, unntatt koordinat-transformasjoner. Når MariaMapServer er startet opp, må man klikke på START knappen for at serveren skal registrere seg hos CORBA Naming Service.



Figur 12-3 Kartoperasjoner blir vist i eget loggvindu

Når MariaMapServer startes, vil også en Interoperable Object Reference (IOR) skrives til filen "C:\Temp\MariaMapServerIOR.txt". Denne tekststrengen er en entydig global identifikator som identifiserer MariaMapServer som et CORBA objekt og som brukes av klienter for å komme i kontakt med kartserveren. For at komponenter som skal benytte seg av kartserveren, skal finne den, må filen "MariaMapServerIOR.txt" kopieres til katalogen "/raid1/gruppe/KKISS/demonstrator" på Pilot. PC'er som skal benytte kartserveren, må lokalt montere opp denne katalogen som "Y:\coabsgrid.v2.0.0beta" for at scriptfilene skal finne "MariaMapServerIOR.txt".

12.2.3 VisiBroker ORB

VisiBroker ORB skal normalt starte under oppstart av maskinen sthmccis, men dersom dette feiler så må den startes manuelt med kommandoen:

```
% /sbin/init.d/osagent start
```

Serveren vil vises i maskinens prosessliste med navnet "osagent".

12.2.4 VisiBroker navnetjeneste

VisiBroker navnetjeneste må startes manuelt på sthmccis etter oppstart av maskin med kommandoen:

```
% StartNameServer
```

I prosesslisten vil navnetjenesten vises som en (av mange) JAVA-prosesser.

12.2.5 TdbmServer

TdbmServer startes på sthmccis med kommandoen:

```
% StartTdbmServer
```

Dersom VisiBrokers ORB eller navnetjeneste ikke er startet så vil oppstart av TdbmServer feile.

12.2.6 EventServer

EventServer startes på sthmccis med % StartEventServer. Kommandoen starter også VisiBrokers eventtjeneste dersom den ikke allerede er startet.

12.2.7 TdbmProxy

TdbmProxy startes på sthrmp1 med kommandoen:

```
% tdbmproxy.rc
```

12.2.8 EventProxy

EventProxy startes på sthrmp1 med kommandoen:

```
% eventproxy.rc
```

12.2.9 Oppstart av demonstrator

Hvis kommandoene skal kjøres fra bin-katalogen står det spesielt. Ellers skal kommandoene kjøres fra coabsgrid.v2.0.0beta/scripts/unix/demonstrator (unix) eller coabsgrid.v2.0.0beta/scripts/win/demonstrator (windows).

Det første som må startes for at demonstartoren skal virke, er Grid-manageren til CoABS Grid. Denne startes fra bin-katalogen med kommandoen:

```
% StartGridManager
```

Når vinduet til manageren er kommet opp bør de gamle logg-filene fjernes ved å trykke på knappen 'Delete logs' under 'Grid controls'. Deretter startes RMI deamon og look-up-servicen ved å trykke på knappen 'Start Grid Deamons'. Hvis ikke begge da startes automatisk, må først 'Control Individual Deamons' og deretter 'Start RMID' eller 'Start LUS' trykkes avhengig av hvilken tjeneste som ikke startet automatisk. Merk at knappen 'Start LUS' først aktiviseres etter at 'Start HTTP' er trykket! Når Loop-up-servicen (LUS) og RMI deamon kjører, kan de andre komponentene startes.

MariaProxy er ikke avhengig av noen andre komponenter, og kan derfor startes når som helst etter at LUS og RMI-deamon er startet. Dette gjøres med kommandoen:

```
% mariaproxy.rc
```
eller
```
> mariaproxy.bat
```
Når mariaproxy er oppe, kan brukergrensesnittet startes med kommandoen:

```
% viewer.rc
```
eller
```
> viewer.bat
```

RMP-komponenten består av tre komponenter som må startes i denne rekkefølgen: Outrigger, selve modellen og estimatoren. Disse startes med kommandoene:

```
% outrigger.rc
% rmp.rc
% estimator.rc
```
eller
```
> outrigger.bat
> rmp.bat
> estimator.bat
```

Klassifikasjonskomponenten kan startes etter at RMP-komponenten er oppe, og startes med kommandoen:

```
% classifier.rc
```
eller
```
> classifier.bat
```

Koblingen mot MCCIS er også avhengig av at RMP-komponenten er oppe, og den består av tre komponenter: TdbmProxy, EventProxy og Harmoniser. Her må de to førstnevnte komponentene være oppe før Harmoniser startes, og de startes med kommandoene:

```
% tdbmproxy.rc
% eventproxy.rc
% harmoniser.rc
```
eller
```
> tdbmproxy.bat
> eventproxy.bat
> harmoniser.bat
```

Når de hittil nevnte komponentene er oppe, kan simuleringsomgivelsen startes. Det første som da gjøres er å starte RTI i bin-katalogen med kommandoen:

```
% StartRTI
```
Deretter kan sensorsim startes på samme sted med:

```
% StartSensorsimDemo –f
```

Når nå sensorsim skriver ut ’Waiting for ReadyToPopulate Announcement’, kan scenario-kontrolleren startes med kommandoen:

```
> controller.bat
```

For å få startet den, må det skrives inn ’1’ bak ’Federates to await’ og knappen ’apply’ må deretter trykkes. Så kan startknappen trykkes, og når kontrolleren skriver ut ’FED: Entering event loop’, kan tidstjenesten og reporterne startes med kommandoene:

```
> timer.bat
> reporter_nansen.bat
> reporter_trondheim.bat
> reporter_uav.bat
> reporter_navalops.bat
```

Når så alle reporterne skriver ut ’FED: Entering event loop’, kan mottakeren startes med kommandoen:

```
> receiver.bat
```

Dersom fridtjofnansen skal rapportere over HF, må HF-modemforbindelsen startes som beskrevet i kapittel 12.2.1.2. Deretter må HF-senderen startes på maskinen tnt3 med

kommandoen:

```
> hfsender.bat
```

mens mottakeren må startes på tnt4:

```
> receiver.bat
```

13 KONKLUSJON

Rapporten har beskrevet demonstratoren for maritim bildeoppbygging som ble utviklet i prosjekt 730 – KKI-Sjø. De fleste sider av demonstratoren er beskrevet med hovedvekt på de tekniske sidene.

Demonstratoren ble utviklet for å kunne vise (isteden for å beskrive) noen av de viktigste anbefalingene om fremtidig utvikling av ledelsessystem for maritime operasjoner. Den har vært vist til mange miljøer i Forsvaret i tillegg til oppdragsgiver og har således oppfylt sin rolle.

Deler av demonstratoren vil blir benyttet i videre arbeider og eksperimentering med kommando og kontroll informasjonssystem med egenskaper som er nødvendige for å kunne realisere nettversbasert forsvar (NBF). Demonstratoren har allerede vist noen slike egenskaper, som f eks nettverksbasert kommunikasjon og dynamisk sammenkobling av programvarekomponenter og videre arbeider vil bygge på disse. Deler av demonstratoren vil kunne spille en viktig rolle i med etablering av distribuert Battle Lab for NBF (36).

Simuleringsomgivelsen til demonstratoren har også spilt en viktig rolle for å bygge opp kunnskap om moderne simuleringsteknologi. Den er den første simuleringen ved FFI som har tatt i bruk moderne simuleringsteknologi knyttet til distribuert simulering vha High Level Architecture. Deler av simuleringsomgivelsen samt den kunnskapen som er opparbeidet vil bli benyttet i videre arbeider med simuleringsteknologi ved FFI.

Litteratur

(1) Navy center for tactical systems interoperability (1997): Operational specification for the over-the-horizon targeting GOLD; Revision C, Navy center for tactical systems interoperability.

(2) NATO (1999): STANAG No. 5511; Tactical data exchange - Link 11/Link 11B, NATO (NATO Unclassified).

(3) NATO (1998): STANAG No. 5522; Tactical data exchange - Link 22, NATO (NATO Unclassified).

(4) ADSIA (1997): STANAG No. 5500; AdatP-3 Part II-IV (cd-rom), baseline 10.0.1 (NATO Confidential).

(5) NATO (1987): Appendix 1 to annex A, STANAG No. 4420 (edition 2), NATO.

(6) Department of defence (1997): MIL-STD 2525A; Department of defence interface standard: Common warfighting symbology, Department of defence USA

(7) ATCCIS permanent working group: ATCCIS Working Paper 5-3; ATCCIS Battlefield Generic Hub Level 2 data model, SHAPE Belgium.

(8) Steffensen P B, Stavnem L, Hede M (1998): GRACE Common Model (GCM) revision 1.2, The GRACE consortium.

(9) Jensen, Finn V (1996): An introduction to Bayesian networks, UCL Press, London, UK, 178.

(10) Hafnor H, et al (2000): Arkitekturer for kommando og kontroll informasjonssystemer - Arkitekturer, komponentbasert systemutvikling og teknologier, FFI/RAPPORT-2000/04582.

(11) Hansen B J, Korsnes R (2001): Algoritmer for kombinasjon av klassifikasjonsinformasjon – Valg av algoritme for kombinasjon av klassifikasjonsinformasjon til bruk i en bildeoppbyggingsdemonstrator, FFI/NOTAT-2001/00177, Forsvarets forskningsinstitutt.

(12) Control of Agent Based Systems: http://coabs.globalinfotek.com

(13) Jini Network Technology: http://wwws.sun.com/software/jini

(14) Statens Kartverk (1990) SOSI et standardformat for digitale geodata (Versjon 1.4).

(15) Mevassvik Ole M (1996): Sensorsim - simuleringsprogram for sjøovervåkning, FFI/NOTAT-96/05092, Forsvarets forskningsinstitutt.

(16) Hansen Bjørn Jervell, Sanden Helge (2000): Sensorsim – brukerveiledning, FFI/NOTAT-2000/06434, Forsvarets forskningsinstitutt.

(17) INRI (2000) INRI C4I Software (ICS) Product Docomentation CDROM

(18) Van Laar D L (2000): Psychological and chartographic principles for the production of visual layering effects in computer displays. In: Displays Volume 22 Issue 4 September 2001

(19) Reynolds Linda (1994): Colour for air traffic control displays. In: Displays Volume 15 Number 4 October 1994

(20) Kuhl, F et al (1999): Creating Computer Simulation Systems, An Introduction to the High Level Architecture, Prentice Hall, New Jersey.

(21) Mevassvik O M, Bråthen K, Hansen B J (2001): A Simulation Tool to Assess Recognized Maritime Picture Production in C2 Systems, In: *Proceedings of the 6th International Command and Control Research and Technology Symposium*, U. S. Naval Academy, Annapolis, MD, 19-21 July 2001.

(22) US DoD (1998): High Level Architecture Rules, Version 1.3.

(23) US DoD (1997): High Level Architecture Object Model Template, Version 1.1.

(24) US DoD (1998): High Level Architecture Interface Specification, Version 1.3.

(25) Simulation Interoperability Standards Organization Inc (1999): Guidance, Rationale, and Interoperability Modalities for the Real-time Platform Reference Federation Object Model (RPR FOM), Version 1.0.

(26) IEEE (2000): IEEE Standard for Modeling and Simulation (M&S) High Level Architecture, Framework and Rules.

(27) IEEE (2000): IEEE Standard for Modeling and Simulation (M&S) High Level Architecture, Object Model Template (OMT) Specification.

(28) IEEE (2000): IEEE Standard for Modeling and Simulation (M&S) High Level Architecture, Federate Interface Specification.

(29) Mevassvik O M, Løkka A (2000): Fusion of radar tracks, reports and plans, Proceedings FUSION'2000 3rd International Conference on Information Fusion, Paris, 3 juli.

(30) Løkka A, Mevassvik O M (2000): Metode for integrasjon av radar track, planer og rapporter i maritim bildeoppbygging, FFI/NOTAT-2000/04614, Forsvarets forskningsinstitutt.

(31) Bråthen Karsten, Bergene Trond, Enemo Geir, Hafnor Hilde, Leere Anton B, Malerud Stein, Mevassvik Ole M, Rose Kjell, Veum Kurt A (2001): (U) Anbefalt fremtidig ledelsessystem for maritime operasjoner, FFI/RAPPORT-2001/02935 (Begrenset).

(32) Bråthen Karsten, Feet Else H, Veum Kurt A, Mevassvik Ole M, Rose Kjell, Hafnor Hilde (2001): (U) Anbefalt konsept for fremtidig ledelsessystem for maritime operasjoner, FFI/RAPPORT-2001/02949 (Begrenset).

(33) http://java.sun.com/j2ee

(34) Object Management Group (2002): Common Object Request Broker Architecture (CORBA) v3.0: Core Specification.

(35) Bråthen Karsten, Armo Knut R, Bergene Trond (2001): (U) Anbefalte tiltak for fremtidig ledelsessystem for maritime operasjoner, FFI/RAPPORT-2001/02936, Forsvarets forskningsinstitutt (Begrenset).

(36) Hansen B J, Mevassvik O M, Bråthen K (2003): A Demonstrator for Command and Control Information System Technology Experimentation. In: *Proceedings of the 8th International Command and Control Research and Technology Symposium*, National Defence University, Washington, DC, USA, June 17-19 2003.

APPENDIKS

A IDL-GRENSESNITT TIL MARIA MAPSERVER

Denne versjonen av MariaMapServer inneholder en funksjon for å hente kart og to funksjoner for koordinattransformasjoner. IDL-grensesnittet er definert ved:

```
#ifndef MAPSERVERIF_IDL
#define MAPSERVERIF_IDL
interface MapServerIF {

enum mapDataFormat        { M_UNDEFINED, M_PNG_FAST, M_PNG_BEST, M_JPG_75,
M_JPG_90, M_BMP24, M_PPM, M_WBMP, M_GIF, M_GIF4BITS};

struct mapData {
  mapDataFormat   eFormat; // format of data
  long                    iSize;          // size of aData in bytes
  string                  sError;         // error message when iSize is 0
  string                  sScale;         // scale eg. 1:50678
  long                    iTimeUsed;      // time in millisecs used to generate map
  sequence<octet>         aData;          // actual data
  };

// Generate map for the given coordinates
mapData GenerateMap(
  in mapDataFormat        eDataFormat,        // format of map data to be returned
  in string               sTemplate,          // Filename (no path) of template to use
  in Boolean              bShadowMap,         // generate shadowed map
  in long                 iMapWidth,          // Requested Map size in pixel
  in long                 iMapHeight,         // Requested Map size in pixel
  in double               dMapHeight,         // Map height in cm (used to calc. Scale)
  in double               dLowerLongitude,    // lower longitude of map to generate
  in double               dLowerLatitude,     // lower latitude of map to generate
  in double               dUpperLatitude      // upper latitude of map to generate
  );

// get coordiantes from x,y position within map
void GetDecimalGrades(
  in long           iMapWidth,          // Map size in pixel
  in long           iMapHeight,         // Map size in pixel
  in double         dLowerLongitude,    // lower longitude of map
  in double         dLowerLatitude,     // lower latitude of map
```

```
    in double       dUpperLatitude,     // upper latitude of map
    in long         iXcoord,            // x-coordinate of requested position
    in long         iYcoord,            // y-coordinate of requested position
    out double      dLatitude,          // returned latitude of iYkoord
    out double      dLongitude          // returned longitude of iXkoord
    );

// get x,y position from lat,lon in map
void GetPixelCoordinates(
    in long         iMapWidth,          // Map size in pixel
    in long         iMapHeight,         // Map size in pixel
    in double       dLowerLongitude,    // lower longitude of map
    in double       dLowerLatitude,     // lower latitude of map

    in double       dUpperLatitude,     // upper latitude of map
    in double       dLatitude,          // latitude of requested position
    in double       dLongitude,         // longitude of requested position
    out long        iXcoord,            // returned x-pixel position
    out long        iYcoord             // returned y- pixel position
    );
};
#endif //MAPSERVERIF_IDL
```

FORDELINGSLISTE

FFIE **Dato:** 1 august 2003

RAPPORTTYPE (KRYSS AV)	RAPPORT NR.	REFERANSE	RAPPORTENS DATO
X RAPP ☐ NOTAT ☐ RR	2003/02748	FFIE/730/134	1 august 2003
RAPPORTENS BESKYTTELSESGRAD		**ANTALL EKS UTSTEDT**	**ANTALL SIDER**
UGRADERT		20	105
RAPPORTENS TITTEL		**FORFATTER(E)**	
DEMONSTRATOR FOR BILDEOPPBYGGING		MEVASSVIK Ole Martin, HAFNOR Hilde, HANSEN Bjørn Jervell, LEERE Anton B, ROSE Kjell, SANDEN Helge, URDAHL Morten, VIKEN Kjell O, BRÅTHEN Karsten	
FORDELING GODKJENT AV FORSKNINGSSJEF		**FORDELING GODKJENT AV AVDELINGSSJEF:**	
Vidar S Andersen		Johnny Bardal	

EKSTERN FORDELING

ANTALL	EKS NR	TIL
		FLO/IKT
1		v/Torstein Haugland
1		FLO/Sjø
1		KNM T

INTERN FORDELING

ANTALL	EKS NR	TIL
9		FFI-Bibl
1		Adm direktør/stabssjef
1		FFIE
1		FFISYS
1		FFIBM
1		FFIN
3		Restopplag til Biblioteket
		Elektronisk fordeling:
		FFI-Veven
		Vidar S Andersen (VSA)
		Ole Erik Hedenstad (OEH)
		Karsten Bråthen (KaB)
		Ole Martin Mevassvik (OMM)
		Hilde Hafnor (HHa)
		Bjørn Jervell Hansen (BHn)
		Anton B Leere (ABL)
		Kjell Rose (KjR)
		Helge Sanden (HSa)
		Morten Urdahl (MoU)
		Kjell O Viken (KOV)
		Tommy Gagnes (ToG)
		Rolf Rasmussen (RRa)
		Geir Sletten (GSl)
		Erik Nordø (ErN)
		Stig Lødøen (SEL)
		Arne Cato Jenssen (ACJ)
		Reinert Korsnes (RKo)
		Richard Moe Gustavsen (RMG)

FFI-K1 Retningslinjer for fordeling og forsendelse er gitt i Oraklet, Bind I, Bestemmelser om publikasjoner for Forsvarets forskningsinstitutt, pkt 2 og 5. Benytt ny side om nødvendig.