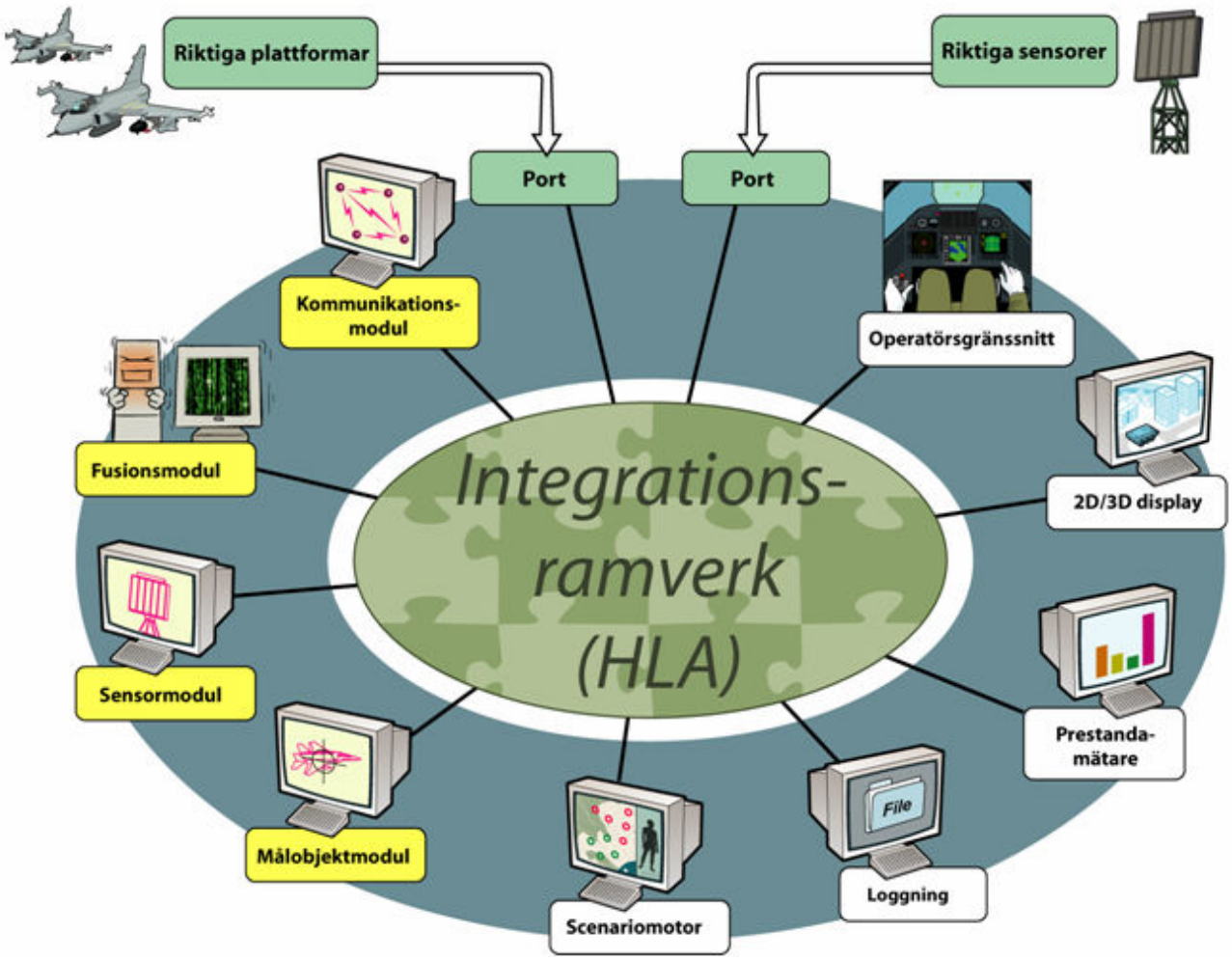
Slutrapport MOSART

Tobias Horney, Mikael Brännström, Joakim Bergman,
Dennis Andersson, Robert Forsgren, Simon Ahlberg, Björn
Lindahl, Anders Törne



FOI-R--1814--SE
ISSN 1650-1942

Ledningssystem
Användarrapport

December 2005

FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1350 anställda varav ungefär 950 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömningen av olika typer av hot, system för ledning och hantering av kriser, skydd mot hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Ledningssystem
Box 1165
581 11 Linköping

Tel: 013-37 80 00
Fax: 013-37 81 00

www.foi.se

Tobias Horney, Mikael Brännström, Joakim Bergman, Dennis Andersson, Robert Forsgren, Simon Ahlberg, Björn Lindahl, Anders Törne

Slutrapport MOSART

FOI-R--1814--SE
ISSN 1650-1942

Ledningssystem
Användarrapport

December 2005

Utgivare
FOI - Totalförsvarets forskningsinstitut
Ledningssystem
Box 1165
581 11 Linköping

Rapportnummer, ISRN
FOI-R--1814--SE

Klassificering
Användarrapport

Forskningsområde
4. Ledning, informationsteknik och sensorer

Månad, år
December 2005

Projektnummer
E7090

Delområde
41 Ledning med samband och telekom och IT-system

Delområde 2

Författare/redaktör
Tobias Horney
Björn Lindahl
Mikael Brännström
Anders Törne
Joakim Bergman
Dennis Andersson
Robert Forsgren
Simon Ahlberg

Projektledare
Tobias Horney

Godkänd av
Martin Rantzer

Uppdragsgivare/kundbeteckning
FM

Tekniskt och/eller vetenskapligt ansvarig
Tobias Horney

Rapportens titel
Slutrapport MOSART

Sammanfattning
Detta dokument utgör slutrapport för FoT-projektet "MOSART – systeminteraktion" som pågått 2003 till 2005. Projektresultaten utgörs av den kunskap som byggts under projektets gång samt den testbäddsspecifikation och mjukvara som tagits fram och presenteras i rapporten. Fokus i rapporten är på MOSART testbädd, dvs det simuleringsramverk som tagits fram i projektet. Denna simuleringsmiljö/simuleringsramverk har utvecklats för att uppnå de mål som funnits för projektverksamheten sedan starten. Det primära målet har varit att öka effektiviteten i FOI forskning genom att möjliggöra integration av forskningsresultat från olika forskningsgrupper. På så vis kan simuleringskomponenter i form av både generella verktyg (tex scenario- och visualiseringsverktyg) och tidigare forskningsresultat återanvändas av många projekt. Ett annat mål har varit att underlätta överföring av FOI forskningsresultat till kund. Detta har uppnåtts bl a genom att testbädden underlättar framtagande av bra demonstratorer genom att den innehåller bra verktyg för tex scenariohantering och 2D- och 3D-visualisering. Testbädden är även tänkt att fungera som en möjliggörare för samarbeten mellan FOI och andra organisationer. Exempel på detta är samarbetet med DSO/DSTA i Singapore som skett med testbädden som bas. Vid framtagandet av testbädden har kunskap förvärvats som i stor omfattning matchar det sista målet med projektet, nämligen kunskapsuppbyggnad kring problem som uppstår vid byggandet av ett nätverksbaserat försvar.
En indikation på att projektet varit framgångsrikt är att ett tiotal forskningsprojekt på FOI redan använder testbädden.

Nyckelord
MOSART, HLA, testbädd, distribuerad simulering

Övriga bibliografiska uppgifter

Språk Svenska

ISSN 1650-1942

Antal sidor: 56 s.

Distribution enligt missiv

Pris: Enligt prislista

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Ledningssystem Box 1165 581 11 Linköping	FOI-R--1814--SE	User report
	Programme Areas 4. C4ISTAR	
	Month year December 2005	**Project no.** E7090
	Subcategories 41 C4I	
	Subcategories 2	

Author/s (editor/s)		Project manager
Tobias Horney	Björn Lindahl	Tobias Horney
Mikael Brännström	Anders Törne	**Approved by**
Joakim Bergman		Martin Rantzer
Dennis Andersson		**Sponsoring agency**
Robert Forsgren		SwAF
Simon Ahlberg		**Scientifically and technically responsible**
		Tobias Horney

Report title (In translation)

Final Report MOSART

Abstract

This document is the final report for the project "MOSART – system interaction" which has run from 2003 to 2005. The project result consists of the knowledge acquired during the project and the testbed specification and software which has been developed and is presented in the report.

Focus in the report is on the MOSART testbed, i.e. the simulation framework which has been developed in the project. This simulation environment/framework has been developed to fulfill the objectives of the project.

The primary objective has been to increase the efficiency of FOI research by enabling integration of research results from different research teams. By doing this, different types of simulation components such as general simulation tools (e.g. scenario and visualization tools) and earlier research results kan be reused in many projects. Another objective has been to make research result delivery to FOI customers more effective. This objective has been fulfilled since the testbed contains good tools for e.g. scenario editing/execution and 2D and 3D visualization. The testbed should also be an enabler for collaborations between FOI and other organizations. An example of this is the collaboration with DSO/DSTA in Singapore which has run with the testbed as a basis. When developing the testbed, much knowledge about how to build a network based defence has been acquired. Building this kind of knowledge was another objective with the project.

An indication of project success is that about ten research projects at FOI are already using the testbed for their simulations and demonstrations.

Keywords

MOSART, HLA, test-bed, distributed simulation

Further bibliographic information	Language Swedish
ISSN 1650-1942	**Pages** 56 p.
	Price acc. to pricelist

Innehållsförteckning

Definitioner och förkortningar

API	Application Programming Interface
DMSO	Defence Modeling and Simlation Office
Federat	Simuleringskomponent som är uppkopplad i en HLA-baserad simulering
Federation	En mängd federater som exekverar ett gemensamt scenario via HLA.
FED-fil	Fil som i HLA 1.3 federationer innehåller den del av FOM:en som behövs vid exekvering
FOM	Federation Object Model
HLA	High Level Architecture
IDE	Integrated Development Environment
MOSART	MOdeling and Simulation for Analysis and Research Test-bed
RPR-FOM	Real-time Platform Reference FOM
RT90	Rikets Nät
RTI	Run-Time Infrastructure
SRV	Räddningsverket
XML	eXtensible Markup Language

1 Introduktion

FoT-projektet "MOSART - systeminteraktion" startades 2002-01-01. Detta är projektets slutrapport. Projektresultaten utgörs av den kunskap som byggts under projektets gång samt den testbäddsspecifikation och mjukvara som tagits fram och presenteras i rapporten.

Avsikten med verksamheten är att skapa en plattform som främjar forskningsområdes- och organisationsöverskridande verksamhet inom t.ex. sensor-, datafusions- och telekrigsområdena, samt underlättar återanvändning av forskningsresultat, t.ex. i större konceptdemonstratorer. MOSART är en ansträngning att öka kostnadseffektiviteten för övrig forskning, att tillhandahålla möjligheter till interna/externa samarbeten samt underlätta resultatöverföring från forskningsprojekt till dess kund.

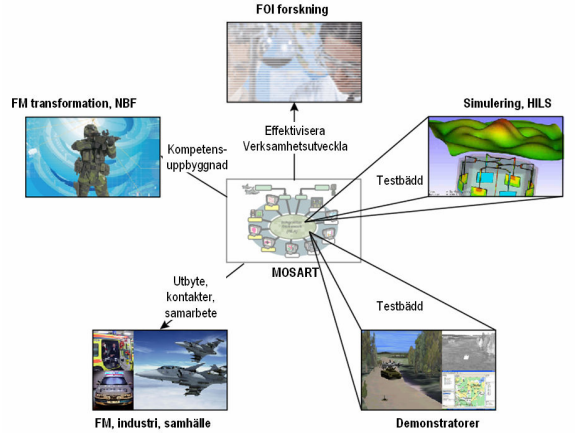
Slutmålen för projektet har varit:

- **Effektivitet:** att möjliggöra integration av forskningsresultat från olika forskningsgrupper (forskningsområdesöverskridande verksamhet) inom FOI så att det enskilda forskningsresultatet kan värderas i ett större sammanhang och bättre demonstratorer kan åstadkommas med mindre resursutnyttjande genom återanvändning av komponenter.
- **Kunskapsuppbyggnad:** att få nödvändiga erfarenheter av och kompetens inom hur en sådan plattform byggs och därmed kunna ge underlag från en labbmiljö så att FOI kan delta i utvecklingen av Sveriges nätverksbaserade försvar, speciellt med avseende på integrations- och arkitekturfrågor.
- **Länk till andra organisationer:** att med hjälp av testbädden möjliggöra utökade kontakter och samarbeten med andra organisationer såsom FM, FMV och industrin.

Genom testbäddsutvecklingen ges möjlighet till studier av t ex sensorer, datafusion och telekrig i en nätverksbaserad miljö. Många av de problem projektet ställts inför är gemensamma med de problem som måste hanteras vid etablerandet av ett nätverksbaserat försvar:

- Integration av system och delsystem på olika nivåer
- Sammankoppling av nya och gamla system
- Sammankoppling av system med olika tidskritikalitet
- Krav på modularitet och förmåga till evolutionär utveckling

I figur 1 återfinns en schematisk beskrivning av syftet med MOSART.



Figur 1. Schematisk beskrivning av syftet med MOSART.

2 MOSART projektresultat

Det huvudsakliga resultatet från MOSART-projektet är den testbädd som tagits fram. Denna testbädd är en simuleringsmiljö där forskningsresultat kan integreras för att på så vis möjliggöra forskningsområdesöverskridande verksamhet. Testbädden möjliggör att enskilda forskningsresultat kan värderas i ett större sammanhang och att FOI forskningsresultat kan överföras till kund på annat (och förhoppningsvis mer lättillgängligt) sätt än via rapporter, t ex genom demonstrationer eller via operatörsgränssnitt där en användare direkt kan interagera med forskningsresultaten i den simulerade värld som MOSART testbädd erbjuder.

MOSART testbädd består huvudsakligen av följande delar:

- Regler för integration och integrationsprogramvara
- Grundläggande funktionalitet inkl. användarinteraktion
- Forskningsresultat och stödmoduler från andra projekt
- Koppling till verklighet
- Tjänstekoncept inkl. integrationsprogramvara
- Omvärldshantering

I nästa kapitel presenteras den övergripande designen av testbädden. Därefter följer ett antal kapitel som presenterar de ovan uppräknade delarna.

Utöver själva testbädden finns:

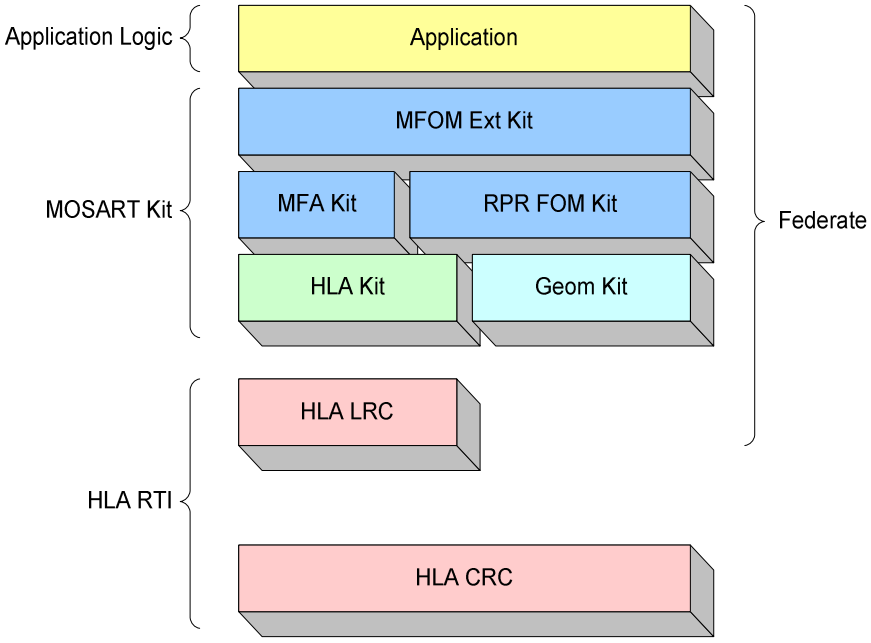
- Idéer kring och genomfört pilotförsök med att integrera simulering av kommunikation i testbädden
- Ett data repository för att göra insamlade sensordata från fältförsök tillgängliga och sökbara
- Olika typer av dokumentation
- Ett förslag på hur programvaran som utgör MOSART testbädd ska göras tillgänglig för andra organisationer

Dessa delar presenteras i varsitt kapitel efter kapitlen som beskriver testbädden.

3 MOSART testbädd – övergripande design

MOSART testbädd består i huvudsak av två delar: integrationsramverk samt ett flertal verktyg (moduler) som kan användas tillsammans i en simulering eller demonstration. Utöver detta tillkommer de forskningsresultat som integrerats i testbädden och därmed finns tillgängliga för återanvändning.

Integrationsramverket bygger på HLA (High Level Architecture) [1] och består av ett antal mjukvarubibliotek samt federationsöverenskommelser. Mjukvarubiblioteken syftar till att förenkla utvecklingen av en simuleringskomponent medan federationsöverenskommelserna sätter upp regler för att simuleringskomponenterna ska vara interoperabla. Figur 2 visar översiktligt hur integrationsramverket ser ut. MOSART federationsöverenskommelser (även kallade regler för integration,se kap 4) är specifikationen för att ta fram en MOSART-kompatibel federat, oberoende om MOSART mjukvarubibliotek används som stöd eller inte.



Figur 2. Översikt av mjukvarubiblioteken i MOSART.

I dagsläget finns fullt stöd för HLA 1.3 och begränsat stöd för HLA 1516 (via adapter till pRTI 1516). Ramverket har plug-and-play-stöd för olika RTI:er såsom Pitch pRTI 1.3 v2, Pitch pRTI 1516 (via adapter), Mäk RTI 1.3 och DMSO RTI 1.3 NG v5. En definitiv övergång till HLA 1516 planeras ske i början av 2006. Bibehållen bakåtkompatibilitet med HLA 1.3 kommer då att vara en viktig aspekt.

I MOSART testbädd finns ett flertal mjukvarubibliotek som väsentligt förenklar och snabbar upp utvecklingstiden för en HLA-federat. Fokus i designen av dessa mjukvarubibliotek har varit att de ska vara enkla att använda. Med endast grundläggande HLA-kunskaper kan en utvecklare snabbt bygga en HLA-federat, där mjukvarubiblioteken hanterar detaljer som hur komplexa datastrukturer konverteras till och från bytedata, håller reda på sk handles och andra andra saker som en applikationsutvecklare inte ska behöva bry sig om. Konvertering av positioner, hastighetsvektorer och orienteringar mellan olika koordinatsystem (transverse-mercator-projektion såsom RT90, geodetiskt, geocentriskt, etc) är ett annat exempel, liksom dödräkning av

positioner för att minimera nätverkstrafik. Alla mjukvarubibliotek finns både i Java och i C++ och är plattformsoberoende för att inte låsa in en användare av testbädden i någon given miljö.

Federationsöverenskommelserna (regler för integration i MOSART testbädd, se kap 4) är de krav som ställs på ingående simuleringskomponenterna (federaterna) i MOSART testbädd. Dokumentet beskriver hur scenarioinformation ska distribueras mellan olika simuleringskomponenter, en simulerings olika faser (init, start, stop, reset) och övrigt som behöver styras upp för att simuleringskomponenter ska vara interoperabla. Utöver federationsöverenskommelserna upprätthålls en referens-FOM (Federation Object Model), *MOSART reference FOM*, som innehåller standard-objekt och interaktioner som är vanligt förekommande bland användarprojekten på FOI. Detta för att undvika att samma fenomen modelleras på olika sätt inom FOI.

Modulerna i MOSART är fristående verktyg som kan användas i olika simuleringssammanhang beroende på krav och preferenser. Några av verktygen är speciellt framtagna för testbädden (Federation Manager, NetScene) för scenariohantering medan andra är anpassade forskningsresultat (MIND) eller kommersiella verktyg (Vega) för visualisering. Flera av modulerna är rena forskningsresultat (t ex sensormodeller) som är resultatet av en lyckad integration för simulering och demonstration. Ett valfritt urval av dessa moduler kan användas för att bygga nästa simulator eller demonstrator. På så sätt kan dessa moduler (t ex forskningsresultat) återanvändas för att relativt smärtfritt ingå i ett annat sammanhang.

3.1 Framtida arbete

Fram till idag har fokus legat på att simuleringar i MOSART framförallt ska kunna ske tidssynkroniserat, då många av de simuleringskomponenter som hittills integrerats, exempelvis de mer avancerade sensormodellerna, inte klarar av att exekvera i realtid. I och med ökat intresse för sk mixed reality där mänskliga operatörer och live-data från sensorer kopplas in i simuleringen så ökar kraven på realtidsprestanda. MOSART testbädd har alltid stött realtidssimuleringar, men då de flesta simuleringar har skett tidssynkroniserat har utvecklingen drivits längst där. Dessa krav på realtidssimulering medför att fokus skiftas något mot bättre prestanda i mjukvarubiblioteken, dock med bibehållen enkelhet för användaren, samt att man ser över sitt val av federationsdesign lite mer riktat mot realtid, t ex att man väljer att inte använda tidstjänsterna i HLA i vissa fall för att dessa drar för mycket tid. Målet med enkelhet kvarstår dock, vilket bl a medför att det inte bör vara någon större skillnad för en utvecklare som tar fram en realtidsfederat jämfört med om man tar fram en tidssynkroniserad federat eller om man vill växla mellan dessa lägen.

4 Regler för integration

De regler för integration, även kallade federationsöverenskommelser eller på engelska *MOSART Federation Agreements*, som tagits fram beskriver hur en applikation ska uppträda för att kunna delta i en simulering i MOSART testbädd. Det dokument som i sin helhet (mycket mer detaljerat än i detta kapitel) beskriver MOSART:s regler för integration heter *MOSART Federation Agreements* och finns tillgängligt för nedladdning på MOSART:s officiella hemsida [2].

4.1 Introduktion

MOSART:s regler för integration är en uppsättning regler som en MOSART-ansluten applikation (sk federat) måste följa för att på ett korrekt sätt kunna delta i en MOSART-simulering. Dessa regler är giltiga för alla simuleringar (sk federationsexekveringar) som görs i MOSART testbädd.

I reglerna ingår bl a följande:

- En uppsättning synkroniseringspunkter finns definierade för att hantera federationens exekveringsstatus (t ex ReadyToRun, ReadyToPause).
- En uppsättning scenariokontrollinteraktioner finns definierade för att hantera scenarioexekveringen (SetScenario, SetScenarioTime, Play, Pause, Reset, ShutdownFederation).
- Ett antal faser som federationsexekveringen kan befinna sig i (Init, Scenario setup, Running och Paused).
- Ett tillståndsdiagram som beskriver när en federat ska byta federationsexekveringsfas.
- En uppsättning interaktioner för att kunna fjärrstyra scenariot före och under exekvering (t ex RemoteCreateObject, RemoteChangeAttribute, RemoteDeleteObject, RemoteAcquireOwnership, RemoteInvokeMethod)
- En uppsättning krav för att en federat ska anses vara ”väluppfostrad”.
- Krav på den FOM som används. De synkroniseringspunkter och scenariokontrollinteraktioner som används i alla MOSART-federationer måste finnas definierade i den FOM som används. De finns definierade i FOM:en *MOSART FOM Extension*.

Den integrationsprogramvara som har utvecklats inom projektet, se kap 5, har tagits fram för att göra det så enkelt som möjligt att uppfylla dessa regler för integration.

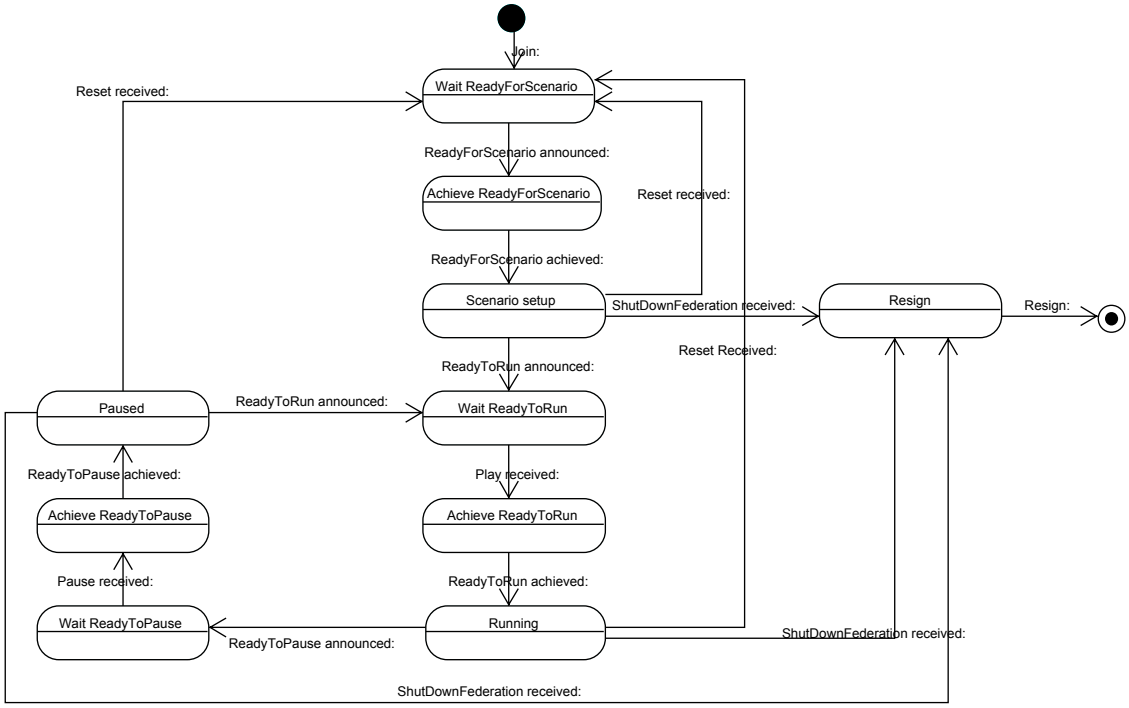
4.2 Federationssynkronisering

I en distribuerad simulering kan flera federater ansluta sig till en federation för att exekvera ett scenario tillsammans. För att få ett deterministiskt resultat i sina simuleringar är det viktigt att federationsexekveringen är synkroniserad så att alla federater tar emot scenarioinformation, stegar fram tid, pausar, byter till nyttt scenario, etc på ett kontrollerat sätt. För att uppnå detta används i MOSART ett antal synkroniseringspunkter och ett antal interaktioner.

En federationsexekvering har delats upp i fyra olika faser:

- **Init:** Federater ansluter sig till federationen
- **Scenario setup:** Scenarioinformation skickas ut och federaterna konfigurerar sig
- **Running:** Scenariot exekveras och simuleringstiden stegas fram
- **Paused:** Scenarioexekveringen är pausad

Det tillståndsdiagram som beskriver hur federater ska flytta sig mellan de olika faserna presenteras i figur 3.



Figur 3. Tillståndsdiagram som beskriver federationssynkronisering i MOSART.

4.3 Fjärrstyrd scenariohantering

I en distribuerad simulering med många federater, ofta fördelade på olika maskiner på olika fysiska platser, är det viktigt att kunna fjärrkontrollera de simulerade objekten. Exempelvis är det lämpligt att en federat som är specialiserad på att simulera en UAV kan skapa simulerade UAV-objekt efter instruktion från en central scenariohanteringsapplikation (en annan federat). Samma sak gäller självklart för många andra typer av objekt, exempelvis sensorer.

Tanken är att ett scenario ska kunna skapas på ett centralt ställe i en scenariohanteringsapplikation. När scenariot ska exekveras fördelar scenariohanteringsapplikationen de olika objekten i scenariot på de federater som finns tillgängliga (enligt instruktion från användaren) och skickar sedan ut instruktioner över HLA till respektive federat där den talar om vilka objekt respektive federat ska simulera och vilka initiala parametervärden den ska använda för respektive objekt. Det är även möjligt att skicka fjärrinstruktioner bl a för att ändra attributvärden i ett objekt eller ta bort ett objekt under simuleringens gång.

Följande interaktioner används för fjärrstyrning av scenariot och finns definierade i MOSART FOM Extension:

- RemoteCreateObject
- RemoteChangeAttribute
- RemoteDeleteObject
- RemoteAcquireOwnership
- RemoteInvokeMethod

Utöver dessa intekationer finns motsvarande interaktioner för att skicka resultatet av fjärrinstruktionen. Dessa är:

- RemoteCreateObjectResult
- RemoteChangeAttributeResult
- RemoteDeleteObjectResult
- RemoteAcquireOwnershipResult
- RemoteInvokeMethodResult

4.4 Krav på väluppfostrad federat

En federat måste uppfylla följande krav för att anses väluppfostrad:

(1) Den försöker inte skapa federationen

(2) Den känner inte till FOM/FED-*filen*

(3) Den hanterar fallet att RTI:n inte är startad när federaten försöker ansluta sig, exempelvis genom ett försök igen/timeout förfarande

(4) Den hanterar fallet att RTI:n är startad men federationen inte ännu är skapad när federaten försöker ansluta sig, exempelvis genom ett försök igen/timeout förfarande

(5) Det är möjligt att konfigurera RTI host och port som den ska koppla upp sig mot

(6) Det är möjligt att konfigurera namnet på federationen den ska koppla upp sig mot och namnet federaten ska ha när den ansluter sig till federationen

(1) och (2) hanteras enkelt genom att inte skriva någon kod som försöker skapa federationen eller läsa FOM/FED-filen.

(3) och (4) hanteras av skelettfederaten. Genom att använda skelettfederaten som utgångspunkt vid federatutveckling uppfyller man automatiskt även dessa två krav.

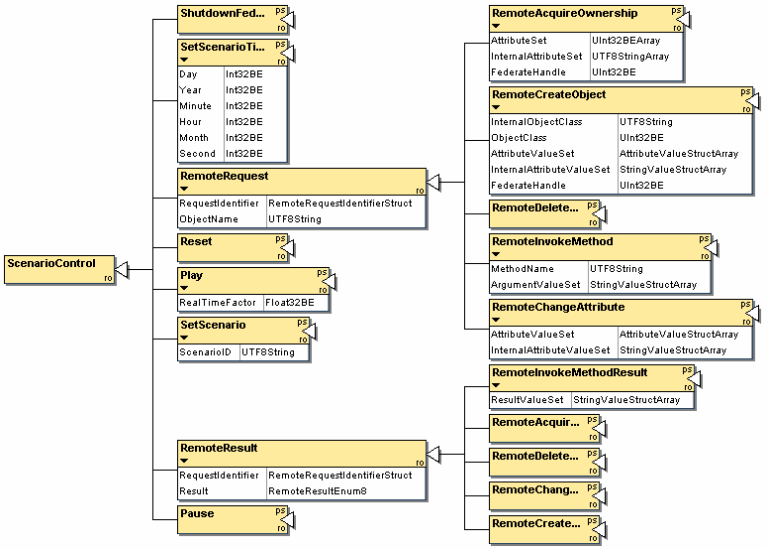
Exempel på hur man kan hantera (5) och (6) finns också i skelettfederaten.

Inläsande av FOM/FED-fil och skapande av federationen sköts i MOSART testbädd av Manager-federaten, se kap 6.2. Denna federat deltar i alla MOSART-simuleringar.

4.5 Krav på FOM

I MOSART kan godtycklig FOM användas. Det finns dock särskilt stöd i testbädden för FOM:ar som baseras på RPR-FOM v2 [3] via RPRFOM kit:et, se kap 5.3.

För att federations- och scenariohanteringen ska fungera krävs dessutom att de synkroniseringspunkter och scenariokontrollinteraktioner som reglerna för integration nyttjar finns med i FOM:en. Dessa finns definierade i FOM:en *MOSART FOM Extension* som enkelt kan stoppas in i den FOM som ska användas. En översikt av scenariokontrollinteraktionerna i *MOSART FOM Extension* finns i figur 4.



Figur 4. FOM-klassdiagram över MOSART:s scenariokontrollinteraktioner.

4.6 Framtida arbete

I dagsläget finns ingen egen fas i tillståndsdiagrammet för federationssynkronisering för hantering av fjärrinstruktioner vid scenario setup. Detta gör att en federat (normalt scenariohanteringsverktyget) inte kan tala om när alla fjärrinstruktioner som ska skickas innan scenariot ska börja exekvera är färdigprocessade. En sådan fas bör läggas till i tillståndsdiagrammet i nästa version av MOSART:s regler för integration.

En annan viktig sak som bör läggas till stöd för i nästa version av reglerna för integration i MOSART testbädd är stöd för sent anslutande federater. I dagsläget stöds inte att en federat kan ansluta sig till en federation efter det att första scenariot i federationsexekveringen har börjat sättas upp. Detta gör att alla federater måste koppla ner sig och sedan ansluta sig igen om man glömmer starta en federat när man ska göra en simulering. Genom att förbättra MOSART:s regler för integration skulle detta kunna hanteras på ett smidigare sätt.

5 Integrationsprogramvara

En av de viktigaste delarna av den tekniska utveckling som skett inom projektet är den integrationsprogramvara som utvecklats. Denna programvara utgörs av ett antal mjukvarubibliotek, även kallade ”kits”, som syftar till att göra det så enkelt som någonsin möjligt för en utvecklare att integrera en programvara (t ex en sensormodell, en signalbehandlingsalgoritm av något slag, en fusionsmodul, ett visualiseringsverktyg eller ett beslutsstöd) i MOSART testbädd. Utöver kit:en finns även en exempelapplikation, även kallad skelettfederat, som visar hur kit:en kan användas för att bygga en komplett MOSART-ansluten applikation. Möjligheten att utgå från skelettfederaten när en ny applikation ska anslutas till testbädden underlättar integrationsarbetet avsevärt.

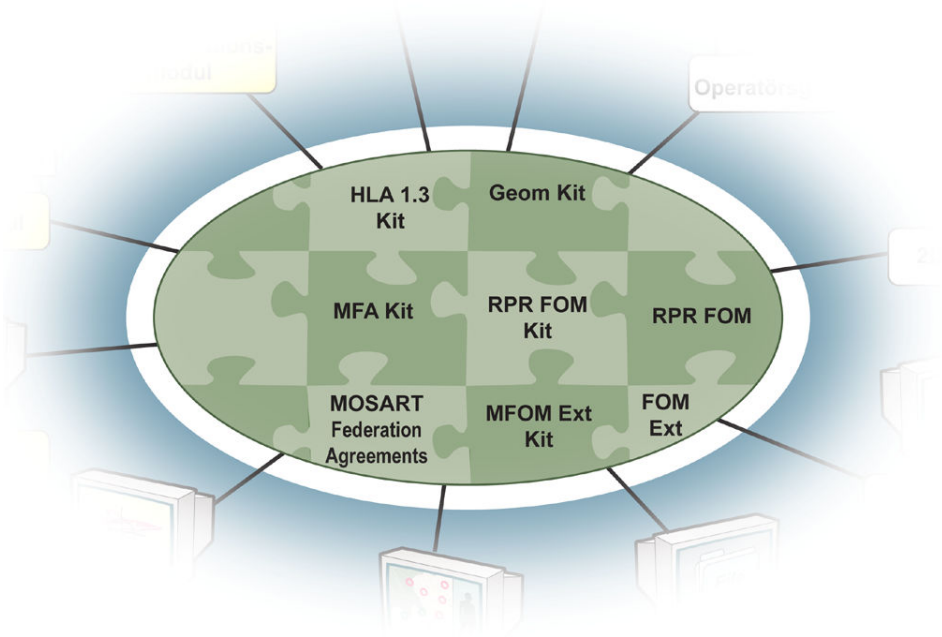
5.1 Översikt av utvecklad integrationsprogramvara

Den integrationsprogramvara som tagits fram är:

- HLA Kit, mjukvara för att hantera basfunktioner i HLA på ett för användaren automatiskt sätt. Detta innebär till exempel att mjukvaran automatiskt uppdaterar attributvärden i programspråksobjekt via HLA, kodning och avkoding av datatyper och trådad hantering av callbacks. Finns för Java och C++.
- RPRFOM Kit, mjukvara som stödjer användandet av RPR-FOM vilken är en standard-FOM som utnyttjas i stor utsträckning för att öka kompatibiliteten mellan olika federationer. Detta kit hanterar bl a prediktion av spatiala data (dödräkning) i RPR-FOMen, nyttjar Geom kit:et för automatisk koordinatsystemskonvertering, etc. Finns för Java och C++.
- MFOMExt Kit, mjukvara som stödjer användandet av MOSART FOM Extension vilket är den FOM som definierar de synkroniseringspunkter och interaktioner som krävs för MOSART:s federations- och scenariosynkronisering. Finns för Java och C++.
- MFA (MOSART Federation Agreements) Kit, mjukvara som implementerar klientsidan av MOSART Federation Agreements och därmed hanterar synkronisering av federationer (t ex start och stopp av simulering), scenariohantering och batchsimulering. Används tillsammans med modulen *FederationManager* som implementerar serversidan av MOSART Federation Agreements. Finns för Java och C++.
- Geom Kit, mjukvara för transformation av koordinater mellan olika koordinatsystem. Detta kit har ingen koppling till HLA, men är något som de flesta federater behöver använda för att kunna fungera i en federation. Finns för Java och C++.
- FedApp Kit, mjukvara för att bygga en federatapplikation på. Innehåller bl a en klass som är en abstraktion av en federat som kan ärvas (specialiseras) så att den implementerar önskat beteende. Finns för Java och C++.
- SkeletonFederate, modul som är ett mjukvaruskal som använder sig av alla de olika integrationsprogramvarorna (kit:en) som finns i MOSART. Används som skal när nya applikationer ska anslutas till testbädden. Finns för Java och C++.

All integrationsprogramvara har dokumenterats, både som enskilda dokument där övergripande designen av respektive kit beskrivs och via API-dokumentation som genererats automatiskt från kommentarer i koden via JavaDoc och DoxyGen.

I figur 5 återfinns en schematisk beskrivning av MOSART integrationsprogramvara som ett lager som federater använder för att ansluta sig till testbädden.



Figur 5. Schematisk beskrivning av MOSART integrationsprogramvara.

5.2 HLA 1.3 Kit

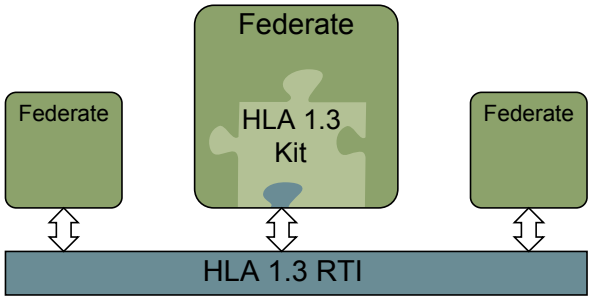
Mjukvarubiblioteket HLA 1.3 kit för C++ och Java innehåller stöd för att snabbt kunna utveckla en HLA-federat. Trots att det standardiserade HLA-API:et har hållits rent och enkelt så är det ett relativt stort steg att ta fram en HLA-federat. Målet med HLA 1.3 kit har varit att minska detta steg, samtidigt som flera av funktionerna i HLA stöds automatiskt i kit:et. Funktionen *provide attribute value update* är ett sådant exempel där federat-utvecklaren inte ens behöver veta att funktionen finns – den hanteras ändå. Detta medför att federaterna snabbare får högre kvalitet och att det är lättare att ta fram en HLA-federat utan att vara HLA-expert. En översiktlig beskrivning av en federation där en federat använder HLA 1.3 kit finns i figur 6.

HLA 1.3 kit tillhandahåller ett lager som gör federaten helt RTI-oberoende, vilket innebär att ett byte av RTI-implementation (från t ex Mäk RTI till Pitch pRTI) inte kräver att en enda rad kod behöver skrivas om, och på Java-sidan behöver applikationen inte ens startas om. Den största förenklingen är dock kopplingen mellan HLA-objekt och objekt i programspråken C++ resp Java. Ett attribut i ett HLA-objekt konverteras automatiskt till ett attribut i t ex ett Java-objekt. Utvecklaren behöver inte hålla reda på vilket sk handle som attributet, objektet eller objektklassen har för att uppdatera objektet. Konvertering av datastrukturer i programspråken till och från bytedata som används som transportformat i HLA har också underlättats ordentligt. I beskrivningen av resp objekt-klass specificeras vilken datatyp ett attribut har samt registrerar ev en sk codec för datatypen. Därefter tar HLA 1.3 kit hand om all konvertering till och från HLA-världen. Nedan listas några av de viktigaste funktionerna i HLA 1.3 kit:

- RTI-implementationsoberoende lager
- HLA-objekt är objekt i Java/C++ (handles döljs, reflection hanteras automatiskt, etc)
- HLA-interaktioner är objekt i Java/C++ (handles döljs, etc)
- Automatisk kodning/avkodning av datatyper (attribut i objekt och parametrar i interaktioner)
- Synkroniseringspunkter
- Tidssynkronisering (blockerande anrop till *time advance*, automatisk tidstämpling i *user supplied tag*)

I dagsläget finns endast ett mjukvarubibliotek för HLA 1.3. I nära framtid kommer en övergång till HLA 1516 att ske, dock med bibehållen bakåtkompatibilitet med HLA 1.3 och olika RTI:er. I samband med övergången till HLA 1516 kommer även designen att förändras en del. Framförallt sker designförändringar för att förbättra prestanda, men även för att uppnå ökad flexibilitet i kopplingen till olika FOM:ar. Ett byte av FOM är i dagsläget ett krångligt arbete som ofta leder till att flera versioner av federaten behöver underhållas. Tanken är att ett FOM-byte ska ske genom att man lägger till en FOM-mappning i sitt bibliotek och att federaten därefter inte behöver förändras för att växla mellan olika FOM:ar. Detta stöd behöver givetvis hantera även saker som ligger utanför FOM:en, bl a tidshantering och scenariohantering.

En mer fullständig och tekniskt djuplodande beskrivning av HLA 1.3 kit:et finns i dokumentet *HLA 1.3 Kit Documentation* som finns tillgängligt för nedladdning på MOSART:s officiella hemsida [2].



Figur 6. Översiktlig beskrivning av en federation där en federat använder HLA 1.3 Kit.

5.3 RPRFOM Kit

Mjukvarubiblioteket RPRFOM kit för C++ och Java innehåller stöd för att enkelt kunna arbeta med RPR-FOM:en [3]. RPR-FOM:en är en standardiserad FOM vars användande är mycket stort såväl nationellt som internationellt. Den innehåller definitioner av olika typer av plattformar och sensorer, mm. RPR-FOM:en utgör ett utökningsbart ramverk som ökar kompatibiliteten genom att skapa standardiserade beskrivningar av vanligt förekommande entitetstyper, mm. RPR-FOM:en kan utökas på många sätt och ändå bibehålla RPR-FOM-kompatibilitet [4].

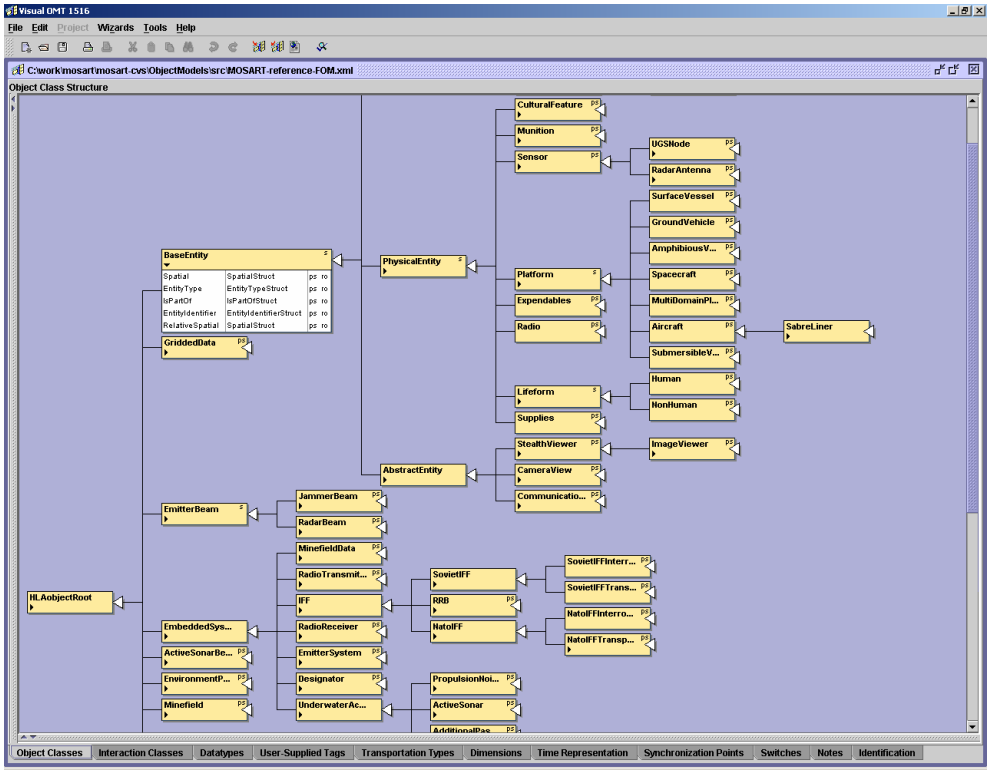
RPR-FOM:en innehåller ett mycket stort antal HLA objekt- och interaktionsklasser samt datastrukturer som används för att beskriva objektklassernas attribut och interaktionsklassernas parametrar. RPRFOM kit:et innehåller implementationer av många av de mest använda objekt- och interaktionsklasserna (dock inte på långa vägar alla). Fler objekt- och interaktionsklasser med tillhörande datastrukturer kan enkelt läggas till efterhand som dessa behövs.

Nedan listas några av de viktigaste funktionerna i RPRFOM kit:

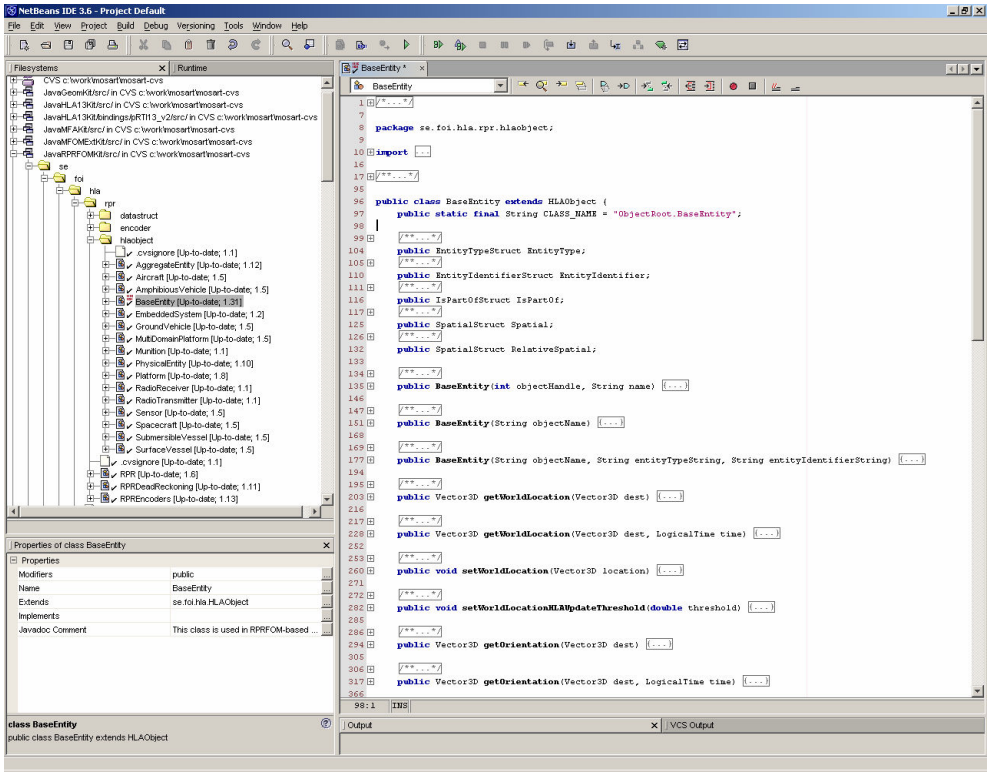
- Innehåller implementationer av många viktiga RPR-FOM objekt och interaktioner inklusive kodning och avkodning av attribut och parametrar.
- Dödräkning av spatial information i egna federatens (publicerade) objekt och andra federaters (abonnerade) objekt genomförs helt transparent för användaren.
- Uppdateringströsklar för när information ska skickas över HLA hanteras.
- Automatiskt val av dödräkningsalgoritm beroende på vilka spatiala parametrar (position, orientering, hastighet, acceleration och vinkelhastighet) som är skilda från noll.
- Automatisk transformation av position, orientering, hastighet och acceleration i alla objekt (publicerade och abonnerade) mellan en federats interna koordinatsystem (kan vara godtyckligt geocentriskt, geodetiskt eller projicerat koordinatsystem) och världskoordinatsystemet som används av RPR-FOM (geocentriskt XYZ på ellipsoiden WGS84). För dessa transformationer används Geom kit, se kap 5.6.

I figur 7 visas en del av MOSART referens-FOM, som bygger på RPR-FOM:en. Objektklassen *BaseEntity* är expanderad så man kan se dess attribut. Jämför med figur 8 som visar publika medlemsvariabler och (en delmängd av) de publika funktionshuvudena i Java-klassen *BaseEntity*, dvs motsvarigheten i Java RPRFOM kit.

En mer fullständig och tekniskt djuplodande beskrivning av RPRFOM kit:et finns i dokumentet *RPRFOM Kit Documentation* som finns tillgängligt för nedladdning på MOSART:s officiella hemsida [2].



Figur 7. FOM-klassen BaseEntity i FOM-editeringsverktyget Visual OMT från Pitch.



Figur 8. Java-klassen BaseEntity i den fria utvecklingsmiljön NetBeans.

5.4 MFOMExt Kit

Mjukvarubiblioteket MFOMExt kit för C++ och Java innehåller stöd för att enkelt kunna arbeta med de utökningar till RPR-FOM:en som gjorts i MOSART.

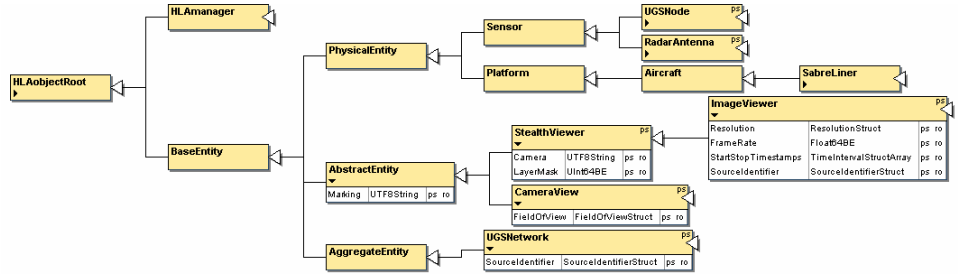
MFOMExt, eller *MOSART FOM Extension* som den egentligen heter, är en FOM som innehåller två delar:

- De scenariokontrollinteraktioner som behövs för MOSART:s regler för integration.
- Utökningar av RPR-FOM:en med nya objekt- och interaktionsklasser för att hantera de typer av entiteter och meddelanden som används i de system som ansluts till MOSART.

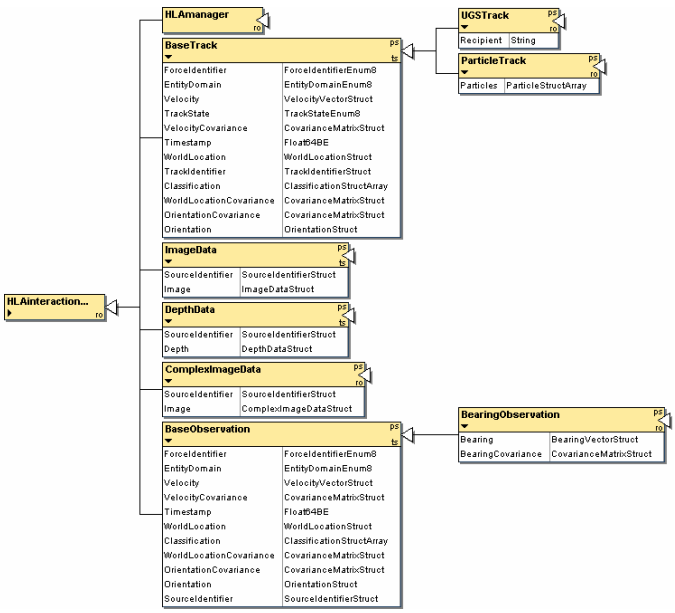
MFOMExt kit:et innehåller implementationer av alla objekt- och interaktionsklasserna i MFOMExt i Java respektive C++.

Implementationerna av scenariokontrollinteraktionerna används av MFA kit:et, se kap 5.5. Även implementationer av fjärrstyrningsinteraktionerna (RemoteCreateObject, etc) finns här.

Implementationerna av de nya objekt- och interaktionsklasserna som idag finns ger bl a stöd för abstrakta entiteter (t ex kameravyer via objektklassen CameraView) samt skickande av sensordata och målobservationer/målspår mellan federater. Delar av MFOMExt objektklasshierarki visas i figur 9 och delar av interaktionsklasshierarkin visas i figur 10.



Figur 9. Delar av (inte komplett!) objektklasshierarkin i MOSART FOM Extension.



Figur 10. Delar av (inte komplett!) interaktionsklasshierarkin i MOSART FOM Extension.

5.5 MFA Kit

När man genomför distribuerad simulering körs många olika applikationer (federater) på olika datorer i ett nätverk. För att få deterministiska resultat i sina distribuerade simuleringar behövs synkronisering av federationen, dvs av alla federaterna. MOSART:s regler för integration, se kap 4, beskriver hur federations- och scenariosynkronisering går till i MOSART testbädd.

MFA kit:et, eller *MOSART Federation Agreements Kit*, stödjer federatutvecklaren i arbetet med att ansluta sin applikation till MOSART genom att implementera alla krångliga detaljer i dessa federationsöverenskommelser (regler för integration) och exponerar ett lättanvänt och lättförståeligt gränssnitt mot federatutvecklaren. MFA kit:et kan sägas implementera klientdelen av reglerna för integration, dvs den del som varje federat ska implementera.

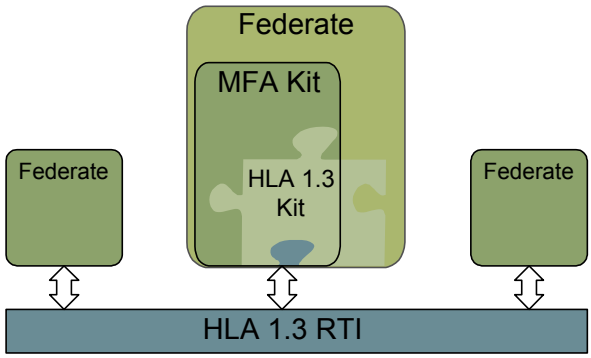
Den andra delen av reglerna för integration, den sk serverdelen, implementeras av specialfederaten MOSART Federation Manager, se kap 6.2. Det är Managern som registrerar synkroniseringspunkterna och skickar ut de flesta scenariokontrollinteraktionerna. Managern deltar därför med nödvändighet i alla MOSART-simuleringar.

Några av de viktigaste funktionerna i MFA kit:et är:

- Hantering av synkroniseringspunkter för federationssynkronisering (ready for scenario, ready to run, ready to pause)
- Hantering av scenariouppsättning (scenario-ID och scenariostarttid)
- Hantering av scenarioexekvering (play, pause, reset, shutdown federation)

I figur 11 visas en översiktlig beskrivning av en federation med en federat som använder MFA kit:et.

En mer fullständig och tekniskt djuplodande beskrivning av MFA kit:et finns i dokumentet *MFA Kit Documentation* som finns tillgängligt för nedladdning på MOSART:s officiella hemsida [2].



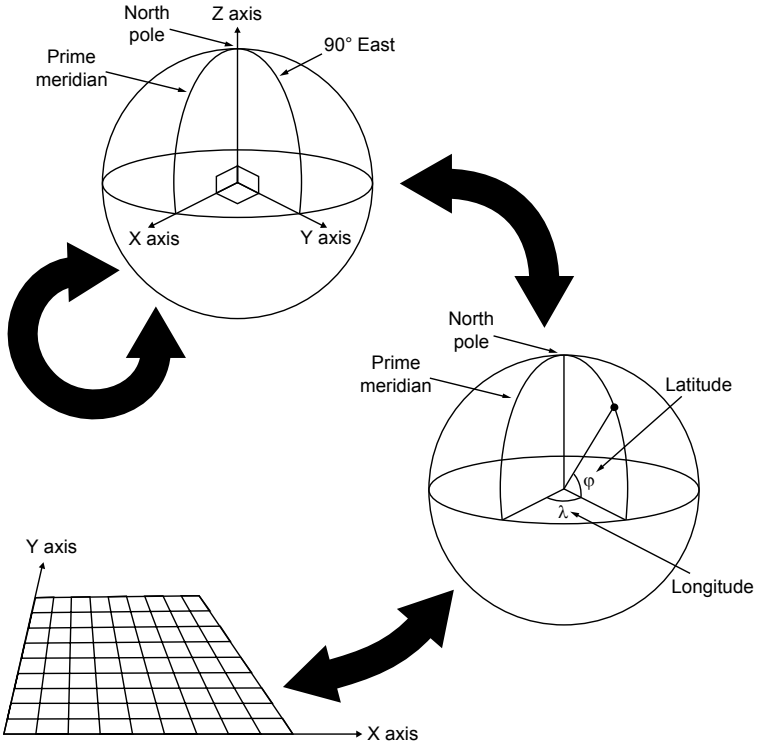
Figur 11. Översiktlig beskrivning av en federation med en federat som använder MFA kit:et.

5.6 Geom Kit

Geom kit för Java och C++ är ett mjukvarubibliotek som hanterar koordinater, koordinatsystem och transformationer i 2D och 3D. De grundläggande koordinatsystemen som hanteras är geocentriskt (kartesiskt koordinatsystem med origo i jordens mitt), geodetiskt (latitud, longitud och höjd) samt valfritt kartesiskt koordinatsystem. Geom kit har stöd för projektioner som bygger på metoden Transverse Mercator, t ex RT90 och UTM. API:et är förberett för andra projektionsmetoder såsom bl a Oblique Mercator, Lambert Conical och Cassini Soldner.

Projektioner och ellipsoider i Geom kit är helt parametriseringsbara vilket medför att t ex projektionen RT90 5 gon V på ellipsoiden Bessel 1841 helt enkelt är en parametersättning av Transverse Mercator och en ellipsoid. Projektioner och ellipsoider kan sättas ihop för att erhålla en transformation mellan två olika koordinatsystem. Figur 12 visar i stort vilka koordinatsystem som stöds samt hur man går mellan dem. För att t ex konvertera från projicerade koordinater i RT90 med ellipsoiden Bessel 1841 till geodetiska koordinater i ellipsoiden WGS84 så måste man först gå från projicerade RT90 till geodetiska, sedan till geocentriska i Bessel 1841, därefter till geocentriska i WGS84 och slutligen till geodetiska i WGS84. Hela denna kedja sätts i Geom kit ihop till ett transformationsobjekt så att man själv slipper hålla reda på detta hela tiden.

En mer fullständig och tekniskt djuplodande beskrivning av Geom kit:et finns i dokumentet *Geom Kit Documentation* som finns tillgängligt för nedladdning på MOSART:s officiella hemsida [2].



Figur 12. Översikt av koordinatsystemen som stöds av Geom kit samt dess övergångar.

5.7 FedApp Kit

FedApp (Federate Application) kit för Java och C++ är ett mjukvarubibliotek som i huvudsak innehåller en abstrakt klass som representerar en federat. Genom att ärva från denna abstrakta klass och implementera de abstrakta metoder den har kan en federatutvecklare på ett mycket smidigt sätt bygga en MOSART-ansluten applikation som uppfyller alla regler för integration, se kap 4.

FedApp kit:et innehåller även ett en konfigurationsklass där konfiguration av federaten kan göras (RtiHost, RtiPort, ConnectionRetries, ConnectionRetryDelay, FederationExecutionName, FederateType, etc). Konfigurationsklassen har stöd för att läsa konfigurationsdata från miljövariabler, properties-filer samt via kommandoradsargument. Konfigurationsklassen kan med fördel utökas med modellspecifika konfigurationsvariabler för den modell eller de modeller som applikationen implementerar.

I princip fungerar den abstrakta federatklassen *FederateBaseApplication* som så att man skapar en specialiserad version av *FederateBaseApplication* genom att skapa en klass som ärver från denna. I denna federatspecifika klass implementerar man de abstrakta metoderna:

- **onPublishAndSubscribe():** Anropas när federaten ska sätta upp sina publikationer och abonnemang.
- **onScenarioSetup(ScenarioInfo scenarioInfo):** Anropas när federaten ska skapa de objekt den ska simulera i denna scenarioexekvering och registrera dessa i HLA. Argumentet *scenarioInfo* innehåller information om scenario-ID och scenariostarttid för det scenario som ska exekveras.
- **onScenarioCleanup():** Anropas när federaten ska städa upp sitt interna tillstånd efter att ett scenario har kört färdigt och bl a avregistrera sina objekt från HLA.
- **onPreTimeAdvance(LogicalTime time):** Anropas före det att tiden i simuleringen ska stegas fram till den tid som anges av argumentet *time*.
- **onAfterTimeAdvance(LogicalTime time):** Anropas efter det att tiden i simuleringen ska stegas fram till den tid som anges av argumentet *time*.

Därefter skapar man ett konfigurationsobjekt och sätter lämpliga värden på parametrarna i detta (eller snarare låter användaren av applikationen sätta dessa värden, t ex via ett GUI). När detta är klart skapar man en instans av den specialiserade federatklassen och anropar connect()-metoden på detta objekt. Som argument till connect()-metoden skickas konfigurationsobjektet som innehåller information om RTI host och port, federationsnamn, etc. Simsalabim och federaten är uppkopplad! För att köra igång simuleringen anropar man sedan run()-metoden i federatklassen och är de abstrakta metoderna som listas ovan korrekt implementerade har federaten dessutom förhoppningsvis det förväntade beteendet. Det finns även en disconnect()-metod som man anropar när federaten vill koppla ner sig från federationen, dvs när run()-metoden returnerat.

5.8 Skelettfederat

SkeletonFederate, eller skelettfederaten, för Java och C++ är en komplett MOSART-ansluten applikation som använder alla de kits som utvecklats inom MOSART för att underlätta anslutning av applikationer till MOSART testbädd.

Skelettfederaten använder följande kit:

- HLA 1.3 Kit
- RPRFOM Kit
- MFOMExt Kit
- MFA Kit
- Geom Kit
- FedApp Kit

Skelettfederaten implementerar en enkel modell för att flytta runt ett eller flera markfordon (GroundVehicle i RPR-FOM:en) i ett 2-dimensionellt plan mellan slumpmässigt valda punkter inom ett specificerat område. Koordinatmässigt sker detta på en ca 400 x 400 m stor yta på ett av fälten vid markstridsskolan i Kvarn.

Skelettfederaten publicerar så många markfordon som anges vid programstart via kommandoraden eller i konfigurationsfil. Dessutom abonnerar den på markfordon och skriver ut information om dessa efterhand som simuleringen fortlöper. Man kan således starta flera skelettfederater och se hur de skriver ut information om sina egna (publicerade) och andra federaters (abonnerade) markfordonsobjekt.

Federaten arbetar internt i det projicerade koordinatsystemet RT90, men med hjälp av RPRFOM och Geom kit:en publiceras detta korrekt enligt RPR-FOM:en i geocentriskt XYZ (WGS84) mot HLA.

Skelettfederaten utgör ett bra exempel på hur kit:en som utvecklats i MOSART kan användas för att skapa en komplett MOSART-ansluten applikation. Den är således en mycket bra utgångspunkt för att bygga nya MOSART-anslutna applikationer och kan med fördel användas när en ny modell ska kopplas in i testbädden. Den är dessutom bra på så vis att den på ett pedagogiskt vis kan visa en utvecklare som tidigare inte arbetat med MOSART testbädd hur en enkel applikation ansluten till testbädden kan se ut.

5.9 Programspråkskopplingar

MOSART testbädd stödjer för närvarande utveckling av federater på tre olika operativsystem i tre olika programmeringsspråk.

De operativsystem som stöds är:

- Windows
- Linux
- Solaris

De programmeringsspråk som stöds är:

- Java
- C++
- MATLAB

Godtycklig kombination av operativsystem och programmeringsspråk enligt ovan fungerar.

5.9.1 Java

Alla kit (mjukvarubibliotek) finns implementerade i Java. Det går således utmärkt att utveckla federater i detta programmeringsspråk på godtycklig plattform där Java finns.

Ett Ant-baserat byggsystem finns inklusive projektfiler för utvecklingsverktyget NetBeans. Det går således bra att utveckla utan IDE (och använda Ant-byggfilerna direkt från kommandoraden) eller med NetBeans IDE. Det bör även vara mycket enkelt att använda en annan IDE om så skulle vara önskvärt.

5.9.2 C++

Alla kit finns även implementerade i C++. Av samma skäl som ovan går det således utmärkt att utveckla federater även i C++. Stöd finns för att bygga federater i C++ på operativsystemen Windows, Linux och Solaris.

För närvarande finns projektfiler för Visual Studio 6 och Visual Studio 7 i Windows och ett automake-baserat byggsystem för UNIX-system (Linux/Solaris).

5.9.3 MATLAB

En MATLAB-koppling som bygger på Java-kit:en finns också tillgänglig. Den består av ett antal Java-klasser, bl a en som representerar en federat. Genom att i MATLAB anropa metoder i denna Java-klass för att konfigurera federaten kan man sedan ansluta till en federation. På federatobjektet kan sedan metoder anropas för att sätta upp publikationer och abonnemang, stega fram simuleringstiden, etc.

Utöver detta finns wrapper-klasser för de viktigaste FOM objekt- och interaktionsklasserna i RPRFOM och MFOM Ext kit:en. Dessa wrapperklasser gör att man från MATLAB kan anropa metoder för att sätta och hämta information ur objekt- och interaktionsklasserna och få hantera data i form av MATLAB-vänliga double-arrayer, etc istället för Java-datastrukturer. På samma sätt som i Java och C++ kan man sedan från MATLAB skapa HLA-objekt, registrera dem i HLA, uppdatera dess attributvärden, få attributuppdateringar på abonnerade objekt, etc.

En exempelfederat i MATLAB-kod finns tillgänglig som implementerar en enkel akustisk sensornod som ger bäringar till markfordon (som den prenumererar på via HLA).

6 Grundfunktionalitet

Utöver integrationsprogramvaran har det inom MOSART-projektet utvecklats ett antal generella verktyg som många projekt kan ha nytta av i framtiden genom att de slipper utveckla denna funktionalitet själva. Dessa moduler utgör ett grundläggande stöd för simulering.

De generella verktygen består av applikationer för federationshantering, editering och exekvering av scenarion, 2D-visualisering, 3D-visualisering samt ett verktyg för logging och återuppspelning av scenarioexekveringar.

Utöver detta finns en koppling till system för användarinteraktion utvecklade på FOIs institution för Människa-System-Interaktion (MSI).

6.1 Översikt av utvecklad grundfunktionalitet

De moduler för grundläggande stöd för simulering som idag finns MOSART-anslutna är:

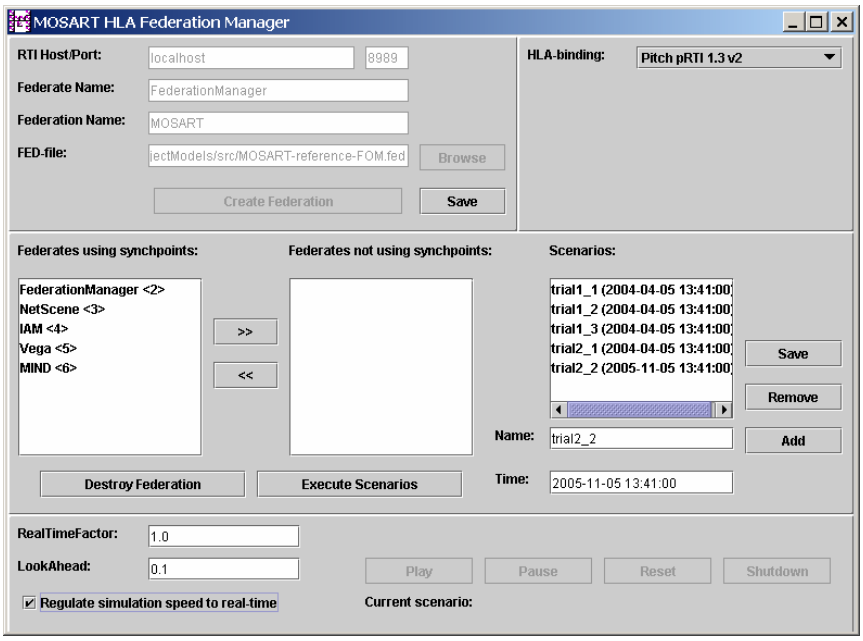
- **MOSART Federation Manager:** Inom MOSART-projektet utvecklad modul som tillsammans med MFA kit:et implementerar MOSART:s regler för integration (*MOSART Federation Agreements*) och därmed bland annat hanterar synkronisering av federationer (t ex start och stopp av simulering), scenariohantering och batchsimulering. Denna modul måste alltid finnas med när man kör MOSART-simuleringar eftersom den implementerar serversidan av *MOSART Federation Agreements*. Det är via denna applikation man styr simuleringens exekvering.
- **FLAMES:** Kommersiellt simuleringsramverk utvecklat av Ternion [5] som kan användas för att utveckla modeller som kan simulera både fysiska och kognitiva beteenden för komplexa entiteter.
- **NetScene:** Inom FOI utvecklad scenarioeditor och en enkel scenariomotor speciellt framtagen för distribuerade simuleringar. Fokus ligger på att *hela* scenariot kan editeras i ett enda verktyg. Editeringen kan ske dels i 2D med karta som bakgrund och dels i 3D. Alla objekt kan parametersättas och konfigureras för att drivas av NetScene eller av någon annan distribuerad simuleringskomponent.
- **MIND:** Inom FOI utvecklad modul för 2D-visualisering som kan visualisera godtyckliga objekt från HLA-världen genom plug-ins som beskriver hur objekten ska visualiseras. I denna modul utnyttja bl a MIND:s välutvecklade karthantering för visualisering av ikoner, text, linjer, areor, etc på karta.
- **VEGA:** Kommersiellt verktyg för 3D-visualisering som kan visualisera godtyckliga objekt från HLA-världen genom plug-ins som beskriver hur objekten ska visualiseras. I denna modul används högupplösta omvärldsmodeller för visualisering av terrängen.
- **MOSART/EW Logger:** Inom FOI utvecklat verktyg för att lagra, analysera och återuppspela en HLA-simulering. Det går att köra loggern som konsolapplikation, Windowsapplikation eller integrerat i en HLA-federat.
- **HiFi Engine:** På FOI-institutionen för MSI utvecklad simuleringsmotor som gör att olika typer av moduler för användarinteraktion kan nyttjas i MOSART-simuleringar, se kap 7.

6.2 MOSART Federation Manager

Managerfederaten måste alltid finnas med i alla MOSART-simuleringar eftersom det är den som säkerställer att federationen hela tiden är synkroniserad. Den utgör serverdelen av reglerna för integration i MOSART testbädd (*MOSART Federation Agreements*, se kap 4). Detta innebär att den skickar ut federationssynkroniseringsinformation som hanteras av övriga federater.

Managern skickar även viss scenariokontrollinformation, nämligen scenario-ID och scenariostarttid. I Managern kan man specificera godtyckligt antal scenario-ID:n (och motsvarande scenariostarttider) och dessa kan sedan exekveras ett i taget eller som en batchkörning om så önskas. Mer detaljerad scenariohantering (utöver scenario-ID och scenariostarttid), t ex vilka entiteter som ska delta i scenariot och hur dessa ska uppträda (parametersättning av modeller) hanteras inte av Managern, utan av lämpligt scenarioverktyg, t ex NetScene (se kap 6.3.2).

En skärmdump från Managern finns i figur 13.



Figur 13. Skärmdump från Managerfederaten.

Nedan listas de viktigaste funktionerna i Managern:

- Gör det enkelt att skapa federationer, t ex specificering av federationsnamn, FOM/FED-fil, etc. Managern behöver ej startas om för att avsluta en federation och skapa en ny.
- Enkel specificering av federationskonfiguration, t ex värddator för RTI, federationsnamn och val av HLA-bindning (Pitch pRTI, DMSO RTI-Ng, MÄK RTI), etc.
- Listar federater som kopplat upp sig mot federationen. Från listan kan användaren tala om för Managern att en viss federat inte stödjer synkroniseringspunkter. Detta är bra för legacyfederater som saknar support för synkroniseringspunkter och ökar på så vis kompatibiliteten mot federater som normalt körs i andra federationer.

- Scenario-ID:n och motsvarande scenariostartpunkter kan specificeras. Managern ser till att alla scenarion exekveras och skickar ut scenario-ID och scenariostartpunkt för varje nytt scenario som ska exekveras.
- Registrerar synkroniseringspunkter och skickar federations- och scenariokontrollinteraktioner i enlighet med reglerna för integration (*MOSART Federation Agreements*, se kap 4) och ser till att federationen hela tiden är synkroniserad. Övriga federater måste hantera informationen från Managern i enlighet med reglerna för integration.
- Enkelt att styra exekveringen av ett scenario (*Play* / *Pause* / *Reset* / *Shutdown*)
- Managern kan anpassa scenarioexekveringshastigheten till realtid (eller godtycklig tidsfaktor gånger realtid) i de fall då vissa federater annars simulerar snabbare än realtid. Detta sker genom att Managern ser till att tidsstegning via RTI:ns tidstjänster inte går fortare än realtid. Inte minst när moduler med människan i loopen är inkopplade är det av största vikt att synkronisera simuleringshastigheten med realtid.

6.3 Scenarioverktyg

I icke-distribuerade simuleringsverktyg finns ofta ett centralt scenarioverktyg för redigering av scenariot innan simuleringen görs. Ibland är scenarioverktyget integrerat i simuleringsmotorn så att scenariot kan påverkas under simuleringens gång. I distribuerade simuleringar finns det inte *en* simuleringsmotor utan det kan finnas *flera* distribuerade simuleringskomponenter som driver olika delar av scenariot. Dessutom kan dessa simuleringskomponenter vara utvecklade av olika leverantörer och ofta ursprungligen skrivna för ett annat sammanhang för att sedan återanvändas och ev anpassas för att passa en viss federation av simuleringskomponenter. Detta ställer nya krav på ett scenarioverktyg och ingående simuleringskomponenter om man vill ha *ett* scenarioverktyg för *hela* den distribuerade simuleringen. NetScene är ett sådant scenarioverktyg (se kap 6.3.2).

I scenarion med få ingående simuleringskomponenter (federater) som behöver konfigureras inför varje scenario kan man ibland klara sig utan ett centralt scenarioverktyg. Ett exempel kan vara när större delen av scenariot drivs av en enda simuleringskomponent (t ex simulering av alla aktörer) och övriga komponenter består av simulering av sensorer, kommunikationssystem med komplicerade inställningar. Detta måste dock ses som en övergångsfas när man vill hantera ett arv av monolitiska simuleringsmotorer. FLAMES är ett sådant verktyg (se kap 6.3.1).

6.3.1 FLAMES

FLAMES (Flexible Analysis and Mission Effectiveness System) är ett simuleringsramverk som är utvecklat av Ternion [5]. Simuleringsramverket kan användas för att utveckla modeller som kan simulera både fysiska och kognitiva beteenden för komplexa entiteter. Verktyget avsågs ursprungligen för simuleringar inom luft-domänen (flyget) men kan även användas för både mark- och undervattensscenarion. FLAMES är uppdelat på ett antal verktyg, FORGE för offline scenarioeditering, FIRE för simuleringsmotor, FLASH för uppspelning av en gjord simulering (även online) samt FLARE för efteranalys.

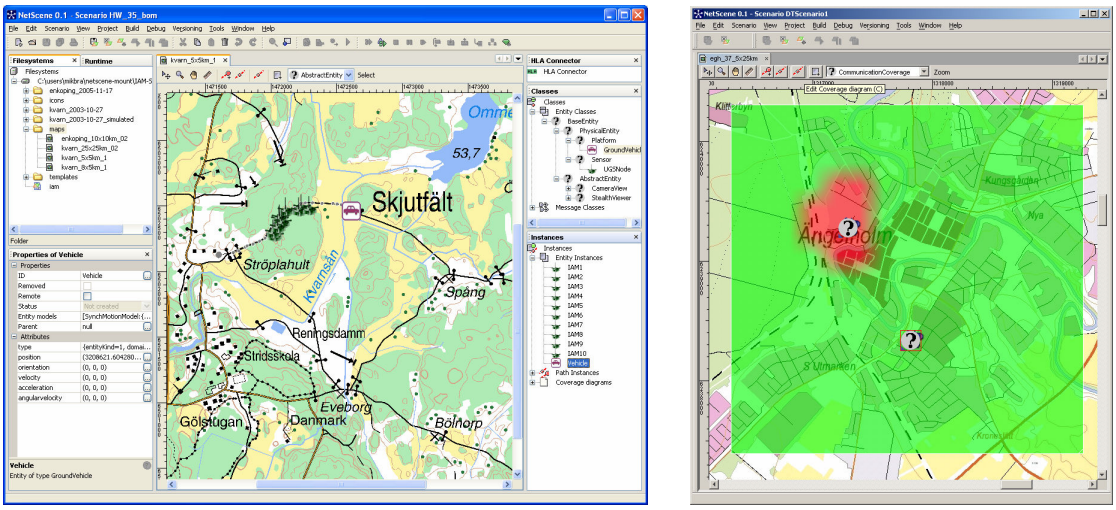
Under år 2003 gjordes en integration av FLAMES med MOSART mha FLAMES tilläggsmodul för HLA [6]. Objekt i FLAMES kunde utbytas med HLA-federationen och tvärtom. Stöd för nya typer av objekt hanteras genom att en sk HLA-manager skrivs. I den version av HLA-modul som användes i det här försöket saknades stöd för tidssynkronisering och synkroniseringspunkter vilket begränsar användningen en del. Det är möjligt att stöd för denna funktionalitet finns i FLAMES HLA-modul idag.

6.3.2 NetScene

NetScene är en scenarioeditor och en enkel scenariomotor för distribuerade simuleringar. Fokus ligger på att *hela* scenariot kan editeras i ett enda verktyg. Editeringen kan ske dels i 2D med karta som bakgrund och dels i 3D. Alla objekt kan parametersättas och konfigureras för att drivas av NetScene eller av någon annan distribuerad simuleringskomponent. Scenarioinformation distribueras sedan till övriga simuleringskomponenter (federater) som sköter själva simuleringen av resp objekt/företeelse. Detta medger en enorm flexibilitet där en viss simuleringskomponent kan bytas ut mot en mer lämplig sådan utan att scenarioverktyget behöver förändras. Viss typ av simulering kan byggas in i NetScene, bl a styrning av olika objekt (t ex plattformar) längs förspecificerade banor med eller utan terrängföljning. Andra enklare attributförändringar kan också med fördel skriptas i NetScene. Tanken är dock att alla avancerade modeller simuleras av andra simuleringskomponenter för att bibehålla den flexibilitet och skalbarhet som distribuerade simuleringar medger.

NetScene är ett FOI-utvecklat verktyg och bygger på NetBeans-platform 3.6, vilket är en delmängd av NetBeans IDE som är en utvecklingsmiljö för Java. Scenariohantering, kartfunktioner, HLA-koppling är implementerade som separata NetBeans-moduler. Genom att

använda NetBeans-plattformen som grund så kan vi återanvända en stor del av fönsterhantering (dockning av fönster mm), plugin-systemet, konfigurationshantering, mm. I skrivande stund pågår en uppgradering av NetScene version 1.0 för att anpassas till NetBeans platform 5.0 samt förbättringar inom flera avseenden. Figur 14 visar hur NetScene ser ut idag.



Figur 14. NetScene och dess olika vyer.

Editering av entiteter (objekt) som har en position kan göras i den integrerade kartvyn (i 2D). Där kan även utplacering av brytpunktsbanor och andra positions-bundna objekt göras. Editering av övriga attribut för scenario-objekt görs i den sk egenskapsvyn som är dynamisk map på vad som är markerat i scenarioverktyget. Egenskapsvyn medger t ex även att attribut för flera objekt kan göras samtidigt.

6.3.2.1 Scenariomodell och scenarioformat

NetScene har ett eget XML-baserat format för scenariofiler och scenariomodeller. Scenariomodellen är en modellering av alla slags objekt, relationer och interaktioner som man vill kunna hantera i ett scenario. Scenariofilen innehåller konfigurerade instanser av objekt och relationer mellan objekt för ett visst scenario och specificerar vilken scenariomodell som används av scenariot. Flera scenarion kan således använda samma scenariomodell. Scenariomodellen har ingen koppling till något givet scenario och påminner om den sk FOM:en i HLA. Hela scenariohanteringen och alla interna objekt är helt oberoende av HLA, även om vissa likheter finns för att möjliggöra enkel koppling till en HLA-federation (se vidare under kap 6.3.2.3)

Scenariomodellen innehåller följande:

- Entitetsklasser och deras attribut, arvshierarki samt möjliga relationer till andra entitetsklasser.
- Meddelandeklasser (interaktionsklasser) och dess parametrar.
- Datatyper som används för typning av attribut och parameterar i entitetsklasser resp meddelandeklasser.

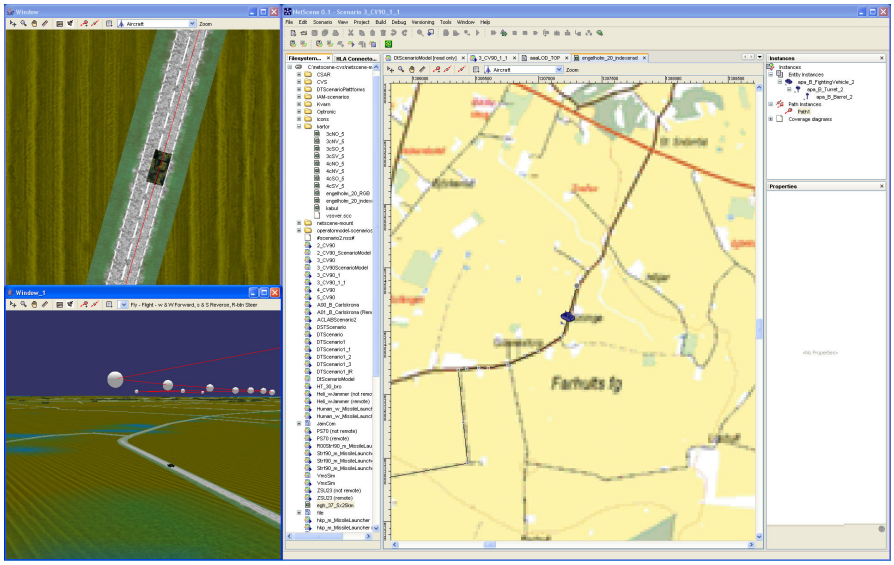
Relationer mellan entiteter kan parametersättas, t ex att en entitet sitter fast på en annan entitet med en relativ position. Nya datatyper kan skapas utifrån structar, arrayer och enkla datatyper som int8, int16, int32, int64, float32, float64, string, vec2d, vec3d, binary data, boolean och enum. I och med att alla inbyggda datatyper är entydigt specificerade (internrepresentation och kodning) så blir även de egna specificerade datatyperna det också.

Dynamiska beteenden för entiteter i ett scenario beskrivs av entitetsmodeller som är Java-kod som implementerar ett fördefinierat gränssnitt (API). Entitetsmodeller kan antingen byggas in i NetScene som en plugin eller utvecklas direkt i NetScene och sedan laddas in utan att NetScene startas om. Entitetsmodeller kan parametersättas pss som entiteterna själva. Varje entitet kan ha noll eller flera entitetsmodeller kopplade till sig som styr beteendet för entiteten. Varje entitetsmodell är kopplad till exakt en entitet, dvs man skapar flera instanser av en entitesmodell om man vill styra flera entiteter. Några av de idag inbyggda entitetsmodellerna hanterar följning av brytpunktsbanor antingen med eller utan terrängföljning. Detta beteende är helt enkelt en dynamisk förändring av attributen position, orientering, hastighet, acceleration m fl för en viss entitet. Flera beteenden kan läggas på varandra. Entitetsmodeller kommunicerar med varandra och omvärlden med meddelanden. Ett meddelande kan t ex vara träff av explosion som en entitetsmodell tolkar som att entiteten ska tas bort eller att skadeattributet ska ändras. Entitetsmodellerna exekveras endast när simuleringen startas, dvs under redigeringen så görs endast en parametersättning av modellerna.

I själva scenariot, som använder sig av en viss scenariomodell, finns alla instanser av entiteter (med attribut, relationer och entitetsmodeller), brytpunktsbanor, areor (dynamisk yttäckning mm), etc. Utöver scenariomodell kan man även specificera vilken terrängmodell som ska användas i scenariot (se mer om terrängmodell nedan). I scenariot kan varje entitet märkas med information om entiteten ska ägas av NetScene eller av en annan simuleringskomponent (t ex federat i HLA). Alla entiteter som ägs av NetScene kommer antingen att vara helt statiska eller uppdateras dynamiskt om någon entitetsmodell är konfigurerad för entiteten. Det initiala tillståndet samt konfigurationsinformation för övriga entiteter kommer att distribueras ut till andra simuleringskomponenter som i sin tur kommer att skapa och styra beteendet för dessa entiteter.

6.3.2.2 Terrängmodell och 3D-vy

I varje scenario finns en koppling till en terrängmodell, som kan vara antingen en shape-fil (vektorbaserad kartinformation) eller en standardterräng (jordellipsoid utan extra höjddata mm). Terrängmodellen kan användas som underlag för entitetsmodeller (t ex terrängföljning), för beräkning av yttäckningsdiagram samt för visualiseringsstöd i 3D. Implementationen av terrängmodellen ligger bakom ett gränssnitt (API) som medför att stöd för andra typer av terrängdatabaser (t ex dynamisk terräng) kan implementeras och pluggas in i efterhand. Idag bygger terrängmodellen på Open Scene Graph (OSG) som är en öppen mjukvara (open source) för 3D-hantering och finns för flera olika plattformar. Terrängmodellen har även en 3D-vy för editering av ett scenario. Detta medger att entiteter i scenariot grafiskt kan placeras ut map höjd och skymdhet i terrängen på ett helt annat sätt än vad som kan göras i 2D-kartan. Figur 15 visar en skärmdump från 3D-vyn i NetScene.



Figur 15. Editering i 3D till vänster och kartvy mm till höger. De vita bollarna är handtag för editering av brytpunktsbanor som stridsvagnen i bilden ska åka längs.

6.3.2.3 HLA-koppling

I NetScene finns en HLA-koppling som fungerar som en brygga mellan alla entiteter och meddelanden i NetScene och objekt och interaktioner i HLA-världen. Kopplingen mellan en entitet i NetScene och ett objekt i en HLA-federation sköts i mha plugins. Stöd för nya klasser av objekt hanteras genom att en ny plugin skrivs. Medan pluginen sköter konvertering av attribut och mappning mellan attribut i HLA-världen och attribut internt så sköter HLA-kopplingen allt annat, bl a registrering och borttagning av objekt samt hantering av den logiska kopplingen mellan HLA-objekt resp NetScene-entitet.

Distribution av scenario-objekt som inte ägs av NetScene sker mha interaktionen RemoteCreateObject enligt MOSART Federation Agreements (se kap 4). Alla attribut för en entitet som inte är modellerat som ett attribut i HLA-FOM:en (t ex konfigurationsdata) skickas med som ett sk internt attribut i RemoteCreateObject-interaktionen. Redigering av fjärr-entiteter under ett aktivt scenario hanteras mha interaktionerna RemoteChangeAttribute och RemoteDeleteObject.

HLA-objekt som inte fanns med i scenariot i NetScene men som dyker upp under simuleringen kommer automatiskt att dyka upp i NetScene som en ny entitet (om det finns en plugin för den objekt-klassen). Det här gör att flera NetScene-federater kan existera i en och samma federation, ev tillsammans med andra scenarioverktyg om man vill det.

6.4 MIND

6.4.1 Bakgrund

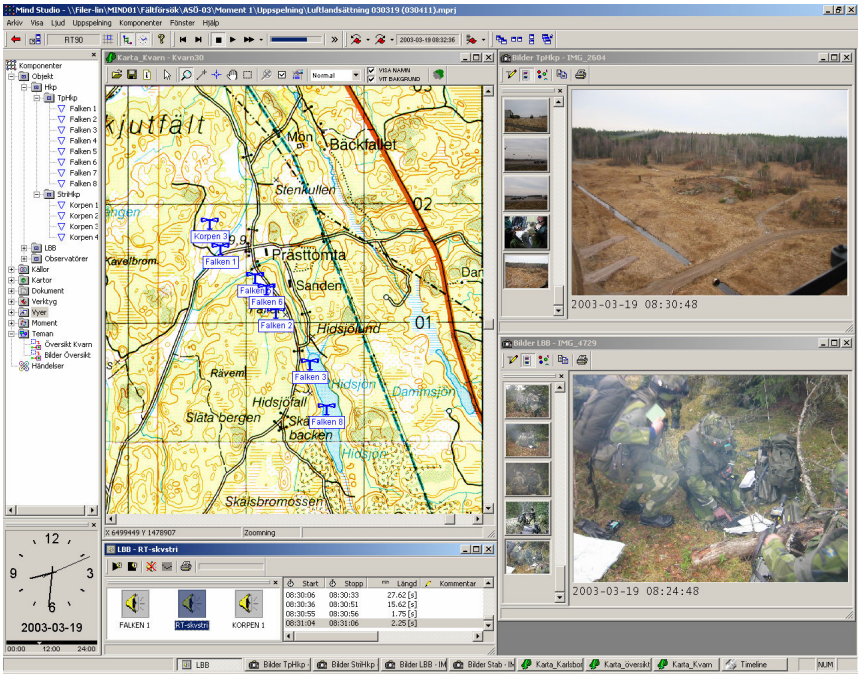
MIND är ett ramverk för visualisering av distribuerade taktiska händelseförlopp som utvecklats av FOI och VSL (Visuell Systemteknik i Linköping) sedan mitten av 1990-talet. Idag är MIND inne på sin tredje version, och utvecklingen av den fjärde har just påbörjats, nu av enbart FOI. Grunden för MIND återfinns i metoden *rekonstrution & utforskning* som utvecklades av Dr. Magnus Morin och Dr. Johan Jenvald, se [7]. Verktyget MIND används för att just rekonstruera händelseförlopp och i efterhand visualisera dem för att kunna utforska händelserna i detalj.

Rekonstruktionen har traditionellt varit baserad på tidsstämplad data från militära övningar. Data som använts för att återskapa dessa övningar är till exempel positionsloggar från GPS:er utplacerade på soldater och fordon, fotografier från övningsområdet, kartor, radiokommunikation, mm. Data sätts sedan samman till en kronologisk modell, ett så kallat *moment*, som visualiseras i MIND med hjälp av en mängd skräddarsydda vyer, t ex:

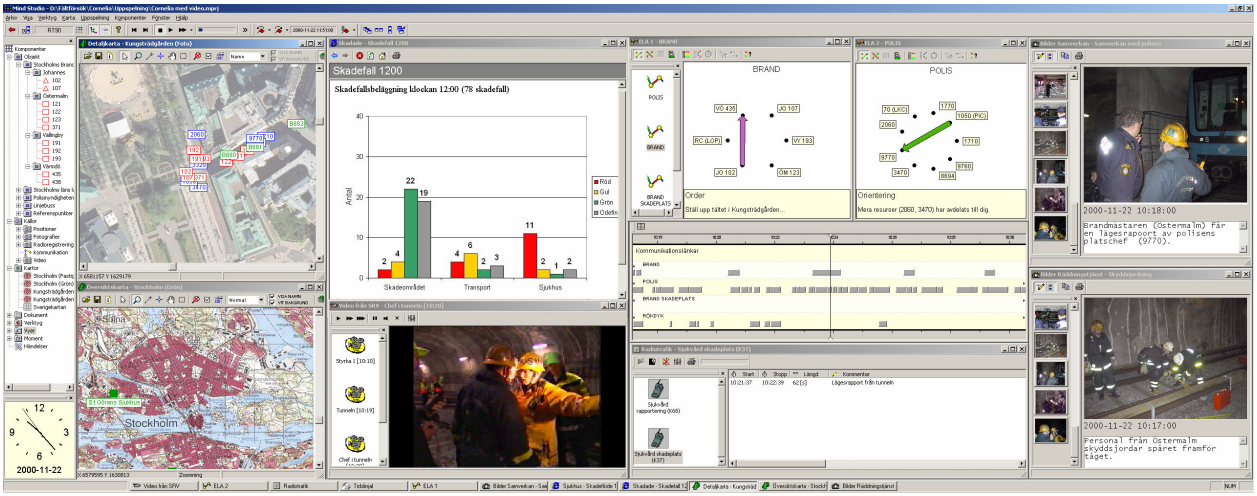
- **Kartvy**: visar enheters positioner och förflyttningar på godtyckliga kartor med hjälp av positionsloggarna från GPS-systemen.
- **Fotovy**: Visar digitala fotografier tagna av observatörer.
- **Videovy:** Visar videoklipp från till exempel övervakningskameror, bildskärmar eller handkameror.
- **Kommunikationsvy:** Visualiserar och spelar upp kommunikation i olika kommunikationsnätverk, t.ex. radiokanaler, chat-system el dyl. Kommunikationen kan klassificeras i ett antal olika dimensioner för att underlätta sökning i och tolkning av datamängden.
- **Dynamisk tidslinjal:** Visar en översikt över händelseförloppet på en abstrakt nivå och ger möjlighet för operatören att definiera tidpunkter eller tidsperioder av särskilt intresse.
- **Dokumentvy:** Visar ett godtyckligt dokument, t ex en plan eller en skriven order.

Själva ramverket MIND är komponentbaserat, vilket gör det enkelt att för varje specifik övning utveckla nya modeller, datakällor och/eller vyer för att påföra och visualisera nya typer av data. På detta sätt är MIND väldigt generellt och flexibelt vilket gör att man kan anpassa det till nya domäner med några mer eller mindre enkla handgrepp. Utöver i armén har MIND använts av bland andra SRV, Singapore Armed Forces Centre for Military Experimentation, FMV LedUtvC och Marinen. En utförligare beskrivning av MIND återfinns i [8].

Figur 16 visar en skärmdump från MIND laddat med data från ASÖ’03, en klassisk arméövning där trupp-position och kommunikation är i fokus. Figur 17 visar systemet i samband med övningen Cornelia som simulerade en tunnelbaneolycka och det efterföljande räddningsarbetet i Stockholm november 2000 och där fokus istället låg på samarbete mellan de inblandade organisationerna.



Figur 16. Arméns slutövning 2003 visualiseras i MIND.



Figur 17. Övningen Cornelia fokuserar på samarbetet mellan olika organisationer efter en tunnelbaneolycka i Stockholm.

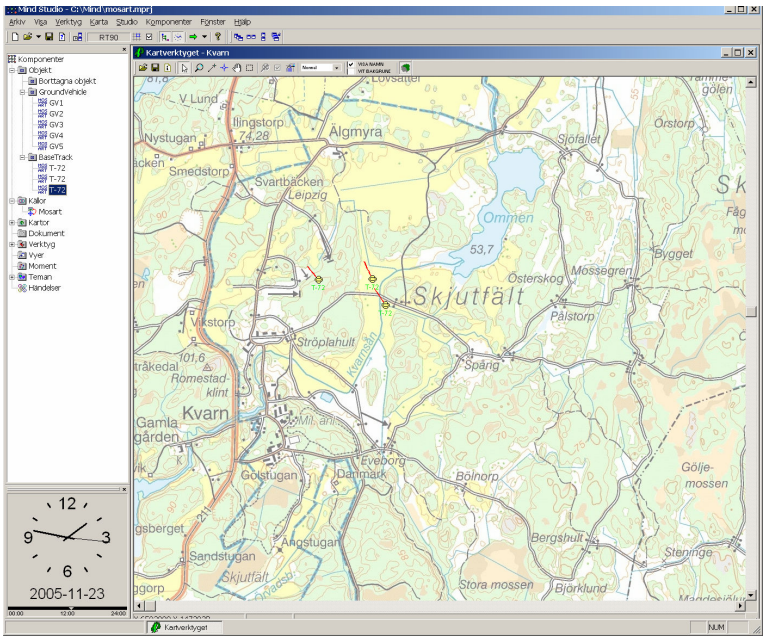
Istället för att efter en övning diskutera hur det har gått och vad som har hänt, kunde man i båda dessa övningar lyfta diskussionen en nivå till att diskutera *varför* det hände. En direkt konsekvens av detta blev att man efter övningen Cornelia kunde påvisa hur ett missförstånd uppstod och ledde till flera onödiga dödsfall. Problemet är idag åtgärdat och sådana missförstånd behöver därför aldrig uppstå i skarpa insatser, i förlängningen innebär det flera räddade liv.

6.4.2 Hur MIND används i MOSART

MIND används idag som ett visualiseringsverktyg för simulerad positionsdata i MOSART. Via den klassiska kartvyn i MIND kan man visa enheters interaktioner med federationen. Med hjälp av olika konfigurationsinställningar kan enheter mappas mot godtyckliga symboler och placeras i olika lager, som var och ett kan aktiveras eller deaktiveras för visualisering. Detta gör att man på ett enkelt sätt kan filtrera den information som ska visas på kartvyn.

Idag finns stöd för att visualisera alla basentiteter i någon form, men framför allt markfordon, sensorer och spår. Sensorer visas med symbol och täckningsyta, markfordon visas med position och möjlighet till att visa fartvektorn medan spår visas med symbol och en liten svans som indikerar spårets historik, samt färgkodning som visar klassning, klassningskonfidens och huruvida spåret är aktivt eller inaktivt. Ett exempel visas i figur 18 nedan.

All data strömmar in till MIND i realtid vilket gör att visualiseringen reflekterar federationens simulerade tid så nära som möjligt.



Figur 18. MIND visar tre aktiva spår av MOSART-simulerade T-72:or i terrängen kring Kvarn. Sensorer, verklig position och täckningsytor är dolda för att ge en så tydlig bild som möjligt av spåren.

6.4.3 Framtida arbete

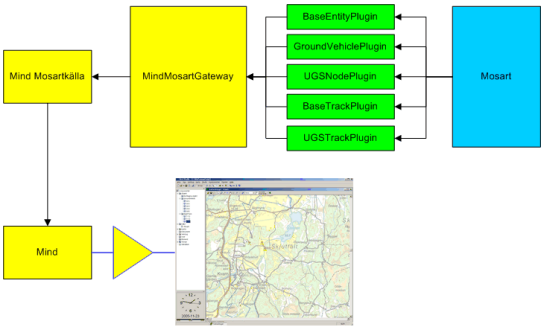
Det är endast en bråkdel av MIND:s kapacitet som används i MOSART idag. Orsaken till detta är att fokus har legat på att få ett fungerande scenario med marksensornät och spårning. Man skulle mycket väl kunna tänka sig att en helt ny vy skapades för att visualisera samtliga objekt i federationen med enkla gränssnitt för att undersöka och/eller modifiera objekt under körningen. För plattformar med flera sensorer, till exempel UAV:er eller CARABAS, skulle man kunna tänka sig specialskrivna vyer som visualiserade objektets status på ett överskådligt sätt. Mikrofoner skulle också kunna visualiseras med nya vyer för att till exempel kunna lyssna på vad som hörs i mikrofonen, det finns i princip ingen information i MOSART-federationen som inte skulle visualiseras på något sätt i MIND.

Det vore också önskvärt att det kunde skapas traditionella MIND-händelser istället för som idag enbart realtidsvisualisering. På detta vis kunde man enkelt spara en körning och visualisera den i efterhand, på samma sätt som görs med klassiska MIND-moment.

Marksensornäten och spårningen är ett intressant område som är svårt att visualisera på ett bra sätt. MIND skulle kunna göra detta mycket bättre och tydligare än vad som görs idag. Exempelvis skulle man vilja kunna se vilken sensor som är ansvarig för spårningen vid varje given tidpunkt och få stöd med att analysera hur väl varje nod fungerar, för att till exempel kunna upptäcka defekta noder. Det finns mängder av alternativa representationssätt som skulle kunna testas och utvärderas med hjälp av MIND.

6.4.4 Teknisk beskrivning av MOSART-kopplingen

MOSART-federationen exponeras till MIND via det visualiseringsplugin-gränssnitt som skrivits i C++. En källa har skrivits i MIND som översätter pluginens anrop till anrop till MIND enligt figur 19. Konfigurering av källan sker inifrån MIND enligt figur 20. Plugins-arkitekturen gör att enklare objekt kan visualiseras genom att lägga till nya plugins. Inga förändringar behöver göras I MIND så länge gränssnittet hålls intakt. Idag finns referensimplementationer för plugins för BaseEntity, GroundVehicle, UGSNode, BaseTrack samt UGSTrack. Gemensamt för varje plugin är att de objekt som visualiseras väljer symbol med hjälp av properties-filerna tillhörande pluginsen.



Figur 19. Dataflödet MOSART –MIND.

Figur 20. Inställningar för MOSART-koppling.

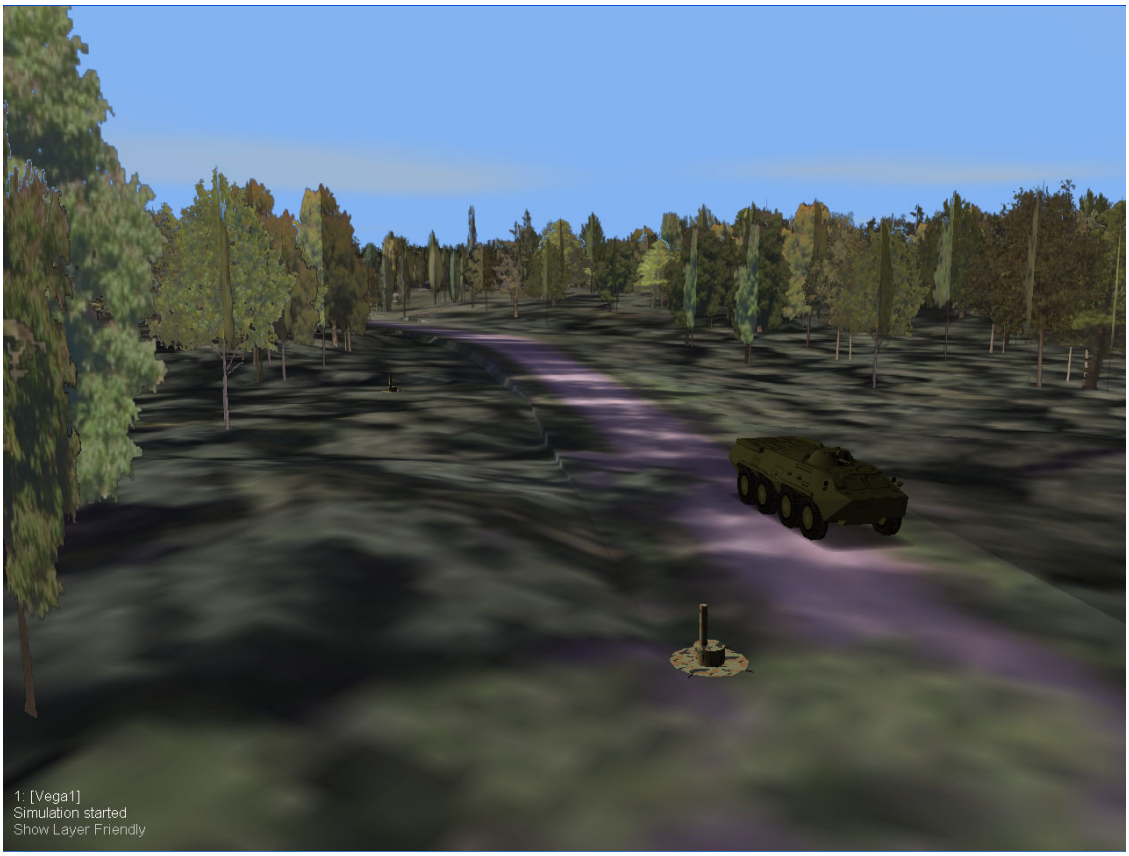
6.5 Vega

Vega är ett kommersiellt verktyg för 3D-visualisering från MultiGen-Paradigm som har kopplats till MOSART testbädd. Applikationen, *VegaFederate*, är utvecklad så att terräng, fordon och andra entiteter och händelser i en MOSART-simulering visualiseras i 3D med hjälp av Vegas 3D-motor, se figur 21 och 23. Idag finns grundläggande stöd för att visualisera alla basentiteter, men framför allt har visualisering av markfordon och sensorer testats.

När man startar *VegaFederate* laddas terrängmodell samt plugins för de objekt som används i simuleringen. I figur 21 och 23 visas entiteter i en omvärldsmodell från Kvarn som har skapats utgående från data från flygburna lasersystem (se kap 11).

Man kan använda *VegaFederate* dels som en ren MOSART-kopplad 3D-visualiserings-applikation med egna plugins eller så kan man integrera den som en vy i en egenutvecklad applikation. Vega är en Windowsapplikation skriven i C++. Den 3D-motor ifrån MultiGen-Paradigm den bygger på finns för Windows, Linux och SGI Irix, vilket möjliggör användande av *VegaFederate* på andra plattformar.

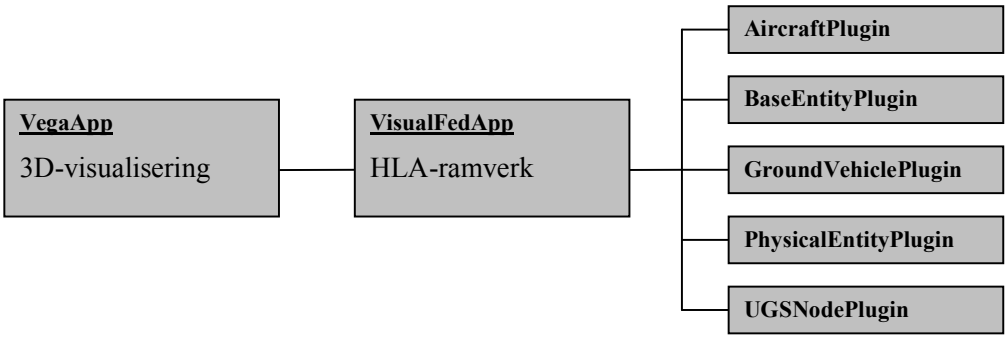
-



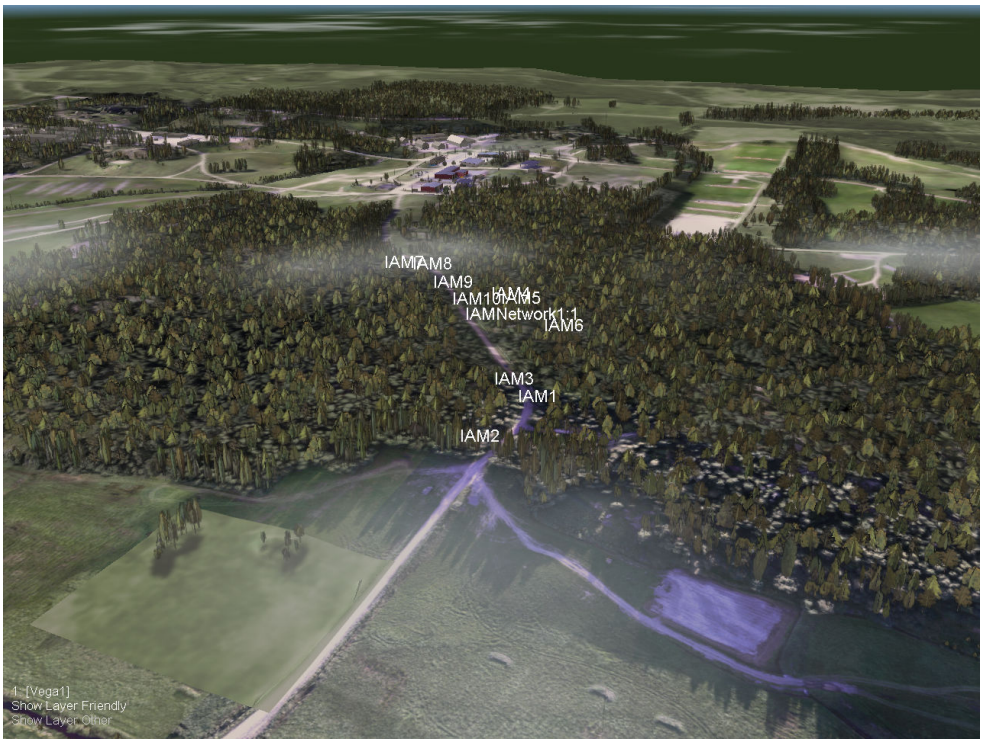
Figur 21. Ett fordon passerar en akustisk sensor i en terrängmodell av Kvarn.

6.5.1 Design

VegaFederate består dels av VisualFedApp som är ett HLA-ramverk som även används av MIND. VisualFedApp håller reda på alla objekts tillstånd samt hur de ska visualiseras, vilket beskrivs av de plugins som används. Plugins-arkitekturen, se figur 22, gör att enklare objekt kan visualiseras genom att lägga till nya plugins. Inga förändringar behöver göras i *VegaFederate* så länge gränssnittet hålls intakt. Idag finns referensimplementationer för plugins för RPRFOM-klasserna BaseEntity, GroundVehicle, UGSNode och Aircraft. Gemensamt för varje plugin är att de objekt som visualiseras väljer symbol/modell med hjälp av properties-filerna tillhörande resp. plugin. Det som händer i VisualFedApp förs sedan över till VegaApp som hanterar själva 3D-vyn, laddar terräng och modeller, etc. Man kan på samma sätt enkelt använd VisualFedApp för att visualisera i en annan 3D/2D motor som exempelvis *SpatialAce* eller *OpenSceneGraph*.



Figur 22. Design av plugins-baserat visualiseringsgränssnitt.



Figur 23. Översiktsvy av Kvarn med akustiska marksensorer utplacerade längs vägen.

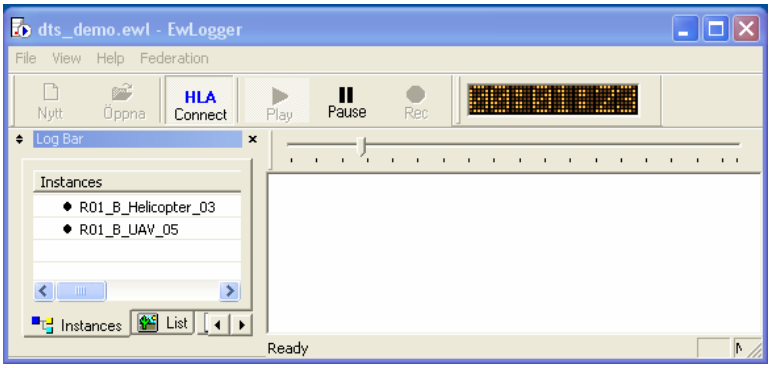
6.6 MOSART/EW Logger

Ett loggningsverktyg kallat EW Logger baserat på MOSART:s kits har utvecklats inom projektet *Duellsimulering Telekrig*. EW Logger utgör en del i EWSim (Electronic Warfare Simulation interface model), som är ett MOSART-integrerat simuleringsramverk främst för telekrigssimuleringar.

Loggningsverktyget är till stor del ett helt generellt loggningsverktyg för MOSART-simuleringar. Denna generella loggnings- och återuppspelningsdel av EW Logger kommer göras tillgänglig som en separat applikation, MOSART Logger. Utöver de generella delarna finns i EW Logger en del specialfunktioner för EWSim.

6.6.1 Funktion

EW Logger är ett verktyg för att lagra, analysera och återuppspela en HLA-simulering. Det går att köra EWLogger som konsolapplikation, Windowsapplikation eller integrerat i en HLA-federat. I figur 24 visas en skärmdump av EW Logger som Windows-applikation.



Figur 24. Skärmdump från EW Logger.

EWLogger kör hela tiden i bakgrunden och loggar kontinuerligt de data som utbyts över HLA. Den fungerar snarlikt en ”bandspelare” med funktioner som play, rec och pause. Vid återuppspelning kan man med en tidslider snabbt hoppa till en önskad tidpunkt i simuleringen och fortsätta uppspelningen från den tidpunkten. Ett flikfönster med en instansvy visar aktiva objekts status.

6.6.2 Filformat och objektlagring

En logg består av tre filer, en projektfil (ewl) med översiktlig information om körningen, en deltafil med HLA-förändringar i tidsordning samt en keyframefil som gör att man enkelt kan läsa upp alla objekts tillstånd i ett givet ögonblick. Allt lagras med hjälp av de HLA-codecs (se kap 5.2) för att göra det möjligt att enkelt att läsa filerna under alla operativsystem som MOSART stödjer (Windows, Linux, Solaris) och i alla programspråk som MOSART stödjer (C++, Java, Matlab). I figur 25 visas ett exempel på en uppsättning loggfiler från EW Logger.

Figur 25. Exempel på en uppsättning loggfiler från EW Logger.

6.6.3 Design

Baskomponenterna för EWLogger är skrivna för att dels kunna integreras i andra applikationer, dels för att kunna kompileras under de operativsystem som MOSART stödjer. All loggning sker i en separat tråd för att påverka simuleringens tidsstegning så lite som möjligt.

6.6.4 Framtida arbete

Följande funktionalitet kommer inarbetas framöver:

- Starta en simulering från en önskad tidpunkt i en inspelning
- Fler analysvyer och grafer
- Möjlighet att lägga till och spara kommentarer knutna till en viss tidpunkt
- Möjlighet att ändra parametervärden vid återuppspelning av scenario

7 Användarinteraktion

Institutionen Människa-System-Interaktion, MSI har stora erfarenheter av försök där användaren står i fokus. MSI har under några års tid utvecklat en plattform med möjlighet att göra avancerade användarförsök. Plattformens syfte har varit att simulera olika operatörsplatser med modifierbara användargränssnitt, avancerade omvärldssimuleringar där kraven på detaljnivå är högre än vad som kan tillhandahållas i de flesta andra simulatorer, operatörsinteraktionen måste fungera utan fördröjning eller andra komplikationer.

Ofta är efterfrågan på FM:s utrustning väldigt hög. Det kan vara svårt att få tillgång till den utrustning som behövs vid ett försök i den utsträckning som är nödvändig för att genomföra försöket på bästa möjliga sätt. Därför är det många gånger viktigt att kunna effektivisera användandet när man väl har tillgång till utrustningen. Detta görs med fördel genom olika förförsök genom någon form av simulering där man simulerar den skarpa utrustningen. Därför ser vi en stor fördel att kunna integrera MSI:s plattform i MOSART testbädd för att i fortsättningen förbättra möjligheterna att genomföra användarförsök med avancerad användarinteraktion.

I figur 26 visas Moog-plattformen som finns i MSI-institutionens labb på FOI i Linköping samt skärmdumpar från HiFi Engine i två olika applikationer.

Ett användningsområde för MSI:s plattform har varit att efterlikna de olika operatörsplatserna i ett stridsfordon 90 (CV90, Combat Vehicle 90) med fokus på förarplatsen, på ett så verklighetstroget sätt som möjligt. Vi valde därför att inrikta oss på att få in CV90-simulatorn i en MOSART-simulering som t ex en händelsegenerator för andra simuleringsparter. Ett naturligt användningsområde som vi sett är i simuleringar där ett fordon med förprogrammerad eller manuellt styrd rutt av väldigt statisk karaktär använts för att generera testdata. Data har sen utnyttjats av olika sensorsystem för att utföra beräkningar på. För att istället få mer realistiskt testdata kan CV90-simulatorn generera det genom att endera låta en förare köra vagnen manuellt eller låta simulatorn styra vagnen med hjälp av den AI-funktionalitet som finns i simulatorn.

Figur 26. Moog-plattform i MSI-labb (vänster), skärmdump på simulerat stridsfordon i Kvarn (mitten) och skärmdump från NBC-simulator i Norrköping (höger).

Andra användningsområden är att olika hjälpsystem som tas fram, t ex för att underlätta för en operatör i en CV90-besättning, kan testas och utvärderas i CV90-simulatorn direkt. På så sätt kan man förbättra systemen och upptäcka eventuella brister i ett tidigt skede som normalt kommer fram först i en simulering av en faktisk stridssituation med dyrbar utrustning.

Att få tillgång till källkod för en militär applikation för att ha möjlighet att ändra på vissa egenskaper som man vill utöka eller förändra, kan ofta vara kostnads- och tidskrävande. Det kan röra sig om att t ex modifiera displayutformning eller lägga till ytterligare gränssnitt i form av 3D-ljud för hotindikering. I många fall är det enklast att göra en enkel imitation av de delar i applikationen man vill modifiera som sen utföra utvärderande försök för att komma fram till om

den förändrade funktionaliteten leder till någon förbättring. Om så är fallet kan konkreta förbättringsförslag presenteras som är starkt förankrade i praktiska användarförsök. MSI-plattformen tillför en styrka då det finns möjligheter att modifiera all funktionalitet samt att mycket erfarenhet av användarförsök finns inom MSI-institutionen. Att ha möjligheten att koppla ihop MSI-plattformen med MOSART testbädd ger ytterligare en fördel då t ex CV90-simuleringen får tillgång till MOSART:s olika moduler såsom t ex sensormoduler för mer realistiska data till operatören, loggningsmodul mm. MOSART testbädd får i sin tur del av funktionalitet som MSI-plattformen kan tillhandahålla såsom:

- Avancerad omvärldspresentation med väldigt hög detaljnivå hos grafik, ljud och fysik som väldigt få andra simulatorer kan presentera.
- Möjlighet att öva flyg, markfordon samt avsutten soldat i samma simulering.
- Avancerad användarinteraktion.
- Tillgång till speciell utrustning såsom rörelseplattform, huvudrörelseregistrering, blickriktningsregistrering mm som redan är integrerad med MSI-plattformen.
- Koppling till Merlin. (Merlin utvecklas av systemteknik och är en arkitektur för modellutveckling)
- Etc.

8 Forskningsresultat i testbädden

För att MOSART testbädd ska vara meningsfull måste den naturligtvis användas. Grundfunktionalitet i all ära, men utan simuleringskomponenter som ska värderas eller användas för att värdera något annat eller användas för att demonstrera något som är intressant ur ett projekts forskningsperspektiv saknar testbädden egentlig mening. Ett av de primära målen med MOSART-projektet har ju hela tiden varit att effektivisera forskningsverksamheten på FOI. Det är för att uppfylla detta mål som testbädden utvecklats. Den uppfyller målet genom att den medgör återutnyttjande av tidigare framtagna resurser i form av generella verktyg (grundfunktionalitet) men även återutnyttjande av andra forskningsresultat som tidigare anslutits till testbädden. Eftersom testbädden är en generell plattform och inte är inriktad för simulering inom något speciellt forskningsområde mer än något annat möjliggör den dessutom forskningsområdesöverskridande verksamhet. Detta återspeglas delvis i den forskningsmässiga bredd projekten som använder MOSART testbädd har.

8.1 Användarprojekt

På FOI finns idag ett antal projekt som använder MOSART testbädd. Nedan presenteras mycket kort vilka dessa projekt är och hur de nyttjar MOSART-miljön. Mer information om en del av dessa användarprojekts MOSART-användande finns i [9].

8.1.1 MOSART II

MOSART II är ett samarbete mellan FOI och DSO/DSTA i Singapore som beställts inom ramen för LedsystT. I detta projekt utgick man från den testbädd som fanns och har utvecklat ytterligare funktionalitet i testbädden i form av:

- Införande av tjänstekonceptet i testbädden, dvs möjligheten att laborera med tjänster, t ex sensortjänster och fusionstjänster, i testbädden
- Koppling till verklighet, dvs möjlighet att strömma data mellan testbädden och verkliga sensorer och plattformar

I november 2005 demonstrerades detta i FM Visionslab i Enköping.

8.1.2 Interaktiva adaptiva marksensornät

Projektet Interaktiva adaptiva marksensornät (IAM) syftar till att visa hur ett nätverk bestående utav marksensorer och kommunikationssystem skulle kunna utformas, samt hur systeminteraktion med användarna i ett sådant system skulle kunna ske [10].

Projektet har använt de scenario- och visualiseringsverktyg som finns i MOSART testbädd samt integrationsprogramvaran för att koppla ihop och köra sina simuleringar. Det simulerade marksensornät som tagits fram i projektet finns nu tillgängligt i MOSART testbädd för andra projekt att använda.

8.1.3 Informationssystem för markspaning

I projektet Informationssystem för markspaning (IS-MS) tas ett frågespråksbaserat system för markspaning fram. Systemet nyttjar data från olika typer av sensorer för att besvara en användares frågor.

I november 2005 genomfördes projektets slutdemonstration tillsammans med fem andra FOI-projekt. Det scenario som visades var uppsatt och exekverat i MOSART testbädd.

8.1.4 Samverkande system och systemtilltro (Tre Ess)

Projektet Tre Ess utvecklar och utvärderar metoder och algoritmer för intern och extern datafusion på operatörsstyrda luft-, sjö- och markbaserade stridsplattformar; intern datafusion för multisensorsystem på enstaka plattformar och extern datafusion för sensordata vilka genererats i flera plattformar.

Förutom sensormodeller och displayer har projektet utvecklat en plattformsburen DataFusionsNod (DFN) som fusionerar och styr ombordvarande sensorer, interagerar med operatör och kommunicerar med DFN på andra plattformar. Arbete pågår för att ansluta DFN till MOSART.

8.1.5 Markspaning med flygande farkoster

Projektet Markspaning med Flygande Farkoster (MFF) huvudsyfte är att studera, utveckla och utvärdera teknik nödvändig vid spaning mot marken med flygande sensorplattformar. Tyngdpunkten ligger på målföljning av markfordon genom fusion av sensordata från flera flygande plattformar sammankopplade i ett nätverk. Algoritmer för målföljning, baserade på partikelfiltret, och metoder för association av målspår från skilda tidpunkter, baserade på genetiska algoritmer, har utvecklats. Aktuella sensorer är IR och SAR (stripmap och GMTI), där simuleringsmodeller har utvecklats i projektet.

Dessa sensormodeller, målföljnings- och fusionsalgoritmer ansluts till MOSART testbädd. MOSART-miljön kan därmed användas för att studera utnyttjandet av andra sensormodeller i fusionen, för att få bättre möjligheter till simulering av intressanta scenarier samt för att kunna visualisera resultateten av fusionsmetoderna.

MFF-projektet bedrivs i samarbete med SAAB Aerospace och är finansierat av NFFP (Nationellt Flygtekniskt Forskningsprogram).

8.1.6 SEMARK

SEMARK är ett paraplyprojekt på avdelningen för Sensorteknik på FOI inom FoT 8 som samordnar fältförsök och simuleringsarbete för sensorer för att utgående från riktiga sensordata skapa förutsättningar för simulering och värdering av sensorer för markläge.

Följande sensormodeller integreras i MOSART-miljön:

- CARABAS II radar
- QWIP IR-sensor
- MultimIR hyperspektral sensor
- 3D lasersensor
- Mikrovågsradar, realapertur och SAR
- Akustisk sensornod

8.1.7 Duellsimulering Telekrig

Projektets mål är att presentera en simuleringsmiljö som på ett realistiskt sätt beskriver nätverksbaserade sensor- och telekrigfunktioner för studier av multispektral systemsamverkan i nätverk [11]. Det ramverk som tagits fram i detta syfte kallas EWSim.

Projektet utvecklar tillsamans med MOSART en scenarioeditor kallad NetScene, se kap 6.3.2, som används i MOSART testbädd. Inom projektet har även EW Logger, se kap 6.6, utvecklats.

Projektet har anslutit EWSim till MOSART testbädd och genomför sina simuleringar i testbädden.

8.1.8 Ledningskrigssimulatorn (LKS)

Ledningskrigföringssimulatorn (LKS) syftar till att studera kommunikation, telekrig och CNO (Computer Network Operations) i ett ledningsnätverk. Detta för att genom simuleringar ge en ökad förståelse för hur ledning fungerar i en miljö som inkluderar både traditionellt telekrig och data- och nätverksoperationer. Kommunikationen mellan noderna lyfts från att enbart behandla länken mellan två noder till att även simulera access- och routingprotokoll i ett nätverk. På detta sätt möjliggörs nätverkssimulering och principstudier av hur nätverket påverkas av störning samt hur nätverket skall agera för att få den radiosignatur som minimerar upptäcktsrisk.

Verksamheten är ännu i sin linda och många frågor återstår att besvara, t ex hur verkliga meddelanden skall överföras/förvanskas och hur snabba trafikförlopp som t ex handskakning i en trafikuppkoppling skall simuleras för att kunna köras i realtid.

Från FOI används EWSim (Electronic Warfare Simulation Interface Model) som är en infrastruktur för telekrigsimuleringar som använder sig av MOSART testbädd.

8.1.9 Datafusion i sensornätverk med fokus på telekrig

Syftet med projektet är att studera vinsterna med att introducera datafusion i sensornätverk för att undertrycka störning och på så sätt uppnå ett robustare beslutsstödsystem [12].

Projektet utnyttjar de telekrigsmodeller som integrerats i MOSART av projektet *Duellsimulering Telekri*g (se ovan) för att simulera störning. En modul kopplad till MOSART som undertrycker störning med hjälp av olika datafusionsmetoder utvecklas och utgör projektets forskningsresultat.

8.1.10 LedsystT C2

Inom LedsystT finns ett antal IPT där överföring sker av kunskap från FOI till LedsystT. I det IPT som arbetar med C2-frågor integreras sensormodeller för IR- och SAR-simulering som tidigare tagits fram i ett FOI-projektet MFF. Avsikten är att nyttja dessa modeller för studier av C2-relaterade frågor, bl a inom *Negativ information* [13].

8.1.11 TEBE

Inom projektet Tekniskt Beslutsstöd ska en plattform för användarexperiment tas fram. Denna plattform ska sedan användas för att genomföra användarexperiment där användare får använda prototyper av olika typer av tekniska beslutsstöd som ansluts till plattformen. En av poängerna med experimenten är att genom att analysera resultaten från användarexperimenten kan beslutsstödsforskningen styras i lämplig riktning.

MOSART-miljön utgör med den funktionalitet som idag finns en utmärkt grund för den plattform för användarexperiment som ska tas fram inom projektet. För att bli en plattform för användarexperiment ställs krav på realtidsprestanda och integration av applikationer med användarinteraktion, se kap 7. Även den funktionalitet för Mixed Reality, se kap 9, som finns i MOSART testbädd är av stort intresse.

8.1.12 Övrigt

Ett flertal andra projekt på FOI är också intresserade av att använda MOSART testbädd och utvärderar för närvarande på vilket sätt och när detta bör ske.

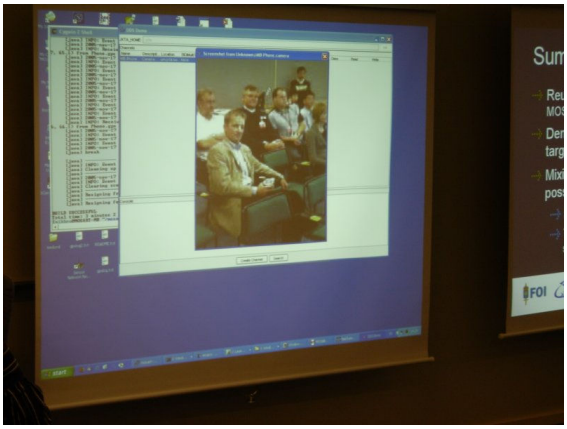
9 Koppling till verklighet

MOSART har fram till idag endast berört *simulering*, även om inspelade sensordata har använts i flera sammanhang. Tidigt i MOSART:s roadmap har koppling till riktiga sensorer och plattformar pekats ut som en viktig del i testbädden. Tanken är att en simulering ska kunna ta data från och styra en riktig sensor eller att positionsdata för riktiga plattformar ska kunna strömmas in i simuleringen i realtid. Själva konceptet är egentligen större än så, där gränserna mellan simulerat och verkligt suddas ut mer och mer, sk mixed reality. En förutsättning (i de flesta fall) för detta är att simuleringen kan exekveras i realtid, vilket inte alltid är fallet. I många fall kan detta avhjälpas genom att resultat från icke-realtids-modeller spelas in på förhand och spelas sedan upp i realtid. Givetvis kan utdata från dessa modeller inte påverkas av externa faktorer vid uppspelningen, men detta behöver inte vara något stort problem i alla sammanhang.

Under året (2005) har flera tester genomförts där inslag av både simulerat och verkligt finns med. I samband med detta arbete har ett återanvändbart mjukvarubibliotek (DDS Kit) tagits fram för distribution av data över ett nätverk mha JXTA, en standard för peer-to-peer-kommunikation.

9.1 Försök: Koppling till verklig sensor

I november 2005 på Visionslab i Enköping demonstrerades konceptet att skicka sensordata i realtid till MOSART testbädd dels genom att data push:as ut från sensorn och dels genom att sensordata begärs från testbädden. I försöket användes kameran på en mobiltelefon som sensor och sensordata (bild från kameran) skickades via GPRS, Internet och WLAN (på Visionslab) till en bärbar dator med MOSART-miljön installerad, se figur 27. På mobiltelefonen var ett speciellt Java-program för fjärrstyrning av kameran installerad. Detta program kommunicerade direkt med peer-to-peer-nätverket som var uppsatt för distribution av sensordata.



Figur 27. Bild från fjärrstyrd mobilkamera under demonstrationen på Visionslab.

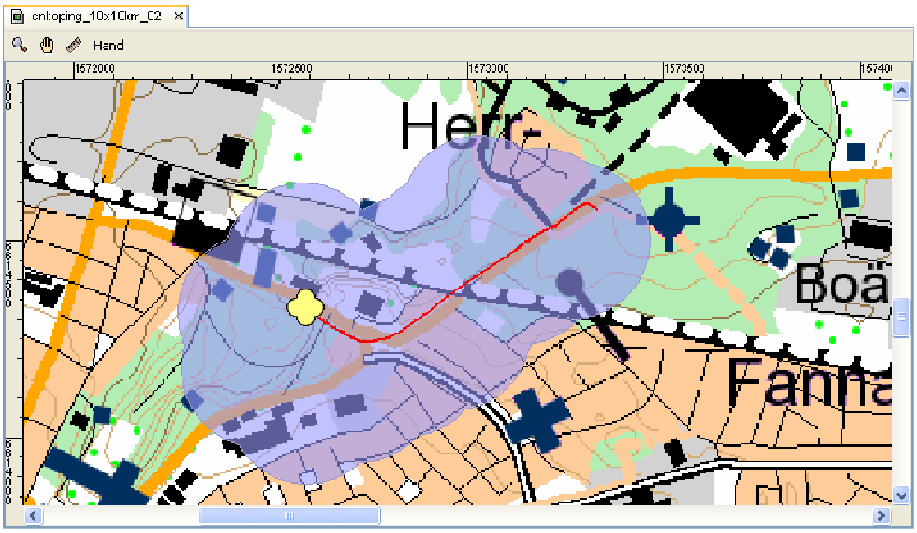
Även om just detta exempel inte är så avancerat så pekar tekniken på möjligheterna med att begära data från en verklig sensor till en simulering. Från den simulerade världen kan sensorn styras (här valdes upplösning på bilden) av sensorstyrningsalgoritmer eller av simulerade aktörer.

9.2 Försök: Koppling till verklig plattform

Det här försöket gick ut på att en bil utrustades med en mobiltelefon och en blutetooth-GPS, se figur 28, som via GPRS (och Internet och WLAN) skickar sin position till en MOSART-simulering.

Figur 28. Mobiltelefon och bluetooth-GPS som användes i försöket.

Även det här försöket gjordes på Visionslab i Enköping i november 2005. GPS-positionen för fordonet användes till att förflytta motsvarande objekt i simuleringsvärden (en bil). I simuleringen lades även ett simulerat marksensornät (från forskningsprojektet IAM) ut som i realtid kunde följa det verkliga fordonet som körde omkring i Enköping, se figur 29. Det här är ett exempel på hur forskningsresultat tidigt kan utvärderas i verkliga sammanhang, t o m innan en prototyp av hårdvaran har tagits fram.



Figur 29. Simulerat marksensornät följer ett verkligt fordon (med GPS) i realtid.

9.3 Nya möjligheter

Tekniken, men framförallt mognaden och standardiseringen inom distribuerade simuleringar, ger oss nya möjligheter att blanda simulerade system, sensorer, etc med verkliga plattformar, soldater, sensorer och system. Idag är det t ex fullt möjligt att utrusta soldater och fordon med var sin GPS för följning i realtid i ett simulerat system. Dessa data kan sedan användas av simulerade sensorer eller för visualisering i en simulator (t ex flygsimulator, stridsvagnssimulator). Om man sedan integrerar simulerade sensorer i verkliga system så är kretsen sluten, dvs att en plattform (t ex en stridsvagn) kan utrustas med simulerade sensorer som ser saker som finns i verkligheten. Den simulerade sensorn behöver inte ens placeras i plattformen den är tänkt att sitta i, utan detta kan göras i en datorhall med rätt beräkningskapacitet.

Andra möjligheter som denna blandning av simulerat och verkligt ger är skalbarhetstester. Antag att ett begränsat antal av en viss hårdvara eller prototypsystem finns tillgänglig men tester med

fler enheter behövs. I det här fallet kan man fylla på med simulerade komponenter för att testa skalbarhet samtidigt som realismen kan bibehållas i den del av systemet där verkliga komponenter används.

En annan aspekt av simulerat och verkligt är att kravställande på ett framtida verkligt system kan förenklas, sk Simulation Based Acquisition (SBA). Om dialogen kring kravställandet görs i form av enkla simuleringskomponenter som förfinas mer och mer i den riktning som användaren önskar kan en kalldusch undvikas vid en leverans av det verkliga systemet. I och med att verkliga plattformar, soldater och system kan påverka simuleringen kan utvärderingen av denna simuleringskomponent göras i den dagliga verksamheten och i de verktyg och system som används där. Den stora vinsten är att användaren kan påverka slutresultatet tidigt i utvecklingsfasen.

10 Tjänstekonceptet i MOSART

10.1 Bakgrund

I Försvarsmaktens strävan mot ett nätverksbaserat försvar har man tagit till sig konceptet tjänstebaserad arkitektur som ett sätt att modularisera och strukturera de resurser som är tänkta att finnas tillgängliga i nätverket. Med ett tjänsteorienterat angreppssätt får man väldefinierade tjänstegränsytor som beskriver hur en viss resurs kan användas och kopplas ihop med andra tjänster.

Målet med att införa tjänstekonceptet i MOSART var att göra det enkelt för forskningsprojekt att utveckla och laborera med tjänster. Fördelen med att använda tjänstebaserad kommunikation är att man kan abstrahera konsumenter och producenter av tjänster. Om man använder tjänster i MOSART är det lättare att malla strukturen på andra tjänstekoncept och tjänstebaserade system, så som tjänster för nätverksbaserat försvar. Detta gör att forskningsresultat i MOSART som förses med tjänstegränssnitt på ett enkelt sätt kopplas ihop med tjänster i andra tjänsteorienterade miljöer, t ex tjänstedemonstratorn i Enköping eller Web Services.

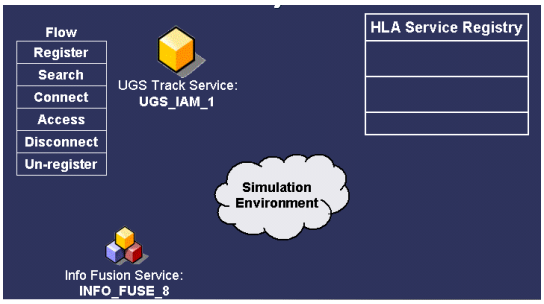
10.2 Design

När tjänstekonceptet skulle införas i MOSART testbädd tittade vi på välkända tjänstebaserade arkitekturer såsom Web Services, FMA och NAF. Vi ville se till att tjänster som implementeras i MOSART ska kunna interagera med tjänster i dessa arkitekturer på ett smidigt sätt. Vi kom fram till att följande saker måste en tjänst kunna göra:

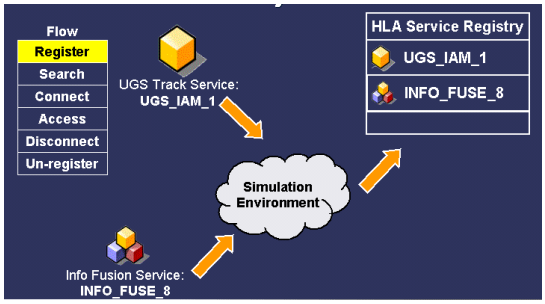
- Registrera sig i nätet och därmed bli tillgänglig för sökning så att andra tjänster kan hitta den
- Tala om vad den producerar (exponera sin tjänstegränsyta mot andra tjänster)
- Söka upp andra tjänster
- Koppla upp sig mot andra tjänster
- Få tillgång till andra tjänsters tjänstegränsytor
- Anropa metoder i andra tjänster via dess tjänstegränsytor
- Ta emot svar från andra tjänster i form av direkt svar resp abonnemang
- Leverera svar på tjänsteanrop från andra tjänster (direkt svar resp abonnemang)
- Koppla ner sig från andra tjänster
- Avregistrera sig från nätet

Registeringen och sökningen av tjänster kräver någon form av tjänstekatalog som tjänsterna kan registrera sig själva i och även använda för att söka upp andra tjänster.

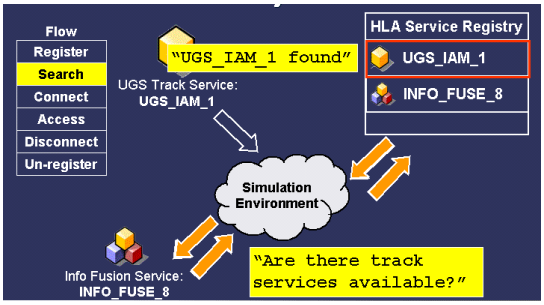
De kit som designats gör att en tjänst på ett enkelt sätt kan fås att uppträda via de ovan listade funktionerna. I figur 30 visas hur två tjänster interagerar med varandra.



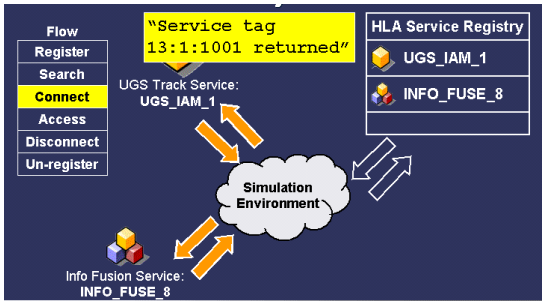
a) Två tjänster och en tjänstekatalog.



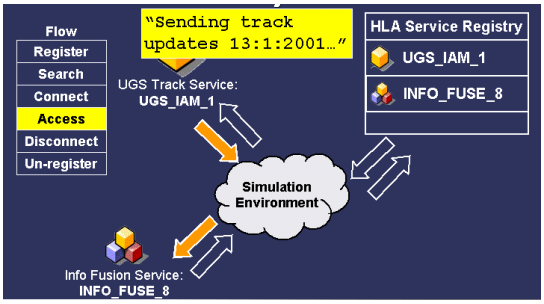
b) Tjänsterna registrerar sig.



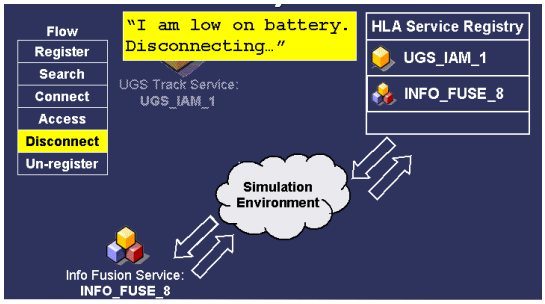
c) En tjänst söker upp en annan.



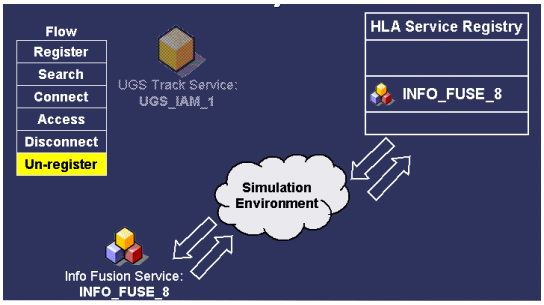
d) En tjänst kopplar upp sig mot en annan.



e) En tjänst levererar data till en annan.



f) En tjänst kopplar ner sig.



g) En tjänst avregistrerar sig.

Figur 30. Två tjänster och en tjänstekatalog (a), tjänsterna registrerar sig (b), den ene söker och hitter den andra (c), den kopplar upp sig mot den andre (d) och får data levererad till sig (e). Sedan kopplar ena tjänsten ner sig (f) och avregistrerar sig från nätet (g).

10.3 Funktion

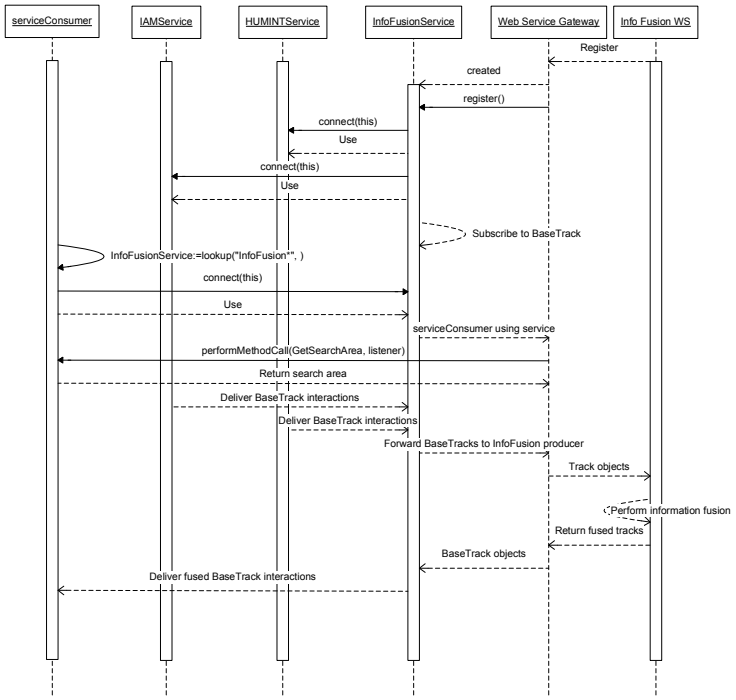
Ett mjukvarubibloitek (Service Kit) för att simulera tjänster i MOSART har i enlighet med planerna implementerats under sommaren och hösten 2005. Utvecklingen har gjorts tillsammans med DSO/DSTA i Singapore inom det LedsystT-finansierade samarbetsprojekt som genomförts mellan sommaren 2004 och hösten 2005. Mjukvarubiblioteket abstraherar det mesta som har med HLA att göra och möjliggör simulering av tjänster, på liknande sätt som *Web Services*, FMA-tjänster och tjänster i *NATO C3 System Architecture Framework* (NAF). Med hjälp av det utvecklade biblioteket kan modeller som finns i MOSART testbädd (t ex sensormodeller, fusionsmoduler, C2-system, etc) enkelt förses med ett tjänstegränssnitt så att experimenterande med att modellera resurser av olika slag som tjänster och koppla ihop dessa via deras tjänstegränsytor kan göras på ett smidigt sätt.

10.4 Demonstration

Tjänstekonceptet demonstrerades i november 2005 på Visionslab i Enköping i samarbete med DSO/DSTA i Singapore. På DSO/DSTA fanns sedan tidigare en Web Services-baserad informationsfusionsmodul, som via en gateway mellan tjänsterna i MOSART testbädd och Web Services-tjänster blev en konsument av målspårstjänster i MOSART testbädd. Dessa målspårstjänster producerades av marksensornätet IAM och en simulerad HUMINT-sensor. IAM- och HUMINT-tjänsterna var implementerade som tjänster i MOSART testbädd med hjälp av Service kit:et.

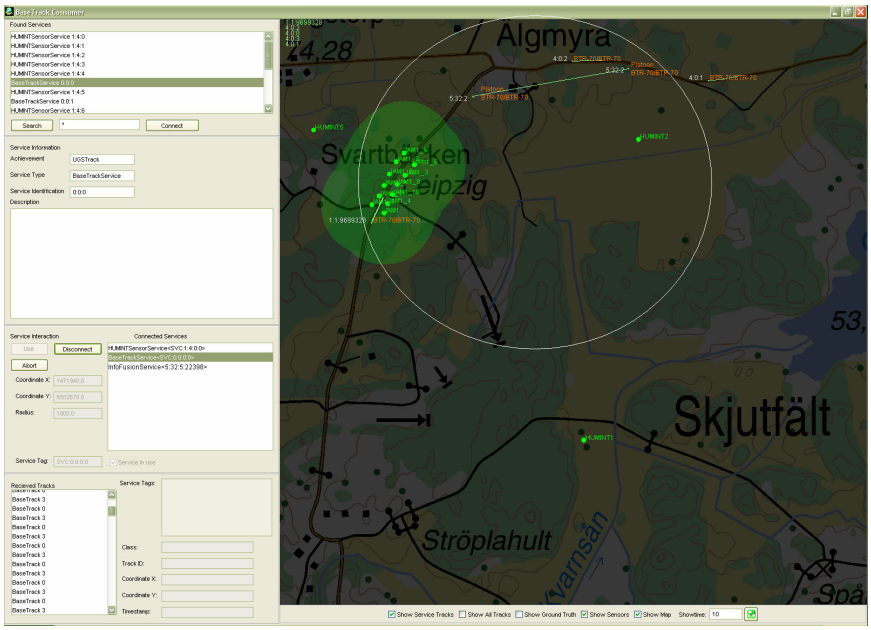
Infofusionstjänsten fusionerar de målspår den får in från de tjänster den konsumerar (IAM och HUMINT). Dessutom får den målspår från Web Services-tjänsten CIN, ett marksensornät utvecklat av DSO. Konsumenten av tjänsten InfoFusionService får aggregerade målspår.

I figur 31 beskrivs konceptuellt hur Infofusionstjänsten interagerar med de andra tjänsterna.



Figur 31. Konceptuell användning av InfoFusionService.

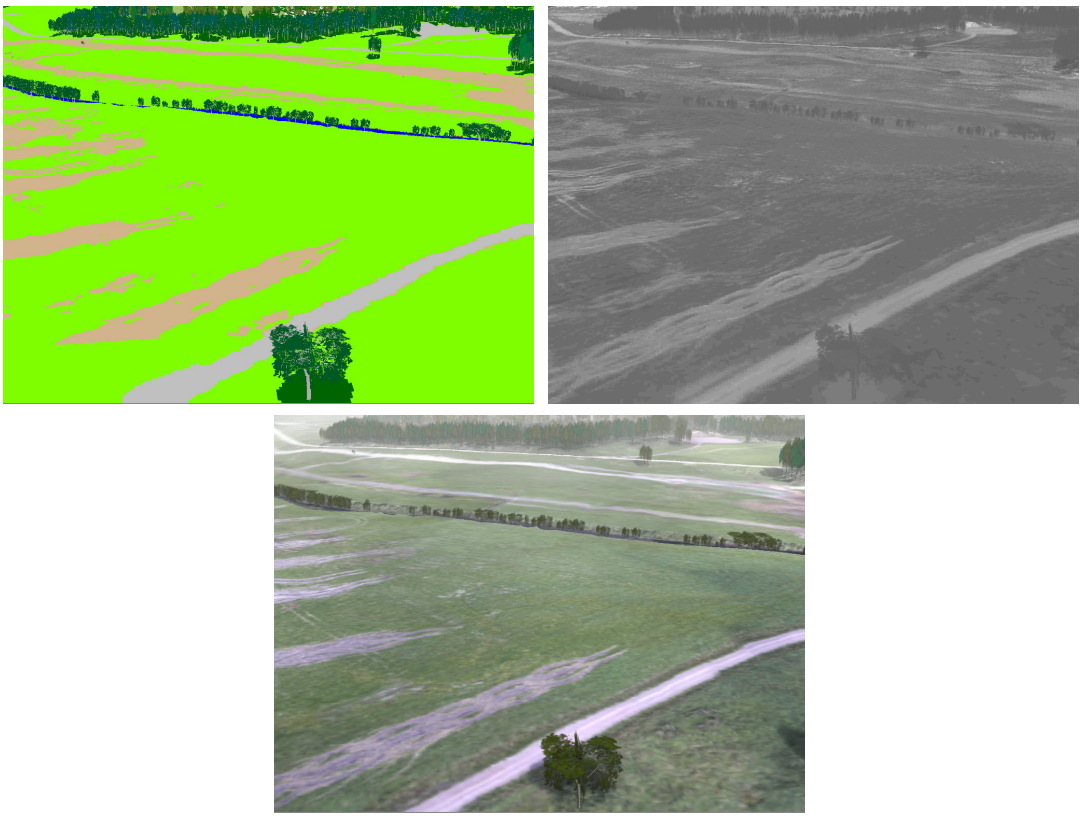
Infofusionstjänsten fungerar som en FMA-tjänst, där en konsument kan börja använda tjänsten utan att ange några parametrar. Infofusionen anropar konsumenten för att få veta vilket område konsumenten är intresserad av. Får den inget svar så levererar den alla fusionerade målspår. En skärmdump från den enkla C2-liknande applikation som utgjorde tjänstekonsument i demonstrationen visas i figur 32.



Figur 32. Aanvändargränssnitt för konsumenttjänsten vid demonstrationen.

11 Omvärldshantering

För att kunna genomföra simuleringar och därefter kvantitativt och kvalitativt validera simuleringsresultatet mot verkliga mätningar krävs en realistisk representation av den naturliga omgivningen med avseende på geometri och sensorspecifika materialparametrar. De omvärldsmodeller som används vid simuleringarna av radar- och IR-sensorer har skapats utgående från data från flygburna lasersystem. Dessa system ger en rumslig upplösning på ca 10-20 punkter/m^2 med en koordinatnoggrannhet på 0.1 meter (i rikets nät), vilket skall jämföras med Lantmäteriverkets standardprodukt med 50 meter mellan höjdvärdena (0.02 punkter/m^2). På FOI Lasersystem har metoder utvecklats för att ur data från flygburna lasermätsystem automatiskt modellera markytan, extrahera information om position, höjd, vidd och trädslag hos vegetation samt rekonstruera byggnader [14]. Genererad information från databehandlingsmetoderna integreras i kommersiella verktyg till omvärldsmodeller med hög geometrisk upplösning. Dessa modeller utökas med information som är relevant för den sensor som skall simuleras, t.ex. materialklassificering av de ingående objekten eller textursättning för en specifik sensor. Materialklassningen används sedan av sensorsimuleringsverktygen, där korrekta fysikaliska parametrar ansätts för varje material. Resultatet blir geografiskt och geometriskt korrelerade omvärldsmodeller, vilket är en förutsättning för att kunna analysera effekter av data- och sensorfusion. Vidare, då omvärldsmodellerna representerar en verklig geografisk plats, är det möjligt att jämföra resultatet från simuleringen med verkliga mätningar.



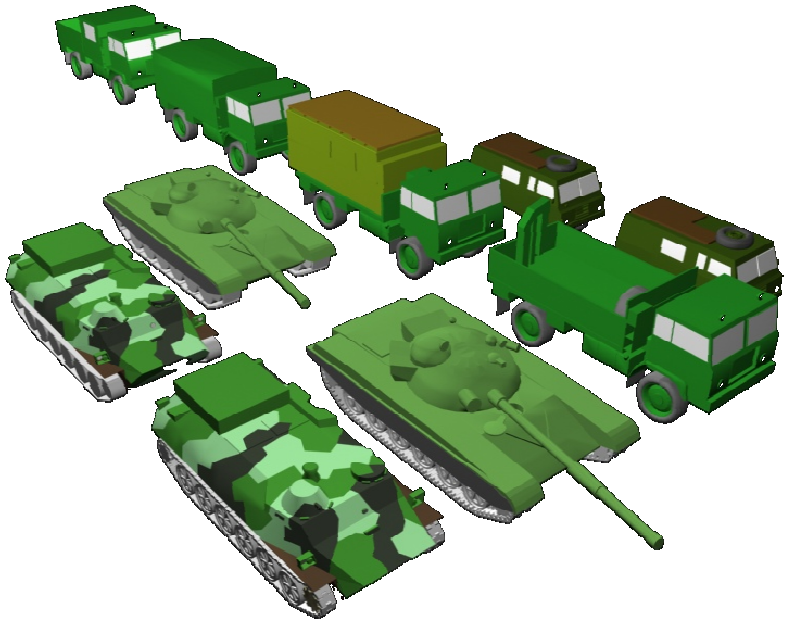
Figur 33. Korrelerade omvärldsmodeller. Ö.v: materialklassad modell för fysikalisk simulering av radar och IR, ö.h: klottertexturerad modell för IR-målsökare, nederst: visuell modell.

Många olika omvärldsmodeller som används av de via MOSART kopplade simuleringarna har genererats. Genereringen har skett utgående från samma data (från högupplösande lasermätsystem), vilket gör att de delar geografiska och geometriska referensramar. Detta underlättar analyser av data fusionerad från de olika simuleringskomponenterna. Se figur 33 för ett exempel på hur korrelerade omvärldsmodeller kan se ut

11.1 Målmodeller

Många simulerade dynamiska scenarion kräver aktörer i form av mål som förflyttar sig eller uppvisar vissa beteenden. I likhet med omvärldsrepresentationen är ett krav att dessa mål skall kunna användas oavsett vilken simulerad sensormodell som används för att observera dem. Detta kräver att målmodellerna har multipla representationer, t.ex. modeller anpassade för simulering av IR, radar eller akustiska sensorer. De målmodeller som användes i de simulerade SEMARK-experimenten [15] måste därför ha representationer för IR, radar, laser och i viss mån akustiska sensorer.

Exempel på några målmodeller som används i SEMARK-projektets simuleringar finns i figur 34.



Figur 34. Målmodeller som användes vid SEMARK-simuleringar (visuella)

11.2 Behov av system för hantering av geografiska data

Då Försvarsmakten går från en centraliserad till en nätverksbaserad struktur innebär detta för hanteringen av geografisk information att även denna förändras. Inte minst beroende på att de ”noder” som utgör nätverket i många fall kan agera ”sensorer” i bred mening. Den information som inhämtas kan vara allt från information om terräng till ren underrättelseinformation. En stor del av denna information kommer även att ha en geografisk koppling. Det geodataunderlag som exempelvis finns i ett ledningssystem bör uppdateras med den nya information som insamlas. Kravet att skapa sig en aktuell lägesbild gör sålunda att denna information måste distribueras i nätverket och henteras på ett konsekvent sätt. Mot bakgrund av dessa nya förutsättningar har verksamhet initierats för att utreda hur dessa data skall representeras och hanteras.

En inriktning för MOSART-projektet är hantering av dessa geografiska data i en distribuerad simuleringskontext, där en HLA-baserad simulering initieras och där varje ingående

simuleringskomponent har ett specifikt geodatabehov. Analogt med FEDEP kan en simuleringskomponents geodatabehov tidigt definieras. Summan av alla ingående simuleringskomponenters geodatabehov utgör sedan den simuleringsgemensamma geodatamängden (jfr FOM). Detta sammansatta behov av geografisk information skall sedan tolkas och hanteras. Syftet är att kunna extrahera denna datamängd ur en ”master environment database” och sedan distribuera dessa data så att simulatorspecifika omgivningsmodeller kan skapas för användning i den distribuerade simuleringen. Inom simuleringen skall sedan geodatahanteraren se till att förändringar i omgivningsinformationen distribueras bland simuleringskomponenterna.

Ett vidare syfte med denna geodatahantering är att de omgivningsmodeller som extraherats på detta sätt har en mycket hög grad av geografisk korrelation.

11.3 Geodatabas

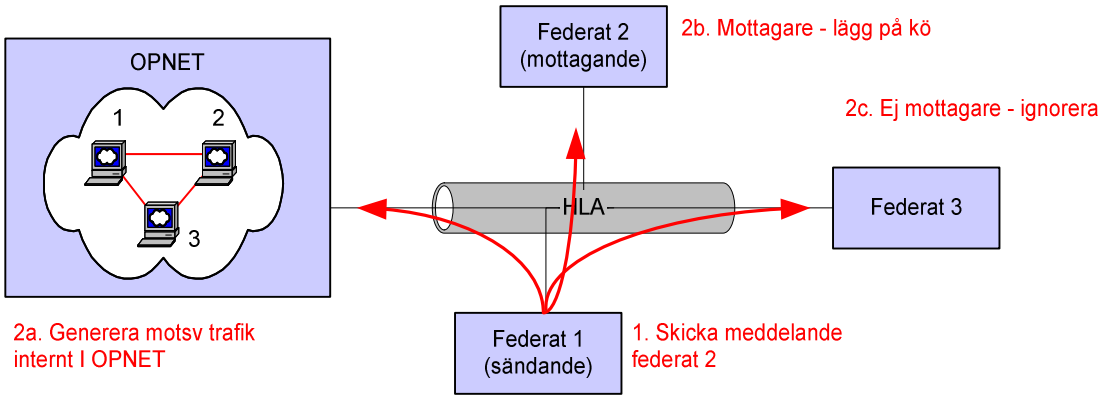
Ett system för hantering av geografiska data i en databas är under uppbyggnad. Detta system skall stödja geodatahantering för simulering samt kunna hantera dynamiska förändrningar i den geografiska informationen på ett konsistent och oberoende sätt. Denna verksamhet bedrivs inom FOI-projektet Dynamisk terräng [16].

12 Simulering av kommunikation

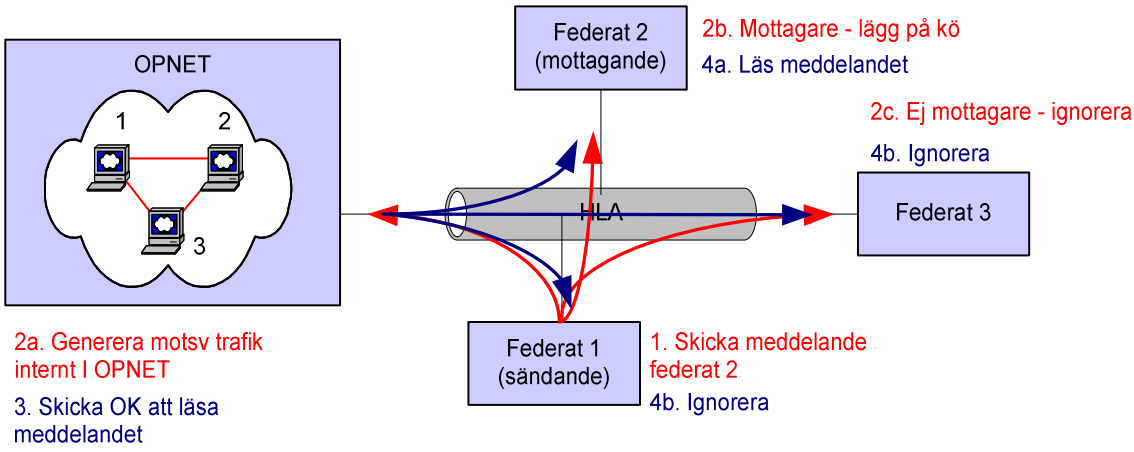
MOSART-miljön har än så länge mest använts till simulering av plattformar, sensorer och fusionsalgoritmer, sk verksamhetsimulering. I många fall utelämnas simulering av kommunikationen mellan olika noder (plattformar, sensorer, etc) utan den förutsätts vara optimal då fokus inte ligger på just kommunikation. I vissa fall kan det vara intressant att se vilka begränsningar kommunikation sätter på fusionsalgoritmer, distribution av sensordata och målspår i nätverk, mm. Samma frågeställning finns inom simulering av kommunikation (mha t ex OPNET) där man vill ha realistisk påverkan från verksamheten, t ex förflyttning av mobila radionoder, mer dynamisk datatrafik på nätverket. Om man kunde knyta ihop dessa simuleringsmiljöer för simulering av verksamhet och simulering av kommunikation i en och samma miljö skulle dessa effekter kunna börja studeras. Det är dessa tankar som drev fram ett pilotprojekt där en integration gjordes mellan OPNET och MOSART.

12.1 Integration MOSART-ISS

Under hösten 2004 gjordes ett försök att integrera MOSART-miljön på FOI med ISS-miljön (InfraStrukturell Simulering) på Visionslab tillsammans med Aerotech Telub. Detta pilotförsök demonstrerades i januari 2005 på Visionslab och erfarenheterna finns i form av en rapport [17]. I försöket användes OPNET som en kommunikationsfederat som beräknar fördröjningar och paketförlust på meddelanden genom nätverket. Från en operatörsvy kunde bilddata begäras från en eller flera bildalstrande sensorer. Effekterna av kommunikation (fördröjning och paketförlust) kunde då åskådliggöras i operatörsvyn. I figur 35 och 36 visas en schematisk bild över hur kommunikationen är tänkt att fungera. Nätverkskommunikation modelleras i HLA som interaktionen *RealWorldMessage* som OPNET prenumererar på. När en federat vill skicka ett meddelande så skickas ett *RealWorldMessage* (egentligen en subklass till *RealWorldMessage*) till alla federater som prenumererar på denna interaktionsklass. Mottagande federat (2) lägger meddelandet på en kö för senare läsning. OPNET tar i samma stund emot meddelandet och genererar motsvarande trafik internt. När meddelandet har kommit fram internt till nod 2 så skickas en HLA-interaktion *ReadyToRead* ut. Federat 2 tar emot denna interaktion och får därefter läsa det ursprungliga meddelandet som var lagt på kö (figur 36).



Figur 35. Federat 1 sänder ett meddelande till federat 2. OPNET simulerar motsvarande trafik internt.



Figur 36. OPNET svarar när det simulerade meddelandet har kommit fram. Mottagande federat 2 får läsa meddelandet.

I försöket användes OPNET:s tilläggsmodul för HLA. Under försöket upptäcktes brister i denna HLA-modul, framförallt det sätt som eventhanteringen och uppstegning av tid sköttes på vilket medförde att väldigt låga prestanda uppnåddes. En föreslagen åtgärd är att utveckla en egen HLA-modul som man har kontroll över och som är anpassad till den FOM och de federationsöverenskommelser man har i sin HLA-federation. Detta arbete behöver inte vara så stort med tanke på alla de mjukvarubibliotek som är framtagna för stöd vid federatutveckling (se kap 5).

13 Data repository

13.1 Bakgrund

Inom FOI har länge funnits ett behov av att kunna sammanställa information om de sensordata som samlas in vid de mätkampanjer som olika projekt genomför. Detta skulle underlätta återanvändning av tidigare insamlade data och allmänt underlätta hanteringen av de data som samlats in.

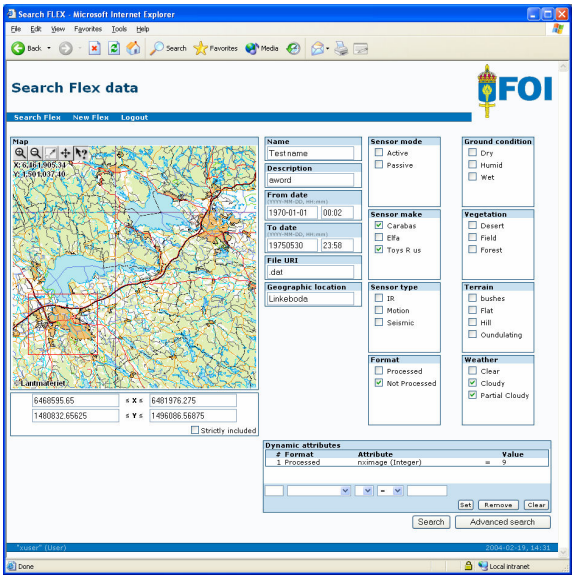
13.2 Funktion

MOSART-projektet har anvarat för upphandlingen av experimentdatabasen FLEX, som lagrar metadata om sensordata, så som tidpunkt när data inmättes, typ av data, plats för data och så vidare. Den pekar också ut den verkliga platsen där sensordata ligger lagrat. Detta möjliggör ordnad arkivering, sökning och hämtning av sensordata. Enkla sensordata i form av bilder och filmer kan man direkt visualisera från databas formuläret.

För mer avancerad användning finns adapter för Java, C++ och MATLAB, där man direkt kan interagera med databasen.

FLEX är en webbapplikation som finns på FOI intranät i vilken man kan logga in med hjälp av godtycklig webbrowser för att regisrera, söka efter eller hämta hem sensordata, etc.

I figur 37 visas en skämdump som visar FLEX söksida. Figur 38 visar en skärmdump från sökresultatsidan i FLEX.



Figur 37. Sida för sökning i FLEX.

Figur 38. Sökresultat i FLEX.

14 Dokumentation

Denna rapport utgör en övergripande dokumentation av MOSART-projektet och den verksamhet som genomförts med fokus på den mjukvara som utvecklats och vad denna kan användas till. För att kunna utveckla nya och ansluta gamla komponenter till MOSART testbädd krävs dock mer detaljerad teknisk information än vad som ryms i denna rapport.

Den tekniska dokumentation som behövs för den som ska utveckla mjukvara med MOSART testbädd som plattform finns i följande former:

- **Denna rapport (och i viss mån tidigare projektrapporter):** Här finns övergripande information om MOSART-projektet inklusive testbädden och dess delar.
- **MOSART hemsida på Internet (www.foi.se/MOSART/):** Här finns övergripande information om MOSART-projektet inklusive testbädden och dess delar. I början av 2006 kommer dessutom information om hur organisationer utanför FOI kan få tillgång till mjukvaran i testbädden att läggas ut.
- **MOSART hemsida på FOI intranät (www-int.foi.se/MOSART/):** Här finns övergripande information om MOSART-projektet inklusive testbädden och dess delar. Här finns även möjlighet att ladda ner både färdigkompilerade versioner av alla kit och moduler samt all den källkod som tagits fram inom projektet. Dessutom finns API-dokumentation för alla kit tillgänglig i HTML.
- **MOSART Wiki på FOI intranät (fusion.foi.se/mosart/):** Här finns detaljerad information för den som vill utveckla mjukvara med MOSART testbädd som plattform. Informationen på Wiki:n är normalt av ganska dynamisk karaktär och kan på ett enkelt och smidigt sätt behöva uppdateras oftare än annan information, därav användandet av en Wiki. Informationen på Wiki:n inkluderar information såsom hur man konfigurerar sin dator för att utveckla mjukvara i MOSART testbädd, hur man kommer åt MOSART CVS, hur man utvecklar nya plugins till MIND och Vega, vanliga frågor som uppstår när man utvecklar mjukvara i MOSART-miljön, etc.
- **MOSART CD-skiva:** Vi planerar att sätta ihop en CD-skiva med den mjukvara som utgör testbädden i färdigkompilerad form. Troligen skapar vi också en version av CD-skivan där all källkod som utvecklats i projektet dessutom finns tillgänglig. På CD-skivan ska även all API-dokumentation finnas samt ett snapshot av informationen på MOSART Wiki:n. På så vis blir CD-skivan en komplett utgåva av MOSART testbädd. De kommersiella komponenter som används i testbädden kommer självklart inte finnas med på skivan utan får anskaffas separat. Den enda kommersiella mjukvara som normalt sett måste anskaffas är en RTI. Vi använder oss i första hand av (och rekommenderar därför) Pitch pRTI. Nya versioner av CD-skivan kan självklart komma att släppas efterhand. I kapitel 15 beskrivs hur MOSART testbädd är tänkt att göras tillgänglig för andra organisationer än FOI. Det är via detta förfarande CD-skivan kommer göras tillgänglig. Informationen på CD-skivan avses även göras tillgänglig för direkt nerladdning över Internet, även om denna nerladdningsfunktion inte finns idag.

15 Tillgänglighet

När projektet under 2004 och 2005 presenterats för FM och industri har intresse väckts för att installera och utnyttja den programvara som utvecklats inom MOSART-projektet. Eftersom det finns intresse från FM och FMV och även från andra externa parter att använda sig av denna programvara och det ligger i FOI:s intresse att programvaran sprids och används i t ex FM relaterade projekt, så har projektet under 2005 undersökt möjligheterna att göra programvaran tillgänglig – via distibution med CD-ROM eller via Internet.

Utgångspunkten är:

- att ett urval av programvaran görs tillgänglig för extern nerladdning via Internet eller via distribution med CD-ROM
- att access för nerladdning skyddas av användar-ID och lösenord
- att FM och FMV generellt kan använda den nerladdade programvaran för det svenska totalförsvarets behov enligt gällande avtal
- att programvaran av övriga organisationer kan testas på prov utan ersättning till FOI enligt på extern websida publicerat generellt licensavtal i en månad efter nerladdning
- att övriga organisationer en månad efter nerladdningen skall ha tecknat ett skriftligt avtal med FOI angående användningen

Ambitionen är att programvaran (inkl. källkod om det behovet finns) skall kunna användas utan kostnad i befintligt skick och även integreras med produkter från industriella leverantörer. Eftersom det idag inte går att förutse vilka aktörer som vill använda programvaran eller i vilka produkter FOIs programvara kan komma att integreras, finns dock en reservation vad gäller licenskostnaden för aktörer och slutanvändare som inte har samarbete med svenska FM eller FOI. Tills vidare kommer därför varje (licensierad) användning att hanteras och beslutas enskilt.

Licenstagaren får göra kopior av, ändra och bearbeta *Programvaran* och även integrera *Programvaran* i annat datorprogram under förutsättning att det finns en tydlig upplysning om att *Programvaran* är upphovsrättsligt skyddad av FOI. Eventuell licenskostnad regleras i ett tilläggsavtal till licensavtalet.

Dessutom:

- måste den licensierade programvaran vara en integrerad del av annat datorprogram (applikation) utan att utgöra applikationens huvudsakliga beståndsdel
- får Programvarans API (Application Program Interface) inte vara synligt för användaren
- får inte källkoden göras tillgänglig
- skall, om *Licenstagaren* är befriad från licensavgift, någon ersättning ej betingas för *Programvaran* (den del som licensierats av FOI) av *Licenstagaren* gentemot tredje part.

En viktig aspekt är att licenstagaren rapporterar fel, brister, rättningar, ändringar i *Programvaran* till FOI.

En installation av programvaran finns i FM Visionslab i Enköping. Den tänkta användningen är som stöd för utvecklingen av demonstationer och experiment.

Referenser

[1] IEEE Std 1516-2000. ”IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)”, Institute of Electrical and Electronics Engineers, Inc (IEEE), 2000.

[2] MOSART:s hemsida (http://www.foi.se/MOSART/)

[3] M. O’Conner, A. Faier. ”Real-Time Platform Reference Federation Object Model (RPR FOM), Version 2.0 Draft 14”, Simulation Interoperability Standards Organization (SISO), 2002.

[4] J. Fischer, B. Case, R. Bertin. ”Guidance, Rationale, and Interoperability Manual for the Real-time Platform Reference Federation Object Model (RPR FOM), Version 2.0D10v1”, Simulation Interoperability Standards Organization (SISO), 2002.

[5] FLAMES homepage (http://www.ternion.com)

[6] M. Tyskeng. ”MOSART – Instructions for use”, FOI-R—1098—SE, appendix F, 2003.

[7] M. Morin. “Multimedia representations of distributed tactical operations”, Linköping studies in science and technology, Dissertation No. 771, Linköping University, 2002.

[8] M. Morin. “MIND—Methods and Tools for Visualization of Rescue Operations”, In Procs of The International Emergency Management Society´s Eighth Annual Conference (TIEMS’2001), 2001.

[9] A. Törne, T. Horney, M. Brännström, P. Hörling, L. Tydén.”Projektutnyttjande MOSART”, FOI MEMO 1128, 2004.

[10] M. Holmberg, A. Lauberts, R. K. Lennartsson. ”Slutrapport för projektet interaktiva adaptiva marksensornät (IAM)”, FOI-R—1450—SE, 2004.

[11] L. Tydén, C. Hedberg, M. Petersson, H. Andersson, C. Wigren, M. Nordin, L. Festin, N. Appleby, D. Haavisto, P. Svensson, J. Bergman, S. Olsson. ”Slutrapport Duellsimulering Telekrig”, FOI-R—1775—SE, 2005.

[12] M. Andersson, M. Nordin, M. Folkesson, R. Forsgren, P. Klum, A. Lauberts, A. Eneroth. ”Samverkande sensorer i en telekrigstillämpning – undertryckning av falska mål med datafusion”, (to be published).

[13] J. Fransson, F. Lantz. ”Negativ information från markspaning”, FOI MEMO 1445, 2005.

[14] S. Ahlberg, M. Elmqvist, Å. Persson, U. Söderman. "Metoder för framställning av högupplösta syntetiska omgivningar/omvärldsmodeller för sensorsimulering", FOI-R—1110—SE, 2003.

[15] P. Grahn, T. Horney, P. Follo. ”Demonstration från de första resultaten av simulering av samverkande sensorer”, FOI MEMO 1439, 2005.

[16] S. Ahlberg et al. ”Omvärldsmodellering/Dynamisk terräng årsrapport 2005”, (to be published).

[17] P. Hörling et al. ”Integration MOSART-ISS: En försöksintegration av OPNET med MOSART”, FMV Ref.ID. LT1O E04-0648, 2005.