FOI-R--1098--SE
December 2003
ISSN 1650-1942
Technical report

Mathias Tyskeng

MOSART - Instructions for use

Command and Control Systems
SE-581 11 Linköping

SWEDISH DEFENCE RESEARCH AGENCY

Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

FOI-R--1098--SE

December 2003

ISSN 1650-1942

Technical report

Mathias Tyskeng

MOSART - Instructions for use

Issuing organization FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	**Report number, ISRN** FOI-R--1098--SE	**Report type** Technical report
	Research area code 4. C4ISR	
	Month year December 2003	**Project no.** E7835, E7084
	Customers code 5. Commissioned Research	
	Sub area code 41 C4I	
Author/s (editor/s) Mathias Tyskeng	**Project manager** Mathias Tyskeng	
	Approved by	
	Sponsoring agency FM/FMV	
	Scientifically and technically responsible Mathias Tyskeng	

Report title

MOSART - Instructions for use

Abstract (not more than 200 words)

Within project MOSART a simulation testbed for integration of research results has been developed. The purpose is to enable integration of results from different research areas at FOI as well as from external partners.

This report is a compilation of FOI memos that describe how the testbed is intended to be used. It consists of the following parts:

- HLA framework
- Module and demonstrator library
- Data repository
- MIND, 2D visualisation
- Vega, 3D visualisation
- Flames, scenario engine and editor

Keywords

Testbed, integration, HLA

Further bibliographic information	**Language** English
ISSN 1650-1942	**Pages** 49 p.
	Price acc. to pricelist

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 Linköping	FOI-R--1098--SE	Teknisk rapport
	Forskningsområde 4. Spaning och ledning	
	Månad, år December 2003	**Projektnummer** E7835, E7084
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 41 Ledning med samband och telekom och IT-system	
Författare/redaktör Mathias Tyskeng	**Projektledare** Mathias Tyskeng	
	Godkänd av	
	Uppdragsgivare/kundbeteckning FM/FMV	
	Tekniskt och/eller vetenskapligt ansvarig Mathias Tyskeng	

Rapportens titel (i översättning)

MOSART - Instruktioner för användning

Sammanfattning (högst 200 ord)

Inom projekt MOSART har en testbädd för integration av forskningsresultat utvecklats. Syftet är att möjliggöra integration av resultat från olika forskningsområden på FOI och resultat från externa partners.
Denna rapport är ett sammanhållande dokument för de FOI memo som beskriver användning av testbädden i MOSART. Dessa omfattar följande delar:

- HLA framework
- Module and demonstrator library
- Data repository
- MIND, 2D visualisering
- Vega, 3D visualisering
- Flames, scenariomotor och -editor

Nyckelord

Testbädd, integration, HLA

Övriga bibliografiska uppgifter	**Språk** Engelska
ISSN 1650-1942	**Antal sidor:** 49 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Contents

1 Introduction

This report describes the different parts of the MOSART testbed and how it can be used by other projects. The purpose is to support user projects in understanding how the MOSART testbed works and how it can be useful to them.

2 MOSART documents

The different parts of MOSART are presented in FOI memos, which are briefly described in this section.

2.1 HLA framework

This document describes the integration framework used in the MOSART testbed, see Appendix A “HLA framework – Instructions for use”.

2.2 Module and demonstrator library

This document describes the MOSART module and demonstrator library, which is a storage for developed modules and demonstrators in the MOSART testbed, see Appendix B “Module and demonstrator library – Instructions for use”.

2.3 Data repository

This document describes the MOSART data repository FLEX, which is a management system for handling of sensor data at FOI, see Appendix C “Data Repository – Instructions for use”.

2.4 MIND – 2D visualization

This document describes the 2D visualization module MIND, which is used in the MOSART testbed, see Appendix D “MIND – Instructions for use”.

2.5 Vega – 3D visualization

This document describes the 3D visualization module Vega, which is used in the MOSART testbed, see Appendix E “Vega – Instructions for use”.

2.6 FLAMES – scenario engine and editor

This document describes the simulation framework Flames, which can be used as a scenario engine and editor in the MOSART testbed, see Appendix F “FLAMES – Instructions for use”.

MOSART
Modeling and simulation for analysis and research test-bed

MOSART – frameworks

HLA framework – instructions for use

FOI Memo 03-2761:1

Author: Mathias Tyskeng **E-mail:** mailto:mattys@foi.se **Telephone:** +46 13 37 85 89	**Coauthors:**
Contents: Describes the MOSART HLA framework	**Version:** 1.0 **Date:** 2003-12-12
Filename: instructions_for_use_HLA.doc **Folder:** \\Filer-lin\MOSART\ projects\MOSART\docs\	

CHANGE HISTORY

VERSION	DATE	CHANGES
0.1	031112	Initial version
1.0	031212	Final version

Contents

1 Introduction

1.1 Document purpose

This document is a short introduction to the HLA framework and how it can be used within FOI. It is not intended to be a detailed introduction to HLA (can be found in [1] and [2]), instead the intention is to introduce typical HLA features, FOI integration software and give examples how HLA is used at FOI.

2 The HLA framework

HLA (High Level Architecture) is a standard architecture for distributed simulation similar to other approaches in the area such as DIS (Distributed Simulation). In this section, an introduction to HLA will be given, but without detailed technical explanations.

2.1 Overview

A simulation setup in HLA is called a federation and it consists of two or more federates and a runtime infrastructure (RTI). A federate is a software application that runs on a computer connected to the network where the simulation takes place. A computer can host one or several federates depending on performance requirements. The RTI is software that coordinates all data exchange in the simulation and it runs on one computer in the simulation network. A federation is characterized by its Federate Object Model (FOM), from which a Federation Execution Data (FED) file can be derived. The FOM (and the FED) describes what data that can be exchanged between federates in the federation. Data that is not named in the FOM is allowed to be exchanged, trying to do so will cause an error. Each federate should have a Simulation Object Model (SOM), which defines all objects a federate might expose to other federates. The SOM is a part of the FOM.
The interface between the RTI and a federate is called ambassador, see Figure 1. The RTI exposes an interface that is called RTI Ambassador, which is used by a federate for all communication from the federate to the RTI. Communication from the RTI to the federate is handled by the interface that the federate must expose, a Federate Ambassador. How the communication is implemented is not part of the standard and therefore different implementations of the RTI may use different communication principles.

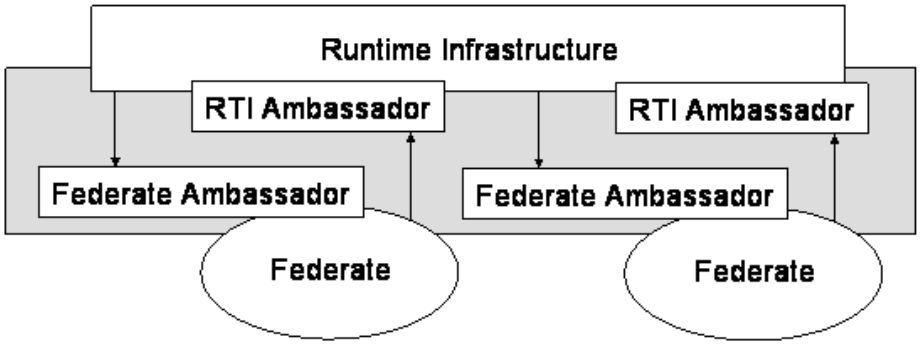

Figure 1. HLA interface between the runtime infrastructure (RTI) and connected federates.

All regular data exchange in HLA is based on publication and subscription. Hence, a federate will publish data that it wants provide other federates with and subscribe to data that it wants to be provided with. Data that is exchanged is always objects with different attributes in an object oriented inheritance structure. A subscription will only result in received data if other another federates has published the object type, registered an instance of the object and provided value updates of the attributes in the object. Hence, there is no guarantee that a subscription to an object class will result in received data.

Time Management is also a service that is provided by HLA. Time management is crucial in many simulations when all federates must have the same simulation time at all times. A federate can be time regulating, i.e. no other federate is allowed to step time before this federate steps its time. A federate can also be time constrained, i.e. that this federate cannot step its time before all the other federates in a federation step their time. Any combination of regulating and constrained is allowed.
This is only a very brief description of HLA. A more thorough introduction to HLA can be found in [2].

2.2 MOSART overview

The MOSART integration testbed has support for integration of software via HLA. In Figure 2, the basic functionality of MOSART is illustrated together with software modules that have been developed by projects other than MOSART.

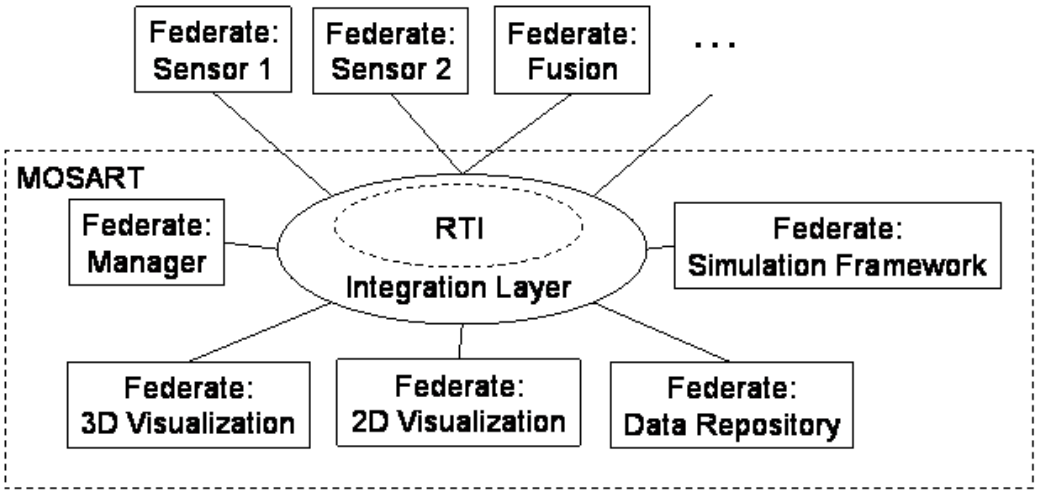

Figure 2. Overview of the MOSART integration testbed.

The testbed contains different RTIs and software for integration (kits) of modules produced in other projects than MOSART. This will be further investigated later in this document. MOSART is built up by the RTI, the integration software and a handful of federates that is useful in most simulations and demonstrators:

- Manager federate: Handles coordination of one or several runs of a simulation in a federation.
- 3D visualization federate: Visualizes objects and the environment in three dimensions. This federate is using Vega [3] for visualization.
- 2D visualization federate: Visualizes objects and the environment in two dimensions. This federate is using MIND [4] for visualization.
- Data repository federate: A tool for management of large amounts of sensor data that can be used in a simulation.
- Simulation framework federate: Controls the object movements in a simulation. One such federate is using the simulation framework FLAMES.

Federates that other projects produce and integrate with MOSART will be kept in a module library so that other projects can use them if they fit the requirements. The same is true for demonstrators that projects produce.

2.3 RTI

The Runtime Infrastructure (RTI) is the software that is responsible for data exchange in a HLA simulation. Since HLA is a standard and not an implementation, there are several different

implementations of the standard. These are not always compatible with each other. Therefore, MOSART supports three different implementations of the RTI: DMSO, Pitch and MÄK.
Contact the persons in the MOSART knowledge network for information about which implementation of the RTI to use.

2.4 Federates

In the MOSART testbed there are some federates that are provided as a baseline resource. These are: manager federate, 3D visualization federate, 2D visualization federate, data repository federate and simulation framework federate.
The manager federate is a federate that synchronizes the federates in a federation during a simulation cycle. The requirement on the federates in a federation which uses the manager synchronization mechanism is that it uses the so called MFAKit provided by MOSART. The manager will notify the federates which scenario that should be used in the simulation, synchronizing startup and finish of the simulation as well as multiple runs of a simulation.
The 3D visualization federate is based on Vega, which has been extended with an HLA connection so that it can receive data to visualize in a simulation. This federate also visualizes high resolution environment models developed at FOI, which can be used in the simulations with MOSART.
The 2D visualization federate is based on MIND, which has been extended with an HLA connection so that it can receive data to visualize in a simulation. MIND can display a lot of objects besides platforms, e.g. paths, sensor coverage and tracks estimates.
The data repository federate is under development and will be a stand alone application to start with. The data repository is a searchable database of sensor meta data that makes it possible to find sensor data located at different locations in the FOI intranet. This application will be a federate which makes it possible to find suitable sensor data for a simulation during the simulation instead of before the simulation.
The simulation framework federate is an application where objects in a simulated scenario can be handled. This can be a tool that is developed within MOSART which handles less complex scenarios as well as a commercial tool with a HLA connection such as Flames which handles very complex scenarios
All these federates are baseline functionality of the MOSART testbed and are accessible to all projects that want to build demonstrators in MOSART. In some cases MOSART provides an HLA connection to an application and the project provides the application itself.
Besides the federates of the baseline functionality there are federates that are developed by other projects. These federates are stored in the MOSART Module Library and can be used by other projects with or without modifications, e.g. sensor modeles. Complete demonstrators are stored in a similar way in the MOSART Demonstrator Library.

2.5 Object Models

In HLA a very central part is the Object Model Template (OMT) which is a structure to describe the information model in a federation. The OMT prescribes the structure of a Federate Object Model (FOM), which is a description of the information that can be exchanged in a federation. From the FOM a Federate Execution Data (FED) file is derived which is used by the RTI in the simulation. For each federate there is a Simulation Object Model (SOM), which is nothing else than a document which describes the information that the federate will expose in the simulation. The information in the SOM should be present in the FOM together with the information of other SOMs.
In MOSART there are a number of standard FOMs that are used for federations in the testbed. The reason for this is to have relatively compatible federates in the testbed library.

The documentation of every federate must contain information about which FOM that it is compatible with.

2.6 HLA and MOSART

In this section an introduction is given to the integration software that has been developed in the MOSART project to simplify the use of HLA for other projects. This software is called kits and is available for both Java and C++. The purpose of the kits is to hide most HLA specific behavior to a user so that the focus can be on the own module and not the integration with MOSART

2.6.1 Java/CppHLAKit

This kit is the basic kit that all federates should use for integration with the MOSART testbed. It contains an implementation of a Federate Ambassador that will handle all callbacks from the RTI without any extra programming by the federate developer. Furthermore, a mechanism for transformation of Java/C++ objects into HLA objects is provided so that all information exchange is made transparent to the user. The only requirement on the user is to use the implemented HLAObject as base class for all objects that are present in the exchange of information in the simulation. The kit also contains a sample federate.

For information and software, contact the MOSART Knowledge Network.

2.6.2 Java/CppRPRKit

This is a kit which supports the RPR (pronounced reaper) FOM. The RPR-FOM contains a large number of objects that can be used in a simulation. The RPR-kit also contains coordinate transformations and deadreckoning algorithms that should be used with the RPR-FOM.

For information and software, contact the MOSART Knowledge Network.

2.6.3 Java/CppMFAKit

This kit is called MOSART Federation Agreement (MFA) and is the integration software that supports the functionality of the manager federate. The implementation of a Federate Ambassador in this kit inherits from the Federate Ambassador of the basic HLA kit and is extended with necessary functionality for synchronization and scenario handling. The kit also contains a skeleton federate that can be used for module implementation.

For information and software, contact the MOSART Knowledge Network.

2.6.4 Java/CppGeomKit

The Java Geom Kit is an API that takes care of things like transformation between <latitude, longitute, altitude> and geocentric <X, Y, Z> using different ellipsoids. Various projections such as UTM and RT-90 are also supported.

For information and software, contact the MOSART Knowledge Network.

3 Using HLA at FOI

In this section an introduction is given to how HLA is used in the MOSART testbed. Two very brief examples are described and illustrated. The first example is a federate which uses the basic HLA kit to integrate with MOSART. The important issues in this example is that the simulated country, in order to be an object in the HLA world, has to be derived from the HLA kit basic object HLA object. This way, a change in attribute value in the country object will result in an update of the attribute value for that country in every other federate that subscribes to country objects. All RTI communication is handled by the HLA kit object HLAFederateAmbassador that the federate must create an instance of. If the federate subscribes to Country as well, a country object is created automatically and all updates of other country object attributes in other federates will be present in the country object.
In this example the user must include the HLA kit in the application.

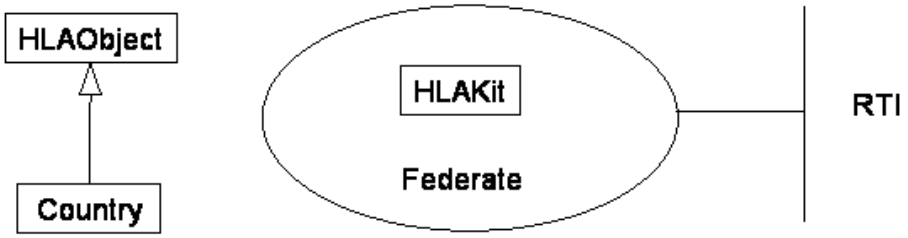

Figure 3. Example 1

The second example is a federate which uses the Manager synchronizing function. The underlying object handling is identical to the first example. The difference is that all communication with the RTI is handled by an instance of MFAKitFederateAmbassador which will handle all synchronization, using interactions. This is however transparent for the user of the kit.
In this example the user must include the HLA kit and the MFA kit in the application.

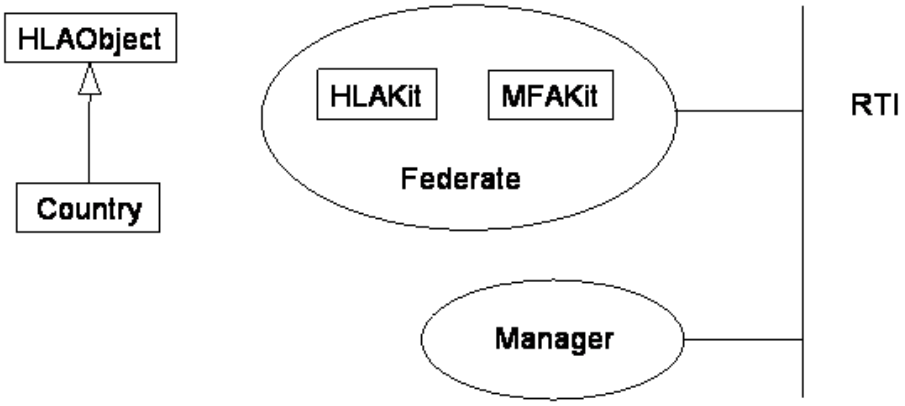

Figure 4. Example 2

The third example uses the RPR kit and the difference from the first example is that the object that is simulated (Car) is derived from the RPR specified object GroundVehicle, which itself is derived from HLAObject in order to use the automatic object and attribute handling. This federate can also use the coordinate transformations in the Geom-kit and dead reckoning algorithms in the RPR kit if necessary.
In this example the user must include the HLA kit and the RPR kit in the application.

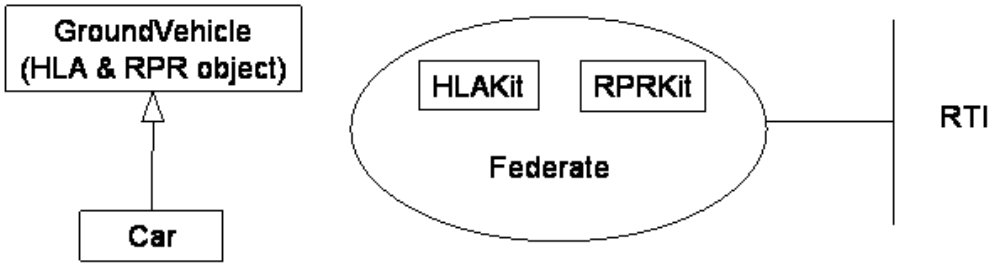

Figure 5. Example 3

4 References

[1] HLA, High level architecture (www.dmso.mil/public/transition/hla)

[2] Kuhl, F., Weatherly, R., Dahmann, J., Creating computer simulation systems – An introduction to High Level Architecture, Prentice Hall PTR, 2000.

[3] Vega, Multigen-Paradigm Inc. (www.multigen.com/)

[4] MIND, FOI, Div. of C2 Systems, Dept. of Systems analysis and IT security

MOSART
Modeling and simulation for analysis and research test-bed

MOSART – module and demonstrator library

Instructions for use

FOI Memo 03-2761:2

Author: Mathias Tyskeng **E-mail:** mailto:mattys@foi.se **Telephone:** +46 13 37 85 89	**Coauthors:**
Contents: Describes the MOSART module and demonstrator library	**Version:** 1.0 **Date:** 2003-12-12
Filename: instructions_for_use_component_and demonstrator_library.doc **Folder:** \\Filer-lin\MOSART\ projects\MOSART\docs\	

CHANGE HISTORY

VERSION	DATE	CHANGES
0.1	030922	Initial version
1.0	031212	Final version

Contents

1 Introduction

This document describes processes related to the demonstrator and module library within MOSART. The intention is to provide a set of checklists for projects using the library resources of MOSART.

2 The demonstrator library

The intention with the demonstrator library is to store demonstrators using the MOSART testbed to enable reuse etc. This is a source for projects where they may find a demonstrator that they can alter and thereby do not have to build a whole new demonstrator.

2.1 Requirements

The requirements on the demonstrator library (found in **Fel! Hittar inte referenskälla.**) are as follows:

1. All modules in the demonstrator must be found in the module library.
2. The demonstrator shall be documented (functional description, user's manual and installation manual).
3. Any platform specific behavior of a demonstrator must be documented in the installation manual or a readme-file.
4. The demonstrator FOM must be present in the library.
5. All demonstrators in the demonstrator library shall be tested.
6. All demonstrators in the demonstrator library shall use configuration management.
7. All demonstrators in the library shall be included in a table of contents that is associated with the demonstrator library. Every demonstrator shall have a representative name and a short description that gives a picture of the overall function of the demonstrator for a potential user.

2.2 Usage processes

Here is a checklist that is useful when someone wants to use a demonstrator in the demonstrator library:

1. Get the latest table of contents of the demonstrator library from a MOSART contact person.
2. Browse through the table of content to find the demonstrator(s) that is suitable (in the current form or with changes) for usage in the project.
3. If one or several useful demonstrator are found in the library, request documentation and software for the demonstrator in question.

This checklist is useful when someone wants to provide a new demonstrator to the module library:

1. Make sure that the requirements of the demonstrator library are fulfilled for the demonstrator that is to be added to the library. This is particularly important when testing and documentation are concerned.
2. Add the new demonstrator through a MOSART contact person by providing software and documentation for the demonstrator.
3. Update the demonstrator library table of contents with the new demonstrator.

3 The module library

The intention with the module library is to store modules that can be used within the MOSART testbed to enable reuse etc. This is a source for projects where they may find modules that they can reuse as they are or alter and thereby do not have to build all modules needed fro their simulations and/or demonstrators.

3.1 Requirements

The requirements on the module library (found in **Fel! Hittar inte referenskälla.**) are as follows:

1. A module in the module library shall (if possible) contain only one service.

2. All software in the module library shall be documented (functional description, user's manual and installation manual).
3. All software in the module library shall be well tested.
4. Any platform specific behavior of a module must be documented in the installation manual or a readme-file.
5. All software in the module library shall use configuration management.
6. All modules in the library shall be included in a table of contents that is associated with the module library. Every module shall have a representative name and a short description that gives a picture of the overall function of the module for a potential user.

The requirements on contributors to the module library are to fulfill the requirements of the module library.

3.2 Usage processes

Here is a checklist that is useful when someone wants to use a module in the module library:

1. Get the latest table of contents of the module library from a MOSART contact person.
2. Browse through the table of content to find the module(s) that is suitable (in the current form or with changes) for usage in the project.
3. If one or several useful modules are found in the library, request documentation and software for the modules in question.

This checklist is useful when someone wants to provide a new module to the module library:

1. Make sure that the requirements of the module library are fulfilled for the module that is to be added to the library. This is particularly important when testing and documentation are concerned.
2. Add the new module through a MOSART contact person by providing software and documentation for the module.
3. Update the module library table of contents with the new module.

4 References

[1] MOSART FOI internal project homepage (http://mikaelb.foi.se/MOSART/)

MOSART
Modeling and simulation for analysis and research test-bed

MOSART – general documentation

Data Repository – instructions for use

FOI Memo 03-2761:3

Author: Mathias Tyskeng **E-mail:** mailto:mattys@foi.se **Telephone:** +46 13 37 85 89	**Coauthors:**
Contents: Describes the MOSART data repository	**Version:** 1.0 **Date:** 2003-12-12
Filename: instructions_for_use_data_repository.doc **Folder:** \\Filer-lin\MOSART\ projects\MOSART\docs\	

CHANGE HISTORY

VERSION	DATE	CHANGES
0.1	031112	Initial version
1.0	031212	Final version

Contents

1 Introduction

This document is a user's manual for the MOSART data repository. The system is a management system for handling of sensor data at FOI.

2 Data Repository

2.1 Overview

The MOSART data repository is a database system at FOI which is used for handling of large amounts of sensor data that are spread all over FOI. The system is called FLEX (FOI i Linköping Experimentdatabas). The idea is to have a searchable database which contains meta data about the sensor data. The actual sensor data is not affected by the introduction of this system and can remain at the current storage location. In fact, in order to keep the data repository intact it is desirable that sensor data is not moved too often. It does not matter how sensor data is stored (file server, external media or safety cabinet), it can be found using the data repository as long as the meta data is fed into the data repository. The difference is that it is only from file servers that the sensor data can be downloaded online.

Below, the architecture of the data repository is illustrated.

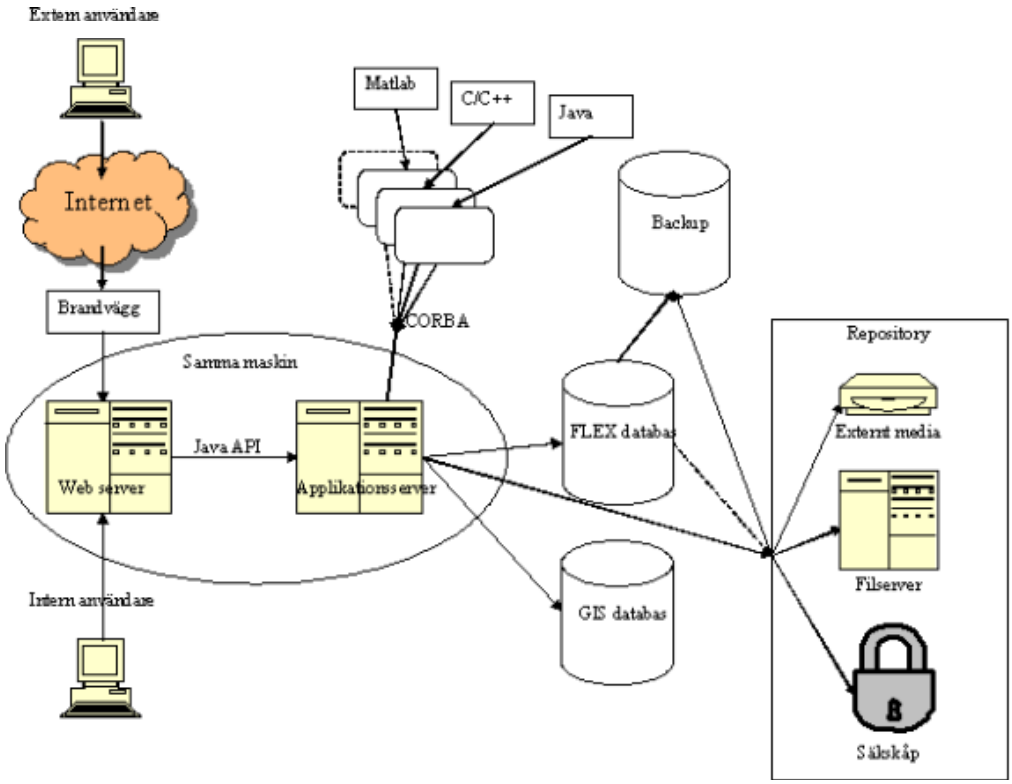

Figure 1. Architecture overview of the data repository.

All usage of the data repository requires the user to log in to the system. There are two ways to access the data repository, a web GUI and software adapters. In the web GUI the user can (after login) search for sensor data according to specified criteria, edit/register new FLEX entry and visualize the sensor data if there is a plug-in for the file format in question. It is possible to do the same actions from the software adapters as in the web GUI, but it is also possible to save the search results to file for later.

There are software adapters for the following languages:

- Java

- C++
- Matlab

In the data repository there are several search parameters that are built in:

- Geographical area (specified coordinates or map frame)
- Time slot (start time and stop time)
- Ground conditions
- Vegetation
- Weather
- Sensor type
- Sensor mode
- Sensor make
- Format
- FLEX group
- FLEX file

Also, there is a name, a text description of the data and so called dynamic attributes that the user can specify when sensor data are registered/edited. Of these attributes there are some that a user must provide to create a FLEX entry; name, description, format, from date, to date and location (coordinates).

2.2 User interfaces

2.2.1 Graphical User Interface

The FLEX data repository is accessible via a web based graphical user interface (GUI). In the GUI a user can do all actions that the FLEX system supports. Below is an illustration of the navigation in the GUI.

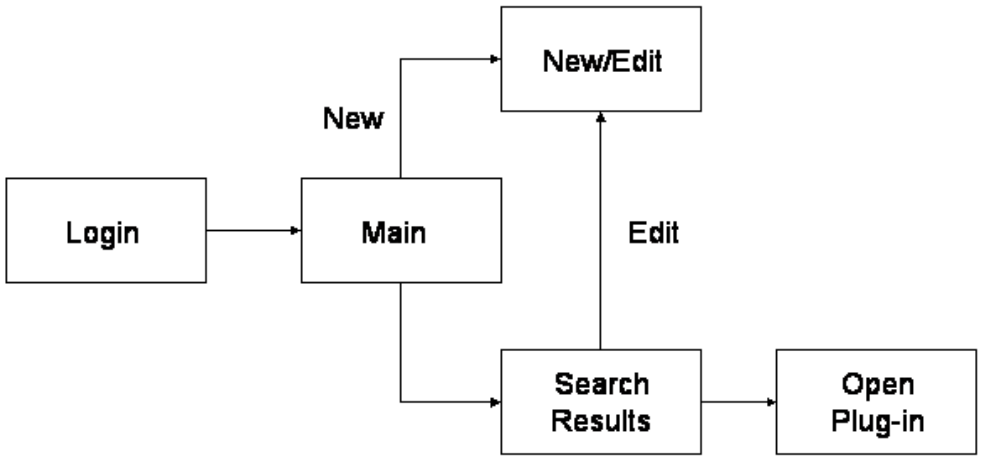

Figure 2. Overview of the navigation in the graphical user interface.

When a user has logged in, the main page is shown (see below). In the main page, it is possible to choose search criteria and when a search is made the result will be displayed in a new window where the matching FLEX entries are shown along with coverage areas in the map. In this dialog, sensor data can be selected for download if it is online and the user has access to the location where the sensor data is stored. If login is required for the storage location, the user will be prompted for a password before access. It is also possible to select sensor data for visualization if the sensor data format has a suitable plug-in in the web browser, e.g. images.

When a FLEX entry is found it can be edited by the user if he/she has the right to do so. A new FLEX entry is added by selecting New FLEX in the main dialog. Any user can add a new FLEX entry.

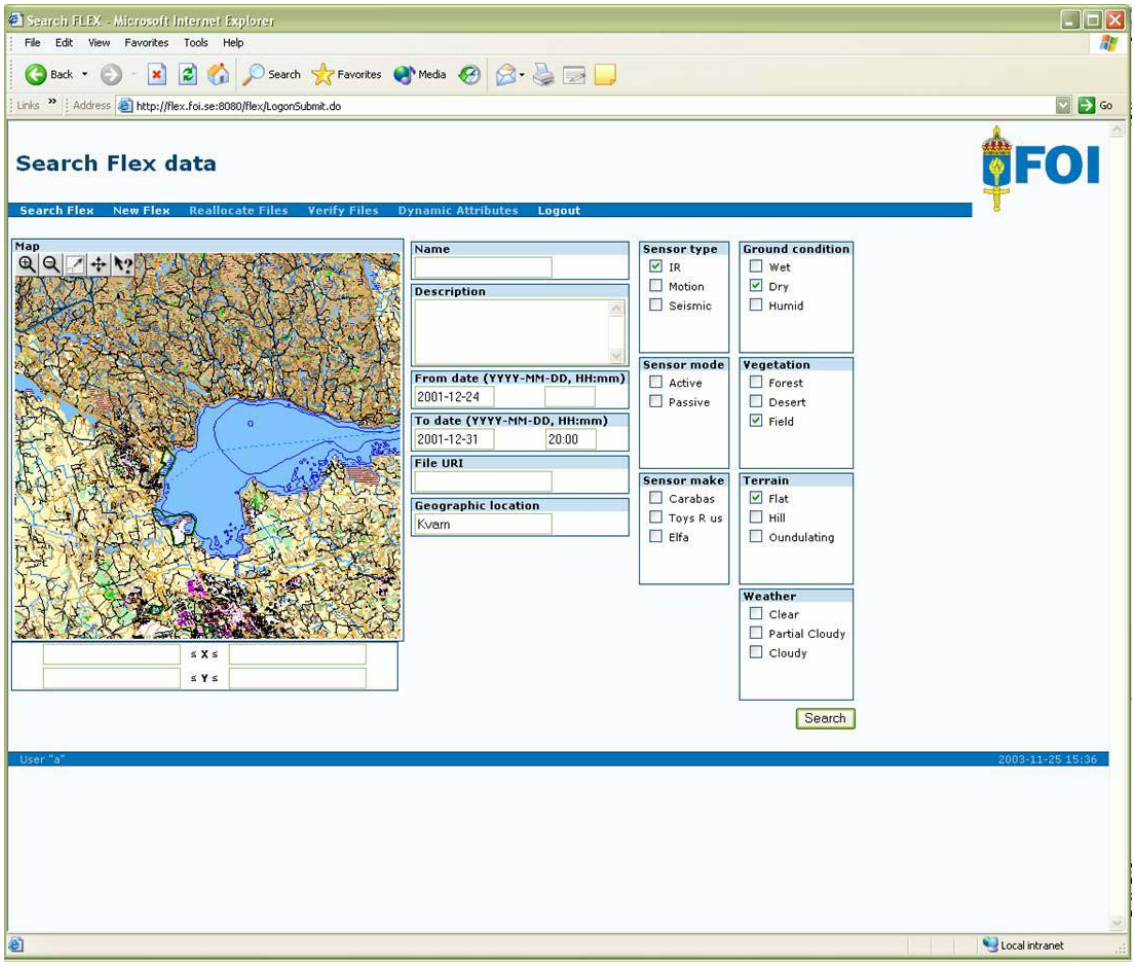

Figure 3. The search FLEX dialog.

2.2.2 Software adapters

Not all users want to use the GUI to access the FLEX system. For users who want to use the FLEX system in their applications there are software adapters that the application can access the FLEX system in the same way that the GUI can. At present there are software adapters for Java, C++ and Matlab.

3 Data Repository Usage

In this section the basics for usage of the FLEX data repository is described.

3.1 Login

Before a person at FOI can become a FLEX user the person has to be registered as a FLEX user in the FLEX user group at FOI. This is done by contacting the IT support unit at FOI. When the person is registered it is also possible to assign the person to a subgroup that will be able to edit data that other members of the subgroup have entered into the FLEX system.

3.2 Search FLEX

Search is conducted in the search dialog in the FLEX system. In the dialog it is possible to refine the search criteria by selecting a number of constraints from the fixed criteria and the map in the dialog. Several actions can be made on the map, zoom in, zoom out, move center and zoom out maximum. A box can be drawn in the map to select the area of interest. Any sensor data that intersects with the area will be considered a match if all other constraints are fulfilled. If the user is an advanced user that knows exactly what he/she is looking for it is possible to use dynamic attributes that the person who created the FLEX entry specified.

The result of the search will be displayed in a new dialog where the matching FLEX entries are shown both in a map and in a list. From the list the user can choose sensor data files for download or visualization if there is a suitable plug-in for the format.

3.3 Edit/create FLEX

Entries in the FLEX data repository are created manually in the New FLEX dialog. This is the same dialog as when a FLEX entry is edited. The difference is that when an entry is changed the original parameter settings are shown in the dialog.

When an entry is created there are some parameters that have to be specified. These parameters are:

- Description (text)
- Geographical area (specified coordinates or map frame)
- Time slot (start time and stop time)
- Format

This means that an entry can be created without specifying a data file which can sometimes be useful.

3.4 Sensor data access

When someone has become a member of the FLEX user group it is possible to log on to the FLEX system. The only restriction in the system is that the user can not access sensor data that is stored at a location where the user has not got rights to read. The FLEX entry however, will be visible to the user so that he/she knows that it exists.

MOSART
Modeling and simulation for analysis and research test-bed

MOSART – visualisation

MIND – instructions for use

FOI Memo 2761:4

Author: Mathias Tyskeng **E-mail:** mailto:mattys@foi.se **Telephone:** +46 13 37 85 89	**Coauthors:**
Contents: Describes the 2D visualisation tool MIND in MOSART	**Version:** 1.0 **Date:** 2003-12-12
Filename: instructions_for_use_MIND.doc **Folder:** \\Filer-lin\MOSART\ projects\MOSART\docs\	

CHANGE HISTORY

VERSION	DATE	CHANGES
0.1	031030	Initial version
1.0	031212	Final version

Contents

1 Introduction

1.1 Document purpose

This document is a short introduction to the MIND visualisation system and how it can be used within MOSART at FOI.

2 MIND

2.1 Introduction

MIND [1] is a software tool which is developed within FOI. The development has been ongoing for several years and experience from field operations, both civilian and military, has been integrated into the system. MIND is a versatile tool for:

- Modelling of military units and tactical events
- Simulation of technical sub-systems
- Visualization of tactical information and common situation picture
- Experimental environment for model based decision support
- Support for evaluation and analysis of operations

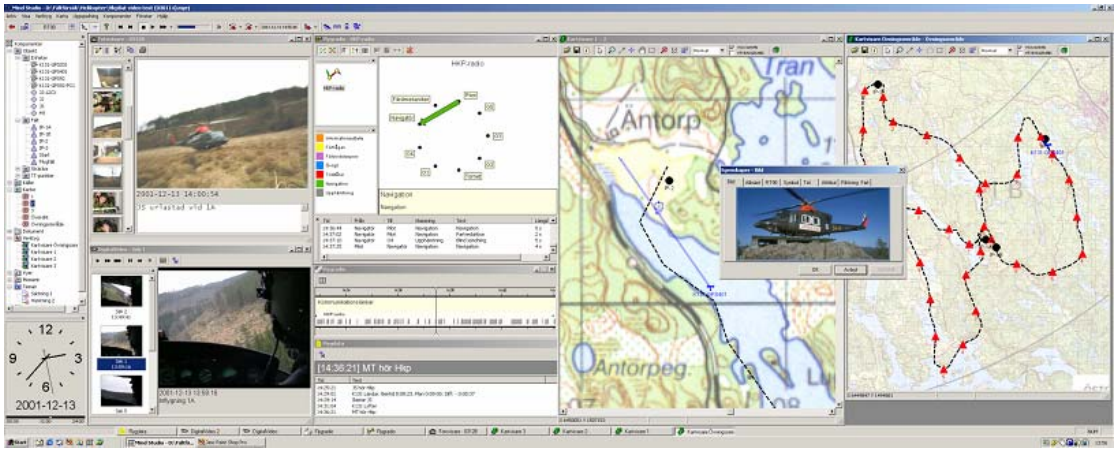

Figure 1. Views in MIND.

2.2 Views

MIND is a window based system, which has windows for visualization of different information called views. The views display communication flow, communication amount, video clips, audio sequences and many other types of information which has been recorded during the operation in question or is gathered at real time. The purpose of the views is to provide the persons involved in the operation a possibility to, either during the operation or after, have an overview over the different events during the operation and use it for e.g. analysis and evaluation.

2.3 Data sources

Every view in the MIND system has some information attached to it. That information is called a data source and it can be a data file or a port to a real or simulated system. Therefore, the MIND system is very easy to expand with new data sources and views that visualize the data from the new source. In many cases there is no need for a new view when a new data source is added, since the new information can be visualized in existing views, e.g. the HLA source used by MOSART.

3 MIND in the MOSART testbed

3.1 Overview

In MOSART, MIND is currently used as a 2D visualization tool and event viewer. The 2D visualization is made in a map view where a map of the user´s choice can be used as background. The event viewer is a window where messages that are sent within the HLA federation can be displayed to a user. This is a good way to put key events in the simulation to the user's attention.
MIND in MOSART is a federate which can subscribe to objects and visualize an icon and/or polygon/area at the position reflected by the HLA federation. Between updates (reflect), the kinematics are dead-reckoned to smoothen the visualization.
Figure 2 shows a screen shot from the MOSART pilot demonstration showing the map view and the event viewer called Game Leader.

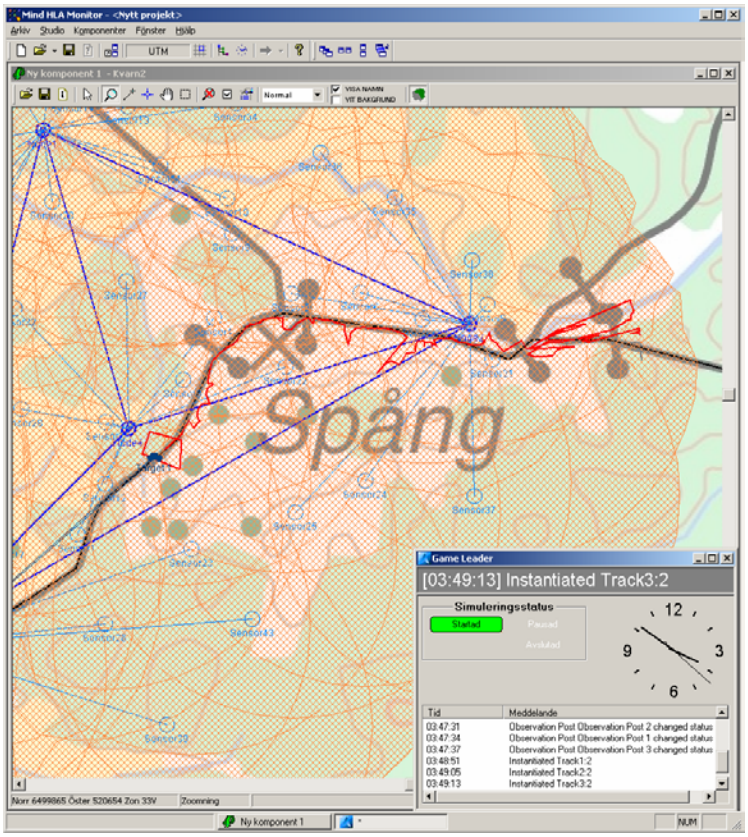

Figure 2. Screen shot of MIND from the MOSART pilot demonstration.

3.2 Data source – HLA

The key component of MIND in MOSART is the HLA data source, which is the gateway from an HLA federation to MIND. Below, the overall architecture of the HLA data source is illustrated.

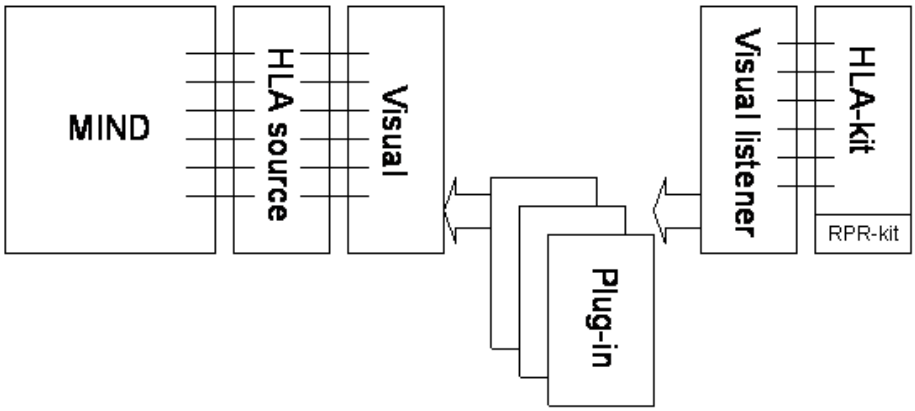

Figure 3. Architecture overview of MIND federate

The MIND federate uses the HLA-kit as all other federates in MOSART. This in combination with the RPR-kit makes the federate handle all types of objects in the RPR-FOM (see [2]) if there is a corresponding plug-in for that object. The HLA-kit uses listeners to implement specific behaviour for a specific call to the HLA-kit. The MIND federate uses the same structure. A listener, called Visual listener, handles all calls to the federate (discover object, reflect attributes, remove object and interact) and uses the plug-in for the object in question. It is an error to try and use objects with no plug-in. A plug-in is a C++ dynamic link library which implements a standard plug-in interface. The name of the dll-file must be the full HLA object name, e.g. objectroot.baseentity.groundvehicle.plugin.dll. An object can also have a property file, which contains initial value for the object type. The property file name is similar to the plug-in, but .plugin.dll is changed to .properties.

Everything that has to do with MIND is in the Visual interface. All calls to MIND is handled by a Visual object which is created by the Visual listener and is a singleton. This Visual object will then be passed to the plug-ins along with a HLA object from the Visual listener, i.e. the plug-ins themselves do not keep internal state data. In the listener the HLA objects will be associated with MapObjects (contains icon, GraphicObjects etc.) that the plug-in can operate on and use in calls to MIND. These objects are created by the plug-in, but kept in the listener. GraphicObjects have subclasses line and area which can be used to build arbitrary symbols in MIND.

Between updates (reflects), the federate dead-reckons the kinematics of the objects by “ticking” all object, using the plug-ins.

This approach makes it relatively easy to start using a new type of object in MIND, since it is only a new plug-in that has to be written.

3.3 MIND usage

When someone wants to use MIND in a federation, the first thing to do is to see if the RPR objects that will be used in the federation are supported by the MIND federate. If not, new plug-ins for the missing objects must be written so that the objects will appear in MIND. Addition of a plug-in requires no recompilation of the MIND federate. If another object than a RPR object will be used, the code for the new HLA object must be added and a recompilation of the federate is needed.

When writing a plug-in, the methods in the Visual object are used to control the symbols in MIND.

In order to use MIND as a visualization tool in a MOSART federation, the procedure is as follows:

1. Start a MIND version that has the HLA data source.
2. In the Component view, right-click on Källor (Sources) and choose Ny component… (New component…).
3. Right-click on HLA-källa (HLA data source) and select Egenskaper (Properties).
4. Under the tab Uppkoppling (Connection), fill in the correct values for the simulation that is planned.
5. Import a map, using the Map component (Kartor).

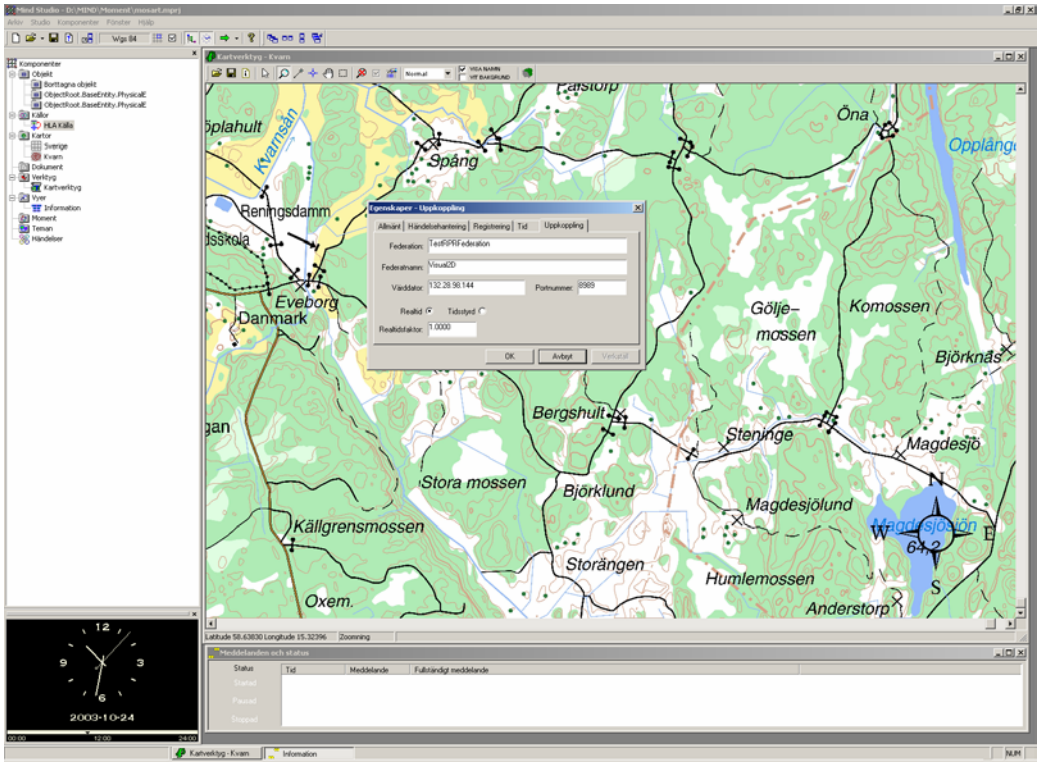

Figure 4. MIND in the MOSART testbed.

4 References

[1] MIND, FOI, Div. of C2 Systems, Dept. of Systems analysis and IT security

[2] MOSART FOI internal project web page: http://mikaelb.foi.se/MOSART/frameworks/hla/object_models/MOSART-RPR-FOM

MOSART
Modeling and simulation for analysis and research test-bed

MOSART – visualisation

Vega – instructions for use

FOI Memo 03-2761:5

Author: Mathias Tyskeng **E-mail:** mailto:mattys@foi.se **Telephone:** +46 13 37 85 89	**Coauthors:**
Contents: Describes the 3D visualisation tool Vega in MOSART	**Version:** 1.0 **Date:** 2003-12-12
Filename: instructions_for_use_Vega.doc **Folder:** \\Filer-lin\MOSART\ projects\MOSART\docs\	

CHANGE HISTORY

VERSION	DATE	CHANGES
0.1	031030	Initial version
1.0	031212	Final version

Contents

1 Introduction

1.1 Document purpose

This document is a short introduction to the Vega framework and how it can be used within FOI.

2 Vega

2.1 Introduction

Vega is a system from MultiGen-Paradigm which is used for 3D visual simulation [1]. It is a system which can be configured for most simulation purposes and it also supports real-time simulation. The system can be easily expanded since it has a plug-in architecture and a number of software interfaces which supports different programming languages. There is support in the system for e.g. distributed simulation, marine applications, symbolic applications and visual effects. There are also modules for sensor and sensor effects that are suited for real-time simulation.
Vega mainly uses terrain and object models in OpenFlight format [2]. Figure 1 shows a screen shot from Vega, which shows a terrain model and a helicopter model.

Figure 1. Screen shot from Vega

2.2 Vega properties

The models of the terrain and objects that Vega uses should be in OpenFlight format. However, when a model has been used in Vega, the model is compiled and stored by Vega. This compiled terrain model can then be used instead of the OpenFlight model with exactly the same result.
There is a file that Vega uses to set up the terrain, objects, coordinates, light conditions, camera views etc. and it has the file ending .adf. This type of file is edited with the program Lynx which is a part of the Vega system.

3 Vega in the MOSART testbed

3.1 Overview

In MOSART, Vega is currently used as a 3D visualization tool. The 3D visualization of the terrain is made from an OpenFlight model that the user can specify in the properties file.
Vega in MOSART is a federate which can subscribe to objects and visualize an OpenFlight model and/or polygon/area/volume at the position reflected by the HLA federation. Between updates (reflect), the kinematics are dead-reckoned to smoothen the visualization.
Figure 2 shows a screen shot from the MOSART pilot demonstration showing a terrain model and an object model that was used.

Figure 2. Screen shot of Vega from the MOSART pilot demonstration.

3.2 HLA connection

The key component of Vega in MOSART is the HLA connection, which is the gateway from an HLA federation to Vega. Below, the overall architecture of the HLA connection is illustrated.

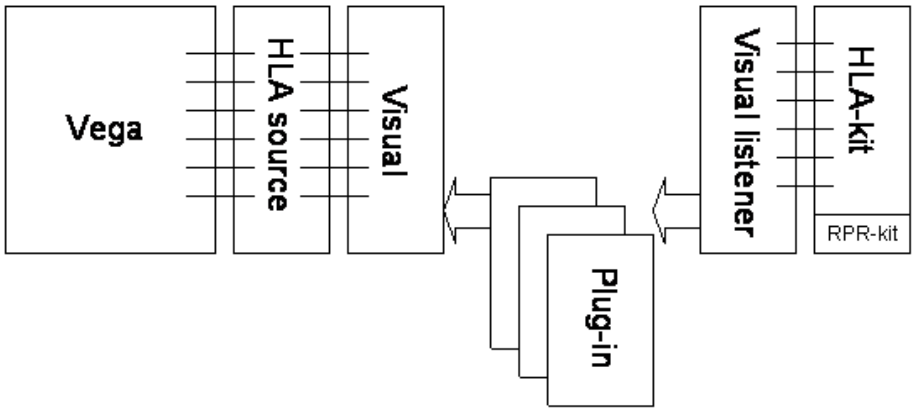

Figure 3. Architecture overview of Vega federate

The Vega federate uses the HLA-kit as all other federates in MOSART. This in combination with the RPR-kit makes the federate handle all types of objects in the RPR-FOM (see [3]) if there is a corresponding plug-in for that object. The HLA-kit uses listeners to implement specific behaviour of specific calls to the HLA-kit. The Vega federate uses the same structure. A listener, called Visual listener, handles all calls to the federate (discover object, reflect attributes, remove object and interact) and uses the plug-in for the object in question. It is an error to try and use objects with no plug-in. A plug-in is a C++ dynamic link library which implements a standard plug-in interface. The name of the dll-file must be the full HLA object name, e.g. objectroot.baseentity.groundvehicle.plugin.dll. An object can also have a property file, which contains initial value for the object type. The property file name is similar to the plug-in, but .plugin.dll is changed to .properties.
Everything that has to do with Vega is in the Visual interface. All calls to Vega is handled by a Visual object which is created by the Visual listener and is a singleton. This Visual object will then be passed to the plug-ins along with a HLA object from the Visual listener, i.e. the plug-ins themselves do not keep internal state data. In the listener the HLA objects will be associated with MapObjects (contains icon, GraphicObjects etc.) that the plug-in can operate on and use in calls to Vega. These objects are created by the plug-in, but kept in the listener. GraphicObjects have subclasses line, area and volume which can be used to build arbitrary symbols in Vega.
Between updates (reflects), the federate dead-reckons the kinematics of the objects by “ticking” all object, using the plug-ins.
This approach makes it relatively easy to start using a new type of object in Vega, since it is only a new plug-in that has to be written.

3.3 Vega usage

When someone wants to use Vega in a federation, the first thing to do is to see if the RPR objects that will be used in the federation are supported by the Vega federate. If not, new plug-ins for the missing objects must be written so that the objects will appear in Vega. Addition of a plug-in requires no recompilation of the Vega federate. If another object than a RPR object will be used, the code for the new HLA object must be added and a recompilation of the federate is needed.
When writing a plug-in, the methods in the Visual object are used to control the symbols in Vega.
In order to use Vega as a visualization tool in a MOSART federation, the procedure is as follows:

1. Prepare the .adf properties file with Lynx.
2. Make sure that Vega is installed on the computer.
3. Start the Vega federate that has the HLA connection.
4. Vega is now ready to receive objects from the HLA simulation.

4 References

[1] Vega, Multigen-Paradigm Inc. (http://www.multigen.com/)
[2] Synthetic environments, FOI, Div. of sensor technology, Dept. of Laser systems

[3] MOSART FOI internal project web page: http://mikaelb.foi.se/MOSART/frameworks/hla/object_models/MOSART-RPR-FOM/MOSART-RPR2-D14.html.

MOSART
Modeling and simulation for analysis and research test-bed

MOSART – frameworks

FLAMES – instructions for use

FOI-Memo 03-2761:6

Author: Martin Jerberyd **E-mail:** martin.jerberyd@foi.se **Telephone:** +46 8 5550 3516	**Coauthors:**
Contents: Description of FLAMES in MOSART	**Version:** 1.0 **Date:** 2003-12-12
Filename: instructions_for_use_Flames.doc **Folder:** \\Filer-lin\MOSART\ projects\MOSART\docs\	

CHANGE HISTORY

VERSION	DATE	CHANGES
0.1	031208	Initial version
1.0	031212	Final version

Contents

1 Introduction

This document is a short introduction to the FLAMES simulation frameworks and how it can be used within MOSART at FOI. For a more detailed description of the FLAMES framework it is recommended to read the FLAMES documentation [1].

2 FLAMES

2.1 Introduction

FLAMES (Flexible Analysis and Mission Effectiveness System) is a simulation framework tool that is developed by Ternion. The framework is used for developing models that can simulate both the physical and cognitive behavior of complex entities. The tool was initially intended for simulation of air-combat but can also been used for both ground and underwater scenarios. The main goal of FLAMES is to shorten the time and simplify the development of simulation models. The development time can be shortened by modifying example models that come with the framework and by using the API functions in the FLAMES kernel. FLAMES also provides the user with a visualisation interface for building and analysing scenarios.

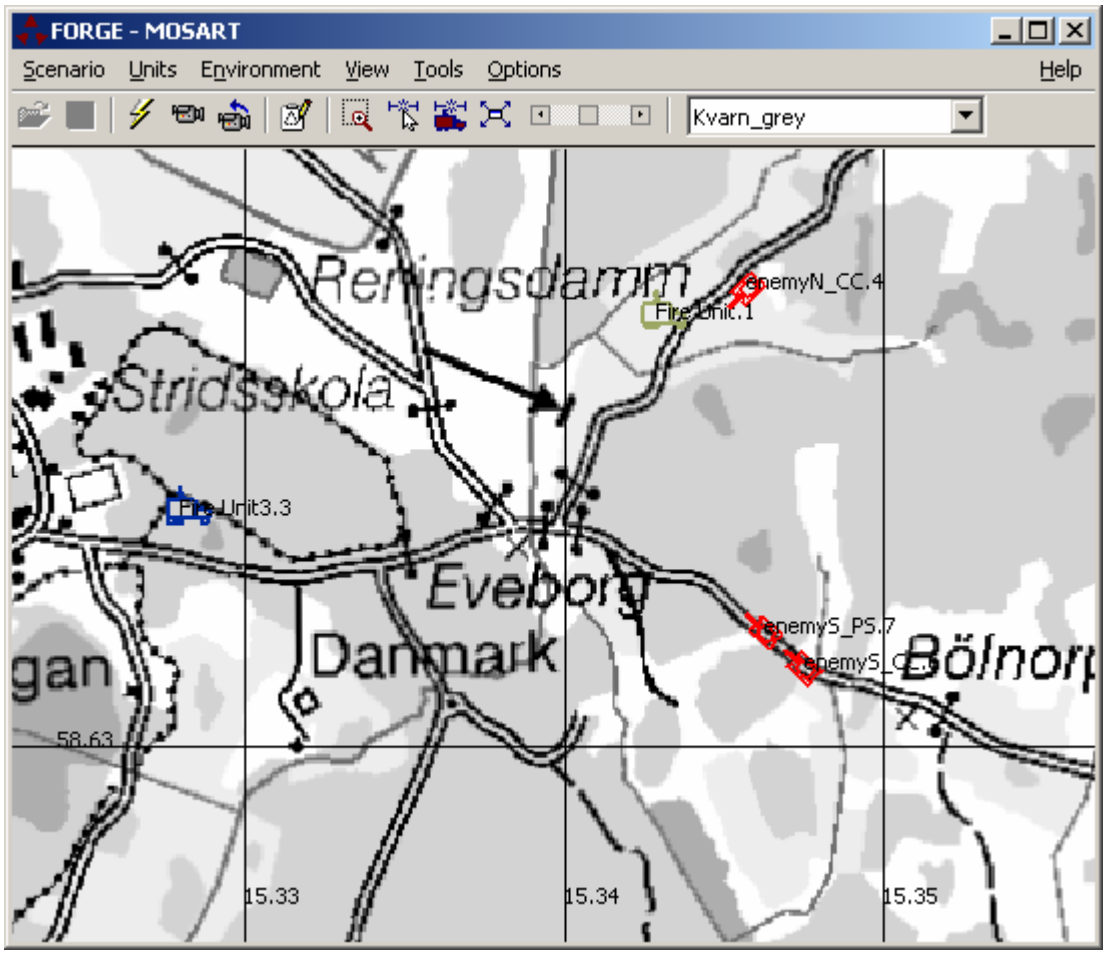

Figure 1. FLAMES scenario generation in FORGE.

2.2 FLAMES system overview

FLAMES consist of four applications: FORGE, FIRE, FLASH and FLARE. To develop a new simulation model for FLAMES each of those applications has to be recompiled together with the code for the new model. This is done by linking the new model source code with the FLAMES libraries,

which are provided by Ternion. Figure 2 shows how FLAMES executes simulations using the four applications just mentioned.

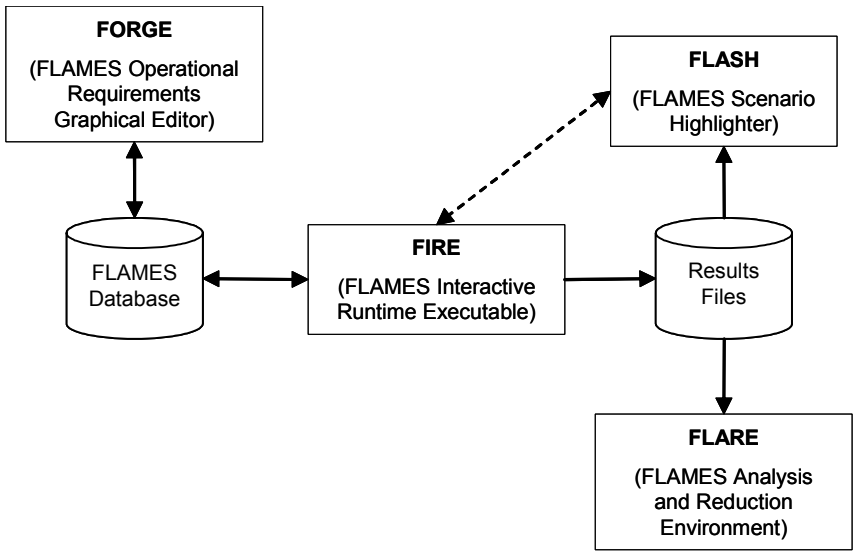

Figure 2. FLAMES system overview

First, FORGE is used to create and edit the scenario. The scenario is created by using a script language to set up the entities the user wants to simulate. By using the script language, the user can employ equipment models to a platform and add cognitive behavior for the unit. The unit can be placed on then right position with a point and click interface directly on the scenario map. The created scenario is saved in the FLAMES database. Secondly, FIRE is used to execute the scenario. The scenario parameters are read from the database. FIRE provides no visualization of the simulation. The result of the simulation is saved in a file, which can be analyzed afterwards. Finally, the result of the simulation is analyzed with FLASH or FLARE. FLASH shows the graphical visualization of the scenario and FLARE is used for viewing text information that could be recorded during the execution of the simulation in FIRE.

The framework supports two main types of model development. Equipment models are used to model physical equipment like: vehicles, munitions, sensors, communication equipments and so on. Cognitive models control the units by implementing Artificial intelligence and other control mechanisms. Messages and communication between the units are also supported by a special FLAMES message model standard.

3 FLAMES in the MOSART testbed

3.1 Overview

In MOSART, FLAMES could be used as the simulation engine and scenario editor. The models developed with the framework and the example models from Ternion could of course also be used in a MOSART simulation. As a simulation engine, FLAMES executes the simulation in real-time and interacts with external HLA federates. External HLA federates could for example be air and ground vehicles or any other type of equipment. For example, FLAMES could update an HLA federation with the position of the units in the simulation. An external HLA federate, modeling a sensor net could then subscribe to the positions of the units and return to FLAMES the track information that the sensors have calculated. The track information that is returned could be used by another model to make a decision about troop movements.

The visualization performance in Flames is rather limited. The visualization could easily be improved by using other visualization tools within the MOSART test bed. MIND is a 2D visualization tool that could be connected via HLA to FLAMES. The simulation results from the FLAMES execution could be analyzed and visualized in real-time.

To make FLAMES compatible with the MOSART test bed, the existent HLA manager has been modified to support the RPR-FOM used in MOSART. Figure 3 shows an example of how FLAMES can connect to different federates within the MOSART test bed.

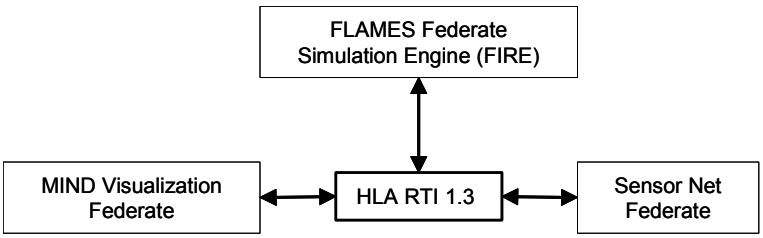

Figure 3. Example of how FLAMES could be used in MOSART

3.2 FLAMES – HLA

The key component of FLAMES in MOSART is the HLA option. The HLA option is an extra package from Ternion that enables the feature to use HLA in FLAMES. The intention of the package is to make the HLA integration easy and to provide a framework for HLA development in FLAMES. All communication to the RTI goes through a special FLAMES component called the FLAMES HLA-Server. Instead of writing normal HLA ambassadors that directly communicate with the RTI, a simpler ambassador that uses the services in the FLAMES HLA-Sever are used. Figure 4 shows all the necessary components in the FLAMES HLA architecture. Note that it is possible to have several ambassadors in the same FLAMES federation.

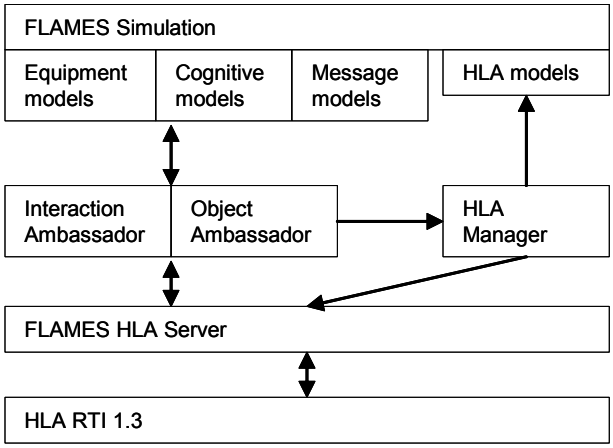

Figure 4. FLAMES HLA architecture

The functions of the FLAMES-HLA Architecture are:

- FLAMES HLA-Sever. The HLA-Sever handles all connections to the RTI. It also updates and reflects all the attributes of the HLA objects registered by the Object and Interaction ambassadors.

- HLA manager. The manager contains functions for creating new HLA units in the FLAMES simulation. It also handles all the settings that are necessary for the federation.

- Ambassadors. The ambassadors register and send the Object and interaction attributes to the HLA-Sever. The ambassador uses the functions in the HLA-manager to create new HLA units in the simulation.

- HLA models. HLA models are necessary to model and display the units and entities that belong to a different federate. The models can for example represent physical platforms and explosion effects. HLA encoders are also necessary for publishing the local units in the FLAMES simulation.

For each new simulation that uses different FOM:s both the ambassadors and the HLA manager have to be modified. A new ambassador has to be created for every new HLA-object that the FLAMES federate needs to publish or subscribe to. FLAMES HLA-Server is provided by Ternion and cannot be modified or changed.

3.3 MOSART – Pilot Scenario

The following objects has been developed and modified for making FLAMES compatible with the MOSART test-bed and the Sensor Net pilot scenario [3]. The FOM that is used is MOSART-RPR2.0 Draft 14.

- FOIMOFRPRManagerClass, is a modified HLA-manager class to handle the RPR-FOM

- FOIMOFPhysicalEntityAmbassador, is an ambassador that publishes and subscribes to the PhysicalEntities in the HLA simulation (this includes for example the positions of the sensors)

- FOIMOSensorNetTrackAmbassador, subscribes on the Track object (This comes from the Sensors). Each track position is represented by an icon in the FLAMES simulation. The track information gathered by the ambassador is sent to the other units in the simulation with a FLAMES model of a radio communication device.

- FOIMOFQPRPRPlatform models the PhsicalEntities that the ambassador subscribes to.

- FOIMOFTRPRPlatEncoder encodes the local platform models in FLAMES to PhsicalEntities objects in the RPR-FOM.

- Cognitive and message models for the Command-Unit

Figure 5 shows the MOSART Sensor net scenario in action. The sensor net in the middle of the map belongs to the Sensor net federate while the units belongs to the FLAMES federate.

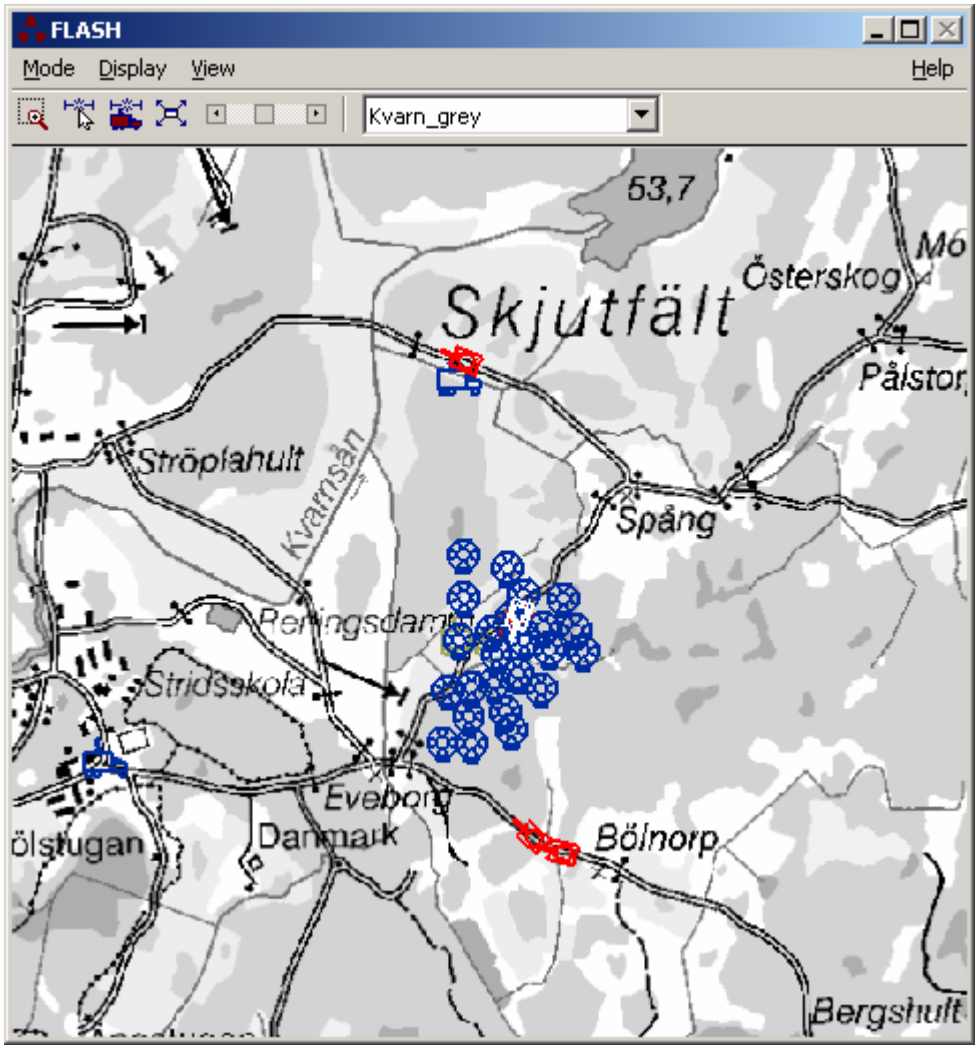

Figure 5. MOSART Sensor net scenario

3.4 FLAMES usage

How to use the HLA option in FLAMES is described in the FLAMES manuals [1].

The current version of FLAMES HLA does not support synchronization services. It is therefore vital that the simulation is executed in real-time, without time constraints and synchronization points. However, this problem should be solved in the next version of FLAMES.

3.5 Summary

FLAMES is now functional in the MOSART testbed and can be used as a scenario engine and scenario editor.

4 References

[1] FLAMES help file manual [flames.chm], Ternion 2003

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

[3] MOSART FOI internal project homepage (http://mikaelb.foi.se/MOSART/)