

Connecting Simulations of Various Architectures in a Central Simulation Framework with Networking Capability to Support Wargaming for the Swedish Armed Forces

Gunnar Hovmark¹, Fredrik Jonsson²

¹Consultant, ÅF, Stockholm, Sweden, gunnar.hovmark@afconsult.com

²The Swedish Defence Materiel Administration, fredrik.jonsson@fmv.se

Abstract -- Models of human behaviour and technical systems based on different simulation architectures are combined together around a central simulation framework with facilities for distributed simulation. Experts in fields such as human behaviour modelling, air defence tactics, missile simulation, radar performance calculation and aircraft performance modelling provide models and data. The COTS simulation framework provides timing, visualization, data logging, connectivity and basic models of some systems. The simulation results are analysed to provide data mainly to support wargaming for the Swedish defence planning activities. The current simulation setup mainly deals with simple air defence scenarios, but size, complexity and time span can be increased and other applicable models and simulations from the Swedish defence community can be added.

1 Introduction and background

The Swedish Armed Forces have appointed the Swedish Defence Materiel Administration (FMV) and the Swedish Defence Research Agency (FOI) to provide simulations to support wargaming for defence planning activities.

The simulations use models and data from various parts of the defence community, and are based on different architectures. All are synchronized and run using a COTS simulation framework. The choice of tools has been made with distributed networking capability in mind.

The aim of the simulations is to evaluate air defence scenarios. Initially quite simple, but the ambition is to gradually increase scope, size, complexity and time span.

Outcomes from the simulations will be used to support adjudication in the wargaming that is performed as a part of the Swedish defence planning process.

2 Central connecting framework

FLAMES from Ternion Corporation (www.ternion.com), was chosen as the central connecting framework for the simulations. The other simulation programs that were considered were mainly in-house developed by Swedish authorities and/or companies, and it was anticipated that the small user base for those programs would have made support less accessible and continuity might have been a problem, so a COTS tool was preferred. FLAMES was already in use in the Swedish defence community, mainly within FOI but also at Saab, FMV and the Swedish Defence University, and others. There are users worldwide and a support organization that responds quickly to any questions we may have. FMV also hosts meetings with a users' group with members from Sweden and Finland.

The FLAMES architecture offers easy setup of simulations by using C or C++ components that represent behaviours, systems and subsystems, and datasets containing data and scripts to connect and configure the components. The "Service" facility in FLAMES is useful for driving other simulations synchronized with the

models that run within FLAMES itself, and external simulations can be connected by using the FLAMES facilities for the DIS (the Distributed Interactive Simulation protocol) or HLA (High Level Architecture) standards.

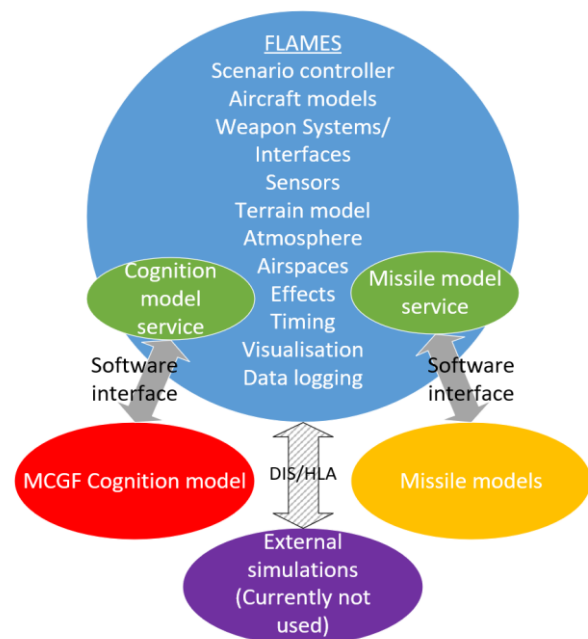


Fig. 1. Diagram showing current setup with auxiliary processes

Fig. 1 shows the main components of the current simulation setup. FLAMES controls the scenario, provides the simulation time, visualisation and datalogging, and also runs a number of models that have been integrated into FLAMES, such as aircraft, terrain, sensors, weapon effects and weapon systems. The cognition models that represent pilot behaviour, known as MCGF, are run in a separate process that is connected to FLAMES via a service and a software interface. The missile models also run in a separate process and are connected in a similar way as the cognition models. The DIS/HLA connection is currently not in use.

3 Networking

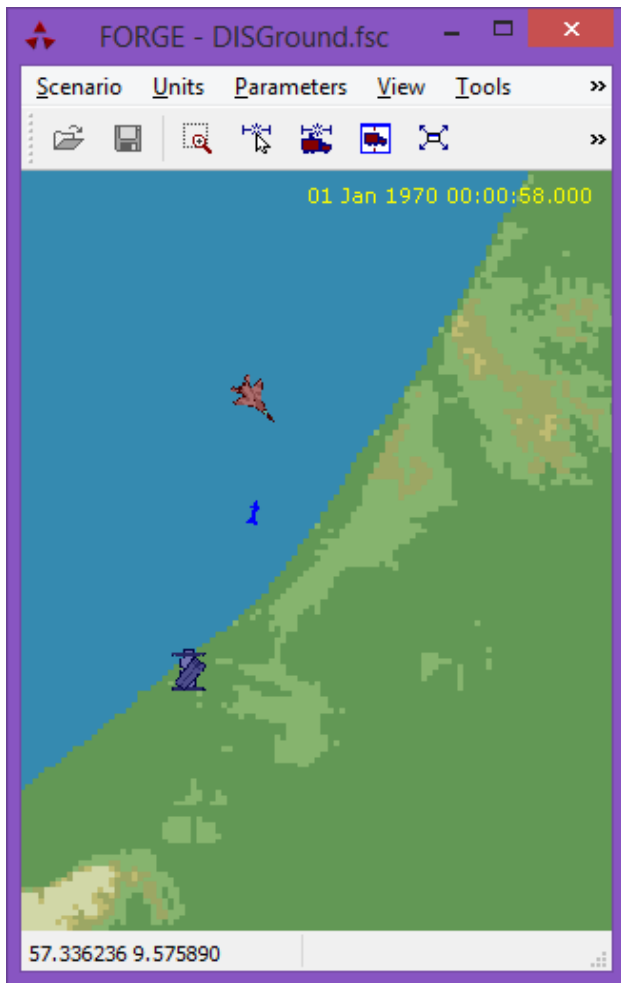


Fig. 2. Ground part of a DIS demo, using aircraft data via DIS

Networking using DIS, see for example reference [1], or HLA, see for example reference [2], can be a way to connect to simulations that are not easily linked directly into the central framework, and it can also be a way to bridge the gap between constructive simulation on one hand, and live and virtual simulation on the other. This capability has so far not been required in the project, but may be useful in the future.

FLAMES provides facilities for DIS PDUs (Protocol Data Units) for Entity State, Detonation, Fire, Start/Resume, Stop/Freeze and Acknowledge.

FLAMES comes with full source code for processing these PDU types, so you have a good start if you need other PDUs.

Platforms and effects can easily be connected to their DIS enumerations (see reference [3]) based on their regular identifiers in FLAMES, so you have to change very little in your already running non-DIS simulation. What you have to do is to make sure that the identifiers of your existing platforms and effects point to the correct DIS enumerations, and connect any DIS enumerations that may come from the outside to the proper identifiers of the corresponding platforms and effects that you want to be present in FLAMES.

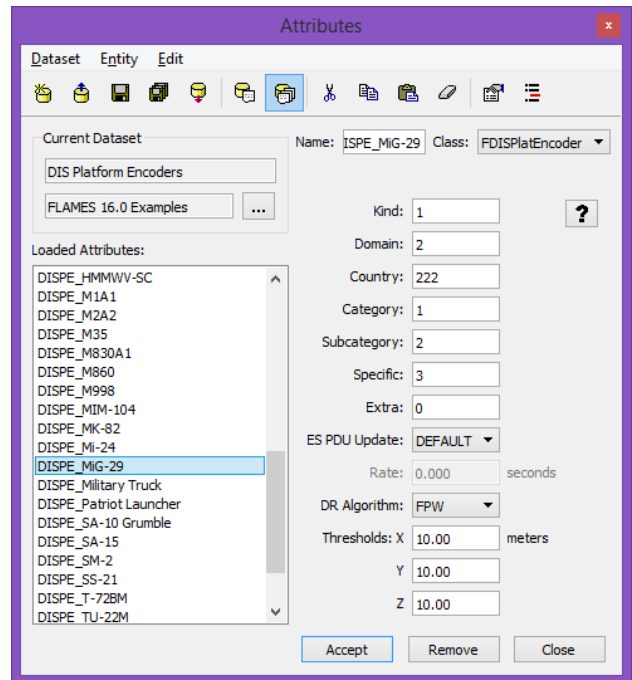


Fig. 3. Encoding, identifier MiG-29 to DIS enumeration

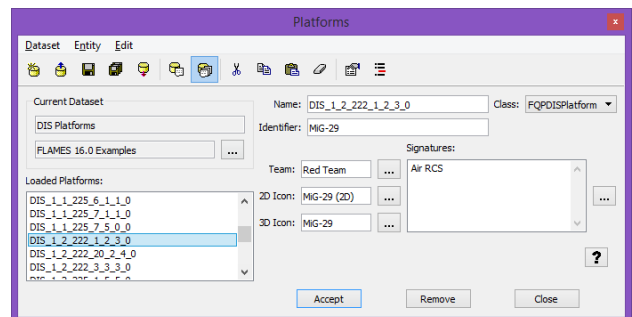


Fig. 4. Decoding, DIS enumeration to identifier MiG-29

The HLA connectivity in FLAMES is based on the old HLA 1.3 standard, so for connection to current HLA real-time interfaces you will probably want to use DIS on the FLAMES side and get a DIS to HLA gateway from a third party.

4 Models

The current activities have focused mainly on models of fighter aircraft, air-to-air missiles and fighter pilot behaviour. So far there has been less emphasis on models of sensors, communications, electronic warfare, surface to air and air to surface capability and the chain of command, so there is still a considerable amount of work to do before there is a complete simulation of a whole air defence scenario.

Models and data are mainly provided by FMV and FOI, but also from defence equipment suppliers. The models used have been developed over a considerable length of time and with very different philosophies regarding simulation architectures. It was at an early stage determined that it would not be practical for all of the models to be converted to code that would fit directly into

the architecture of one central simulation program. The alternative method of choice in the current project is to use the FLAMES “Service” facility, a class that is suitable for initializing and calling a separate simulation program with a fixed time step. C functions can then be added in FLAMES to call dedicated functions and subroutines in the separate simulations to exchange data and perform interactions.

4.1 Missile Models, Refbib and Rbsim

The air to air missiles that are used in the simulations come from a software package known as “Referensbiblioteket”, the Reference Library, Refbib for short. This started out in the 1990s as a common source for missile and aircraft simulations to be used within the Swedish defence community. A substantial effort was invested in these models in the form of CFD calculations, wind tunnel tests, data from manufacturers, analysis, and assumptions made by knowledgeable people in some cases where data were insufficient. The large organization that was behind this work in the 1990s has shrunk considerably, but there has been a continuous effort to keep the models up-to-date and to add new ones when needed.

The models are updated by use of manufacturers’ data whenever it’s available, and with data from calculations and estimates as necessary. The emphasis of the models is on dynamics, i.e. thrust, aerodynamic forces and inertia, while models of guidance systems and warheads are relatively simple at this time.

The initially chosen design for the models, simulation software and development tools was based on UNIX and FORTRAN, with extensive use of FORTRAN features such as COMMON blocks and DATA statements. Interfacing with other simulations is performed mainly by calls to FORTRAN functions and subroutines, and the core of the simulation is calls with fixed time steps to the subroutine Rbsim. One single call to Rbsim advances all active missile models one time step forward.

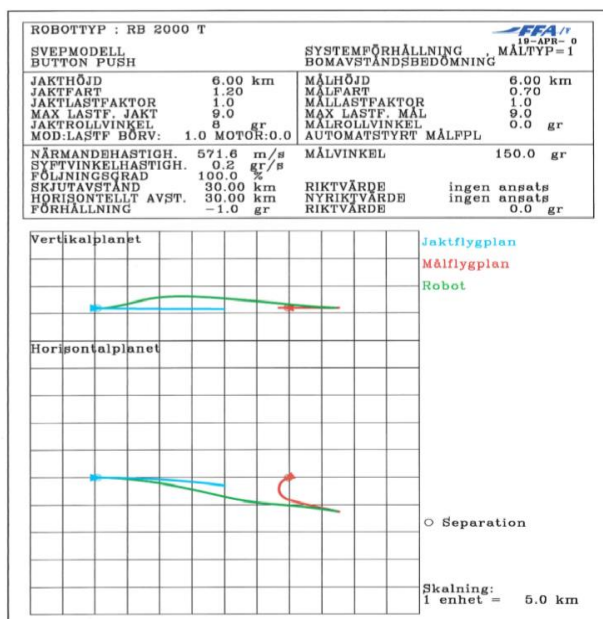


Fig. 5. Typical plot from missile model development

4.2 Aircraft Models, FMV TeknUnd and FLAMES

One important development effort in the preparations for the simulations was to create realistic aircraft models. Since the external models that were available consisted mainly of data tables, and because FLAMES comes with a simple aircraft model that is a good beginning for a more realistic model, it was chosen to build these models directly into the FLAMES architecture. FMV’s Technical Intelligence Department (FMV TeknUnd) can provide extensive performance data for various aircraft. The data comes in multi-dimensional tables such as C_{D0} as a function of altitude and Mach number, C_L as a function of angle of attack and Mach number, fuel consumption and thrust at full power as a function of Mach number and altitude, and a few others. This is translated into a table format called AER, which is a simple plain text format for multi-dimensional tables, see reference [4].

Table 1. A fictional aerodata table in AER format

CDICL2		
CDi/CL^2 as function of Mach and CL:		
140317		
2		
CL		
MACH		
0	0	0
0	0.7	0
0.1	0	0.1
0.1	0.7	0.15
0.1	1.	0.16
0.1	2.0	0.18
0.2	0	0.15
0.2	0.7	0.18
0.2	1.	0.19
0.2	2.0	0.12
0.5	0	0.15
0.5	0.7	0.18
0.5	1.0	0.10
0.5	1.5	0.09
1.0	0	0.12
1.0	0.7	0.15
1.0	1.0	0.16

FLAMES comes with complete source code for a large number of models known as Bundled Models. They should mainly be viewed as patterns for how to use every function in FLAMES, but they are also very useful as starting points for creating one’s own models. They are coded in C or C++ and provide a component based approach to including models in FLAMES simulations. The bundled aircraft model was modified to use data from the tables, and its equations of motion were updated to provide a three degrees of freedom aircraft model with a performance that is very close to what the data tables and diagrams provided by TeknUnd indicate. The models are driven by FLAMES’

built-in event-based simulation. However, our settings in FLAMES mean that everything is in essence a time-based simulation with a fixed time step of 0.02 seconds, just like Rbsim.

Specific Excess Power: 128 % fuel, ISA
 Aircraft configuration: F-16C Block 50, 1
 1WL ADM:120 2WL ADM:9 3WL no pylon 4WL no pylon 5PL no pylon 6 1:131 (D) max 77R no pylon 8WR no pylon 9WR no pylon 10WR ADM:9 11WR ADM:120

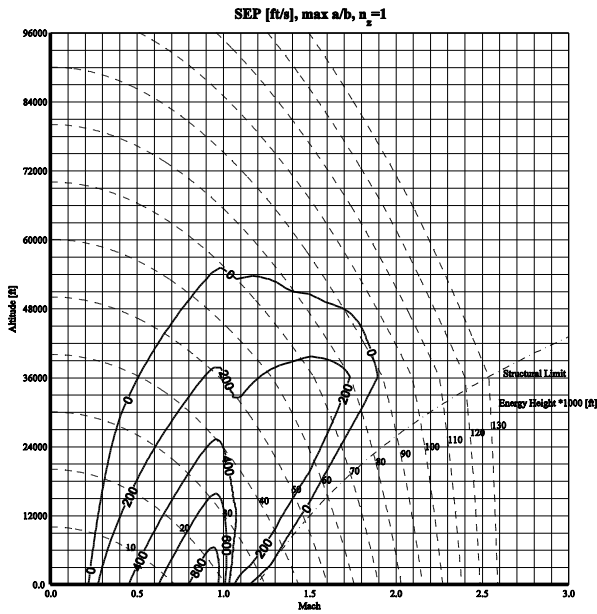


Fig. 6. Specific Excess power diagram used to verify table data

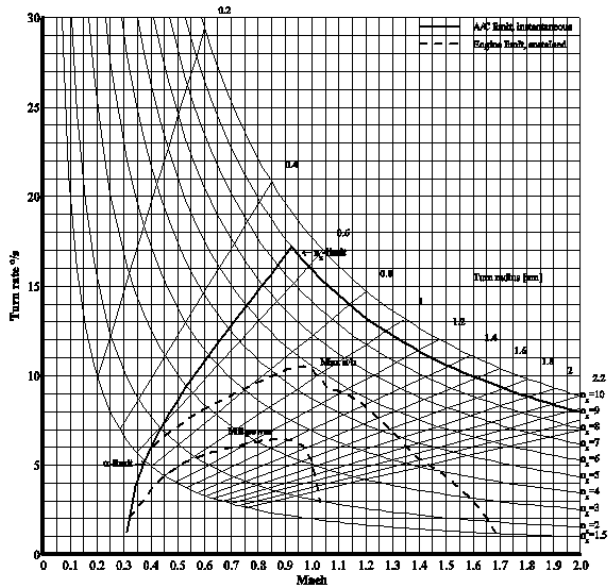


Fig. 7. Turn rate diagram used to verify table data, “open” F-16

The source code for the bundled aircraft model has been altered and expanded with the following additions:

It now includes fuel consumption, so the pilot model will attempt to return to base when the fuel level reaches “Bingo fuel”.

The weight and drag of external stores is taken into account.

If the pilot model tries to climb or turn too abruptly at low speed the plane will stall (but it will recover very quickly).

The attitude angles are calculated in a slightly more realistic way. Heading and path angle just follow the velocity vector. Roll is calculated based on load factor, path angle and inclination of the turning plane.

Large heading changes require a little cheating. The pilot models may demand a heading change of 180 degrees. If the nose of the plane is pointing down, the turning plane based on that attitude and that heading change may in part be located under ground level, which causes a crash. To avoid that, turns are made in pieces of twenty degrees, where the final attitude angle is reached almost immediately. It also means that large heading changes are almost always performed as level turns.

Time constants have been added for roll rate and load factor growth, to make the reversal of a turn take a realistic amount of time.

There are also limits for roll rate and for load factor.

The aircraft model takes the pilot model’s climb and acceleration demands and adapts them to a climb angle and speed that is better suited to what the aircraft can actually achieve.

4.3 Cognition Models, FOI’s MCGF

FLAMES includes bundled cognition models that can be used to create behaviours for all the units in the simulation. The project’s first demonstrations to the Swedish air force were made with the use of these models, but we were soon ordered to use MCGF instead. MCGF is an acronym for Merlin Computer Generated Forces. Merlin is a component-based architecture that was developed by FMV, FOI and others some ten years ago, to some extent with the ambition to create a common simulation architecture for the Swedish defence community, much like Refbib but with a more extensive and more modern approach. See reference [5]. The Merlin programs have their own time manager, and the time is advanced by a call from the FLAMES service at each time step. The cognition models decide about their actions approximately three times per second, simulator time.

FOI have chosen a behaviour tree approach to behaviour modelling within the Merlin architecture. The tree is defined in xml files that the user has full access to, while the actions, the “leaves”, used in the behaviour tree are defined in “ordinary” C++ methods where the access is left to a smaller number of experts. For more information about behaviour trees, see reference [6] and reference [7]. You can read a little more about the pilot models in reference [8].

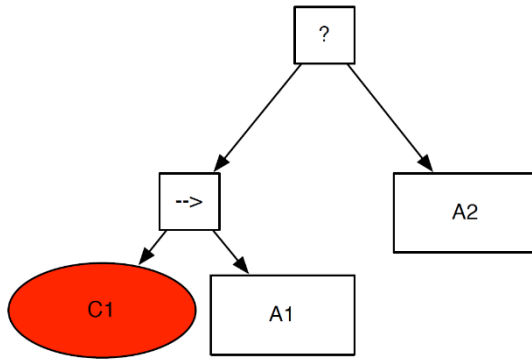


Fig. 8. Small part of behaviour tree, from reference [6]

The behaviour trees have been developed over a period of several years by scientists at FOI in collaboration with pilots and tactics experts at FLSC, the Swedish Force Air Combat Simulation Center. The trees represent three major behaviour types: blue fighters, red escort fighters and red air-to-ground attack.

The blue fighters' normal behaviour is to follow an air corridor until they reach the location for their combat air patrol pattern, then keep patrolling until they either need to return to base for lack of fuel, or until they encounter the enemy. If the enemy is detected the fighters will engage and continue fighting until they run out of fuel or missiles, or are shot down. If all enemies within sight are destroyed and the fighters still have fuel and missiles they go back to patrolling.

The red escort fighters will escort another group of red aircraft until they encounter the enemy. Then they will switch to air-to-air combat mode and behave quite similar to the blue fighters.

The red air-to-ground attack aircraft perform a mission that consists of following a pre-defined mission profile. If they reach the point where they can launch the missiles or drop their bombs, the mission is a success. No assessment of damage to the targets is made in the current simulations. The attack aircraft are fully aware of all other aircraft within a given pre-defined distance, and will try to avoid enemy fighters while keeping the formation together and following the mission profile.

The pilot models have four "behaviour controls" that are the main facilities for experimenting with pilot performance. They are Accuracy, Aggression, Speed and Randomness. It is not necessarily so that a high value is good and a low value is bad.

High accuracy makes the pilot model utilize missile performance more effectively but also makes its behaviour more predictable to the opponent.

High aggression increases the likelihood both of killing the enemy and of being killed.

High speed makes the pilot model react more quickly but sometimes also to act too soon.

Randomness has not been used so far, partly because it is already possible to get quite widespread outcomes by small changes to various parameters in the scenarios such as initial altitude, speed and position.

For the user, other things that can easily be changed and experimented with in the behaviour trees by editing

the xml files are altitudes, Mach numbers and formations used at the different stages of the mission, how close units must be for the pilot model to be interested, various time delays and reaction times, limits for path angles and limits for low level flight.

4.4 Environment model

The environment model is simple. The world is elliptical and the geodetic system is WGS-84. On top of that terrain elevation data can be added. Currently only terrain data for northern Europe is used, in DTED 0 resolution. The distance between elevation data points is then approximately 900 meters, which is quite sufficient for air-to-air scenarios. Smaller areas, size depending on available memory, can be modelled in DTED 1 or DTED 2 resolution, i.e. with approximately 90 meters or 30 meters between elevation data points. The elevation data is used to calculate line of sight data and sensor coverage, and to register flight into terrain. It is also visualized in the form of a map as seen in fig. 9 below.

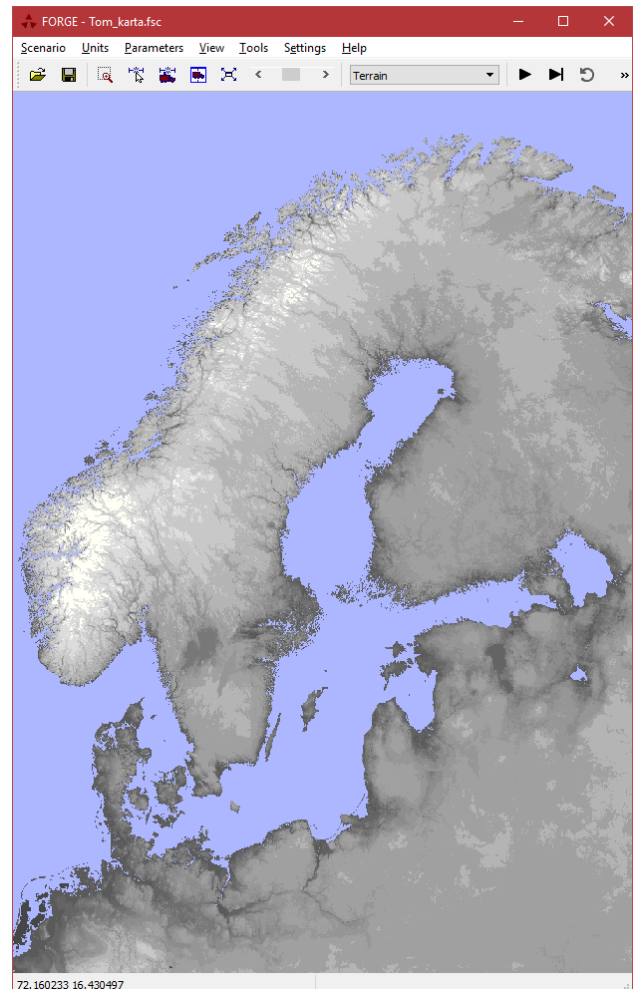


Fig. 9. Terrain created from DTED 0 data

The terrain data can also be seen in a perspective view that is sometimes useful to visualize how the different units move.

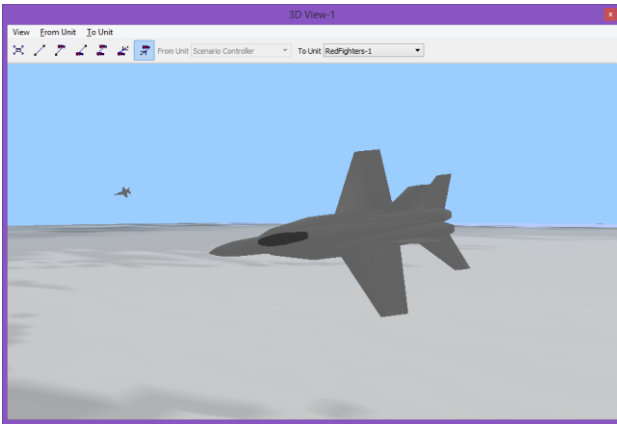


Fig. 10. Perspective view based on DTED 0 data

4.5 Other models

Sensors and weapon systems are modelled using the FLAMES Bundled Models with only minor changes. Sensors and countermeasures against sensors are the most important next steps in the model development.

5 Connection to FLAMES, Integration in MS Windows

The codes for missile models and for pilot models have been developed and tested in their own environments over a long time, and converting them to run within the FLAMES architecture was expected to require extensive changes that would be costly and potentially risky. An approach was needed where changes to the code were as small as possible.

Microsoft Windows is the main environment when working with FLAMES, and the main development tool is Visual Studio, and that had to be taken into consideration, although some versions of FLAMES are also available for Linux. The solutions for the missile models and the pilot models turned out a little different as will be shown below, although there are many similarities.

5.1 Missile models

Making the Rbsim and missile models work in Windows was a little challenging. The missile models and the entire set of tools that are used to develop them are based on FORTRAN and run in an Ubuntu environment, i.e. a Linux distribution. After looking over our alternatives we settled for the following solution:

The development and testing tools are still in Ubuntu, only the missile models and the programs necessary for running the simulation are used in Windows.

The FORTRAN compiler used is the one found in MinGW, “Minimalist GNU for Windows” that can be downloaded for anyone to use. MinGW also contains some necessary runtime DLLs.

The make utility is also a GNU product, from GNUWin32. The makefiles were altered a little in order to work both in the Ubuntu and the Windows environment.

To link the object code generated in MinGW together with Visual Studio code a number of special steps are needed:

You need to have the tool dumpbin available. This is achieved by running Visual Studio’s file vcvarsall.bat in a command window like this:

```
>C:\WINDOWS\system32\cmd.exe /k ""C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" x86
```

Then in the same command window you run your makefile that creates your .dll file.

The makefile for Windows has to include the following lines after the rbsim.dll file has been built in MinGW:

```
dumpbin /exports rbsim.dll
rem Now edit the rbsim.def file!
pause
lib /def:rbsim.def /out:rbsim.lib /machine:x64
```

During the pause the rbsim.def file must be edited to begin with EXPORTS on the first line, followed by the names of all the functions that dumpbin printed, like this:

```
EXPORTS
abstodb_
aktiv_rb_
alim_
angle_
angletohorizon_
antennagaindb_
.
.
zontrig_
zr_g1_
zr_rb_
```

The last line in the makefile then creates the file rbsim.lib that the Visual Studio linker can use to link the rbsim functions so they can be called by the service in FLAMES. This has proven to work very well, and with a few compiler directives and the test

ifeq (\$(OS), Windows_NT), the same makefiles can be used both in Ubuntu and in Windows. Rbsim.dll in Windows contains a number of special FORTRAN subroutines that can be called from C functions in FLAMES to submit and retract data to and from the missile simulation.

5.2 Pilot models

The MCGFs are normally developed and used in an Ubuntu environment, but for FLAMES use FOI re-built everything using Microsoft Visual Studio 2010 in Windows. It will be migrated to a later version of Visual Studio, but for the moment our main development environment is Visual Studio 2010.

FOI also provided an adapter between the pilot models and the aircraft model, sensor models and weapon system model that are run in FLAMES. It communicates with the FLAMES environment using the standard command, query and message utilities in FLAMES.

6 Visual Studio versions

FLAMES itself is currently developed using Microsoft Visual Studio 2017, but simulation developers can also use Visual Studio 2010 and Visual Studio 2013.

7 Scenario setup

FLAMES comes with complete source code for a tool named Scenario Controller. This has been extended for the project's needs and is a convenient way to set up and control scenarios. It is now typically used to set up formations of aircraft with initial altitudes, headings, speeds, positions and separations and send them out on their missions. Additional formations can be thrown into the scenario at predetermined points in time. Parameters that define the scenario can be inserted in the form of FLAMES scenario variables that can be set either from scripts or from a GUI.

Examples of scenario variables in fig. 11 below are BlueUnitsize and BlueLeaderStartLat.

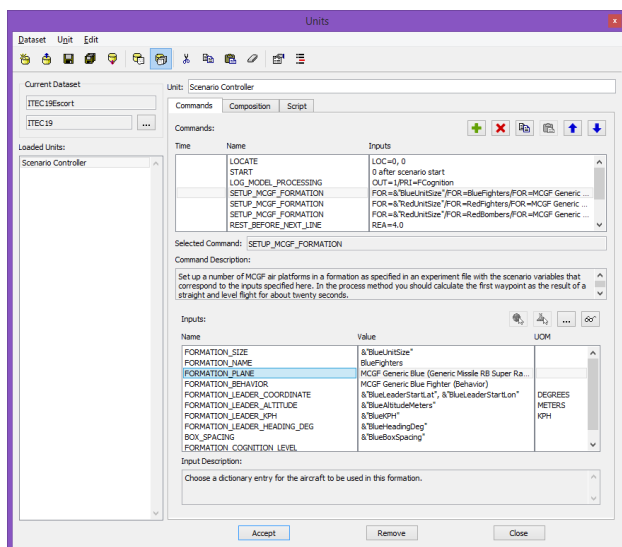


Fig. 11. Scenario Controller Window

Fig. 11 shows a Scenario Controller in the Unit Editing window, which is one of the main tools for setting up scenarios. Scenario Controller is the only unit that is set up directly in this example, and all aircraft units are set up by commands to Scenario Controller. Other units that can be added directly in the Unit Editing window are for example radar sites, various parts of the command chain, and ground vehicles. Commands that are used to determine what the unit will do are listed in the Commands window, and parameters to be used by each command are listed in the Inputs window.

7.1 Extensions to Scenario Controller

ASSIGN_COMMANDER_TO_FORMATION: Tells the leader of an aircraft formation that it should take orders from a controller on the ground.

ATTACK_GROUND_TARGET_IN_FORMATION: Tells a formation of attack aircraft to fly to a launch point and launch munitions against a ground target.

MCGF_SET_PROTECTION_GROUP: Orders MCGF controlled escort fighters to protect another formation of aircraft.

START_MCGF_FORMATION_ON_HEADING: Starts a formation of MCGF controlled aircraft on a given heading.

FLY_MCGF_IN_CORRIDOR: Orders an MCGF controlled formation to fly along a given air corridor.

SETUP_MCGF_FORMATION: Composes a formation of MCGF controlled aircraft with a given number of aircraft, at a given start time, at a given altitude and with given speed and heading. I.e. the formation will immediately start cruise flight. There are eight pre-defined formation types, and a "CUSTOM" formation can also be defined.

START_MCGF_FORMATION: Starts a formation that has been set up by SETUP_MCGF_FORMATION. Optionally tells the formation where its base is, so it can return to base.

The commands are tailored for the MCGF type pilot models, but there are also similar commands for the FLAMES type models.

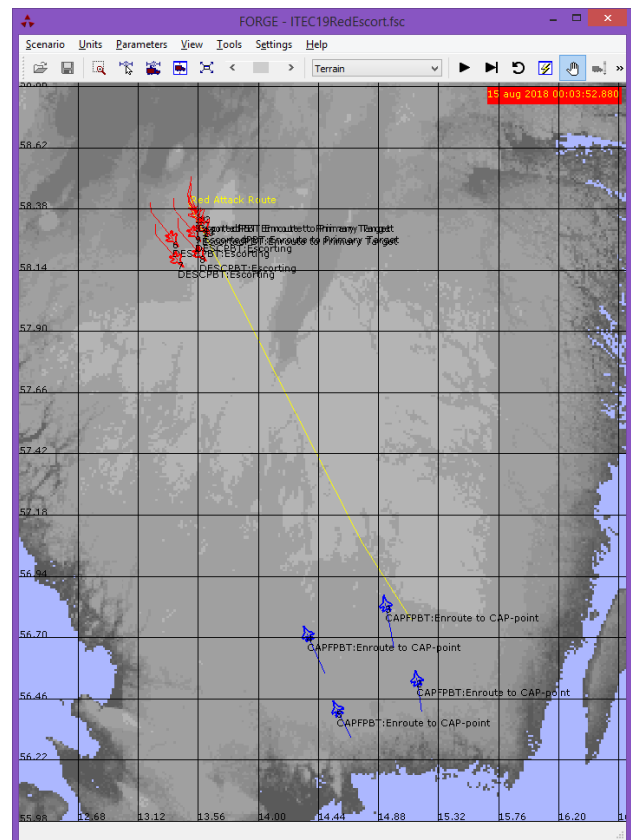


Fig. 12. Beginning of the scenario

8 Data collection and processing

The various simulation programs can log their own data, but it was chosen to channel all data logging through FLAMES. As default, FLAMES can log detects, launches, hits and misses, when objects are destroyed and the tracks of all objects, and some data about the simulation's performance. There is also the option to store a playback file that contains everything that is needed to play a recorded simulation back in the tool FLASH.

In addition to the above, any variable in the simulation

can be logged, although a bit of programming is needed to do this. Examples with complete source code are provided with FLAMES.

Most of the data logging is performed in a pure text format in .csv files, as labels and numbers separated by semicolons. That format is easily imported into MATLAB, Excel or a PostgreSQL database. MATLAB is mainly used to plot recorded data for debugging and verification. Excel is used for “simple” statistics, which usually are the most useful, while PostgreSQL can be used for more complex data processing. An advantage with PostgreSQL is that the PostGIS spatial database extender is convenient for storing and processing data for land areas and effects of air to ground munitions, for example to evaluate damage to runways.

8.1 PostgreSQL data processing example

A Java program provides a GUI for loading .csv log files into the database. The same GUI is then used for running a PostgreSQL script that provides statistics such as:

- How many aircraft participated on each side?
- Who detected who first?
- Who launched first?
- At what distance were opponents detected?
- Who were able to fire before being detected?
- Who were never detected at all?
- How many missiles were launched from each side?
- At what distances were the missiles fired?
- How many aircraft were destroyed on each side?
- In air to ground scenarios (currently not used):
- How many air to ground munitions were launched?
- How much of the target was destroyed?

8.2 Further data processing

Most of the data processing is performed by FOI, and the results are presented as statistics and converted into likelihoods of outcomes depending on a number of initial scenario parameter values such as number of aircraft, numbers and types of missiles, available fuel, altitudes and speeds. This is still a work in progress and will not be dealt with in detail here.

9 Future model development

The current setup is useful for developing methods and to simulate scenarios that are limited in space and time, but all models need to be refined to various extents. Sensors, communications, electronic warfare, surface to air and air to surface capability and the chain of command need to be modelled in more detail. Models of refuelling and rearming are available but are not currently in use. Models of logistics, maintenance and repairs can be added to create scenarios that span over several days, to give an idea of logistics requirements and perseverance.

10 Conclusion

The project began with the notion that there were a number of dissimilar data sets and models out there that would be beneficial to join together in a constructive simulation environment. We looked for a way to do this with a well-documented and supported tool. Our finding was that the choices were very few. The advantage we found with FLAMES over the few alternatives was that it has several customers and is well documented, that there were already people in the Swedish defence community that had substantial experience from using it, and that it had an already existing and functioning support organization. For an old C programmer the ways to incorporate components in FLAMES also look a lot less complicated than some of the alternatives.

This paper shows a way to take advantage of a COTS tool with built-in networking facilities, technical support and a wide user community, and combine it with in-house-developed software of various ages and architectures.

The work so far has mainly been a matter of using what we already had in a useful and efficient way, and create an environment that can be gradually improved. It is expected that the current sources will continue to contribute and that new partners will emerge.

Lessons learned:

There are plenty of things you don’t have to do yourself if you choose the right COTS tool.

Don’t rewrite code more than necessary, find other ways to integrate it with the simulation. Code is not obsolete just because it’s not developed using the latest methods.

Constructive simulation with advanced computer generated players and adequate system model fidelity is an excellent way to improve the understanding of how a dynamic situation can develop and what factors are important for the outcome, especially if scenarios can be set up and modified easily.

11 Terms and abbreviations

Bundled models	Models that are delivered with FLAMES, with complete source code. Contain examples of how (nearly) all functions in FLAMES are used.
BVR	Beyond visual range.
Combat Air Patrol (CAP)	Here: A flight pattern where two or more aircraft fly back and forth in a manner that at all times allows at least one of them to direct its radar in the direction from which the enemy is expected. Enemy aircraft are engaged when they get closer than a pre-determined distance.

Dataset	Group of data in the FLAMES database. Each dataset can contain data about for example a number of platforms or a number of behaviours. Scenarios are built mainly by configuring and changing datasets.
FIRE	FLAMES Interactive Runtime Executable (which ironically is the least interactive of the FLAMES programs). FLAMES program that runs simulations fast without graphics.
FLAMES	Flexible Analysis and Mission Effectiveness System. Simulation framework supplied by Ternion Corporation.
FLASH	FLAMES Scenario Highlighter. FLAMES program that is used to visualize recorded simulation data.
FLSC	The Swedish Air Force Air Combat Simulation Center.
FOI	The Swedish Defence Research Agency.
FMV	The Swedish Defence Materiel Administration.
FMV TeknUnd	FMV's Technical Intelligence Department.
FORGE	FLAMES Operational Requirements Graphical Editor. FLAMES program that is used for setting up and testing of scenarios.
Event driven simulation	The simulation is not run at a steady pace, but is driven by scheduled events.
MBDA	European missile system supplier. Manufacturer of the Meteor air-to-air missile. Owned by Airbus, BAe Systems and Leonardo.
MCGF	Merlin Computer Generated Forces. Behaviour models developed by FOI. Based on the Merlin architecture and behaviour trees.
Merlin	A component based software architecture for simulation models. Developed by FOI and FMV. Currently used by FLSC, and others.
Mermoc files	Xml (extended markup language) files that are used to define various characteristics in MCGF, for example the structures of the behaviour trees.

Pattern	The highest model aggregation level in FLAMES. A unit in a FLAMES simulation is often an instance of a pattern. A pattern usually consists of two or more dictionary entries.
Refbib	"The reference library". A large number of FORTRAN source code files and data files that define a number of aircraft and missile models suitable for "easy" inclusion in various simulation environments. For example FLAMES.
Saab	Swedish company. Important supplier to the Swedish armed forces, for example of the Gripen combat aircraft.
Scenario variables	Variables that are stored in datasets in FLAMES and can be used to configure units and scenarios.
Time based simulation	The simulation is performed with constant length time steps, and events happen because of the states that are integrated forward in the various models.
Weapon System	Model in FLAMES that represents the interface between the human behaviour model and the munition. Sometimes also automates some of the behaviour model's tasks.
Unit	A unit in FLAMES is for example a complete instance of an aircraft with pilot, armament, sensors, radio, countermeasures and so on. It could also for example be a missile, a vehicle with driver or a ship with crew and all on-board systems.

References

- [1] 1278.1-2012 - *IEEE Standard for Distributed Interactive Simulation--Application Protocols* (2012)
- [2] B. Möller, M. Karlsson, B. Löfstrand, Å. Wihlborg, S. Eriksson, M. Johansson, F. Klasson, P. Svensson, P. Aktanius, *The HLA Tutorial, a Practical Guide for Developing Distributed Simulations*, Pitch Technologies AB (2012)
- [3] SISO-REF-010-2015, *Reference for Enumerations for Simulation Interoperability, Version 21* (2015)
- [4] T. Stavöstrand, U. Teige, B. Sundberg, O. Karlström, B. Busk, *Generiska Aerodatamodellen*, Saab AB (1996).
- [5] E. Salling, P. Strömbäck, J. Lindh, C. Hildings, N. Wallin, M. Fredriksson, *MERLIN Software Architecture Document*, FOI-R--2486--SE, (2012)
- [6] K. Hultgren, A. Rensfelt, E. Salling, P. Ögren, *En introduktion till beteendeträd*, FOI Memo 4630, (2013)
- [7] Kristoffer Hultgren, Anna Önehag. *Behavior Trees*. FOI-D--0748--SE, 2016-11-14
- [8] Kristoffer Hultgren. *MERLIN Computer Generated Forces (MCGF)*. FOI MEMO 5529, 2015-12-04.

Authors

Gunnar Hovmark is a consultant employed by ÅF, an engineering and design company within the fields of energy, industry and infrastructure. He has recently worked in various simulation related projects within the Swedish Defence Materiel Administration, FMV. He is a graduate from the aeronautical engineering programme at the Royal Institute of Technology in Stockholm and has many years of experience of virtual and constructive simulations within The Aeronautical Research Institute of Sweden and the Swedish Defence Research Agency.

Fredrik Jonsson is working as a Project manager for M&S at the Swedish Defence Materiel Administration (FMV). He holds a M.Sc. degree in Mechanical Engineering from Linköping Institute of Technology (LiTH), Sweden. Over the last fifteen years he has been working with different M&S projects focused on M&S infrastructure and design. Today he is responsible for a project that aims to develop and maintain an ability to use simulation to support the Swedish Armed Forces defence planning activities.