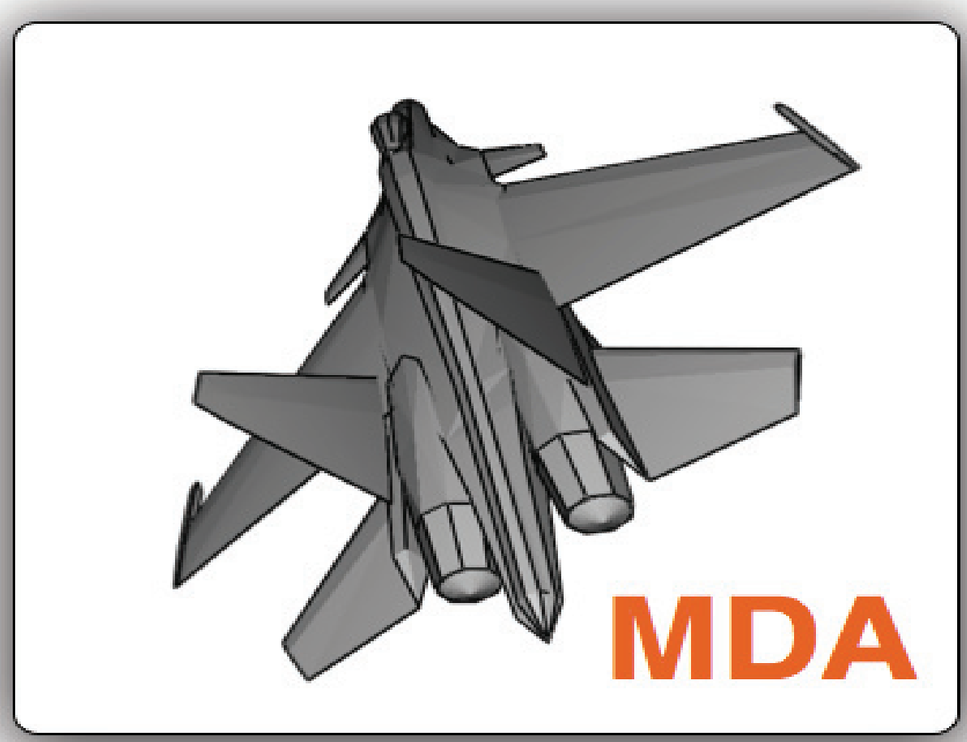
MERLIN Defence Analyzer v1.2

User Manual

EMIL SALLING, PETER STRÖMBÄCK



FOI, Swedish Defence Research Agency, is a mainly assignment-funded agency under the Ministry of Defence. The core activities are research, method and technology development, as well as studies conducted in the interests of Swedish defence and the safety and security of society. The organisation employs approximately 1000 personnel of whom about 800 are scientists. This makes FOI Sweden's largest research institute. FOI gives its customers access to leading-edge expertise in a large number of fields such as security policy studies, defence and security related analyses, the assessment of various types of threat, systems for control and management of crises, protection against and management of hazardous substances, IT security and the potential offered by new sensors.

FOI
Defence Research Agency
SE-164 90 Stockholm

Phone: +46 8 555 030 00
Fax: +46 8 555 031 00

www.foi.se

FOI-R--3625--SE
ISSN 1650-1942

December 2012

Emil Salling, Peter Strömbäck

MERLIN Defence Analyzer v1.2

User Manual

Titel	MERLIN Defence Analyzer v1.2 Användarmanual
Title	MERLIN Defence Analyzer v1.2 User Manual
Report no	FOI-R--3625--SE
Month	December
Year	2012
Pages	48
ISSN	ISSN-1650-1942
Customer	Swedish Armed Forces
Project no	E36500
Approved by	Lars Höstbeck Head, Division for Information and Aeronautical Systems
Division	Information and Aeronautical Systems

FOI Swedish Defence Research Agency

This work is protected under the Act on Copyright in Literary and Artistic Works (SFS 1960:729). Any form of reproduction, translation or modification without permission is prohibited.

Abstract

MERLIN Defence Analyzer (MDA) is an application for interactively running and analyzing MERLIN models. This manual describes how to use and install version 1.2 of the application.

The intended reader of this document is working with simulation models for assessing and analyzing weapon systems and computer generated forces. The document covers how to use the components and functionality provided in the v1.2 release of MDA. Development of components and plugins are not covered.

MDA includes the capability to generate scenarios of multiple platforms, visualize behaviors, control simulations by setting simulation time step, logging, replaying of generated results as well as parameter variation on full scale scenarios.

Keywords

MDA, Simulation, MERLIN, MERLIN Defence Analyzer

Sammanfattning

MERLIN Defence Analyzer (MDA) är ett program för att interaktivt köra och analysera MERLIN-modeller. Denna manual beskriver hur man installerar och använder version 1.2 av MDA.

Dokument riktar sig till läsare som arbetar med simuleringsmodeller för utvärdering och analys av vapensystem och syntetiska aktörer och innehåller en beskrivning av komponenter och funktioner. Utveckling av komponenter och pluginer beskrivs ej.

I MDA finns förmågan att sätta upp scenarion innehållande flera plattformar, visualisera beteenden, styra simuleringsingstid, logga resultat, spela upp tidigare körningar och genomföra parametervariation på fullskaliga scenarion.

Nyckelord

MDA, Simulering, MERLIN, MERLIN Defence Analyzer

Contents

1 Introduction **9**
1.1 Scope . . . 9
1.2 Definitions, Acronyms and Abbreviations . . . 9
1.3 Intended Use . . . 10
1.4 Supported Platforms . . . 10
1.5 Installing MDA . . . 11
1.5.1 Prerequisites . . . 11
1.5.2 Environment Variables . . . 11
1.6 Running MDA . . . 12
1.7 Reference Frames . . . 12

2 Components in MDA **13**
2.1 Launch Missile . . . 14
2.2 Signatures . . . 14
2.3 Radar . . . 14
2.4 Logging . . . 14
2.5 Drawable Object Model . . . 14
2.6 EMChannel Drawables . . . 14

3 User Interface **15**
3.1 Scenario Inspector . . . 15
3.1.1 Entities Tab . . . 15
3.1.2 Actors Tab . . . 16
3.1.3 Drawables Tab . . . 16
3.2 Entity Inspector . . . 17
3.2.1 Info Tab . . . 17
3.2.2 Actors Tab . . . 18
3.2.3 Drawables Tab . . . 18
3.2.4 Attachables Tab . . . 18
3.3 3D Scenario View . . . 19
3.4 Toolbar . . . 19
3.5 Camera Navigation . . . 19
3.6 Keyboard Shortcuts . . . 19

4 Creating Scenarios and Entities **21**
4.1 Adding Entities . . . 21
4.2 Decorating Entities . . . 21
4.3 Create an Entity Prototype . . . 21

4.4 Editing a Prototype 21

5 Running a Scenario **23**
5.1 Preparing the Carrier 23
5.1.1 Actor for Launching Missiles 23
5.1.2 Aircraft Radar 24
5.2 Preparing the Target 25
5.3 Saving the Scenario 26
5.4 Firing the Missile 26
5.5 Running the Simulation 26
5.6 Stopping/Reloading a Scenario 26
5.7 Inspecting the Missile 26
5.8 End Conditions 27
5.9 Default Scenario Actors 27
5.10 Extend Visualization 27

6 Logging **29**
6.1 Scenario Entity Logger 29
6.1.1 Log structure 29
6.1.2 File entity.xml 29
6.1.3 File entity.log 30
6.1.4 File event.log 31
6.2 Component Logger 31
6.2.1 Manually adding a Component Logger 32
6.2.2 Automatically creating Component Loggers 32
6.2.3 Log Files 33

7 Parameter Variation **35**
7.1 Overview 35
7.2 Example 35
7.2.1 Step 1. Design the Scenario 36
7.2.2 Step 2. Setup End Conditions 36
7.2.3 Step 3. Add Loggers 37
7.2.4 Step 4. Save Scenario 37
7.2.5 Step 5. Configure Parameter Variation 37
7.2.6 Step 6. Running the Parameter Variation. 39
7.2.7 Step 7. Analyze Results 39

8 Overview of MDA Extensions **41**
8.1 Joystick support 41
8.2 HLA networking 41
8.3 Computer Generated Forces - CGF 41

8.4 Missile simulation models 41

A MDA Components **43**

B Scenario File Format **45**

1 Introduction

The Swedish Defence Research Agency has developed a realtime architecture for simulations of airborne weapon systems and platforms. This architecture is called MERLIN.

This document describes MERLIN Defence Analyzer (MDA) which is an application for interactively running and analyzing MERLIN models such as aircrafts and missiles. In MDA these models are referred to as scenario entities, or just entities for short. Entities can have subsystems such as radars and signatures. The behavior of both entities and subsystems can be studied in MDA.

MDA has several ways of running, controlling and displaying different features of the models. This is done through three fundamental concepts in MDA; *Actors*, *Drawables* and *Attachables*. These will be explained in more details in chapter 2.

MDA is built upon the concept of plugins. This makes it possible to extend MDA and adapt it for your specific needs by using the built-in components, as well as adding your own extensions. MDA allows you to extended the functionality of your simulation models and GUI.

1.1 Scope

This document describes version 1.2 of MDA and is intended to be a user manual. It covers how to use the components and functionality provided in the v1.2 release, but does not cover how to develop your own components for MDA.

1.2 Definitions, Acronyms and Abbreviations

$AVALON_HOME is the path to your MERLIN installation folder. This path is typically set using your system environment variables. Using the `$AVALON_HOME` path MDA knows where to search for MERLIN model dependencies and extensions to the core application. The MDA application is also installed in `$AVALON_HOME/MDA`.

CGF Computer Generated Forces.

Component A software component is a module that encapsulate a set of related functionality and data.

DIS Distributed Interactive Simulation (DIS) is an IEEE standard for realtime simulations. See also `http://en.wikipedia.org/wiki/Distributed_Interactive_Simulation`.

EMChannel is a software component in MERLIN that handles the distribution of electromagnetic signals between entities.

EMEmitter is a software component modeling an emitter of electromagnetic radiation.

EMReceiver is a software component modeling an object that receives and interprets electromagnetic radiation. A radar is typically both an EMEmitter and an EMReceiver.

EMReflector is a software component modeling objects that reflects electromagnetic radiation, like aircrafts.

HLA High Level Architecture. See `http://en.wikipedia.org/wiki/High_-Level_Architecture_(simulation)` for more information.

IR Infra Red.

MERLIN is a software architecture used for real time simulation models. MDA is constructed to run and evaluate models developed using MERLIN. MDA also uses the MERLIN architecture for its own building blocks, See MERLIN software architecture document for an overview of this package[1].

MERMOC A mermoc is an xml-file describing a model composition and a fundamental part of MERLIN. MERMOC is the abbreviation for MERLIN Model Composition and the abbreviation mostly occurs as file extension the the model compositions (*.mermoc).

MDA MERLIN Defence Analyze, a software for analyzing scenarios and simulation models for defence purposes.

Qt Qt is a GUI platform independent framework used in to create the graphical user interface. See `http://qt.digia.com` for more information about Qt.

RCS Radar Cross Section.

WGS84 World Geodetic System 84, is a standard describing the features of the earth and often used in geodesy, aerospace and navigation applications. See `http://en.wikipedia.org/wiki/World_Geodetic_System`.

1.3 Intended Use

MDA was initially designed to aid in the evaluation and verification of missile models in air-to-air duels. MDA was then an abbreviation for MERLIN Duel Analyzer. Today it has been used for several different applications and nowadays MDA is an abbreviation for MERLIN Defence Analyzer. A few of the applications in which MDA has been used are listed below:

- Studying missile duels using MERLIN missile models.
- Verifying developed missile models by analyzing their behavior.
- Developing and running CGF models.
- Scenario parameter variation studies.
- Evaluation of chaff bundle radar interaction.

1.4 Supported Platforms

MDA is built on a platform independent architecture which means that most common platforms are supported. However, in the 1.2 version release of MDA no prebuilt packages are available for Microsoft Windows.

Prebuilt releases of MDA are available for 32 and 64bit Linux.

[1] Salling et al, MERLIN Software Architecture Document, FOI-R–2486–SE, 2009

1.5 Installing MDA

MDA is delivered as a compressed file, `MDA_1_2_Linux_i686_Release.tar.gz`, and installed as a standard MERLIN product by decompressing the file into a directory called `AVALON_HOME`. For more information about MERLIN installations see *MERLIN Software Architecture Document*, FOI-R–2486–SE.

1.5.1 Prerequisites

MDA has a few MERLIN dependencies. In your `AVALON_HOME` directory you must have the following packages:

- MERLIN version 1.5.3
- MERLIN_EXAMPLES version 1.4.7
- MERLIN_UTILITIES version 1.2
- MDA version 1.2

You also need a few third party libraries installed on your system.

- Qt 4.6 (or higher)
- OpenGL
- libxml2
- zlib
- Java 6.0 (or higher)

It is recommended that these are installed using the operating system's own package management system. For Ubuntu and Debian distributions installing third party libraries can be done by using `apt-get`. Note that zlib and libxml2 are often preinstalled in linux/unix distributions.

```
apt-get install libqt4
apt-get install libxml2
apt-get install zlib1g
```

On some systems you might need to complement the libqt4 with libqt4-webkit.

```
apt-get install libqt4-webkit
```

1.5.2 Environment Variables

In order to let MERLIN know where all deployed products are located, an environment variable must be set called `$AVALON_HOME` and the operative system needs to know where the shared libraries are located. On Linux and Unix this is done by setting the following environment variables (using bash):

```
export AVALON_HOME=YOUR_PATH_TO_AVALON_HOME
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$AVALON_HOME/merlin/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$AVALON_HOME/merlin_utilities/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$AVALON_HOME/MDA/lib
```

1.6 Running MDA

If your environment variables are correctly set you should now be able to run MDA by writing the following command in a terminal:

```
> $AVALON_HOME/MDA/bin/MDA
```

When MDA is started from the command line there are some options available. These will be displayed if you add the `--help` option as shown below.

```
> $AVALON_HOME/MDA/bin/MDA --help
MDA - MERLIN Defence Analyzer
Usage: MDA [OPTION..]

Options:
--help   : print help
-h       : print help
-help    : print help
-s <arg> : start up with scenario (absolute mermoc path)
-wgs84   : startup simulation in WGS84 round world model

Examples:

Start MDA with scenario.mermoc preloaded:
> $AVALON_HOME/MDA/bin/MDA -s $PWD/scenario.mermoc

Start scenario in wgs84 mode
> $AVALON_HOME/MDA/bin/MDA -wgs84
```

1.7 Reference Frames

By default, MDA starts up in a flat earth, NED-frame, where vectors expressed in the fixed inertial frame are expressed relative to North, East and Down. It is also possible to run MDA with a WGS84 earth representation where vectors expressed in the inertial frame are expressed in ECEF (Earth-Centered-Earth-Fix) coordinates.

To run MDA in a WGS8 mode add the `-wgs84` option to the command line. MDA will then start up with a default scenario in WGS84. E.g.:

```
> $AVALON_HOME/MDA/bin/MDA -wgs84
```

When running in WGS84 mode, the position of the entities are expressed in longitude and latitude in the entity inspector view for convenience.

Note that a scenario has to be constructed in WGS84 *or* flat earth. In version 1.2 of MDA there is no functionality to convert between scenarios defined in different frames.

MDA stores which reference frame that is used in the scenario file and automatically switches to the correct frame when opening the scenario.

2 Components in MDA

Components are an important concept in MDA that bring different types of functionality to a scenario and allows the user to add/change behaviors or appearances in the scenario, such as adding a RCS-signature to a target, coloring hostile entities red, generating log files for all entities in the scenario, etc.

Components in MDA are organized into different categories. *Scenario components* are separated from *entity components* (see figure 2.1), where scenario components act globally on the whole scenario and entity components act on a single entity in the scenario.

The scenario- and entity components are divided into *actors*, *drawables* and *attachables*. The concept of actors and drawables are available as both scenario components and entity components while attachable are only available for entity components.

Actors are components that influence the simulation, often by user input. An example of a entity actor is a component that launches a missiles when the user presses a button.

Drawables are components that add visualization to the scenario or to the entity.

Attachables are passive components that add extended (but passive) functionality (like and RCS- or IR-signatures).

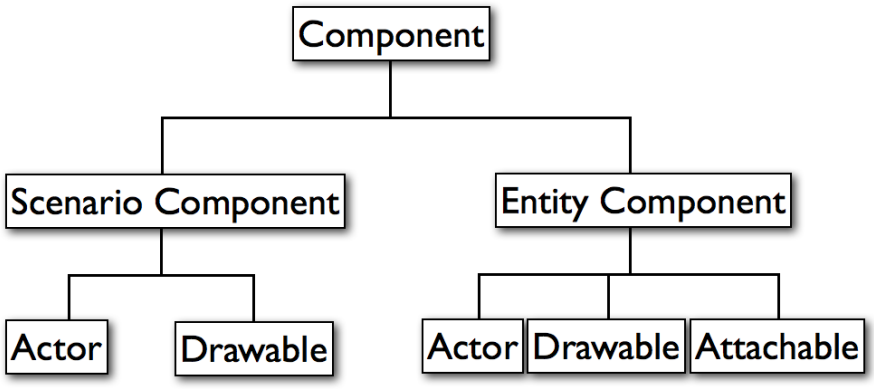


Figure 2.1: The components in MDA are grouped into *scenario components* and *entity components*. The concept of *actors* and *drawables* are available for both scenario components and entity components while attachables are only available for entity components.

Section 3.1 covers the user interface for accessing the *scenario components* and section 3.2 for accessing the *entity components*. In the following sections some of the more important components (actors/drawables/attachables) that are shipped with MDA are reviewed. In appendix A a complete list of all the components shipped with the standard release of MDA is presented.

2.1 Launch Missile

The LaunchMissile entity actor equips an entity with the capability to fire munitions. Most aspects regarding preparation of a MERLIN missile can be configured in the GUI. You are able to choose among different firing modes, engagement order number and target. IR missiles can be opened and slaved while attached to the aircraft.

2.2 Signatures

If you want your entities to be detectable by various systems, you must add suitable signatures to them. Radar systems require the existence of a radar cross section (RCS) signature for an entity to be detected. Infrared (IR) systems require an IREmitter signature to be detectable. In MDA signatures are implemented as attachables.

2.3 Radar

If your missile needs radar up-link, you must setup your carrier with a radar. This is performed in the same way as adding a signature to an entity (see section 2.2). In all MDA installations there is an ideal radar called merlin_RadarProxy, which you can use for this purpose.

2.4 Logging

When you examine a scenario (or many sets of scenarios) you likely need to log various data from the simulation. There are two different types of logging actors included in MDA, the Scenario Entity Logger and the Component Logger. The Scenario Entity Logger logs the position, velocity, orientations and events for all the entities in the scenario. The Component Logger acts on a specific entity and is capable of logging a significant part of all sub models of the entity. In chapter 6, a more detailed description of logging is given.

2.5 Drawable Object Model

This entity drawable let you associate 3D-model (a Wavefront 3D obj file) for visualization of your entity. There is a collection of object models shipped with MDA that you can use.

2.6 EMChannel Drawables

There is a collection of scenario drawables that draw the EMRecievers, EMReflectors and EMEmitters currently registered in EMChannel.

3 User Interface

The MDA user interface revolves around the main window which can be seen in figure 3.1. The main window consist of four sections. To the left is the *Scenario Inspector*, which contains information related to the scenario. In the middle there is a *3D view* of the scenario is displayed, and to the right the *Entity Inspector* is displayed which contains information related to the currently selected entity. A toolbar with control for the scenario runtime is available in the top of the main window.

Both the *Scenario Inspector* and the *Entity Inspector* can be rearranged within the main window. You can also resize them and drag them out of the main view to become floating tool views. MDA is filled with tooltips that appears when hovering over the different GUI elements with the mouse pointer.

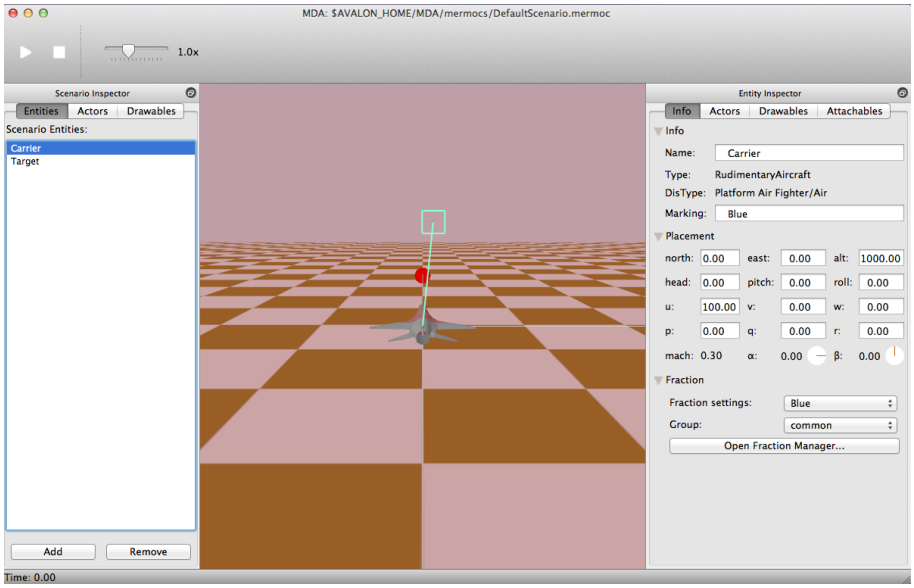


Figure 3.1: MDA user interface. To the left is the *Scenario Inspector*, to the right is the *Entity Inspector* and in the middle the scenario is displayed in 3D.

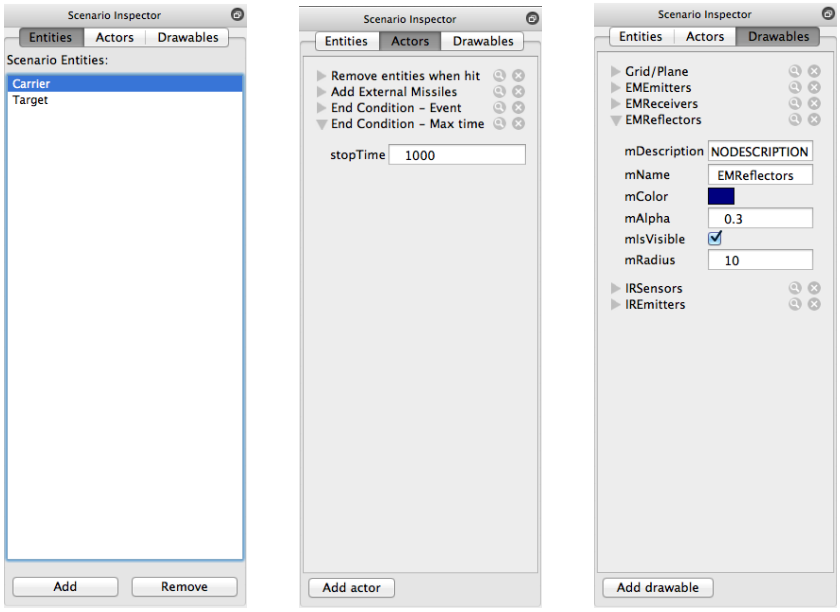
3.1 Scenario Inspector

Properties related to the scenario is provided in the the *Scenario Inspector*. It contains three different tabs: Entities, Actors and Drawables.

In the bottom of each tab view there is an Add button to add entities, actors or drawables, depending on the active tab. For the Entities tab there is also a remove button. Removing scenario actors and drawables is instead done by pressing the cross icon, see figure 3.2.

3.1.1 Entities Tab

The Entities tab (see figure 3.2a) contains a list of all entities in the scenario. To select an entity, you can left click the corresponding entity name in the list. When the selection is made, the selected entity will receive camera focus in the



(a) Scenario Entities. (b) Scenario Actors. (c) Scenario Drawables.

Figure 3.2: Scenario Inspector views.

3D-view and the *Entity Inspector* will update and display the selected entity properties.

By right-clicking an entity name a popup menu appears with options for editing the entity. This allows you to save the configured entity, create a duplicate (clone) and add actors, drawables and attachables to the entity. The menu contains the same information as the *Entities* in the applications menu.

Renaming an entity is done in the *Entity Inspector*, and doing so will also change the name in the entity list.

3.1.2 Actors Tab

This tab (see figure 3.2b) displays all scenario actors contained in the scenario. The scenario actors helps you to control the simulation. You can add scenario actors to the scenario by clicking the *Add actor* button at the bottom of the tab. Press the add button and a file dialog displaying all available scenario actors will appear. You can remove existing scenario actors by pressing the remove icon, placed on the right side of the component name.

3.1.3 Drawables Tab

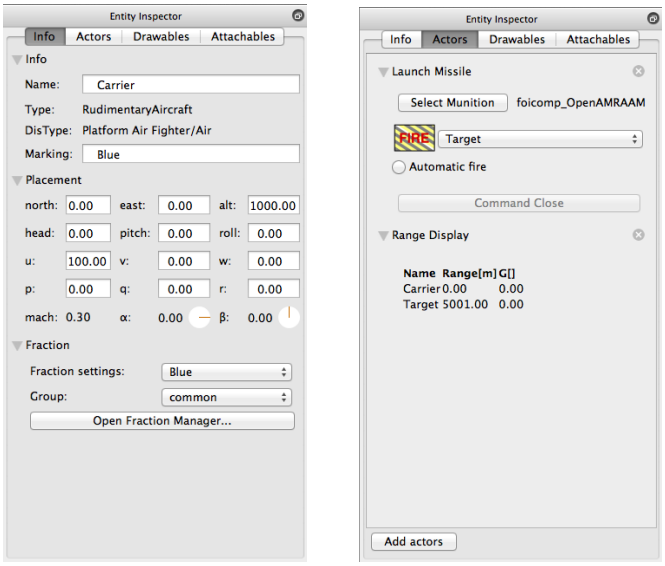
This tab (see figure 3.2c) displays all scenario drawables contained in the scenario. Similarly to the Actors tab, you can add scenario drawables to the scenario by clicking the *Add drawable* button at the bottom of the tab. You can remove existing scenario drawables by pressing the remove icon, placed on the right side of the component name.

3.2 Entity Inspector

The entity inspector, which is located in the right section of the MDA main window, displays properties of the currently selected entity. It consists of four tabs: Info, Actors, Drawables and Attachables (see figure 3.3).

The actor, drawable and attachable tabs contain elements that decorate the platform. These can be a subsystem such as a radar signature (an attachable), a 3D-graphics model of an F-16 (a drawable) or a user interface to select, configure and fire a missile (an actor). As a user you can add a lot of features to the entities by adding different sets of actors, drawables and attachables.

A set of different components are included with the MDA release. A complete list of these components is available in appendix A. You can also browse the different available components by adding a new component. MDA will then display a list of all available components with their corresponding descriptions.



(a) Entity Info tab.

(b) Entity Actors tab.

Figure 3.3: The Entity Inspector has four tabs. Here the Info tab and the Actors tab are displayed.

3.2.1 Info Tab

The Entity Information tab (see figure 3.3a) has three sections, Info, Displacement and Fraction. The first section of the entity information tab displays the entity instance name (a unique name in the simulation), a platform type name, and the text string version of the DIS type. There is an editable field for entity marking, which is a string that can contain application/scenario specific entity descriptions.

The Displacement section is where you define and view the placement of platform. There are editable fields for position and orientation orientation with respect to the fixed inertial frame and velocity and angular velocity with respect to the body frame. Mach number, angle of attack (alpha) and angle of sideslip (beta) are displayed as well. Hover with your mouse over the fields to display the unit of the field.

The Fraction section of the Info tab display which fraction and group the entity

is part of. Fraction settings defines which team the entity is a member of, for instance *Red*, *Blue* or *Neutral*. Group is a marker for a group within the fraction and could for example be *Strike One* or *Escort Two*. The fraction and group are especially important when running computer generated forces simulations (see section 8.3). When running a non-CGF scenario, fraction and group is just a marker and has no effect on the simulation.

3.2.2 Actors Tab

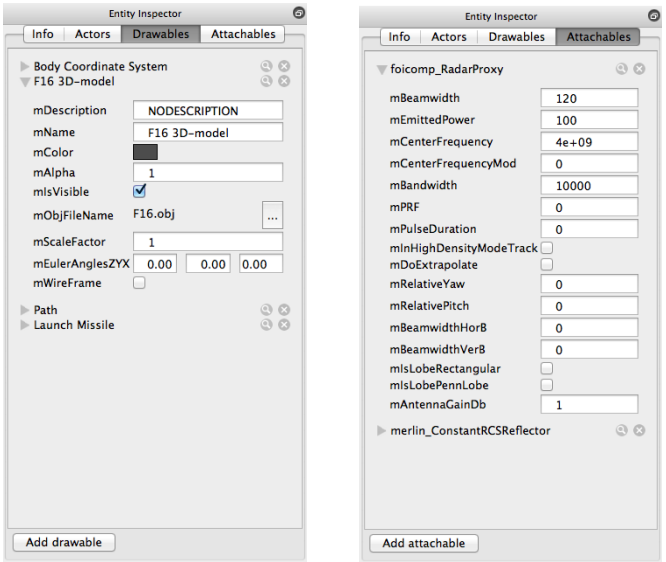
In the Actors tab all actors associated with the selected entity are listed, see 3.3b. An entity actor is a component that allows you to "act" on the entity in some way. Adding actors to an entity is typically the way to add user interaction to an entity, or to influence the simulation.

3.2.3 Drawables Tab

In the Drawables tab all drawables associated with the selected entity are listed, see figure 3.4a. An entity drawable, as the name implies, adds some graphical properties to the platform. A typical example is when you want to add a 3D representation of your entity or a drawable that display the trajectory of entity.

3.2.4 Attachables Tab

In the Attachables tab all attachables associated with the selected entity are listed. Attachables are a special kind of components that implement the MERLIN interface called IToScenarioEntityAttachable. This is an interface constructed for runtime connection of an entity to a subsystem in MERLIN. Examples of attachables are signatures (IR, RCS), radars, chaff or flare ejecting systems.



(a) Entity Drawables tab.

(b) Entity Attachables tab.

Figure 3.4: The Entity Inspector has four tabs. Here the Drawable tab and the Attachable tab are displayed

3.3 3D Scenario View

The 3D view in MDA displays the scenario. What will be displayed in the 3D view is determined by the added scenario entities and their corresponding *Drawables*. The scenario may also have scenario drawables adding graphics to the scene. By adding your specific set of *drawables* you can emphasize what is important to visualize in your problem set. The camera is always focused on the currently selected scenario entity. User interaction with the 3D view is described in section 3.5.

3.4 Toolbar

The toolbar at the top of the MDA window expose functionality to control the scenario runtime, start/pause and stop (reload) the scenario. You can use the slider to alter how fast the simulation runs to make simulation time progress faster than clock time or slower. By default, this slider is set to 1.0. I.e. one second of simulation time equals one second of real world time. 2.0 means the simulations runs twice as fast as real time.

Figure 3.5: The UI toolbar has controls for starting, stopping and pausing the scenario. You can also change the realtime multiplier using the slider.

Pressing the stop button will reload the scenario. Use the pause button for pausing. You can also press the spacebar to pause the scenario.

3.5 Camera Navigation

Camera navigation in the MDA 3D view is very simplistic at this time. The camera always focuses on the currently selected entity. By left mouse clicking and dragging in the 3D view you change the view angle of the entity. By default you will rotate around the down vector and control pitch relative to the ground. You can change the camera distance from the entity by using the mouse scroll wheel or by clicking META (Ctrl on Windows/Linux, Cmd on OS X) and left mouse drag up and down in the 3D view. In the camera menu there's also a *Focus* command that set the camera to focus at close distance from the entity (shorthand is Ctrl/Cmd F).

Pressing and holding shift while navigating the camera will speed up the camera movements.

The camera menu also exposes a toggle command called *Switch follow host*. If you enable this setting, the camera will roll along with the selected entity.

3.6 Keyboard Shortcuts

Some predefined keyboard shortcuts are available. These are listed in table 3.1

Table 3.1: Listing of keyboard shortcuts.

Name	**Linux**	**OS X**
Open Scenario	Ctrl+O	Cmd+O
Save Scenario	Ctrl+S	Cmd+S
Save Scenario As	Shift+Ctrl+S	Shift+Cmd+S
New Scenario	Ctrl+N	Cmd+N
Start/Pause	Space	Space
Focus Entity	Ctrl+F	Cmd+F

4 Creating Scenarios and Entities

On startup, MDA displays a simple default scenario. This scenario consists of two aircrafts called Carrier and Target. The target aircraft has been provided with both EM- and IR signatures. The carrier is prepared with a radar and an actor to be able to fire missiles.

Using the default scenario as a start is often a good idea for a new users, but if you want to start a new scenario from scratch, select *New Scenario* under the File menu. A new scenario will contain no entities, no scenario actors drawables.

4.1 Adding Entities

Adding entities to the scenario can be done either by selecting *New Entity* under the entities menu or by pressing the *Add* button on the bottom under the *Entities* tab in the Scenario Inspector. When doing so, a file dialog showing will appear displaying all entities installed on your system. For an entity to show up in this dialog it need to implement the MERLIN IMutableScenarioEntity interface (MERLIN IMissiles are however not displayed here). In the file dialog, decorated entities are also displayed.

4.2 Decorating Entities

Selectively adding functionality to entities is a key concept in MDA. The three different ways to add functionality to an entity are the *Actors*, *Drawables* and *Attachables* (see section 3.2). We call a simulation model that has one or several of these add-on's a decorated model/entity.

To decorate an entity select it in the Scenario Inspector to display the properties of the entity in the Entity Inspector. You can then browse the decorations already associated with the entity and their settings. To add a new actor/drawable/attachable use the *Entities* menu or the *Add* button at the bottom of the Entity Inspector.

You will most likely want to add signatures to your entities. If you don't, weapons, radars etc, will not be able to detect them. The signatures are added to the entity under the Attachables tab.

4.3 Create an Entity Prototype

As a user you often want a ready set of models to use. As described in section 4.2 you can decorate your entities with the graphics and sub system properties you want. To save a setting for an entity select it and choose *Save Entity* in the Entity menu. You can also right click the entity in the entity list (in the Scenario Inspector) and select *Save Entity* from the popup menu.

4.4 Editing a Prototype

To edit an existing prototype, add it to your scenario and modify it there. Then save it, by using *Save Entity* as described above, with the same name. Don't forget to remove the prototype when you are done editing it, unless you want it to be a part of the current scenario file.

5 Running a Scenario

In this section, how to run a scenario in MDA will be explained through the example of the simple air duel that is the default scenario.

The default scenario is preloaded with two aircrafts, one called *Carrier* and the other *Target*. Selecting the Carrier in the entity list allows the user to examine the entity properties in the Entity Inspector. In the *Info* tab in the Entity Inspector, it can be seen that the Carrier is located at an altitude of 1000 meters traveling at 100 m/s.

5.1 Preparing the Carrier

Before being able to launch missiles from the carrier, there are two components that needs to be prepared. The carrier needs an actor that launches missiles and a radar to detect and provide the missile with up-link data. Here, the functionality of these two components will be described.

5.1.1 Actor for Launching Missiles

Switching to the Actors tab in the Entity Inspector allows the user to see which actors are associated with the entity. In the default scenario the carrier has an actor called *Launch Missile*. This actor provides the carrier entity with the capability to fire a missile towards a target, see figure 5.1.

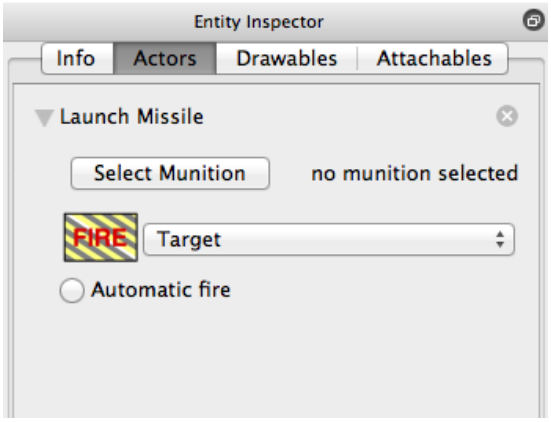


Figure 5.1: The Launch Missile actor.

At this point in time no munition model has been selected and firing is therefore not possible. Pressing the fire button will display an error message. To select a munition, press the *Select Munition* button. A dialog will appear listing all available munitions in your MERLIN installation, see figure 5.2. You should at least see three munitions called *xmo_ActiveRadarX1*, *xmo_SemiActiveRadarX1* and *xmo_IRMissileX1*. Selecting a munition will display the munition properties. Feel free to browse the available missiles before selection. For this example we will select the *xmo_ActiveRadarX1* munition, an active radar missile. After selection is completed, notice that the *xmo_ActiveRadarX1* name now is shown next to the *Select Munition* button, displaying your selection.

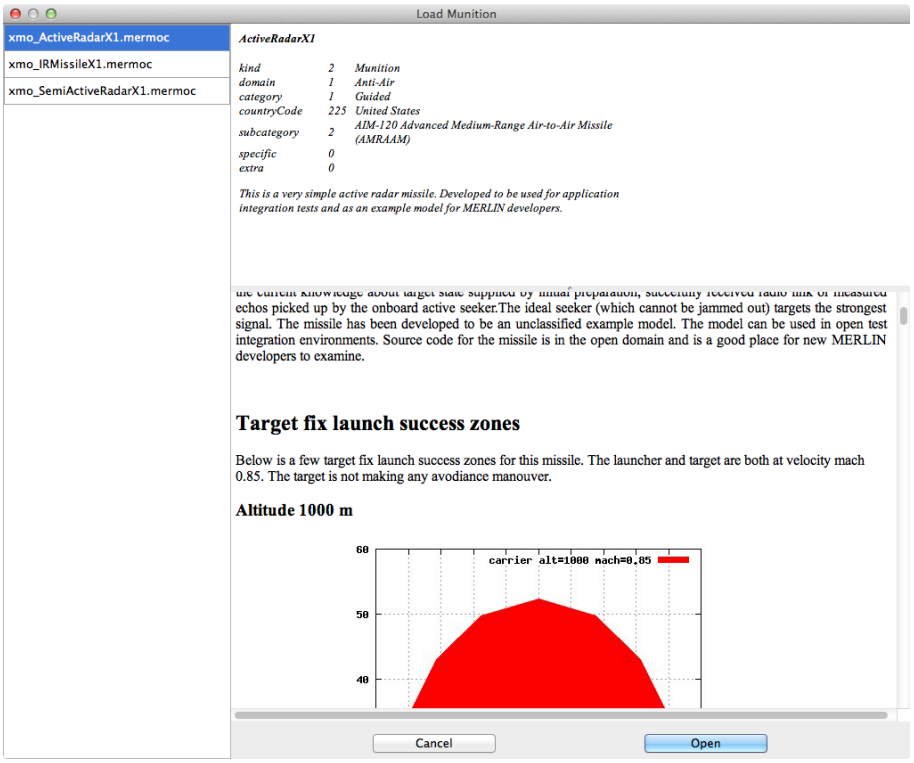


Figure 5.2: Dialog for selecting munition.

Next to the fire button, see figure 5.1, there is a drop-down menu where you can select which target you want to fire at. When pressing the popup menu both the *Target* and *Carrier* entity names are listed in the menu. The *Carrier* name is followed by a "- self" tag, informing you that this is your own entity. Make sure the *Target* entity is selected or the entity will try to fire against itself. In the 3D-view there will be a green box for each available target and a line from the carrier to the selected target.

As can be seen in figure 5.1 there is a selection for *Automatic fire*. If this is selected, a missile will automatically be launched when the simulation start. This is mostly intended for offline and batch simulations.

The Launch Missile actor also provides the capability of launching the missile with different engagement order numbers (EON) and in different launch modes. These depend on which type of missile is used.

5.1.2 Aircraft Radar

Shifting to the *Attachables* tab, an attachable called *merlin_RadarProxy* can be seen. This is a simple, ideal radar providing the capability to send link to the selected munition. In the 3D view, this radar (which is an EMEmitter) is displayed as a red transparent cone. The radar can be removed by pressing the cross icon at the far right of the *merlin_RadarProxy* component, see figure 5.3. In the 3D-view the red cone representing the emitting radar device has now disappeared. Switching back to the actors tab there will also be a warning message in the *Launch Missile* actor:

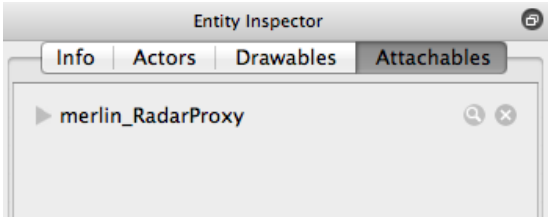


Figure 5.3: A radar attached to the aircraft. Removing Attachables is done by pressing the cross icon.

> **WARNING!** Entity doesn't have any radar and this missile needs link/illumination.

No target uplink will be sent to the missile if fired in this state. How the missile will respond to this depend on the specific missile implementation. The radar can be re-attached by pressing the *Add attachable* button in the *Attachables*. This brings up a list of all available attachable components. Select the *merlin_RadarProxy* and the red cone reappears in the 3D view and the warning message in the *Launch Missile* actor disappears.

5.2 Preparing the Target

To inspect the properties of the *Target* entity, select it in the scenario inspector *Entities* tab. When doing this you can immediately see a few differences by just looking at the target in the 3D view. A blue transparent sphere is drawn around the entity and if you look closely you can also see white transparent ellipsoids, see figure 5.4. The blue sphere and white ellipsoids are visualizations of the entities electromagnetic signature and IR-signatures, respectively.

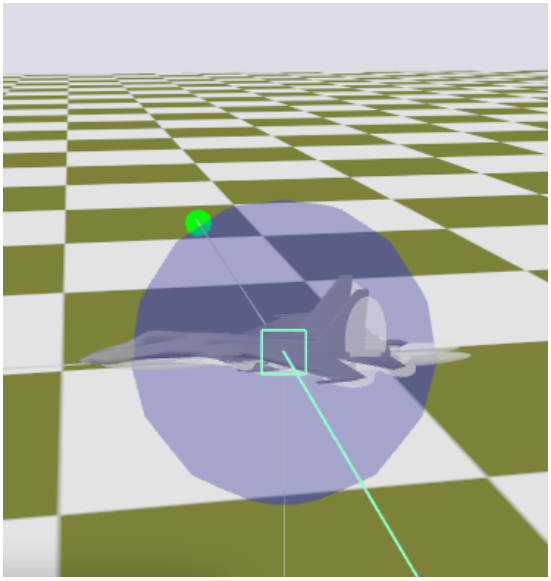


Figure 5.4: 3D view of the target where the RCS and IR-signatures are shown.

By selecting the *Attachables* tab, the two components `merlin_ConstantRCS-`

`Reflector` and `merlin_SimpleIRSignature` are shown. The signatures can be removed by pressing the cross icon in the *Attachables* tab or the *Add attachable* button. The 3D-visualization will disappear or appear as the signatures are removed or added. For a radar missile to acquire its target the target must have a radar cross section, and for an IR missile an IR-signature must be present.

It is possible to select if a graphics object, such as the SU37 3D-model in figure 5.4, should be displayed or not. This is done by toggling a property called *mIsVisible*, which is available in the entity *Drawables* tab when expanding e.g. the *SU37 3D-model* component.

5.3 Saving the Scenario

To save the scenario, select *Save* or *Save Scenario As...* from the **File** menu. The name of the saved scenario will appear as the title of the MDA main window.

5.4 Firing the Missile

Firing a missile is done by selecting the carrier entity and pressing the fire button in the *Launch Missile* component under the actors tab. A new entity is then added to the Scenario Inspectors *Entities list* which represents the missile. It is named something like `Carrier_ActiveRadarX1_id3`. The first substring, `Carrier`, informs us that it was the *Carrier* entity that created the missile, the second substring `ActiveRadarX1` is the name of the specific missile, and finally a unique id is provided.

The missile has now been created, but since the simulation has not yet started, the missile is located at the center of the Carrier in the 3D view.

5.5 Running the Simulation

To start the simulation, press the *Play* button at the top left MDA window. The time indicator in the bottom left corner of the MDA main window will start ticking and the entities moving. If the missile hits the target both the target and the missile will be removed from the scenario by default. The simulation can be paused by pressing the *Pause* button or the space bar key.

5.6 Stopping/Reloading a Scenario

Once you have launched the scenario you can easily return to the last saved state of the scenario by pressing the *Stop* button. The time indicator in the bottom of the left corner of the MDA main Window will reset to zero and the two entities will return to their starting position.

Be careful: Any changes done to the scenario since the last time it was saved are lost when the scenario is reloaded.

5.7 Inspecting the Missile

During the simulation you can select the missile entity and inspect its status during flight. Just as for the Carrier and Target, the *Info* tab shows the current position, velocity and orientation of the missile.

In the *Actors* tab an actor called *Missile Inspector* is available, providing information about the missile seeker states and a list of events related to the missile, see figure 5.5.

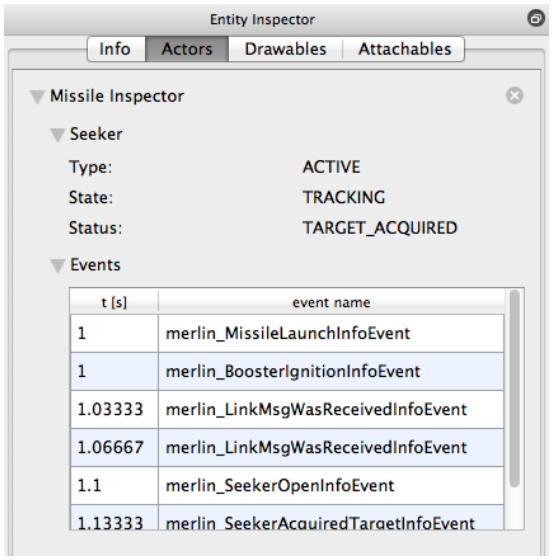


Figure 5.5: The *Missile Inspector* actor is automatically added to missiles when launched and provides information about the seeker state and missile related events.

5.8 End Conditions

In the default scenario no end conditions for the simulation is included. The simulation will continue forever. It is possible to add end conditions to the scenario. This is done by adding a specific actor to the scenario, which is done in the *Scenario Inspector* under the *Actors* tab.

Two generic end condition actors are provided with MDA. One called `act_MaxSimTimeEndCondition` and one called `act_GenericEventEndCondition`. The `act_MaxSimTimeEndCondition` actor allows the user to set a time when the simulation should end and the `act_GenericEventEndCondition` allows the user to set a MERLIN event and will stop the simulation when such an event is received. An example of such an event is the `merlin_MunitionDetonationTerminalEvent` which is sent by missiles when detonating.

When an end condition actor is triggered, the simulation is paused and the user can study the results of the simulation.

5.9 Default Scenario Actors

The default scenario has two scenario actors already setup when loaded. The *Remove entities when hit* actor which, as the name suggests, remove entities when they have been hit. A hit occurs when a `merlin_MunitionDetonationTerminalEvent` has been received. If this actor is included in the scenario, all entities within the blast range will then be removed.

In some models, missiles are launched without interacting with MDA and the Launch Missile actor. It is then impossible for MDA to obtain information about them unless the *Add External Missiles* actor is included in the scenario.

5.10 Extend Visualization

As has been shown before, it is possible to extend the 3D visualization of both the scenario and the entities. For studies of missile behaviors the scenario

drawable called `MissileInfo` is recommended. It provides a predicted impact point, current closing velocity, distance to target, time to target and seeker states directly into the 3D view, see figure 5.6.

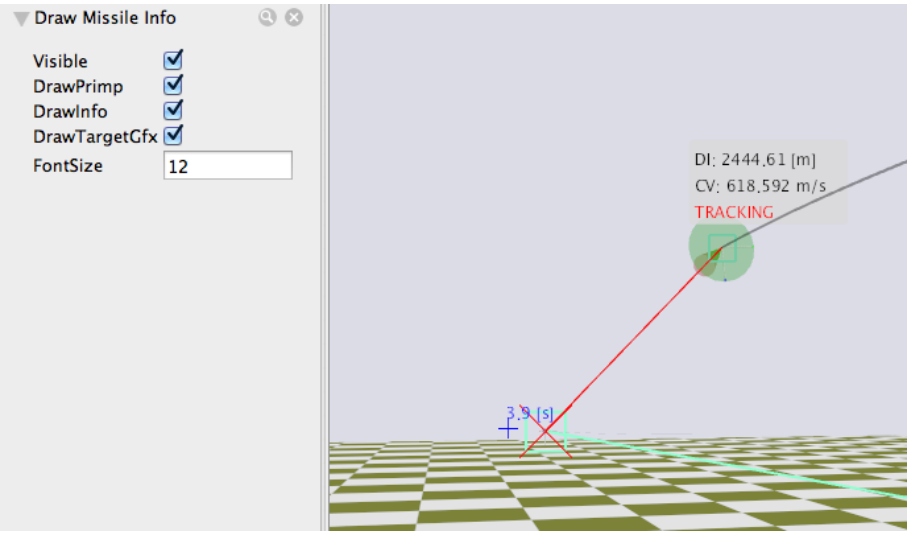


Figure 5.6: The Missile info drawable can display missile properties in the 3D view such as distance to target, closing velocity, seeker state etc.

6 Logging

The functionality of logging is provided in MDA through Actors and two such actors are included in the release of MDA 1.2. The first one is a high level log and works on all scenario entities. This is the Scenario Entity Logger which can be enabled by adding the scenario actor called `act_ScenarioEntityLogger` to the scenario, se section 3.1.2.

The second provided logging functionality is a detailed component logger associated with a specific entity or a specific type of entities. It is available as both a scenario actor (`act_ComponentLoggerAttacher`) and as an entity actor (`act_ComponentLogger`). When used as a scenario actor, the user must select which prototype that should be logged (e.g. `xmo_ActiveRadarX1.mermoc`).

6.1 Scenario Entity Logger

The Scenario Entity Logger provides the capability of logging kinematic data from all entities in the scenario. Here kinematic data refers to the positions, velocities, and orientations of the entities.

As soon as the `act_ScenarioEntityLogger` is added to the scenario it will start to log the available and any added entities.

6.1.1 Log structure

The Scenario Entity Logger creates a directory where MDA was started from, with the same name as the name of the scenario (e.g. DefaultScenario). In this directory, new directories for each of the entities are created (e.g. DefaultScenario/Target). In these directories log files are stored for each individual entity. The log files are all provided as ascii text files.

For each entity the Scenario Entity Logger will create three types of log files.

- entity.xml - which contain DIS type information about the entity.
- entity.log - which contain position, velocity and orientation information.
- event.log - which contains all event history related to the entity.

For the Default Scenario the log structure would look like this:

```
DefaultScenario/
                Carrier/
                        entity.xml
                        entity.log
                        event.log
                Target/
                        entity.xml
                        entity.log
                        event.log
```

6.1.2 File entity.xml

An example of entity.xml file for a missile called *Carrier_Active_id3* in the Default Scenario is displayed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<info>
    <name>Carrier_Active_id3</name>
    <type>ActiveRadarX1</type>
    <id>3</id>
    <dis>
        <type>2 1 1 225 2 0 0</type>
        <kind>Munition</kind>
        <domain>Anti-Air</domain>
        <category>Guided</category>
        <countryCode>United States</countryCode>
        <subcategory>
        AIM-120 Advanced Medium-Range Air-to-Air Missile (AMRAAM)
        </subcategory>
        <specific>NIL</specific>
        <extra>NIL</extra>
    </dis>
</info>
```

6.1.3 File entity.log

The entity.log file contains the kinematic state data in a blank space separate ascii file. The first line is a header, describing the individual columns. This line starting with a Matlab comment sign.

The logged data in the file has the following structure:

- Column 1 represents the simulation time in seconds.
- Column 2-4 represents the position (here in North, East and Down coordinates but could be ECEF x,y,z.) in units of meters.
- Column 5-8 represents the orientation in the form of a quaternion. The quaternion represents the transformation of the vector $\mathbf{x}$ expressed in body coordinates, B, to fixed frame coordinates, E, by the quaternion multiplication $\mathbf{x}^E = \mathbf{q} \circ \mathbf{x}^B$. Column 5-7 represents the vector part of the quaternion and column 8 the scalar part.
- Column 9-11 represent the velocity in body frame coordinates (Forward, Right and Down) in units of meters per second.
- Column 12-14 represents the angular velocity of the body expressed in body frame coordinates (commonly denoted as pqr) in units of radians per second.

Below is an example of an entity.log file:

```
% t[s] getDisplacement_0 getDisplacement_1 getDisplacement_2 ...
0.04 4 0 -1000 0 0 0 1 100 0 0 0 0 0
0.06 6 0 -1000 0 0 0 1 100 0 0 0 0 0
0.08 8 0 -1000 0 0 0 1 100 0 0 0 0 0
0.1 10 0 -1000 0 0 0 1 100 0 0 0 0 0
0.12 12 0 -1000 0 0 0 1 100 0 0 0 0 0
0.14 14 0 -1000 0 0 0 1 100 0 0 0 0 0
```

6.1.4 File event.log

The event.log file provides a list of the events that was emitted by this specific entity. The file consists of 5 columns. The first line is a header starting with a Matlab comment character.

- Column 1 represents the time in seconds.
- Column 2-4 represents the position of the entity at the time of the event. Position is given in fixed frame coordinates, here in the flat NED-frame but could be ECEF coordinates as well. Position is provided in units of meters.
- Column 5 represents the triggered event in a string format.
- Column 6 is optional and provides additional data related to the event.

Below is an example of an event.log for a missile.

```
% t[s]  entityPosE             EventName            additional data
0      0       0    -1000     merlin_MissileFiredInfoEvent \...
                                      launcher:Carrier target:Target  distance:5001
1    200       0    -1000     merlin_MissileLaunchInfoEvent
1    100       0    -1000     merlin_BoosterIgnitionInfoEvent
1.03 103.38    0     -999.995 merlin_LinkMsgWasReceivedInfoEvent
1.06 106.87    0     -999.979 merlin_LinkMsgWasReceivedInfoEvent
1.1  110.47    0     -999.952 merlin_SeekerOpenInfoEvent
1.13 114.18    0     -999.914 merlin_SeekerAcquiredTargetInfoEvent
2.06 261.86   -2.32  -996.207 merlin_LinkMsgWasReceivedInfoEvent
3.06 507.36  -35.65 -1013.84  merlin_LinkMsgWasReceivedInfoEvent
4.06 847.64  -104.49 -1049.1  merlin_LinkMsgWasReceivedInfoEvent
5.06 1289.4  -197.74 -1088.04 merlin_LinkMsgWasReceivedInfoEvent
6.06 1837.5  -298.6  -1118.29 merlin_LinkMsgWasReceivedInfoEvent
7.06 2493.72 -388.53 -1130.77 merlin_LinkMsgWasReceivedInfoEvent
8    3202.17 -448.83 -1122.53 merlin_BoosterBurnOutInfoEvent
8.06 3255.93 -452.01 -1121.2  merlin_LinkMsgWasReceivedInfoEvent
9.06 4041.33 -480.00 -1094.05 merlin_LinkMsgWasReceivedInfoEvent
10.0 4789.18 -476.19 -1058.97 merlin_LinkMsgWasReceivedInfoEvent
11.0 5501.72 -443.50 -1019.48 merlin_LinkMsgWasReceivedInfoEvent
11.5 5816.91 -418.93 -1000    merlin_MunitionDetonationTerminalEvent \...
                                          killed: Target missDistance=0.00624176
```

6.2 Component Logger

The Component Logger allows you to log detailed information about entities and its subcomponents. It uses the interface loggers that are build in to MERLIN and automatically generated by the MERLIN build system. Thus, all MERLIN models/components you have installed on your system will have an appropriate logger.

The Component Logger comes in two flavors, one that is added directly to an entity and one that is added to the scenario but acts on a selected class of entities.

6.2.1 Manually adding a Component Logger

If you have an existing entity in the scenario, an aircraft for example, you can add a *ComponentLogger* to this entity. This is done in the Entity Inspector, Actors tab.

The Component Logger provides you with a list of all subcomponents in the entity and the capability to select which methods/data within those subcomponents you want to log, see figure 6.1. Some of the subcomponents make little sense to log but are included since they are automatically generated. [1].

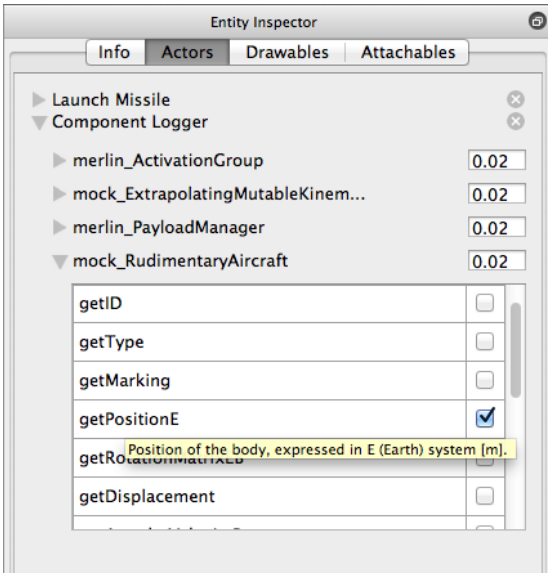


Figure 6.1: An MDA ComponentLogger entity actor is added to an entity and the model sub component, *mock_RudimentaryAircraft*, method *getPositionE* is selected for logging.

By hovering over each method with the mouse pointer you get a tooltip describing the method. Logging is enabled by clicking the checkbox next to the method name. You can also specify how often you want the logger to query the methods for each component. The default sampling time for logging is set to 0.02 seconds, i.e. 50 Hz.

If you save the scenario all logging settings are saved and therefore persistent the next time you open the scenario.

Currently the Component Logger functionally is fairly complex and unintuitive for non MERLIN-developers. It is however a powerful method to probe into the behavior of subcomponents of complex models and is therefore provided with MDA and described in this document.

6.2.2 Automatically creating Component Loggers

In many cases models are created during the execution of the scenario. One example of this is missiles. Many missiles can be fired during a scenario and adding loggers manually to the missile entities becomes a lot of work.

[1] The GUI display all components inside the model in the order they appear in the mermoc file). If you open up the components you find a list of all the available methods for that particular component.

To solve this problem a scenario Actor called `act_ComponentLoggerAttacher` is provided with MDA. This scenario Actor lets you specify a specific model filename, see figure 6.2, (e.g. `xmo_ActiveRadarX1.mermoc`) and provides the user with a similar GUI as for the *ComponentLogger* where subcomponents can be selected for logging. As soon as an entity is created that matches the model filename, a *ComponentLogger* is automatically added to that entity with the same logging settings as specified in the scenario actor template, see figure 6.3.

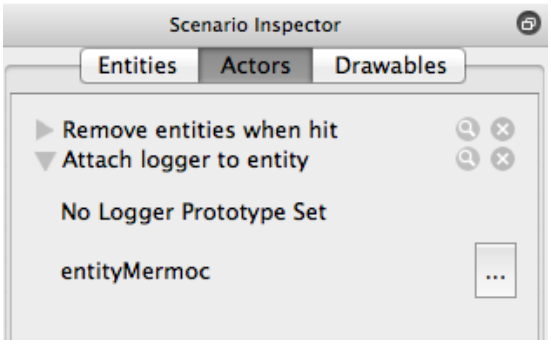
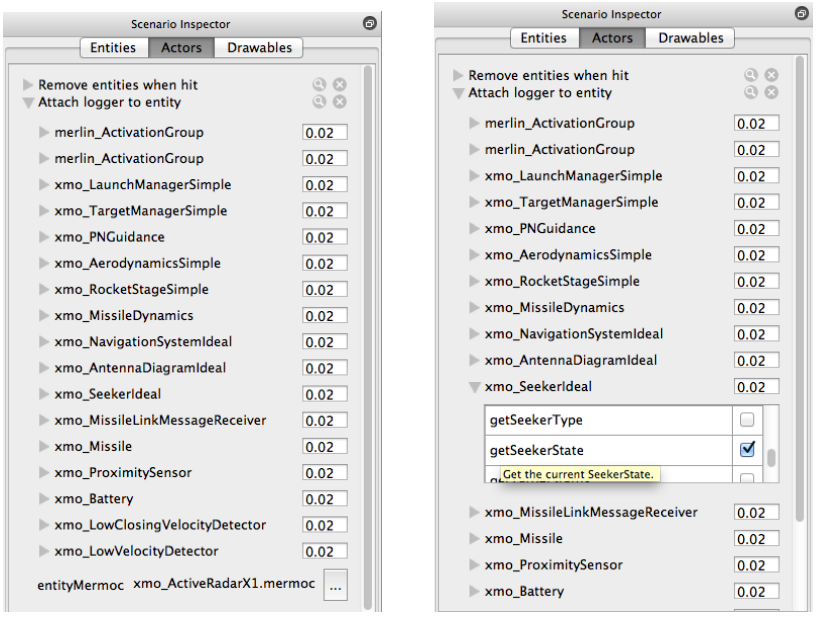


Figure 6.2: An MDA ComponentLoggerAttacher scenario actor just added to the scenario but still not configured.



(a) ComponentLoggerAttacher with xmo_ActiveRadarX1 selected.

(b) ComponentLoggerAttacher with xmo_ActiveRadarX1 and SeekerIdeal getSeekerState method selected.

Figure 6.3: ComponentLoggetAttacher with logging prototype set.

6.2.3 Log Files

The Component Logger stores the data into an ordered folder structure. The root folder for the log output has the same name as the scenario and is placed

in MDA's current working directory, typically the same directory as where you launched MDA.

Inside the root log folder you will find a directory for each logged entity. Those folders are named after the entities name in the scenario. In each entity log folder you find a log file for each logged entity component. The log files are named after the name of the components, followed by their order in the mermoc file (closing document order, which is the same order as you see in the logger GUI). The numbering starts with #1.

The path to the log for the Target entity in the default scenario could look like this *DefaultScenario/Target/mock_RudimentaryAircraft#5.log*. I.e., the filename indicates that we have logged properties from the model component mock_RudimentaryAircraft, which is the fifth component in the scenario entity with name *Target* in the scenario named *DefaultScenario*. A complete logging folder might look like figure 6.4.

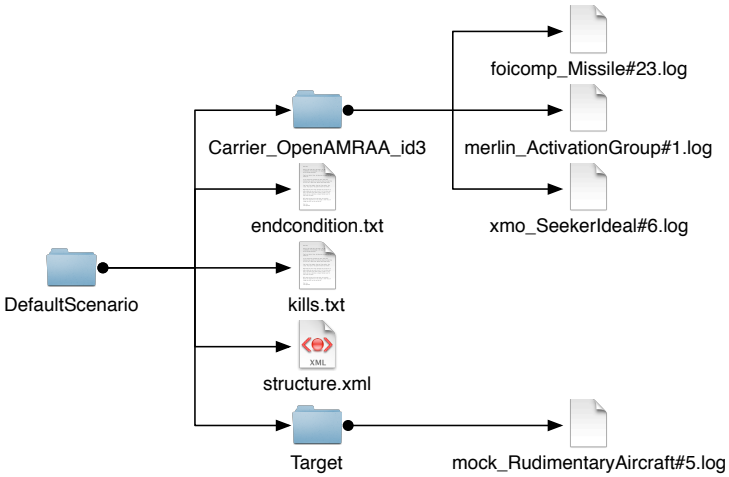


Figure 6.4: An example of the folder structure created by the loggers

The file *structure.xml* contains an xml description of the logging files. The component log files are ascii files with columns separated by spaces. The first line is a header describing the individual columns. The name of logged methods appear in the header for their respective column. Methods that return vectors or quaternions append a number at the end of the method name to indicate which element of the vector the column represents.

Below is an example of the method getPosition() being logged.

```
% t[s] getPositionE_0 getPositionE_1 getPositionE_2
0.02     2 0 -1000
0.04     4 0 -1000
0.06     6 0 -1000
0.08     8 0 -1000
0.1      10 0 -1000
```

7 Parameter Variation

When studying a model or a scenario it is often useful to be able to run the scenario in many different variations and to analyze the results to achieve a greater understanding of the studied problem.

MDA has a built-in engine for parameter variation that can be used together with the logging functionality described in chapter 6 to study variations in complex scenarios.

In this chapter the parameter variation functionality will be described through an example based on the Default Scenario delivered along with MDA.

7.1 Overview

How to Use the built-in parameter variation engine in MDA can be described in a series of steps:

Step 1 Design you scenario in MDA by setting up and configuring the models you want to include. Make sure the scenario can run without any user interactions. For example make sure to use features such as the *Launch Missile*'s automatic fire (see section 5).

Step 2 Setup *End Conditions* that will terminate the simulation. You can use the built in variants or create your own solutions, see section 5.8.

Step 3 Add loggers for the data you want to extract and evaluate, see chapter 6.

Step 4 Save the edited scenario. The variation engine uses the saved scenario file to run the scenario with different settings. If the scenario is not saved the engine will not run the scenario with correct setup.

Step 5. Select which parameters that should be varied and how they should vary. This is the parameter dialog that appears if you select *Parameter variation...* in the *Simulation* menu.

Step 6. Run the parameter variation.

Step 7. Analyze the results.

In version 1.2 of MDA it is only possible to vary parameters that are single floating point values, like mass, time, etc. This means that it is not possible to vary parameters like vectors.

It is possible to vary parameters of sub models that appears as a link to another mermoc files. In the example in section 7.2 this will be demonstrated on a missile.

7.2 Example

In this section the steps of using parameter variation will be explained through a concrete example, using the Default Scenario provided with MDA.

7.2.1 Step 1. Design the Scenario

The first step to running parameter variation is to create/design your scenario.

- Load the default scenario.
- Select the *xmo_ActivateRadarX1* missile in the *Launch Missile* entity actor for the *Carrier*.
- Make sure to check the *Automatic fire* checkbox. This check box will make the launch missile actor fire a missile as soon as the scenario is started.
- Save the scenario under some suitable name, for instance *ParameterVariationTest*.
- Test running the scenario and make sure you hit your target.
- Press the *stop* button to reload the saved version of the *ParameterVariationTest* scenario.

7.2.2 Step 2. Setup End Conditions

Next, some break condition need to be added to make sure the scenario end when some criterion is reached. In this example we will use two built in end condition scenario actors. *MaxSimTimeEndCondition* and *GenericEventEndCondition*. The *MaxSimTimeEndCondition* stops the scenario after a time limit has been reached.

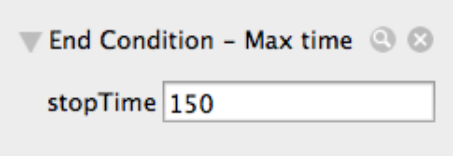


Figure 7.1: The scenario end condition Max Time set to 150 seconds.

This end condition is useful for catching the cases when no other end condition has occurred. It can be used as a safe guard if no other end condition are triggered and thus avoid the simulation to continue forever.

- Activate the *Actors* tab in the Scenario Inspector and press the Add actor button. Add the *MaxSimTimeEndCondition* actor.
- Set the stopTime to 100 seconds, which is enough for this scenario.
- Save the scenario and run it. You can speed up the simulation time by adjusting the time multiplier slider in the top toolbar. Notice how the scenario is paused after 100 seconds have passed. Press the stop button to reload the scenario.

Now, add an end condition for when a missile detonates.

- Activate the *Actors* tab in the Scenario Inspector and press the Add actor button. Add the *GenericEventEndCondition* actor, see figure 7.2.

- Select the merlin_MunitionDetonationTerminalEvent. To see the whole GUI for this actor you might have to widen the Scenario Inspector tab. Make sure to widen it until you see the full width of the actor.
- Save the scenario.
- Run the scenario to make sure it pauses when a missile hits the target.
- Reload the scenario by pressing the stop button.

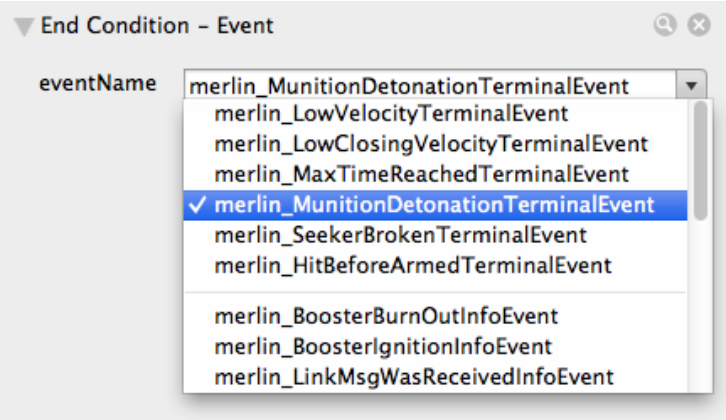


Figure 7.2: A GenericEventCondition with the default merlin_MunitionDetonation event chosen.

7.2.3 Step 3. Add Loggers

Adding loggers to a scenario is described in details in chapter 6.

- Add the *ScenarioEntityLogger* by pressing the Add actor button in the Actors tab of the Scenario Inspector and selecting it.

7.2.4 Step 4. Save Scenario

As a precaution, save the scenario again, before configuring the parameter variation.

7.2.5 Step 5. Configure Parameter Variation

Configuring the parameter variation is done by selecting which parameters in the scenario file that should be varied, between which values and with what kind of distribution. The scenario is presented to the user as a raw object graph containing dependencies between objects (entities and sub components) and the parameters. Appendix B describes the file format of the scenario in more detail and could be useful for better understanding the object graph shown in the parameter variation dialog.

In this example the mass of a missile will be varied.

- Open the *Parameter Variation* dialog by selecting the *Parameter Variation* field under the *Simulation* menu.
- The displayed variation dialog that opens up should look somewhat similar to figure 7.3.

- Locate the mMissileMermoc parameter under mda_Scenario>mEntityList >mda_ScenarioEntity>mEntity>mActorList>act_LaunchMissile.
- Double click the mMissileMermoc to open a second object graph window. This window then displays the object graph of the missile model.
- Locate the parameter called EmptyMass available under xmo_Missile >mDynamics>EmptyMass, see figure 7.4a.
- Double click EmptyMass to display a dialog for selecting the type of variation, see figure 7.4b
- Select start value 80, end value 120 and number of values 10. Use the constant step variation type and press OK.
- Close the parameter variation dialog for the missile by pressing Done.

By default the variation type is set to *Constant step*. The *Constant step* will accept a parameter range, defined by a start value, an end value and the number of steps the range should be partitioned into. The pop up menu let you select other variations as well such as *uniform distribution* and *normal distribution*, see figure 7.4b.

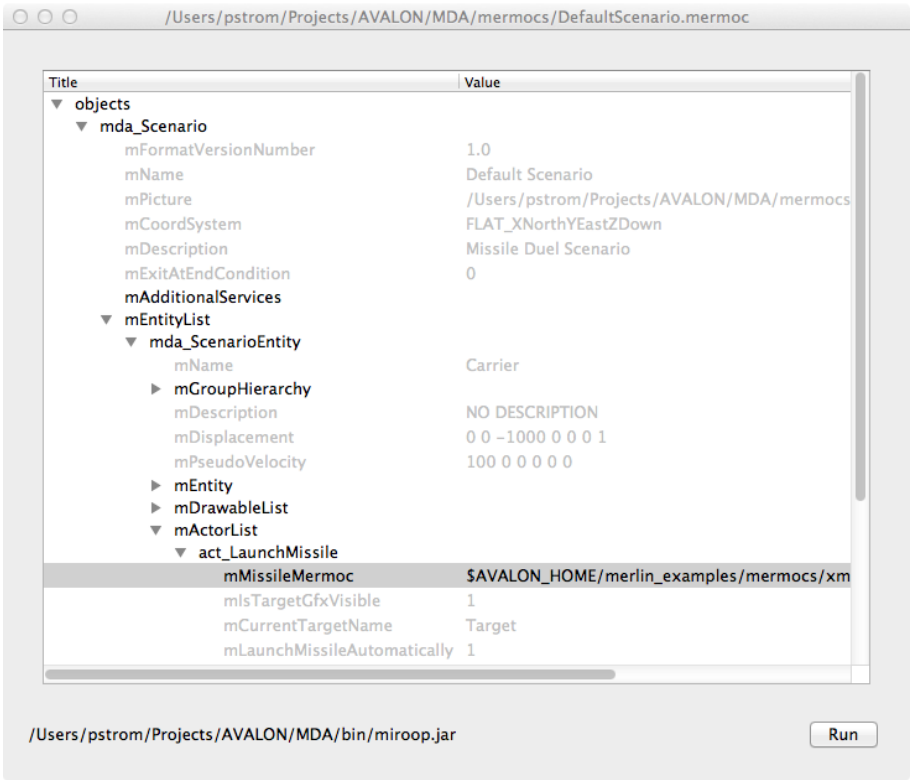
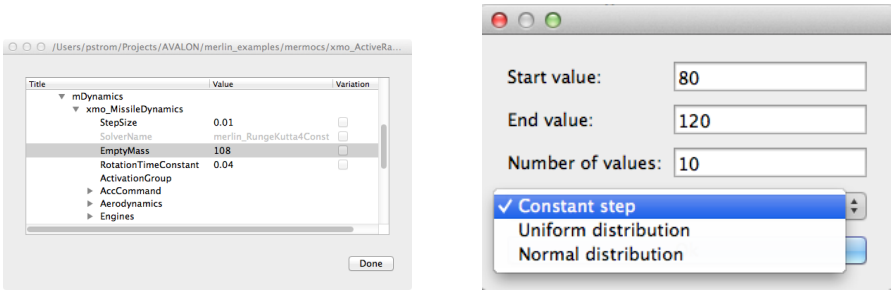


Figure 7.3: In the parameter variation dialog all scenario properties are shown. Only the non greyed properties can be configured for variation at this time.



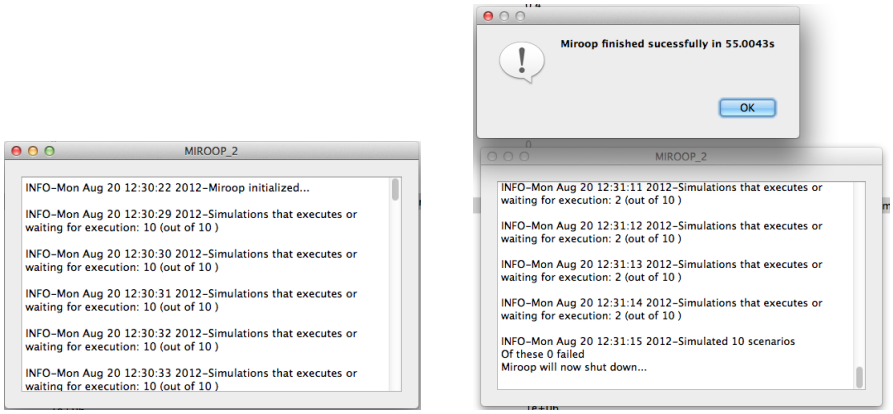
(a) The EmptyMass parameter that should be varied.

(b) Different distributions are available for the parameter variation.

Figure 7.4: Dialog for varying the parameter *EmptyMass* with a constant step between 80 kg and 120 kg.

7.2.6 Step 6. Running the Parameter Variation.

To start the parameter variation, press the *Run* button in the Scenario Variation dialog box. When you do this, MDA displays a status dialog showing the progress, see figure 7.5. When the variation is finished close all dialogs.



(a) Progress dialog initiating.

(b) Run finished.

Figure 7.5: Progress of the parameter variation.

7.2.7 Step 7. Analyze Results

In your current working directory, there will now be a directory called *MIROOP_0*. If you have run multiple variation executions you may have several MIROOP directories with an increasing number suffix. Inside the *MIROOP_0* directory you will find ten (the number of variations we ran) Sim_X directories, each containing data for the variation runs. There are also two files describing the setup of the variation. These are called miroop-run.txt and miroop.xml (see figure 7.6).

Inside the Sim directories there is a copy of the original scenario file, *ParameterVariationTest.mermoc*. This file contains the scenario mermoc for this specific variation. In this example they will look the same for all variations, since the parameter that was varied was in another mermoc file. Instead a folder called *LocalAvalon* has been created that contains the missile mermoc with the variation settings.

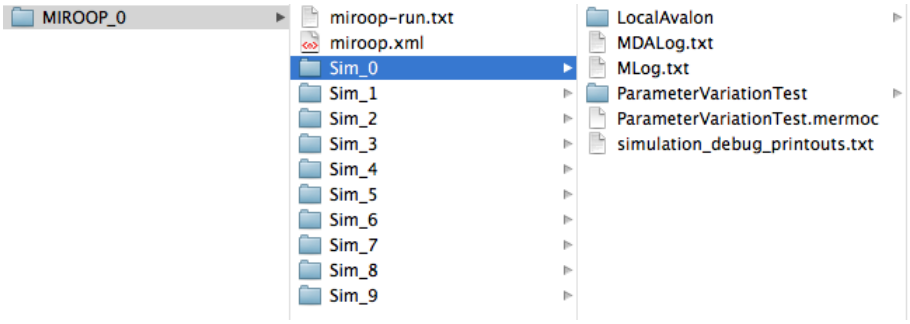


Figure 7.6: During a parameter variation a directory called MIROOP_0 is created containing sub folders with results from each different parameter variation.

The loggers have also created a folder with the scenario name, *ParameterVariationTest*, containing all of the scenario logs. In this folder the results will be located. Read chapter 6 for descriptions of the scenario output folder and its contents.

Finally there are two MERLIN and MDA log output files *MLog.txt* and *MDALog.txt*. These files are mostly used for debugging.

7.2.7.1 Quick Visualization of the Results

One quick way to visualize the outcome of all variation runs is to open the automatically generated scenario called *ResultScenario.mermoc*. It will play back all entities from each scenario run simultaneously. Note that this file can become very large depending on how many variations you have performed.

It is suggested to use this feature only for small numbers of variations. It can be useful when defining the scenario, to quickly find out the effect of the chosen variation.

Note also the scenario actor *ScenarioEntityLogger*, must be present in the scenario for the *ResultScenario* to be created. Output from the parameter variation example described in this chapter can be seen in figure 7.7. Notice how the missile trajectories differ due to the changed empty mass. If you run the scenario you will also find that the time of impact also differs.

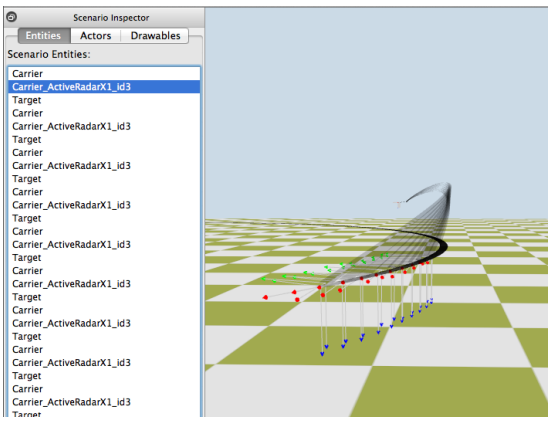


Figure 7.7: An EmptyMass variation visualized by running the ResultScenario.

8 Overview of MDA Extensions

MDA is built using a plugin architecture and allows you to configure your simulations in endless ways without enforcing all users to suffer from unused components in their respective GUI.

This document only describe the available components that is part of the MDA core package. Several extensions has been made for MDA (and simulation models) that are not part of the MDA core package. This section briefly covers a few of those to make you aware of their existence.

8.1 Joystick support

The FOI aircraft software includes an MDA extension for connecting USB joysticks to MDA. This extension (it's an entity actor) will work with any joystick, but has extra support for ThrustMasters® A10 Wartog HOTAS™ hardware, including both the thrust and stick.

8.2 HLA networking

MDA has an HLA 1516 evolved connection with the FOM from the P2SN 2.0 [1] agreement. At this time this solution is constructed as a separate program package. In future releases of MDA HLA support will be included in the standard installation.

8.3 Computer Generated Forces - CGF

FOI has built a set of pilot behaviors which can be used within MDA. There are also some MDA extensions to let the user configure and analyze the behaviors in detail.

8.4 Missile simulation models

MDA support all MERLIN models. In the standard deployment of MDA v1.2 only the MERLIN example missiles are included.

[1] The P2SN Federation Agreements and FOM (P2SN FOM) is a reference document intended to be used as a base-line when developing P2SN based federations. P2SN stands for Persistent Partner Simulation Network

A MDA Components

The components included in the standard release of MDA are listed in this appendix with a short description of their functionality. A more complete documentation of the components are built into MDA.

Entity Actors

Name	Description
act_ComponentLogger	Add logging facilities to an entity
act_LaunchMissile	Add functionality to equip an entity with a missile, configure it and and fire against targets
act_LogReader	You can replay a stored trajectory using this component
act_RangeDisplay	Displays a table with current range to all other entities in the scenario

Entity Drawables

Name	Description
drawable_ObjModel	Renders Wavefront obj 3D objects
drawable_Acceleration	Render an axis displaying the current load factor of the entity
drawable_AlphaBeta	Visualizes alpha and beta of the entity
drawable_CoordinateSystem	Render a body fix coordinate system
drawable_NEDCoordinateSystem	Render a North, East, Down inertial system that follows with the entity
drawable_EMEmitter	Render an EMEmitter if there is an EMEmitter model associated with the entity
drawable_Path	Creates a path trailing the entity
drawable_PathTube	Creates a circular path trailing the entity

Entity Attachables

Name	Description
merlin_ConstantRCSReflector	Adds a spheric constant size electromagnetic reflector to the entity
merlin_RadarProxy	Add an ideal radar to the entity
merlin_F16IRSignature	Add a volumetric Infrared Red F16 signature to the entity
merlin_SU27IRSignature	Add a volumetric Infrared Red SU27 signature to the entity
merlin_JAS39IRSignature	Add a volumetric Infrared Red JAS39 signature to the entity
merlin_JA37IRSignature	Add a volumetric Infrared Red JA37 signature to the entity

Scenario Actors

Name	Description
act_AutoAddExternalMissiles	An actor that monitor all created missiles and attaches them to the scenario without user intervention
act_ComponentLoggerAttacher	A component that can be configured to attach a component logger to each entity of a specified type
act_ScenarioEntityLogger	This component log kinematic data for all entities in the scenario. Output can be read back with the act_LogReader
act_RemoveDetonatedEntities	Removes all entities that are within range of any MunitionDetonation terminal event
act_GenericEventEndCondition	An end condition that monitor MERLIN events
act_MaxSimTimeEndCondition	An end condition that can be configured with a max simulation time

Scenario Drawables

Name	Description
drawable_Detonation	Render a simplistic detonation at MunitionDetonation events
drawable_EMEmitters	Render all EMEmitters registered in EMChannel
drawable_EMReceivers	Render all EMReceivers registered in EMChannel
drawable_EMReflectors	Render all EMReflectors registered in EMChannel
drawable_GridTerrain	Render a configurable infinite grid
drawable_IREmitters	Render all IREMitters registered in IRChannel
drawable_IRSensors	Render all IRSensors registered in IRChannel
drawable_LinkGfx	Display animated graphics for link sent and received events
MissileInfo	Display a selectable set of missile related info, such as closing velocity, distance to target, predicted point of impact etc.
drawable_Plane	Draws a gridded plane using lines
drawable_TexturedPlane	Draws a texture mapped plane
DrawClockTime	Draw time manager clock time

B Scenario File Format

The scenario file format in MDA is based on the MERLIN mermoc standard. A mermoc is an XML file that describes the relation between objects and the values of the parameters in the object.

In this chapter the MDA scenario file format will be explained by going thought the DefaultScenario.mermoc file and provide the XML Schema for the scenario file format.

Example

Below is a truncated listing of the scenario file called DefaultScenario.mermoc.

As can be seen in the listing, the scenario is constructed from five major parts:

- Scenario Information, containing information about the scenario and a flag if the program should exit when an end condition is reached.
- A list of additional services. MERLIN is provided with a basic set of services such as gravity models, atmosphere models and time managers for numerical integration of differential equation. Here it is possible to extend the basic set with extra services, required by your models.
- A list of entities in the scenario. In the default scenario there are two entities, the Carrier and the Target. The entities them selfs are also described with XML. Here the content of their XML description was replaced with dots to make the illustration of the scenario easier to read.
- A list of drawables in the scenario. These are also described with XML but truncated here for improved readability.
- A list of actors in the scenario. Two actors are available in the DefaultScenario, Automatically removal of detonated entities and automatic insertion of missiles created from other entities (and not directly from MDA).

```
<?xml version='1.0'?>
<objects>
  <mda_Scenario>
    <!-- Scenario Information -->
    <mFormatVersionNumber value="1.0" />
    <mName value="Default Scenario" />
    <mPicture value="$AVALON_HOME/MDA/mermocs/DefaultScenario.png" />
    <mCoordSystem value="FLAT_XNorthYEastZDown" />
    <mDescription value="Missile Duel Scenario" />
    <mExitAtEndCondition value="0" />

    <!-- List of additional services --->
    <mAdditionalServices list="objectPtr">
    </mAdditionalServices>

    <!-- List of entities in the scenario --->
    <mEntityList list="objectPtr">
```

```
        <mda_ScenarioEntity>...</mda_ScenarioEntity> <!--Carrier-->
        <mda_ScenarioEntity>...</mda_ScenarioEntity> <!--Target-->
    </mEntityList>

    <!-- List of Scenario Drawables --->
    <mScenarioDrawableList list="objectPtr">
        <drawable_GridTerrain>...</drawable_GridTerrain>
        <drawable_EMEmitters>...</drawable_EMEmitters>
        <drawable_EMReceivers>... </drawable_EMReceivers>
        <drawable_EMReflectors>...</drawable_EMReflectors>
        <drawable_IRSensors>... </drawable_IRSensors>
        <drawable_IREmitters>...</drawable_IREmitters>
    </mScenarioDrawableList>

    <!-- List of Scenario Actors --->
    <mScenarioActorList list="objectPtr">
        <act_RemoveDetonatedEntities/>
        <act_AutoAddExternalMissiles/>
    </mScenarioActorList>

  </mda_Scenario>
</objects>
```

XML Schema

Below is a listing of the XML Schema that describes the scenario file format.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="mda_Scenario" substitutionGroup="complexObject">
  <xsd:annotation>
   <xsd:appinfo>
    <implements>
     <mda_IScenario/>
    <mda_IComponent/>
   </implements>
  </xsd:appinfo>
  <xsd:documentation xml:lang="en">
  This is representation of a scenario.
  </xsd:documentation>
  </xsd:annotation>

 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="complexObjectBase">
    <xsd:all>

    <xsd:element name="mDescription" type="std_stringParamMember">
     <xsd:annotation>
      <xsd:documentation xml:lang="en">
      Description of the scenario
      </xsd:documentation>
     </xsd:annotation>
```

```
 </xsd:element>

 <xsd:element name="mName" type="std_stringParamMember">
  <xsd:annotation>
   <xsd:documentation xml:lang="en">
    Short name of the scenario
   </xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="mPicture" type="merlin_FilePathParamMember" minOccurs="0">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">
  Optional path to a picture that describe the scenario.
  </xsd:documentation>
 </xsd:annotation>
</xsd:element>

<xsd:element name="mCoordSystem" type="std_stringParamMember" minOccurs="0">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">
  The coordinate system used by the scenario.
  merlin::ITerrain::CoordinateSystemName.
  </xsd:documentation>
 </xsd:annotation>
</xsd:element>

<xsd:element name="mEntityList" type="ObjectPtrListMember">
 <xsd:annotation>
  <xsd:appinfo>
   <withInterface name="mda_IScenarioEntity"/>
  </xsd:appinfo>
 </xsd:annotation>
</xsd:element>

<xsd:element name="mScenarioDrawableList" type="ObjectPtrListMember" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <withInterface name="gfx_IScenarioDrawable"/>
  </xsd:appinfo>
 </xsd:annotation>
</xsd:element>

<xsd:element name="mScenarioActorList" type="ObjectPtrListMember" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <withInterface name="mda_IScenarioActor"/>
  </xsd:appinfo>
 </xsd:annotation>
</xsd:element>

<xsd:element name="mAdditionalServices" type="ObjectPtrListMember" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
```

```
    <withInterface name="merlin_IServiceControl"/>
    </xsd:appinfo>
   </xsd:annotation>
  </xsd:element>

  <xsd:element name="mFormatVersionNumber" type="std_stringParamMember">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">
     Format version number for future compatibility issues.
    </xsd:documentation>
   </xsd:annotation>
  </xsd:element>

  <xsd:element name="mExitAtEndCondition"
               type="merlin_BoolParamMember"
               minOccurs="0">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">
    Setting if the scenario should force an application
    exit or pause (or just output end condition data).\n
    Defaults to false. Ie dont exit at endcondition.
    </xsd:documentation>
   </xsd:annotation>
  </xsd:element>

  </xsd:all>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

</xsd:element>
</xsd:schema>
```