AADL Inspector 1.3

User Manual

Ellidiss Technologies
http://www.ellidiss.fr
aadl@ellidiss.fr

Contents

1 Introduction

AADL Inspector is an analysis framework for source texts complying with the standard definition of the Architecture Analysis and Design Language (**AADL**). The formal definition of the **AADL** language can be found in the **SAE AS-5506B** document and general information about this language is available on the **AADL** committee web site: www.aadl.info.

AADL Inspector is packaged in a light weight standalone distribution that minimizes installation and maintenance effort in order to ease the every day use of the product on standard personal computers or network servers. The product is available for both **Windows** and **Linux** platforms.

The goal of **AADL Inspector** is to encompass a variety of tools to process a complete **AADL** specification composed of a set of text files. These files can be created within **AADL Inspector** itself, loaded from pre-existing local or remote libraries or automatically generated by an import wizard. **AADL** files can also be organized into projects to facilitate the management of large models. The processing tools can be used to analyse various facets of the architecture or to offer code and documentation capabilities. These processing tools are organized in a modular and extendable way so that additional tools can easily be included.

The standard installation of **AADL Inspector** 1.3 includes the following processing tools:

- **AADL** syntactic analysis
- instance model overview and declarative model statistics (*Metrics*)
- architectural (static) semantics (*Naming*, *Legality*, *Consistency* and *ARINC653*)
- real-time (dynamic) semantics (**Cheddar** and **Marzhin**)

It is also possible to modify the current **AADL** specification either with a direct edition of the opened source text files, or by using a set of wizards that enable advanced functions such as:

- select the root of the system instance hierarchy
- set the thread priorities according to the given scheduling protocol (RM, DM)
- bind threads to available processors with allocation algorithms

- modify the main thread real-time properties in a spreadsheet
- apply uniform text formatting rules (autoformat)
- convert older versions of source text into **AADL** v2.1
- import **OMG UML MARTE** models into **AADL**

The current version of **AADL Inspector** covers the following standard definitions:

- **AADL** Core (v1.0, v2.0 and v2.1)
- **AADL** Behaviour Annex (v1.0 and errata)
- **AADL** Error Model Annex (v1.0 and 2.0 draft 0.97)
- **AADL** Data Model Annex
- **AADL ARINC 653** Annex

2 Before starting

2.1 Installation

Installation of the product only requires the following easy actions:

- get a copy of the installation package for the desired platform (**Windows** or **Linux**) from the **Ellidiss** download site: http://www.ellidiss.com/downloads.asp

- run the installation program on **Windows** or uncompress and expand the archive file on **Linux**.

- launch the `AADLInspector` executable file located in the appropriate `bin` subdirectory of the installation directory, or the corresponding desktop shortcut on **Windows**.

Downloaded packages usually come with a temporary trial license that can be used free of charge. If you purchased the product or this temporary license has expired, please contact **Ellidiss** customer support service to get the appropriate license information and installation procedure that fits your situation. A standard installation requires less than 40 Mbytes of free disk space.

Currently, only the 32bits versions of the executable files are provided. Before running the product on 64bits platforms, please check that the 32bits compatibility package is installed (i.e. `ia32-libs` on **Linux**)

2.2 Distribution contents

After having properly been installed on the computer, the **AADL Inspector** installation directory contains the following subdirectories:

- `bin.w32` or `bin.l32` subdirectory
- `bin.common` subdirectory
- `config` subdirectory
- `examples` subdirectory
- `doc` subdirectory

`bin.w32` contains platform dependent executable files for **Windows**, whereas `bin.l32` contains the corresponding files for **Linux** platforms.

`bin.common` contains `.jar` files that are common to all platforms.

In addition, after a first launch of the tool, a directory is created to store temporary files. The actual location of this temporary directory can be customized by the `logDirectory` parameter in the `config/AIConfig.ini` file, or the `-l` command line option. Default location of the temporary directory is within the user's home directory.

2.2.1. Bin subdirectories

These directories contain the executable files for the current platform or **Java** archive files that are shared by all platforms. The only external requirement is the availability of a proper **Java** Run-time Environment (**JRE**) that is only needed to run the simulator. These files are:

- `AADLInspector` main executable file
- `AIMonitor` remote process monitoring executable file
- `aadlrev` executable file (**AADL** syntactic analyser)
- `xmlrev` executable file (**XML** syntactic analyser)
- `sbprolog` executable file (**Prolog** engine)
- `cheddarkernel` executable file (**Cheddar** schedulability analyser)
- `Marzhin`, `VAgent` and `VCore` **Java** archive files for the **Marzhin** simulator

`aadlrev 2.5` is a standalone **AADL** syntactic analyser that is used by the **LMP** (Logic Model Process) technology to convert **AADL** specifications into list of **Prolog** predicates. This utility tool can analyse textual **AADL** files that comply with **AADL** 2.1 (*SAE AS-5506B*) , the **AADL** Error Model v1 (*SAE AS-5506/1 Annex E*), the **AADL** Behaviour Annex (*SAE AS-5506/2 Annex D*), and the **AADL** ARINC 653 Annex (*SAE AS-5506/2 Annex F*). In addition, the draft 0.97 of the new **AADL** Error Model v2 (future *SAE AS-5506/3 Annex E*) is also supported by `aadlrev`. Most of the **AADL** 1.0 (*SAE AS-5506*) and 2.0 (*SAE AS-5506A*) syntax is also recognized and can be automatically converted into the newest 2.1 format. More information about **AADL** can be found at: http://www.aadl.info

`xmlrev 1.0` is a standalone **XML** syntactic analyser that is used by the **LMP** (Logic Model Process) technology to convert **XML** or **XMI** serialized models into list of **Prolog** predicates.

`cheddarkernel 3.0` is a command-line version of the **Cheddar** v3 schedulability analysis tool. **Cheddar** v3 models (`.xmlv3`) are generated from the **AADL** specification thanks to a dedicated **LMP** model transformation. **Cheddar** outputs (feasibility tests reports and static chronograms) are displayed in the **AADL Inspector** graphical interface. **Cheddar** is an open source project managed by the University of Brest: http://beru.univ-brest.fr/~singhoff/cheddar

`sbprolog 2.5` is an open source **Prolog** engine that is used by the **LMP** (Logic Model Processing) technology. **AADL Inspector** uses **LMP** to implement the various **AADL** rules checkers and model transformations. **SB-Prolog** was developed by State University of New York at Stony Brook and the University of Arizona.

`marzhin 2.0` is a multi-agents simulator implementing the **AADL** run-time. It consists of three **Java** archive files and requires a **Java** Run-time Environment (**JRE**) to operate. No **JRE** is provided with the **AADL Inspector** distribution. **Marzhin** v2 models (`.xml`) are generated from the **AADL** specification thanks to a dedicated **LMP** model transformation. **Marzhin** outputs (dynamic chronograms) are displayed in the **AADL Inspector** graphical interface. **Marzhin** is developed in collaboration by **Virtualys** and **Ellidiss** Technologies.

2.2.2. Config subdirectory

This directory contains initialization and configuration files that are used by the executable files.

The files having a `.sbp` extension contain a binary form of the **LMP** (Logic Model Processing) rules that are used to perform each analysis action. Checkers provide a direct textual report into the **AADL Inspector** window, whereas adaptors perform dedicated model transformations to interface with ancillary tools such as **Cheddar** and **Marzhin**. More rules files will be added in the future and it is also possible to include corporate or project specific checking or adaptor tools in simply adding a rule file into this directory ("plug and check" feature). Please contact **Ellidiss** Technology (aadl@ellidiss.fr) to use this advanced feature.

The files having a `.ais` extension contain a description of each **AADL Inspector** processing tool or wizard. Each processing tool or wizard provides one or several services that will be available via menu options or buttons. Each service is described by a sequence of elementary commands. Note that the `.aip` plugin description files that where available with the previous versions of **AADL Inspector** are no more usable.

The `AIConfig.ini` file contains the declaration of three sets of user variables: *config*, *plugins*, *gantt*, *accelerators* and *license*. These options are not supposed to be changed by the end user without an explicit recommendation from the technical support.

With the standard distribution, the `config` directory contains the following additional sub-directories and files:

2.2.2.1. Common plugins

These plugins can be removed and customized. New plugins can also be added there. They are not platform dependent. They are located in the `plugins.common` subdirectory.

AADL Static Analysis:

- `AadlAnalysis.ais`: tool description file
- `metrics.sbp`: **AADL** metrics checker
- `naming.sbp`: **AADL** naming rules checker
- `legality.sbp`: **AADL** legality rules checker
- `consistency.sbp`: **AADL** consistency rules checker
- `arinc653.sbp`: preliminary version of the **ARINC 653** rules checker

Schedulability Analysis:

- `Cheddar.ais`: tool description file
- `schedulability.sbp`: **AADL** to **Cheddar** model transformation

Simulation:

- `chronogram.sbp`: chronograms configuration rules
- `marzhin.sbp`: **AADL** transformation rules for **Marzhin**
- `Marzhin.xml`, `MarzhinLogs.xml`: simulation configuration files

Wizards:

- `aadlgen.sbp` : **AADL** generator (unparser)
- `readRTProperties.sbp` : **AADL** real-time properties reader
- `writeRTProperties.sbp` : **AADL** real-time properties writer
- `rootselector.sbp`: **AADL** instance model configuration rules
- `Marte.ais`: wizard description file
- `marte.sbp`: **UML MARTE** to **AADL** transformation rules

2.2.2.2. Platform dependent plugins

The subdirectories `plugins.w32 and plugins.l32` are not currently used. Their role is to give the ability to call platform dependent commands during the execution of a service provided by a processing tool or wizard.

2.2.3. Examples subdirectory

This directory contains a few **AADL** examples to practice the use of **AADL Inspector**. Two kinds of files are proposed: individual **AADL** source files (`.aadl`) or **AADL** Inspector project files (`.aic`) that provide a list of individual **AADL** source file pathname or URL. It is recommended to load a project file instead of individual AADL files, in order to open all the required **AADL** packages and Property Sets that are required to activate the analysis tools.

- `dataflow.aic`: basic set of three periodic tasks
- `messages.aic`: use of queued events
- `shared_data.aic`: use of *Priority_Ceiling_Protocol*
- `client_server.aic` : use of remote subprogram calls
- `timed.aic`: use of *Timed* dispatch protocol
- `hybrid.aic`: use of *Hybrid* dispatch protocol
- `display_system.aic`: large model
- `mars_pathfinder.aic`: show priority inversion problem
- `satellite.aic`: use remote access to the AADLib github (requires internet access)
- `ARINC653` directory: three variants of a simple partitioned system
- `Common` directory : a few predefined AADL Property Sets and libraries

2.2.4. Doc subdirectory

This directory contains a list of `.pdf` files (including this manual) that can be opened from the help menu of the graphical user interface.

2.2.5. Command line options

AADL Inspector can be launched from a command line. The following optional parameters are available:

- `-a file1.aadl, file2.aadl,…`
  open the specified **AADL** files
- `-r dir1, dir2,…`
  open all the **AADL** files contained in the specified directories
- `-p project1.aic, project2.aic,…`
  open all the **AADL** projects contained in the specified directories.
- `-l logdirname`
  use the specified location to create the temporary files. If used, this information overrides the one specified by the *logDirectory* parameter in the `AIConfig.ini` file.
- `--plugin tool.service`
  start a service of a tool as defined in a `.ais` file of the config directory.
- `--result file`
  store the result file in the specified file
- `--result stdout`
  only for **Unix** systems. Display the plugin result in the console.
- `--show false`
  launch **AADL Inspector** without showing the graphical interface.

A particular case where **AADL Inspector** is launched in command line mode is its integrated use with the **Stood AADL** graphical modelling tool. With recent versions of **Stood** (**Stood** 5.3 or more recent), **AADL Inspector** is automatically called after the generation of textual **AADL** code has been completed. The list of **AADL** files that are loaded are all those related to the **Stood** project, i.e. the current design as well as the various libraries.

Another example of use of the command line activation of **AADL Inspector** is to run **Cheddar** on a set of specified **AADL** files and get the results in a specified output file:

```
bin.l32/AADLInspector
   -p examples/dataflow.aic
   --plugin Schedulability.cheddarTheoTest
   --result dataflow.xml
   --show false
```

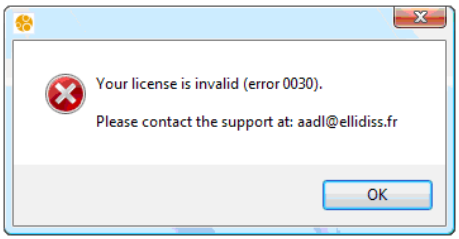
Such a command will create a file containing the result below (fragment). The detailed description of the **Cheddar** output is provided in a separate annex document.

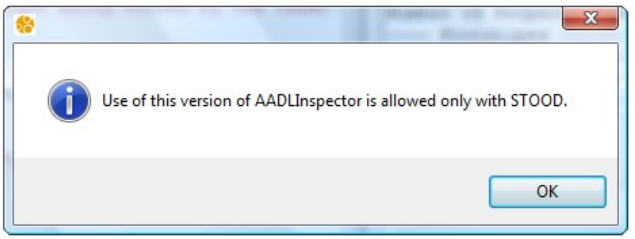
```
<results>
  <feasibilityTest name="processor utilization factor" …>
    <computation name="base period" reference="all" value="300" …/>
    <computation name="processor utilization factor with deadline"
     reference="all" value="0.78333" …/>
    <computation name="processor utilization factor with period"
     reference="all" value="0.78333" …/>
    …
  </feasibilityTest>
  <feasibilityTest name="worst case task response time" …>
    <computation name="response time"
     reference="root.my_platform.CPU.my_process.T1" value="15" …/>
    <computation name="response time"
     reference="root.my_platform.CPU.my_process.T2" value="10" …/>
    <computation name="response time"
     reference="root.my_platform.CPU.my_process.T3" value="5" …/>
    …
  </feasibilityTest>
</results>
```

2.3 License

A valid license is required to use **AADL Inspector**. Various kind of licenses are available, including free of charge evaluation and education licenses. Payment of a license fee is required for commercial or industrial usage of **AADL Inspector**. Please contact your **Ellidiss** sales representative for more details.

License information is stored in the `AIConfig.ini` file. Licenses can be attached to a particular computer and limited in time. In case of a mismatch between the license information stored in the `AIConfig.ini` file and the computer identification or the current date, an error message box is displayed.



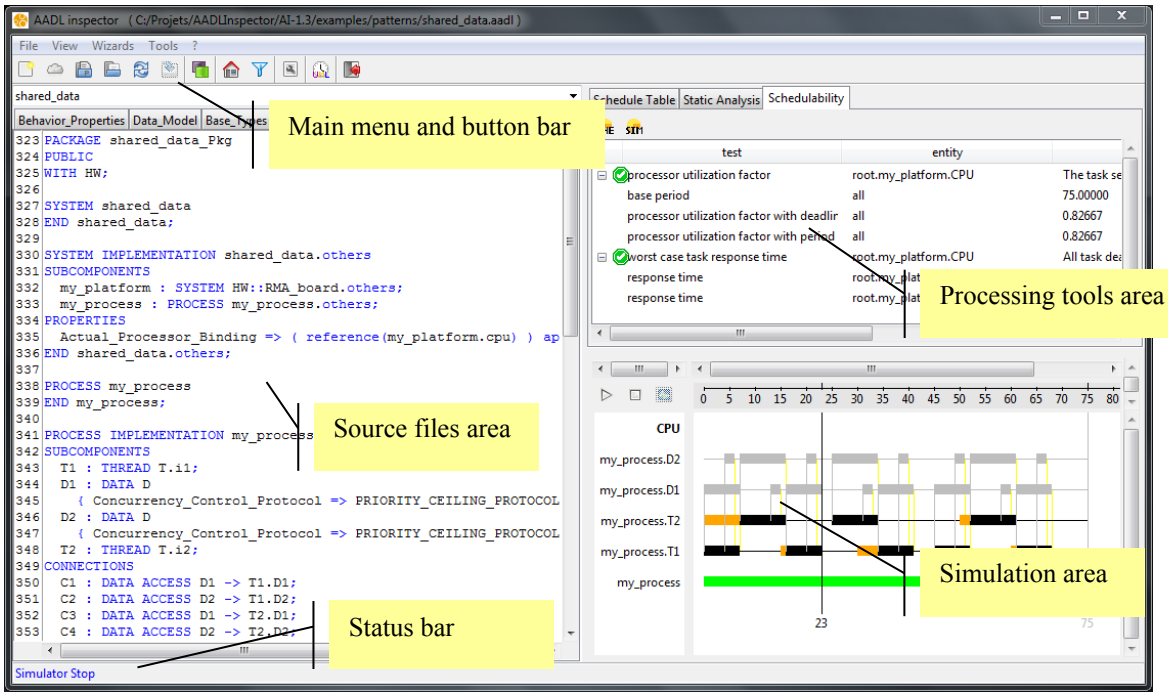


An error number is provided to help identifying the license problem:

- `0010`: this license has expired
- `0020`: this license is not compatible with the encryption key
- `0030`: this license is attached to another computer
- `0040`: this license is linked to a **Stood** license
- `0050`: this license is not valid for this version of the product
- `0060`: this licence is invalid (empty licenseKey)
- `0070`: this licence is invalid (unspecified version)
- `0080`: this licence is invalid (bad licenseKey encoding)
- `0090`: this licence is invalid (bad version encoding)

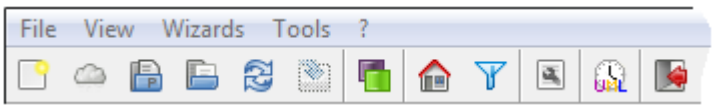
3 Graphical Interface

AADL Inspector opens a single window that encompasses a main menu bar, a button bar, a source files area, a processing tools area, a simulation area and a status bar, as shown below:



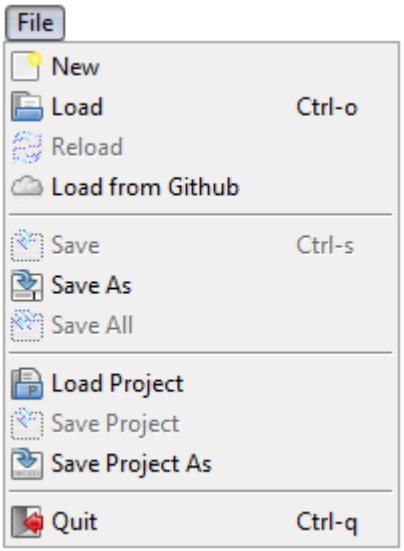
3.1 Main menu and button bar

The menu bar contains the following pull-down menus: *File*, *View*, *Wizards*, *Tools* and *Help*. The button bar provides shortcuts for often used menu options.



3.1.1. File menu

The *File* menu controls all the actions related to the **AADL** files that are processed by the tool. The tool can process several files that contribute together in defining a complete **AADL** specification. The recommended way to manage multiple files is to link them with an **AADL Inspector** project file (`.aic`). There is no particular restriction or naming rule for the **AADL** files. In practice, for most of the processing tools, all the loaded files are concatenated together before being processed by the analysis tools. Please note that load ordering may have an impact on obtained result, especially if the root of the **AADL** instance hierarchy has not been explicitly defined.

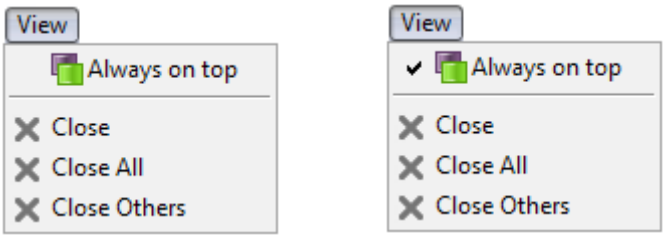


- *New*: create a new AADL virtual file in memory.
- *Load*: load the contents of the specified AADL files into memory.
- *Reload*: cancel any non saved change for the current file.
- *Load from Github*: load files from remote AADL libraries (requires internet access).
- *Save*: save the changes for the current file.
- *Save As*: save the current memory contents into a new file.
- *Save All*: save the changes for all the loaded files.
- *Load Project*: load the content of all the AADL files of the specified `.aic` project into memory.
- *Save Project*: save the changes for the current project.

- *Save Project As*: save all the opened files into a new `.aic` project.
- *Quit*: quit **AADL Inspector**

3.1.2. View menu

The *View* menu provides a menu item that constrains **AADL Inspector** window to remain on the foreground of the computer screen.

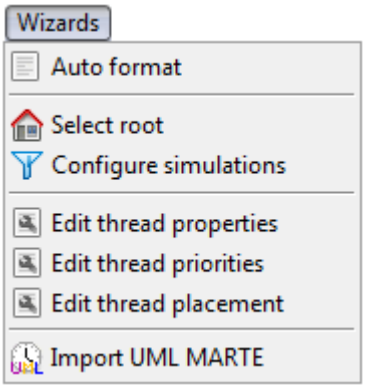


When this menu item has been called once, a tick is put against its label to indicate that this constraint has been enabled. Another activation of this menu item will then disable the constraint and remove the tick.

The other items displayed in this menu are:
- *Close*: unload the current file.
- *Close All*: unload all the loaded files.
- *Close Others*: unload all the loaded files but the current one.

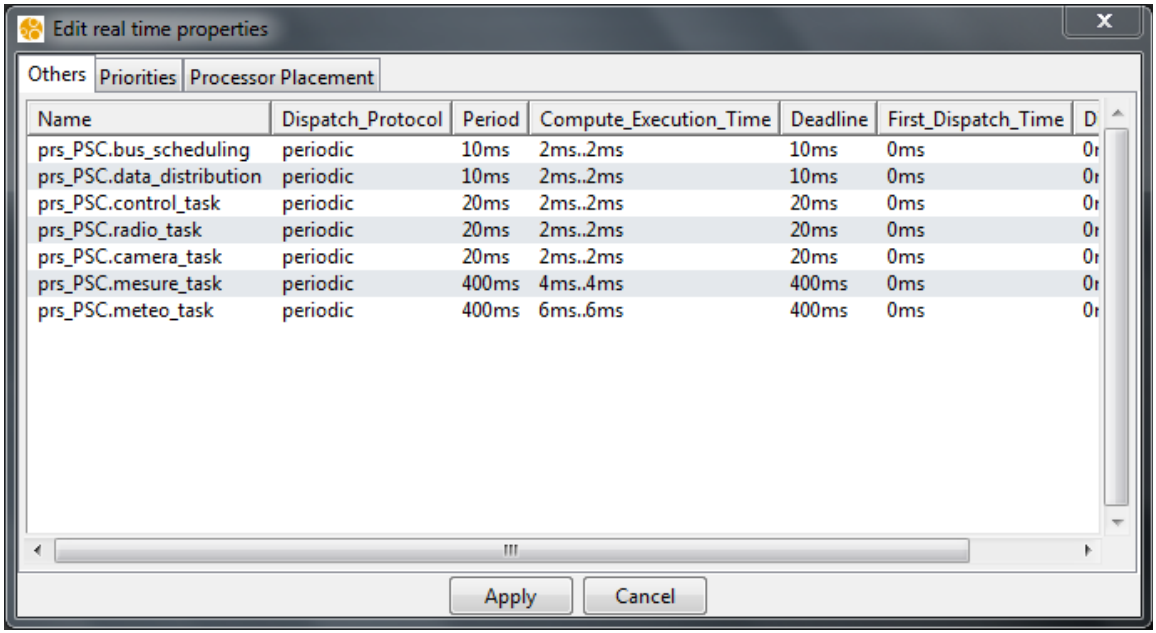
3.1.3. Wizards menu

The *Wizards* menu provides advanced functions that can be used to perform changes on the input **AADL** specification. When possible, the original source text is not modified, and the changes are applied to an extension of the main system implementation of the project instead.



- *Auto format*: re-write the current **AADL** file in a normalized form. This feature can also be used to convert **AADL** v1 and v2.0 files into **AADL** 2.1 syntax.
- *Select root*: provide the list of candidate system implementations and select the one to be the root of the **AADL** instance hierarchy.
- *Configure simulations*: Specify the processor to simulate and identify the entities that must be displayed in the graphical simulation traces.
- *Edit thread properties*: Open a spreadsheet to edit various real-time properties and apply them to the current model.
- *Edit thread priorities*: Compute the priorities according to rate or deadline monotonic algorithms.
- *Edit thread placement*: Generate processor binding according to various placement algorithms.
- *Import UML MARTE*: Load a `.uml` file compliant in **XMI** syntax, apply the **MARTE** to **AADL** model transformation and display the generated **AADL** specification.

When real-time or binding properties have been modified, the corresponding **AADL** property associations are declared as contained properties of an extension of the current root system implementation. The current root system can be selected with the *Select root* wizard, otherwise it is (in decreasing priority order) the first found system implementation containing an `AI::root_system` property association, or the first found system implementation containing an `Actual_Processor_Binding` property association, or the first found system implementation that is not instantiated as a subcomponent.



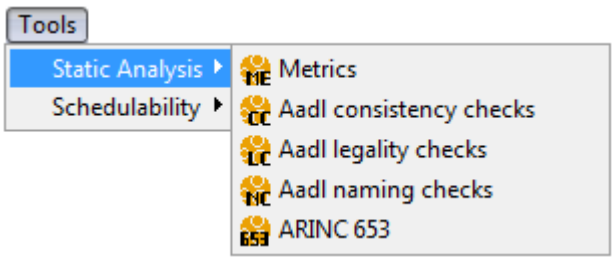
The extended root system is created in memory only and is located in a new `::proxy` subpackage. The newly created system contains a `AI::root_system` property association so that it becomes the new current root system. Note that if the user wants to use this property to define the root system in the original file, he must not use the property value "`selected`" which is reserved for the proxies.

mars_pathfinder::AI_adaptation::proxy

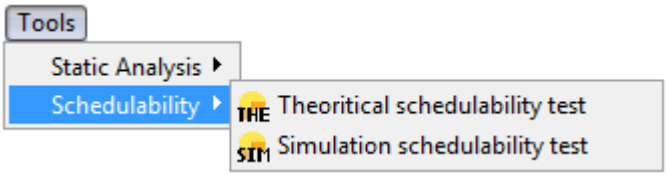
```
1  PACKAGE mars_pathfinder::AI_adaptation::proxy
2  PUBLIC
3  WITH mars_pathfinder::AI_adaptation;
4  WITH AI;
5
6  SYSTEM sys_mars_pathfinder
7  EXTENDS mars_pathfinder::AI_adaptation::sys_mars_pathfinder
8  END sys_mars_pathfinder;
9
10 SYSTEM IMPLEMENTATION sys_mars_pathfinder.impl
11 EXTENDS mars_pathfinder::AI_adaptation::sys_mars_pathfinder.impl
12 PROPERTIES
13   AI::root_system => "selected";
14   Priority => 6 APPLIES TO prs_PSC.bus_scheduling;
15   Priority => 7 APPLIES TO prs_PSC.data_distribution;
16   Priority => 3 APPLIES TO prs_PSC.control_task;
17   Priority => 5 APPLIES TO prs_PSC.camera_task;
18   Priority => 1 APPLIES TO prs_PSC.mesure_task;
19   Priority => 2 APPLIES TO prs_PSC.meteo_task;
20 END sys_mars_pathfinder.impl;
21
22 END mars_pathfinder::AI_adaptation::proxy;
23
```

3.1.4. Tools menu

The *Tools* menu provides access to the processing tools and services that are defined in the `.ais` files located in the `config` directory. Two tools are available with the standard distribution: *Static Analysis* and *Schedulability*. The latter one uses the **Cheddar** tool.



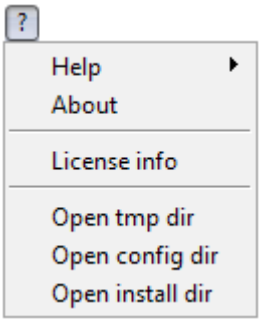
- *Metrics*: provide statistics about both the instance and the declarative **AADL** models.
- *AADL consistency checks*: verify the consistency rules defined by the standard.
- *AADL legality checks*: verify the consistency rules defined by the standard.
- *AADL naming checks*: verify the consistency rules defined by the standard.
- *ARINC 653*: limited set of basic verifications for partitioned systems.



- *Theroritical schedulability test*: set of feasibility tests checked by **Cheddar**.
- *Simulation schedulability test*: set of tests based on the static simulation computed by **Cheddar**.

3.1.5. Help menu

The *?* menu provides information about **AADL Inspector**.



- *Help*: open the help files. Note that the name of the help file directory and the application that is used to open it can be customized in the `AIConfig.ini` file. By default, this application will be the default one for `.pdf` files on **Windows** and `xpdf` on **Linux**.

- *About*: display the version of the software.
- *License info*: provide information about the license.
- *Open tmp dir*: open the temporary subdirectory.
- *Open config dir*: open the configuration subdirectory.
- *Open install dir*: open the installation subdirectory.

3.1.6. Button bar

The button bar provides another entry point for a few selected menu actions.

Effect of these actions is described in the corresponding menu section. Button association with menu bar items is given below from left to right:

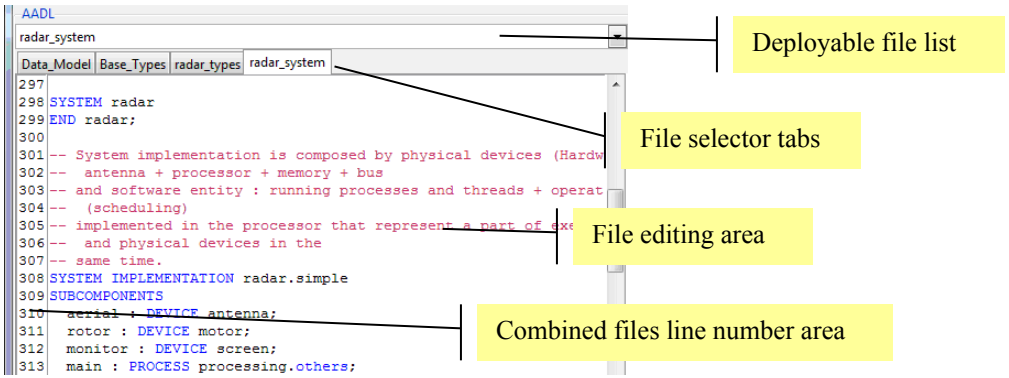
- *File/New*
- *File/Load from Github*
- *File/Load Project*
- *File/Load*
- *File/Reload*
- *File/Save*
- *View/Always on top* (toggle button)
- *Wizards/Select root*
- *Wizards/Configure simulations*
- *Wizards/Edit thread properties, priorities and placement*
- *Wizards/Import UML MARTE*
- *File/Quit*

3.2 Source files area

The **AADL** files that are processed by the tool are managed with the *Source files area*. The tool can process several files that contribute together in defining a complete **AADL** specification. There is no particular restriction or naming rule for the **AADL** files. In practice, all the loaded files are concatenated before being processed by the analysis tools. Please note that load ordering may have an impact on obtained result, especially while looking for the root of the **AADL** instance hierarchy. Displayed line numbers reflect this ordering as it refers to the resulting file concatenation.

The source file area is composed of:

- a *deployable file list*
- a set of *file selector tabs*
- a *file editing area*
- a *line number area*

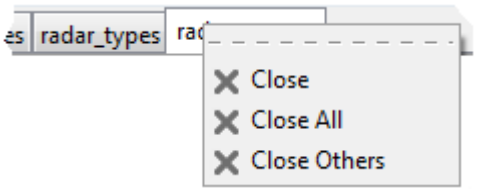


The *deployable file list* provides access to all the loaded files, whereas the *file selector tabs* only show a limited selection of these files for a quicker access.

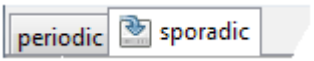
To load a file in the editing area a drag and drop action is possible instead of using the *File/Load* menu: open the appropriate directory, select the desired file, maintain the left mouse button down and drag the mouse until the **AADL Inspector** window is reached. Pease note that this function is not available for Linux distributions.

To find all the occurrences of a word in the displayed file, select the desired word and repeat the Ctrl-F action to navigate in the file.

A contextual menu (right mouse button click) is associated to the current *file selector tab*. It provides an easy way to close files. Effect of these menus is similar to the corresponding ones in the *File* menu.



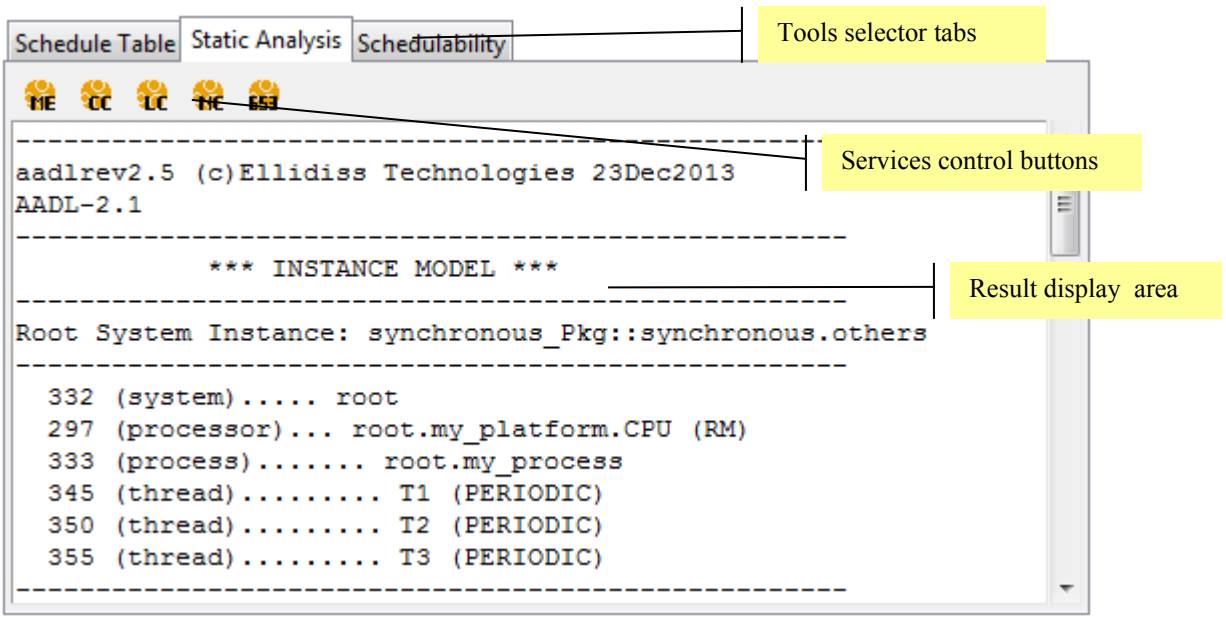
When a file has been modified, an icon appears on the tab to indicate that the changes are not saved.



3.3 Processing tools area

The *processing tools area* allows for selecting the processing tool to be applied on the **AADL** files loaded in the source files area, and displaying the corresponding execution result. This area is composed of:

- a set of *tool selector tabs*
- one or several *service control buttons*
- a read-only *result display area*



- *Tools selector tabs* can be configured by adding or removing tool description files (`.ais` files) in the `config` subdirectories of the installation directory

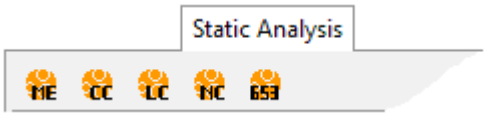
If one of the analysed file is modified, the background colour of the result display area becomes gray to indicate that the information is potentially out of date.

When the selected analysis tool can not be executed normally for the current **AADL** specification or if the **AADL** syntax is not correct, the corresponding error message will appear in the *Report* tab. Clicking on the line number included in the error message will highlight the corresponding line in the *source files area*.

While working on large **AADL** projects, processing actions may take a significant time (up to a few minutes). Depending on the processing tool that is running, other user actions may be allowed or not and the display may not be refreshed during that time.

3.3.1. Static Analysis

The static analysis tool encompasses a set of independent rules checkers that verify various facets of the semantic correctness of the source **AADL** specification. Each rules checker is implemented as a service of the static analysis tool and can be activated by pressing on the correspondent button:



- the *ME* button launches the Metrics checker.
- the *CC* button launches the **AADL** Consistency rules checker.
- the *LC* button launches the **AADL** Legality rules checker.
- the *NC* button launches the **AADL** Naming rules checker.
- the *653* button launches the **ARINC 653** rules checker.

More detailed explanations about the scope of each of these checkers can be found in separate documentation documents.

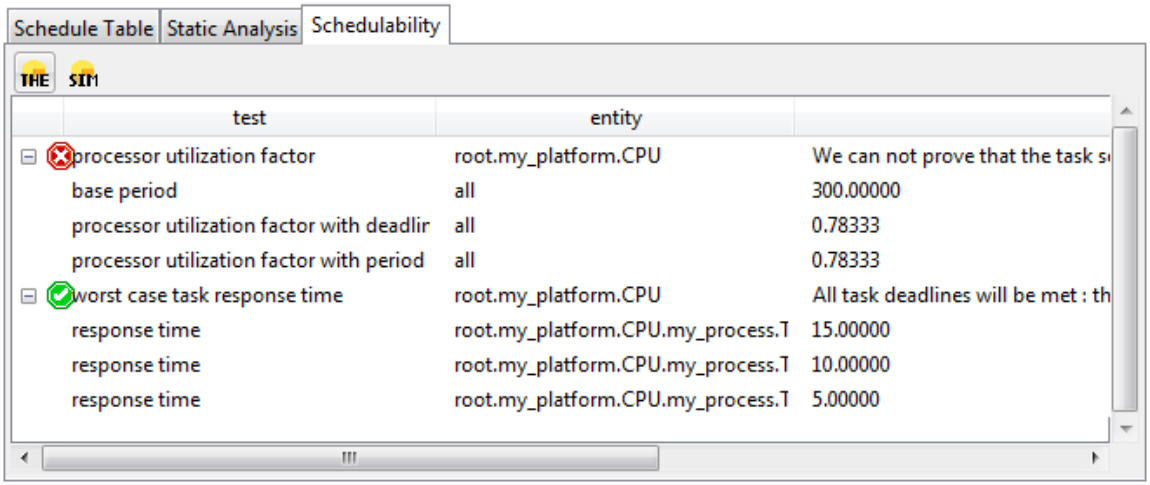
When an error, warning or information message is displayed by a processing tool, the line number of the corresponding **AADL** code is highlighted in both the *source files area* and *processing tools area*. Clicking on an error line number of the *processing tools area* automatically changes the source code display in order to let the erroneous line be visible.

3.3.2. Schedulability analysis

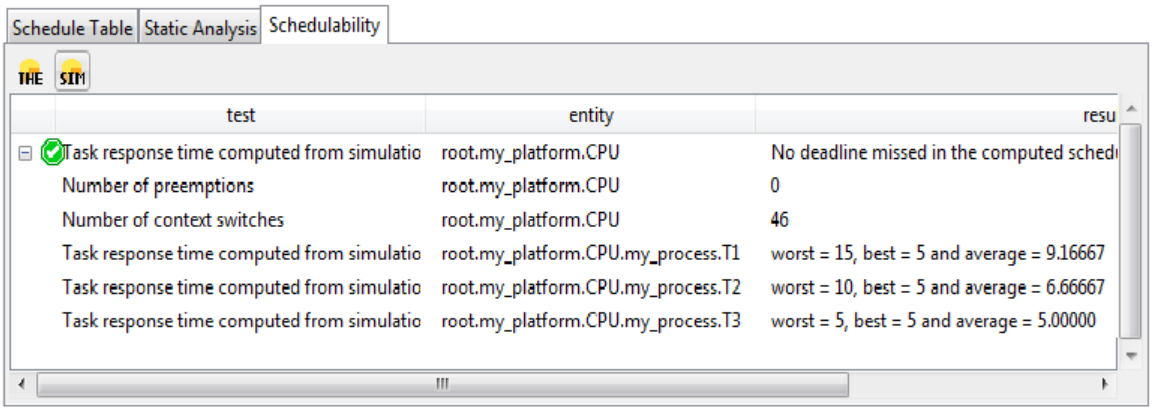
When the *schedulability* tab is selected, two buttons are proposed to activate the analysis services provided by Cheddar:

- the *THE* button launches the Scheduling Theoretical Tests
- the *SIM* button launches the Scheduling Simulation Test

Theoritical tests compute processor utilization factor and threads response time when the corresponding conditions are met.



Simulation tests provide information about the number of pre-emption and context switch as well as threads response time. This static simulation can only be run for periodic systems.

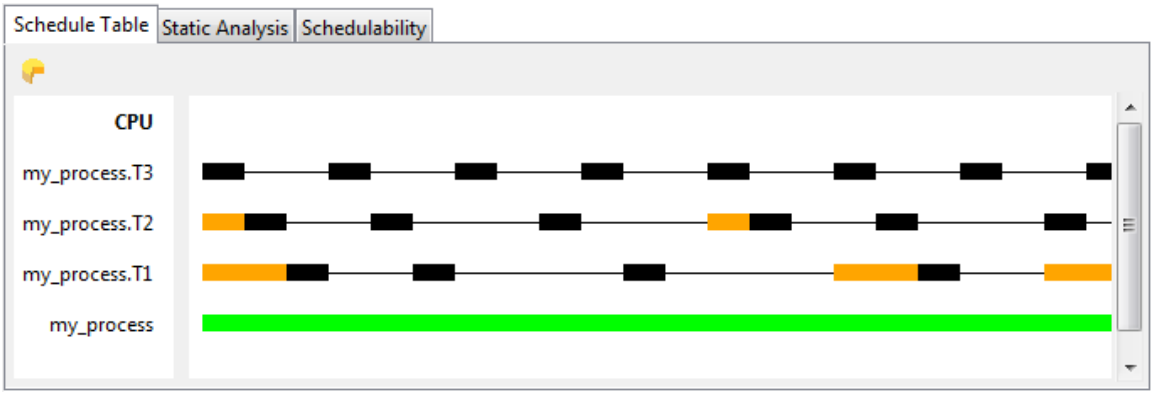


More detailed explanations about the scope of each of these tests can be found in a separate documentation document.

3.3.3. Schedule table

Cheddar can also produce a graphical representation of the timing behaviour of the real-time system being analysed. This *Schedule Table* is a result of the static simulation and may not be available on any kind of system.

Note that it is required to click on the yellow button to launch the computation of the *Schedule Table*.



Time lines are displayed for each process, thread and shared data that are active on the currently selected processor. The time scale and meaning of each used colour are shared with the dynamic simulator that is described below.

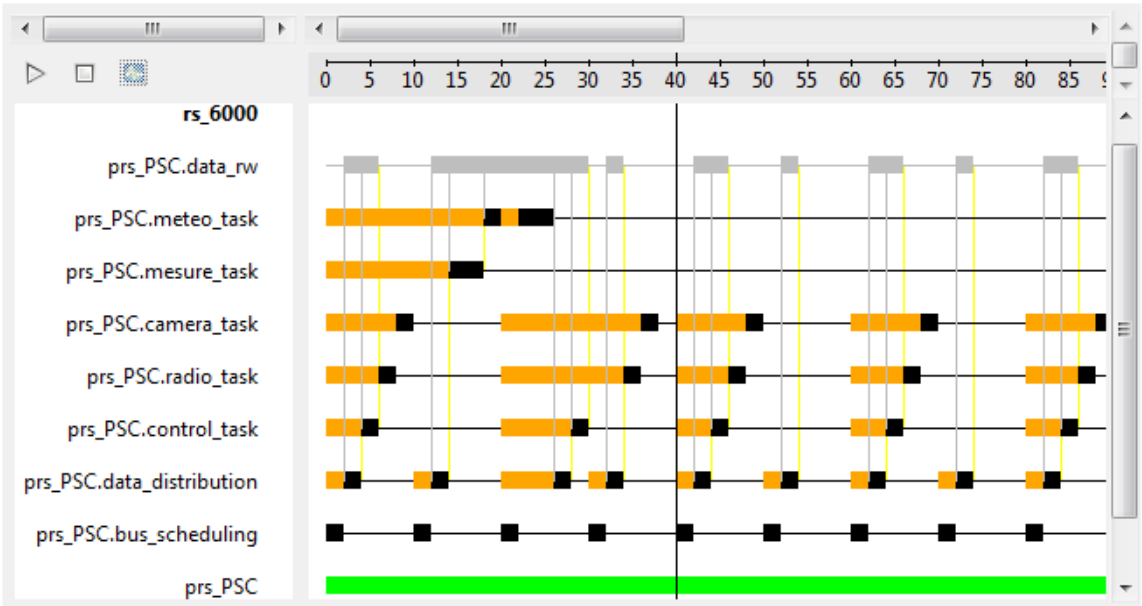
3.4 Simulation area

The *simulation area* is dedicated to controlling and displaying the output of the **Marzhin** dynamic simulator. This simulator complements the static simulator that is provided by **Cheddar** but is event-driven and can analyse more kinds of real-time systems. The counter part is that the obtained time lines are not the result of mathematical computation and are thus less dependable.

The simulation area is composed of:

- a specific *toolbar*
- a simulator output area showing time lines for each process, thread and shared data that is active on the currently selected processor.

In addition, the *Configure simulations* wizard can be used to set up the time scale and filter the entities to be displayed for the simulation. This wizard can be activated from the *Wizards* menu or the corresponding button in the main buttons bar.



3.4.1. Simulator toolbar and control panel

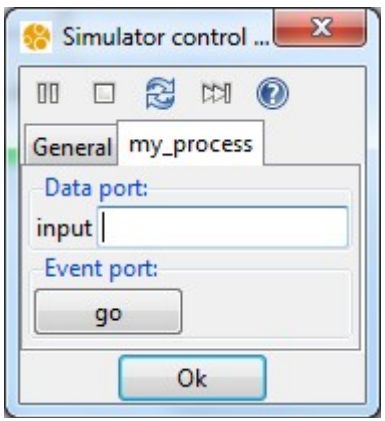
The simulator toolbar is composed of the following buttons:

▷	start the simulator
▯▯	pause the simulator
□	stop the simulator
	open the simulator control panel

The *simulator control panel* can be used to send commands to the simulator. It is also possible to control the simulator from the *Tools* menu. Depending on the status of the simulation, the following actions are proposed:

▷	start the simulator
▯▯	pause the simulator
□	stop the simulator
▷▯	go to the current tick
	refresh the simulation input file
	provide help about the simulation trace

When the *General* tab of the *simulator control panel* is selected, it allows for selecting the simulation speed. Note that the speed factor can be customized in the `AADLConfig.ini` file.



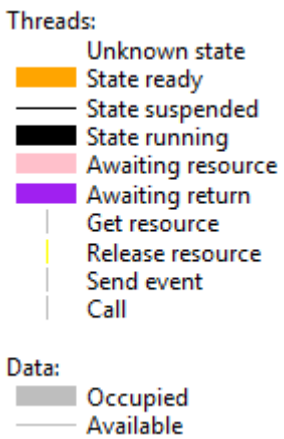
For each partition (process) that is bound on to the current processor, another tab is added to the simulator control panel. This tab can be used to dynamically modify the value of input data ports and send event ports in the process interface. Note that only integer

values are processed for data ports. These data and events can be used by the **AADL** behaviour annex subclauses in threads in order to control conditional execution at simulation runtime.

3.4.2. Simulator chronograms

Each chronogram displays the run-time information for a given processor. A separate time-line is shown for each partition (process), each thread and each shared data component.

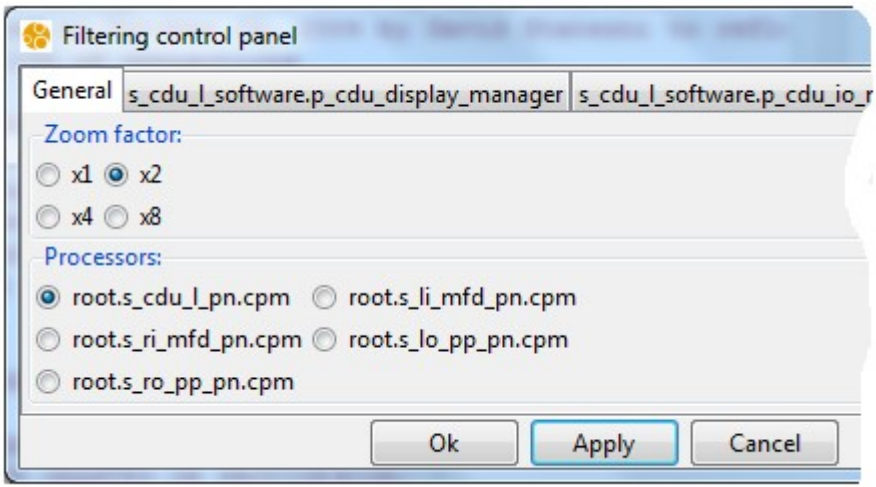
The colour code that is used for the time lines can be configured in the `AIConfig.ini` file and displayed by the *help* button of the *Simulation control panel*.



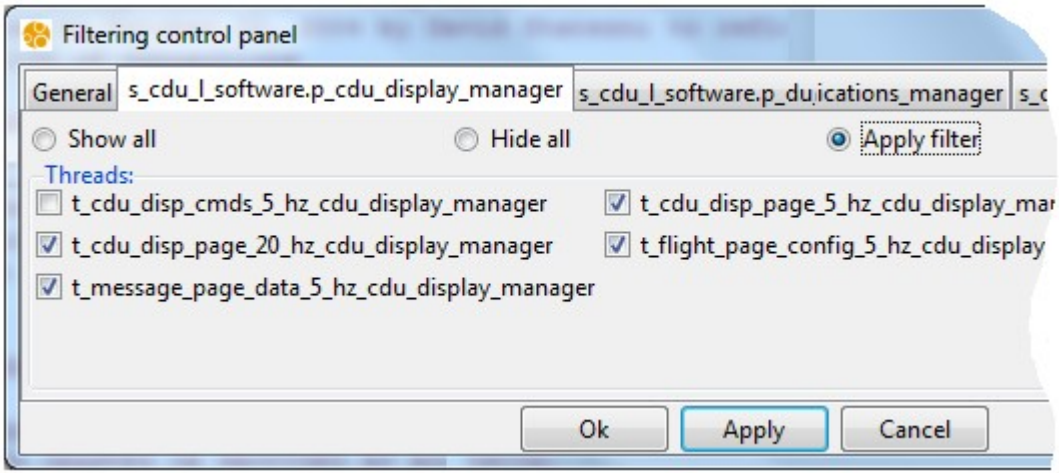
The *Simulator control panel* can be used to start, pause and stop the simulation, to change the simulation speed and to dynamically inject data or events through ports of the current process.

The *Configure simulations* wizard opens the *Filtering control panel* that can be used to display the chronogram for another processor, to reduce the number of time-lines and to change the zoom factor (width of a time tick).

Within the *Filtering control panel*, the *General* tab can be used to define the zoom factor and select the processor to be displayed in the simulation area. Note that all the processors are simulated and it is possible to switch the display to show the state of another processor at any time.



The *Filtering control panel* also provides an additional tab for each process that is running on the current processor. This tab can be used to filter the number of entities to be displayed.



To restrict the number of threads or shared data to be displayed, select the *Apply filter* radio button and unselect the components to be removed. It is required to validate the changes by clicking on the *Apply* button.

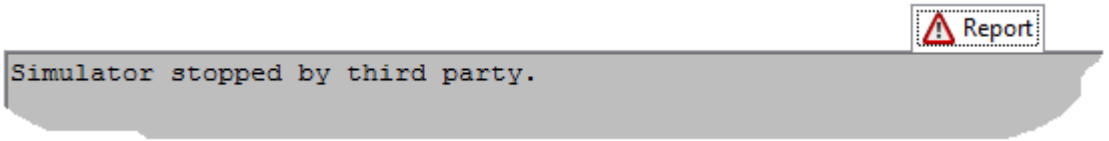
Note that zoom factor, threads or data property changes also affect the **Cheddar** *Schedule Table*.

Note that **AADL** time units conversion is not taken into accounts. In order to obtain consistent results, it is required that the same time unit for all the timing properties in the **AADL** model.

3.5 Status bar and Error Report

The status bar located in the lower part of the window shows various informational or error messages generated by **AADL Inspector**:

When relevant, detailed error messages are displayed in the *Report* tab.



www.ellidiss.com

Sales office:
TNI Europe Limited
Triad House
Mountbatten Court
Worall Street
Congleton
Cheshire
CW12 1AG
UK
info@ellidiss.com
+44 1260 291 449

Technical support :
Ellidiss Technologies
24 quai de la douane
29200 Brest
Brittany
France

aadl@ellidiss.fr
+33 298 451 870