

# **MARZHIN 2.0**

## **AADL Inspector**

### **Plugin Manual**

*Ellidiss Technologies*  
<http://www.ellidiss.fr>  
*aadl@ellidiss.fr*

# 1 Multi-Agents simulation principles

**Marzhin** simulator that is embedded inside **AADL Inspector** is based on a “Multi-Agents” simulation kernel. The main advantage of this approach is that each **AADL** thread is implemented by an autonomous “Agent”, which behaviour directly follows the semantics specified by the standard. Threads interactions are only described architecturally and the complexity of the global scheduler is thus dramatically reduced.

The simulator operates as follow: each simulation cycle starts with an evaluation of the priority of each thread according with the specified scheduling and concurrency control protocols. Then all the threads are randomly awoken and react according to their current state and specified dispatch protocol. Finally, one instruction belonging to one of the eligible threads is executed, the new state of each thread is evaluated and a new cycle can begin.

**AADL Inspector** uses its own **AADL** import feature that is implemented with the **LMP** technology developed by Ellidiss. This approach makes use of a generic **AADL** syntactic analyser (*aadlrev*) that produces a list of **Prolog** predicates describing the **AADL** specification to be analysed. An **AADL** to **Marzhin** model transformation expressed by a set of **Prolog** rules is then used to generate the corresponding appropriate **XML** input file for **Marzhin**. This model transformation is stored in the *marzhin.sbp* file within the *config* directory of the product installation.

The **AADL** instance hierarchy is dynamically deduced from the **AADL** declarative model defined by the current set of opened files. The root of the instance hierarchy is automatically set to the first found System component declaration that contains an `Actual_Processor_Binding` property association.

## 2 Supported AADL Run-Time semantics

### 2.1 Processor

The current version of **Marzhin** only supports a single **AADL** Processor. The first processor subcomponent that is found in the deduced instance hierarchy will be seen as the one to simulate. Then, an `Actual_Processor_Binding` property value is searched within the encompassing **AADL** System implementation in order to identify the corresponding **AADL** Processes and the specified **AADL** `Scheduling_Protocol` property value. Finally, all the **AADL** Thread, Subprogram and Data subcomponents, as well as the corresponding **AADL** connections are analysed to initialize the simulation.

<i>Supported Scheduling Protocols</i>	
Rate_Monotonic_Protocol (or RM)	each thread must have a <code>period</code>
Deadline_Monotonic_Protocol (or DM)	each thread must have a <code>deadline</code>
POSIX_1003_Highest_Priority_First_Protocol (or HPF)	each thread must have a <code>priority</code>
Hierarchical_Offline_Protocol (or ARINC653)	Each processor must define the following properties: <code>ARINC653::module_major_frame</code> <code>ARINC653::partition_slots</code> <code>ARINC653::slots_allocation</code>

In case of a rate monotonic scheduling protocol, thread priorities are set according to the value specified in their standard **AADL** `period` property value. In that case, the highest priority is given to thread having the shortest period.

In case of a deadline monotonic scheduling protocol, thread priorities are set according to the value specified in their standard **AADL** `deadline` property value. In that case, the highest priority is given to thread having the shortest deadline.

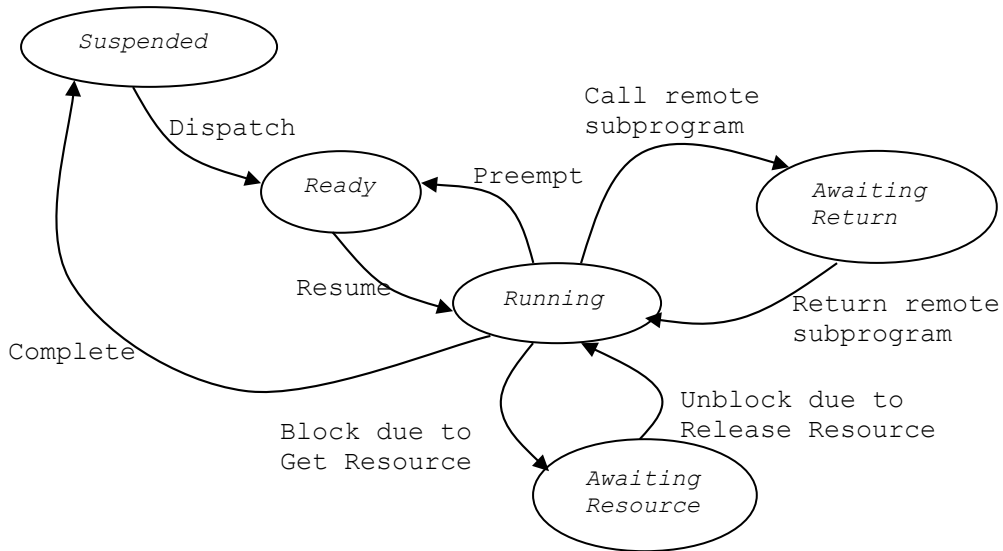
In case of a highest priority first protocol, thread priorities are set according to the value specified in their standard **AADL** `priority` property value. Note that the internal representation for priorities ordering is reversed: for the simulator, the highest priority value is zero. When priority values are set in the **AADL** specification, there are known by the simulator as “user priority” values.

In case of a hierarchical scheduler (**ARINC 653**), the AADL model must comply with the ARINC 653 Annex to the standard. In particular, each partition must be defined by the binding between a Virtual Processor and a Process. In addition, the ARINC653::Module\_Major\_Frame, ARINC653::Partition\_Slots and ARINC653::Slots\_Allocation must be specified with appropriate values.

The simulator also supports the AADL property Scheduler\_Quantum which specifies the maximum continuous amount of time that can be allocated to a given thread. In practice, this property is only used by **Marzhin** to schedule multiple Background threads with a round robin protocol. If no Scheduler\_Quantum is specified, the highest priority background thread runs until completion.

## 2.2 *Thread*

The AADL standard defines the possible states and state transitions of a thread during run-time. The subset of this state automaton that is currently supported by **Marzhin** is shown below:



A thread is in state *Suspended* when it has finished all its computation and is waiting for another dispatch. When it is dispatched, it reaches state *Ready* where it stays until it is the highest priority thread, in which case it moves to state *Running*. In state *Running*, the thread executes its instructions until it completes and goes back to state *Suspended*. Meanwhile, its execution may be delayed by higher priority threads, in which case its state becomes *Ready*, by unavailable resources, in which case its state becomes *Awaiting Resource*, or by remote subprogram call execution, in which case its state becomes *Awaiting Return*.

Thread dispatch is controlled by the **AADL** *Dispatch\_Protocol* property value. **Marzhin** supports all the dispatch protocols that are specified by the AADL standard.

<i>Supported Dispatch Protocols</i>	
Periodic (default)	dispatched periodically with specified period
Aperiodic	dispatched by received events
Sporadic	same as Aperiodic, with a minimum inter-arrival time
Timed	same as Aperiodic, with a timeout

Hybrid	disjunction of Periodic and Aperiodic dispatch conditions
Background	dispatched when the processor is free

Threads are described by their standard **AADL** features, property values and annexes. **Marzhin** only considers the following the ones specified below:

<i><b>Supported Thread Features</b></i>	
Event Port	used to dispatch Aperiodic, Sporadic, Timed and Hybrid threads
Data Port	local variable whose value can be accessed within a Behavior Annex
Event Data Port	combination of a Data Port and a Event Port
Provides Subprogram Access	same as Event Port
Requires Subprogram Access	used to express a remote subprogram call
Requires Data Access	used to express a remote data access

<i><b>Supported Thread Properties</b></i>	
Period	required for Periodic, Sporadic, Timed and Hybrid threads
Deadline	required when the scheduling protocol is Deadline Monotonic Protocol
Priority	required when the scheduling protocol is POSIX 1003 Highest Priority First Protocol
Compute_Execution_Time	required when no behaviour annex is specified

In addition, a few additional standard **AADL** properties can be used to refine the behaviour of dispatching events:

<i><b>Supported Feature Properties</b></i>	
Queue_Size	maximum number of stored events or calls (default is 1)
Dequeue_Protocol	may be OneItem or AllItems

When a thread is in state `Running`, it executes a sequence of instructions that can be specified by a `Behavior_Specification` annex, otherwise default instructions are deduced from existing features and properties. Supported instructions are described in a

further section.

### 2.3 *Shared Data*

**AADL** data subcomponents can be shared between thread subcomponents thanks to data access features and connections. If no `Behavior_Specification` annex subclause is defined for the thread accessing a shared data subcomponent, the corresponding critical section will be considered starting at `Dispatch` time and ending at `Complete` time. However, it is possible to define a more precise critical section using appropriate instructions in a `Behavior_Specification` subclause.

By default, no specific protocol is supported by the simulator for concurrent access to shared data subcomponents. The only one that is supported at that time is `Priority_Ceiling_Protocol` which requires to be specified in a proper `Concurrency_Control_Protocol` property.

<i>Supported Data Properties</i>	
<code>Concurrency_Control_Protocol</code>	can be used to ensure mutually exclusive access with: <code>Priority_Ceiling_Protocol</code>

### 2.4 *Subprograms*

A client-server communication can be expressed in **AADL** thanks to subprogram access features. Client thread must have a `requires` subprogram access feature whereas the server thread must have a `provides` subprogram access feature. Both features must then be linked by a proper `access` connection. Note that local calls to subprogram subcomponents within the same thread are not supported yet.

The behaviour of the called subprogram must be expressed within the corresponding subprogram component classifier, under the form of a `Behavior_Specification` annex subclause or a `Compute_Execution_Time` property value.

<i>Supported Subprogram Properties</i>	
<code>Compute_Execution_Time</code>	specify use of the processor for a given duration. Required when no behaviour annex is specified



## 2.5 Instructions

**Marzhin** can process a limited set of execute instructions that may be explicitly specified with standard **AADL** `Behavior_Specification` annex subclauses attached to threads or subprogram component classifiers, or implicitly from existing features and properties.

<i>Supported Behavior Annex Actions</i>	
Computation	specify use of the processor for a given duration
!<	specify lock of a shared data ( <code>GetResource</code> )
>!	specify unlock of a shared data ( <code>ReleaseResource</code> )
!	specify send of an out event or call of a subprogram
:=	assigns a value to an out data port
If	specify a conditional action block
For	specify a loop action block

Note that only integer values are currently supported for Data Ports.

## 2.6 Main shortcomings

In addition to the fact that the simulation can only deal with a single processor, this first version of **Marzhin** suffers from three main restrictions.

The first limitation resides in that each instruction defines in a Behavior Annex subclause has a fixed duration of one execution tick. If both a `compute_execution_time` property and a Behavior Annex are provided, the latter will be taken into accounts.

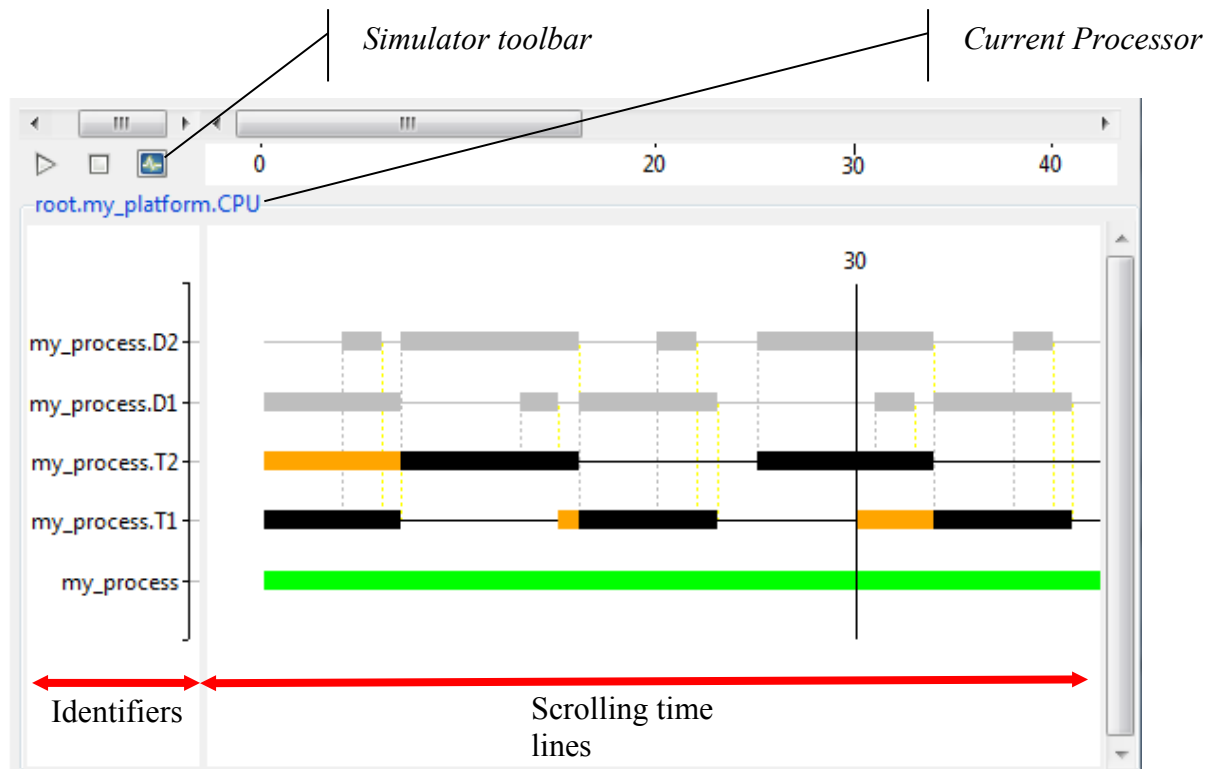
The second restriction is related to the time scale. As opposed to normal **AADL** specifications, it is not possible to use different time units within the same **AADL** specification to simulate. Time values are interpreted by their integer value, and a value of one represents a single tick in the simulation trace.

Finally, as described above in this section, **Marzhin** does not support the entire set of

AADL constructs yet. In particular, AADL modes and mode transitions are currently not implemented.

### 3 Simulation output



When the simulator is running, it sends information about the current state of each element of the architecture such as partitions (processes), threads, port and subprogram queues, data sharing status. This information is refreshed at each simulation step (tick) and displayed in the simulation area. It can also be stored in a file when the appropriate control buttons are used.









#### 3.1 Chronograms

The chronogram represents the activity of the current processor. In case of a multi-processors system, the processor to be analysed can be specified in the simulator control panel. Each chronogram is composed of a separate time line for each process, thread, and shared data.



In case of a multi-partitions processor (**ARINC 653**), the time slots allocated to each process is represented by a discontinuous line. Otherwise, the single process is shown as a continuous line. Possible states for processes are:

Active	
Inactive	

Each thread is identified by the name of the process followed by the thread subcomponent full name (i.e. its name in the instance hierarchy). Thread state is represented by a symbol, as specified in the table below:

Unknown	
Running	
Ready	
Awaiting Resource	
Awaiting Return	
Suspended	

Each shared data subcomponent is identified by the name of the processor followed by the data subcomponent full name. Data sharing state is represented by a symbol, as specified in the table below:

Busy	
Free	

Each thread feature is identified by the full name of the thread subcomponent followed by the name of the feature. Feature state is represented by a symbol, as specified in the table below:





Get_Count on In Event (Data) Port	1..9 or + if more than 9
-----------------------------------	--------------------------

Get_Count on Provides Subprogram Access	1..9 or + if more than 9
---	--------------------------






The graphical properties can be controlled to best fit screen width and identifiers length. This can be done thanks to the Simulator Properties button and the scrolling time lines:

### 3.2 *Simulator toolbar and control panel*

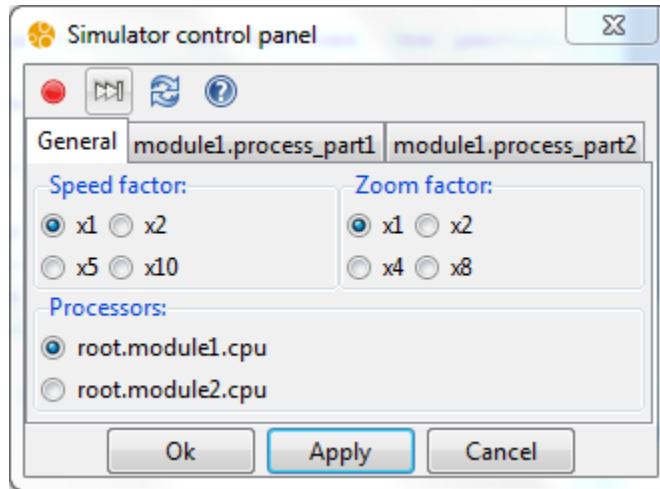
The simulator toolbar is composed of the following buttons:

	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:

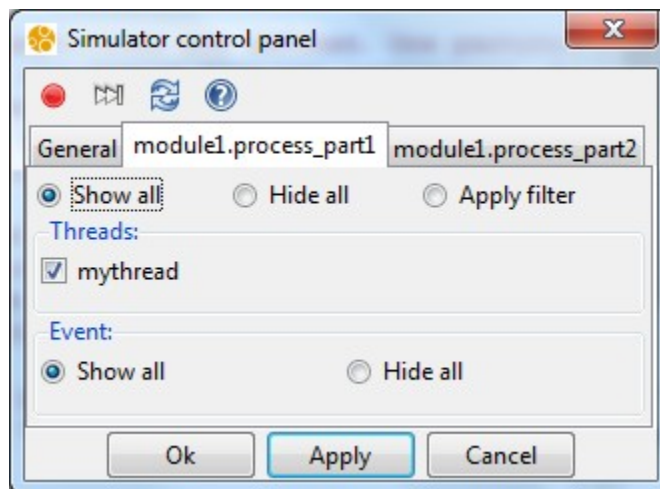
	start recording the simulation trace
	stop recording the simulation trace
	go to the current tick
	refresh the simulation input file
	provide help about the simulation trace

When the *General* tab of the *simulator control panel* is selected, it allows for selecting the processor to analyse, the simulation speed and the time line zoom factor:



Note that the speed factor and the zoom factor values 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 filter the information to be displayed on the chronogram.



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.





[www.ellidiss.com](http://www.ellidiss.com)

Sales office:

TNI Europe Limited  
Triad House  
Mountbatten Court  
Worall Street  
Congleton  
Cheshire  
CW12 1AG  
UK  
[info@ellidiss.com](mailto:info@ellidiss.com)  
+44 1260 291 44

Technical support :

Ellidiss Technologies  
24 quai de la douane  
29200 Brest  
Brittany  
France

[aadl@ellidiss.fr](mailto:aadl@ellidiss.fr)  
+33 298 451 870