

# Cecilia-Workshop



## ***CECILIA WORKSHOP***



**SATODEV**  
SAFETY TOOLS DEVELOPMENT

# *User's Manual*

Version 6.1

2022

Copyright © 2022 Dassault Aviation

## **Abstract**

This document is the user manual of the *Cecilia-Workshop* software which belongs to Dassault Aviation, on a Windows or Linux client station.

## **Limitation on Warranties and Liability**

The information contained in this document is subject to change without prior notice and in no way constitutes an engagement by Dassault Aviation. The reproduction or transmittal of any part of this document in any form, by any means (including electronic, mechanical, photocopy and recording), for any other purpose than the personal use of the person making the copy is strictly prohibited without the written consent of Dassault Aviation.

No statement contained in this document, including statements regarding capacity, suitability for use, or performance of products, shall be considered a warranty by Dassault Aviation for any purpose or give rise to any liability by Dassault Aviation.

In no event will Dassault Aviation be liable for any incidental, indirect, special or consequential damages (including lost profits) arising out of or relating to this document or the information contained in it, even if Dassault Aviation has been advised, knew or should have known of the possibility of such damages.

Copyright © 1999 - 2022 Dassault Aviation, All rights reserved.

# Table of Contents

<b>1. Presentation .....</b>	<b>6</b>
1.1. Introduction .....	6
1.2. Hardware architecture .....	6
1.3. Installation .....	6
<b>2. First step with Cecilia-Workshop .....</b>	<b>7</b>
2.1. Cecilia-Workshop launch .....	7
2.1.1. Database selection .....	7
2.1.2. Licenses Management .....	9
2.1.3. Cecilia-Workshop Launching .....	10
2.2. Work environment .....	11
2.2.1. Cecilia-Workshop principle .....	11
2.2.2. Presentation .....	11
2.2.3. Menu bar .....	12
2.2.4. Main tool bar .....	12
2.2.5. Work area .....	14
2.3. Main functionalities .....	14
<b>3. Technical object .....</b>	<b>18</b>
3.1. Technical object edition .....	18
3.2. Data access rights management .....	21
<b>4. Common approach .....</b>	<b>23</b>
4.1. Law .....	23
4.2. Named parameters management .....	24
4.3. Event model (Failure Rate Base - FRB) .....	25
4.4. Distribution .....	26
4.5. Attributes management .....	26
4.6. Event settings .....	27
<b>5. Boolean models .....</b>	<b>29</b>
5.1. Fault Tree .....	29
5.1.1. Fault Trees edition .....	29
5.1.2. Events edition .....	33
5.1.3. List of events .....	34
5.1.4. Minimal cut set calculations (MCS) .....	37
5.1.5. Common attribute analysis (CAA) .....	42
5.1.6. Minimum equipment list computations (MEL) .....	45
5.1.7. Reduce to flight duration .....	47
5.1.8. List of errors .....	48
5.1.9. Sub-Tree .....	48
5.2. Common Cause Failure (CCF) .....	50
5.3. Dependable System Faults (DSF) .....	51
<b>6. Model-Based Safety Assessment (MBSA) .....</b>	<b>54</b>
6.1. AltaRica - Support language for MBSA .....	54
6.1.1. Historical .....	54
6.1.2. Main Characteristics .....	54
6.2. Input AltaRica model .....	55
6.2.1. General .....	55
6.2.2. Icon manager .....	55
6.2.3. Type management .....	56
6.2.4. Operator management .....	60
6.2.5. Component management .....	64
6.2.6. Equipment management .....	72
6.2.7. Model management .....	79
6.3. Verify or Translate model .....	94
6.3.1. Syntactical check .....	95
6.3.2. Property check/Control .....	97
6.3.3. Model translation .....	100

6.4. Simulation Step by Step .....	101
6.4.1. Simulation configuration .....	101
6.4.2. Simulation launch .....	101
6.4.3. Simulating .....	103
6.4.4. Transitions triggering .....	105
6.4.5. State variable modification .....	106
6.4.6. Simulator display window .....	107
6.4.7. Other functionalities during simulation .....	113
6.5. Models processing .....	117
6.5.1. FaultTree generation with ABC .....	117
6.5.2. Sequences generation (Generic) .....	121
6.5.3. FMEA generation help .....	125
6.5.4. Others tools .....	128
<b>7. Input / Output .....</b>	<b>130</b>
7.1. Setup .....	130
7.2. Print / Data report .....	130
7.2.1. General principle .....	130
7.3. Import/Export data into XML format .....	134
7.3.1. MBSA models Export .....	134
7.3.2. MBSA models Import .....	135
7.3.3. Fault tree models Import .....	137
7.3.4. Fault tree models Export .....	138
7.4. Fault trees DAG Import/Export .....	139
7.4.1. Import .....	139
7.4.2. Big tree .....	140
7.4.3. Export .....	141
7.5. Others import format possible .....	142
7.5.1. ARB file import .....	142
7.5.2. Cafta file import .....	143
<b>8. FMEA .....</b>	<b>145</b>
8.1. Main principles .....	145
8.2. FMEA models .....	145
8.2.1. The FMEA model manager .....	145
8.2.2. Creating a model .....	146
8.2.3. XML import of an FMEA model .....	160
8.2.4. XML export of an FMEA model .....	161
8.3. The FMEA tables .....	162
8.3.1. Fmea tables management .....	162
8.3.2. Creating an FMEA table .....	163
8.3.3. Editing an FMEA table .....	165
8.4. Referencing data in FMEA tables .....	174
8.4.1. Referencing an FMEA table data from an FRB event model .....	174
8.4.2. Referencing an FMEA table data set from an FMEA table .....	177
<b>9. Grid computing .....</b>	<b>180</b>
9.1. Introduction .....	180
9.2. Grid computing interface .....	181
9.2.1. Presentation of interface .....	181
9.2.2. Requests menu .....	181
9.2.3. Distribution menu .....	181
9.2.4. Network menu .....	182
<b>10. Others functionalities .....</b>	<b>184</b>
10.1. Event global list .....	184
10.1.1. Introduction .....	184
10.1.2. Data generation .....	184
10.1.3. Data extraction .....	184
10.1.4. Data display .....	185
10.2. Batch computation .....	187
10.3. Task manager .....	189



<b>A. Laws and uncertainties .....</b>	<b>190</b>
<b>B. Probabilities computations .....</b>	<b>196</b>
<b>C. Importance factors .....</b>	<b>201</b>
<b>D. Minimal Cut Set utilities .....</b>	<b>203</b>
<b>E. Language AltaRica .....</b>	<b>208</b>
<b>F. AltaRica Extended Grammar .....</b>	<b>223</b>
<b>G. AltaRica model verification .....</b>	<b>228</b>

# 1. Presentation

## 1.1. Introduction

Cecilia-Workshop tool allows to perform several safety studies using two different approaches:

- MBSA approach (Model Based Safety Analysis) ;
- FTA approach (Fault Tree Analysis).

The software includes the following functions:

- Modeling of functional and dysfunctional behavior of systems by means of reusable components libraries;
- Construction and interactive graphical simulation of system architectures;
- Automatic control of the consistency of system behavior;
- Automatic generation of fault trees concerning targeted unexpected events;
- Create fault tree in order to analyze the conditions and factors causing an undesired event. The fault tree is a graphical representation of these causes using Boolean (AND, OR, etc...) expression between events;
- Several laws are implemented in the software to describe a different items behavior;
- Cecilia-Workshop permits to import of Boolean formulae with several calculations (like minimal cut sets, compute after first failure, ...).

This user's manual describes the working environment of Cecilia-Workshop tool.

## 1.2. Hardware architecture

The Cecilia-Workshop tool is available in the following work environments: Any PC platform with, at least:

- RAM: 4 Go Minimum (8 Go recommended);
- Window 7 and 10 (64 bits).

## 1.3. Installation

The installation of software Cecilia-Workshop requires an access to a database.

Cecilia-Workshop manages in the current version:

- either a native access to Oracle database;
- either a native access to MySQL database;
- either a native access to H2 database;
- either a native access to Postgres database.



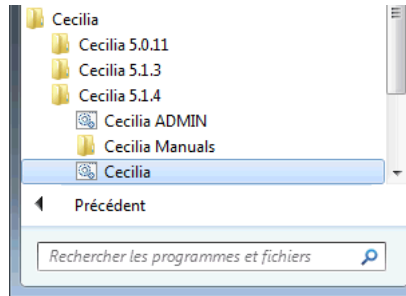
After version 6.0, only H2 and Postgres database will be available.

cf. Installation Guide for more information.

## 2. First step with Cecilia-Workshop

### 2.1. Cecilia-Workshop launch

To start the application, go to **Start menu** and choose the shortcut folder created for the application.



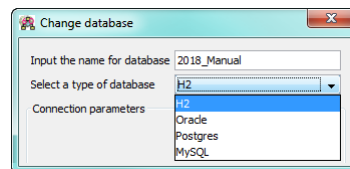
To launch Cecilia-Workshop, you can also double click on  Cecilia.bat located in the Cecilia/Cecilia-6.x.y directory.

Cecilia-Workshop launcher enables to select database for software.

When you launch the application for the first time, the database is not chosen. Before doing anything, you must specify the database that will be used. To do that, the window **Change database** appears.

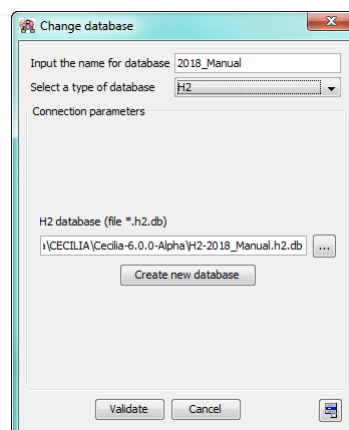
#### 2.1.1. Database selection

The window of database change is the following:



This window allows selecting the database type between:

- **H2**: In this case, you have to specify the H2 file (\*.h2.db)



You can also create a new H2 file specific for Cecilia-Workshop with the button **Create new database**.



- **Oracle:**

The connection parameters are:

- **Host:** Server supporting the database (including the DNS tree structure).
- **Port:** Connection port to the database.
- **Base:** Name of the database instance.
- **Login:** Connection login to the database.
- **Password:** Login password to the database.

For experienced users, it is also possible to directly enter the database connection URL instead of the parameters fields. The syntax of an URL is the following one: **Host:Port@Base**.

- **MySQL:** The connection parameters are identical to the connection parameters for an Oracle database.
- **Postgres:** The connection parameters are identical to the connection parameters for an Oracle database.

-  The Oracle, MySQL and Postgres databases are accessible only if these drivers are available, so the specified JDBC driver packages have to be selected during the installation.
-  After version 6.0, only H2 and Postgres database will be available.

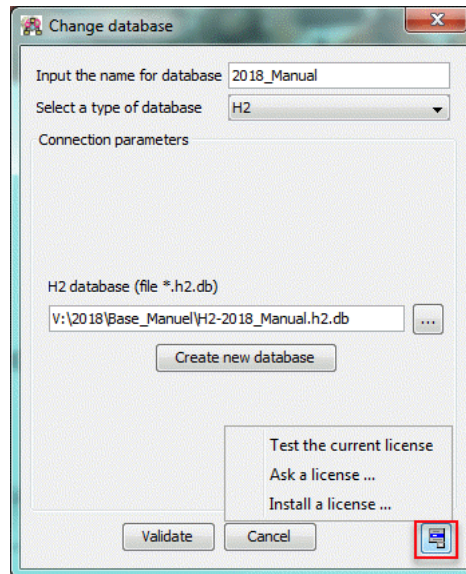
In the validation window (when users click on **Validate** button), connection with database is tested. In this case, error message can appear:

- If parameters in the database are not correct (Inaccessible server, unknown user ...): in this case, check the input and call the informatics support if needed.
- If the database schema is not compatible: it results generally from a database which was not updated.

## 2.1.2. Licenses Management

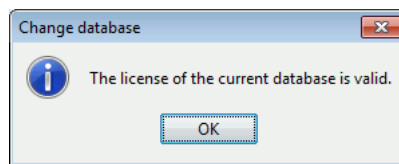
Cecilia-Workshop is protected with licenses. In this case, in order to manage licenses, a button is available at the bottom right in **Change database** window.

The button allows displaying the following menu:

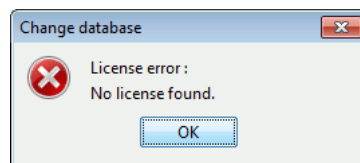


The command **Test the current license** allows verifying that the current license is available

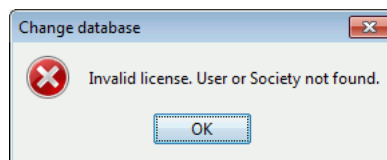
In this case, the following message is displayed:



Otherwise, there is no license for this module



There is an error with the license

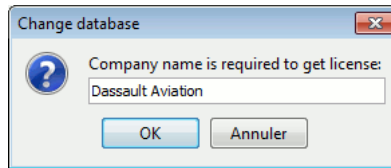


The possible errors are:

- No license found.
- Invalid license.
- Expired license
- Invalid software version
- Invalid license: Format not recognized.
- Host machine does not match host registered in license
- Invalid license: User or society not found.

The command **Ask a license** allows generating the `Key.txt` file for the current module and the selected database. You have to send this file to sales contact to ask or update your license.

A window appears allowing filling your organization name (company, university ...).

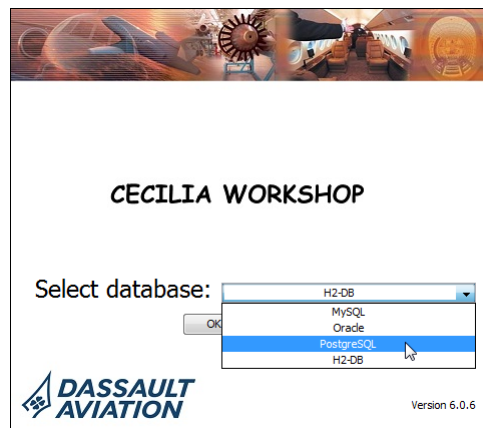


The command **Install a license** allows saving a license from a `license.txt` file. You received this file from your sales contact in return of your license demand.

If a problem appears, refer to the previous errors list (errors during the command "Test the current license").

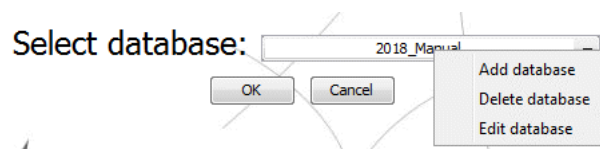
### 2.1.3. Cecilia-Workshop Launching

Once database connection is defined, launching window appears.



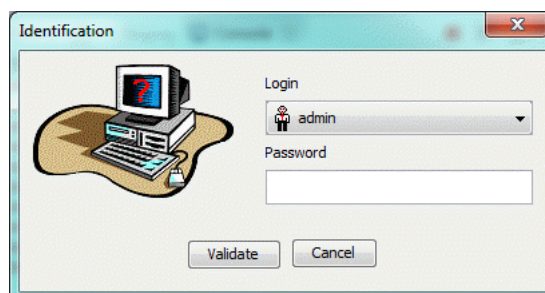
This window permits to selection using a drop-down list which database will be used for Cecilia-Workshop launching.

It is possible to add a new database or to modify the selected database parameters with a right click in the drop-down list.



If there is a license or a connection problem, an error message is displayed in red. In this case license problem must be resolved (cf. Section 2.1.2, "Licenses Management")

After selected the database to consider, the following window is displayed in order to identify user in application.



When you create a new database, there is only one user, the administrator (login admin, password admin).

## 2.2. Work environment

### 2.2.1. Cecilia-Workshop principle

Cecilia-Workshop allows to edit connected technical objects. A **Technical objet** is an item which can be saved/modified/used in the application. It is for example a fault tree, a sub-tree, a named parameter, an event model, a MBSA component, an AMDEC, ...

Technical object can be versionning it means it is possible to create several **versions** of a same technical object.

Technical objects are aranged in the tree structure in **Folders**. A fault tree can be saved in a system folder; and this system folder is arranged in a project folder.

Each Cecilia-Workshop **Tree** allows to create / to edit / to modify a limited number of technical object. For example, **Components** tree allows only the management of a MBSA component whereas the **Project** tree gathers fault tree an MBSA models (and also FMEA, DSF and boolean formulae).

Cecilia-Workshop is a multi-user software so several concepts exist: **user**, **group** of users and **access right**.

A given user belongs to at least one user group. It is possible to define the folder access right (read only, read and write or no access) for each user group. This allows to manage the confidentiality of data, but also the responsibility of each user.

When a user edits a technical object (usually to modify it), the element is temporarily **locked**. Then, another user can only open the technical object in read only. The object is unlocked when the edition (with or without saving) ends.

Each technical object may need, and so **depends** on other technical objects. For example: a fault tree may have one or more events defined through event models.

A link to a technical object can be represented by a **Path** which corresponds to a succession of files, the name of the technical object and its version.

It is then possible to know all the technical objects that depend (directly or indirectly) on a given object.

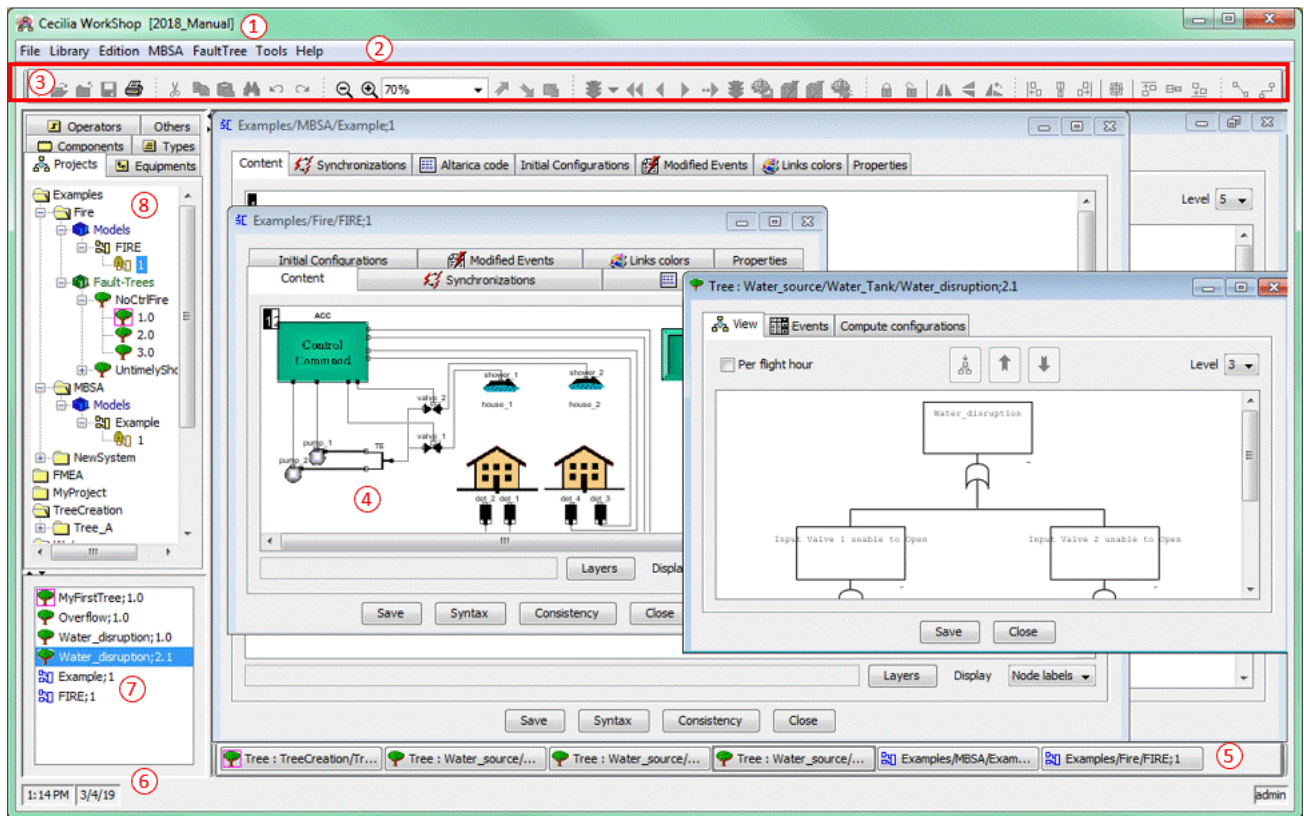
If this object cannot be never modifiable again (because it corresponds to a deliverable for example), it is possible to **freeze** it. This action will also freeze all the technical objects necessary for its definition.

### 2.2.2. Presentation

The Cecilia-Workshop tool work environment is divided into eight parts:

- A title bar (1);
- A menu bar (2) giving access to the tool functions;
- A main tool bar (3) enabling activation of some tool functions;
- A work area (4) in which are displayed the views and windows associated with the opened technical objects;
- The lower bar (5) indicates the objects opened in the work area. When the mouse cursor is positioned on an object in this bar, an information bubble indicates the project or model name;
- An information bar (6) indicating the time and date;
- An area (7) indicating the list of opened technical objects as well as their breakdown; the dimensions of this area can be adjusted by means of the arrows located on the top right part;
- Tree structure to access to each type of technical objects (Project, Equipment, Components, Types, Operators,...) (8); the dimensions of this area can be adjusted by means of the arrows located on the top right part.





### 2.2.3. Menu bar

The menu bar is located at the top of the main window. It gives access to the various Cecilia-Workshop tool functions. It consists of several menus:

- **File** menu: it contains all commands about projects or systems (Create, Open, Save, Close, Freeze, Print and Quit).
- **Library** menu: it contains all commands about tree structure (Add, Remove, Edition, Copy, Cut, Paste and the image manager).
- **Edition** menu: it contains all commands about object edition (Undo, Redo, Remove, Cut, Copy and Paste).
- **MBSA** menu: it contains all commands about the exploitation of the system model in the course of edition (consistency check, simulation, dataflow generation, Java conversion).
- **FaultTree** menu: it contains all functions about the fault tree tools.
- **Tools** menu: it contains all the configuration functions about the Cecilia-Workshop tool (Preferences, pluggins, ...).
- **Helps** menu: it accesses to "About".

### 2.2.4. Main tool bar

The main tool bar is located underneath the menu bar. It can be customized and can be composed of seven tool bars each of which enable a quick activation of the Cecilia-Workshop tool functions.

This tool bar can be customized in the following way: Use the **Tools - Preference ...**. The Preferences window is displayed. Select the **Tools bars** tab and tick the tool bars which will be displayed on the main tool bar on top of the screen.

- **Standard tool bar**: it gives access to the following functions:
  - Open system;
  - Close system;
  - Save system;
  - Print architecture (this command is not specific to an architecture).



- **Edition tool bar**: it gives access to the following functions:



- Cut;
- Copy;
- Paste;
- Search model or component;
- Undo last command;
- Redo last command.



- Design tool bar: it gives access to the following functions:
  - Vertical mirror: reverses the selected object vertically, that is, from top to bottom;
  - Horizontal mirror: reverses the selected object horizontally, that is, from left to right;
  - Rotate 90: rotates the selected object by 90 around the center of the object.



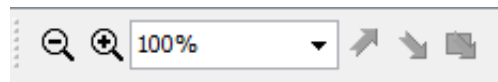
- Links tool bar:
  - In the first part describes the different ways to break the links ;
  - The second part allows to display or not the link direction;
  - The last part configures the thickness of the links.



- Alignment tool bar: the selected elements can be aligned vertically, horizontally, in the middle, etc...



- Navigation tool bar: it zooms in and out on the various graphs displayed in the work area and helps to navigate in the architectures:
  - Backward zoom;
  - Forward zoom;
  - Zoom definition;
  - Up;
  - Down;
  - Down in another view.

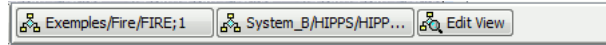


- Simulation tool bar: it dedicated to the simulation of the system model and gives access to:
  - Start simulation;
  - Initialize;
  - Go backward;
  - Go forward;
  - Go next;
  - Save current state as initial state;
  - Stop simulation;
  - Debug information window;
  - Event selection;
  - Modify a state value.

## 2.2.5. Work area

The work area is the main part of the Cecilia-Workshop tool graphic interface. It enables display of the various views associated with the objects or system architectures in current edition.

This area comprises, in its lower part, a bar which displays buttons corresponding to all opened windows.

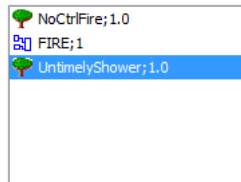


If the number of views is important, a display mechanism associated to the two rectangular buttons located on both sides of the bar allows displaying of the information bar respectively to the right or the left, and thus to reach the buttons previously invisible.

Possible actions:

- A Popup menu accessible by a right click on one of the buttons, can restore, reduce, extend or close the edition window of the associated view.
- The moving of the mouse cursor on one of the icons located on the open objects bar, allows us to have an information bubble indicating the system or the model name in course of edition.

At the bottom left, all documents open are listed:



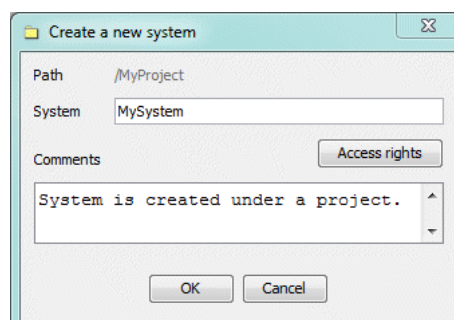
## 2.3. Main functionalities

The aim of Cecilia-Workshop software is to have one tool able to answer at all type of safety studies performed in the aeronautic domain.

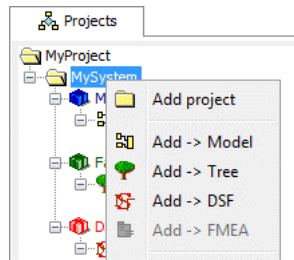
- **Fault trees:** top-down analysis on a system and for a particular undesired event.
- **FMEA:** exhaustive study of all items in order to identify failures modes and their effects.
- **MBSA model:** used for the integration of the functional and dysfunctional aspects. It permits to obtain a different view using for safety studies and for validate the model using step by step simulation. In a second part, these models can be converted in Boolean formulae.
- **Dependable System Faults (DSF)** is an object that can be either a failure rate or a tree depending on user's selection. DSF is especially convenient to allow one sub-system supplier to work considering the failures of the sub-system inputs (power, data...) while the system at the input is still not sufficiently defined to provide a consistent tree of the input failure. This allows to either isolate sub-systems from each other's or interconnect them when they provide the sufficient level of maturity (depending on user's choice).

Cecilia-Workshop is based on an tree structure. The different trees are:

- **Projects:** The top level in this tree structure is the project level which can be divided into several system.

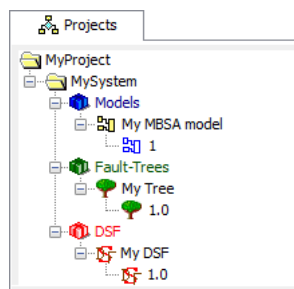


When a project and a system is created, it is possible to choose the approaches (such as Fault trees, MBSA model, DSF, FMEA):

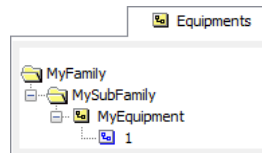


In a same project, the different system can be realized with various approaches and in a same system, several models can be created.

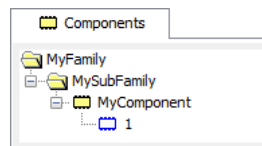
The last level is the model version. All the version in a same model are saved in the system.



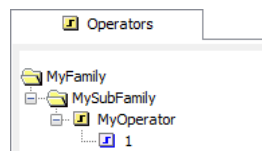
- **Equipment** tree is composed with family and sub-family. The top level is the creation of a family which can include several sub-family and at least the name of the equipment and its version.  
An equipment is only used in MBSA model (see Section 6.2.6, “Equipment management” for settings) and can be made up with components..



- **Components** tree is also composed with family and sub-family. The top level is the creation of a family which can include several sub-family and at least the name of the component and its version.  
A component is only used in MBSA model (see Section 6.2.5, “Component management” for settings).

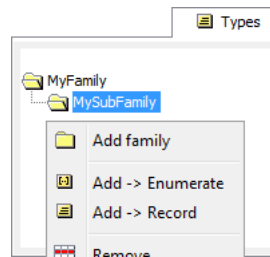


- **Operators** tree is made up with family, sub-family and the the name of the operator and its version. An operator is only used in MBSA model (see Section 6.2.4, “Operator management” for settings).

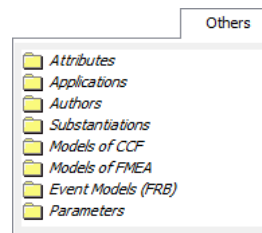


- **Types** has the same tree structure as components it means Family and Sub-family. After that, it is necessary to select an enumerate type or a record type.

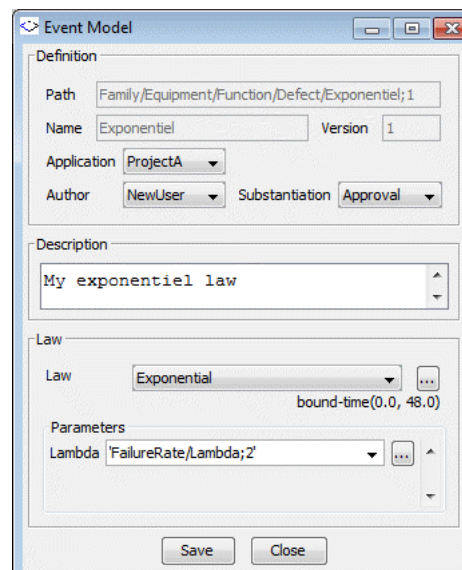
A type is only used in MBSA model (see Section 6.2.3, “Type management” for settings).



- **Others** tree structure is not empty. The following folders are already included:



Attributes, Applications, Authors, Substantiation, models of CCF, models of FMEA and parameters are used in event model (see Section 4.3, “Event model (Failure Rate Base - FRB)” for way to use.)



- **Attributes** are a label that can be affected to any undeveloped event of tree. After define attributes, it is possible to perform specific calculations using these labels to have specifics results for example in a part of an aircraft only (see Section 4.5, “Attributes management”).  
Attributes can be used in terminal events of fault trees, components in MBSA model, in a DSF considered as terminal event or in the definition of the matrix of average risk approach.
- **Applications** allows to specify the project or application to apply the event model. For example, it is possible to specify if the event model is used only for ProjectA or for all project.
- **Authors** represents the person who are in charge of the event model.
- **Substantiations** is used to justify the event model. It is possible to input a date and a comment to precise expert opinion, experience plan, breaking test for example.
- **Models of CCF** defines models of CCF (see Section 5.2, “Common Cause Failure (CCF)”).

**Models of FMEA** defines models of FMEA (see Section 6.5.3, “FMEA generation help”).

- **Event models** permits to ensure consistencies of the failure rates used and to define also the failure laws associated. For each failure rate, a clear description of the failure mode the failure rate is associated to and its traceability are welcome in the Description field (see Section 4.3, “Event model (Failure Rate Base - FRB)”).
- **Parameters** can be used to define some rate / period ... by a generic name. This provides more flexibility in case of change of this parameter value (definitive change or sensibility study for example) (see Section 4.2, “Named parameters management”).

In each folder it is possible to create a new object by creating family and sub-family or directly the object for Applications and authors.

FRB can be used as well in a **fault tree** as in a **MBSA model**.

## 3. Technical object

### 3.1. Technical object edition

The aim of this chapter is to explain the different commands available to create, modify, edit, etc... the technical objects and associated folders.

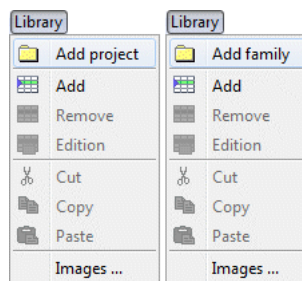
As described in Section 2.3, “Main functionalities” each technical object are creating in a folder.

For **Project** tab the first level is called Project which can contain different system.

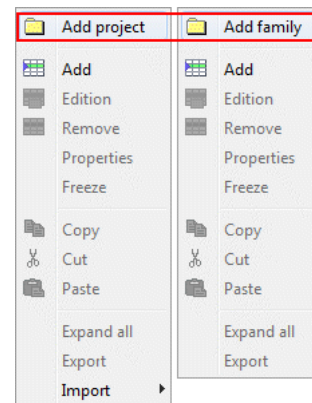
For all the others tab the first level is called Family and can be divided in sub-family.

The first level can be creating using menu bar or contextual menu:

Click on the **Library**



A right click on the left tree allows to create a new family or project by clicking on



In function of the folder or technical object, several options are availables.

**Add project / Add family** This function is used to create the first level of the tree structure.

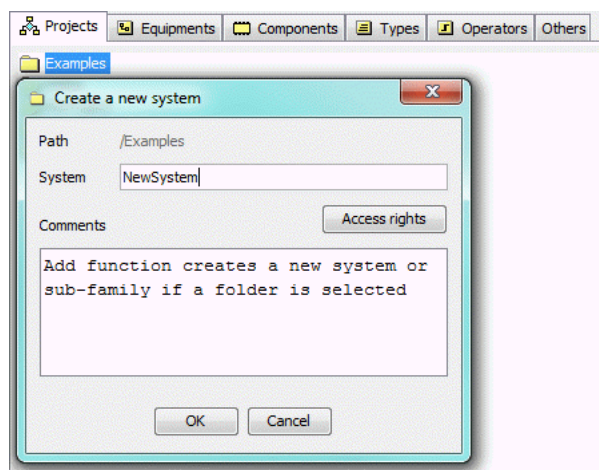


This operation is not available in the **Others** tab in which folders are already created.

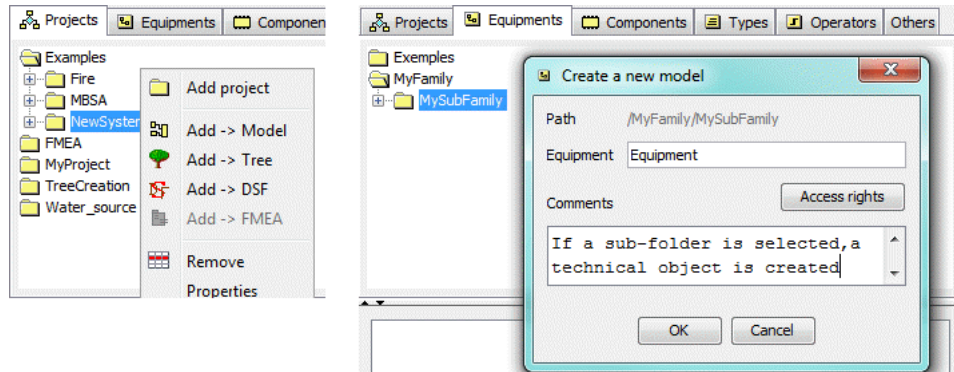
**Add**

This button can be used:

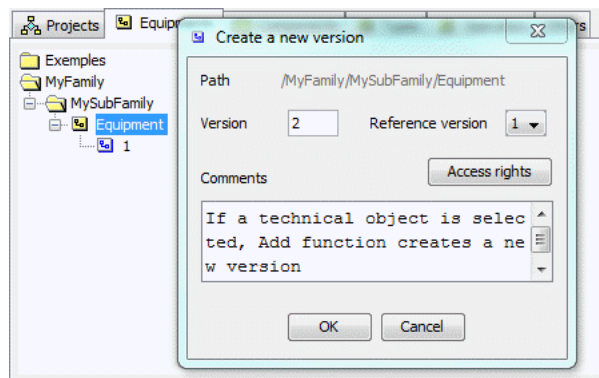
- to create a sub-folder if a folder is selected.



- to create a technical object if a sub-folder is selected.



- to create a new version of the technical object. In this case the reference version can be defined.




## Edition

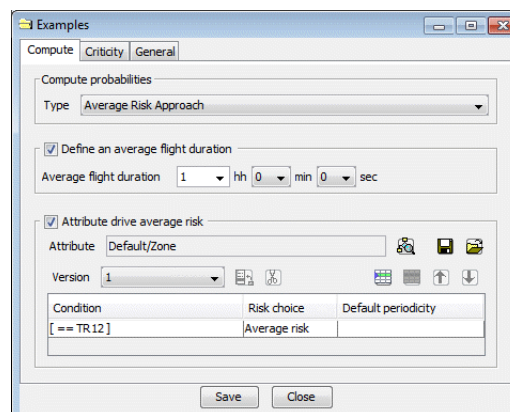
This button allows to open a technical object if a technical object version is selected or to see Project computation properties.

The window is different according to the object. But the behavior is the same with a button **Save** to save the modifications, **Close** to close the object and **Logs** to appear error message.

When **Edition** is selected on a technical object version the aim is to open the object in the work area.

If the window is closed using  a message ask to confirm to save or not the modification.

If this function is used in a Project folder name, a window with compute properties is displayed.



**Compute** tab is described in Project properties.

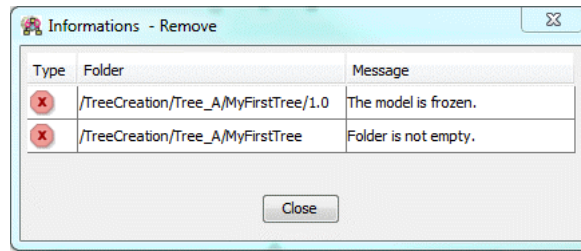
## Remove

To delete selected element.

The Remove command can delete all selected folders, but with the following restrictions:

- a technical object cannot be deleted if it is frozen, locked, used by another technical object;

- a folder can only be deleted if all of these subfolders have been deleted. It is necessary to have write access rights to each folder that will be deleted.



## Properties

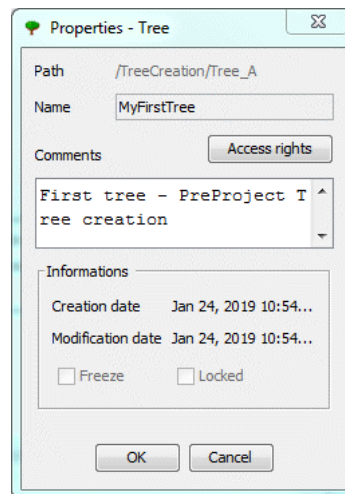
This action allows to see model or project properties.

If a folder is selected, the access rights can be defined (see Section 3.2, “Data access rights management”) and a comment can be added.

If a version number is selected, it is possible to define access right and add a comment to the technical object.



Properties can be used to rename the technical object or a folder.



## Freeze

With this functionality it is possible to freeze models.



This operation is very risky and must be performed with extreme care, because it doesn't allow later modifications on the stilled type version.



This "Frozen" attribute can be also visualized via the model properties editor (It can be obtained by the **Properties** command of the contextual menu or from the general menu File.



**Copy**

To copy a technical object or a folder.



**Cut**

To cut a folder.



It is not allowed to cut a version.



**Paste**

To paste a technical object or a folder.

A copy / paste on a technical object version is equivalent to an Add function in the reference version. A copy / paste on a folder creates a copy of the folder and all the sub-folders.

## Expand all

To expand all the tree structure.

## Export

To export in a xml file technical object and their depends object.

## Import

To import technical object

## Refresh

To refresh the technical object



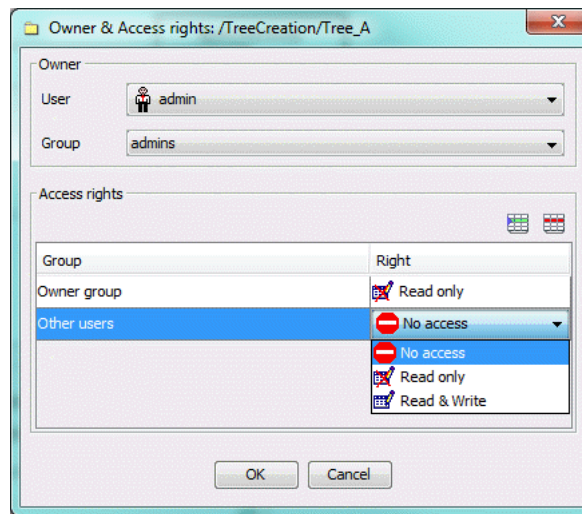
### 3.2. Data access rights management

In the database configuration mode Cecilia-Workshop allows the access right management to data (generic models in library and System architectures) thanks to dependencies and visibility attributes defined on each Cecilia-Workshop object.




This window is accessible by a right click on the element in **Properties / Access right**.

The top part called **Owner** presents the users declared in the database and in the right list are presented all the groups that contains the user selected. In order to set the owner and group owner, the user first select a user in list and then a group in the other list.

The lower part called **Access rights** shows the access rights set to the group owner of the node on the left part and to the other groups on the right part. To change an access right, the user simply click on the corresponding radio button. If the user click on the **Default** button, the access rights are set by default to 'read only' for the owner group and to 'No access' for the other groups.



Project access rights defines the user and the group. For each group, it is possible to define the right for the model:

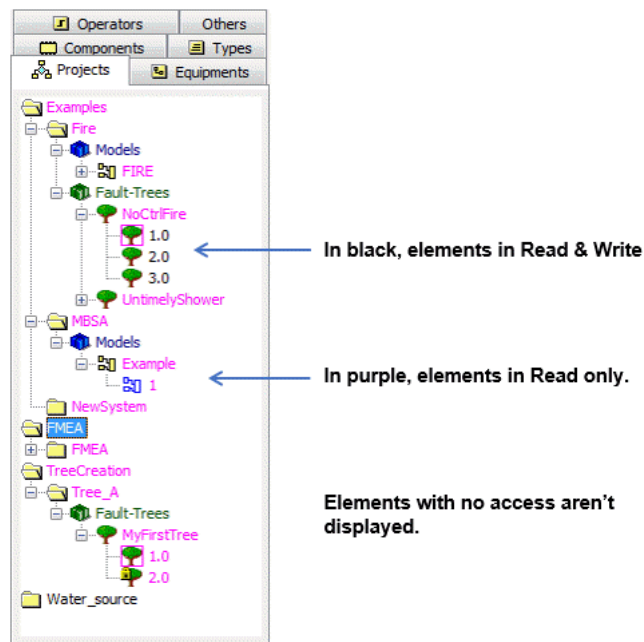
-  : selected group has no access.
-  : selected group can only read the model.
-  : selected group can read and write the model.

During the creation of a Cecilia object, the access rights defined by default are:

- Read Access Right for users of the workgroup to which this object is reattached (only the owner of this object has an Read/Write Access Right);
- No access for the users of the other workgroups.

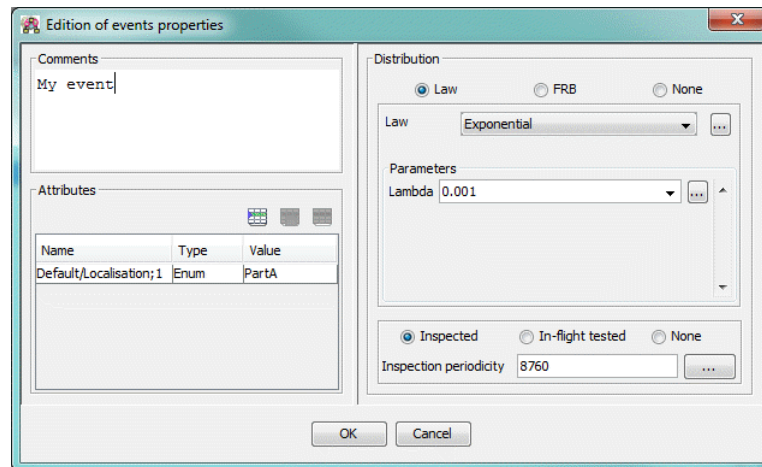
Only the administrator of Cecilia database or the owner of a given object can modify its attributes of access rights by using the Editor of properties.

According to the **Access right** attributes by the administrator to the user, the color of the tree structure gives information of the access right.

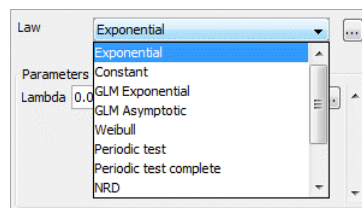


## 4. Common approach

The aim of this chapter is to describe the common part to edit events.



### 4.1. Law



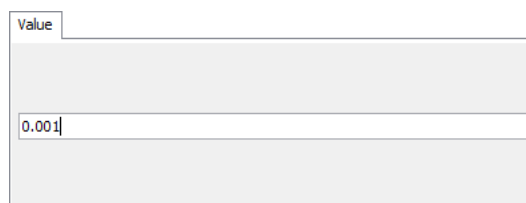
First, the event is link to a Law. When law radiobutton is check, it is possible to chose the law from all the available laws in the drop-down list located in the upper section. Many laws are available and you can find the meaning for each law under Section 1, “Description of the laws” for the description of all the law.)

The button allows to specify a coefficient modifier for the law (see Section 2, “Modifier coefficient of the law”)

After, in function of the law , user sets the parameters for the law in the lower part of the window. For each parameter a numerical value can be entered.

With parameter can be defined by:

- The value can be a simple numerical value (example: 0.001, 2.0E-02, etc.).



- The value can be linked to a parameter.



- The value can be defined by an uncertainties law (see Section 3, “Uncertainties on the parameters”) (example: uniform 1.000E-03 3.000E-03, lognormal 1.000E-03 3.000E-03, etc.)

- The value can be defined using a FMEA elements.

## 4.2. Named parameters management

The named parameter manager panel allows the user to create named parameters reusable as many as he wants, on the laws defined on component events. If the value of a named parameter changes, all of the laws that use it are automatically impacted.

The named parameters are located in the **Others** tab and can be created by a right click on the **Parameters** folder.

The named parameter properties are the following ones:

- Name** of the named parameter
- Version** of the named parameter
- Description** of the named parameter
- Type** of the named parameter
- Value** or **Law** or **FMEA** tab: several choice of the value of the named parameter:

- The value can be a simple numerical value (example: 0.001, 2.0E-02, etc.).

- The value can be defined by an uncertainties law (see Section 3, “Uncertainties on the parameters”) (example: uniform 1.000E-03 3.000E-03, lognormal 1.000E-03 3.000E-03, etc.)

- The value can be defined using a FMEA elements.

To the named parameter types corresponds a domain of definition for the values. The named parameter type can be of the following values:

- **Probability:** the value must be between 0 and 1 included
- **Rate:** the value must be a real greater than or equal to 0
- **Duration:** the value must be a real strictly greater than 0
- **Factor:** the value must be a real greater than or equal to 0
- **Boolean:** the value must be equal to 0 or to 1

When all the fields are filled in the user clicks on the **Save ase** button to confirm the actions or on the **Close** button to cancel the actions.

## 4.3. Event model (Failure Rate Base - FRB)

Cecilia-Workshop provides a Failure Rate Base to ensure consistencies of the failure rates used.

The event model library allows the user to create generic event definitions that can be used by the events defined on component models.

The event models contain generic definition and can be used by several data on Cecilia-Workshop. The event models are referenced by the events defined on components on Cecilia-Workshop. The way used to define these associations is described in the Section 4.6, “Event settings”.

The FRB are located in the Others tab and can be created by a right click on the **Event model** folder.

The first field named **Path** contains the path on where the new node will be created.

The field **Name** contains the name of the new node.

Each event model is affected to an application, an author and if necessary a substantiation. These data are defined under corresponding static nodes in the Other tab (see Section 2.3, “Main functionalities”).

The bellow part of the panel is dedicated to the definition of the event model fault law.


An event model is define by a law and its parameter, so first, it is necessary to choice a law and its parameter in the combo box. The law and the options for the parameter are described in Section 4.1, “Law”.

## 4.4. Distribution

A distribution it's a law or a FRB with the type of inspection. There are 3 different type of inspection:

- **Inspected** means Dormant.The dormancy period is to be defined in the **Inspection periodicity** field.
- **In-flight tested** is related to a dormant failures that is inspected event within the time of a flight (dormant event with a dormancy duration lower than flight time). In this case it is necessary to fill in the **Exposure time**.
- **None**: the event is not inspected.



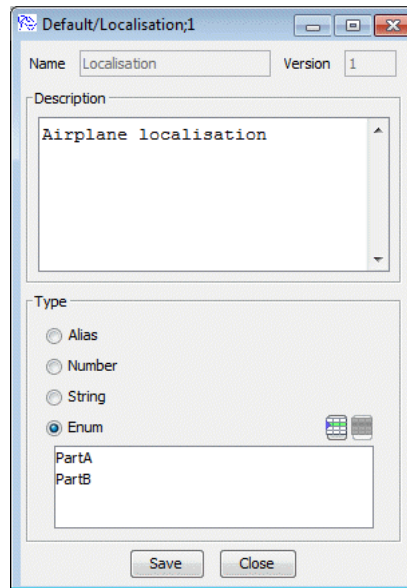
The button  allows to use a numerical value or a named parameter defined as delay type.

## 4.5. Attributes management

The attribute manager panel allows the user to create attributes that are used to qualify events defined on system components.

The attributes are located in the **Others** tab. By a right click on the **Attributes** folder the following actions are available:

To create a new attribute it is first necessary to **Add** in attribute folder in Other tab. A double click on the version number (or used **Edit** button) permits to edit the attribute.





The attribute properties are the following ones:

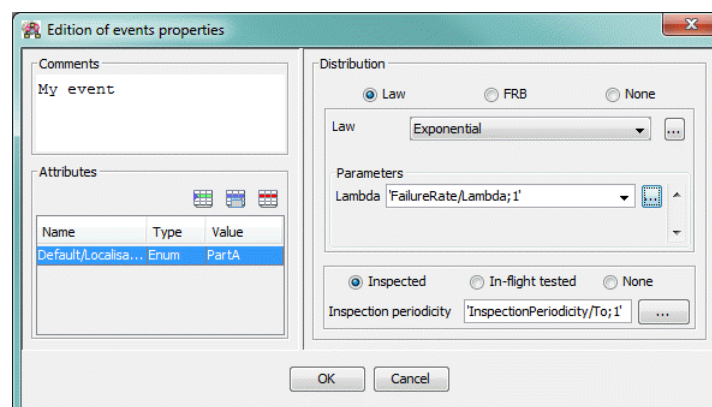
- **Name** of the attribute
- **Description** of the attribute
- **Type** of the attribute

The attribute type can be of the following values:

- **Alias**: This kind of attribute is just a tag that can be set or not on an event;
- **Number**: This kind of attribute is used to affect an integer value to the events;
- **String**: This kind of attribute is used to affect a free text to the events;
- **Enumeration**: This kind of attribute is used to affect a value from a pre-defined list to the events.

If the attribute is an enumeration, the button  (resp ) permits to add (resp delete) the event list.

## 4.6. Event settings






All the information described in this chapter can be used in the edition of an event. These properties are the same for a terminal event in a fault tree or in a MBSA element.

This window is divided into 3 parts: Comments, Attributes and Distribution.

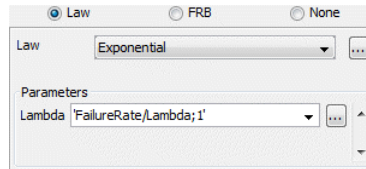
1. **Comments** fields to input for example a description of the event.


2. **Attribute** can be added to the event using the following buttons:

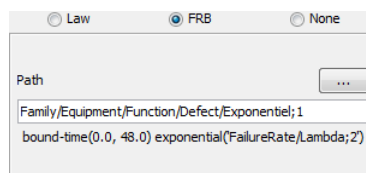
-  add a new attribute;
-  edit an attribute;
-  delete the selected attribute.

3. In the **Distribution**, the event can be defined:

- by a law selected in the law available in Cecilia-Workshop and its parameter as described in Section 4.1, “Law”.



- by a FRB as detailed in Section 4.3, “Event model (Failure Rate Base - FRB)” in this case it is necessary to fill in the path field using .




A description of the FRB is given under the path.

- by nothing.

Finally, the last part concerned the type of inspection of the event. The radiobutton allows to consider the event as **Inspected** with a specific **Inspection periodicity**, **In-flight tested** with an **exposure time** or **None** inspected (see Section 4.4, “Distribution” for details)



A parameter can be used for the duration.



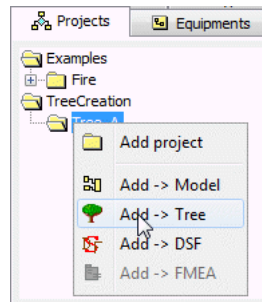
## 5. Boolean models

### 5.1. Fault Tree

#### 5.1.1. Fault Trees edition

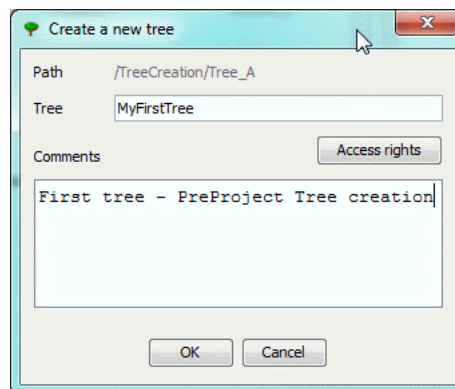
A Fault tree is a type of model saved in the **Projects** tab.

To create a tree, it is first of all necessary to create a Project, a System (see Section 3, “Technical object”) and then the fault tree by a right mouse click.



All the functions described in Section 3.1, “Technical object edition” can be used in a Fault tree in particular **Properties** to rename a Fault Tree, **Copy**, **Paste**, **Cut**, **Remove**, etc... .

The following window appears to inform the name and the revision of the fault tree.

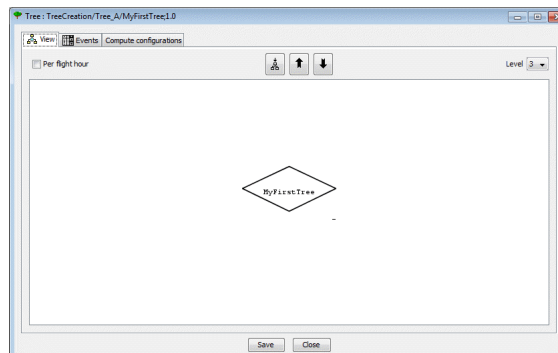


When the fault tree is created, it can be opened by double click or via the contextual menu and select **Edition**.

Creation of a new tree is performed:

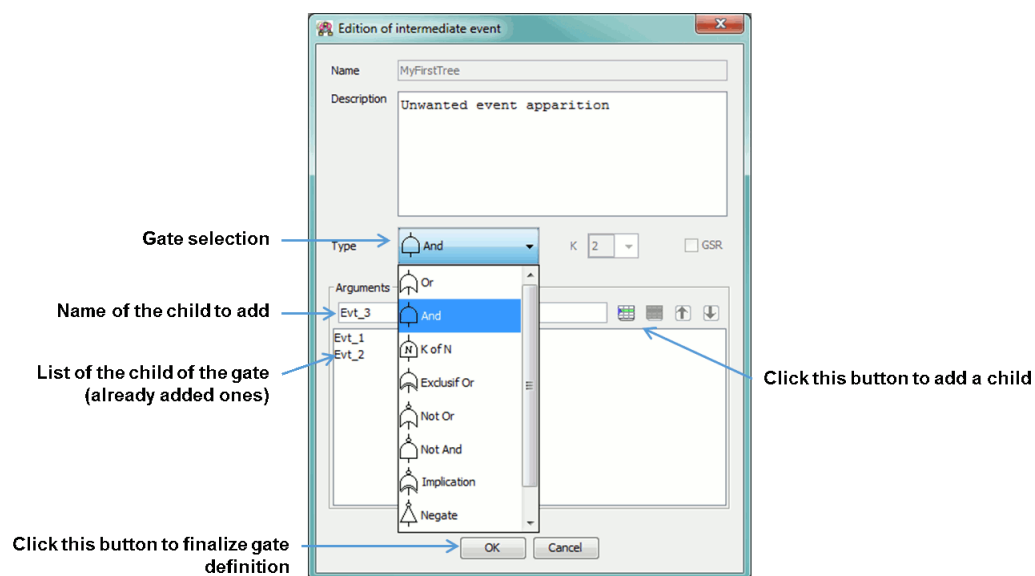
- By typing the tree name in the **Tree name** field;
- By modifying, if necessary, the default access rights (**Access rights** tab details are given in Section 3.2, “Data access rights management”);
- By typing, if necessary, a general comment (**Comment** tab) for the fault tree;
- By validating the creation of the tree by clicking on the **OK** button (this button is active only if the system name has been typed and if it does not already exist) or click on the **Cancel** button to quit the Create tree window.

When the tree has been created, it is automatically inserted in the document manager (in an alphabetical order) as well as its version.







The top event is created. Now it is possible to create the different events by a double click on the top event.




The following window appears to create the tree:









Following icons can be used to:

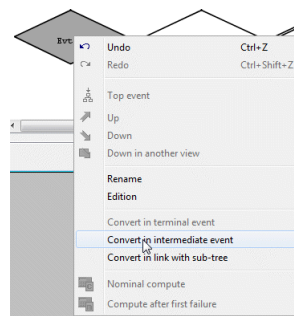
-  add a new argument;
-  delete the selected argument;
-  get up the selected argument;
-  get down the selected argument.

The meaning of the gate is the following:

	<b>OR</b> The output occurs if at least one of the input occurs.
	<b>AND</b> The output occurs only if all inputs occur.
	<b>K out of N</b> The output occurs if at least K of the N input occur ( $K < N$ ).

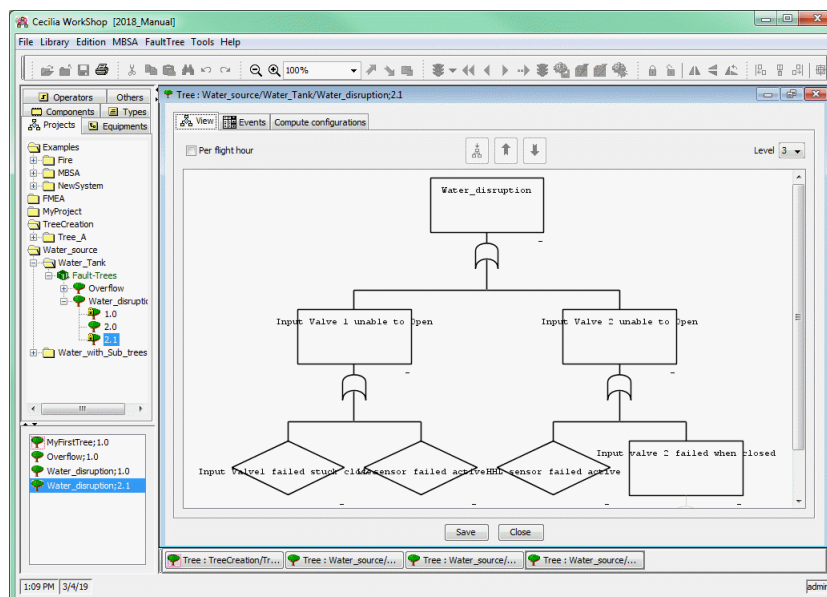
	<b>Exclusive OR</b> The output occurs if one of the input occurs.
	<b>Not OR</b> The output occurs if none of the input occurs.
	<b>Not And</b> The output occurs if at least one of the input is not occurred.
	<b>XNOR</b> The output occurs if both of the inputs to the gate are the same.
	<b>NOT GATE</b> The output occurs if the input is not occurred. The exit logic state is the contrary of the entry logic state.
	<b>If... Then... Else...</b> The output is worth the value of the event 'then' if the event 'if' occurs and is worth the value of the event 'then' otherwise.

By default, the children are terminal event. A terminal event can be converted to a gate (intermediate event) by a right click on it and select **Convert in intermediate event**.






Using the same way, it is possible to convert a gate to a terminal event by selecting **Convert in terminal event**.

Events settings are described in Section 5.1.2, “Events edition”



The graphical representation of a tree as it is displayed in the first tab of the window. The following button can be used to:

- : to go back directly to the top event of the tree;
- : to go down of the tree;
- : to go up of the tree.
- The graph level can be adjusted by means of the Level field.

Results can be reduced to one flight hour using the dedicated check box (see Section 5.1.7, “Reduce to flight duration”).

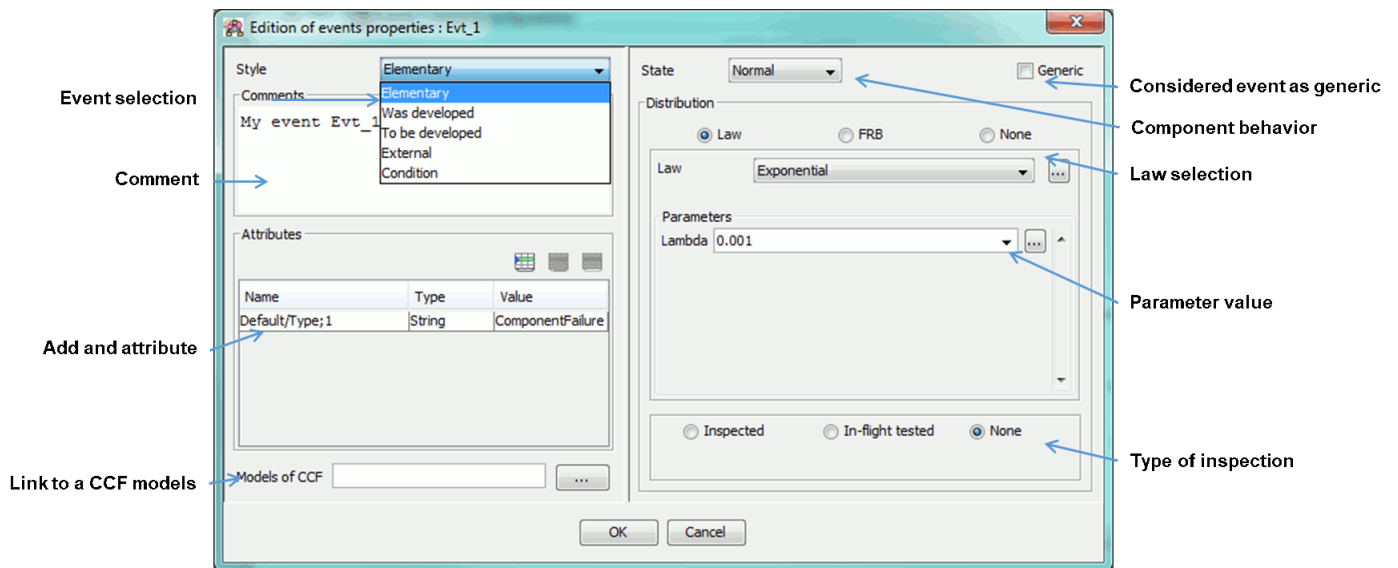
To select several events, click on these events with the mouse left button while depressing the SHIFT key. The last event selected becomes the current event. Click on an event with the left button while depressing the SHIFT key to invalidate it. In this case, if the event was the current event, the last selected event becomes the current event.

The zoom options, located on the tool bar can be used for this representation type.

### 5.1.2. Events edition

When fault tree is created, users can parameters events with a double click on it.

Terminal event can be also configured by a double click on the event.



**i** Common event settings (it means **Comments**, **Attributes**, and **Distribution** are described in Section 4.6, “Event settings”).

There are 5 different styles of terminal event:

	<b>Elementary event:</b> used for events which could be analysed in more details but for which it is not useful with regards to the performed study.
	<b>Was developed event:</b> used to indicate the event will be link to a sub-tree (see Section 5.1.9, “Sub-Tree”).
	<b>To be developed event:</b> for components failures which must be developed in more detail before completing the study.
	<b>External event:</b> event which has occured or will occur with certainly according to the configuration (State value equal to true).
	<b>Condition event:</b> event that is a condition of occurence of another event (conditions are used with If Then Else gate).

**Models of CCF** allows to use CCF for this event using the

**State** tab permits to select the behavior of the component. 3 cases are possible:

- **Normal** means following the law chosen by the user;
- **True** to indicate the event is always true.
- **False** in this case the event is false.

**i** If one (or more) event(s) which is (are) visible on the graph is (are) of the "always true" or "always false" type its (their) representation color will be the color defined in the options for this type of event (color selection for true event or color selection for false event). Furthermore, if some events cannot be modified, their representation color will be the color defined in the options for this event type (color for events which cannot be modified).

**Generic event** check box indicates if this fault tree is generic (cf. Section 5.1.9, “Sub-Tree”).

The last part is to indicate the **Distribution** for the event it means to define event failure law and exposition times is described in Section 4.4, “Distribution”.

The event representation is different according to the type of inspection.

Event type Type of inspections	Elementary	Was developed	To be developed	External	Condition
None					
Inspected					
In-flight tested					

To validate all modification it is necessary to click on **Validate** button.

## 5.1.3. List of events

Each tree opened in Cecilia-Workshop has a representation giving the list of its events. The aim is to display ll the element in a table form and the link between us to create the boolean formulae.

This mode is given in the second tab of the Fault Tree called Events.

Events	Definition	Comments	Given laws	State
HHL sensor failed active	-		exponential 1.9000E-05	Normal
HL sensor failed inactive	-		exponential 3.8000E-05	Normal
Input Valve 1 failed stuck open	-		exponential 5.7000E-05	Normal
Input Valve 1 unable to Open	('Input Valve 1 failed stuck closed'   'L...		-	-
Input Valve 2 failed stuck closed	-		exponential 5.7000E-05	Normal
Input Valve 2 unable to Open	('HHL sensor failed active'   'Input va...		-	-
Input Valve 2 was commanded to closure	('Input Valve 1 failed stuck open'   'H...		-	-
Input Valve 1 failed stuck closed	-		exponential 5.7000E-05	Normal
Input valve 2 failed when closed	('Input Valve 2 failed stuck closed' & '...		-	-
LL sensor failed active	-		exponential 1.9000E-05	Normal
LL sensor failed inactive	-		exponential 3.8000E-05	Normal
Water_disruption	('Input Valve 1 unable to Open'   'Inp...		-	-

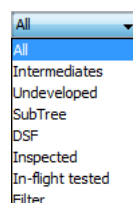
The table represents all the element in the Fault Tree and can be used to create or modify a Fault Tree.

A double click on the **Events** column allows to edit or rename the selected event.

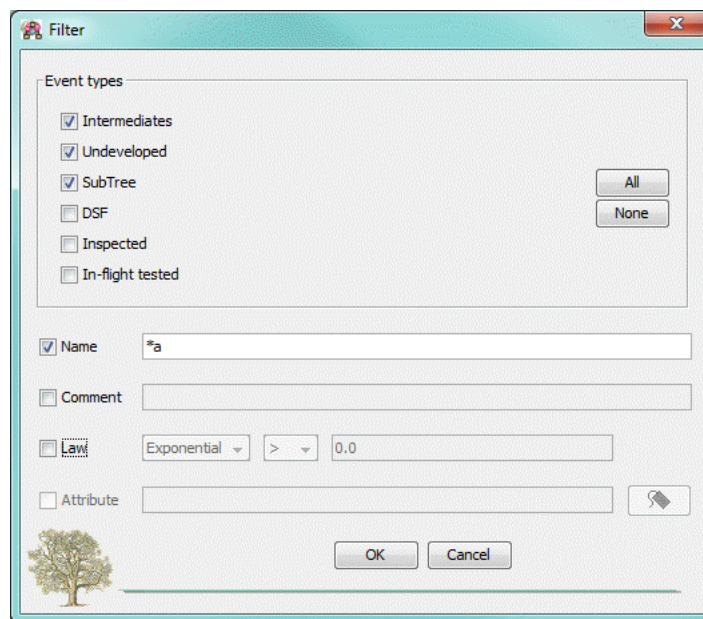
A double click on the others column displays the event edition window to configure the event.

A right click on an event gives acces of the contextual menu and allows for example to convert a terminal event to an intermediate event and reciprocally.


- allows to select all the elements displaysin the table.
- The user can filter the listed events by choosing **Filter** in the selector menu. At this time, the Filter tab is selected and appears at the top. When filtered events are displayed, it is possible to create one's own filter (**Filter** selected).



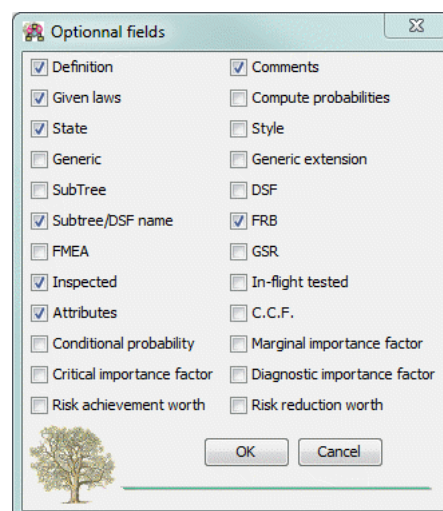
When Filter is selected the following window appears to precise the type of filter wanted.



This tab shows two areas: event type and search string. The first area offers to the user the possibility to close up the filtering on a certain type of events (Intermediate, undeveloped, Sub-Tree, DSF, Inspected or Inflight tested). The second area allows the user to apply the filter on events whose name or comment complies with the search criterion specified in the corresponding text field. The string defining the search criterion can contain as many \* characters as desired (for example: \*abc\*def\*)

When a filter is created,  is activated to display the filter window.

- Results can be reduced to one flight hour using the dedicated check box (see Section 5.1.7, “Reduce to flight duration”).
- The columns displays can be configured (**Fields** tab):



With the table, users have access to the following functionalities:

- Selection of events indicated in the list via a selector (All, intermediate, undeveloped, Sub-tree, Inspected, Inflight tested, Filter or attribute filter);
- Display of the number of displayed events depending on the number of total events;
- The width column can be adjusted by positioning the mouse cursor on the header of the upper table until the cursor is visible; move the mouse with the left button depressed



The column width is simultaneously modified in both tables);

- The columns can be shifted (except the first one) by clicking on the column header of the upper table, then by shifting it to the desired position

As for the graphical representation window, the events list enables selection of one or more tree events:

- A single click in the list, invalidate all the events previously selected and chooses as current event the one which has been clicked;
- To select a second event, click on the event with the mouse left button while pressing the CTRL key. This event thus becomes the current event;
- To remove an event from the list of selected events, click on it while pressing the shift key. If this event was the current event, the last selected event becomes the current event;
- To select a set of contiguous events, click on the first event, then on the last one while pressing the SHIFT key.



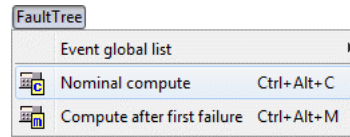
If one or several selected events are of the "always true" or "always false" type, its selection color will be the color defined in the options for this type of event (color selection for true event or color selection for false event).



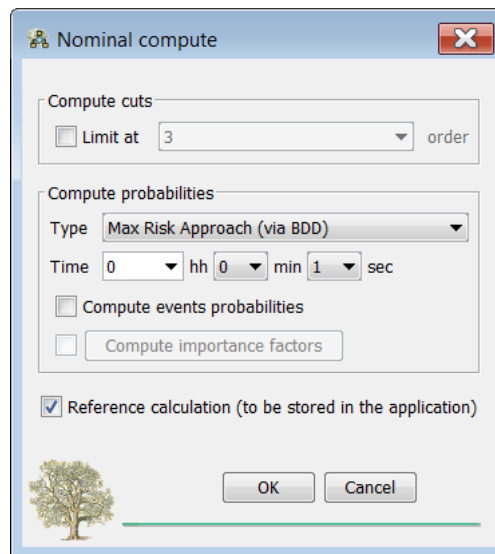
### 5.1.4. Minimal cut set calculations (MCS)

The Fault Tree tool is coupled to the Cecilia-Workshop calculation engine for the computation of tree minimal cut set.

The calculation are accesible using the menu **FaultTree-Nominal compute**

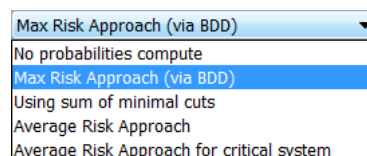


The following window appears in order to set-up the calculation:



If necessary it is possible to limit calculation to the selected order by checking **Limit at** option and choose the cut set maximal order.

Several type of calculations can be performed.



- **No probabilities compute:** to have results without probabilities;
- **Max Risk Approach (via BDD)**
- **Using sum of minimal cuts:** Max Risk Approach (via minimal cuts)
- **Average risk approach**
- **Average risk approach for critical system**

For more informations about these types of computation, see Appendix B, *Probabilities computations*.



Default computation can be defined in project level. In this case, with a right click on the name project select **Edition** menu. Then, computation can be define in the **Compute** tab.

Condition	Risk choice	Default periodicity
[ == PBIT ]	Maximum risk	
[ == Pre Flight ]	Average risk	'InspectionPeriodicity/To;1'

In this tab default calculation for all the project are defined. User can select the average flight duration (used in reduce to a flight duration detailed in Section 5.1.7, “Reduce to flight duration” ) and the type of computation.

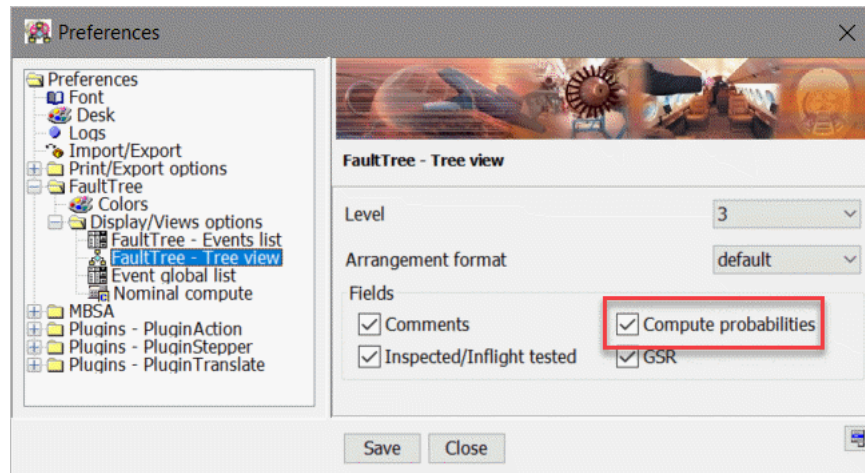
In this case, default computation appears in bold with the **[prj]** at the end.

It is however possible to perform other type of calculations.

In bold, default computation fill in project properties

If necessary, a duration (in hours) for the time can be typed, in case when some probability laws for tree basic events depend on the time factor (i.e. Exponential law);

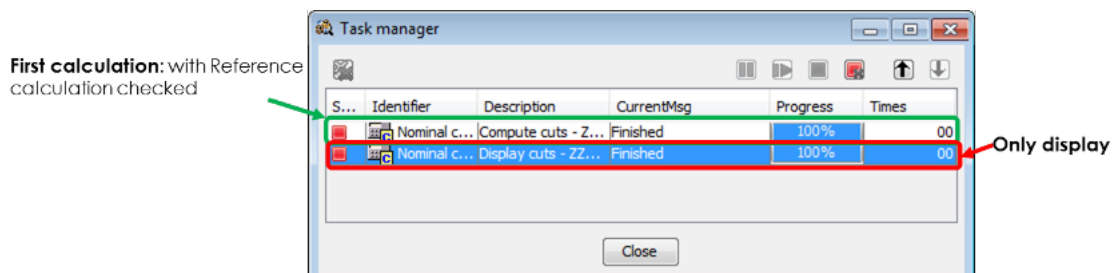
Check **Compute events probabilities** when Max Risk Approach (via BDD) is selected permits to display the events probability directly in the tree. For that, in the **Tools - Preferences - Fault Tree - Tree View**



menu and check **Compute Probabilities**.

To **Compute importance factors** it is necessary used Aralia 4 for BDD engine. The different importance factors are detailed in Appendix C, *Importance factors*

**Reference calculation** stored the calculation in the databased. If this option is checked, if there are any modifications on the model, user can display the results using the compute windows.

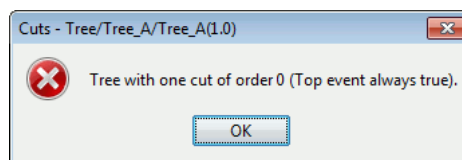


Click on the **OK** button to launch the calculations.

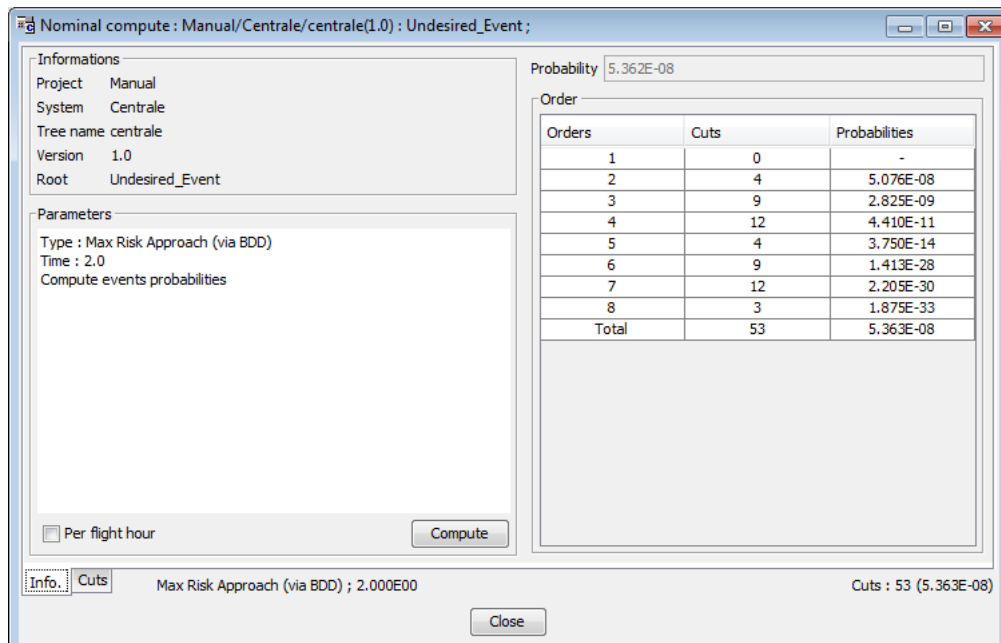
**i** According to the **Users preferences**, computation is launch in local or distributedand using Aralia4 or CeciliaBDD (see Grid computing)

When computing is over, two cases are to be considered:

- The tree does not contain cut set lower or equal to the order specified in the computing window;



- Computing has been successful. Select the criteria tab to gain access to extraction criteria for the resulting minimal cut set.

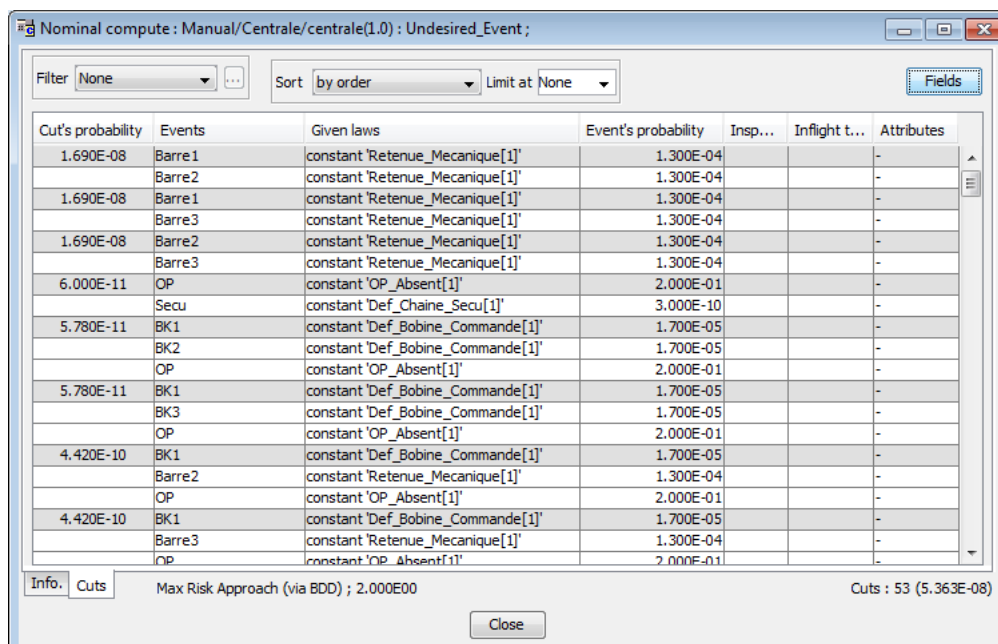


This window is composed with 3 different parts:

- **Information tab** which summarizes the Tree information;
- **Parameters** for the type of calculations performed and time needed.
- In the **Order** part all minimal cuts set are described with the order, the number of cut set of each order and the probabilities.


Results can be reduced to one flight hour using the dedicated check box (see Section 5.1.7, “Reduce to flight duration”).

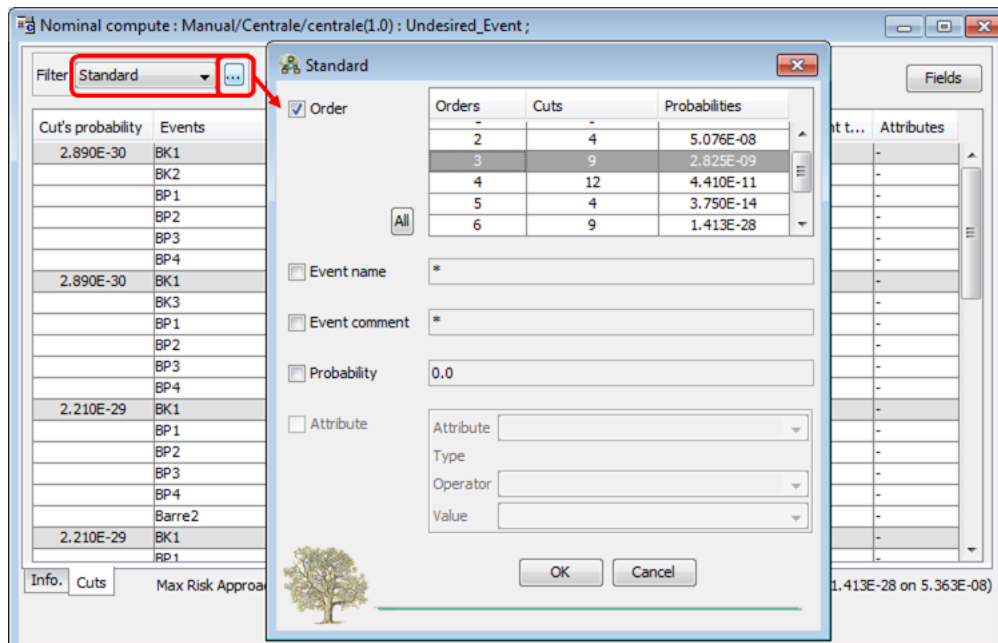
The Cut list window is then positioned on the List tab.



The minimal cuts set can be filtered and/or sorted using different options

## Standard filter

A click on  to select the filter.



It is possible to filter by:

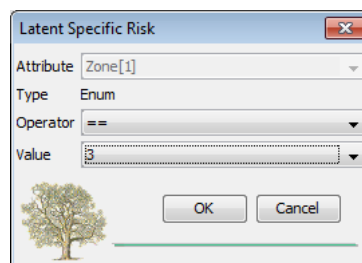
- **Order:** for example, user can appear only cuts with an order 3;
- **Event name or comment:** to display only the minimal cut set containing the events whose names/comment comply with the filter defined in the Enter your choice field;
- **Probability:** to display only the minimal cut set whose probability of occurrence is lower or equal to the value defined in the Enter your choice field;
- **Attribute:** to display only events with specific attributes.

### Relative weight filter

It is possible to filter by **relative weight**. The aim is to display only the minimal cut set in decreasing order of probability which the sum is strictly less than the failure condition probability percentage (value defined in the Enter your choice field).

### Latent specific risk filter

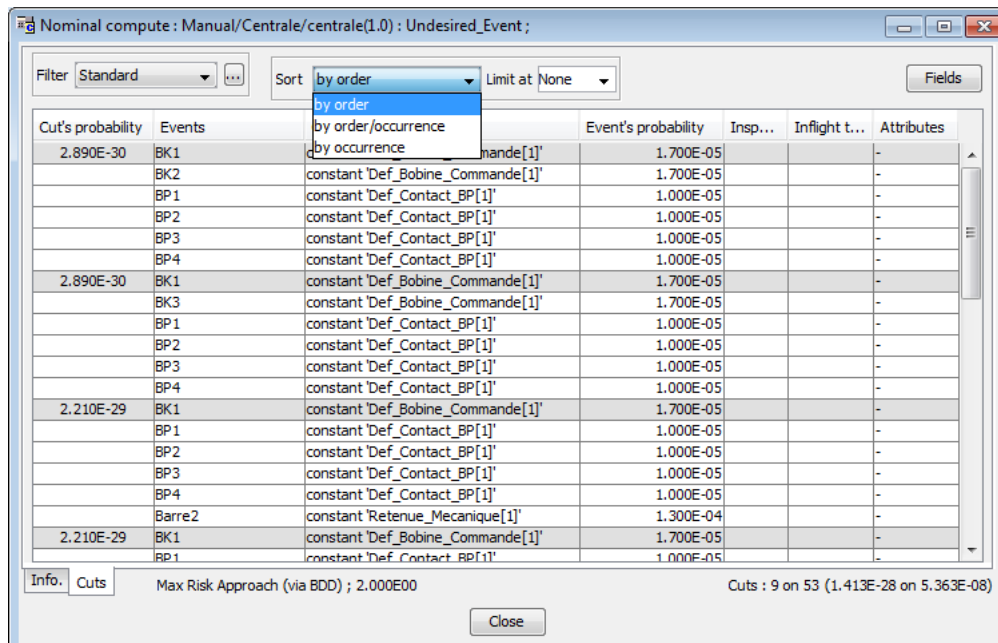
This filter can be used only if there are attributes in the tree.



The minimal cut set are filter using the following rules: a minimal cuts with an order N is kept only if:

- N-1 events validates the attribute criteria defined by the user in **Latent Specific Risk** window;
- 1 event doesn't validate the attribute criteria.

## Sort

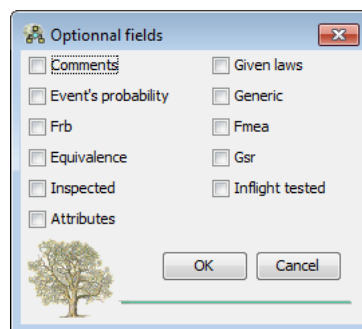


3 different sorts are proposed:

- by **order** to display cut set in an increasing order;
- by **order/occurrence** to display cut set in an order and increasing occurrence for cuts set with the same order;
- by **occurrence** to display cut set in an increasing occurrence;

It is also possible to **limit** the display of products, either with digital way. For example **limit at 10** reveal the 10 first ones).

The user can add some additional data relative to the undeveloped events by using the **Fields** tab.



### 5.1.5. Common attribute analysis (CAA)

The **common attribute analysis (CAA)** treatment allows to perform verification when attributes are correctly filled for each event.

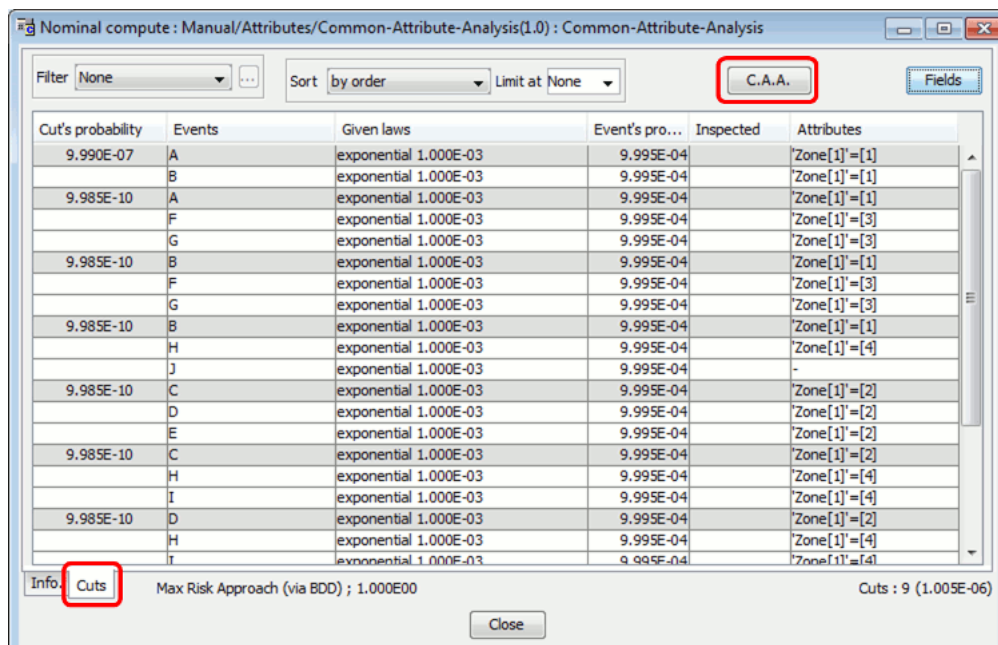
The aim is to analyze the minimal cut sets not with an event point of view but according to Attributes.

For example the following events are linked with the following zones:

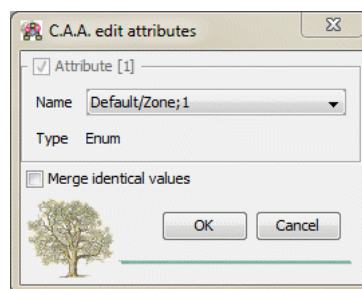
Events and Attributes links	
A	Zone 1
B	Zone 1
C	Zone 2
D	Zone 2
E	Zone 2
F	Zone 3
G	Zone 3
H	Zone 4
I	Zone 4
J	-

Associated MCS		
Order	Cuts	Attribute cuts
2	{A, B}	{Z1, Z1}
3	{C, D, E}	{Z2, Z2, Z2}
	{A, F, G}	{Z1, Z3, Z3}
	{B, F, G}	{Z1, Z3, Z3}
	{C, H, I}	{Z2, Z4, Z4}
	{D, H, I}	{Z2, Z4, Z4}
	{B, H, J}	{Z1, Z4, -}
4	{E, F, H, I}	{Z2, Z3, Z4, Z4}
	{E, G, H, I}	{Z2, Z3, Z4, Z4}

After a MCS calculations, selection **C.A.A** in the **Cuts** tab.



The following window appears in order to select the attribute to perform C.A.A. calculations.



If **Merge identical values** is checked, for a minimal cuts set if 2 events are in the same attribute only 1 attribute is mentioned.



The result of the computation leads to this window:

C.A.A. : TreeCreation/Tree\_A/Tree\_MCS\_Attribut(1.0) : Common-Attribute-Analysis

Attribute cuts	Nb.	Probability
MCS (origin)	9	1.005E-06
1 Default/Zone;1 : {Zone 1;Zone 1}	1	9.990E-07
1 Default/Zone;1 : {Zone 1;Zone 3;Zone 3}	2	1.997E-09
1 Default/Zone;1 : {Zone 1;Zone 4;#Undef}	1	9.985E-10
1 Default/Zone;1 : {Zone 2;Zone 2;Zone 2}	1	9.985E-10
1 Default/Zone;1 : {Zone 2;Zone 4;Zone 4}	2	1.997E-09
1 Default/Zone;1 : {Zone 2;Zone 3;Zone 4;Zone 4}	2	1.996E-12

Orders	Cuts	Probabilities
1	0	-
2	1	9.990E-07
3	6	5.991E-09
4	2	1.996E-12
Total	9	1.005E-06

Sort by order

Cut's proba...	Events	Comments	Given laws	FRB	Inspected	Attributes
9.990E-07	A		exponential 1.000E-03	-		Default/Zone;1'=...
	B		exponential 1.000E-03	-		Default/Zone;1'=...
9.985E-10	A		exponential 1.000E-03	-		Default/Zone;1'=...
	F		exponential 1.000E-03	-		Default/Zone;1'=...
	G		exponential 1.000E-03	-		Default/Zone;1'=...
9.985E-10	B		exponential 1.000E-03	-		Default/Zone;1'=...
	F		exponential 1.000E-03	-		Default/Zone;1'=...
	G		exponential 1.000E-03	-		Default/Zone;1'=...
9.985E-10	B		exponential 1.000E-03	-		Default/Zone;1'=...
	H		exponential 1.000E-03	-		Default/Zone;1'=...
	J		exponential 1.000E-03	-		-
9.985E-10	C		exponential 1.000E-03	-		Default/Zone;1'=...
	D		exponential 1.000E-03	-		Default/Zone;1'=...
	E		exponential 1.000E-03	-		Default/Zone;1'=...
9.985E-10	C		exponential 1.000E-03	-		Default/Zone;1'=...
	H		exponential 1.000E-03	-		Default/Zone;1'=...
	r		exponential 1.000E-03	-		Default/Zone;1'=...

Close

**Attribute cuts** are described in the first table. In this part, the cuts are ordered in function of the attribute.

If the option **Merge identical values** is checked, in this table in the case of 2 events have the same attributes, the attributes are not repeated.

Without Merge identical values			With Merge identical values		
Attribute cuts	Nb.		Attribute cuts	Nb.	
MCS (origin)	9		MCS (origin)	9	
1 Zone[1] : {1;1}	1		1 Zone[1] : {1}	1	
1 Zone[1] : {1;3;3}	2		1 Zone[1] : {2}	1	
1 Zone[1] : {1;4;#Undef}	1		1 Zone[1] : {1;3}	2	
1 Zone[1] : {2;2;2}	1		1 Zone[1] : {2;4}	2	
1 Zone[1] : {2;4;4}	2		1 Zone[1] : {1;4;#Undef}	1	
1 Zone[1] : {2;3;4;4}	2		1 Zone[1] : {2;3;4}	2	

In the top right corner, cuts are sorted by events or by attributes:

In this tab, minimal cuts set are sorted by attributes cuts. In the example, there are 6 different attribute cuts and 2 cuts have only 1 attributes (for example zone 1), 2 have 2 different zones, etc...

Attr.'s cuts order	Evt.'s cuts order	
Orders	Cuts	Probabilities
1	0	-
2	1	9.990E-07
3	4	5.991E-09
4	1	1.996E-12
Total	6	1.005E-06

Attr.'s cuts order		Evt.'s cuts order
Orders	Cuts	Probabilities
1	0	-
2	1	9.990E-07
3	6	5.991E-09
4	2	1.996E-12
Total	9	1.005E-06

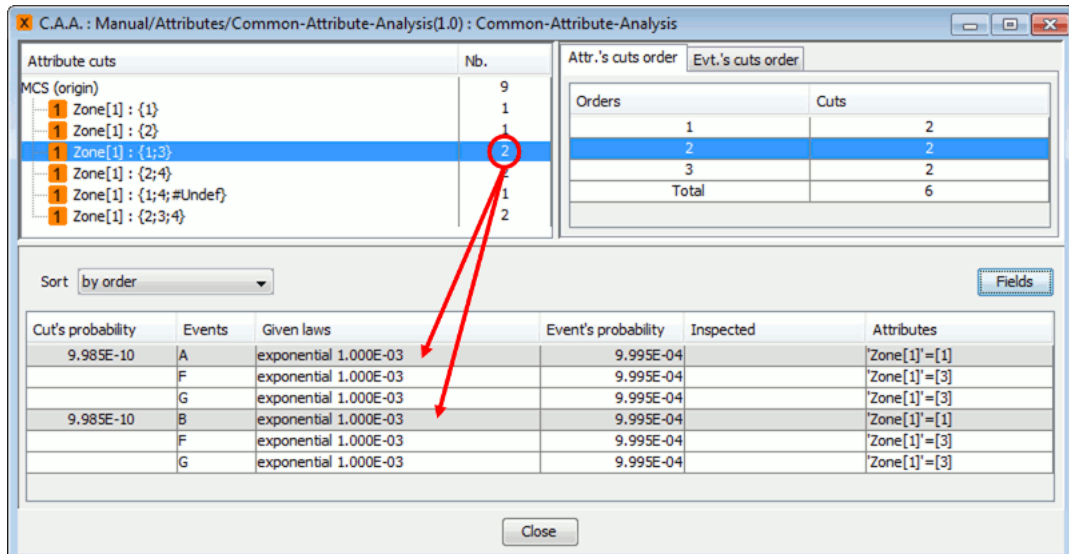
In this tab, minimal cuts set are sorted by events. In the example, there are 9 different cuts and 1 cuts have only 2 events, 6 have 3 events, etc...

In the last part, the different cut is described in function of the attribute cuts selected in the table in the top left corner. For example, If MCS is selected (in blue line), all cuts are detailed whereas if another line is in blue, only the cuts made by this attribute cuts are detailed.

As previously, it is possible to sort the cuts by **order**, by **order/occurrence** or by **occurrence**.



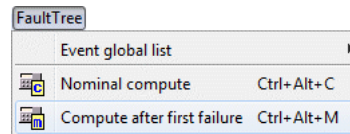
The **Field** button can be used to manage the columns displayed in the table.



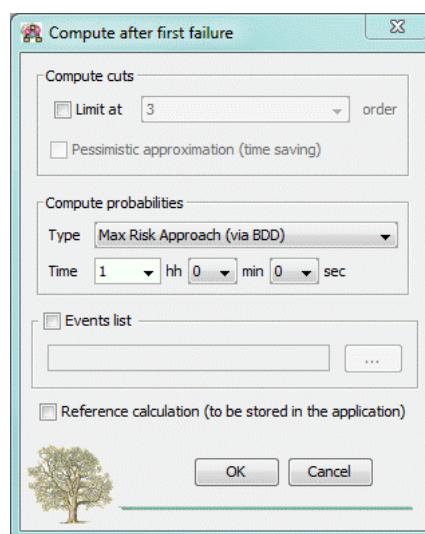
### 5.1.6. Minimum equipment list computations (MEL)

The computations after first failure are done to make easier the building of the M.M.E.L (Master Minimal Equipment List). At first, a nominal minimal cut set computation is done (see Section 5.1.4, “Minimal cut set calculations (MCS)”). Then several minimal cut set computation are done, each time with one basic event set to its fail state (true state) and the others set to their nominal state (which can be any state different as the normal one).

The computation after first failure can be launched in the menu: **FaultTree-Compute after first failure:**



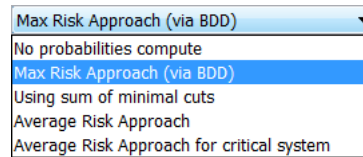
The following window appears in order to set-up the calculation:



The settings are the following:


- **Limit at** option permits to choose the cut set maximal order (in order to limit calculation to the selected order);
- Select the **Pessimistic approximation (time saving)** option in order to save a large amount of time (this option is described further in this section);

- Several type of calculations can be performed.



- **No probabilities compute:** to have results without probabilities;
- **Max Risk Approach (via BDD)**
- **Using sum of minimal cuts:** Max Risk Approach (via minimal cuts)
- **Average risk approach**
- **Average risk approach for critical system**

For more informations about these types of computation, see Appendix B, *Probabilities computations*.

- **Time** permits to specify a duration (in hours) for the time, in case when some probability laws for tree basic events depend on the time factor (i.e. Exponential law);
- If **Events list** is checked, calculations will be performed only in the events list loaded by the user using the  button.
- **Reference calculation** stored the calculation in the databased. If this option is checked, if there are any modifications on the model, user can display the results using the compute windows.
- Click on the **OK** button to launch the calculations.

The result is visible in the **MEL tab**:

Events [Nominal]	MEL probabilities 2.669E-04	Factor -	order 0 0	order 1 2	order 2 3	order 3 2	order 4 0	order 5 3	order 6 2
Barre1	1.000E00	3.747E03	1	0	0	0	0	0	0
Barre3	1.000E00	3.747E03	1	0	0	0	0	0	0
BK1	2.669E-04	1.000E00	0	2	3	2	0	3	2
BK3	2.669E-04	1.000E00	0	2	3	2	0	3	2
Secu	2.002E-01	7.502E02	0	3	0	0	1	0	0
K1b	3.668E-04	1.374E00	0	2	4	1	0	4	1
K1h	3.668E-04	1.374E00	0	2	4	1	0	4	1
K3b	3.668E-04	1.374E00	0	2	4	1	0	4	1
K3h	3.668E-04	1.374E00	0	2	4	1	0	4	1
OP	2.945E-04	1.103E00	0	5	2	0	0	0	0
BK2	2.669E-04	1.000E00	0	2	3	2	0	3	2
BP1	2.669E-04	1.000E00	0	2	3	2	3	2	0
BP2	2.669E-04	1.000E00	0	2	3	2	3	2	0
BP3	2.669E-04	1.000E00	0	2	3	2	3	2	0
BP4	2.669E-04	1.000E00	0	2	3	2	3	2	0
Barre2	2.669E-04	1.000E00	0	2	3	2	0	3	2

Max Risk Approach (via BDD) ; 2.000E00

☐ Per flight hour

Close

The result window shows a table composed of several columns. It contains a line per basic event. Each of these lines means the result of the minimal cut set computation with the corresponding event set to the fail state (true state). The meaning of the columns is as follows:

- 1<sup>st</sup> column: **Basic event name**;
- 2<sup>nd</sup> column: **MEL probabilities**;
- 3<sup>rd</sup> column: **Factor** (%) corresponds to the MEL events probabilities divided by the MEL nominal probability;
- From the fourth column: Minimal cut number for the order indicated by the column header label.

The red line indicates an error. When the mouse is on the red line a tooltip appears in order to describe the problem at the user. Results can be reduced to one flight hour using the dedicated check box (see Section 5.1.7, “Reduce to flight duration”).

**Pessimistic approximation (time saving)** (only if calculation is limited at X orders) option.

This option does the following steps:

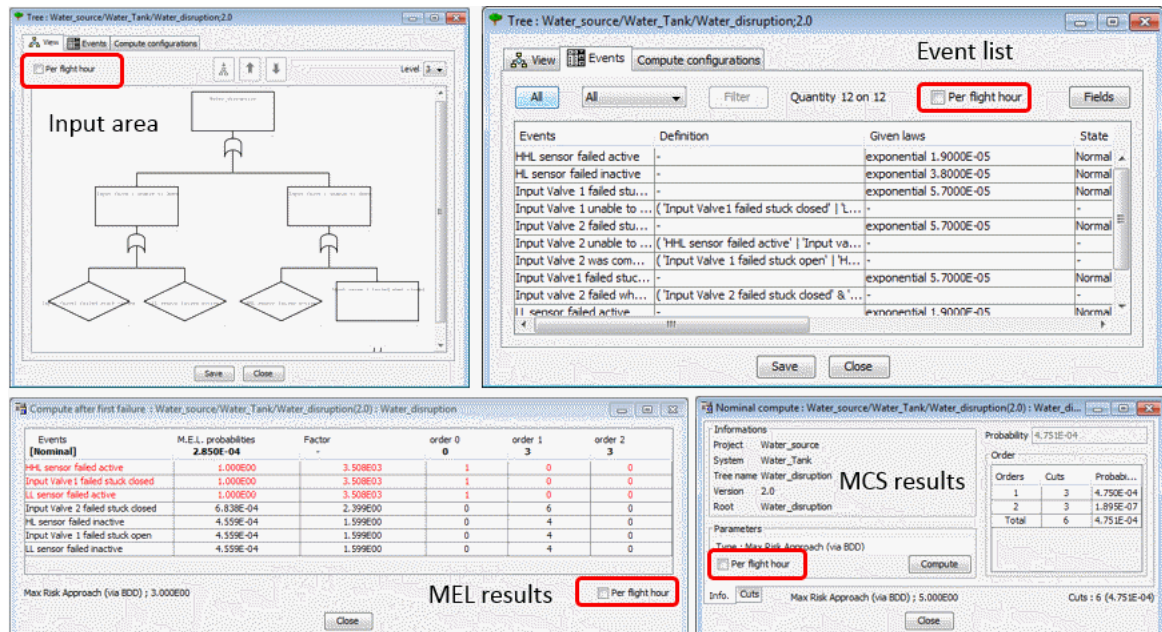
- Launch a nominal computation on the edited tree at the order N+1, where N is the order indicated in the Limited at option field;
- Build a tree representing the logical disjunction of the minimal cut sets obtained by the previous computation;
- Launch a computation after first failure on this built tree.

Using this method, although the computation results are not correct, the pessimistic approximation obtained is sufficient in some situations and then allow precious time savings.

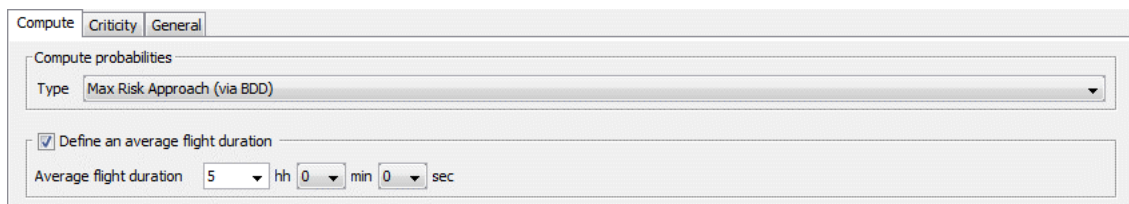
### 5.1.7. Reduce to flight duration

Results can be reduce per flight hour. In this case, the results are divided by the calculation time to be referred at 1 hour. This option is available in 4 several windows:

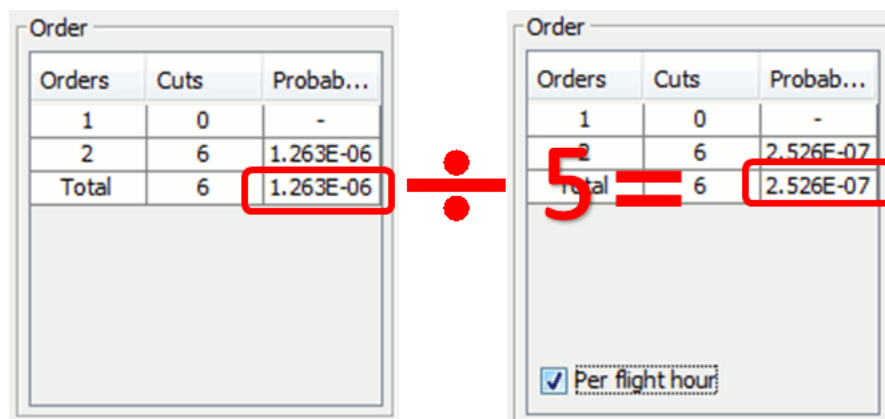
- Input area view;
- Event list;
- MCS results;
- MEL results.



In the **Project properties**, User can define the average flight duration.



In this case, even if the calculation time is more or less than the average flight duration, when a reduction per flight hour is asked the the average flight duration is considered and to have the reduction the result is divided by the average flight duration.



### 5.1.8. List of errors

The list, specific for each opened tree, displays any potential errors which may have been detected during opening of a tree:

- Detection of inconsistencies between tree events and associated sub-tree events;
- Detection of absence of sub-trees linked to the tree to be opened;
- Detection of access problem to one or several sub-trees associated to the tree to be opened;
- Detection of no existence of attributes associated to some events in the tree being edited.

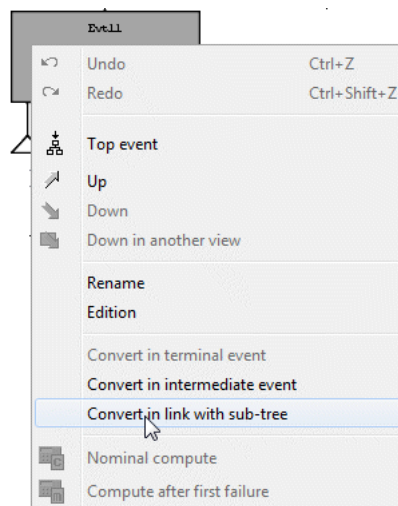
If one of these errors occurs during tree opening a message appears.

### 5.1.9. Sub-Tree

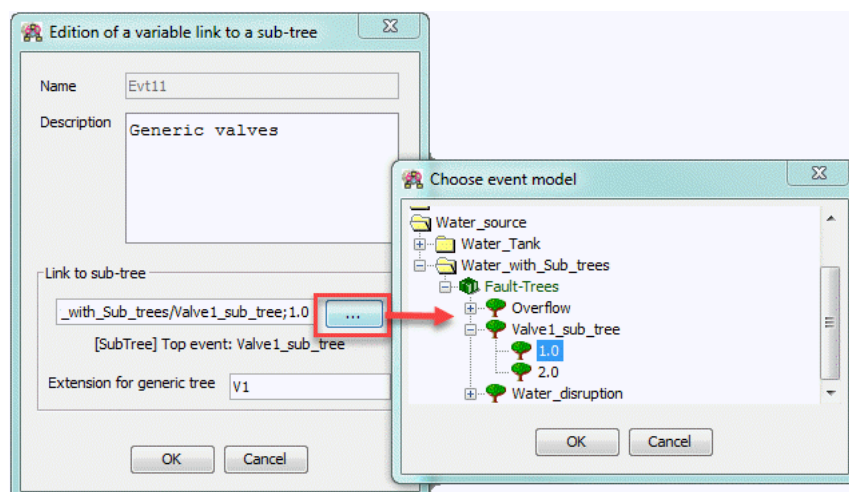
Sub-trees allow to cut a study/a tree in different part. They also have the advantage to be able to be used in different trees (that can be effective to avoid the duplication of common tree to various undesired events).

So that the user can connect a sub-tree to a main tree, it is necessary:

1. Create a tree in a classic way which will correspond to the sub-tree;
2. Create an event in the main tree which will have the role of connector with the sub-tree;
3. by a right click on the event, select **Convert in link with sub-tree**.



The following window appears to select the sub-tree.



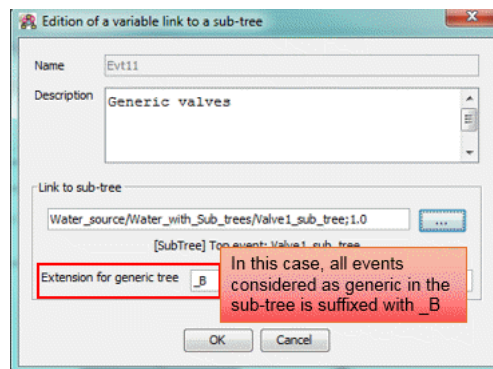
In the event list, event in red are sub-tree events (see Section 5.1.3, “List of events”).

Events	Definition	Comments	Given laws	State	Sub
Evt11	('Input Valve 1 failed stuck open'   'I...	Generic valves	-	-	Wat
Evt12	-	-	-	Normal	-
Evt_1	-	My event Evt_1	exponential 1.0000E-03	Normal	-
Evt_2	(Evt11   Evt12)	-	-	-	-
Evt_3	-	-	-	Normal	-
Evt_4	-	-	-	Normal	-
Evt_5	-	-	-	Normal	-
HL sensor failed inactive	-	-	exponential 3.8000E-05	Normal	-
Input Valve 1 failed stu...	-	-	exponential 5.7000E-05	Normal	-
Input Valve 1 not comm...	('HL sensor failed inactive'   'LL senso...	-	-	-	-
LL sensor failed inactive	-	-	exponential 3.8000E-05	Normal	-
MyFirstTree	(Evt_1 & Evt_2 & Evt_3 & Evt_4 & E...	Unwanted event apparition	-	-	-

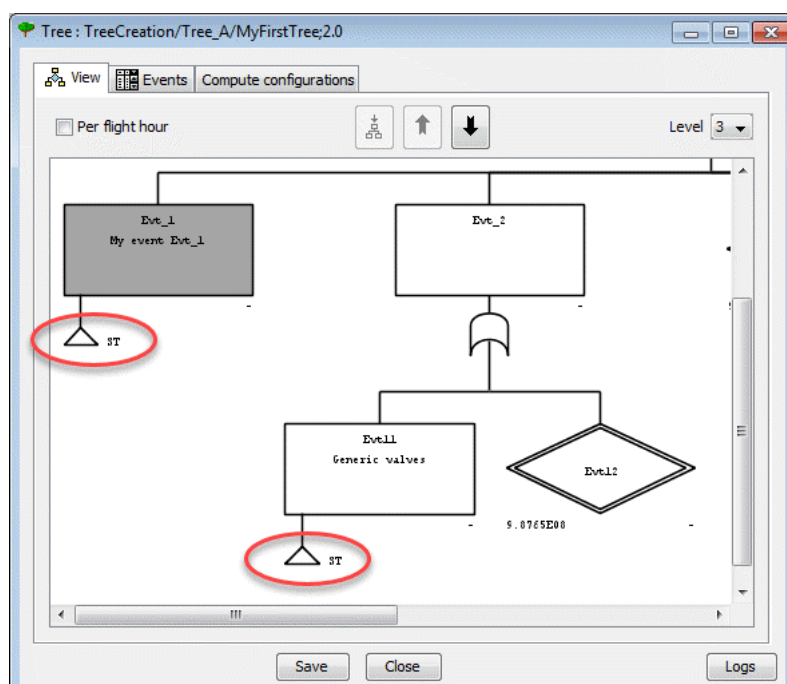
In this first example, a sub-tree is linked to the fault tree but if several events are connected to the same sub-tree, the basics events are the same in each part.

To consider different events, it is necessary to consider the events in the sub-tree as **Generic** (see Section 5.1.2, “Events edition”).


When done, one can link the sub-tree and define a suffix that will be concatenated to the name of events (intermediate and undeveloped) that were tagged as generic.

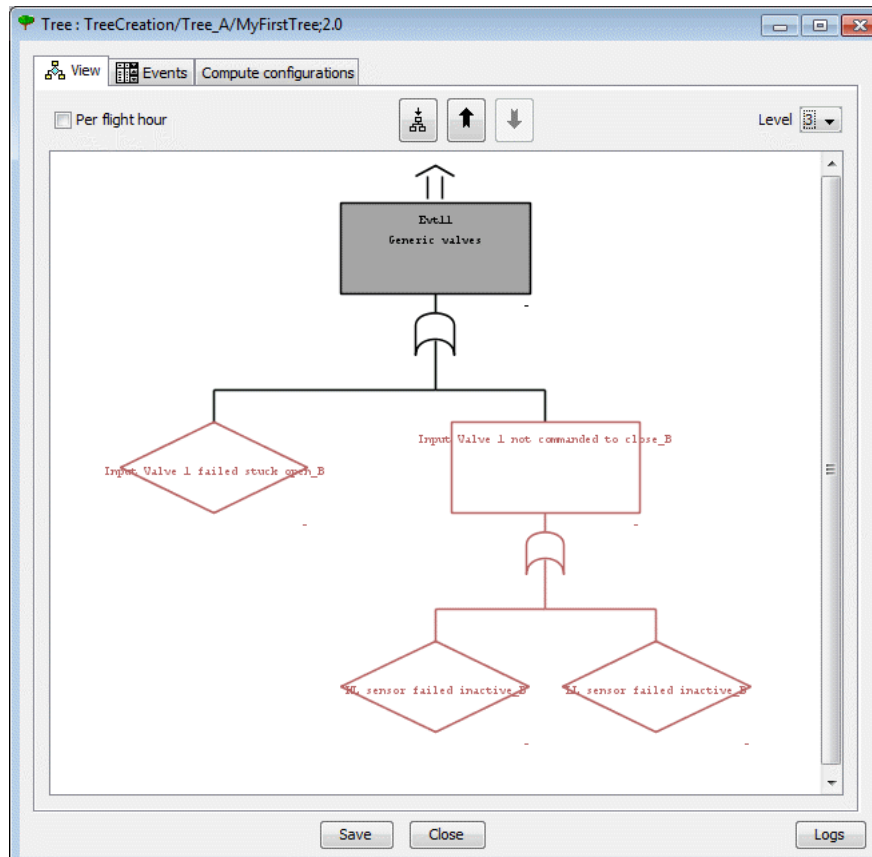


In the main tree, the identification **ST** informs the user that a sub-tree is linked.





Using the  button, user can see the sub-tree. Events which made of the sub-tree are displayed in blue with the chosen suffix by the user.



## 5.2. Common Cause Failure (CCF)

The common cause failures (CCF) are a sub set of the dependent failures. These are failures with the same origin, simultaneous of two or more components technologically identical, in a same system. CCF groups are managed in the **Others** tree structure (see Section 3.1, “Technical object edition” for creation).

CCF groups are associated to basic events. So, a CCF group manager window is associated to each open tree.

The screenshot shows a window titled "CCF\_Valves;1". It contains the following fields and controls:

- Name:** CCF\_Valves
- Version:** 1
- Description:** CCF used for all the valves
- Model:**
  - Type of model: Beta model
  - Beta: 0.02
- Law:**
  - Law: Exponential
  - Parameters:
    - Lambda: 1e-7

At the bottom, there are "Save" and "Close" buttons.

When the CCF object is created, the setting is the following:

- **Description:** CCF group description;
- **Model:** the three possible **Type of model** are implemented: Beta factor (Beta), multiple Greek letters (MGL) and the Alpha method (Alpha). When the type is selected, it is necessary to input the associated parameter;
- **Law:** fields which define the CCF probability law.

Click on the **Save** button to validate.

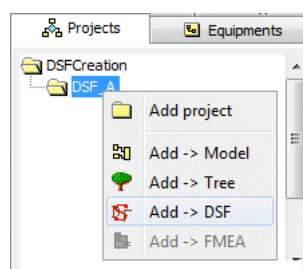
## 5.3. Dependable System Faults (DSF)

During the development of a complex system, one have to split the overall system is sub-systems that can be managed by a team. As these sub-systems are not isolated from each other's but are more or less intricate, sharing some devices, one being supplier one from the others, it is sometimes difficult to assess the safety of one without considering the other ones.

The DSF is an object that can be either a failure rate or a tree depending on user's selection. DSF is especially convenient to allow one sub-system supplier to work considering the failures of the sub-system inputs (power, data...) while the system at the input is still not sufficiently defined to provide a consistent tree of the input failure.

This allows to either isolate sub-systems from each other's or interconnect them when they provide the sufficient level of maturity (depending on user's choice).


A DSF can be created in the **Projects** tab after creating the necessary tree (see Section 3.1, "Technical object edition"). Create a DSF as follows:




A DSF is only a connection point. The tree and the FRB objects have to be defined somewhere else and linked with the DSF.

The DSF can be link:

- With a FRB if only a data is used (first part of the window).
- With a Sub-tree (already created) if details are provided (lower part);

 It is obligatory to connect a DSF to a FRB but not to a Fault Tree.

## Computing with DSF

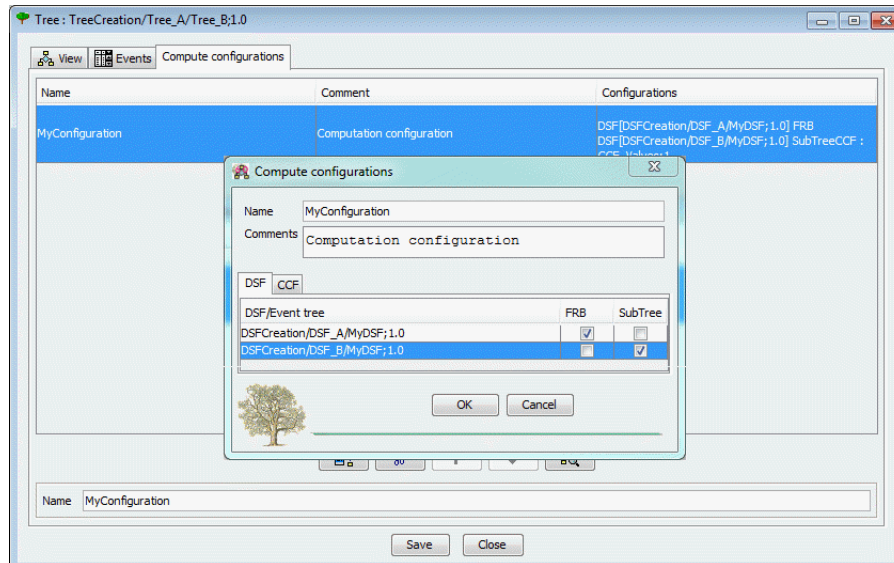
After creating a DSF, it must be linked to an event using  as made for a sub-tree.

When willing to compute a fault tree which referenced DSF, the user can select the computing options (selection between the FRB and the sub-tree) in the DSF tab of the compute window:

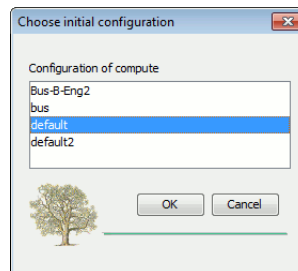
- In **Fault tree**, select **Compute configurations**;




- Create configurations with a name and a type of DSF used (FRB or Sub-tree);



- When user starts computation (minimal cut set, compute after first failure), a pop-up allows to choose the configurations used for calculations.



-  Note that the user can perform its selection and save it to ease future computations or to allow batch computing.

## 6. Model-Based Safety Assessment (MBSA)

### 6.1. AltaRica - Support language for MBSA

#### 6.1.1. Historical

Cecilia-Workshop is a model-based safety assessment platform based on the Altarica language. AltaRica is a modelling language for MBSA developed in first by the LABRI in 1999 for safety assessment of industrial systems. In 2000, OCAS project is to integrate Altarica language. And the first operational version appeared in 2001.

OCAS provides a graphical modelling environment, which allows us to describe dynamic transition systems.

This language allows:

- a behavioral description with a close descriptions studied systems;
- the integration of the functional and dysfunctional aspects to obtain different point of views for safety studies and for the check different parameters from the same model;
- to facilitate a rigorous compilation towards the safety models (Fault tree, petri nets, sequence) and to check (Lustre, Esterel)

This language is based on components library and the architecture is controlled by automatic consistency-control.

Moreover, the model can be validated by using step-by-step simulation.

#### 6.1.2. Main Characteristics

An Altarica model is a hierarchical model. Each nodes can communicate through flows and synchronizations.

2 types of nodes exist:

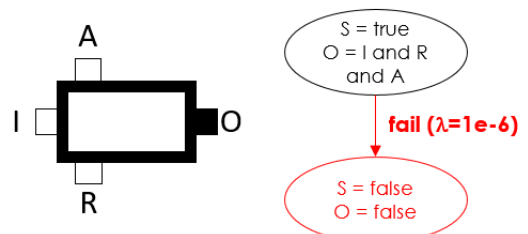
- component represents a single process of the system, it cannot contain definition of others nodes or synchronizations;
- equipment represents a container for nodes; it may contain declarations of others and synchronizations.

A node is defined by:

- a name;
- internal variables only visible for the node and introduced by state;
- external variables called flows which can be visible by others nodes;
- transitions which describe the component behavior;
- constraints which describe the allowed configuration for variables. This assertion links flow and state variable.

For example:

```
node block
  flow
    I,A,R : bool : in ;
    O      : bool : out ;
  state S: bool;
  event fail;
  trans
    S=true |- fail -> S := false;
  assert
    O = (S and I and A and R);
  init S := true;
  extern
    law <event fail> = exponential(1e-6);
edon
```



## 6.2. Input AltaRica model

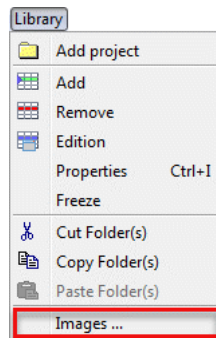
### 6.2.1. General

A MBSA model is created in a **Project** folder.

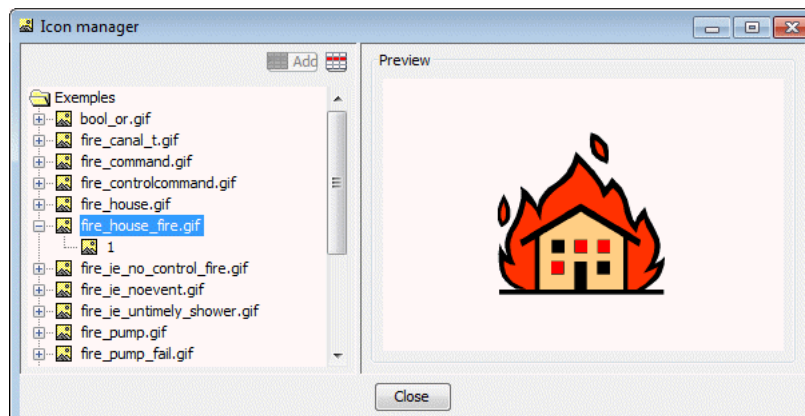
Section 3, “Technical object” describes the creation and the operations (like rename, freeze, delete, etc...) available for the technical elements.

### 6.2.2. Icon manager

The **icon manager** is accessible via the **Library** menu:




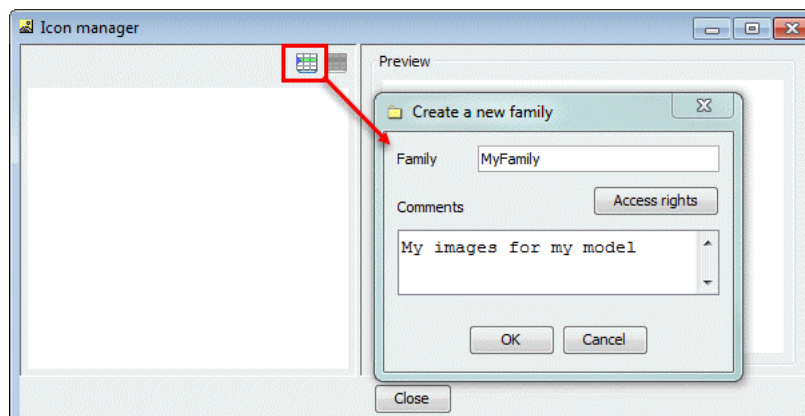
The following window appears with all the icons saved in the database and which can be used inside the MBSA model.



The **icon manager** allows selection of the icon(s) which graphically describe(s) the component behavior during the simulation of the architecture system:

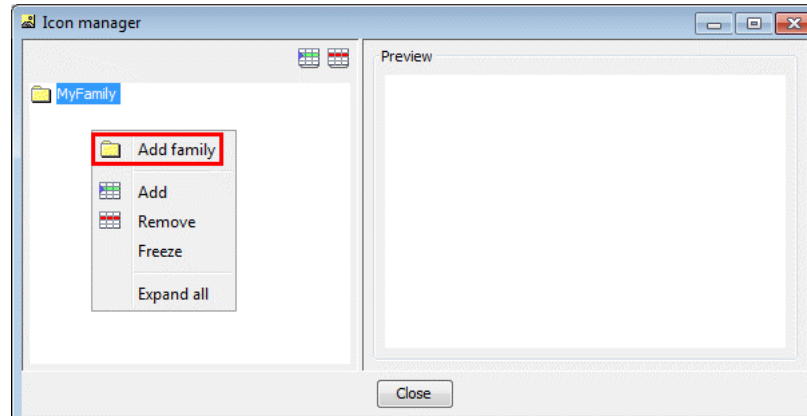
- In the left part, there is a preview of the icon.
- In the right part, there is the directory and icon manager:

At the first launching, Icon manager is empty, to create new icons library click on the  icon. A new windows appears to type the name of the new library, *e.g.* "MyFamily" and a comment.



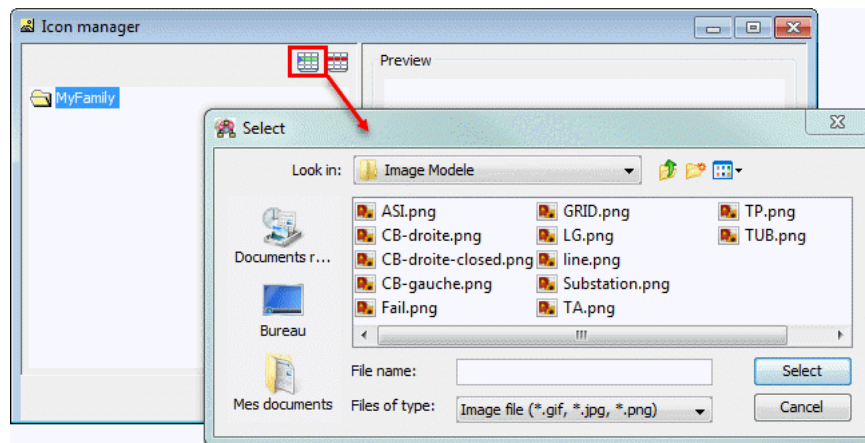


A right click allows to create new folder selecting **Add family**.



When folder is created, icon is used to select the image.

Now, it is possible to add image in the folder using icon.

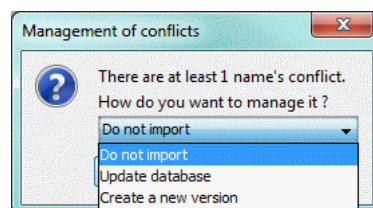


A window appears to select the directory in which the icon is stored and choice the desired icon.



The last reached directory is memorized and is automatically displayed during the following selection.

If an image with an existing name is already loaded, user can choose if it is a new version or an update of the current image.



icon can be used to delete an image or a folder.

## 6.2.3. Type management

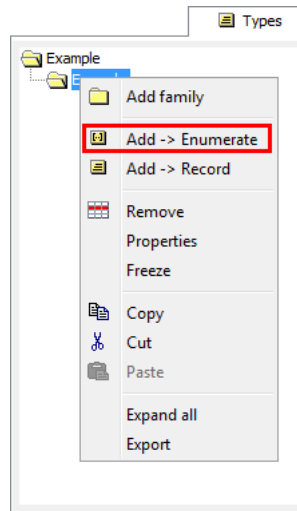
### 6.2.3.1. Create new enumerate type

In order to avoid having to type variables values of "enumerated" type in the **I/O** and **Etat** editor component model tabs, it is possible to define enumerated types in Library.

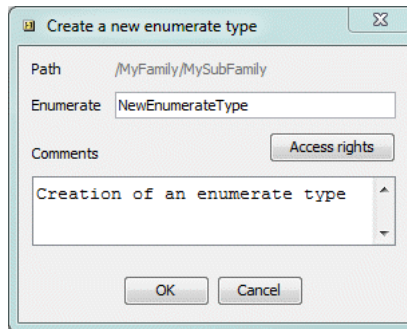
After creating a Family (see Section 3, "Technical object"), a **Enumerate** type can be created:

- Click on the Types tab to gain access to the list of types in the left part of the screen.

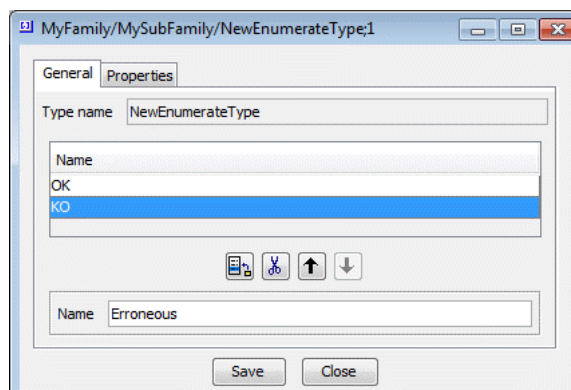
- Click with the mouse right button to gain access to the contextual menu or use the Library menu.
- Click on the **Add -> Enumerate** command.



- In the Type editor window and in the Type name field, type the name.



- When the technical object is created, a double click on its version value allows to set up the enumerate type. In the Name field at the bottom of the window, type a value name.



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To move down in the list.



To move up in the list.



In the Types tab in the left part of the screen, the enumerate type is indicated by the symbol.

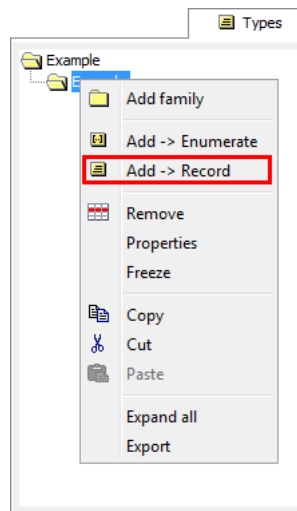
## 6.2.3.2. Create new structured type

The structured type allows the transit of several flow information variables through the same link. In the Altarica model, the enumerate type is a list of values for a variable (of state or flow) whereas a structured type is a list of flow variables.

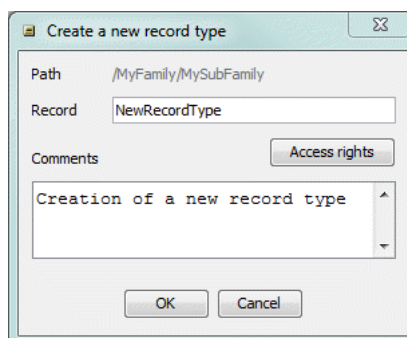
The use of structured types allows the regrouping of several flows (information flow variables) on one bus and thus to reduce the number of graphic links to create in order to connect the components (equipment) during the edition of the architecture system. Only the flow variables (in or out) can be declared with this type of structured field (the state or local variables don't have any access).

After creating a Family (see Section 3, “Technical object”), a **Record** type can be created:

- Click on the Types tab to gain access to the list of types in the left part of the screen.
- Click with the mouse right button to gain access to the contextual menu or use the Library menu.
- Click on the **Add -> Record** command.

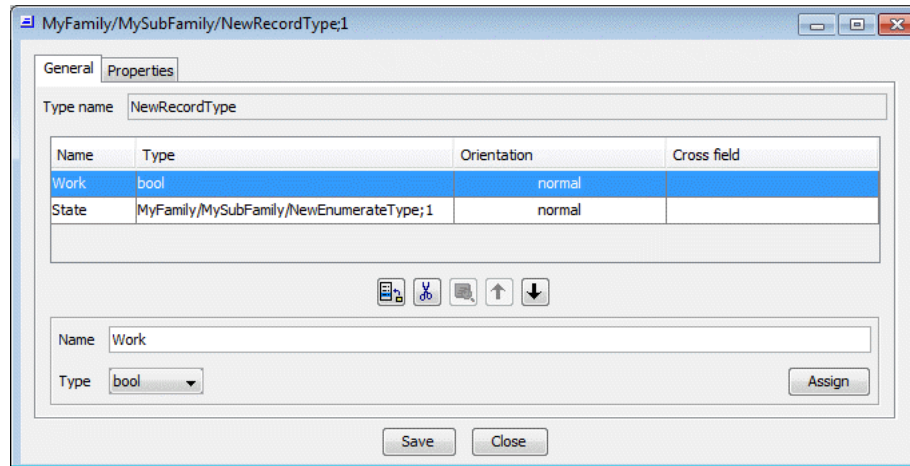


- In the Type editor window and in the Type name field, type the name.



- When the technical object is created, a double click on its version value allows to set up the record type.

In the Name field at the bottom of the window, type a value name.



The following icons are used for creating a type:



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To edit a parameter type if it is predefined (or double click with the mouse right button in the Type cell of the selected line),

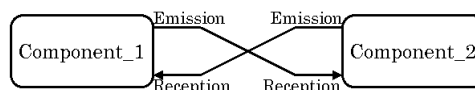


To move down in the list.



To move up in the list.

- In the **Type** menu, select the new type.
- Click on the **Assign** button to modify the parameter type, if necessary; the type is displayed in the Type column.
- Click in the Orientation column; select the orientation: normal or inverse.
- Click in the Crossfield column which is used to show a bidirectional flow



- Type the desired cross field and validate with the **Enter** key.

## Transcription in Altarica code:

The definition of the structured type flow generates two connection studs:

1. a port in;
2. a port out .

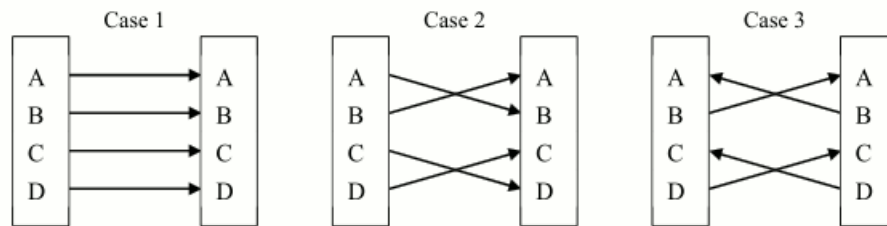
Each connection stud is composed of n fields (flow parameters).

In general the field has the same orientation as its reference stud. With the clause inverse we can specify if the fields of the stud in or stud out have an inversed orientation.

The figure below specifies the various relations of possible equality between the fields of the stud out and stud in according to the crossfields and orientations defined for the flow parameters (A, B, C, D).

- Case 1: A data structure with direct assignments;
- Case 2: A data structure with cross field assignments;

- Case 3: A data structure with inversed and cross field assignments.



In order to simplify the Altarica code interpretation presented below, the equality relations are oriented and thus defined using the assignments.


To access the type field, the character is "^"

```
link
  flow A,B,C,D : int ;
  assert
    in^A := out^A;
    in^B := out^B;
    in^C := out^C;
    in^D := out^D;
  knil
```

```
link
  flow A,B,C,D : int ;
  assert
    in^A := out^B;
    in^B := out^A;
    in^C := out^D;
    in^D := out^C;
  knil
```

```
link
  flow A,B,C,D : int ;
  inverse out^A; out^C;
  in^B; in^D;
  assert
    in^A := out^B;
    out^A := in^B;
    in^C := out^D;
    out^C := in^D;
  knil
```

These commands can also be activated via a contextual menu obtained by clicking in a cell with the mouse right button.

- i** In the Types tab in the left part of the screen, the enumerate type is indicated by the  symbol.
- i** In the list of "predefined" types, only the list of enumerate types appears (the list of record types is not displayed) because a record type cannot contain another record type.

## 6.2.4. Operator management

### Definition

The operators allow simplifying the writing of the assertions for the components within the Altarica code. The description of the transfer function is very close to the description of a component model, with the close difference that this function doesn't describe behavior. An operator is thus a component model which has an output flow and a certain number of input flows, the assertions enable to define the output flow according to input flows.

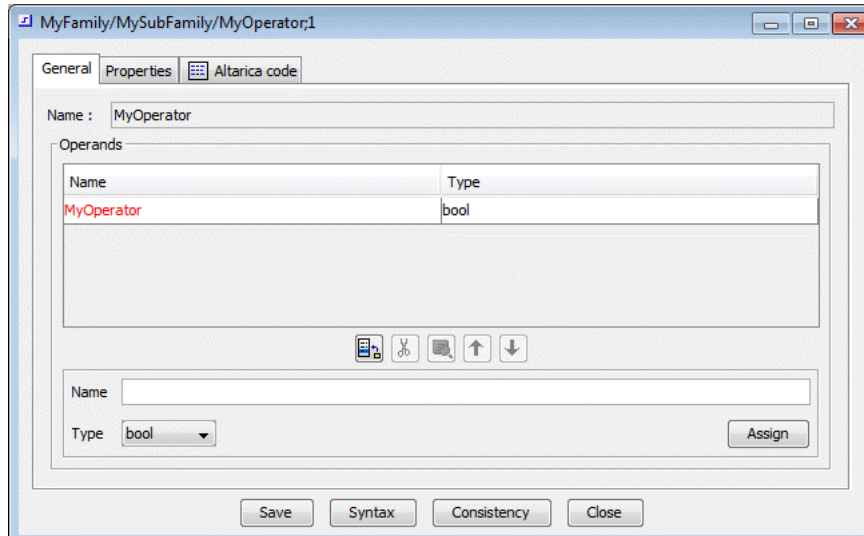
The flow type variables associated with the definition of an operator are unspecified and can be structured. In this case, the structured flow parameters should not be reversed. Moreover, equality between structured flows, takes systematically into account the crossings defined for the structured field.

- i** The operators can be used only within the components (or equipment) models assertion. The use of operators in the transitions definition (Altarica code) is currently forbidden.

After creating a Family (see Section 3, "Technical object"), a **Enumerate** type can be created:

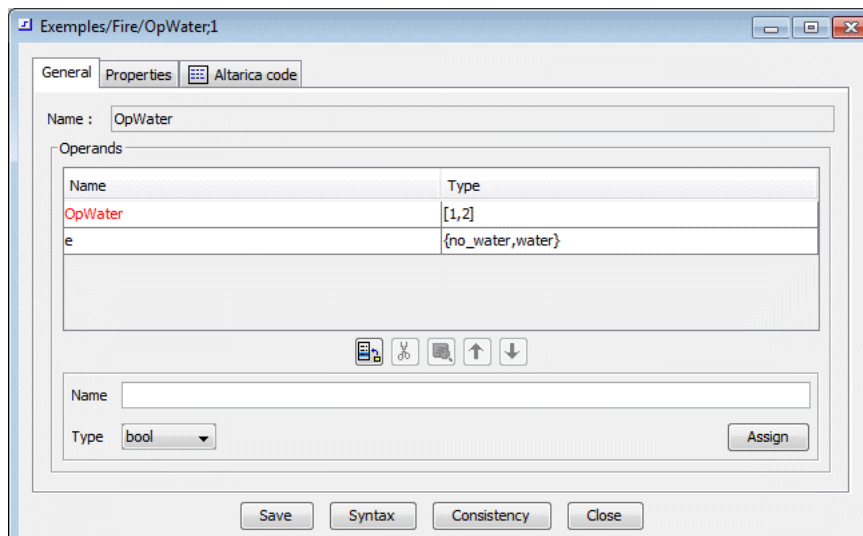


- The window Operator editor is displayed; it is composed of 3 tabs:



- General,
- Properties,
- Altarica code.

## 6.2.4.1. General tab



In the General tab carries out the following operations:

- In the **Name** field (at the top left in the tab), type the operator name, *e.g.* opWater.



In the Operands area, the first line is dedicated to the operator result and the first operand has always the same name as the operator; this first line cannot be removed.

- In the **Name** field at the bottom left of the **Operands** area, type the operand name.
- Use the **Type** menu and define the operand type.
- To modify the type of an existing operand, select it and click on the **Assign** button.



The type of an operator or of one of its operands may be a record-type if the parameters of this type are neither crossed over nor inverted.



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To edit a parameter type if it is predefined (or double click with the mouse right button in the Type cell of the selected line),

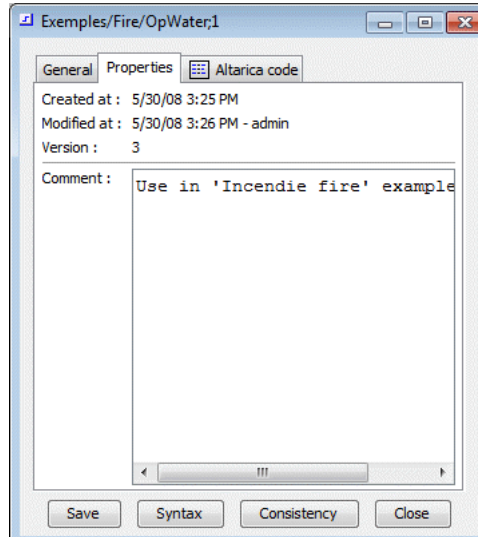


To move down in the list.



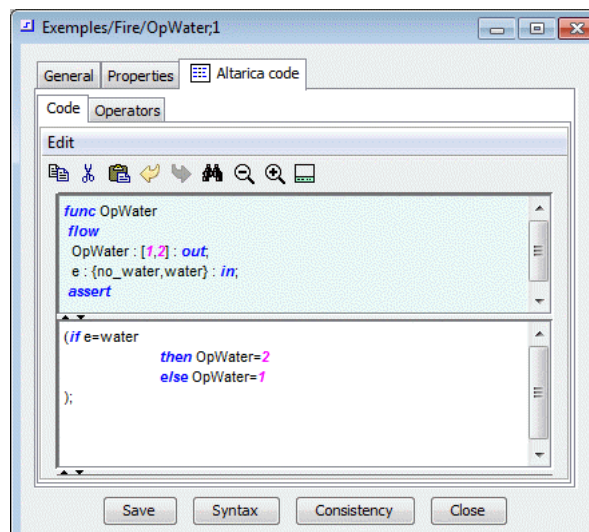
To move up in the list.

## 6.2.4.2. Properties tab



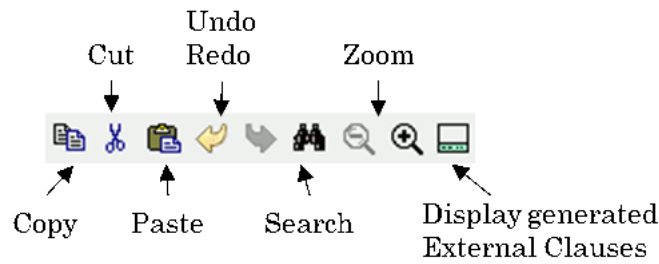
- The Properties tab allows visualization of the model properties:
  - Creation date,
  - Last modification date,
  - Version.
- The **Comment** field allows the filling of a comment describing the operator function.

## 6.2.4.3. Altarica code tab



The Altarica code tab contains:

- An icon bar:



- A display zone (non editable) for the operands and their types. The Altarica code displayed in this zone has been automatically generated from the information filled in the General tab.

This generation is done as following:

- Creation of an output flow variable carrying the operator name,
- Creation of input flow variable for every operator operand,
- This end of definition is indicated by the key word assert.

A zone for typing the operator Altarica code. The Altarica code syntax of an operator is identical to the component assertions (see . Appendix E, *Language AltaRica* ).

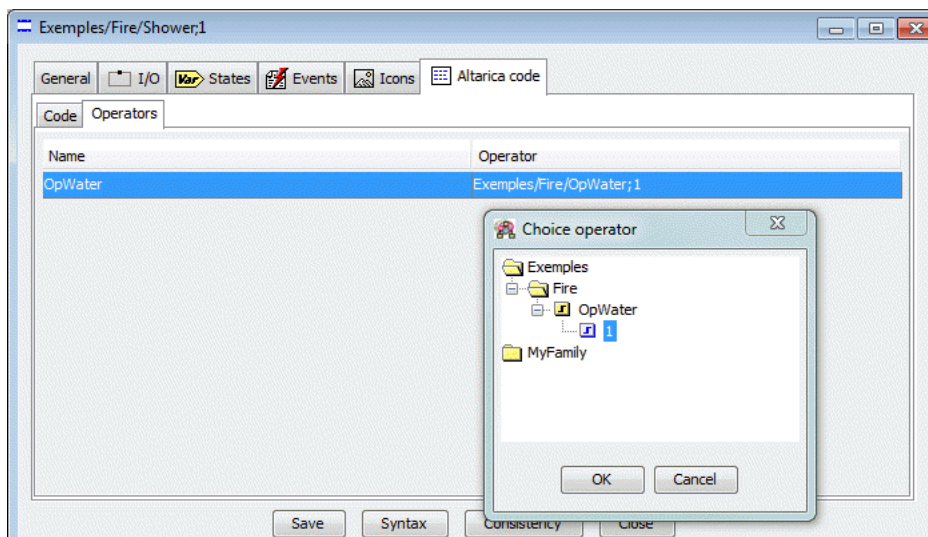
```
func OpWater
flow
OpWater : {1,2} : out;
e : {no_water,water} : in;
assert
(
  if e=water
  then OpWater=2
  else OpWater=1
);
```

The assertion must specify the result of the transferred function, namely the assignment of the exit flow variable associated with the operator (OpWater) according to the entry variables.



Operators can be used to define the script definition of another operator.

If operators are used in Altarica code, Operators tab is accessible. It allows selecting operator family and operator version, if different operators have the same name. If there is no ambiguity, the only family containing the operator is selected with the last;version.



- When the typing is done, click on the **Syntax** button in order to carry out the syntax control of Altarica script. In the event of detection of syntax error, an error window appears describing the error and indicating the associated script line.

The most common syntax errors are:

- A typing error in the Altarica code, creates a false identifier that doesn't correspond to the text in the header dedicated to the display of variables specified in the tabs.
- Assertions define assignment of variables of different types (two types enumerated differently, one type enumerated with Boolean...).
- Click now on the button **Consistency** in order to carry out the consistency control of Altarica script defined for the transfer function associated to the operator.

In case of detection of consistency errors, an error window appears describing in a precise way the consistency problems detected on the Altarica code.

The most often consistency errors detected are:

Existence of a combination of operands values (input flow variables) for which the result of the operator is not defined.

Existence of a combination of operands values (input flow variables) for which the result of the operator (output flow variable) has 2 different values.

### 6.2.5. Component management

A component model under Cecilia-Workshop can be considered as a flow processor which:

- has an internal state;
- reacts to events;
- receives and/or send information through (I/O) interfaces which enable him to communicate with other components (by flow propagation).

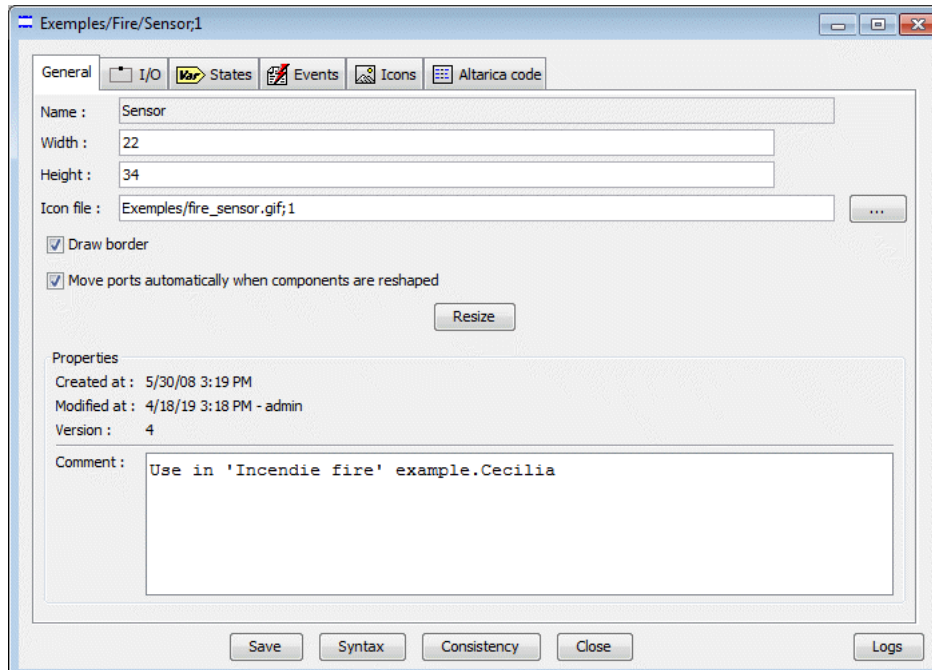
When an event occurs, the component determines the internal change of state induced by his release, and the new flow values emitted by its interfaces.

The editor model for the component model allows the user to describe his mechanisms by the means of the following tabs:

- The **General** tab (See Section 6.2.5.1, “General tab” ) allows to fill the general information associated to the model.
- The **I/O** tab (See Section 6.2.5.2, “I/O tab” ) allows to fill the list of the information flow variables transmitted or received by the component.
- The **States** tab (See Section 6.2.5.3, “State tab” ) allows to fill the list of the component state variables characterizing the various states of the component: functional or dysfunctional;
- The **Events** tab (See Section 6.2.5.4, “Events tab” ) allows to specify the events on which react the component (phenomena modifying the intern state of the component).
- The **Icons** tab (See Section 6.2.5.5, “Icons tab” ) allows to specify the graphic representations corresponding to the various states of the component (used during the graphic simulation of the system).
- The **Altarica code** tab (See Section 6.2.5.6, “Altarica code tab” ) allows to specify the state changing conditions of the component and how the information flow propagation is carried out in each one of these states.

When the model created is saved, it is automatically inserted in the administrator (alphabetically) as well as his version.

### 6.2.5.1. General tab



The **General tab** allows the following information to be typed in:

- **Name:** type the model name,



In order to save the model, the name is the only obligatory field.

- **Width:** ll (icon width default value),
- **Height:** hh (icon height default value),
- **Icon file:** the button enables selection of the icon to be assigned to the model.



Icons are image files in "\*.gif" or "\*.jpg" format.

- Tick **Draw border** to display or not the border associated to the icon.
- Tick **Move ports automatically when components are reshaped**.
- Click on the **Resize** button to validate, if necessary, the icon larger dimensions.
- **Comment:** fill the comment describing the created component.

All the model editor tabs contain four common buttons:

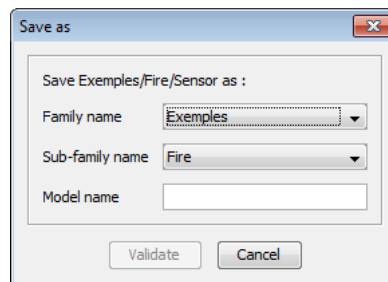
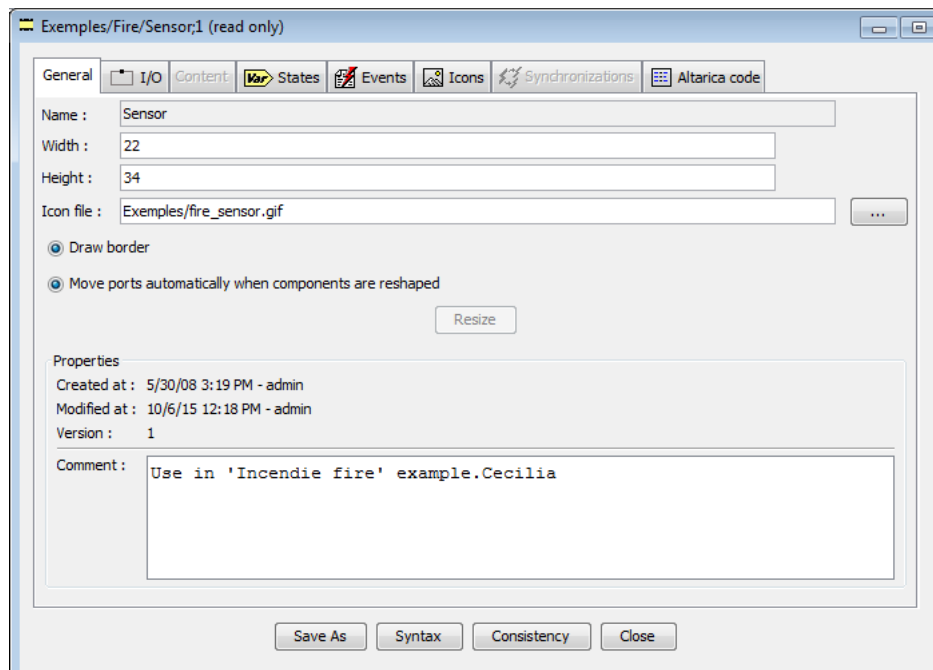
- **Save:** allows storing of all typed information,
- **Syntax:** allows a syntax control on the Altarica code,
- **Consistency:** allows a consistency control on the Altarica code,
- **Close:** allows closure of the model editor.

Two icons on the top right hand side allow window size modification:

- : to reduce the window;
- : to extend the window.



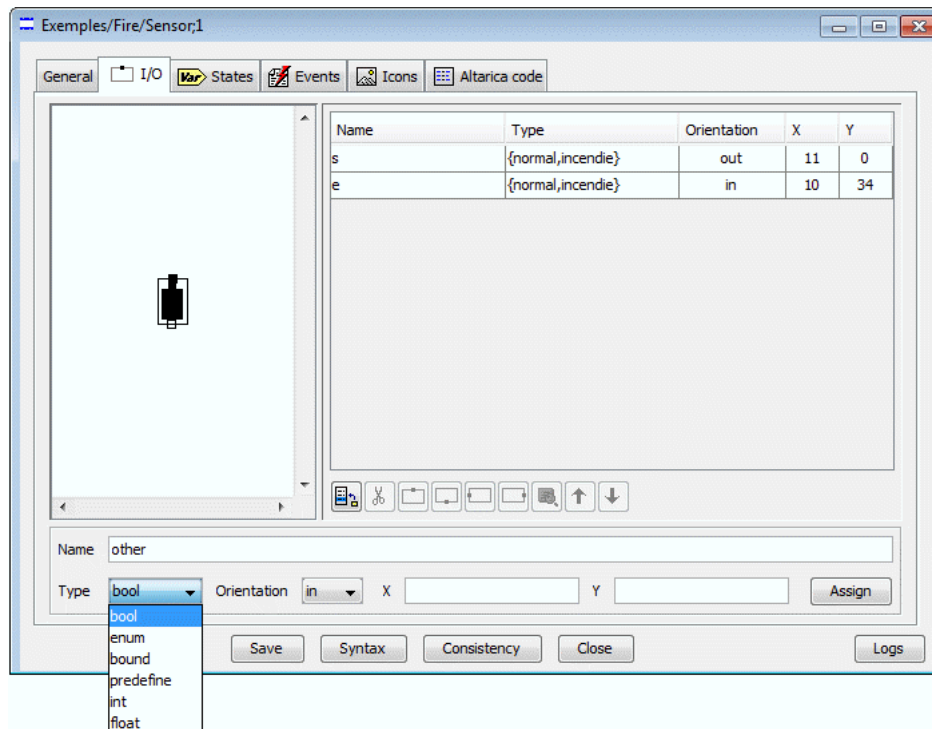
A component may be un-modifiable, It happens when another user as locked this component, or if component is opened from the simulation view. In these cases, the **Save** button is replaced by a **Save as** button



The Save As button is available in models Editors as soon as models can't be modified

### 6.2.5.2. I/O tab

The **I/O** flow variables are the support for the information flows exchanged between the different components of the system.




- In the **Name** field, type the **I/O** name.
- Use the **Type** menu and choose the variable type:
  - **bool** : boolean variable, containing the constants true and false;
  - **enum** : enumerate variable; values must be separated by commas without spaces, *e.g.*: null,low,high.



The "space" character is forbidden when keying in the enumerate variable.





- **bound** : variable whose value varies between a minimum value and a maximum value,
- **predefine** : predefined type in library
- **int** : integer variable
- **float** : float variable

Click on the  icon or the Enter key; the I/O appears on the component border (the default type is "bool").

The line corresponding to the typed I/O is added in the I/O table with the following information (Name, Type, Orientation, X, Y); X and Y indicate the coordinates of the input (in) or output (out) port location on the icon with respect to an origin at the top left-hand side; the default orientation is "in".

To modify the orientation of an I/O, click with the mouse left button and then the right button on the Orientation cell; in the pull-down menu, assign the flow orientation (in or out).

Position the **I/O** on the model by clicking on the corresponding icon:


- Top:  icon;
- Down:  icon;
- Left:  icon;
- Right:  icon;



An input is identified by a white rectangle and an output by a black one.



Repeat the same procedure for any additional I/O.

If necessary, select the I/O and click on the  icon to delete any selected I/O

Three other icons are available:



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To move down in the list.




To move up in the list.



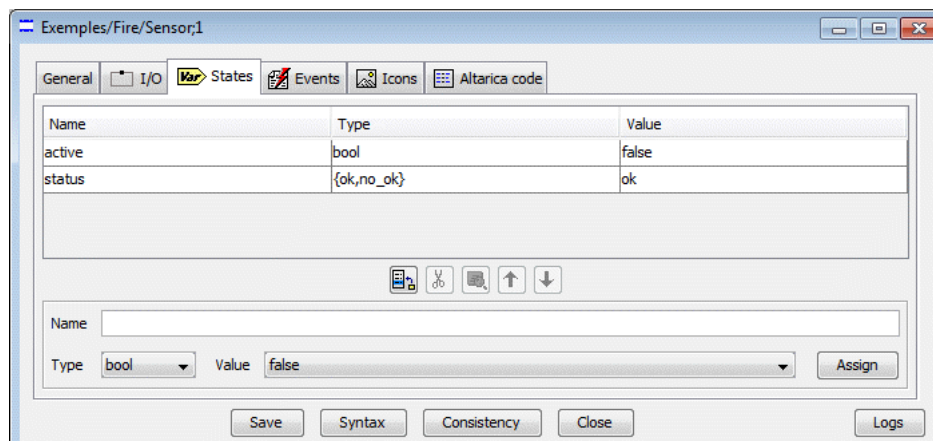
To edit the properties of the selected event.




If an I/O is selected, the I/O added with the  icon is positioned before the selected I/O. If no I/O is selected, the I/O is positioned at the end of the list.

To modify the type of an existing I/O, select the I/O; choose the type and click on the **Assign** button; the type corresponding to the selected I/O is updated.

### 6.2.5.3. State tab




The **States** tab allows typing of a component states:

- In the **Name** field, type the name of the state variable.
- Use the **Type** menu and choose the variable type:
  - **bool**: boolean variable,
  - **enum**: enumerate variable, *e.g.* ok,failure,
  - **predefine** : predefined type in library. For the version 3.0 only the enumerate types must be used for the state variables.
- Click on the  icon or the Enter key.

As shown in the previous figure, the state is displayed on a line which indicates: its Name (Status), the values domain of its Type (OK,KO) and its default Value (OK). The OK value is the one taken by default before the simulation, during the initialization of the state variable.



If one of state variables is selected, the added state variable (with the  icon) is positioned before this selected variable. If no state variable is selected, the added state variable is positioned at the end of the list.




To modify the type of an existing state variable, select the state variable: select a new type for this variable and click on the Assign button to associate the selected type to the state variable.



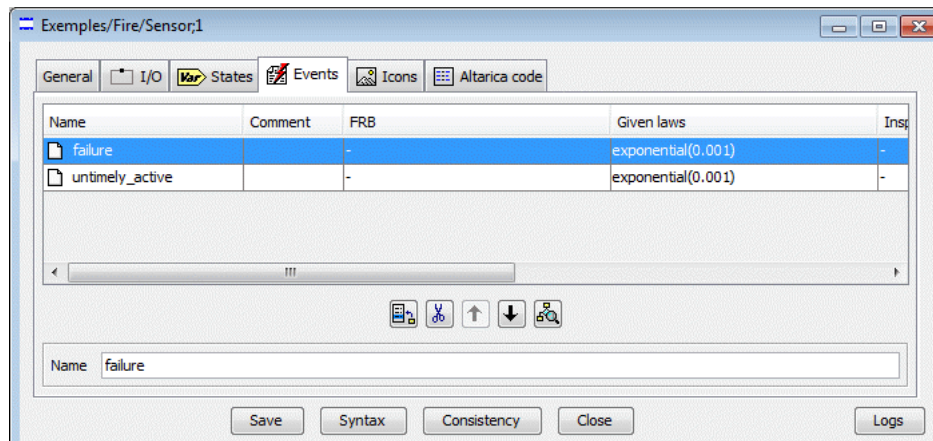
To modify the name of an existing state, click on the Name cell and modify the name and validate with the Enter key.

Click on the Value cell of the selected state variable to modify, if necessary the default value.

Three other icons are available:

- Edit the preset type of the state variable: use the icon  or do a left double click on the cell Line Type of the selected state variable,
- Move to the top of the line of the selected state variable:  icon;
- Move to the bottom of the line of the selected state variable: .

## 6.2.5.4. Events tab



The **Events tab** allows typing of component events:

- In the **Name** field, type the event name.
- To modify the name (or law) of an existing event, double click on the Name (or Law) cell; modify the name and validate the new name (or law) with the Enter key



If an event is selected, the added event is positioned before the selected event. If no event is selected, the added event is positioned at the end of the list



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



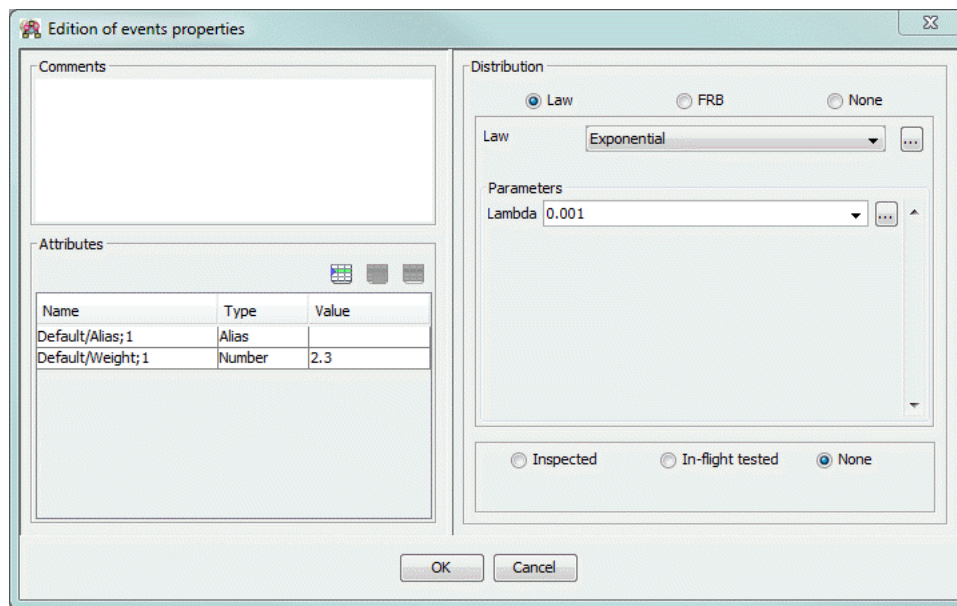
To move down in the list.



To move up in the list.

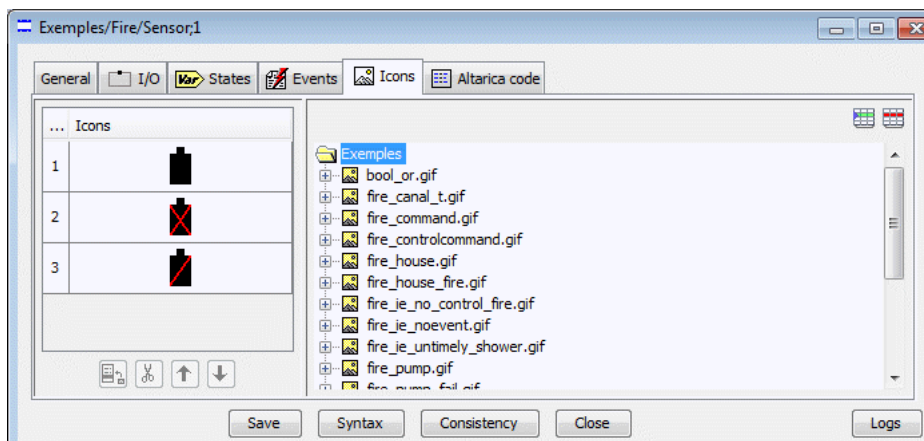


To edit the properties of the selected event.



This window is described in Section 4, “Common approach”

## 6.2.5.5. Icons tab







The Icons tab allows selection of the icon(s) which graphically describe(s) the component behavior during the simulation of the architecture system.

In the right tab, there is the icon manager described in Section 6.2.2, “Icon manager”

In the left part, there is a preview of the icon used in the item.

- To add an icon in the list:
  - Select the desired icon in the icon manager.

Click on  to transfer the selected icon from the right to the left part of the tab which displays at the end of the list the icon number (used in the Altairica code), and the icon graphical representation

- Click on  , to remove the selected icon from the left side of the tab.
- By using the two icons (  and  ) we can move the selected icon upwards or downwards and thus modify the sequence order in the icon list.



Within the Altarica code, the icon displayed for a given state is identified by a number. Thus, modifying the icons order in the list induces indirectly a change of icon in the Altarica code.



The selection of the icon graphical representation or its number in the left part of the tab, automatically selects in the right part of the tab the name of the icon in the corresponding directory.

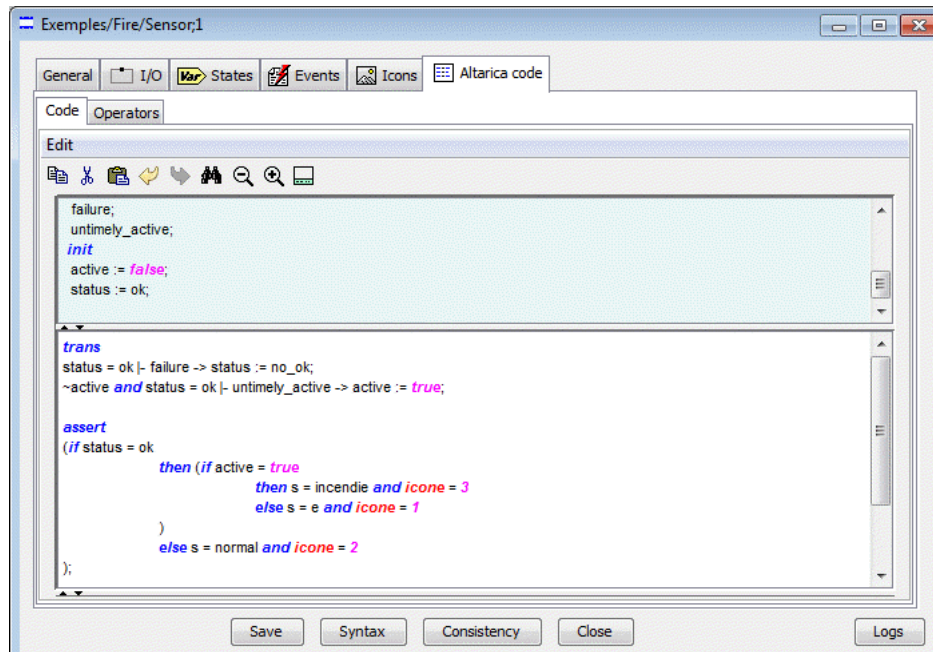


When the component doesn't change its graphical representation, do not create icons in the icons tab: select the icon only in the **General** tab.



It is recommended that all the icons of the same component have the same size during the graphical simulation of the state changes.

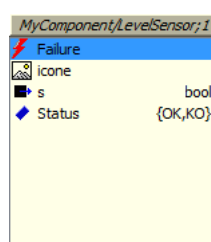
### 6.2.5.6. Altarica code tab



The Altarica code tab allows writing of the Altarica code which conditions the component operation (cf. Appendix E, *Language AltaRica*).

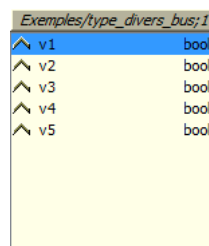
The Edition area comprises from top to down:

- An icon bar (Copy, Cut, Paste, Find/Replace, Zoom out, Zoom in, Display generated External Clauses).
- The initial states display area is non-editable and the icon is used to display/mask this area.
- A data summary area (non-editable) concerning the current component (node name, icons used, input and output variables as well as their type (boolean,...), the state variables as well as their type, the events, icon variable [1,3]).
- The Altarica code typing area (for the assertions (assert) and transitions (trans)); the typing order for these assertions or transitions is indifferent.
- The short cut Ctrl+<SPACE> gives access to the variables selector for the search and selection of the state variables and the flow variables.



In order to select a variable, proceed as follows:

- Click in the filling zone of the Altarica code,
- Use the shortcut Ctrl+<SPACE> to activate the variables selector and obtain the list of component variables,
- Double-click on the desired variable, the name <Variable\_Name> will then be inserted in the text editor at the current position of the cursor,
- For a flow variable with a Record type, the selection of the flow parameters is carried out in the following way:
- Type the character "^" (Alt Gr+<^>) following the name of the variable <I/O\_Name>^, (eg. eb^), to reveal the list of the associated flow parameters:



Double-click on the desired parameter, the name of the parameter will then be inserted in the text editor at the current position of the cursor: <I/O\_Name>^<Flow\_Name> (ex. eb^p1).

Click on this selection using the right button of the mouse; a vertical window appears indicating the two possible values of "state\_": ok or stuck\_open.



The color code for the Altarica code is the following:

- Keyword (for example: trans, assert, and, or, if, then, else...): blue,
- Numerical values: purple,
- Operators name: green,
- Icons: red,
- Punctuation in bold.



If operators are used in Altarica code, Operators tab is accessible (cf. Section 6.2.4, “Operator management”). It allows selecting operator family and operator version, if different operators have the same name. If there is no ambiguity, the only family containing the operator is selected with the last version.

Finally click on the button Syntax, to do the syntax control of the written Altarica script. In the event of a syntactic error, an error window appears describing precisely the kind of error and indicating the line of the script.

In the event of a **consistency** error, an error window appears describing precisely the kind of consistency error of the Altarica code.

The consistency problems detected are:

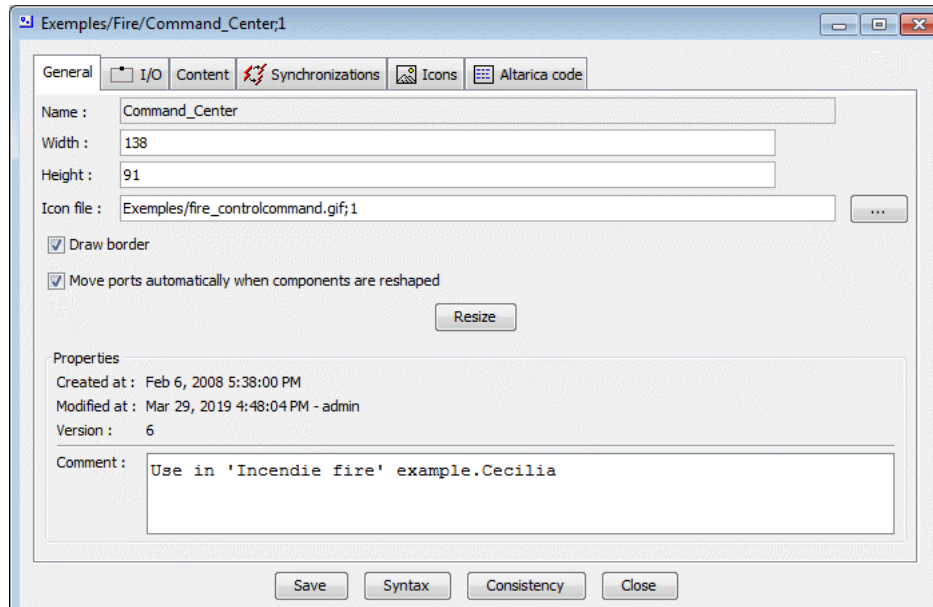
- An exit flow variable is not defined whatever the internal state.
- There is an internal state and a combination of input flow variables in which an exit flow variable is not defined.
- There is an internal state and a combination of input flow variables in which an exit flow variable has 2 different evaluations.

## 6.2.6. Equipment management

The Editor model window has the following tabs:

- **General,**
- **I/O,**
- **Content,**
- **Synchronizations,**
- **Icons,**
- **Altarica code.**

### 6.2.6.1. General tab



The **General tab** allows the following information to be typed in:

- **Name:** type the model name,



In order to save the model, the name is the only obligatory field.

- **Width:** ll (icon width default value),
- **Height:** hh (icon height default value),
- **Icon file:** the button enables selection of the icon to be assigned to the model.



Icons are image files in "\*.gif" or "\*.jpg" format.

- Tick **Draw border** to display or not the border associated to the icon.
- Tick **Move ports automatically when components are reshaped**.
- Click on the **Resize** button to validate, if necessary, the icon larger dimensions.
- **Comment:** fill the comment describing the created component.

All the model editor tabs contain four common buttons:

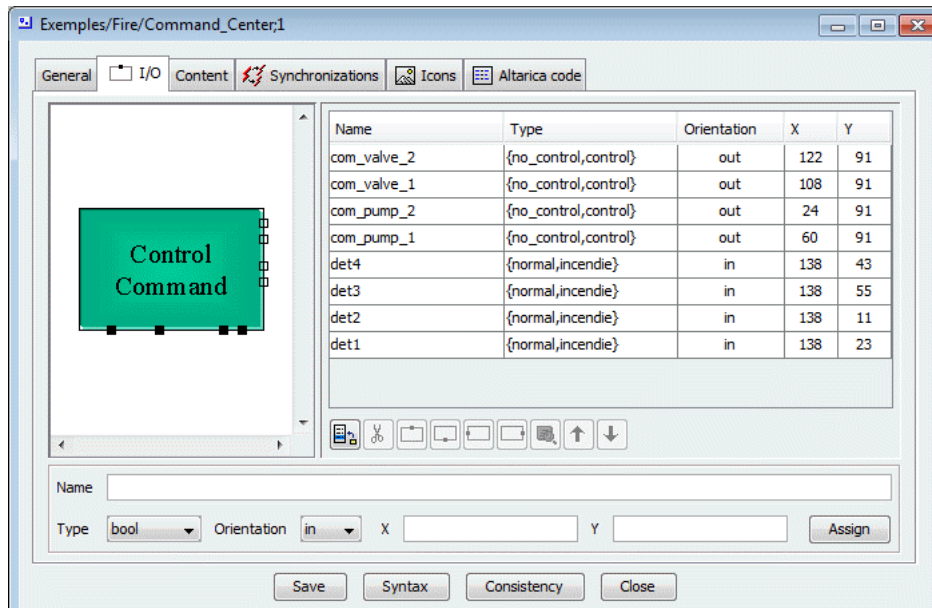
- **Save:** allows storing of all typed information,
- **Syntax:** allows a syntax control on the Altarica code,
- **Consistency:** allows a consistency control on the Altarica code,
- **Close:** allows closure of the model editor.

Two icons on the top right hand side allow window size modification:

- : to reduce the window;
- : to extend the window.

### 6.2.6.2. I/O tab

The I/O (Inputs/Outputs) tab allows configuration of the input or output flows within an equipment.




- In the **Name** field, type the **I/O** name.
- Use the Type menu and choose the variable type:
  - **bool** : boolean variable, containing the constants true and false;
  - **enum** : enumerate variable; values must be separated by commas without spaces, *e.g.*: null,low,high.



The "space" character is forbidden when keying in the enumerate variable.





- **bound** : variable whose value varies between a minimum value and a maximum value,
- **predefine** : predefined type in library
- **int** : integer variable
- **float** : float variable

Click on the  icon or the Enter key; the I/O appears on the component border (the default type is "bool").

The line corresponding to the typed I/O is added in the I/O table with the following information (Name, Type, Orientation, X, Y); X and Y indicate the coordinates of the input (in) or output (out) port location on the icon with respect to an origin at the top left-hand side; the default orientation is "in".

To modify the orientation of an I/O, click with the mouse left button and then the right button on the Orientation cell; in the pull-down menu, assign the flow orientation (in or out).


Position the **I/O** on the model by clicking on the corresponding icon:

- Top:  icon;
- Down:  icon;
- Left:  icon;
- Right:  icon;



An input is identified by a white rectangle and an output by a black one.

Repeat the same procedure for any additional I/O.

If necessary, select the I/O and click on the  icon to delete any selected I/O



Three other icons are available:



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To move down in the list.




To move up in the list.



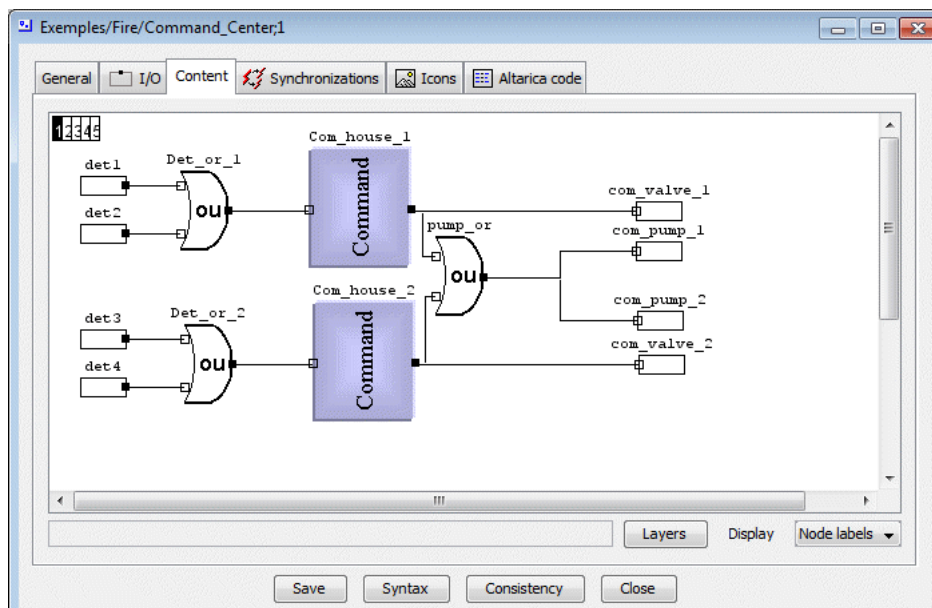
To edit the properties of the selected event.



If an I/O is selected, the I/O added with the  icon is positioned before the selected I/O. If no I/O is selected, the I/O is positioned at the end of the list.

To modify the type of an existing I/O, select the I/O; choose the type and click on the **Assign** button; the type corresponding to the selected I/O is updated.

## 6.2.6.3. Content tab



When an equipment is created, it contains only the Input/Output nodes; the latter can be positioned anywhere inside the equipment.

The Content tab indicates the equipment internal architecture; modelling an equipment is identical to modelling a system architecture.

Connections between the Inputs/Output nodes of the equipment and the I/O components it contains are identical to the connections between components.



When equipment is indexed in architecture and has:

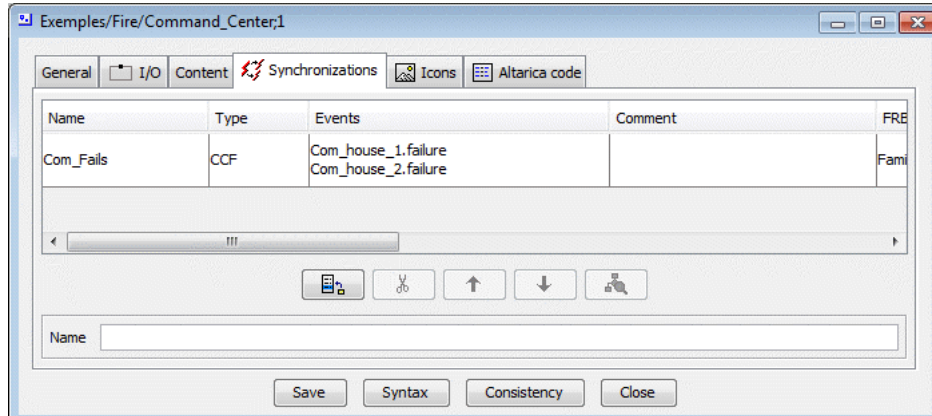
1. A non-empty content, it cannot be modified in the architecture,
2. An empty content, it can be modified in the architecture.

- The Layers button allows users to control the display of different graphical elements
- The Display button allows users to control the display of different architecture type.




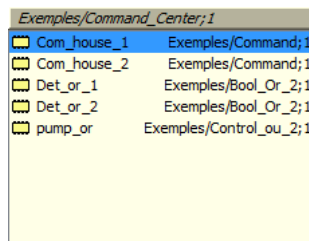
- None: no name is displayed,
- Node Labels: all the instance names are displayed above the components and equipment,
- Link labels: the type transported by the link is displayed at the center of the link for all links
- All labels: Display "Node labels" and "Link labels".

#### 6.2.6.4. Synchronization tab



The **Synchronization** tab allows specifying synchronizations whose. The activation of these synchronizations during the simulation phases will allow synchronizing the occurrence of certain number of component events defined in the equipment architecture ( **Content** tab).

- Fill the synchronization name in the line editor of the Synchronizations area.
- Click on  , to transfer the name in the synchronizations list defined for the equipment.
- Fill in the line editor from the field Law (using a law cf. Appendix A, *Laws and uncertainties* or a FRB cf. Section 4.3, “Event model (Failure Rate Base - FRB)” )
- Chose the synchronization type:
  - CCF (Common Cause Failure): either events appears individually (failure without common cause), or events that can appear at the same time appear at the same time.
  - Diffusion: all events that can appear at the same time appear at the same. If the event cannot appear, it will not impact. In this synchronization type, the events cannot appear individually.
  - Synchronization: events can only appear at the same time. In this synchronization type, the events cannot appear individually.
- Fill the events list having to be synchronized, using the line editor associated to the event zone. In order to facilitate the filling of this assertion, the short cut Ctrl+<SPACE> gives access the hierarchical navigator for the search and selection of component events.



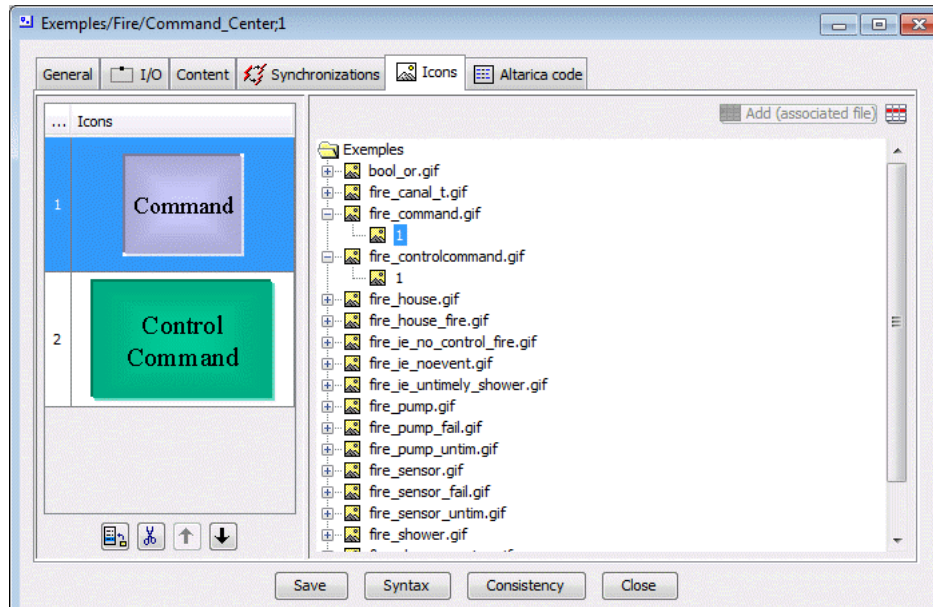
- Double-click on the selected event; it will then be inserted in the events list having to be synchronized.

To complete the events list will be synchronized, repeat the same procedure.



During the system simulation phases, synchronizations will be added at the list of events.

### 6.2.6.5. Icons tab







The Icons tab allows selection of the icon(s) which graphically describe(s) the component behavior during the simulation of the architecture system.

In the right tab, there is the icon manager described in Section 6.2.2, “Icon manager”

In the left part, there is a preview of the icon used in the item.

- To add an icon in the list:
  - Select the desired icon in the icon manager.

Click on  to transfer the selected icon from the right to the left part of the tab which displays at the end of the list the icon number (used in the Altarica code), and the icon graphical representation

- Click on , to remove the selected icon from the left side of the tab.
- By using the two icons (  and  ) we can move the selected icon upwards or downwards and thus modify the sequence order in the icon list.



Within the Altarica code, the icon displayed for a given state is identified by a number. Thus, modifying the icons order in the list induces indirectly a change of icon in the Altarica code.



The selection of the icon graphical representation or its number in the left part of the tab, automatically selects in the right part of the tab the name of the icon in the corresponding directory.

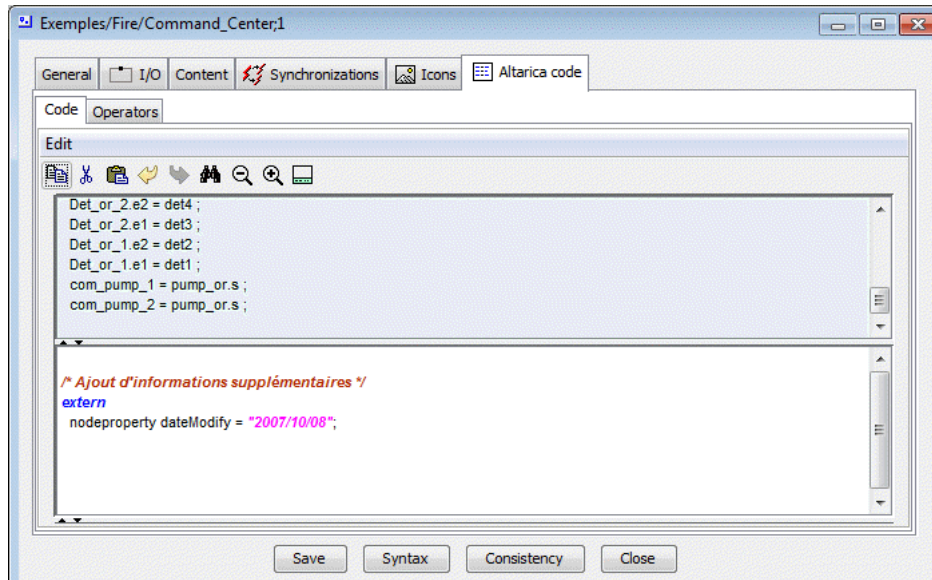


When the component doesn't change its graphical representation, do not create icons in the icons tab: select the icon only in the **General** tab.



It is recommended that all the icons of the same component have the same size during the graphical simulation of the state changes.

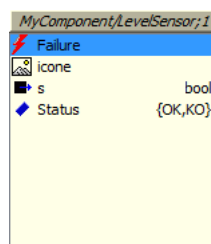
## 6.2.6.6. Altarica code tab



The Altarica code tab allows writing of the Altarica code which conditions the component operation (cf. Appendix E, *Language AltaRica*).

The Edition area comprises from top to down:

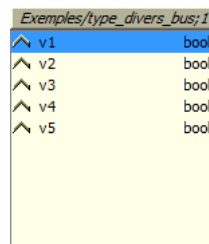
- An icon bar (Copy, Cut, Paste, Find/Replace, Zoom out, Zoom in, Display generated External Clauses).
- The initial states display area is non-editable and the icon is used to display/mask this area.
- A data summary area (non-editable) concerning the current component (node name, icons used, input and output variables as well as their type (boolean,...), the state variables as well as their type, the events, icon variable [1,3]).
- The Altarica code typing area (for the assertions (assert) and transitions (trans); the typing order for these assertions or transitions is indifferent.
- The short cut Ctrl+<SPACE> gives access to the variables selector for the search and selection of the state variables and the flow variables.



In order to select a variable, proceed as follows:

- Click in the filling zone of the Altarica code,
- Use the shortcut Ctrl+<SPACE> to activate the variables selector and obtain the list of component variables,
- Double-click on the desired variable, the name <Variable\_Name> will then be inserted in the text editor at the current position of the cursor,
- For a flow variable with a Record type, the selection of the flow parameters is carried out in the following way:

- Type the character "^" (Alt Gr+<^>) following the name of the variable <I/O\_Name>^, (eg. eb^), to reveal the list of the associated flow parameters:



Double-click on the desired parameter, the name of the parameter will then be inserted in the text editor at the current position of the cursor: <I/O\_Name>^<Flow\_Name> (ex. eb^p1).

Click on this selection using the right button of the mouse; a vertical window appears indicating the two possible values of "state\_": ok or stuck\_open.



The color code for the Altarica code is the following:

- Keyword (for example: trans, assert, and, or, if, then, else...): blue,
- Numerical values: purple,
- Operators name: green,
- Icons: red,
- Punctuation in bold.



If operators are used in Altarica code, Operators tab is accessible (cf. Section 6.2.4, “Operator management”). It allows selecting operator family and operator version, if different operators have the same name. If there is no ambiguity, the only family containing the operator is selected with the last version.

Finally click on the button Syntax, to do the syntax control of the written Altarica script. In the event of a syntactic error, an error window appears describing precisely the kind of error and indicating the line of the script.

In the event of a **consistency** error, an error window appears describing precisely the kind of consistency error of the Altarica code.

The consistency problems detected are:

- An exit flow variable is not defined whatever the internal state.
- There is an internal state and a combination of input flow variables in which an exit flow variable is not defined.
- There is an internal state and a combination of input flow variables in which an exit flow variable has 2 different evaluations.

## 6.2.7. Model management

The Editor model window has the following tabs:

- **Content,**
- **Synchronizations,**
- **Altarica code.**
- **Initial configuration,**
- **Modified Events,**
- **Links colors,**
- **Properties,**

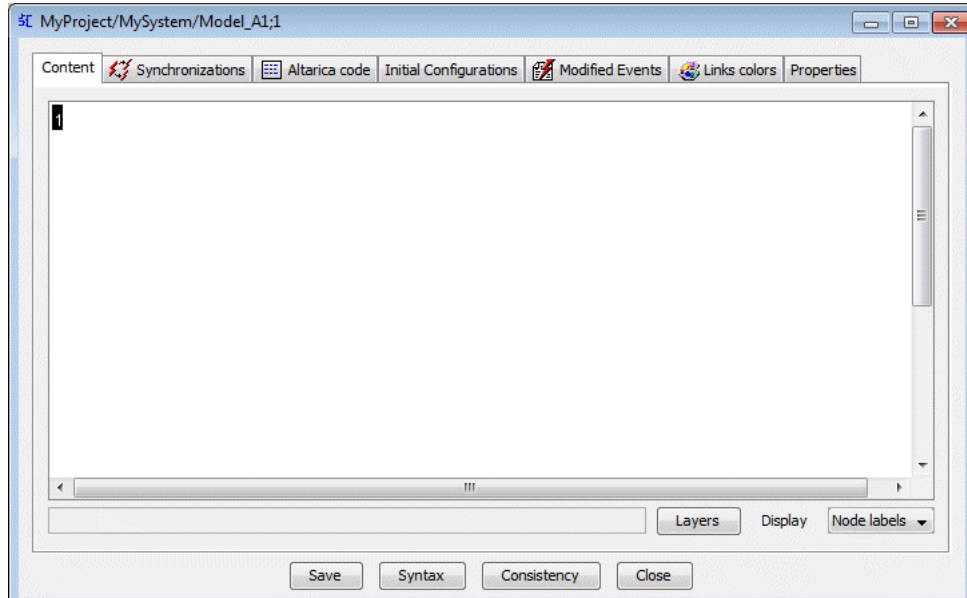
### 6.2.7.1. Content tab

#### 6.2.7.1.1. Create a model

In order to create a model architecture by using components and equipment in library, proceed as follows:

- In the Projects tab, create a project, for example "MyProject" and then create a system, *e.g.* "MySystem" inside this project (see Section 3, “Technical object”).

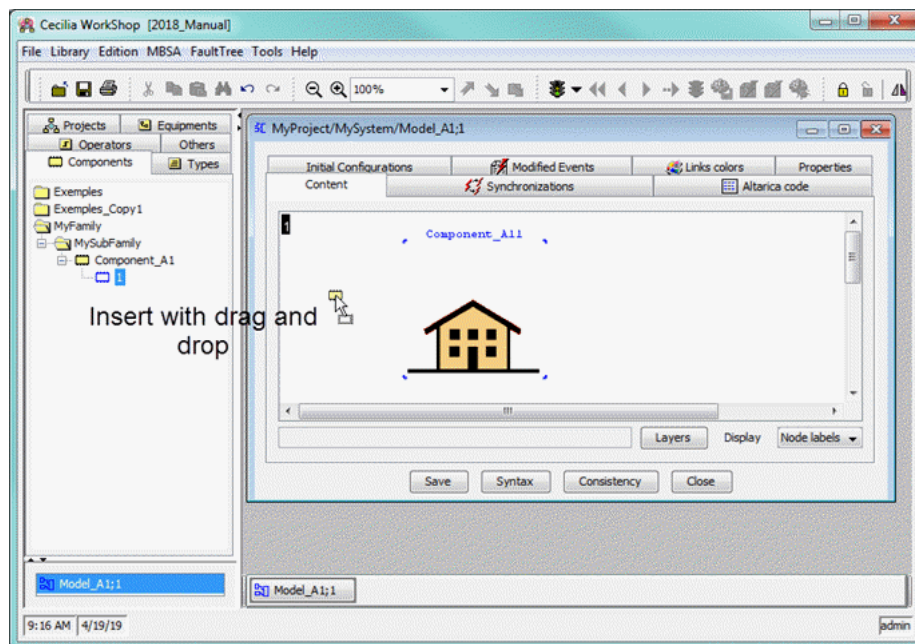
- A double click in the version numero displays the work area.



## 6.2.7.1.2. Insertion of a component in the architecture

In order to insert a component in the architecture, proceed as follows:

- Click on the Components tab and select the component, *e.g.* "Component\_A1" in order to build up the model architecture (see Section 6.2.5, "Component management" to create a component).
- Double click on the component "Component\_A1" to access the versions tree structure.
- Select the selected component version and drag and drop in the work area.



The same way is used to insert equipment.


When they are inserted, components and equipment are created with a default name:

- <component model name>xx: for a component name, where xx is an increasing number;
- <equipment model name>yy: for an equipment name, where yy is an increasing number.

The Edit model command starts reading of the model and enables access to the various tabs of the Model editor concerning the selected component or equipment.

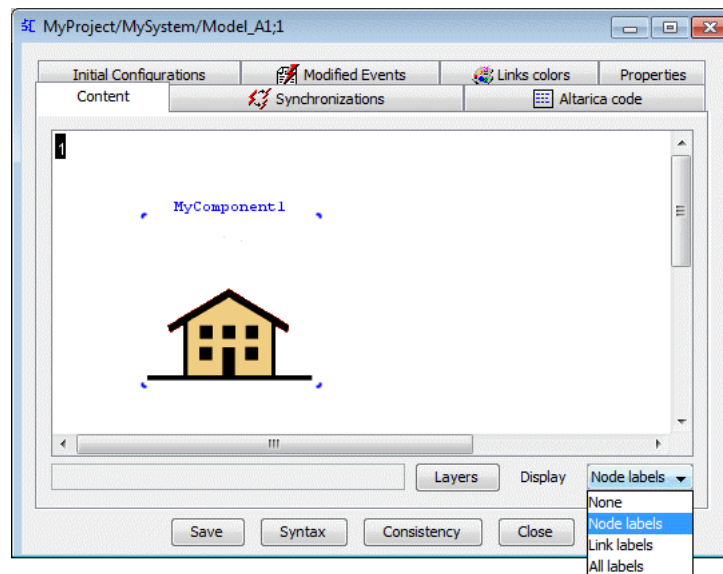



A double click on a component or equipment allows the editing of its model without using the contextual menu.

The **Rename** command (  ) or a double click with the mouse left button on the component or equipment name when the Labels option of the Display menu is selected, enable renaming a component or an equipment.

Use the Display menu to define the display type:

- None: no name is displayed;
- Node Labels: all the instance names are displayed above the components and equipment;
- Link labels: the type transported by the link is displayed at the center of the link for all links;
- All labels: Display "Node labels" and "Link labels".



The **Replace** command (  ) or a double click with the mouse left button on the component or equipment name when the Models option of the Display menu is selected, enable replacement of a model by a model already existing in the library.






If the new model has the same interfaces (inputs/outputs with same names and types) as the previous model, the links eventually present in the hardware architecture will be saved, otherwise they are deleted.

The **Cut**, **Copy** and **Paste** commands allow the corresponding operations on equipment, a component or a link. For components and equipment, the instance name of the newly created object follows the rules of the action of adding a component or equipment.

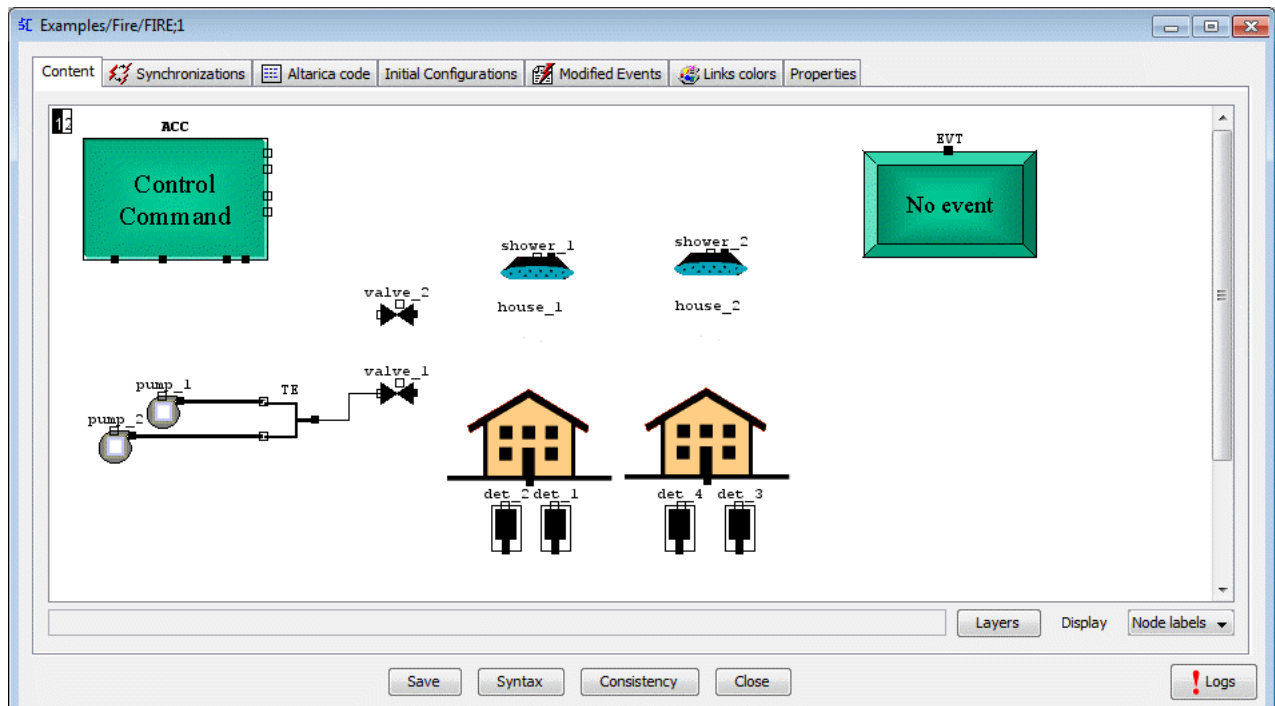
The **Lock** command enables locking of a component or of a link in architecture to avoid its displacement; the selection edges of a component which were in red color appear grey. If in an architecture only one component is locked, unlock it and then lock the overall architecture.

In an architecture, the equipment name appears in bold characters which enable the use of the following icons or commands:




-  : move up in an architecture to display a higher level architecture;
-  : move down in an architecture to display a lower level architecture;
-  : move down to another view to display a lower level architecture in a new window.



Carry out the same transfer procedure for all the components or equipment making up the system architecture:



Use the following design commands of the contextual menu (with the mouse right button) or the icons of the Design tool bar to optimize the layout of components or equipment in the workarea:

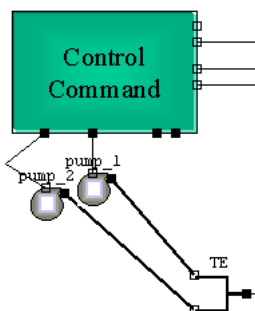
- Vertical mirror: ;
- Horizontal mirror: ;
- 90 rotation: .

Maintaining Left-Click on an element allows moving it into the workspace. It can also be made with the keyboard arrows (dot par dot).

### 6.2.7.1.3. Link two components

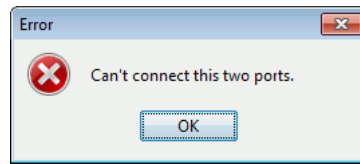
In order to link two components via their communication ports:

- Click on the output black square of one and establish a link to the input white square of the other one.
- When the link is established, if necessary, click on the link and move it; this entails appearance of a breakpoint (white color: selection and black color: non-selection).





This link procedure checks the compatibility of the connected inputs/outputs (direction: in/out; type: inverse/normal/cross field).

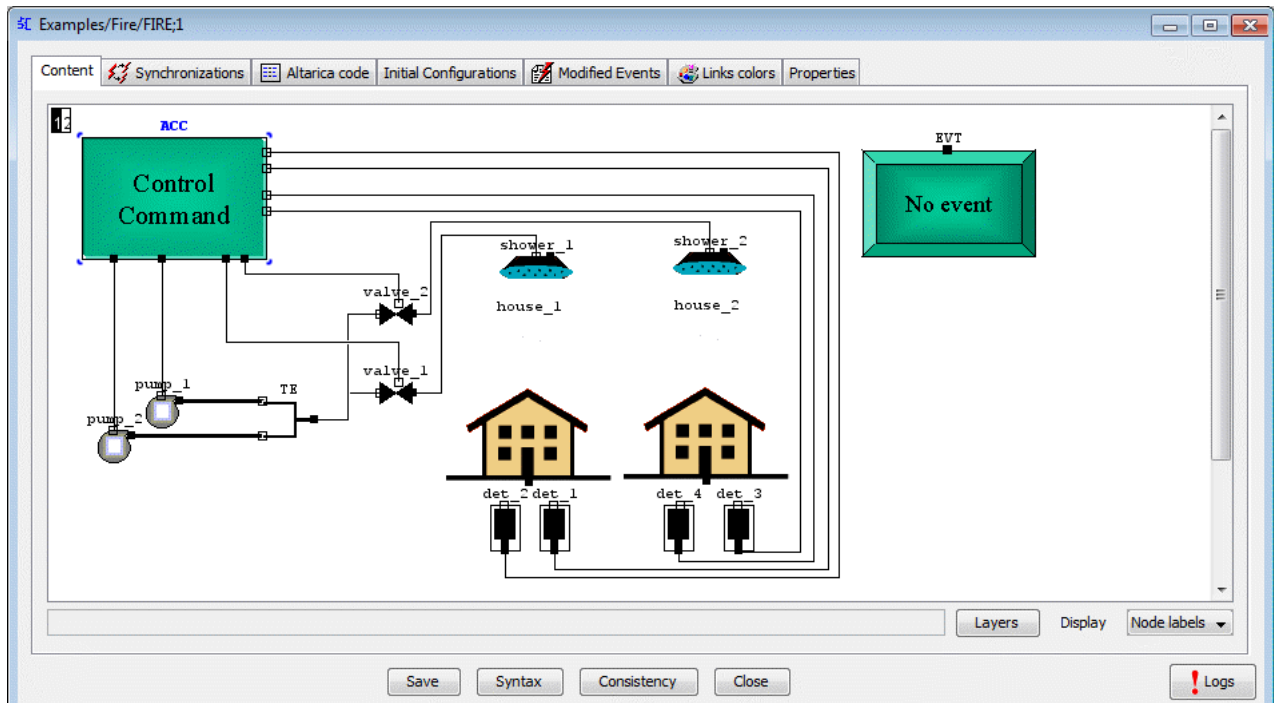


Once link is made, if necessary, click and move the link, it creates a break-point (white: selected, black: not selected). Create break-point to avoid link intersection.

Click on a link and use the Link command of the contextual menu (with the mouse right button), or the icons of the Link tool bar, to optimize the path of the links between the components or equipment:

- Direct:
- Right angle:
- Right angle from right to bottom:
- Right angle from left to bottom:
- Right angle from right to top:
- Right angle from left to top:
- Possibility of positioning an arrow on a link in order to visualize the direction of a flow (2 arrows in the event of bidirectional flow:

The overall system architecture with its entire links is displayed in the work area:

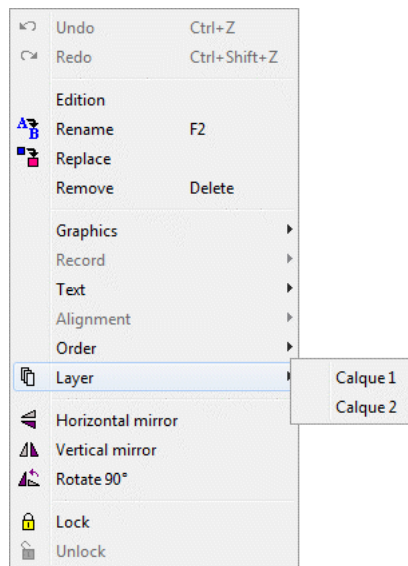


## 6.2.7.1.4. Utilization of display layers

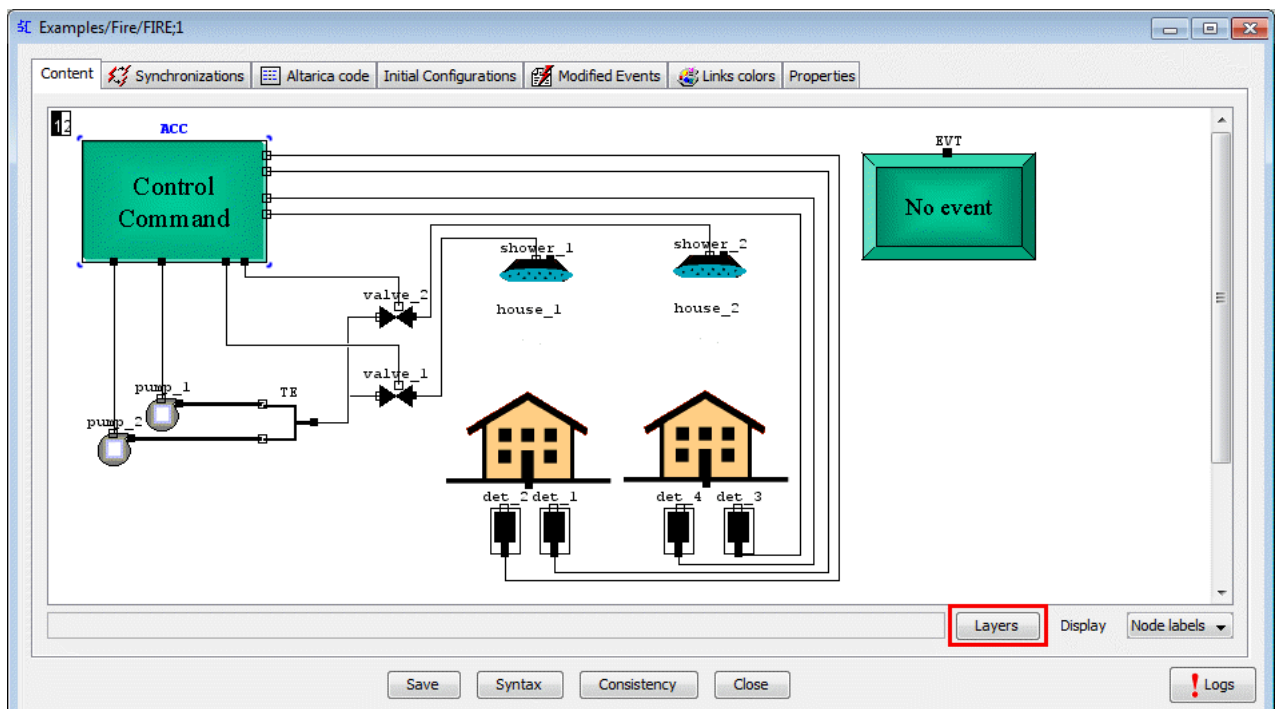
The user has 5 display layers. Graphical objects representing components, equipment, system architecture links are put into these layers according to display properties.

In order to put graphical elements in the different display layer, user must proceed as follows:

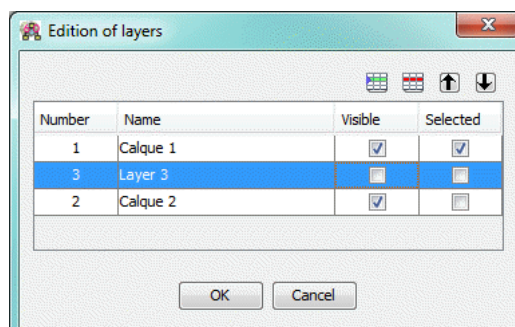
- Select the graphical element (component, equipment, link),
- In contextual menu (right click) select the layer on which the element must be put.



In order to control the elements display, the user must click on **Layers** button to display Definition of display Layers window.



With the window of layer definition, display priority order can be defined for the 5 layers in which the user has put equipment, components or links.





This button can be used to add a new item.



This button can be used to delete an item.



To move down in the list.



To move up in the list.

**Order:** the user can select the layers display priority (1:foreground layer, to 5:background layer).

**Name** field: double-click on the area in order to edit the layer name. The default name is "Layer n".

**Visible** layers: the user can choose if a layer is visible or not

**Selected** layer in architecture edition: this area is made of a check box whose selection shows the active layer choice in edition of system architecture. Components or equipment inserted in system architecture during edition will be put on this layer.

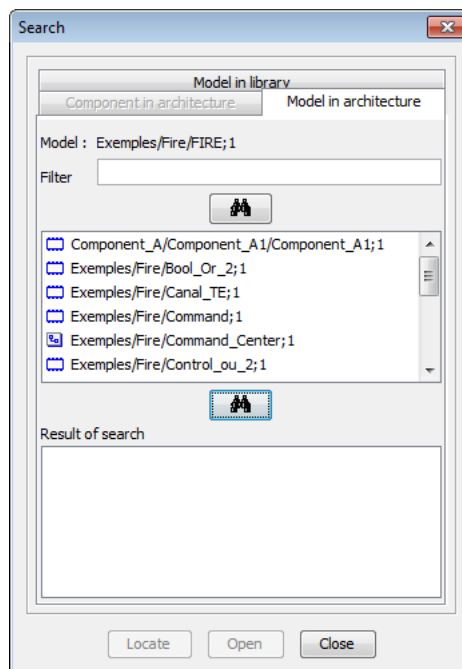
The final system visibility is made by the 5 layers superimposition according to their priority.

## 6.2.7.1.5. Search a component model in a modelisation

Search for a component as follows:

- Use the **Edition - Search** menu or type the **CTRL + F** command on the keyboard or use the icon in the Standard tool bar.

The Search window is displayed.

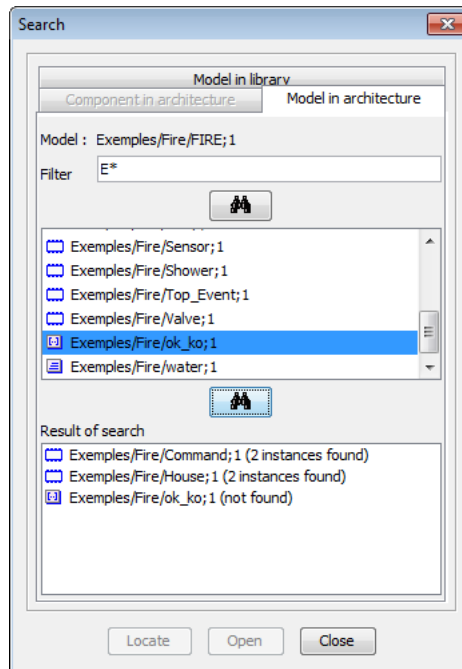


- Click on the Component tab and in the Filter field, type the component name as follows: <Family\_Name>/<Model\_Name>

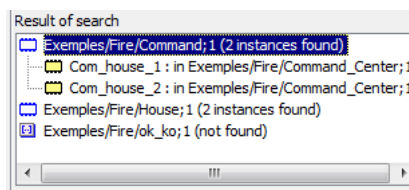


the character "\*" can be used to decrease the search space (example: H\*/val\*) or to complete a partially-known-name (examples: HYD\*/val\*, \*/valve).

- Click on the icon, the component searched for (or the components having a name corresponding to the filter used) is displayed in the middle part of the window.



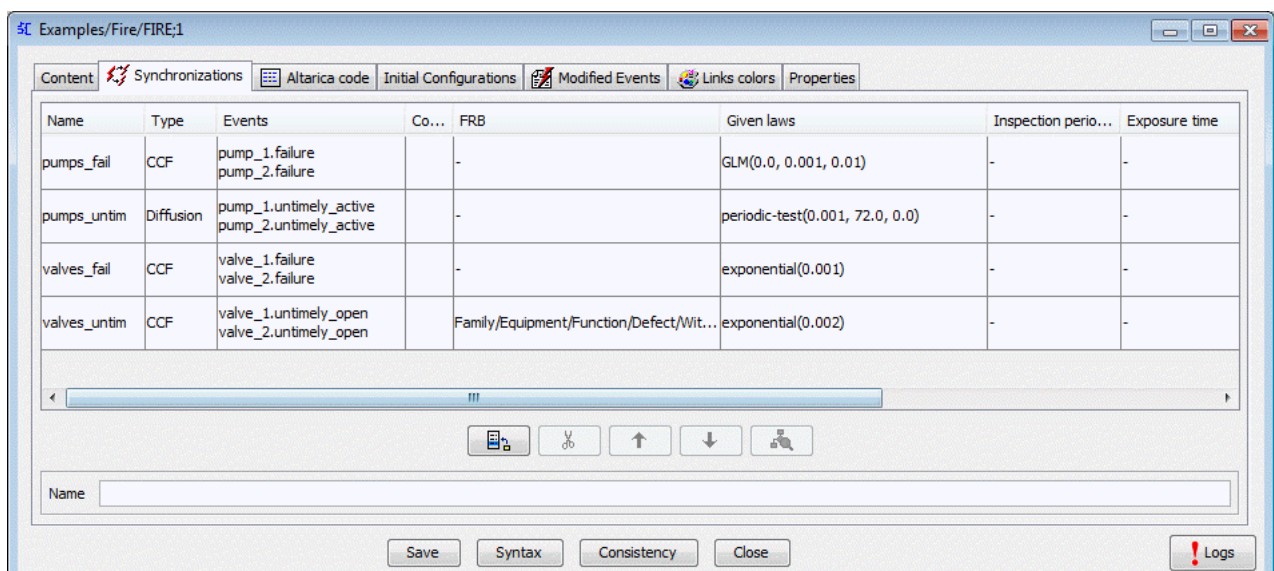
- Select the searched model, click on the second icon, the search model and the number of found instances appear at the bottom (result of search).
- Double click on the model to unroll the list of model instances in the system. It provides information's about localization.





- Select an instance, click on the Locate button in order to move the view on this instance.

## 6.2.7.2. Synchronization tab

When this menu is selected the following window appears:



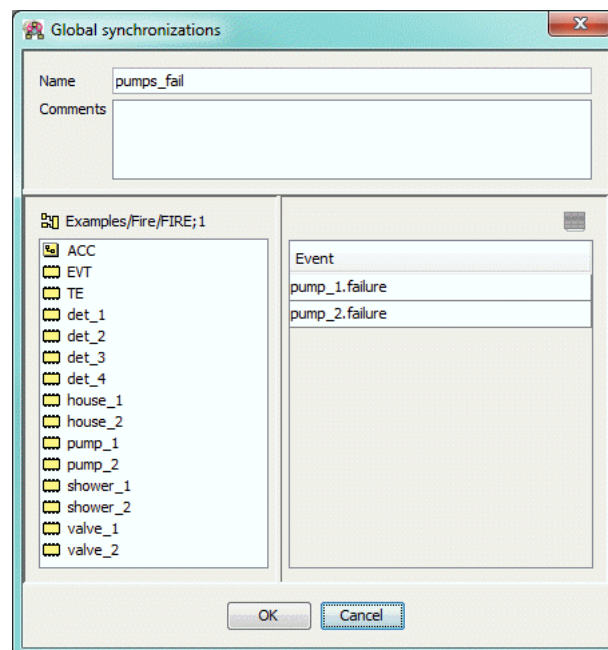
The activation of these synchronizations during the simulation phases will allow synchronizing the occurrence of certain number of component events defined in the architecture. For example, when the event "pumps\_fail" occurs, Pump 1 and Pump2 will be down in the same time.

- Fill the synchronization name in the line editor of the Synchronizations area.
- Click on , to transfer the name in the synchronizations list defined for the equipment.
-  opens the windows describes in Section 3, "Technical object"



A double click on **Law** column or **FRB** column allows also the define a law (see Appendix A, *Laws and uncertainties* or a FRB (see Section 4.3, "Event model (Failure Rate Base - FRB)")

- Chose the synchronization type:
  - **CCF (Common Cause Failure)**: either events appears individually (failure without common cause), or events that can appear at the same time appear at the same time.
  - **Diffusion**: all events that can appear at the same time appear at the same. If the event cannot appear, it will not impact. In this synchronization type, the events cannot appear individually.
  - **Synchronization**: events can only appear at the same time. In this synchronization type, the events cannot appear individually.
- Fill the **events** list having to be synchronized, using the line editor associated to the event zone. In order to facilitate the filling of this assertion, the short cut Ctrl+<SPACE> gives access the hierarchical navigator for the search and selection of component events.



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To move down in the list.

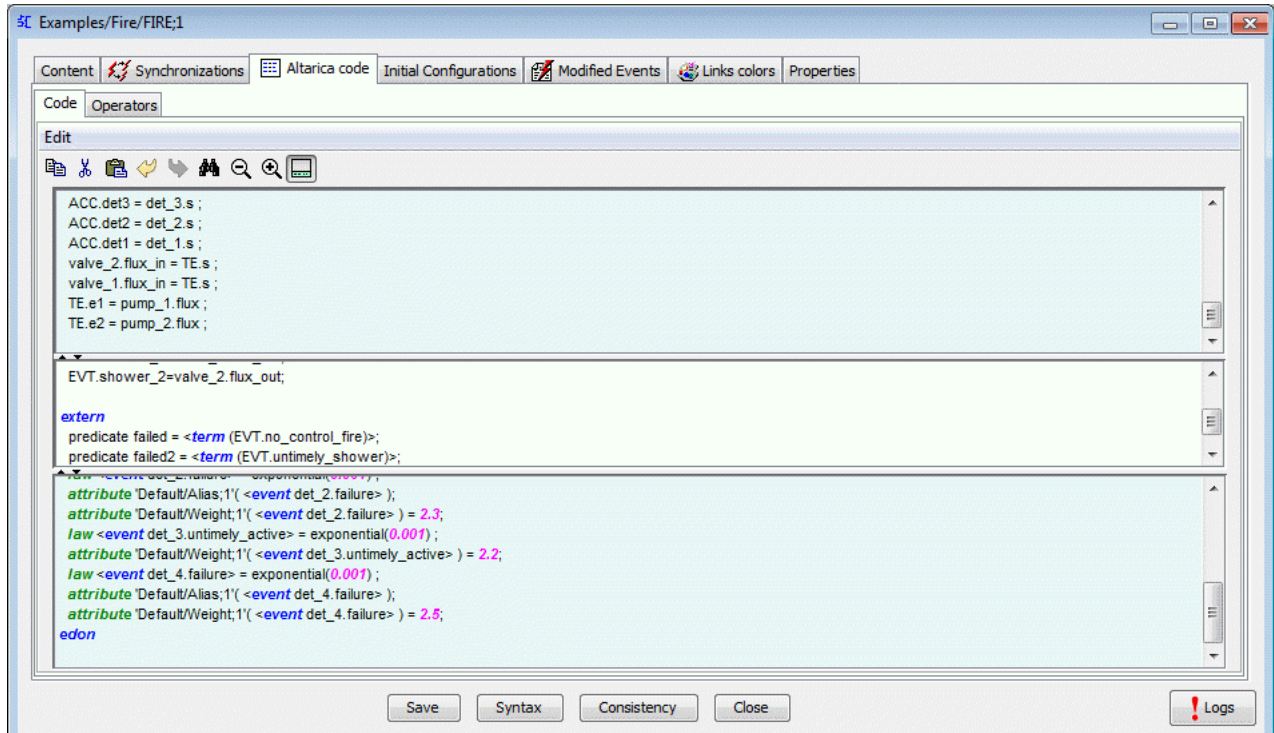


To move up in the list.



To edit the properties of the selected event.

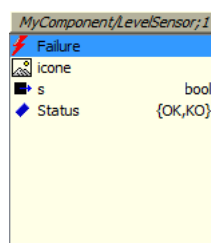
### 6.2.7.3. AltaRica tab



The AltaRica code tab allows writing of the AltaRica code which conditions the component operation (cf. Appendix E, *Language AltaRica*).

The Edition area comprises from top to down:

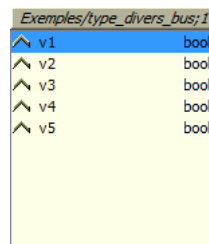
- An icon bar (Copy, Cut, Paste, Find/Replace, Zoom out, Zoom in, Display generated External Clauses).
- The initial states display area is non-editable and the icon is used to display/mask this area.
- A data summary area (non-editable) concerning the current component (node name, icons used, input and output variables as well as their type (boolean,...), the state variables as well as their type, the events, icon variable [1,3]).
- The AltaRica code typing area (for the assertions (assert) and transitions (trans); the typing order for these assertions or transitions is indifferent.
- The short cut Ctrl+<SPACE> gives access to the variables selector for the search and selection of the state variables and the flow variables.



In order to select a variable, proceed as follows:

- Click in the filling zone of the AltaRica code,
- Use the shortcut Ctrl+<SPACE> to activate the variables selector and obtain the list of component variables,
- Double-click on the desired variable, the name <Variable\_Name> will then be inserted in the text editor at the current position of the cursor,
- For a flow variable with a Record type, the selection of the flow parameters is carried out in the following way:

- Type the character "^" (Alt Gr+<^>) following the name of the variable <I/O\_Name>^, (eg. eb^), to reveal the list of the associated flow parameters:



Double-click on the desired parameter, the name of the parameter will then be inserted in the text editor at the current position of the cursor: <I/O\_Name>^<Flow\_Name> (ex. eb^p1).

Click on this selection using the right button of the mouse; a vertical window appears indicating the two possible values of "state\_": ok or stuck\_open.



The color code for the Altarica code is the following:

- Keyword (for example: trans, assert, and, or, if, then, else...): blue,
- Numerical values: purple,
- Operators name: green,
- Icons: red,
- Punctuation in bold.



If operators are used in Altarica code, Operators tab is accessible (cf. Section 6.2.4, "Operator management"). It allows selecting operator family and operator version, if different operators have the same name. If there is no ambiguity, the only family containing the operator is selected with the last version.

Finally click on the button Syntax, to do the syntax control of the written Altarica script. In the event of a syntactic error, an error window appears describing precisely the kind of error and indicating the line of the script.

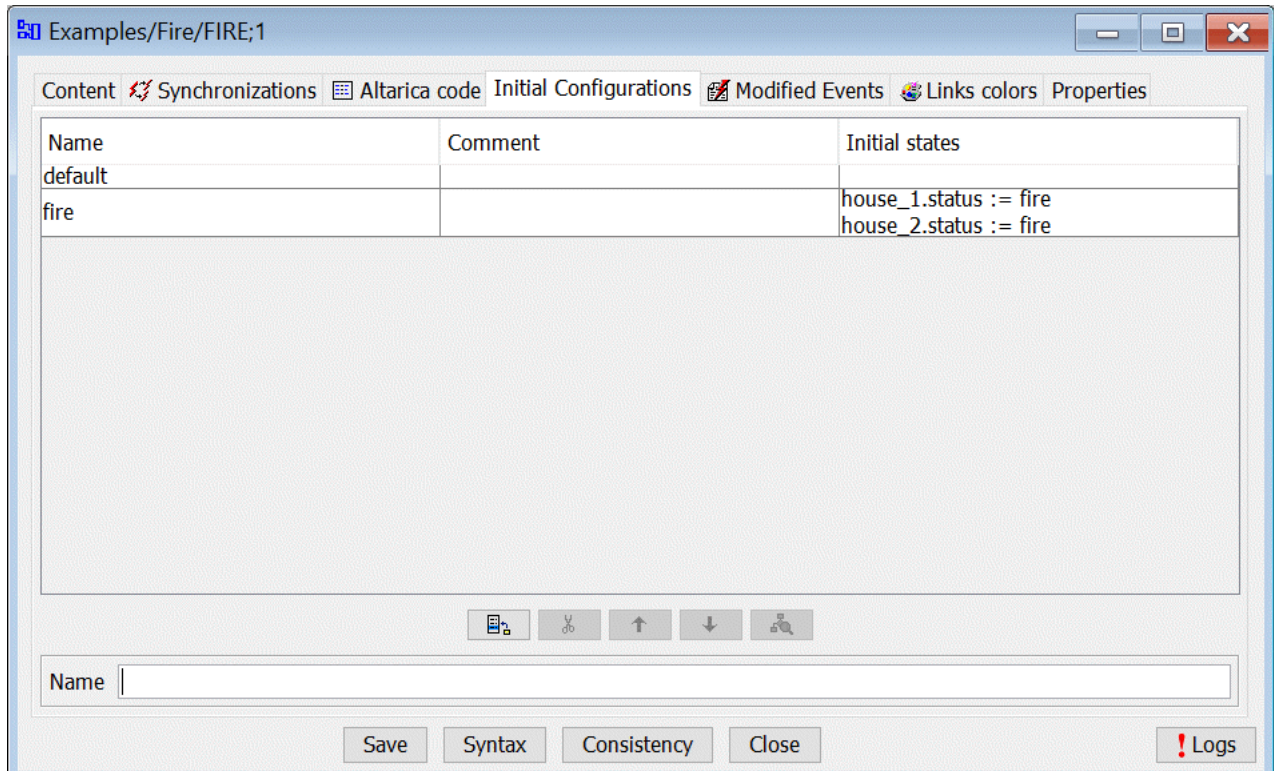
In the event of a **consistency** error, an error window appears describing precisely the kind of consistency error of the Altarica code.

The consistency problems detected are:

- An exit flow variable is not defined whatever the internal state.
- There is an internal state and a combination of input flow variables in which an exit flow variable is not defined.
- There is an internal state and a combination of input flow variables in which an exit flow variable has 2 different evaluations.

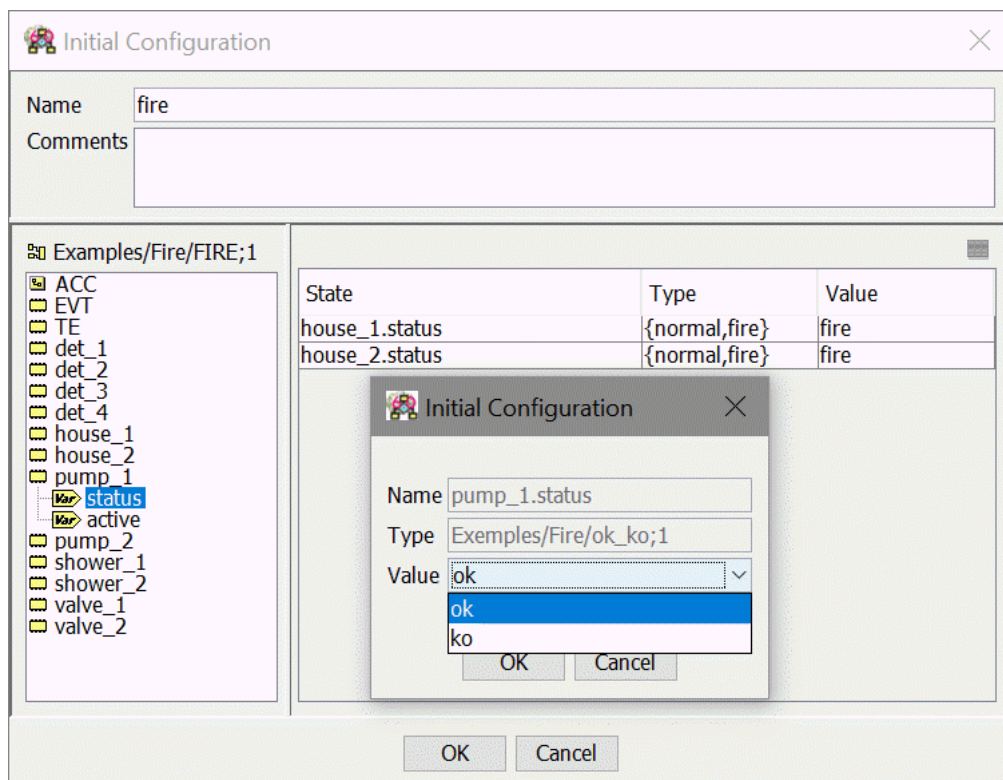


## 6.2.7.4. Initial Configurations tab




In order to modify initial state of a variable, double-click on this variable or click on . It will display state variable modification window.

- In the **Name** field, type a name for new configuration (name of the new initial state of the system) then click on ; is displayed in **Configuration** frame.
- A double-click on the configuration column or on icon allows to open the configuration window to select the state of the various variables;



Select the variable, with a double-click a windows appears to choose the variable value. When the configuration is selected, a click on **OK** saves the configuration.

Select a previously saved configuration as follows :

- Select a configuration (name of system initial state previously saved).
- Click on  icon to load configuration. Values of state variables are displayed in the left frame : **Initial states**.

The means of the icons are:



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To move down in the list.



To move up in the list.



To edit the properties of the selected event.



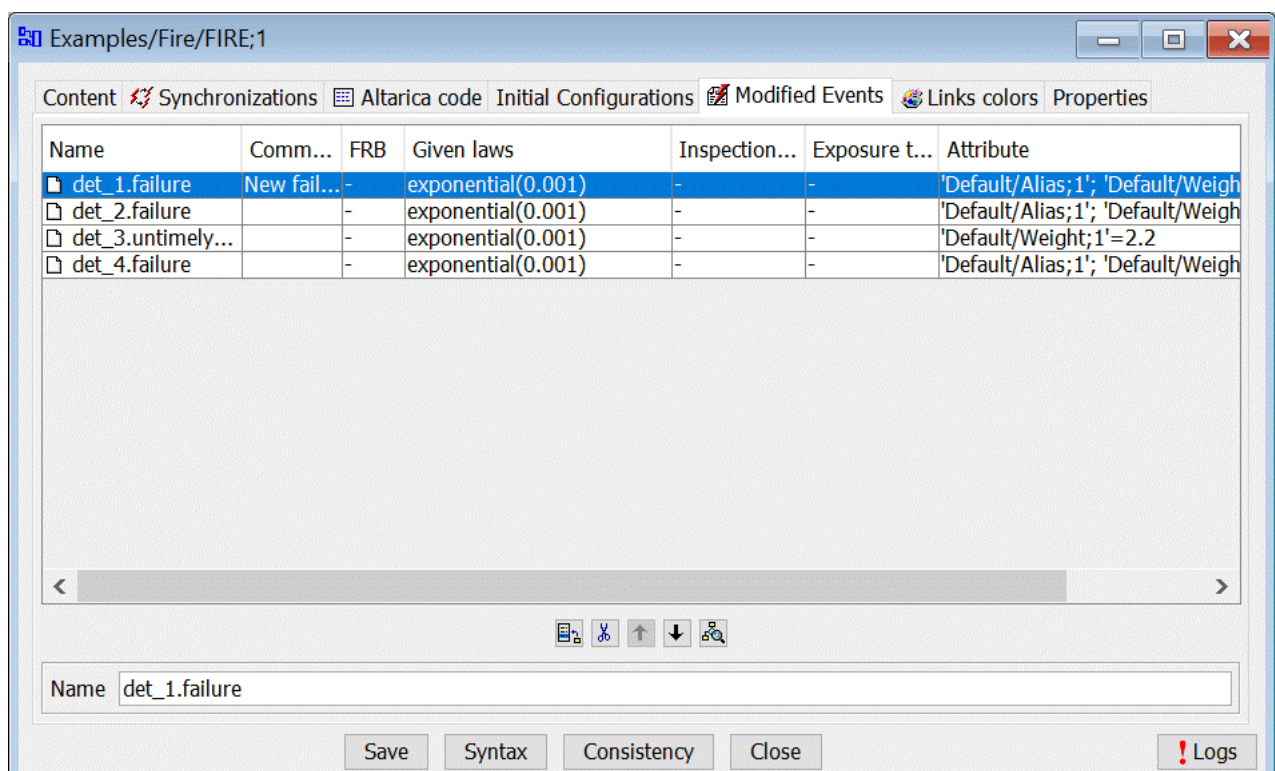
These configurations could be used for simulation, and don't depend on type of simulation. The choice of configuration will be made at simulation launch.




A new configuration can be defined automatically during simulation , see Section 6.4.7.1, “Initial configuration saving during simulation”.


## 6.2.7.5. Modified Events tab

Events tab enables to modify events.



In order to add a new events, fill the event name in the line editor.

Then, click on , to transfer the name in the event list.

Double click on an event column (except name column) or use  button, it displays event properties window. The window describes in Section 3, “Technical object” appears to set the selected event.

The means of the icons are:



This button can be used to add a new item.



It is also possible use the Enter key.



This button can be used to cut or delete an item.



To move down in the list.



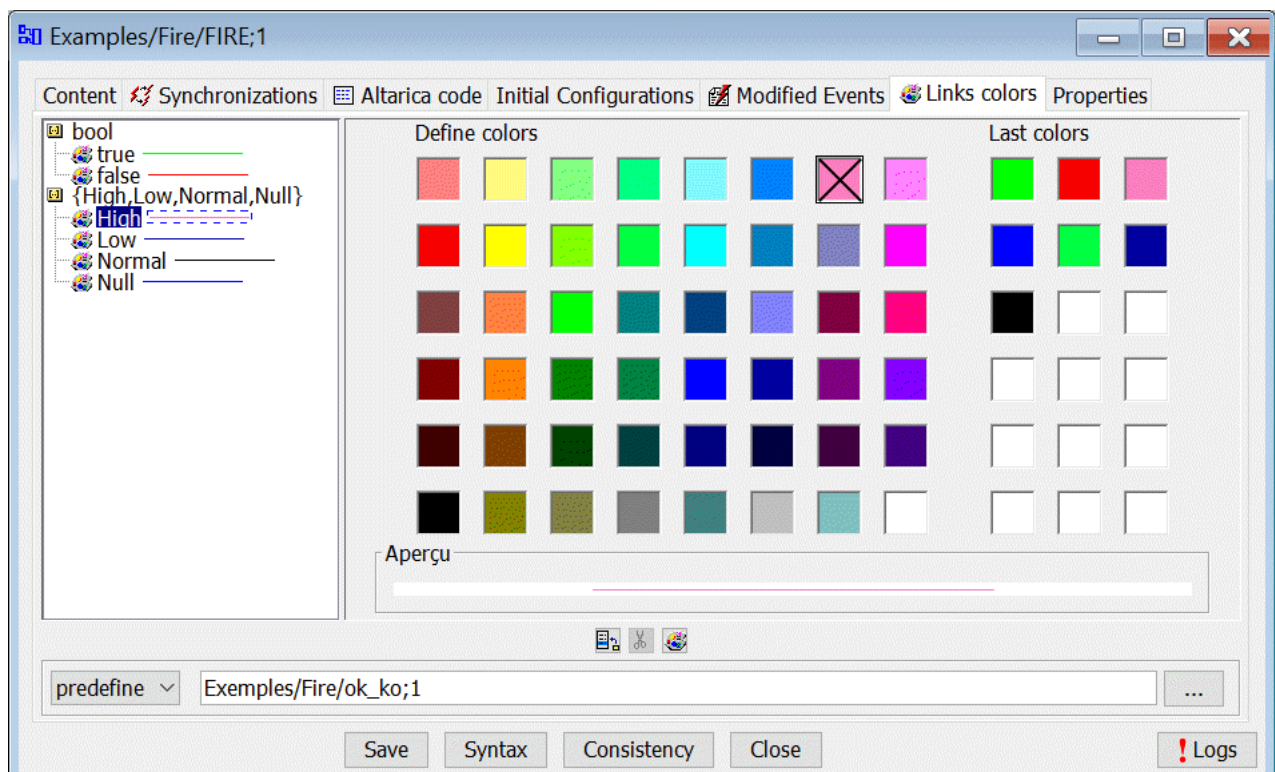
To move up in the list.



To edit the properties of the selected event.

## 6.2.7.6. Links colors tab

The Links colors tab is:





This tabs contains in its right part a color palette, and in its left part, an area displaying the link colors arranged according to their type.

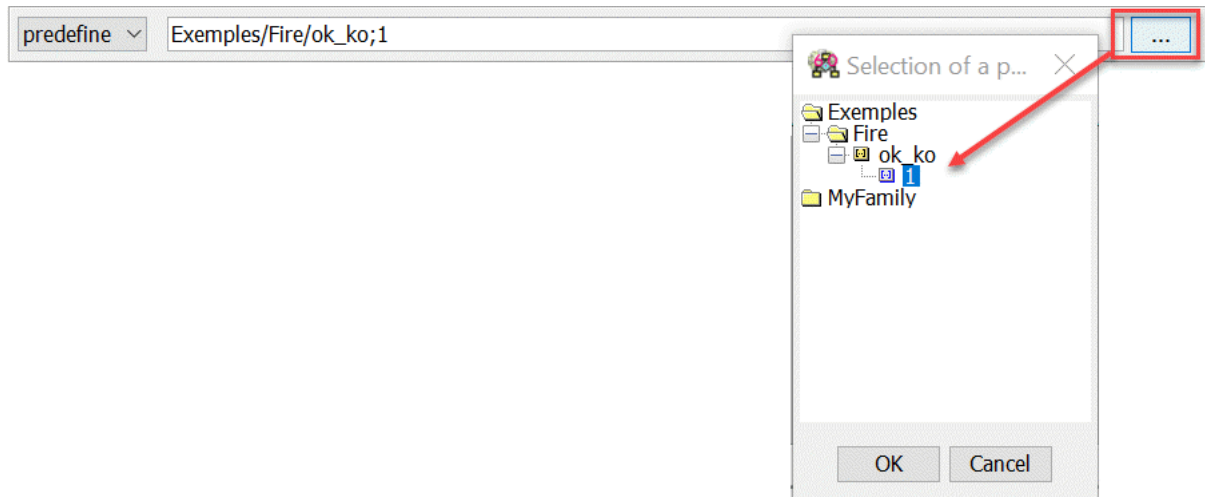
When a new architecture is created, the boolean type is the default type with two colors (modifiable):

- True: green;
- False: red.

If a new architecture contains links (non-boolean) which must be colored, carry out the following steps.


- Use the type selection button associated with the link (enum, bound or predefined). Only the predefined and enumerate types are explained.
- Fill in the name field and click on the  icon or use the Enter key.

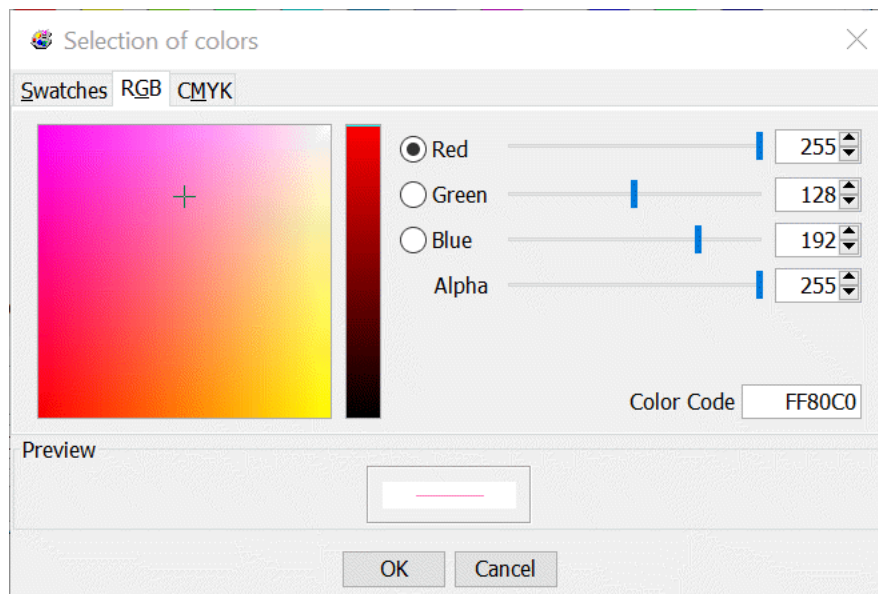
To select a predefined type variable, use the  to select the predefined variable in a new window:






It is possible to create an enum by typing the possible value separated by comma:



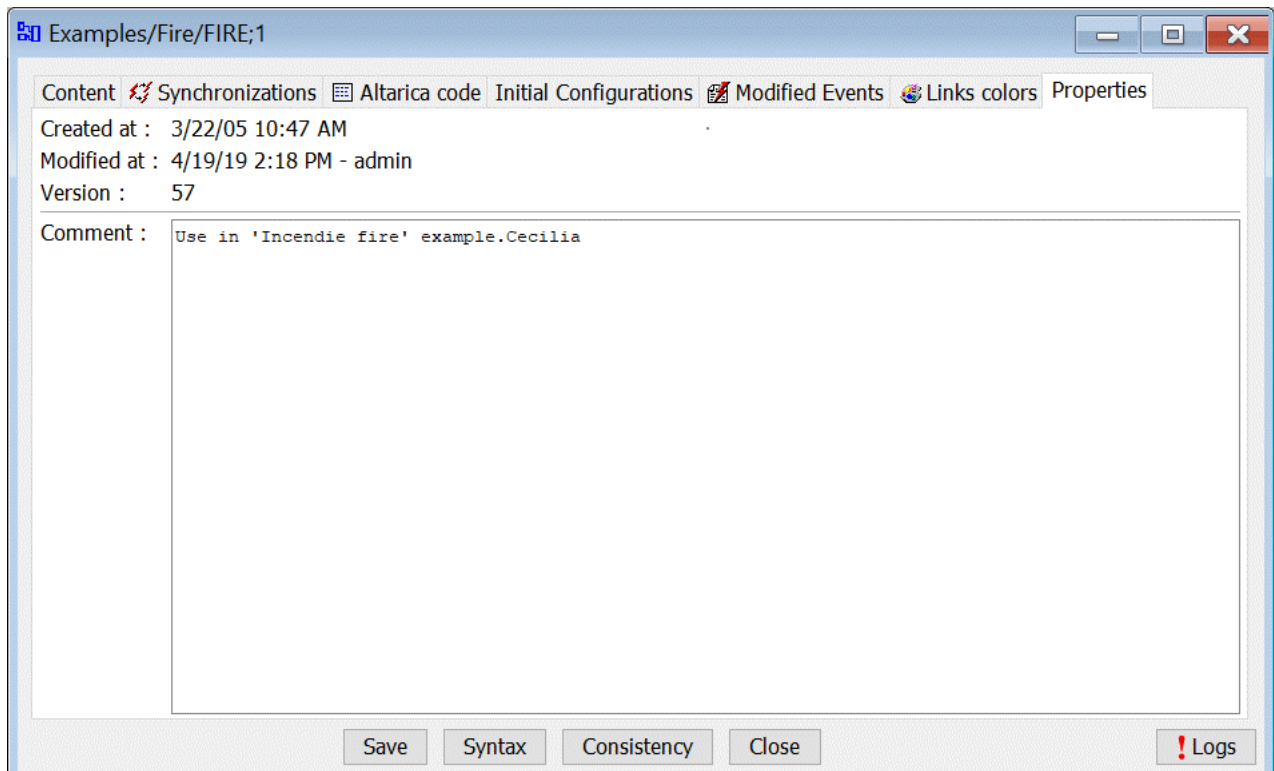
- Double click on an element, *e.g.*, pressure; the four possible values for pressure links are displayed:
  - High;
  - Low;
  - Normal;
  - Null.
- Select a value, *e.g.* Normal.
- In the color palette, select a color; the Recent area on the right indicates the last selected colors.
- Click on the  icon to display new colors:





-  The color assignment is specific for a system; the colors are saved with the system.
-  An existing enumerated type can be deleted by clicking on the corresponding icon.
- Display the results of this color assignment during system simulation (see Section 6.4, “Simulation Step by Step”).
-  When a link is of the record type, the colors applicable to the link are those associated to the type of the first parameter in the record.

## 6.2.7.7. Properties tab



- The Properties tab allows visualization of the model properties:
  - Creation date,
  - Last modification date,
  - Version.
- The **Comment** field allows the filling of a comment describing the operator function.

## 6.3. Verify or Translate model

Cecilia-Workshop enables to imitate AltaRica model. For treatment part, it generates an AltaRica Extended-format file. Translator plugins takes this file in input and generates a file in another format which is more workable for other plugins.

Currently available formats are:

- AltaRica DataFlow: This format is widely use in AltaRica environment
- AltaRica Mec5: A model-checking tool (Made by the LaBRI : Laboratoire Bordelais de Recherche en Informatique ; 'Inventor' of AltaRica language)
- Moca12 or Moca13: Stochastic simulator base on predicates Petri nets (Property of TOTAL corp.)
- Other specific format used inside Cecilia-Workshop whorkshop.

These translators are based on modules allowing reading of files in AltaRica Extended format (so it enables to verify syntax and semantics), allowing conversion of assertions to dataflow equations, allowing setting flat of a model, allowing elementary properties check, ...

Some plugins are made in order to interface the following functionalities in Cecilia-Workshop:

- **Syntactical checker** verifies that code is complying with AltaRica Extended format.
- **Property Control** verifies model semantics, and its setting flat. It also verifies elementary properties like external clauses validation or loop in assertions.
- **Translation** translates model in predetermined format (model can possibly be anonymized for confidential reasons).
- **External tools** launch an external tool for current model possibly converted by a translator.


## 6.3.1. Syntactical check

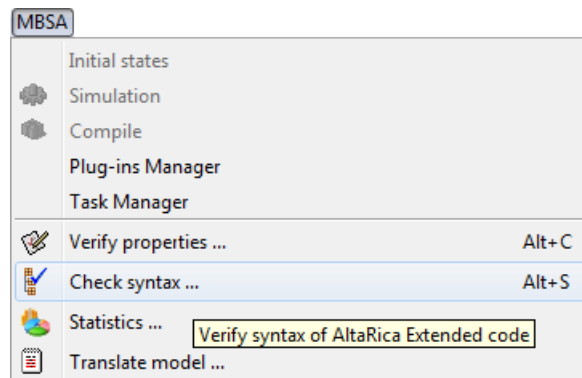
Syntactical check verifies that components, equipments and/or systems are consistent with AltaRica Extended's syntax.

This function will display error if AltaRica code generated by software (for flow, states, events,...) or typed by user (guard, transitions, ...) isn't correct.

### 6.3.1.1. Syntactical check launching

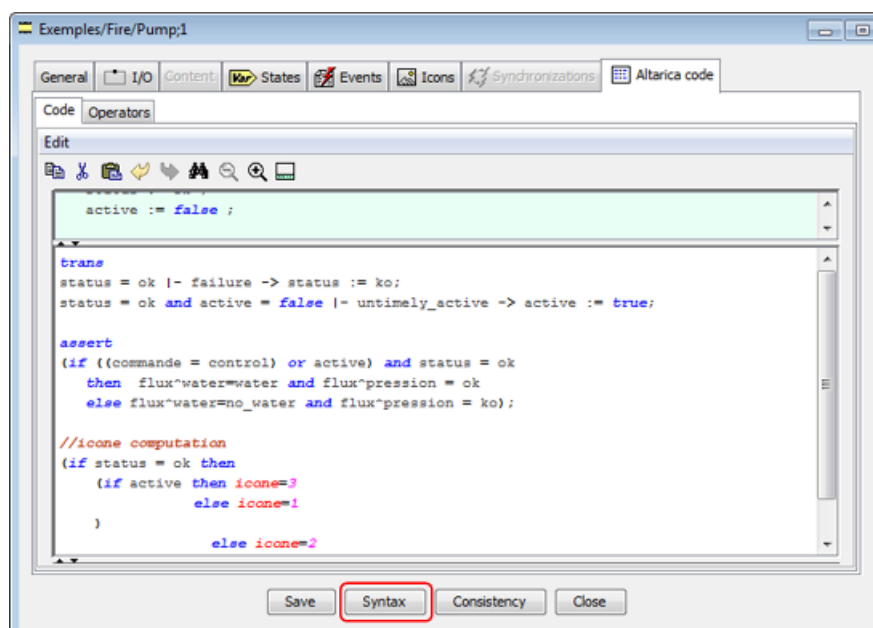
#### 6.3.1.1.1. For systems

In order to launch syntactical control for current system, use the **Check syntax** command with click on  in **MBSA** menu.

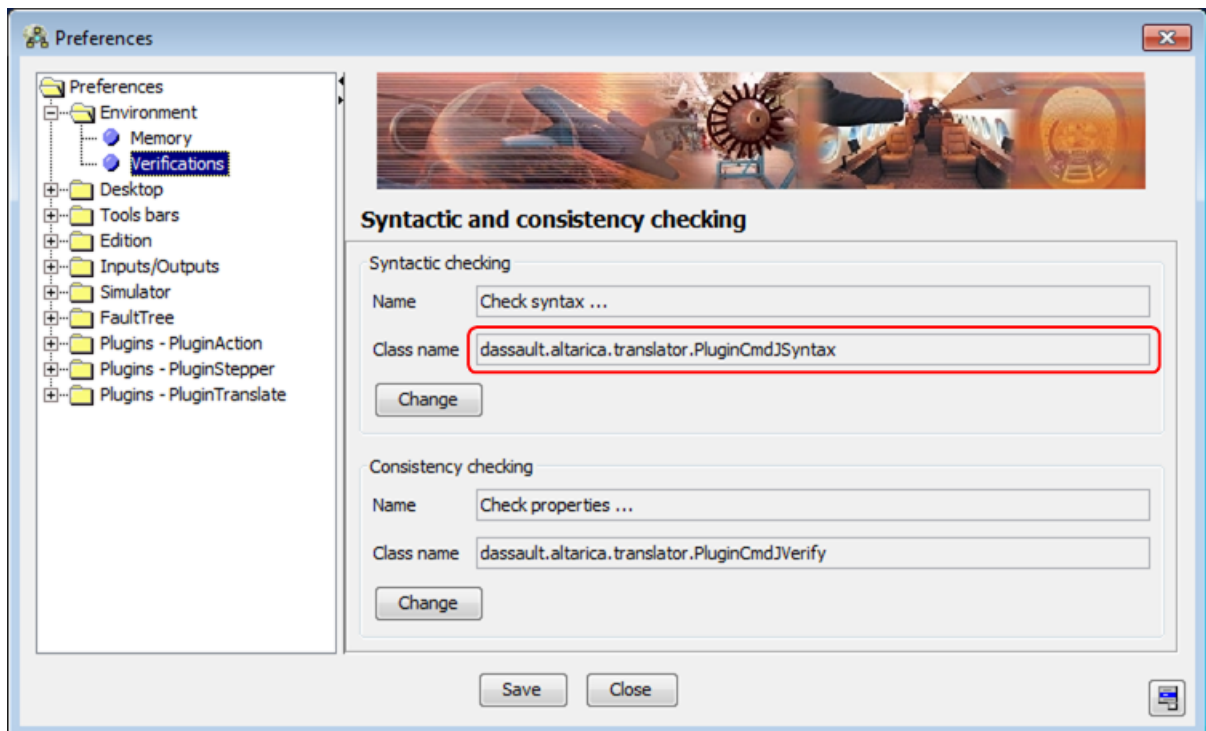


#### 6.3.1.1.2. For components, equipments or operators

Syntactical control for components, equipments or operators is made in their edition window (button **Syntax**).



During the first use, you have to verify that syntactical checker linked with this button is the one of translator. It can be made in preferences of Cecilia-Workshop (=> Menu **Options**, Command **Preferences**, Path **Preferences/Environment/Verifications**).

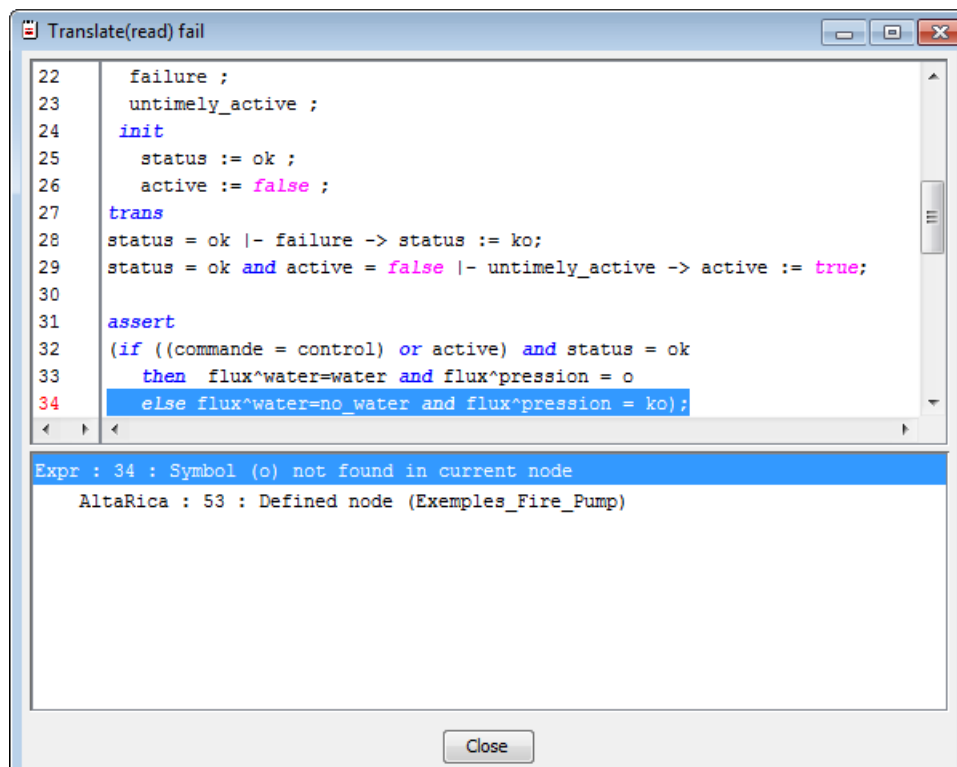


The class name of syntactical check must be `dassault.altarica.translator.PluginCmdJSyntax`. On the contrary, another syntactical checker will be used. In order to change the syntactical checker to be used for components, equipments or operators, click on button **Change** (of part **Syntactic check**).

## 6.3.1.2. Result of syntactical check

If there is no mistake, a message will specify it.

On the contrary, a windows **Syntax error** is displayed.





The upper part displays AltaRica code generated by Cecilia-Workshop. Every line with a number displayed in red or violet contains at least one error. The bottom part displays causes of the error. A click on the error message allows to select the line that could cause problem.

## 6.3.1.3. List of usual errors

1. The exact wording of identifier is defined in the upper editor which is dedicated to visualization of variables definitions (state, flow, icon, event) specified in previous tabs. A typo error in AltaRica code will imply a wrong identifier spelling which won't match exact wording.
2. Wrong syntax of transitions declaration ('=' operator for guards, ':=' operator for assignments).
3. Assertions define assignments on variables from different types (two different enumerate type, one enumerated with a boolean, ...)
4. Key words `trans` or `assert` has been forgotten.
5. A missing bracket in `if ... then ... else ...` operator. Good code indentation, in imbricate `if ... then ... else ...` expression, usually enables to avoid these issues. Each closing bracket must correspond to an opening one.

```
if ... then ... else ...
  (if <boolean-expression> then <expression1> else <expression2>)

if ... then ... else ...
  (if <boolean-expression1>
    then <expression1>
    else (if <boolean-expression2>
          then <expression2>
          else <expression3>
        )
  )
)
```

## 6.3.2. Property check/Control

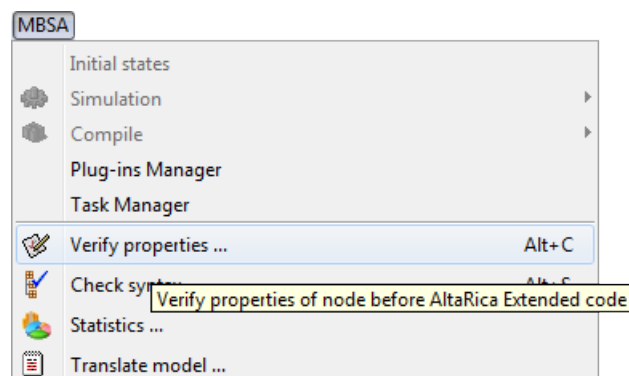
Property check (or control) enables to validate semantics and to verify a certain number of properties on the model.

Verified properties allow to validate - a priori - that a model is compatible with tools that will be used. For example, it's possible to validate - a priori - that model doesn't use arithmetic operators which will avoid tree generation.

### 6.3.2.1. Launching of property control

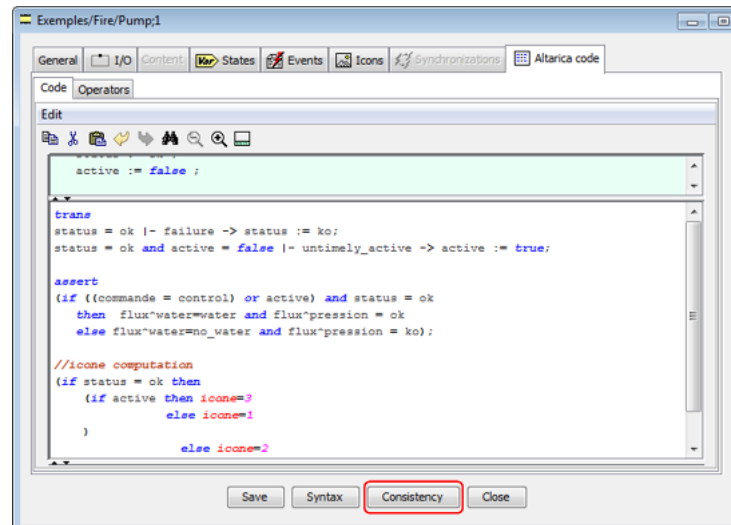
#### 6.3.2.1.1. For systems

In order to verify properties of current system, use the command **Check properties ...** command with click on  in **MBSA** menu.

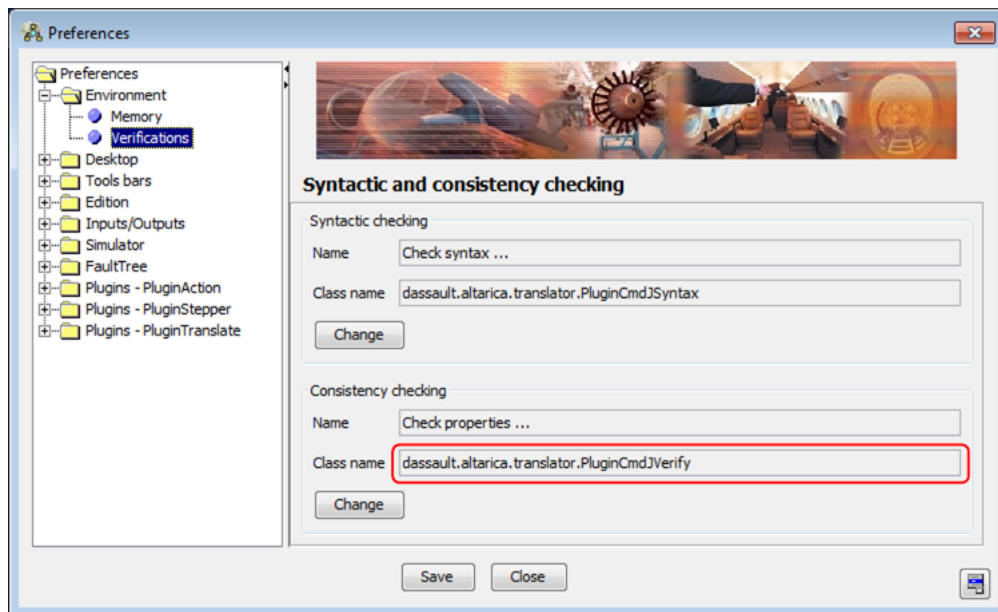


## 6.3.2.1.2. For components, equipments or operators

Property control for components, equipments or operators is made in their edition windows (button **Consistency**)



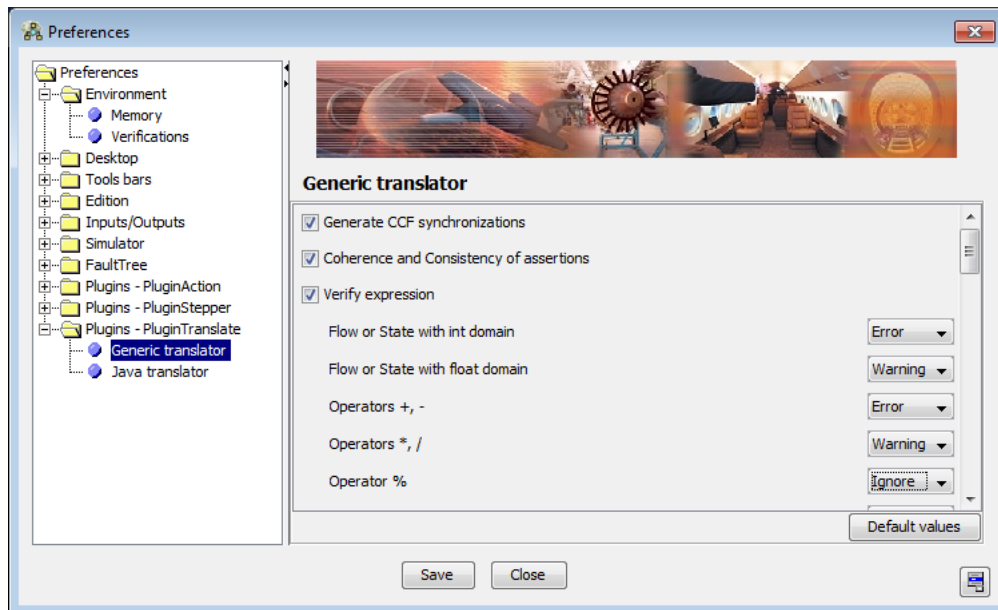
During the first use, you have to verify that property control linked with this button is the one of translator. It can be made in preferences of Cecilia-Workshop (=> Menu **Options**, Command **Preferences**, Path **Preferences/Environment/Verifications**).



The class-name of consistency-control must be `dassault.altarica.translator.PluginCmdJVerify`. On the contrary, another consistency-control (or property checker) will be used. In order to change the consistency-control to be used for components, equipments or operators, click on button **Change** (in **Consistency checking** frame).

### 6.3.2.2. Choose properties to be defined

It's possible to define parameters for properties check in order to focus on properties having impact on treatment tools that will be used.



For each properties to be checked, it's possible:

- either to consider it's not an issue: **Ignore**;
- or to display a message: **Warning**;
- or to stop treatment: **Error**.

Inside property check, one warning or error is enough to display **Error(s) & Warning(s)** window.

In the future, select Error for a property will stop any treatment needing translator.

In order to easily define parameters for a set of properties, they are grouped in categories. To Ignore a category (uncheck and/or choose **Ignore**) enables to ignore every property/category included in it.

### 6.3.2.3. List of checkable properties

- Generate CCF (Common Cause Failure) synchronization: Used during translation from AltaRica Extended format to AltaRica format (*e.g.* Section 3, “Translation into 'standard' AltaRica”).
- Completeness check and assertion consistency: Used during 'dataflowisation' of assertions (*e.g.* Section 4, “Convert assertions to assignments”).
- Verification on expressions
  - Verification of integer variables presence
  - Verification of float variables presence
  - Verification of operators + and – presence.
  - Verification of operators \* and / presence.
  - Verification of operator % presence.
  - Verification of operators min and max presence.
  - Verification of cardinal operators : # ( . . . ).
- Verification of warring transitions (same guard, same event)
- Verification of transition guards
  - Verification of flow presence inside transitions
  - Verification of always active guards
- Verification of synchronization type
  - Synchronization type

- BroadCast (Diffusion) type
- CCF (Common Cause Failure) type
- Synchronization with events belong to same sub component
- Verify node with local simulation (e.g. Section 5, “Local simulation of components”).
  - Presence of dynamic component
- Verification of generated Java code for Java simulator. This verification enables to confirm that code can be compile by Java compiler
- Verification of loops inside assertions
- Verification of model events
  - Verification of instantaneous event presence
  - Verification of temporized event presence
  - Verification of law presence for each event
- External clauses verification
  - External clauses verification remark
  - External clauses verification law
    - Verification of compatibility between laws with *Aralia*
    - Verification of compatibility between laws with *Moca12*
  - External clauses verification parameter
  - External clauses verification attribute
  - External clauses verification nodeproperty
  - External clauses verification priority
    - Verification of priority affectation only for temporized events.
  - External clauses verification bucket
  - External clauses verification preemptible
  - External clauses verification observer [deprecated]
  - External clauses verification predicate
  - External clauses verification property
- Always flatness during Mec5 translate.

Event priority is managed in a different manner in AltaRica Extended (overall definition with integer associated to deterministic events) and in AltaRica-Mec5 (local definition with partial order of events at the component level).

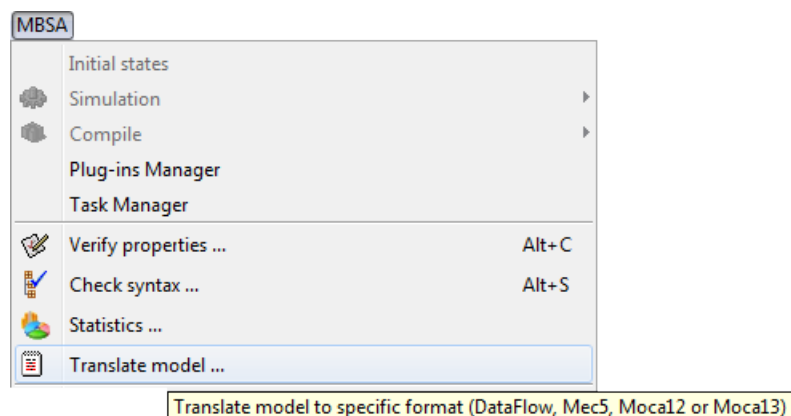
In order to keep semantic equivalence between tools, a setting flat of the model must be done if the model uses instantaneous events or if there is a priority..

### 6.3.3. Model translation

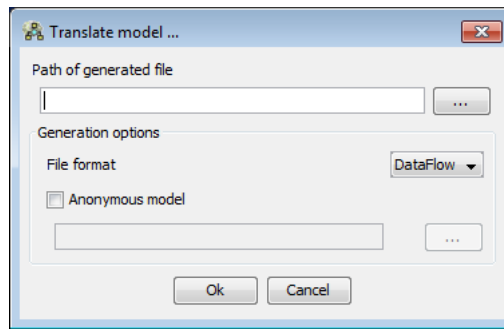
This function allows current model exportation to a file specified by the user in a specific format.

#### 6.3.3.1. Command Translate model ...

In order to launch current system translation, use the **Translate model ...** command with click on  in **MBSA** menu.



The following window enables to define parameters for model translation.



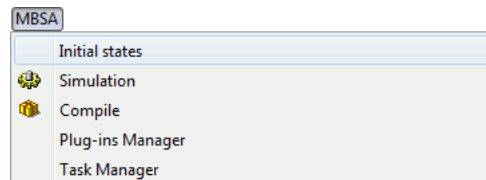
- Type result-file name (either directly, or with ... button).
- Select output format in the **Display format** list: **DataFlow**, **Mec5**, **Moca12**, **OTools**
- You can generated an **Anonymous model**(principally for confidential reasons) In this case, a file containing link between original model and anonymized one can be generated.

## 6.4. Simulation Step by Step

### 6.4.1. Simulation configuration

Before simulation launching, initial states have to be defined for simulation.

- Use **MBSA - Initial states** menu in order to define initial conditions.



- **Initialization** waiting window is displayed.
- Then, initial states window is displayed.

This window enables to choose a new initial state of the system.

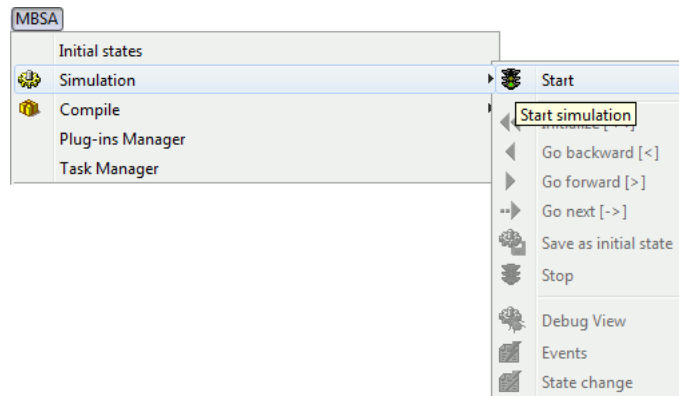
Default value of initial state is the set of default value from all component models used in the system.


There are two tabs, one for state variables and the other for events.


### 6.4.2. Simulation launch

In order to launch interactive simulation, you can:

- Either use **MBSA - Simulation - Start** menu

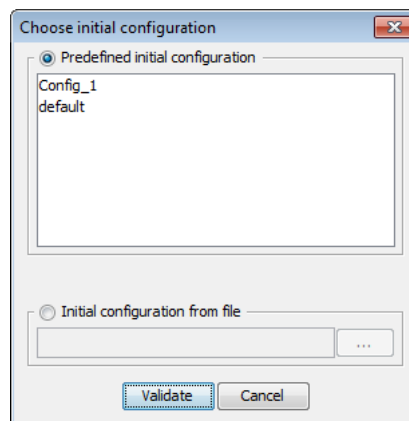


- or click on  icon.

 When many step by step simulators are available, you can choose the one you want by clicking on the little arrow next to the start icon. The selection of a simulator will launch the simulator, and memorize this choice for the next simulation.



Before simulation starts, initial configuration can be chosen.



Initial configuration can be:

- Either chosen among previously saved configurations (cf. Section 6.2.7.4, “Initial Configurations tab”).
- or defined externally with a configuration file.

The file format must be a list of couples { name of state variable, initial value of variable } as follows:

```
equipA.compA.Tension = Null
equipA.compB.Tension = Failed
...
```

It corresponds with the following syntax:

```
<init-list>
::= <init-def> ( ' ' <init-def> )*
<init-def>
::= <hierarchy-path> ':' <expression>
::= <hierarchy-path> '=' <expression>
<expression>
::= true
```

```

::= false
::= <integer>
::= <float>
::= <identifier>
<identifier> ::= '[a-zA-Z][a-zA-Z0-9_-]*';
<hierarchy-path> ::= <identifier> ('.' <hierarchy-path>)*

```

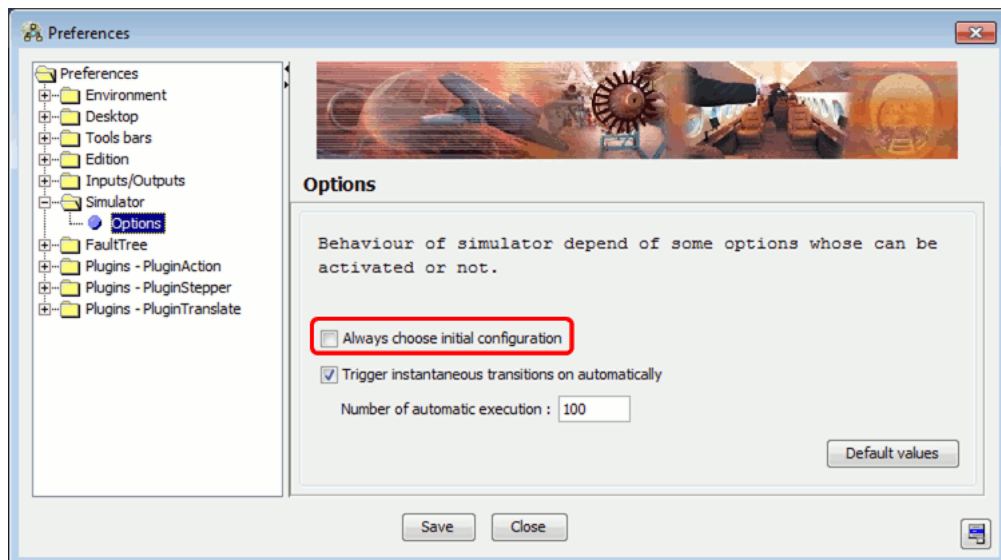
Each variable in file must be defined in the simulated model.

Otherwise, an error message is displayed and current action is stopped.

For initial configuration file, it is also possible to use an xml result file coming from generic sequence generation or FMEA Generation. In this case, initial configuration will be the one defined for generation.

Once configuration is selected, click on **Validate** button to launch simulation.

**i** The window allowing configuration choice is usually displayed only if there are many previously saved configurations. The display of this window can be forced in software preferences. (**Options - Preferences...** command).

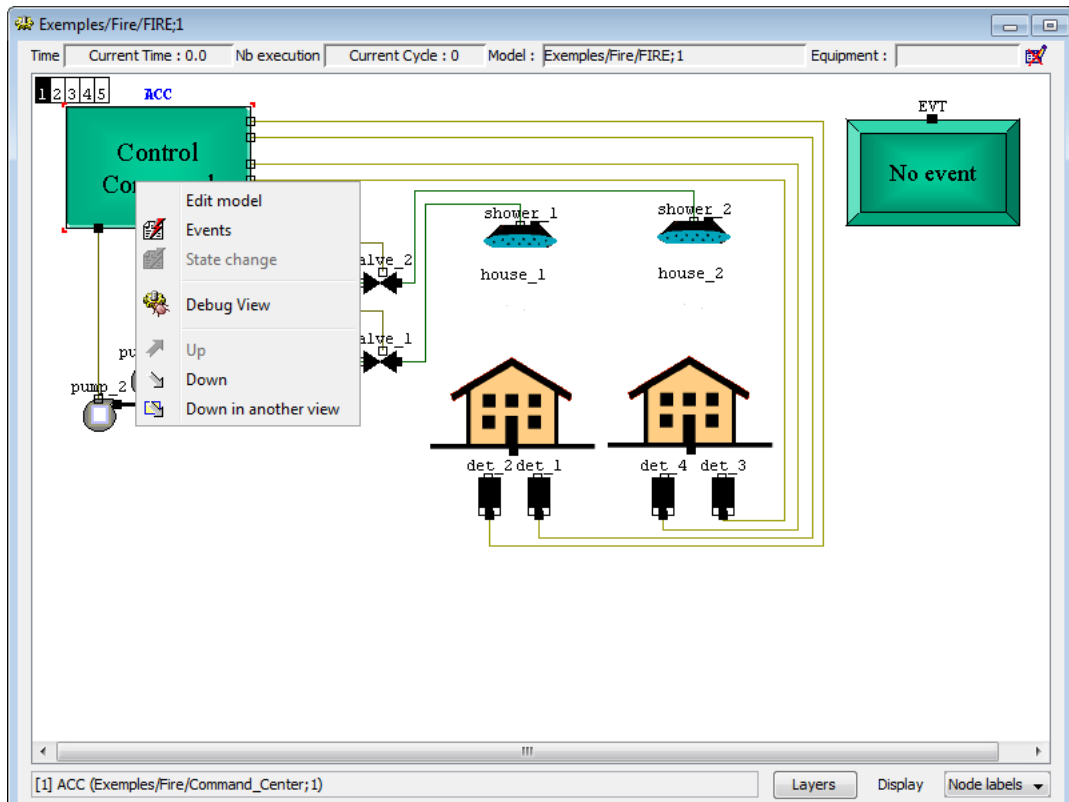


When simulation starts, many verifications are made on every component model in architecture. If an error is detected, a window is displayed and simulation doesn't start.

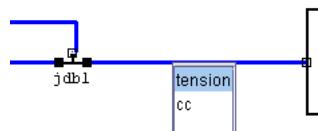
## 6.4.3. Simulating

When simulation starts, initial state of system is displayed: colors of links show value of their variables, icons represent the components in their initial states.





When the graphical link represents a bus on which several flow parameters transit, a right click on this link gives access to the flow parameters list in order to modify dynamically the graphical display order of the flow parameters. The link color resulting from this change of order corresponding to the value of the flow variable located in 1st position



#### 6.4.3.1. Commands available in simulation

Following commands are available in simulation:

- **⏮ (Initialize):** Reinitialize system in its initial state. This action cancels all event triggering made during simulation,
- **⏪ (Go back):** enables to go back (one step), this action cancels the last triggering of simulation.
- **⏩ (Go forward):** enables to redo (forward) a step that has been previously canceled,
- **➡ (Next):** enables to trigger the next event of scheduler, cf. Section 6.4.6.7, “Scheduler”,
- **💾 (Save as initial state):** enables to memorize current system state during simulation in order to create initial states. cf. Section 6.4.7.1, “Initial configuration saving during simulation”,
- **🛑 (Stop):** Stop simulation and close window,
- **📄 (Informations window):** enables to display information about current simulator (history, scheduler, list of variables, ...); cf. Section 6.4.6, “Simulator display window”,
- **🔍 (Event selection):** enables to trigger an event/transition of the simulator; cf. Section 6.4.4, “Transitions triggering”,
- **📝 (State variable value):** enables to modify state variables of selected element; cf. Section 6.4.5, “State variable modification”.



Errors like division by 0 or consistency error that have not been verified on a component can appear during simulation. If such an issue appears, a message is displayed and simulation is stopped.



Some errors that are not definitive can appear, such as a too large number of instantaneous transition triggering.

### 6.4.3.2. Move along equipment/systems view

You can move along graphical view in order to display the system and its sub-equipment.

In order to move along, three icons are available in popup menu and in tool bar:

- ↗ (**Up**): Go to upper hierarchical level
- ↘ (**Down**): Go to lower hierarchical level
- 🖼️ (**Down in another view**): Display selected event in another view



A double click on an equipment enables to go **Down**. On components, a double click launch component edition in read-only mode.

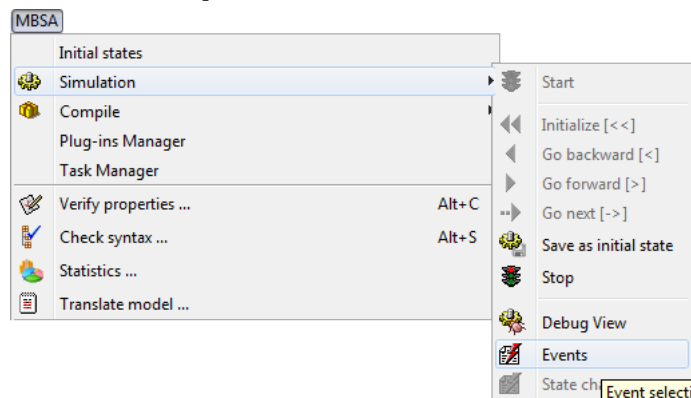
### 6.4.4. Transitions triggering

Event triggering /transition firing

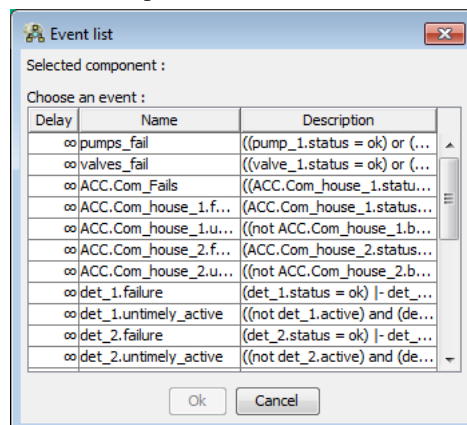
Simulation enables to validate a system. System is simulated from its initial state. Then events can appear during life of systems. In AltaRica, action of these events is defined in transition definitions.

To trigger an event means to trigger (to fire) a transition labeled with this event.

Events/transitions are associated with components. To trigger/fire a transition, select component, then use **MBSA - Simulation - Events** command (or one of its equivalents in toolbar)



If at least one of the transitions associated with component are fireable, the following window is displayed.





The first column displays delay associated with the transition. In case of stochastic (not determinist) events, delay doesn't mean anything. In fact, transition is triggered randomly. The displayed delay is infinite. For determinist transitions see Section 6.4.7.3, "To take determinist events into account - Temporized Transitions".

The name of the event labeling transition is displayed in the second column, the third one reminds transition definition.


Sort is available on columns (double click on header).

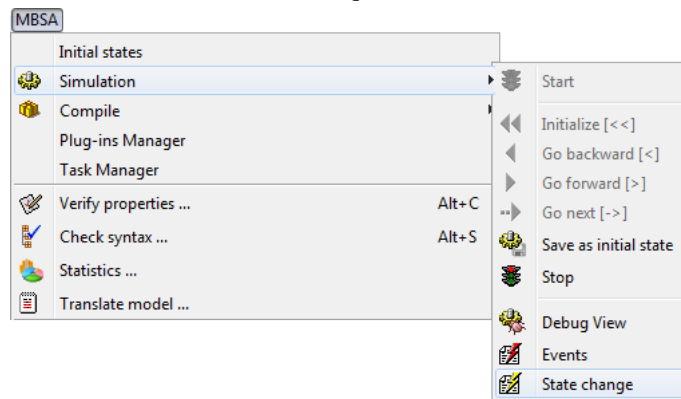
Select a transition and click on **Ok** button to trigger transition.

-  In order to trigger a synchronization at equipment level, select the equipment and do the same things. Then, the window displays fireable transitions of equipment and of its sub-components.
-  In order to trigger a global synchronization (defined at system level), don't select anything. The window displays all system transitions that are fireable.

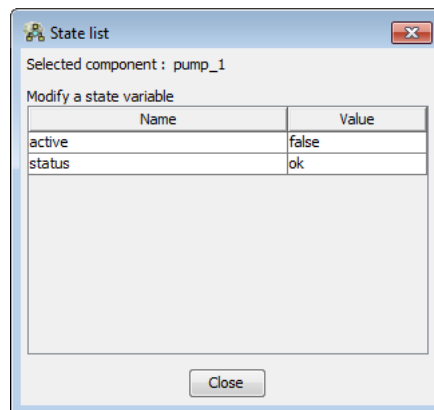
## 6.4.5. State variable modification

Component state variables can be directly modified. It's mainly used to configure system without transition triggering. If state variables are used as parameters (*e.g.* for component capacity), it enables to make local modifications in order to see how the system reacts.

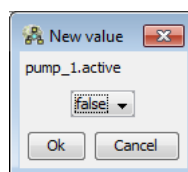
Select **MBSA - Simulation - State variables** (or one of its equivalents in toolbar ) to modify a state variable.



The following window is displayed:



Double-click on the value to be modified (column **Value** for the considered state variable). The following window displays different available values for the variable.



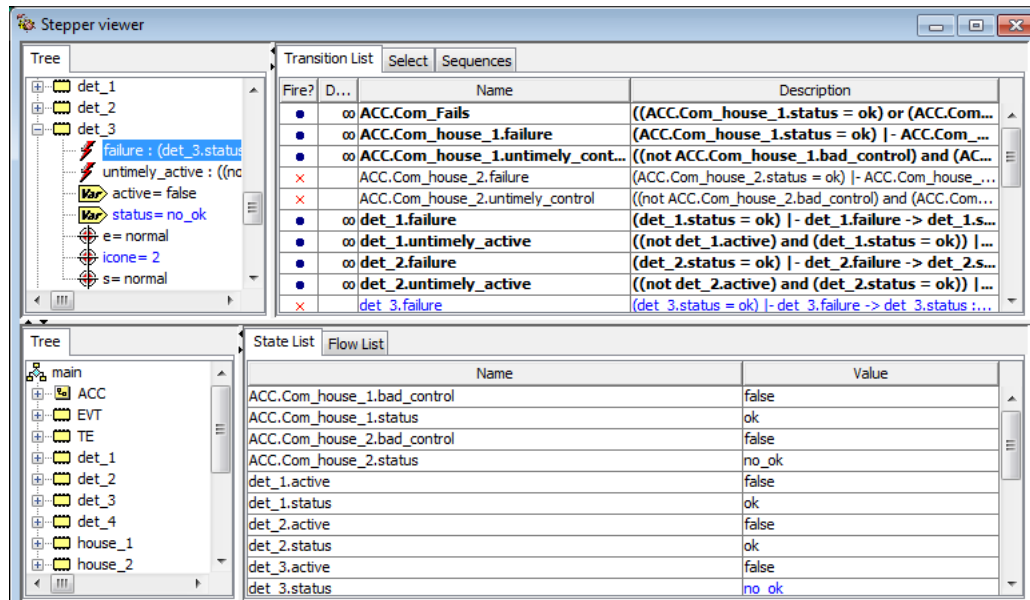
Click on **Ok** button to validate changes.

The simulation result is displayed on screen: component icon changes and colors of some links are modified because of state variable modification.

## 6.4.6. Simulator display window

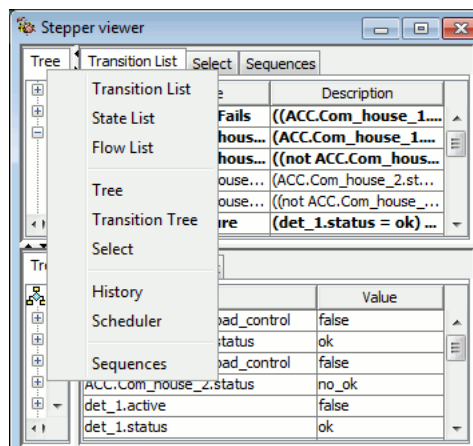
### 6.4.6.1. Generalities

Simulator display window provides further information than graphical representation.



This window is splitted in 4 panels. Each panel can display different views thanks to tabs. View can be moved from tab to tab with a "drag and drop".

Click right on a empty panel or on the header in order to display Contextual menu. It enables to add different types of views in current panel.



Each view has its own functions. Views are described in following chapters.

#### 6.4.6.2. Transition list

**Transition list** enables to display fireable or not fireable transitions of current simulation.

Fire?	D...	Name	Description
×		ACC.Com_Fails	((ACC.Com_house_1.status = ok) or (ACC.Com_hou...
●		ACC.Com_house_1.failure	((ACC.Com_house_1.status = ok)  - ACC.Com_hous...
×		ACC.Com_house_1.untimely_control	((not ACC.Com_house_1.bad_control) and (ACC.Co...
×		ACC.Com_house_2.failure	(ACC.Com_house_2.status = ok)  - ACC.Com_hous...
×		ACC.Com_house_2.untimely_control	((not ACC.Com_house_2.bad_control) and (ACC.Co...
●	∞	det_1.failure	((det_1.status = ok)  - det_1.failure -> det_1....
●	∞	det_1.untimely_active	((not det_1.active) and (det_1.status = ok)) ...
●	∞	det_2.failure	((det_2.status = ok)  - det_2.failure -> det_2....
●	∞	det_2.untimely_active	((not det_2.active) and (det_2.status = ok)) ...
×		det_3.failure	(det_3.status = ok)  - det_3.failure -> det_3.status ...
×		det_3.untimely_active	((not det_3.active) and (det_3.status = ok))  - det_...
●	∞	det_4.failure	((det_4.status = ok)  - det_4.failure -> det_4....
●	∞	det_4.untimely_active	((not det_4.active) and (det_4.status = ok)) ...
●	∞	house_1.fire	(house_1.status = normal)  - house_1.fire -> ...
●	∞	house_2.fire	(house_2.status = normal)  - house_2.fire -> ...
●	∞	pump_1.failure	(pump_1.status = ok)  - pump_1.failure -> p...
●	∞	pump_1.untimely_active	((pump_1.status = ok) and (pump_1.active = ...
●	∞	pump_2.failure	(pump_2.status = ok)  - pump_2.failure -> p...
●	∞	pump_2.untimely_active	((pump_2.status = ok) and (pump_2.active = ...
●	∞	pumps_fail	((pump_1.status = ok) or (pump_2.status = ...
●	∞	valve_1.failure	(valve_1.status = ok)  - valve_1.failure -> v...

Columns show:

1. If transition is valid (blue ball) or not valid (red cross).
2. Delay associated with transition. In case of stochastic transitions, delay is infinite.
3. Name of the element associated with transition.
4. Description of transition given by simulator. In case of Java1 Simulator, it is definition of transition.

#### 6.4.6.3. Variable list

**State list** (respectively **Flow list**) enables to display state variables (respectively flow variables) of currently simulated model.

Name	Value
ACC.Com_house_1.bad_control	false
ACC.Com_house_1.status	no_ok
ACC.Com_house_2.bad_control	false
ACC.Com_house_2.status	no_ok
det_1.active	false
det_1.status	ok
det_2.active	false
det_2.status	ok

Name	Value
ACC.Com_house_1.commande	no_control
ACC.Com_house_1.detecteur	normal
ACC.Com_house_2.commande	no_control
ACC.Com_house_2.detecteur	normal
ACC.Det_or_1.e1	normal
ACC.Det_or_1.e2	normal
ACC.Det_or_1.s	normal
ACC.Det_or_2.e1	normal
ACC.Det_or_2.e2	normal
ACC.Det_or_2.s	normal
ACC.com_pump_1	no_control

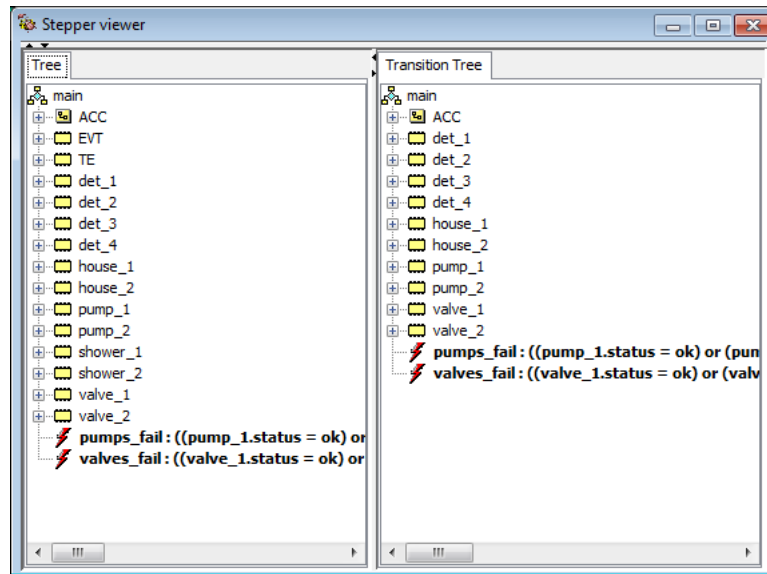
Table columns can be sorted.

Variables whose value has changed during last firing are displayed in bleu.

Value of a state variable can be directly modified (double-click on the value to be modified).

#### 6.4.6.4. Hierarchical view - Tree

**Tree** view enables to see all information during simulation with a hierarchical way.



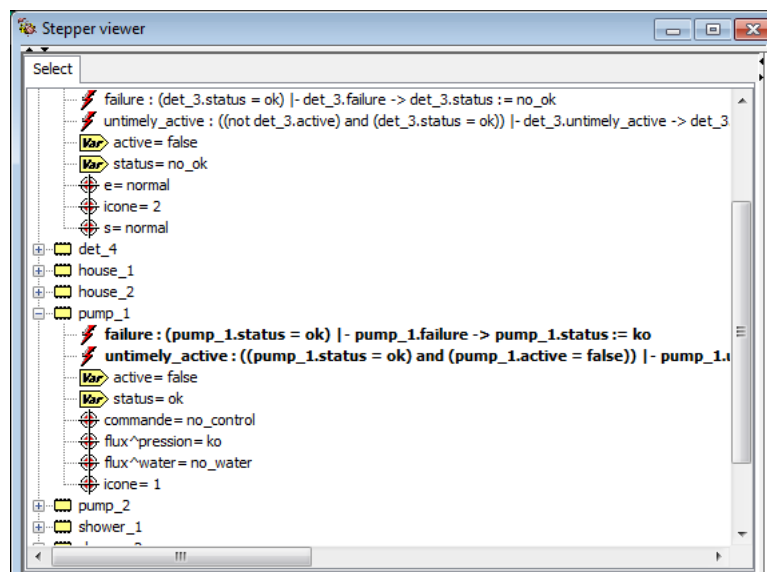
**Transitions Tree** displays all transitions with a hierarchical way. If a component/equipment doesn't have any transitions, it is not displayed in the list.

Transitions are in bold, variables whose value has changed during last step are displayed in blue.

Selection of component/equipment or change of hierarchical level in graphical simulator leads to selection of the node in the tree.

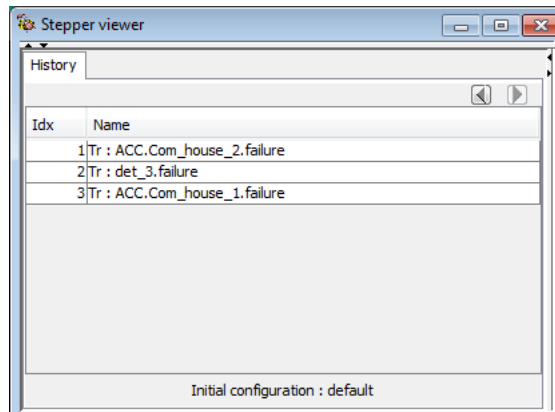
#### 6.4.6.5. Select view

**Select** displays tree related to the selected graphical component.



#### 6.4.6.6. Transition History

**History** enables to see which transition have already been fired/triggered.



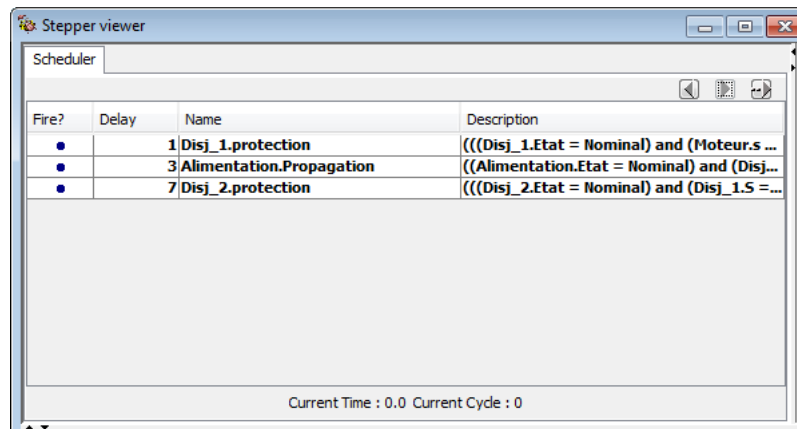
When the end of the list is in gray, it means that user has gone back with **Go Back** [<] command ◀. History can be redone with **Go forward** [>] command ▶.

A double-click on a transition enables to go forward to the state before selected transition triggering.

If a state variable is modified, it is displayed in history.

#### 6.4.6.7. Scheduler

**Scheduler** displays determinist transitions that are valid. They are sorted following the order in which they will be fired in step by step simulator.



→ **Next** command triggers the first transition of scheduler.

For more information about determinist transitions, cf. Section 6.4.7.3, “To take determinist events into account - Temporized Transitions”

#### 6.4.6.8. Sequences

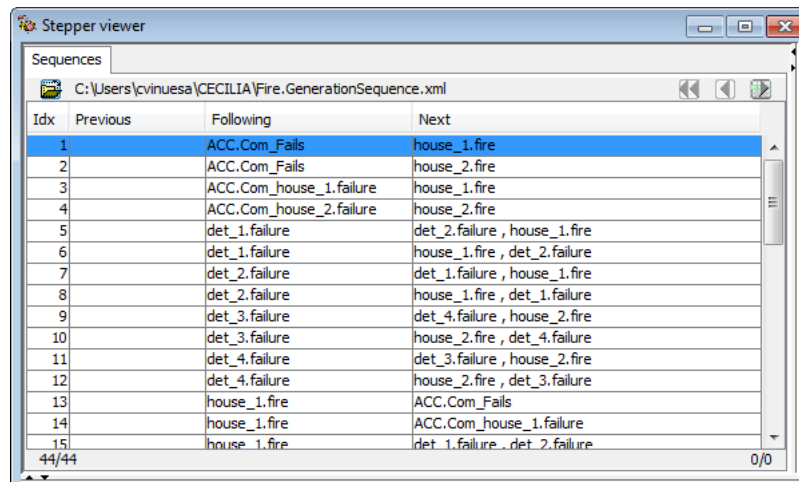
**Sequences** enable to graphically replay sequences coming from sequence generation cf. Section 6.5.2, “Sequences generation (Generic)” or from other tools.

You can:

- Validate sequences (functions of search algorithm, some sequences may not lead to wanted result)



- Help user to understand meaning of sequences (in a model behavior point of view), with graphical display (link colors, icons) and textual display (state/flow variable value).




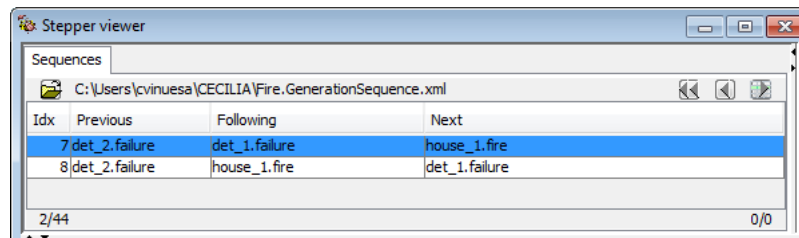
Idx	Previous	Following	Next
1		ACC.Com_Fails	house_1.fire
2		ACC.Com_Fails	house_2.fire
3		ACC.Com_house_1.failure	house_1.fire
4		ACC.Com_house_2.failure	house_2.fire
5		det_1.failure	det_2.failure , house_1.fire
6		det_1.failure	house_1.fire , det_2.failure
7		det_2.failure	det_1.failure , house_1.fire
8		det_2.failure	house_1.fire , det_1.failure
9		det_3.failure	det_4.failure , house_2.fire
10		det_3.failure	house_2.fire , det_4.failure
11		det_4.failure	det_3.failure , house_2.fire
12		det_4.failure	house_2.fire , det_3.failure
13		house_1.fire	ACC.Com_Fails
14		house_1.fire	ACC.Com_house_1.failure
15		house_1.fire	det_1.failure , det_2.failure

Each sequence is represented by a table line.

Columns give:

1. ID or number of the sequence.
2. **Previous** event of the sequence, that is to say events already fired.
3. **Following** fireable event, that is to say the following event that will be fired for the selected sequence.
4. Remaining event (the end of sequence).

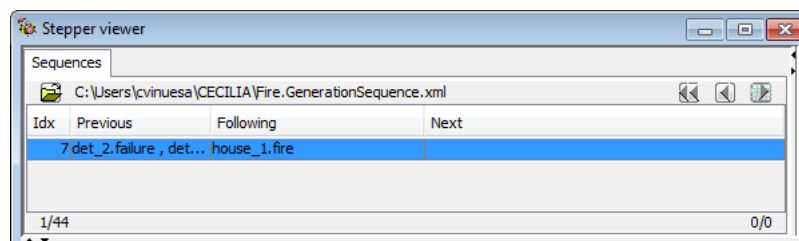
Select  button to skim through the sequence. The **following** event is fired. A double-click on sequence or pressing space bar have the same effect.



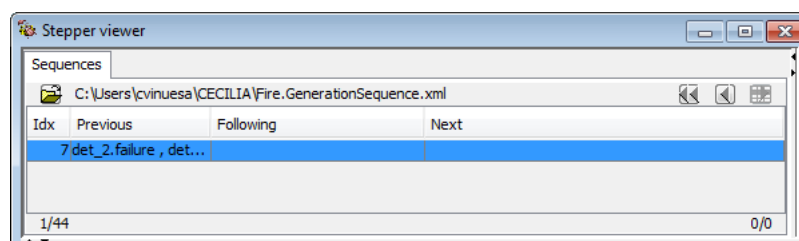
Idx	Previous	Following	Next
7	det_2.failure	det_1.failure	house_1.fire
8	det_2.failure	house_1.fire	det_1.failure

List is updated functions of current sequence (i.e event history).


Repeat to skim sequence through the end.



Idx	Previous	Following	Next
7	det_2.failure , det...	house_1.fire	



Idx	Previous	Following	Next
7	det_2.failure , det...		

**Sequences** are linked to a set of sequences coming from a file. To load a sequence file, use  button.

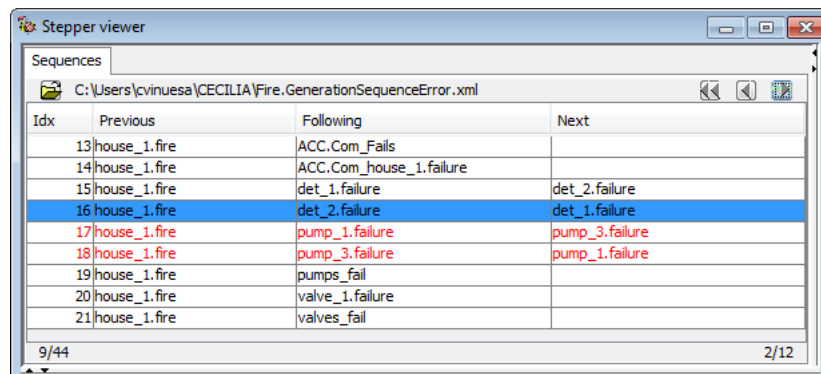
File must be in XML format of generic sequence generator (Cf appendix on generic tools for more informations).

It must fit to the following format:

```
<?xml version='1.0'?>
<seqgen>
...
<result>
  <seq><tr evt="synthesys_2.failure"/>
    <tr evt="screen_1.failure"/>
    <tr evt="screen_pilot.and_2.out_1"/></seq>
  <seq><tr evt="synthesys_2.failure"/>
    <tr evt="screen_1.failure"/>
    <tr evt="screen_pilot.relay_2.untimely_close"/></seq>
...
</result>
...
</seqgen>
```

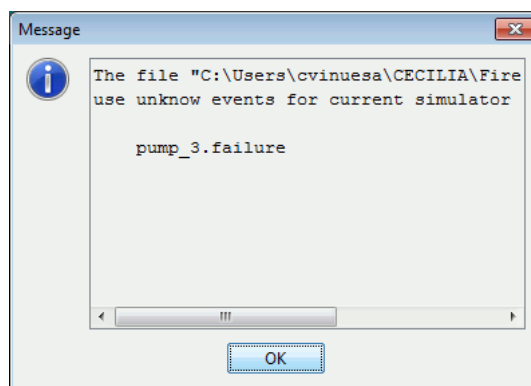
Presence of events in the model is verified during file loading.

If an error happens,



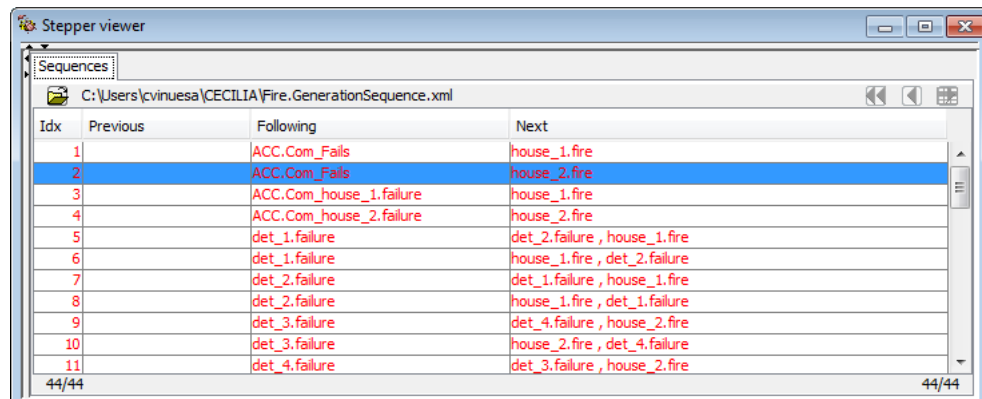
Idx	Previous	Following	Next
13	house_1.fire	ACC.Com_Fails	
14	house_1.fire	ACC.Com_house_1.failure	
15	house_1.fire	det_1.failure	det_2.failure
16	house_1.fire	det_2.failure	det_1.failure
17	house_1.fire	pump_1.failure	pump_3.failure
18	house_1.fire	pump_3.failure	pump_1.failure
19	house_1.fire	pumps_fail	
20	house_1.fire	valve_1.failure	
21	house_1.fire	valves_fail	

- Sequences having issues are displayed in red
- Number of these sequences is displayed in the bottom right.
- To display errors, double-click on the bottom right.



For each simulation step, sequences of the current list are validated: one and only one transition labelled with following event must be fireable.

Otherwise, no valid transition or more than one, next event is displayed in red (bold or italic) and sequence cannot be played.



Idx	Previous	Following	Next
1		ACC.Com_Fails	house_1.fire
2		ACC.Com_Fails	house_2.fire
3		ACC.Com_house_1.failure	house_1.fire
4		ACC.Com_house_2.failure	house_2.fire
5		det_1.failure	det_2.failure , house_1.fire
6		det_1.failure	house_1.fire , det_2.failure
7		det_2.failure	det_1.failure , house_1.fire
8		det_2.failure	house_1.fire , det_1.failure
9		det_3.failure	det_4.failure , house_2.fire
10		det_3.failure	house_2.fire , det_4.failure
11		det_4.failure	det_3.failure , house_2.fire

In the left bottom corner, current number of sequences and total number are displayed.

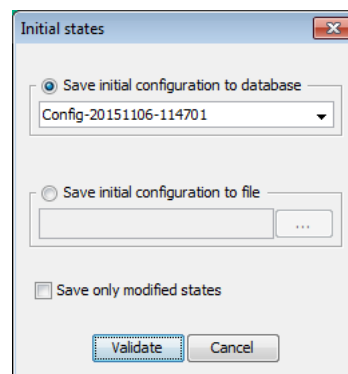
## 6.4.7. Other functionalities during simulation

### 6.4.7.1. Initial configuration saving during simulation

During simulation, current state can be saved as initial configuration.

This command is available in **System - Simulation - Save as initial state** menu, or with  icon.

When command is selected, following window is displayed:




You can save current state as initial configuration:

- Either as pre-saved configuration of the system: In this case, validate or modify the proposed name. If you choose an existing configuration name, it will be erased. You can see saved initial configuration in **System - Initial states** menu. (cf. Section 6.2.7.4, “Initial Configurations tab”).
- Or in a text file whose format is defined in simulation launching section .


The **Save only modified states** checkbox enables to keep only differences with default initial state of the system.

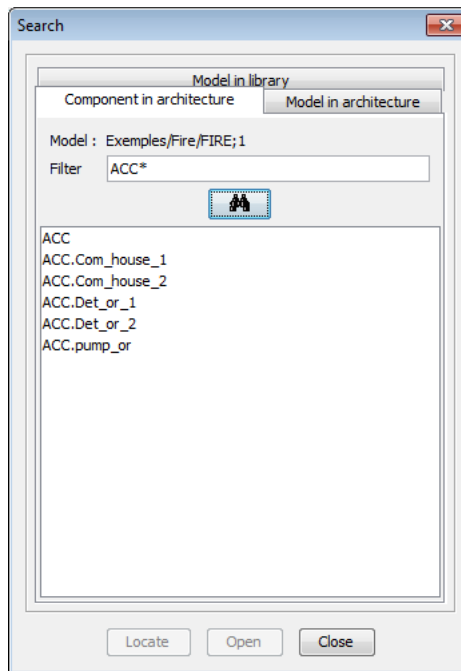
### 6.4.7.2. Search for component during simulation

During simulation on a system architecture, the search for an instance of component is made as follows:

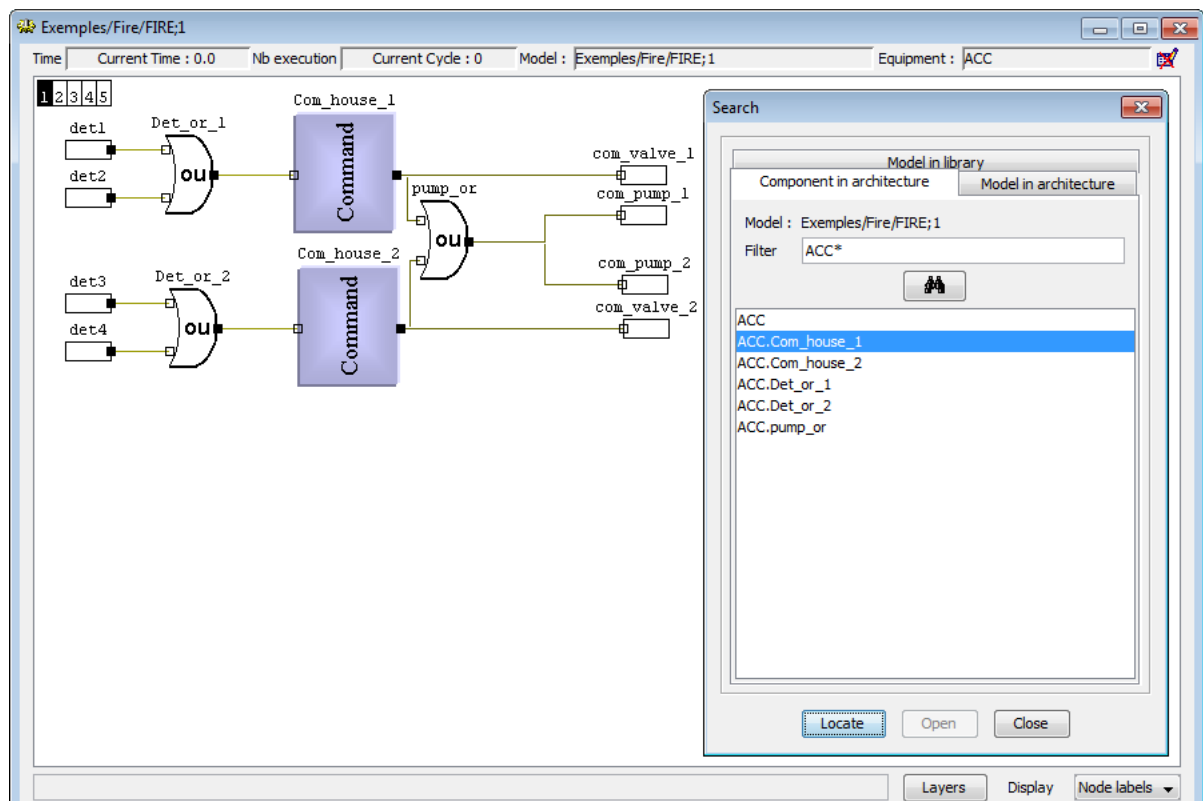
- Use **Edit - Search** command (Ctrl+B) or  icon in edit toolbar. Then, **Search** window is displayed.
- In a system, click on **Component** tab. In **Filter** field, type the name (case sensitive) of searched component as follows: [Equipement\_Name] . [Component\_Name]

The ' \* ' character can be used to limit search, or to complete name of a hierarchical level or a component name (examples : HYD\* . Y\_VF\* , \* . Y\_VF , \*Y\_VF\* ).

- Click on  icon. The searched component (or components having name matching filter) is (are) displayed in the lower part of the window.



- Select searched component.
- Click on **Locate** button. The hierarchical level containing component is displayed and component is selected.



### 6.4.7.3. To take determinist events into account - Temporized Transitions

#### 6.4.7.3.1. Introduction

In original AltaRica model, events are not classified in categories. In case of dysfunctional studies, you need to classify events in different categories.

Failures are random, they have to be considered as stochastic events. In order to do that, we link event to a probability law (exponential, weibull, ...) It is done with external clause `law` of the model (See Standard manual of Altarica Extended language).

System re-configuration event that arise from failure, must be considered either as instantaneous event, or temporized one (must happen at the end of a  $\delta$  time). This type of event is defined with a Dirac probability law with  $\delta$  parameter.

```
extern law <event reconfig> = Dirac(3);
```

If  $\delta$  is equal to 0, event is instantaneous. If one of these transitions is fireable, it must be fired instantaneously.

If  $\delta$  is greater than 0, a transition which is fireable à  $t$ , will be fired after  $\delta$  time only if transition stays fireable between  $t$  and  $t + \delta$ .

Temporized event is an extension of Altarica model which is not a reconsideration of overall behavior.

#### 6.4.7.3.2. Consideration in scheduler

If temporized events are taken into account, a scheduler must be managed in order to sort transitions functions of determinist delay related to events labelling transitions, and functions of dates when transitions become fireable.

There are only temporized transitions in scheduler. In fact, stochastic events can happen at any time, possibly between 2 temporized events.

Scheduler view displays temporized transitions that are valid. This list is sorted functions of ( $\Delta t$ ) fire delay related to each transition.

Fire?	Delay	Name	Description
•		1 Disj_1.protection	(((Disj_1.Etat = Nominal) and (Moteur.s ...
•		3 Alimentation.Propagation	((Alimentation.Etat = Nominal) and (Disj...
•		7 Disj_2.protection	(((Disj_2.Etat = Nominal) and (Disj_1.S = ...

Current Time : 0.0 Current Cycle : 0

When a temporized transition is fired, simulation current time is increased of  $\Delta t$ .

#### 6.4.7.3.3. Instantaneous transitions

Instantaneous transitions are often used to update information inside AltaRica model, and for instantaneous reconfiguration of system. It represents the expected functionality of the system. When model is OK, users don't want to see these transitions anymore, they want step by step simulator to take into account and hide them.

Simulator can be set-up in order to automatically fire instantaneous transitions. In this case, after each state change, simulator verifies that scheduler is not empty. If it is not, and if the first transition is instantaneous, simulator triggers this transition without asking user.

If simulator is set-up to automatically fire instantaneous transitions, it may be blocked in some case where there is always a fireable instantaneous transition in scheduler.

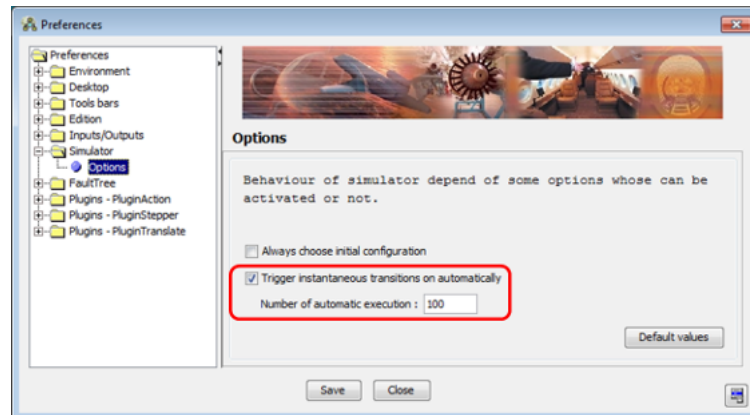
In order to stop this simulation, step by step simulator can display a message saying that too many instantaneous transitions have been fired sequentially.

When an instantaneous transition is fired, a `cycle` number is increased, When a non-instantaneous transition is fired, the `cycle` number is reset to 0.



Scheduler view displays current time in bottom left corner, and current cycle in bottom right corner.

Step by step simulator can be configured in software preferences. (=> **Options** menu, **Preferences** command, **Preferences/Simulation/Options** path)



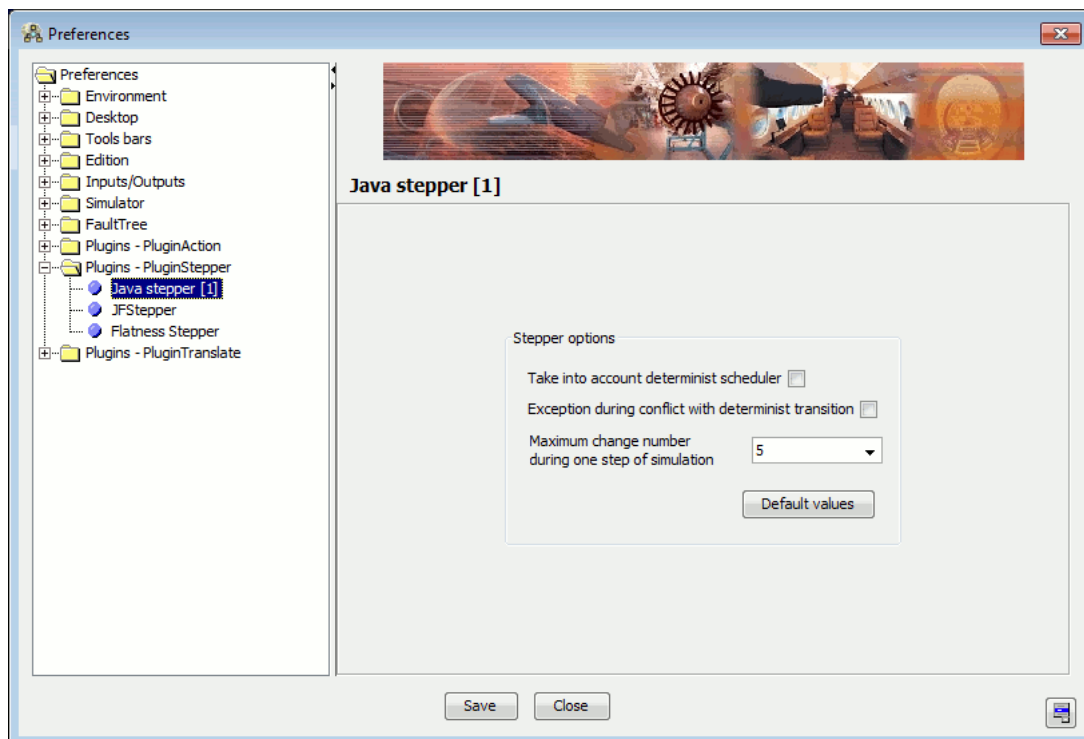
#### 6.4.7.3.4. Temporal conflict between 2 transitions

A good question that users should ask is : *What happens when 2 temporized transitions are fireable at the same time ?*

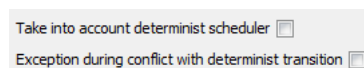
There is no warranty of conflict management with step by step simulator. To help user to manage conflict, they can define a priority on model event. It can be made with `priority` external clause. (See Standard manual of Altarica Extended language)

#### 6.4.7.4. Preference

Stepper options can be configured in **Options - Preferences** in **Plugins - PluginStepper**.



The first two options are common to the three steppers:



If the option **Take into account determinist scheduler** is ticked, the deterministic scheduler is considered.

The second checkbox manages the conflict in case of temporal conflict.

The third option is used to manage to loop in a simulation.

- In Java stepper: with the **Maximum change number during one step of simulation**.
- In JFStepper: with the **Maximum 'fix point' loop during one step of simulation**.
- In Flatness stepper: with the **Maximum 'fix point' loop during one step of simulation**.

## 6.5. Models processing

The aim of this chapter is to present the different plugins available in Cecilia-Workshop. These plugins allows to:

- generate fault trees;
- evaluate the sequence;
- generate the FMEA.

### 6.5.1. FaultTree generation with ABC

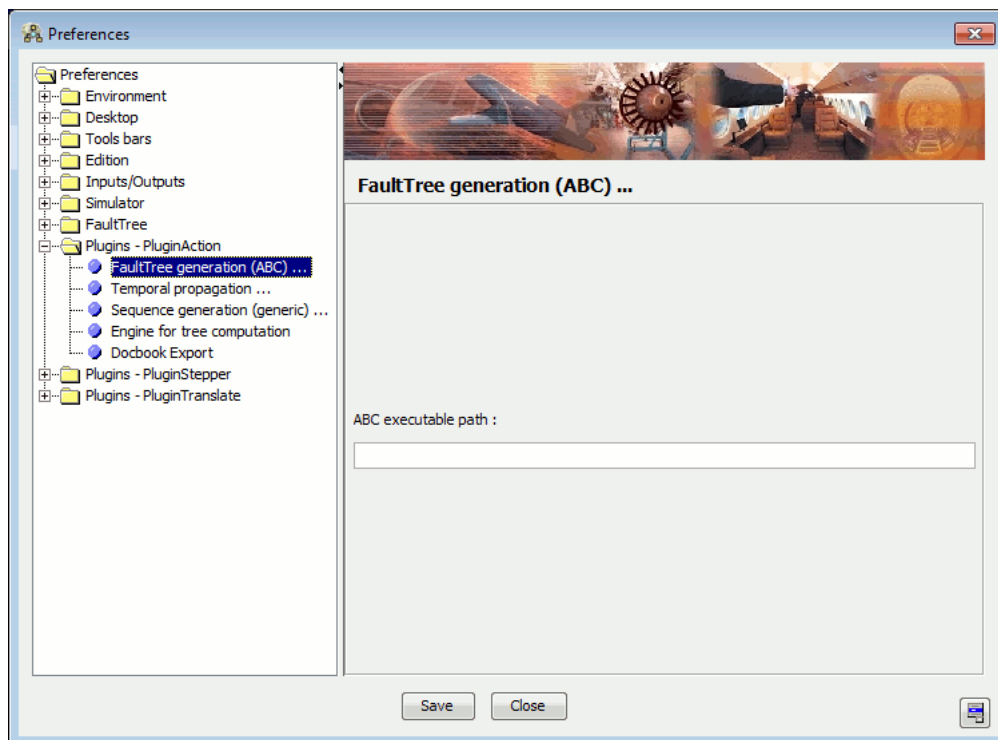
FaultTree generation consists in compiling AltaRica model into Boolean formula. Compilation algorithm uses a deductive approach from groups of "independent" components. Boolean formula generation is made with reachability graphs associated to each group.

Fault-trees are generated in *Aralia* format. This format contains equations (tree structure), laws, attributes associated with model events, and parameters of generated laws. Generated trees can be either used directly in *Aralia* with command scripts, or imported in Cecilia-Workshop or in another fault-tree-editor which supports *Aralia* native format.

#### 6.5.1.1. Install/Parameter

FaultTree generation is based on *ABC* software. This software is available in plugin installation directory.

Users' preferences allow to specify *ABC* location.



#### 6.5.1.2. FaultTree generation requirements

It's important to notice that AltaRica model can be dynamic (reconfiguration, maintenance, ...) and that a fault-tree is an instantaneous view of a system, that's to say a static view. Therefore, an AltaRica model may not be compilable into Boolean formula.



FaultTree generation implies some model restrictions. These restrictions are:

- Dynamic behavior of components

Because fault-tree is a static model, it's impossible to compile a model whose final state depends on faults order. The two following restrictions ensure a static behavior:

- For each fault-combination, if a fault-permutation is workable, every other permutation must be workable and must lead to the same state of the component.
- Transitions must be independent of component flows. In other words, component reachability graph must only depend on its initial state and its local failures.

- Uncomplete or inconsistent component(s)
- Too complex component(s)

A too complex component will generate a local combinative explosion during calcul of its reachability graph.

- Looped System

A looped system is a system with circular definition of its flow variables. That is to say, a X variable is define with a set of AltaRica assertions which depend on variable X.

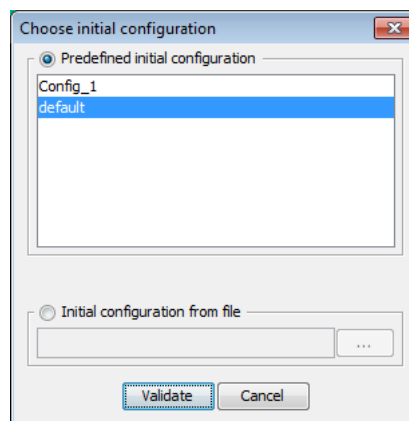
These restrictions ensure a good behavior of compilation algorithm. Check properties (when config is correct) enables to spot things that don't fit these restrictions.

### 6.5.1.3. FaultTree generation launching



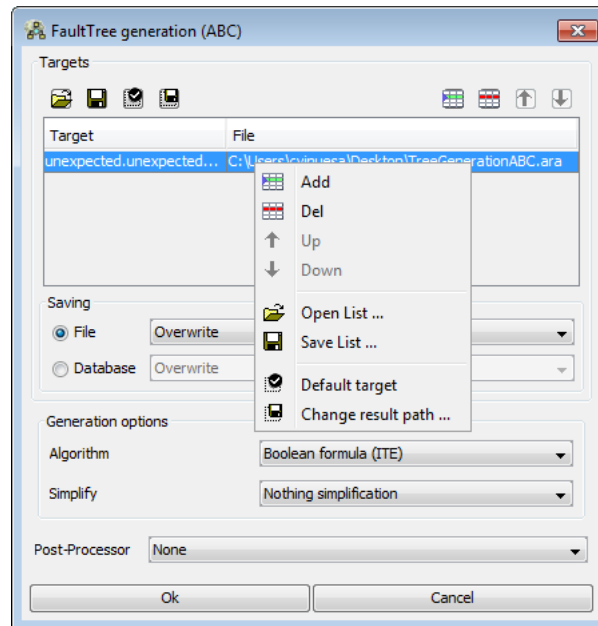
In order to launch FaultTree generation, use the **FaultTree generation (ABC)...** command.

A window is displayed when model has many initial configurations (**System** menu, **Initial states ...** command), in order to select initial configuration to take into account.



Feared event is defined thanks to a calculation-target. The target corresponds to a specific value of a model variable (If the variable is equal to the selected target the feared event happens). Calculation parameters are usually associated with a calculation-target (for example, the result file name).









The following dialog window enables to define one (or several) target(s) which must be taken into account during FaultTree generation. A fault-tree will be generated for each computation target.



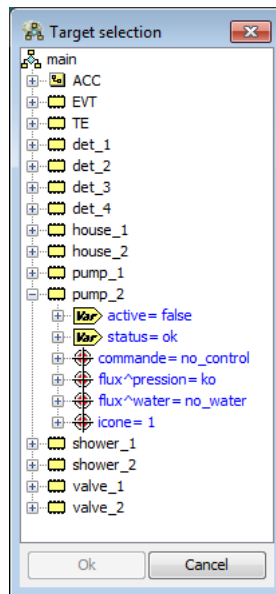
This window manages a list of computation-targets. When this window is validate, every typed information are validate to ensure that there is no incomplete line, no wrong line, targets correspond to a variable of processed model, file can be written ....

Moreover, if result-files already exist, a dialog window will ask for old file deletion.

Some icons/actions enable to modify the list of calculation targets.

	Add a calculation-target (a line in the list). The target is duplicated from a default calculation target.
	Remove selected computation target.
	Climb selected target up.
	Go selected target down.
	Define default computation-target from selected target. Every new target will be created using this default one.
	Save target-list in a XML file in order to re-use it.
	Load a list of computation targets.
	It enables to define result files associated to targets thanks to a pattern which take into account informations linked to calculation targets. The result file name will be define replacing generation tag %xxx% by their contents : %num % => order number in the list, %var% => variable name, %val% => value to use, ...

Double-click on an element of the list enables to edit it with an appropriate editor.



Target editor opens a window and shows every model variable in a tree view. Select the value corresponding to the feared event to take into account.

File editor displays a standard file chooser window.

Two **Algorithm** are available in order to generate Boolean formula :

1. **Boolean Formula (SOP)** : compile sum of product into simple Boolean formula.
2. **Boolean Formula (ITE)** : compile sum of product into a tree structure 'If ... then ... else ...' (more powerful in case of weighty systems).

Three **Simplify** levels are available:

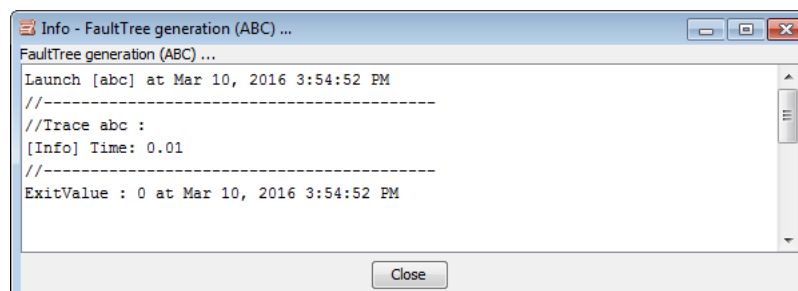
- **Nothing simplification**
- **Constant propagation**: In generated fault tree, boolean connectors having true or false arguments are simplified.
- **Unreferenciation**: simplifies the tree by deleting useless intermediate variables.

When this window is correctly filled and validated, **ABCTree** task is added to the task manager of Cecilia-Workshop

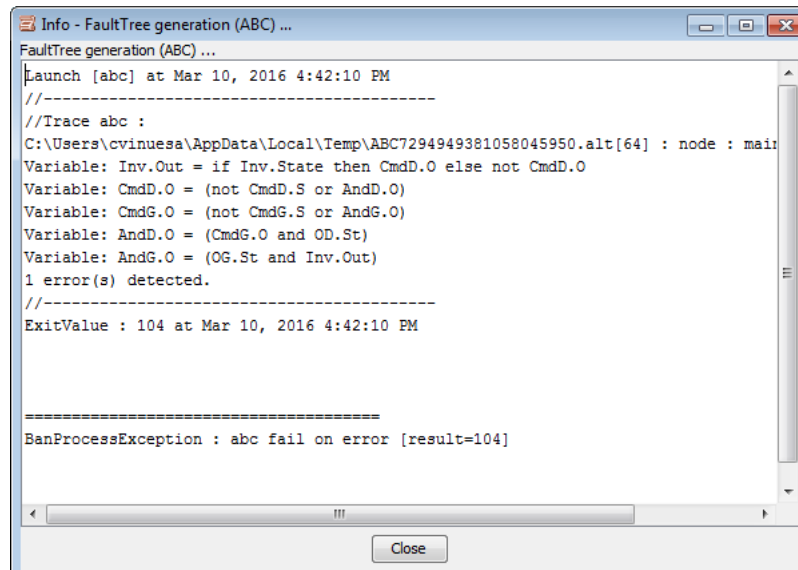
## 6.5.1.4. Result of FaultTree generation

A the end of fault tree generation, a window is displayed. It shows *ABC* execution trace.

If everything is all right, this trace should be empty.



In the contrary (looped system, ...), *ABC* displays an error message.



### 6.5.2. Sequences generation (Generic)

Sequences generation consist in finding every way leading to a specific state (feared event), from an initial state of a system. Number of path to run through are exponential, strategies are set in order to explore the most interesting ones.

It's possible to do a sequences generation from the moment it's possible to do a step by step simulation. This is the main advantage compared with FaultTree generation. So, it's possible to do a sequences generation on a dynamic system (and possibly with loops if the stepper can process systems with loops).

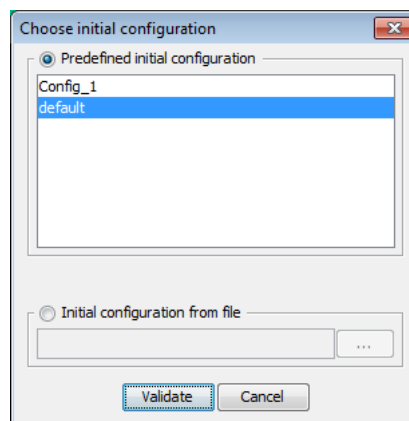
Processing time is proportional to the system size, but exponential to the depth of the ridding through. It implies a long processing time, it's the reason why this method have to be used only when tree generation can't be used (dynamic system and/or looped system).

#### 6.5.2.1. Sequences generation launching



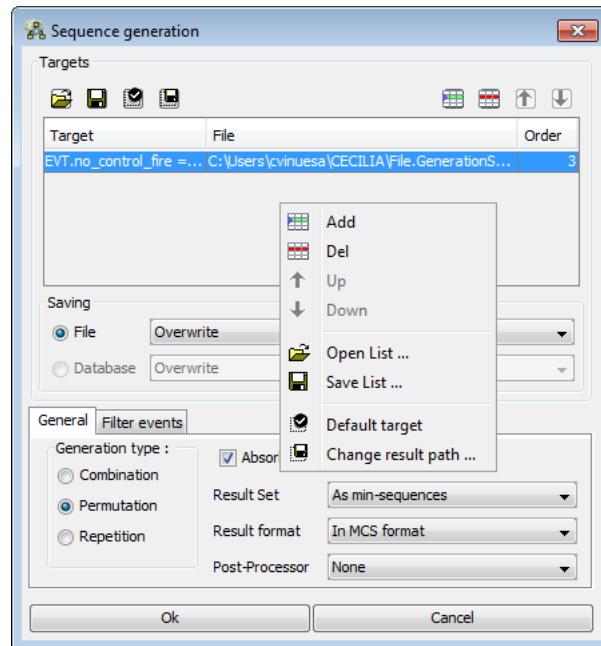
In order to launch sequences generation, use the **Sequences generation (Generic)** command.

A window is displayed when model has many initial configurations (**System** menu, **Initial states ...** command), in order to select initial configuration to take into account.



Feared event is defined thanks to a calculation-target. The target corresponds to a specific value of a model variable (If the variable is equal to the selected target the feared event happens). Calculation parameters are usually associated with a calculation-target (for example, the result file name).









The following dialog window enables to define one (or several) target(s) which must be taken into account during sequences generation. A result file will be generated for each computation target.



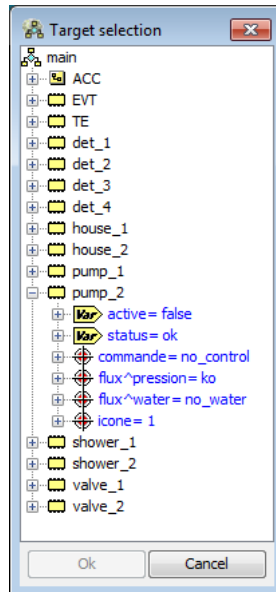
This window manages a list of computation-targets. When this window is validate, every typed information are validate to ensure that there is no incomplete line, no wrong line, targets correspond to a variable of processed model, file can be written ....

Moreover, if result-files already exist, a dialog window will ask for old file deletion.

Some icons/actions enable to modify the list of calculation targets.

	Add a calculation-target (a line in the list). The target is duplicated from a default calculation target.
	Remove selected computation target.
	Climb selected target up.
	Go selected target down.
	Define default computation-target from selected target. Every new target will be created using this default one.
	Save target-list in a XML file in order to re-use it.
	Load a list of computation targets.
	It enables to define result files associated to targets thanks to a pattern which take into account informations linked to calculation targets. The result file name will be define replacing generation tag %xxx% by their contents : %num % => order number in the list, %var% => variable name, %val% => value to use, ...

Double-click on an element of the list enables to edit it with an appropriate editor.



Target editor opens a window and shows every model variable in a tree view. Select the value corresponding to the feared event to take into account.

File editor displays a standard file chooser window.

The third column enables to choice the maximum order of sequences, that's to say the number of events/transitions of a sequence. The transition number doesn't take into account instantaneous transitions if their automatic firing is activated (user preferences).

The **Generation type** specifies the sequences generation strategy.

- **Combination:** During the simulation phases, every combination identified is simulated once in a given order. Within a combination the same event is drawn only once.
- **Permutation:** During the simulation phases, every identified combination is simulated in all orders. Within a combination the same event is drawn only once.
- **Repetition:** Each combination is simulated in all the orders. Within a combination the same event is drawn many times.

If **Absorbent top event** option is checked, sequence-generator stops its exploration when top event is reached.

The **Result set** specifies the type of result:

- **As sequences (Not minimized):** All found sequences will be memorized.
- **As min-sequences:** Only minimal sequences will be memorized.
- **As min-cuts:** Found sequences will be minimized assuming they are minimal cuts, that's to say without ordering notion of event appearance.

The **Format result** specifies the format of result:

- **In MCS format:** Result file contains found sequences in a format which is equivalent to *Aralia* minimal cuts. That is to say:

```
products(MRS('ER.O.true')) =
{ 'CCF_BK', 'CCF_BP', 'AU.def' }
{ 'CCF_BK', 'Op.def', 'AU.def' }
...
{ 'AU.def', 'Op.def', 'Safety.def' }
end
```

- **In Aralia format:** Result file contains found sequences in Aralia equation syntax (with laws, parameters and attributes).
- **In XML format:** Result file contains found sequences in XML format included parameters computation, abstract of result, model informations, ...

```

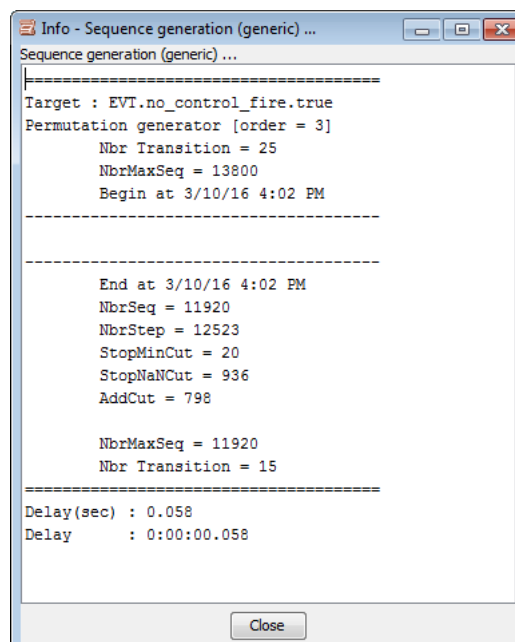
<?xml version='1.0'?>
<seqgen>
  <define>
    <target name="unexpected.unexpectedState" value="true">
      <param name="resultset" value="minseqs"/>
      <param name="finder" value="permutation"/>
      ...
    </target>
  </define>
  <abstract>
    /* Order of products :
    3 132
    */
  </abstract>
  <result>
    <seq><tr id="1" evt="synthesys_2.failure"/>
      <tr id="16" evt="screen_1.failure"/>
      <tr id="19" evt="screen_pilot.and_2.out_1"/></seq>
    <seq><tr id="1" evt="synthesys_2.failure"/>
      <tr id="16" evt="screen_1.failure"/>
      <tr id="27" evt="screen_pilot.relay_2.untimely_close"/></seq>
    ...
  </result>
  <model>
    <flow name="unexpected.unexpectedState" domain="bool" orientation="out"></flow>
    <state name="location_module.bus._state" domain="{no_ok, ok}">
      <init value="ok"/></state>
    <state name="location_module.calculnum1.state_" domain="{no_ok, ok}">
      <init value="ok"/></state>
    ...
    <event name="location_module.bus.failure">
      <law value="exponential(1.0E-4)" aralia="..." moca="..."></event>
    <event name="rescue.failure">
      <law value="exponential(1.0E-4)" aralia="..." moca="..."></event>
    ...
    <nodeproperty name="projectName" value="Exemples/AIRCRAFT_AIRCRAFT_LOCATION"/>
    ...
  </model>
</seqgen>

```

When this window is correctly filled and validated, **SeqGen** task is added to the task manager of Cecilia-Workshop

### 6.5.2.2. Result of sequences generation

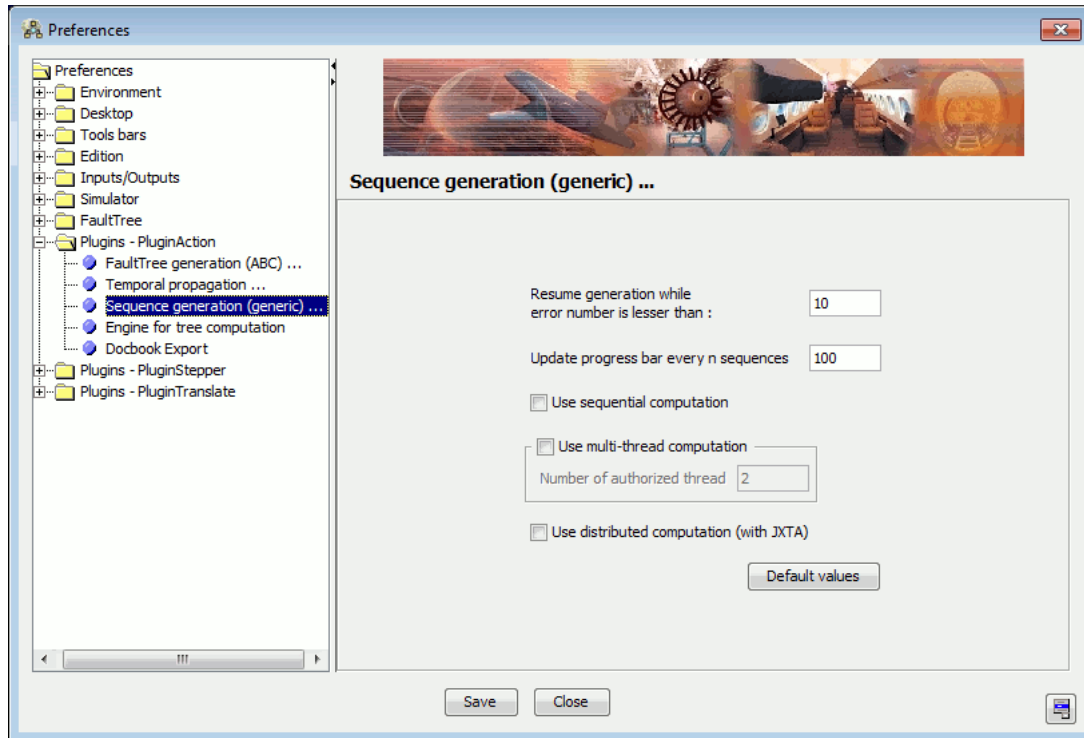
If everything is all right, for each computation target, a window displays a summary of computation parameters, statistics, and possible error-messages.





Sequences generator can come up against inconsistencies of the system, or instantaneous loop (that's to say a series of too many instantaneous transitions)

## 6.5.2.3. User preferences



User preferences allow sequences generation configuration.

- The allowed error number enables to continue sequences generation despite errors, assuming number of errors is below authorized one. This option allows not to stop generation after first error, so it's easier to debug.
- **Update progress bar every n sequences** enables to display sequences generation progress inside 'task manager'.
- **Use sequential computation** is used to generate sequences one after the others when there are several targets.
- If **Use multi-thread computation** is checked, the sequence generation is made in parallel, in several processors at the same time.
- In case of **distributed computation** Grid computing is used.

## 6.5.3. FMEA generation help

FMEA generation help aims at synthesizing consequences of unitary failures on models. It consists of visualization of changes made on system for each component-elementary-failure.

It's important to notice - in my view - that a system-FMEA has to be done before every modeling. It enables to identify every constituent failure mode to take into account during modeling.

This tool enables to respond to different needs.

It can be useful:

- To make base of an FMEA for system documentation.
- To validate system modeling by comparing this generation result to a preliminary FMEA.
- To be the base of reflection for breakdown diagnostic. Eventually, the objective is to find a minimal set of significant signals (flow variable values) allowing to determine location of a maximum number of breakdown. Signature associated with a set of signals should allow to find the component to be repaired or to be changed.

Overall principles of algorithm consists in starting from an initial state and firing each model transition, and then display every model variables.

## 6.5.3.1. FMEA generation help launching



In order to launch FMEA generation, use the **Help to FMEA generation** command.

The following dialog window enables to define FMEA generation parameters.

Result will be stored in a text file. The ... button linked to **Result file** displays a file chooser.

Events to be taken into account during transition firing can be filtered. State or flow variables to be displayed in result can also be filtered. To do this, check corresponding box and enter pattern allowing to filter data.

Pattern is defined thanks to Java regular expression that must be applied on the name of filtered data.

Exemples :

- 'def' => Everything containing 'def' in its name
- 'def|ccf' => Everything containing 'def' or 'ccf' in its name  
(to restrict to failures)
- '\$Equ1\.' => Everything beginning whit 'Equ1'. (to restrict to equipments)
- 'diag^' => Everything ending whit 'diag'.

Associated with a strict naming rule, filters can restrict generation to data which are important for analysis.

The **Only value changed variables** Check-box filters results in order to hide variables whose values are the same as their values at initial state.

When this window is correctly filled and validated, **GenFMEA** task is added to the task manager of Cecilia-Workshop. On a standard system, this task shouldn't take a long time.

## 6.5.3.2. Result of FMEA generation

If everything is all right, task ends without error.

Result file is like:

```
<?xml version="1.0"?>
<fmeagen>
  <define>
    <param name="onlychange" value="true" />
  </define>
  <result>
    ...
```

```

<tr id="6" evt="B1.EA.def">
  <state name="B1.EA.Def" value="true"/>
</tr>
<tr id="5" evt="B1.Kb.def">
  <state name="B1.Kb.Def" value="true"/>
</tr>
...
<tr id="3" evt="CCF_B">
  <state name="B1.Kb.Def" value="true"/>
  <state name="B1.Kh.Def" value="true"/>
  <state name="B2.Kb.Def" value="true"/>
  <state name="B2.Kh.Def" value="true"/>
  <state name="B3.Kb.Def" value="true"/>
  <state name="B3.Kh.Def" value="true"/>
</tr>
...
</result>
<model>
  <flow name="AU.Stop" domain="bool" orientation="out">
    <init value="false"/></flow>
  <flow name="B1.BNeg" domain="KTest_Potentiel" orientation="in">
    <init value="Neg"/></flow>
  ...
  <state name="AU.Def" domain="bool">
    <init value="false"/></state>
  <state name="B1.EA.Def" domain="bool">
    <init value="false"/></state>
  ...
  <event name="AU.def">
    <law value="constant(1.0)" aralia="constant 1.0" moca="drc 0"/></event>
  <event name="B1.EA.def">
    <law value="exponential(EALbd)" aralia="exponential EALbd " moca="exp g..EALbd"/></
event>
  ...
  <parameter name="&apos;B1.Kh.lbd&apos;" value="0.0010" aralia="0.0010" moca="0.0010"/>
  <parameter name="&apos;B1.Kb.lbd&apos;" value="0.0010" aralia="0.0010" moca="0.0010"/>
  ...
</model>
</fmeagen>

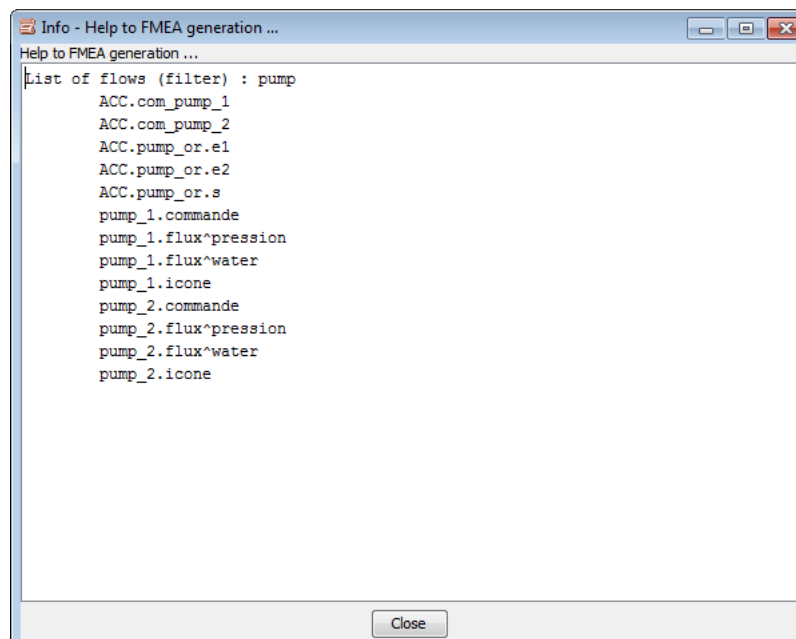
```

It displays first the parameters of the generation.

It displays event name of each transition followed by the value of each system variable (here, having their value changed).

To end, it displays information about system flows, states and events implied in these results.

If data has been filtered, a window displays data that have been taken into account for generation:



### 6.5.4. Others tools

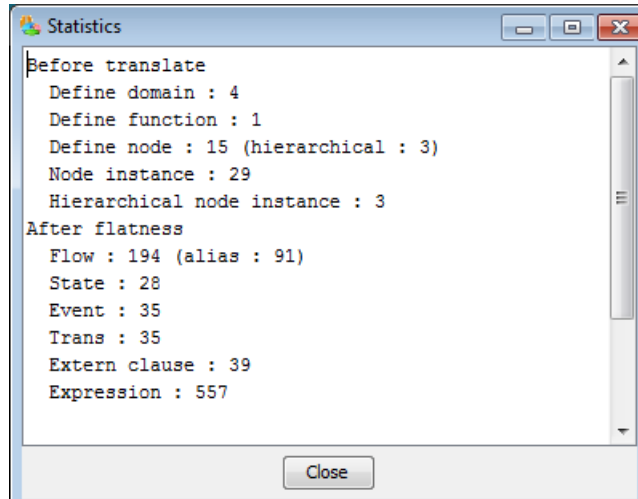
#### 6.5.4.1. Statistics



**Statistics** command enables to display information about current model size.

Statistics module relies on translation module. Syntax errors, grammatical errors, or errors of setting flat, suspend statistics display.

On the contrary, **Statistics** window is displayed.



Statistics are parted in two categories:

1. Before translation (Cf. Section 3, “Translation into 'standard' AltaRica”): Display number of high level objects which are manipulated (domains, operators/functions, nodes/components ...)
2. After setting flat (Cf. Section 6, “Setting flat of model”): Display number of low level objects (flow variables, state variables, events, transitions, ...)

Alias match flow variable defined with equality of type `<out> = <var>` where `out` is a flow variable and `var` a variable (flow or state).

#### 6.5.4.2. Verification of sequences

Verification of sequences consists in replaying results of sequences generation on the current model, in order to display possible differences.

The function needs a result of sequences generation in XML format (other formats are not managed).

Information extracted from this file are:

- sequences to be played
- target of computation
- initial state of model

A simulation is launched on current model. This simulation is initialized with initial state found in file. Then, each sequence of result file is played. If there is an issue during progress of sequence, an error message is displayed. When a sequence is entirely played, computation target must be verified. If it is not verified, an error message is displayed.

##### 6.5.4.2.1. Launch verification of sequences

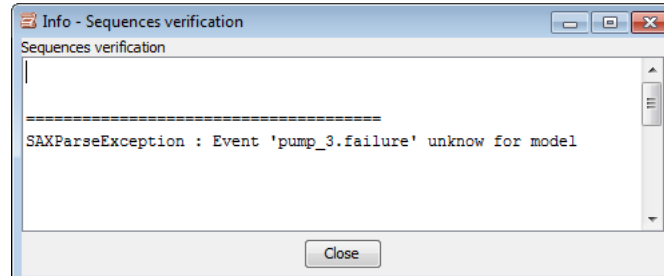


In order to launch sequences checking, use **Verification of sequences ...** command.

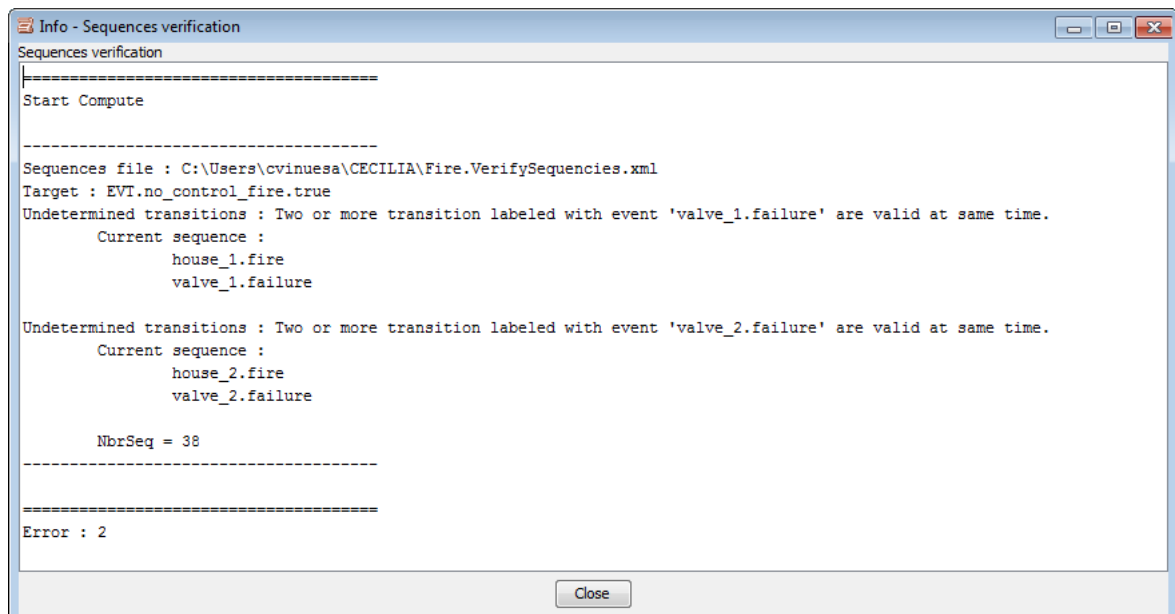
A window is displayed, it enables to select file containing sequences to be verified. Once file is selected, a **SeqVerif** task is added to task manager of Cecilia-Workshop.

### 6.5.4.2.2. Results of verification of sequences

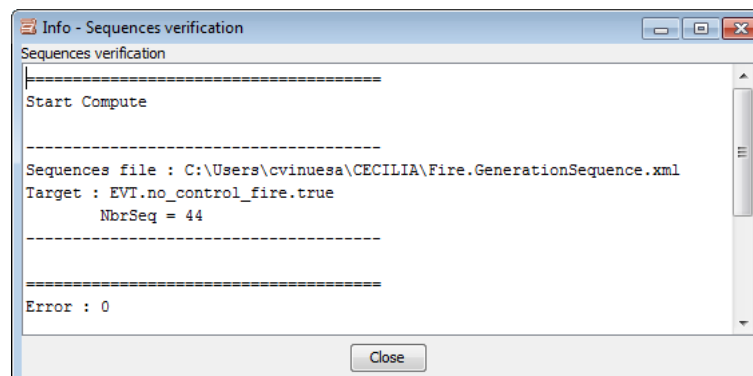
If model is not compatible with result file, either in events linked to transitions of sequences, or at computation target level or at initial setup level, an error message is displayed and verification doesn't start.



If some sequences have problems, an error message is displayed.



Finally, if everything works fine, the trace window displays the number of sequences that have been done.

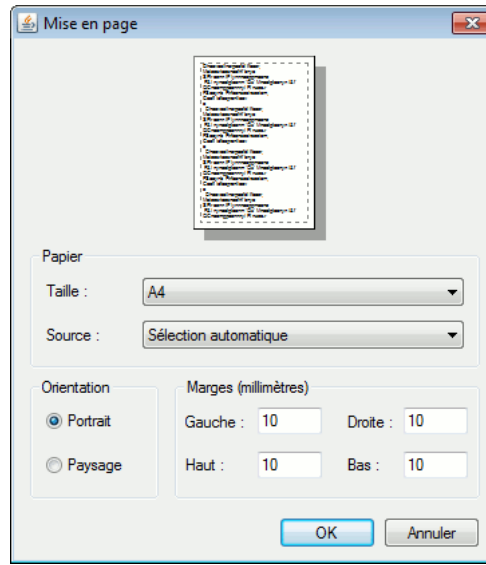


## 7. Input / Output

### 7.1. Setup

Prepare the printing of a model or system or of the System links window as follows:


- Select the **File - Printing format** menu.
- The Page Setup window is displayed and in the Paper area, select:



- The size (A4 or A3);
- The source (manual feeding or automatic selection).
- In the Orientation area, tick the Portrait or Landscape mode.
- Each format or orientation modification automatically updates the scale factor of the page displayed.
- In the Margin area, type the margins (left, right, up or down).
- Click on the **OK** button to save the new printing format, or click on the **Cancel** button to return to the previous format prior to this window opening.

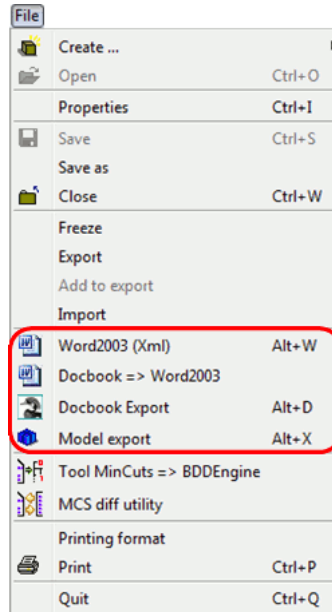
### 7.2. Print / Data report

#### 7.2.1. General principle


To print a model, user can use the  in the **File - Menu** to print the current view.

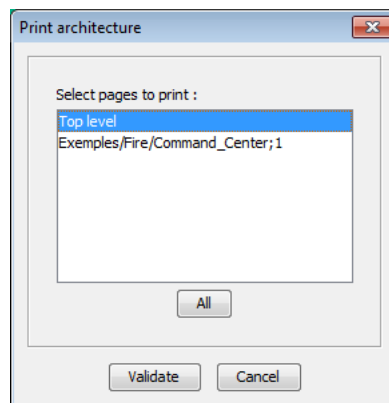
## 7.2.1.1. Model and Modelisation

The user can export the model display in the current page using the different exports available in the **File** menu:



To print a modelisation, user can proceed in the following way:

- Select the system version you want to print, in tree of the considerate model;
- Open Modelisation;
- Select the **File - Print** menu or click on the  icon to print selected system. When printing architecture, select the pages to be printed.



- Click on the **Validate** button, or click on the **Cancel** button to abort the printing;




In the **Preferences window - Options** tab, it is possible to select a file containing a logo which will be added when printing.

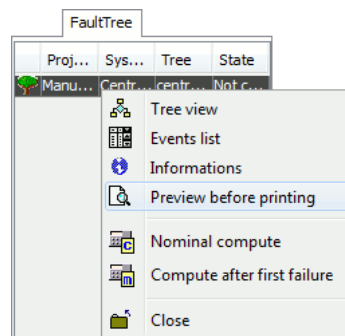
## 7.2.1.2. Print Tree


The graphical representation of a Tree in a file can be print using the following procedure:

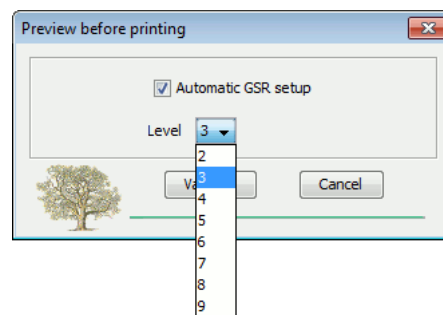
- Select the tree to be saved in a file;



- Click on the  icon in the tool bar or click with the right mouse button on the tree icon in the opened tree list

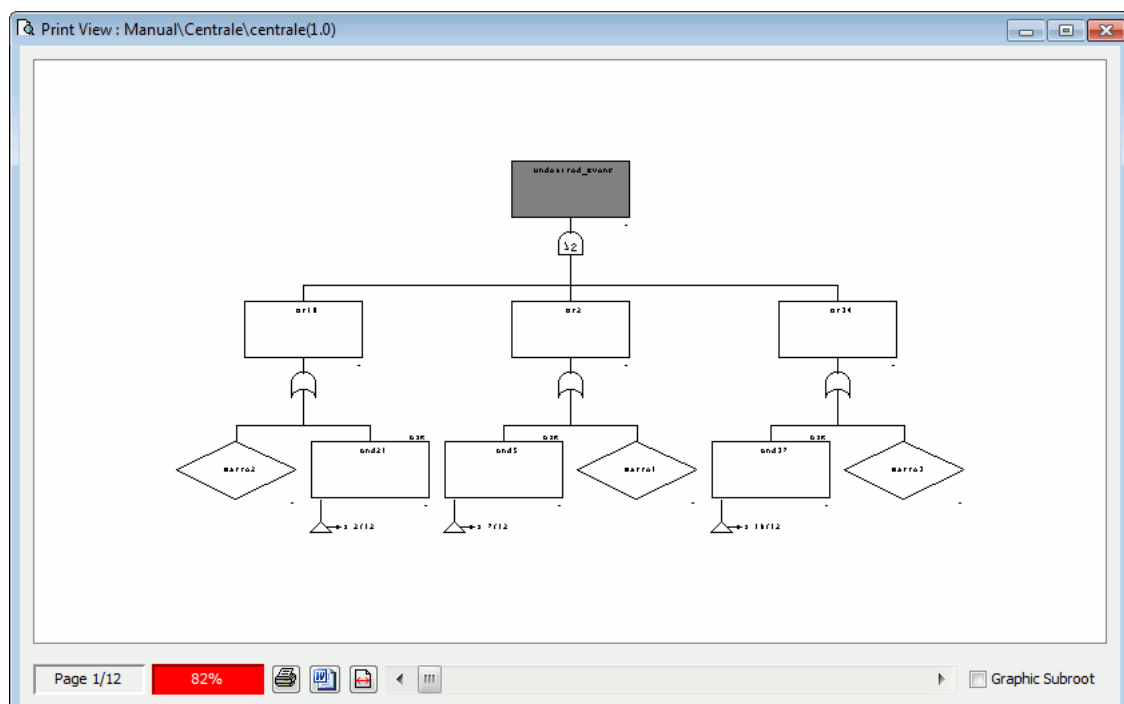



and select the  Preview before printing option and the following window appears:

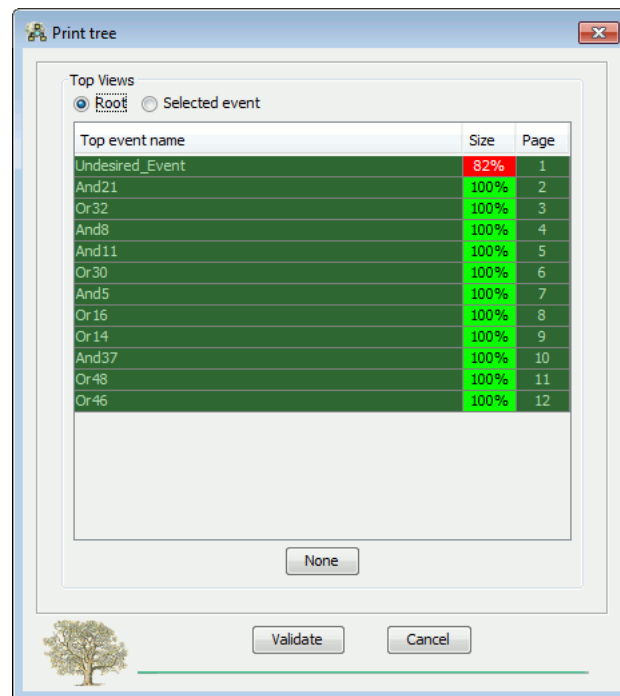



This window allows the user to choose the level of the Graphical Sub Roots either automatically or manually. It means the level which the tree is cut in a point of view of the graphical representation.

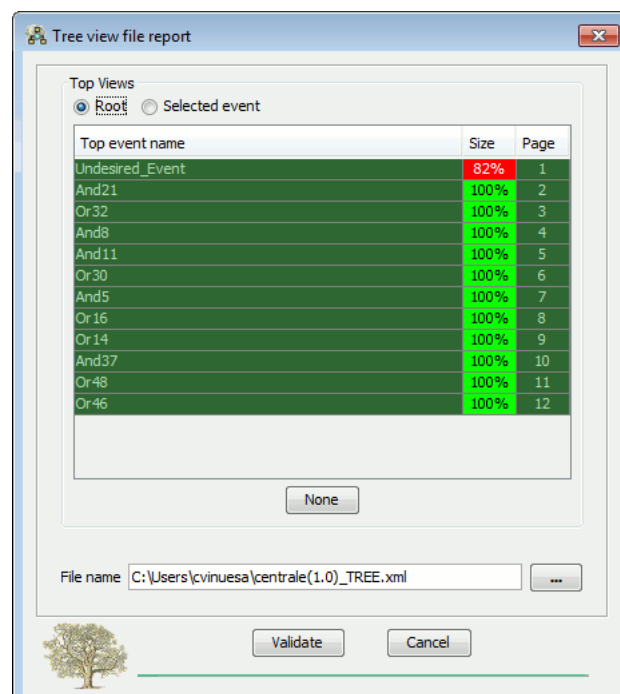
If the user clicks on the **Validate** button, the following preview window then appears:




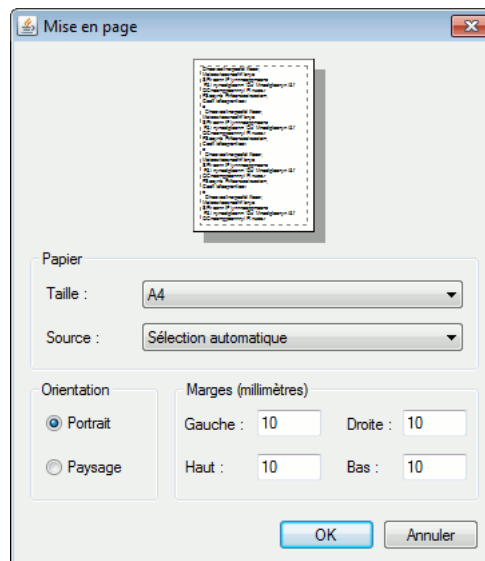
The icon  permits to print the tree representation. The following window appears to choose the options:



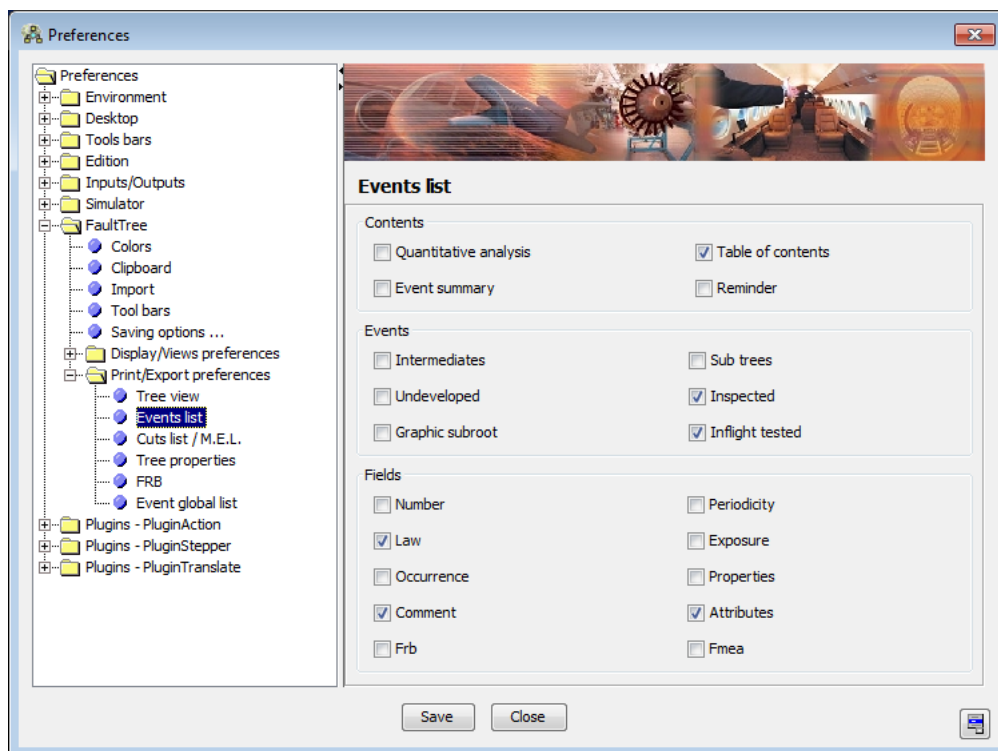
The icon  permits to export the Tree in xml version which can open with Word. The following window appears to choose the options:



The icon  permits to open the Page Setup window.



In the **Preferences window - Options** tab, it is possible to select a file containing a logo which will be added when printing.



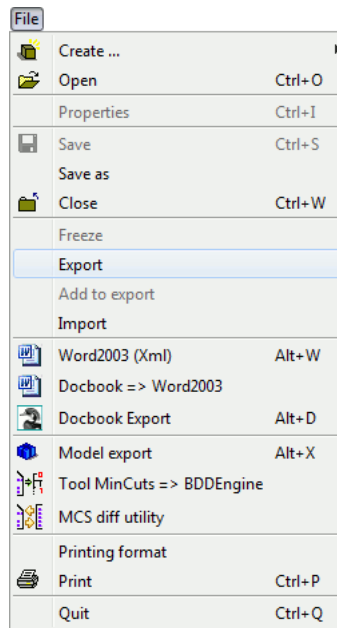
## 7.3. Import/Export data into XML format

In order to facilitate data exchanges, Cecilia-Workshop allows export and import of every Cecilia-Workshop modelisation (models, component, equipment, type, operator) into XML Format.

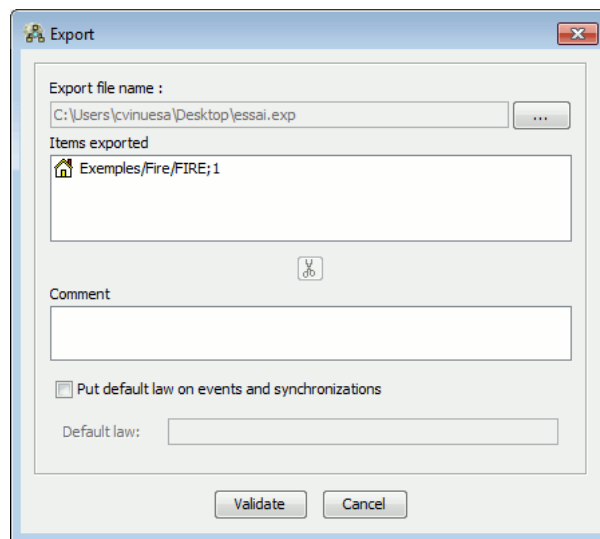
### 7.3.1. MBSA models Export


To export data, user must proceed as follows:

- Select **Export** command of **File** menu;



- Activate the export setup;



- Click on  button in Export File Name Area;
- Chose the export file,
- The user must now add the elements that have to be exported. Select an element in the object manager and use the **File/ Add to Export** menu, or Drag and Drop the object in "Items exported" area.
- The Comment area allows exported data description.
- If **Put default law on events and synchronizations** is checked during export: the default law is only put on events and synchronizations that have no law. Use the Default law area to enter the default law
- Click on **Validate** button to export or on **Cancel** to cancel exportation

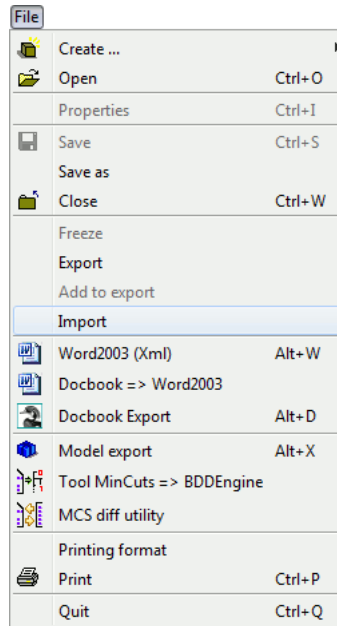


Freeze Objects can't be exported.

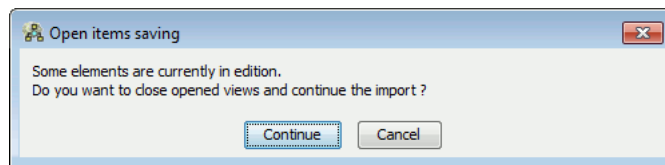
## 7.3.2. MBSA models Import

To import data, user must proceed as follows:

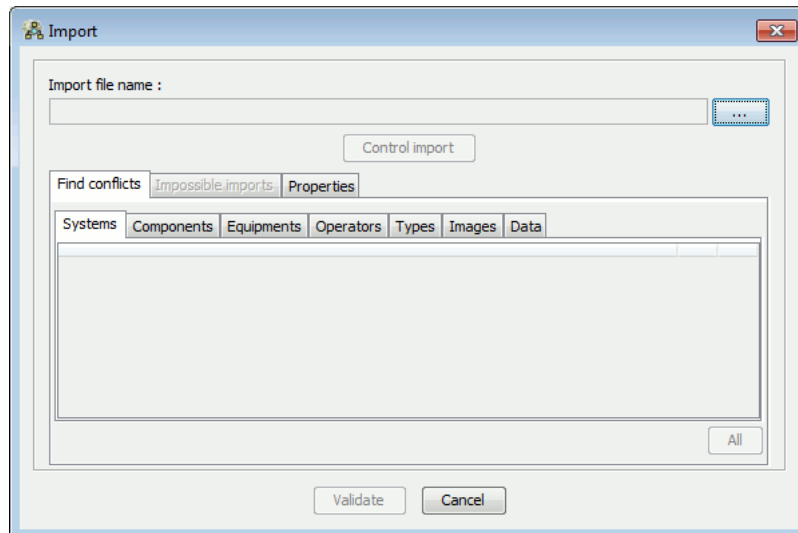
- Select **Import** command of **File** menu;




- If documents still open, the following window appears to close the project:




- Activate the export setup;





- Click on  button in Import File Name Area;
- Chose the export file (extension .exp) containing Cecilia-Workshop data. The **Properties** tab allows visualization of properties associated to the imported data (creation date, comment, and owner);
- Click on Check Import to launch validity check. This check do the following operations:
- XML Format validity Check. Detected errors are displayed un "impossible imports" to: data access issues (family or project reachable in read only mode, model or system version unreachable or reachable in read only mode)

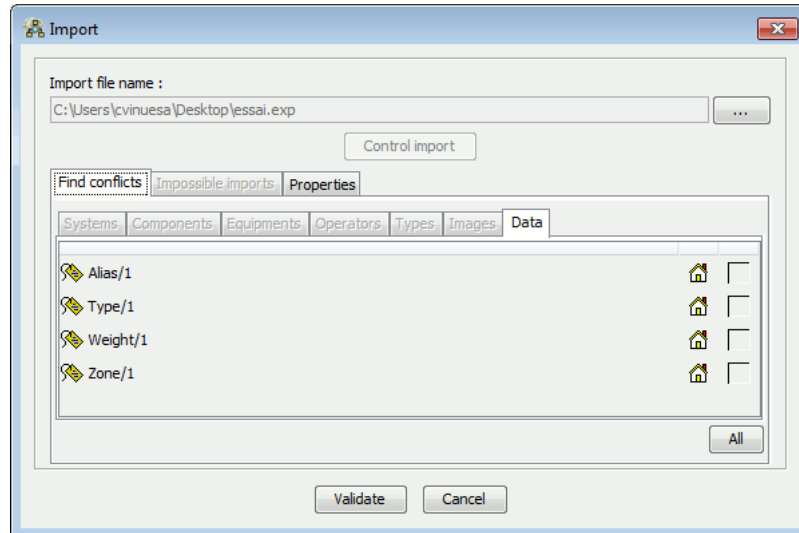
 Only ascendant compatibility is ensured. (File exported with 3.1 version can't be imported with previous version).

- Detection of conflict between imported data and data in database.

 This difference detection is made essentially with the last modification date of each imported element compared to the one in the database. Detected conflicts are displayed depending on their type in the following tabs: Systems,

Components, Equipment, Operators, Types, and Images. By default, data in database are not updated during data importation.

Nevertheless, the user can specify, for each conflict, an update of the database with a click on  . To indicate this update, the lock icon is associated to the imported element. The **All** button allows update of all displayed conflicts in the tab.



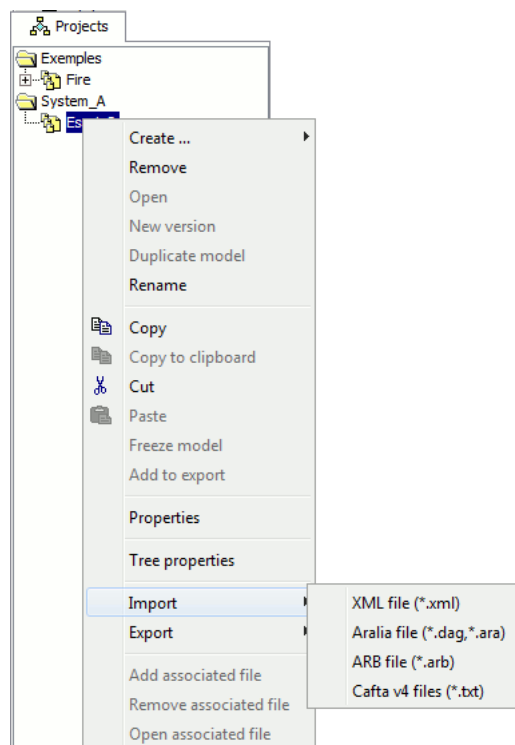
- Click on **Validate** to import or **Cancel** to cancel importation.

### 7.3.3. Fault tree models Import

This command allows the user to import projects, systems or trees in the database from XML files.

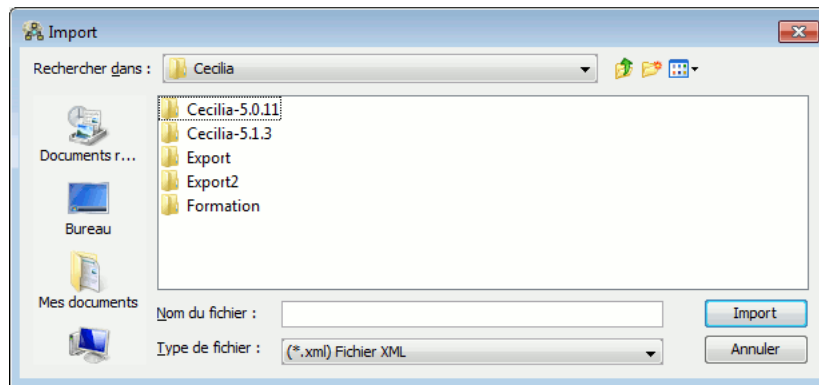
The files must contain either projects or systems or trees but not different types of objects at the same time.

To import data from XML files, proceed as follow:



- Click on the mouse right button in the manager to gain access to the pop-up menu;
- Select the XML import option.

- The following window appears:



The user selects the XML file to be imported and click on the **Import** button to start the import or on the **Cancel** button to cancel his action.

Two cases can occur at this time: either the XML file contains projects or it contains systems or trees. In the first case, import proceeds without any further information.

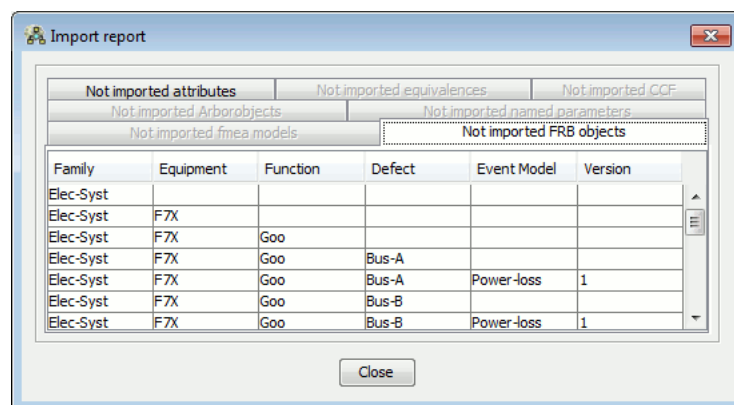
The user has to select the node under which he wants the imported data to appear: either a system if the file contains trees or a project if the file contains systems.

This window shows the data (projects, systems and trees) being imported.

At the end of the import, a report is shown to the user if needed. It shows the whole not imported data. This one is that which is already in the data base. It can be:

- Project elements (projects, systems or trees);
- Named parameters;
- Attributes;
- Equivalencies;
- FRB elements.

Each data type is presented in a dedicated tab as follows:



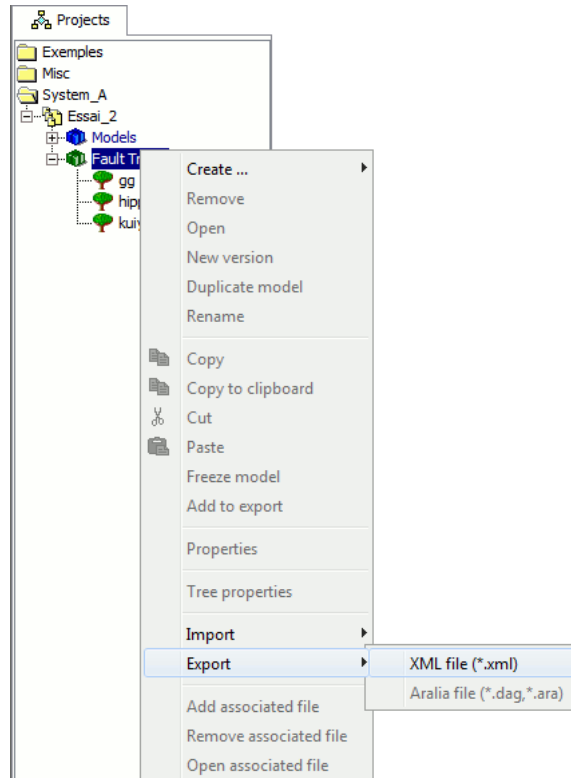
## 7.3.4. Fault tree models Export

This command allows the user to export projects, systems and trees contained in the database to XML formatted files.

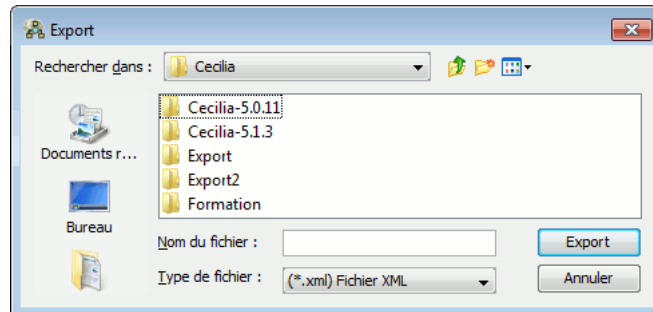
To export either projects, systems or trees, proceed as follows:

- Select either projects, systems or trees in the document manager;
- Click on the mouse right button to gain access to the pop-up menu;

- Select **XML export** option.



The following window appears:



The user selects the XML file to use to export the data inward and click on the **Export** button to start the export or on the **Cancel** button to cancel the action.

## 7.4. Fault trees DAG Import/Export

### 7.4.1. Import

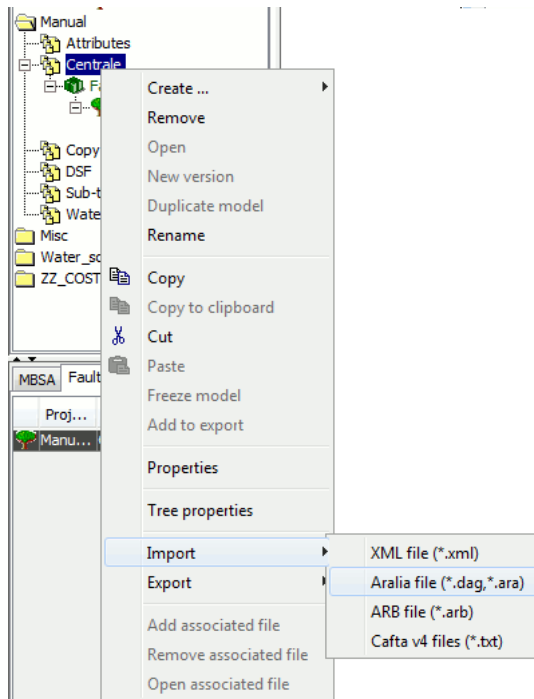
This command allows the user to import trees contained in Aralia formatted files (extensions DAG or ARA).

**To import a tree contained in a Aralia file** , proceed as follow:

- Select a system in the document manager;
- Click with the mouse right button on the document manager to gain access to its pop-up menu;



- Select **Import - Aralia file (\*.dag, \*.ara)** option.

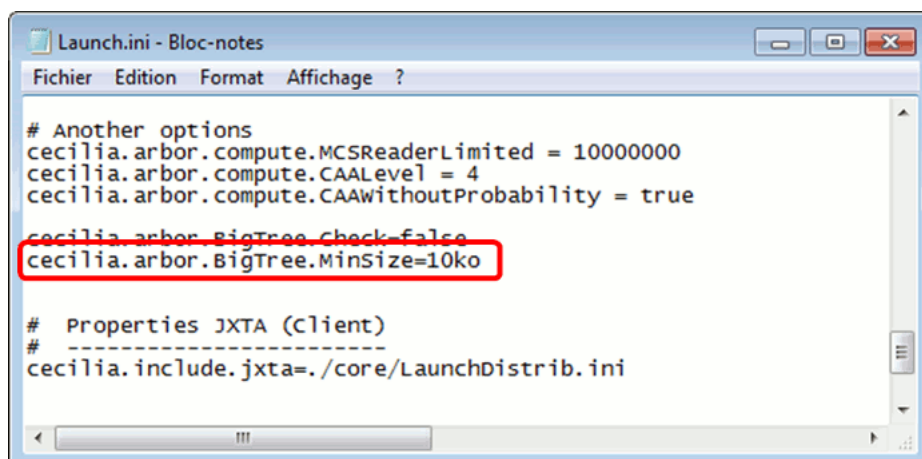


The user selects the file to be imported and press the **Open** button to start the import or the Cancel button to cancel its action. At the end of the import, the new created tree appears under the previously selected system node in the document manager.

## 7.4.2. Big tree

For some studies, the Aralia equations are huge (Aralia files of several dozens of Mbytes). These trees equations are not editable by Cecilia-Workshop software. In these cases, FaultTree tool offers a limited edition by enabling modification only on basic events and the global tree is not displayed. Qualification of big sized trees is determined at the import.

User can define the minimum size to consider a Fault Tree as Big Tree in the `Launch.ini` file.

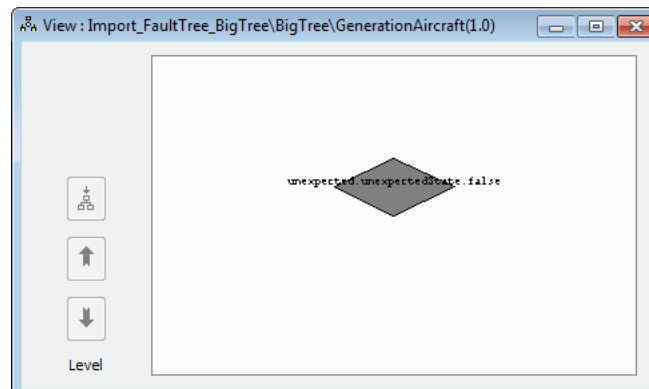


Exported files with a size superior to 10ko will be imported as equations BigTree

The consequences for a tree to be a big tree are the following:

- Their edition is limited (see Section 5.1.2, “Events edition” );

- It is not possible to modify the structure (structure not visible in the TreeView);



- However, in the event list user can see all the events and the root event:

Events	Comments	Given laws	Inspected	Attributes
location_module.bus.fail...		exponential 1.000E-04		--
location_module.calcul...		exponential 1.000E-04		--
location_module.calcul...		exponential 1.000E-04		--
location_module.calcul...		exponential 1.000E-04		--
location_module.calcul...		exponential 1.000E-04		--
location_module.calcul...		exponential 1.000E-04		--
rescue.failure		exponential 1.000E-04		--
screen_1.failure		exponential 1.000E-04		--
screen_2.failure		exponential 1.000E-04		--
screen_3.failure		exponential 1.000E-04		--
screen_pilot.and_2.out_0		exponential 1.000E-04		--
screen_pilot.and_2.out_1		exponential 1.000E-04		--
screen_pilot.and_3.out_0		exponential 1.000E-04		--
screen_pilot.and_3.out_1		exponential 1.000E-04		--
screen_pilot.not_1.out_0		exponential 1.000E-04		--
screen_pilot.not_1.out_1		exponential 1.000E-04		--
screen_pilot.not_2.out_0		exponential 1.000E-04		--
screen_pilot.not_2.out_1		exponential 1.000E-04		--
screen_pilot.relay_1.un...		exponential 1.000E-04		--
screen_pilot.relay_1.un...		exponential 1.000E-04		--
screen_pilot.relay_2.un...		exponential 1.000E-04		--
screen_pilot.relay_2.un...		exponential 1.000E-04		--
screen_pilot.relay_2.un...		exponential 1.000E-04		--

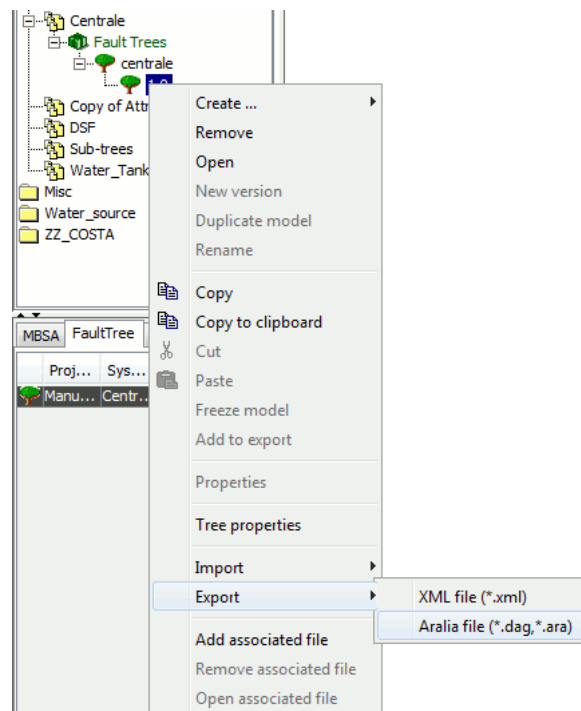
- Using the event list, user can modify basic events: modify the type of laws, the value, the inspection, link to a FRB but it is not possible to link to a Sub-Tree;
- It is possible to print the event list;
- It is not possible to print the Tree;
- It is not possible to export in DAG version;
- It is possible to export in XML version;

## 7.4.3. Export

To export a tree in a DAG format , proceed as follow:

- Select a Tree in the document manager;
- Click with the mouse right button on the document manager to gain access to its pop-up menu;

- Select **Export - Aralia file (\*.dag, \*.ara)** option.

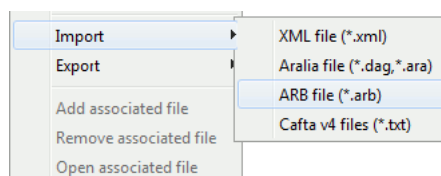


The user selects the file where he wants to save the export file.

## 7.5. Others import format possible

It is also possible open the following fault tree format:

- ARB file :
- Cafta V4 files



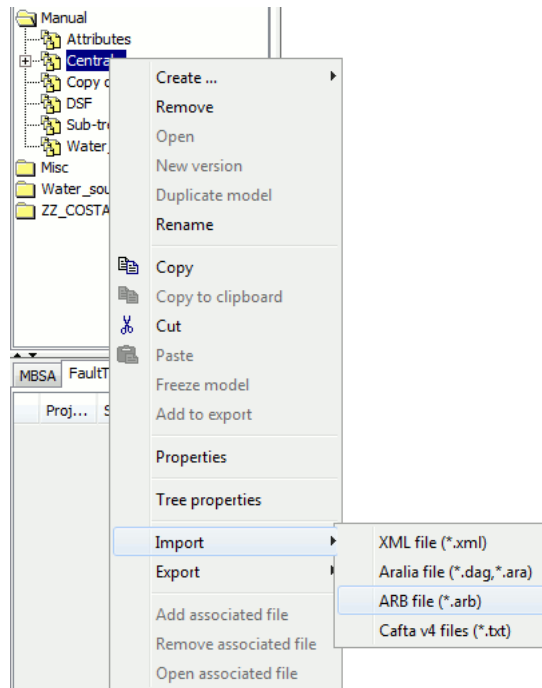
### 7.5.1. ARB file import

This command allows the user to import trees contained in ARB formatted files.

To import a tree contained in an ARB file, proceed as follow:

- Select a system in the document manager;
- Click with the mouse right button on the document manager to gain access to its pop-up menu;

- Select **Import - ARB file (\*.arb)** option.



The user selects the file to be imported and press the **Open** button to start the import or the Cancel button to cancel its action. At the end of the import, the new created tree appears under the previously selected system node in the document manager.

## 7.5.2. Cafta file import

This command allows the user to import fault trees from Cafta files. Cafta export its data in several files. The file data types are as following:

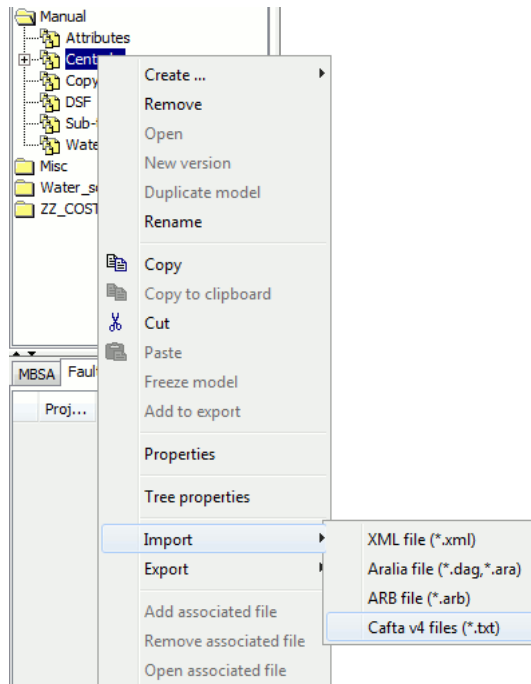
- Gate list in alphabetical order;
- Gate list in a depth first order;
- Gate description;
- Basic event probability/description;
- Basic event full definition (with type code).

One of the gate list file is required as well as one of the basic event description file. The gate description file is optional.

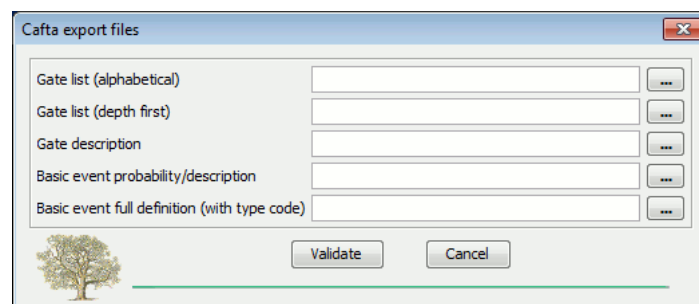
To import a tree contained in Cafta files, proceed as follow:

- Select a system in the document manager;
- Click with the mouse right button on the document manager to gain access to its pop-up menu;

- Select **Import - Cafta v4 file (\*.txt)** option.



The following window appears:



- Select the files to be imported using ;
- Press the **Validate** button to start the import or the **Cancel** button to cancel its action.

At the end of the import, the newly created tree appears under the previously selected system node in the document manager.

## 8. FMEA

### 8.1. Main principles

The FMEA part contains two notions under Cecilia: the notion of FMEA model and the notion of FMEA table.

The FMEA models allow to define the different relationships between entities appearing in the studied FMEA model and the associated table structure and presentation. An entity-relationship model is used to describe entities and relationships present in the model. The FMEA model management is done throughout a dedicated window.

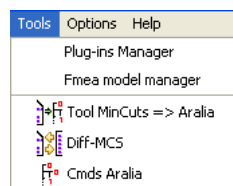
The FMEA models are used during FMEA tables building. Then, an FMEA table refer an FMEA model. Under Cecilia, the FMEA tables are managing in the same way as the fault trees. They are, in the document manager, attached under system folders and are managed in versions.

The FMEA table data can be referenced by rate under the Failure Rate Bank. In the same way, in an FMEA table, a numerical data in a cell can be referenced by another cell in another table.

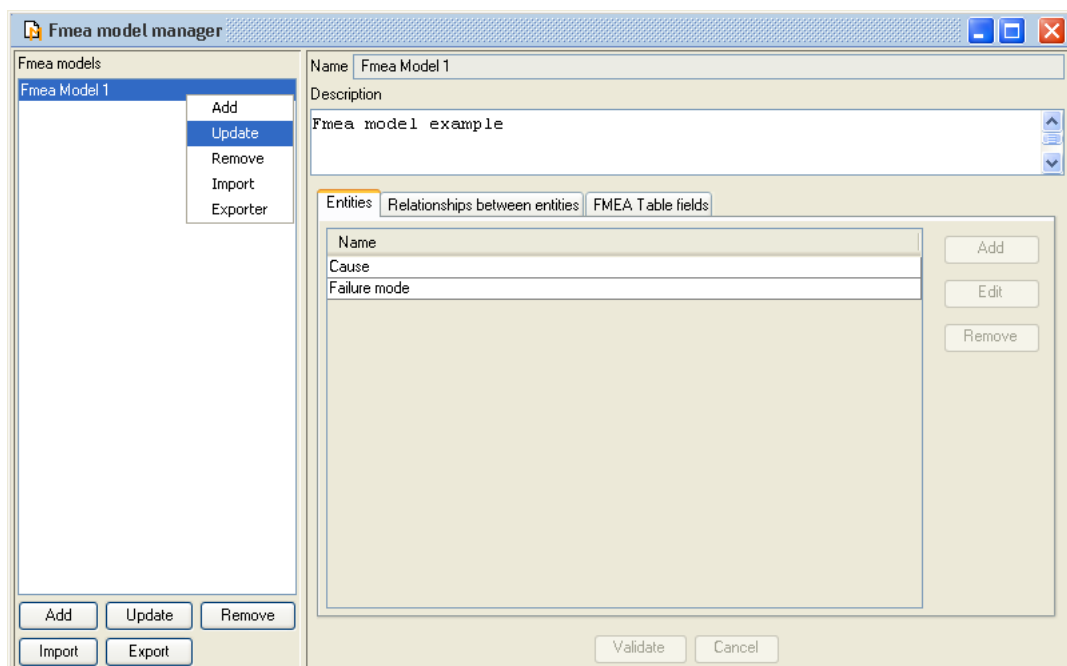
### 8.2. FMEA models

#### 8.2.1. The FMEA model manager

The FMEA model manager window is accessible with the **FMEA model manager** command located in the **FaultTree Tools** menu as shown on the following figure:



The following figure shows the FMEA model manager window.



This window allows to do all the actions required to manage FMEA models.

The following actions are provided to the user:

- **Add** a new FMEA model
- **Update** the selected FMEA model
- **Remove** the selected FMEA model
- **Import** an FMEA model contained in an XML formatted file
- **Export** the selected FMEA model into an XML formatted file

The FMEA model manager window contains two main areas.

The left part of the window shows the existing FMEA model list. This part allows the user to select one or some models on which apply an action.

The right part of the window shows the FMEA model definition selected in the list on the left part. It contains several fields as the FMEA model name, its description as well as three tabs, the goal of which will be described below in the document.



The right part of the window is enabled only when **Add** and **Update** commands are used.

### 8.2.2. Creating a model

To create an FMEA model from scratch, the user launches the **Add** command. The right part of the FMEA model manager window became enabled to modification. The only requested field is the **Name** field.

In order to validate the model creation, the user presses the **Validate** button. If he wants to cancel all the modifications, he presses the **Cancel** button.

In order to be usable, a model has necessarily to contain at least one column then at least one entity model.

An FMEA model building is realized by the three following steps:

- Creation of the entity models,
- Creation of the relationships between entity models,
- Definition of the entity model path used in the model based tables.
- Creation of the model based table columns.

The following chapters describe in details each of these steps.

#### 8.2.2.1. Creating an entity model

An entity model owns an identifier and a set of attributes. This information has to be filled in the **Entities** tab in order to validate the entity model creation.

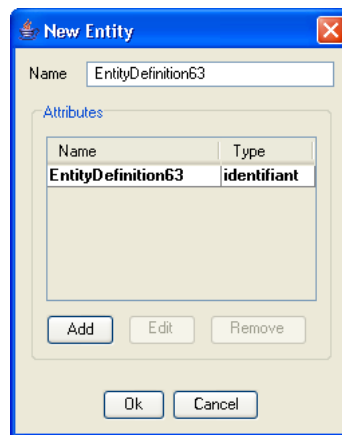
The model identifier allows the user to identify in a formal way base, one entity of the model. So, it must be unique for the concerned model. Concretely, the identifier is an attribute having as name, this one of the model and as type, the special one named 'identifiant' derived from the 'string' type. An identifier is then a character string.

Even though it is displayed in the attribute table, the 'identifiant' attribute is by nature neither modifiable concerning its definition not removable. Its presence in the table is just to inform the user about its existence and its importance for its future use.

The identifiers are used at several points:

- when referencing an 'identifiant' attribute while defining a column,
- when using an 'identifiant' attribute while matching file fields with FMEA table columns on a textual import,
- etc.

To create an entity model, the user selects the **Entities** tab and presses the **Add** button to show the following dialog window:

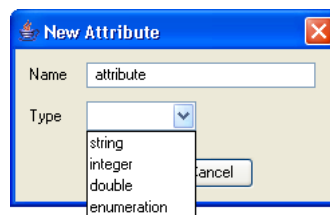


The 'New Entity' dialog window has a title bar with a close button. It contains a 'Name' text field with the value 'EntityDefinition63'. Below it is a section titled 'Attributes' containing a table with two columns: 'Name' and 'Type'. The table has one row with 'EntityDefinition63' in the 'Name' column and 'identifiant' in the 'Type' column. Below the table are three buttons: 'Add', 'Edit', and 'Remove'. At the bottom of the dialog are 'Ok' and 'Cancel' buttons.

Name	Type
EntityDefinition63	identifiant

The dialog window contains a field that contains the entity model name to create or to modify and an area containing the entity model attribute list. During the entity model creation process, the **Name** field is filled by default with an automatic generated name. The user is invited to change this string.

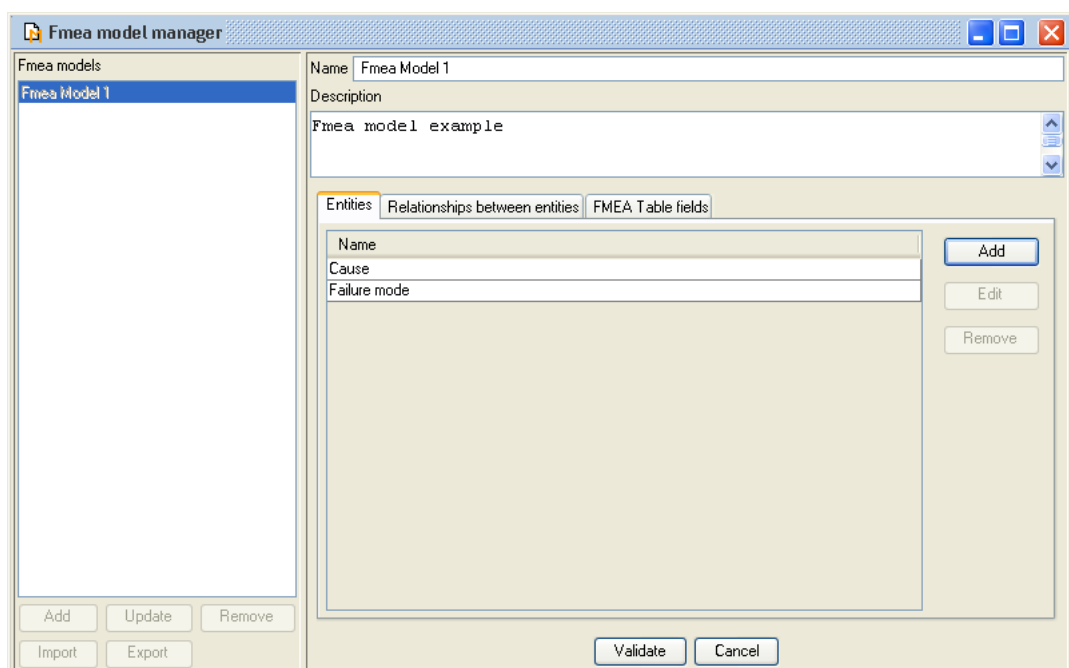
To add new attributes, the user presses the **Add** button on this dialog window in order to show the following dialog window:



The 'New Attribute' dialog window has a title bar with a close button. It contains a 'Name' text field with the value 'attribute'. Below it is a 'Type' dropdown menu with a list of options: 'string', 'integer', 'double', and 'enumeration'. To the right of the dropdown is a 'Cancel' button.

The user fills the name of the attribute in the **Name** field, select its type among the possible ones (string, integer, double or enumeration) and confirm the modifications by pressing the **Validate** button. If he wants to cancel all the modifications, he presses the **Cancel** button.

Supposing that the user has created the both following entity models: 'Failure mode' and 'Cause' during the 'Fmea Model 1' FMEA model creation or modification, the FMEA model manager window will appear as follows :



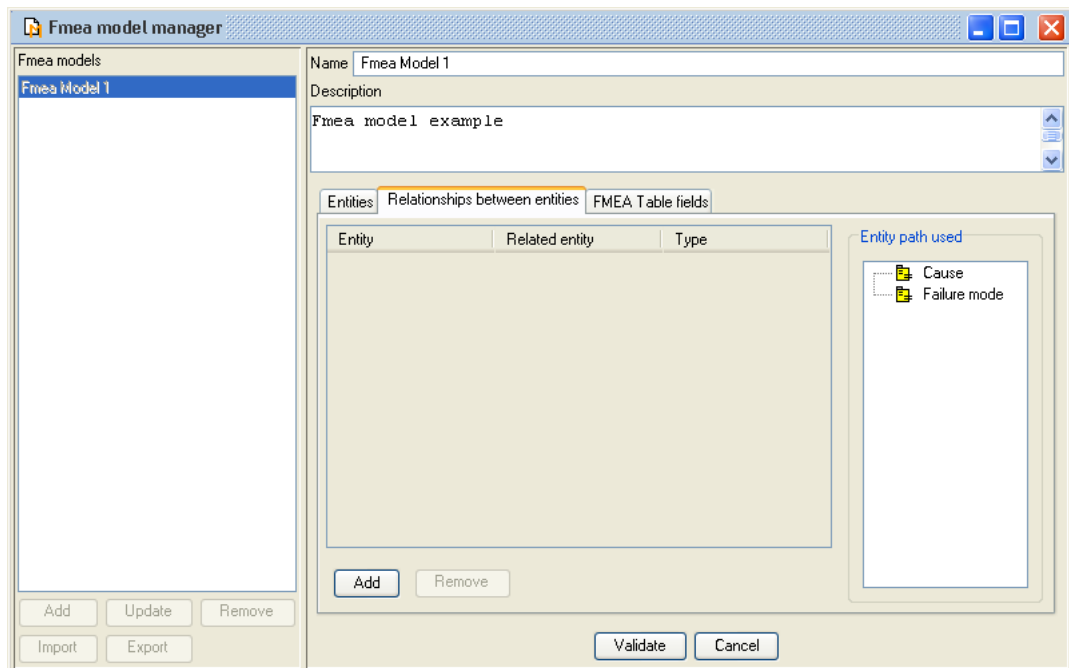
The 'Fmea model manager' window has a title bar with standard window controls. It features a left sidebar with a tree view showing 'Fmea models' and 'Fmea Model 1'. The main area has a 'Name' field with 'Fmea Model 1' and a 'Description' text area with 'Fmea model 1 example'. Below these are three tabs: 'Entities', 'Relationships between entities', and 'FMEA Table fields'. The 'Entities' tab is active, showing a table with two columns: 'Name' and 'Type'. The table has two rows: 'Cause' and 'Failure mode'. To the right of the table are 'Add', 'Edit', and 'Remove' buttons. At the bottom of the window are 'Validate' and 'Cancel' buttons.

Name	Type
Cause	
Failure mode	



### 8.2.2.2. Creating relationships between entity models

The creation of the relationships between entity models is done throughout the corresponding **Relationships between entities** as shown on the following window:



Before creating a relationship, the user has to create at least two entities. A relationship defines the association existing between two entity models in the entity-association underlying model. Two association cardinalities are allowed:

- 1-1: an entity of the first model is associated to only one entity of the second model,
- 1-N: an entity of the first model can be associated to several entities of the second model.

To create a new relationship, the user presses the **Add** button located above the relationships list. The following window appears:



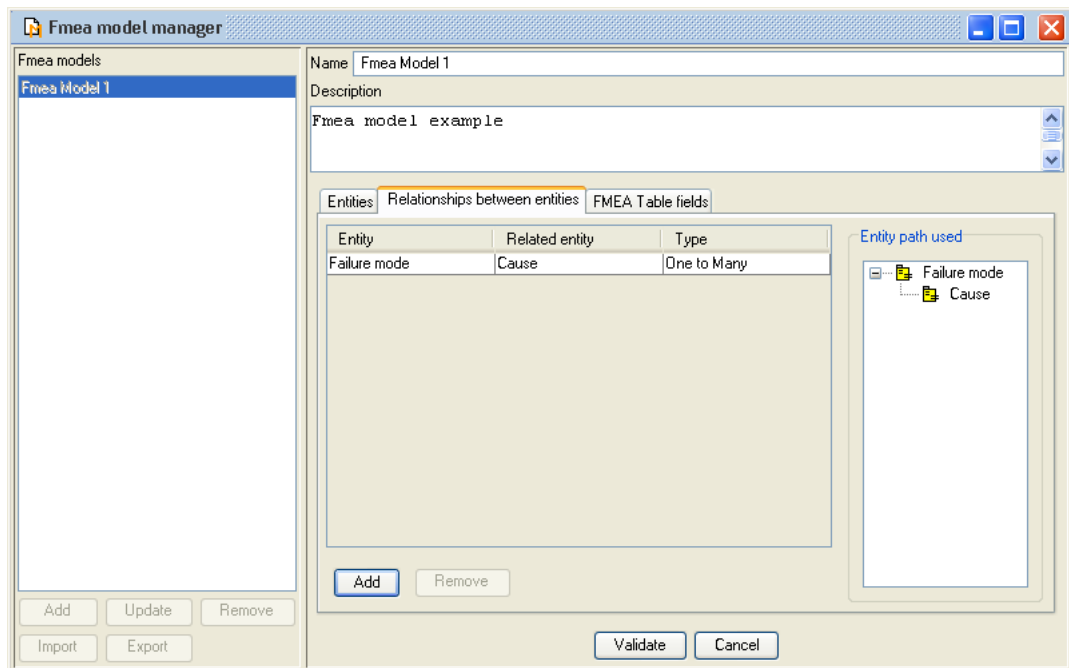
Then, the user selects the both extremities of the relationship and its type (One to One or One to Many) in function of the nature of the association.

To validate the modifications, he presses the **Validate** button, to cancel he presses the **Cancel** button.

Using the previous example, if the user creates the following relationship:

- 1-N entre 'Failure mode' and 'Cause',

then the FMEA model manager window appears as shown on the following figure:



The reader will notice the new information inside the **Entity path used** area on the right side of the tab. This information is described in the following section.

In a worry of clearness, in all the rest of the document, we will adopt the following convention: the model located in the 'Entity' column of the table will be named the source of the extremity and this one located in the 'Related entity' the destination of the extremity. This will allow avoiding all the confusions when using these notions.

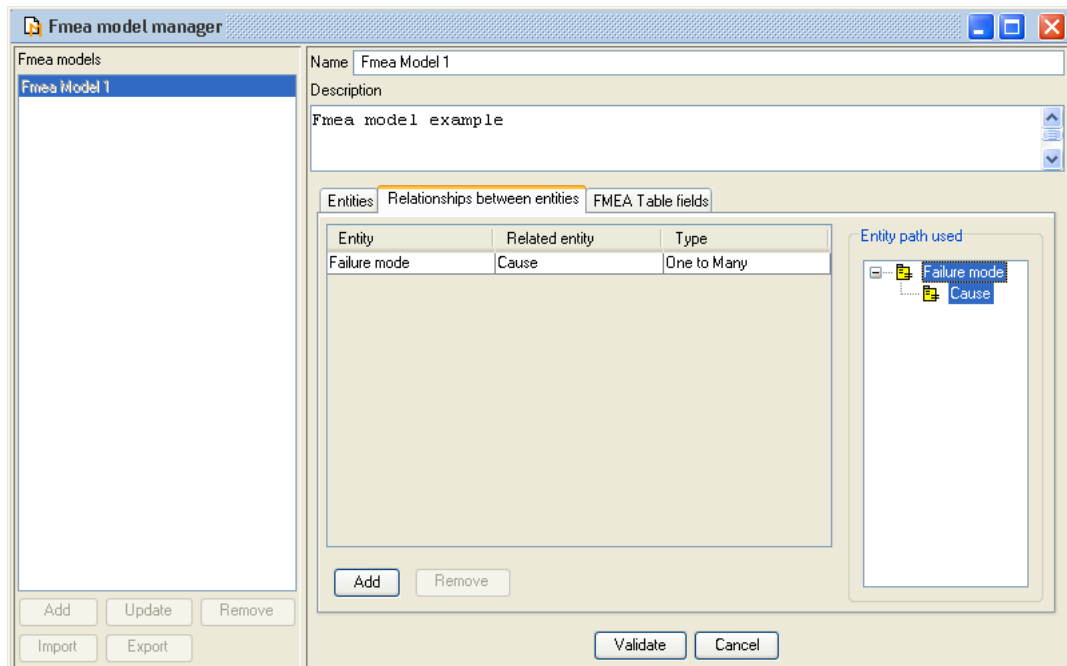
### 8.2.2.3. Defining the entity path used in the FMEA model based tables

The definition of the entity path used in the FMEA model based tables is done throughout the **Entity path used** area of the **Relationships between entities** tab.

The user should remind that one line of the FMEA table contains necessarily the data of one set of entity linked between them by relationships. A path is then built by all these entities, deduced directly of the corresponding relationship path.

The **Entity path used** area contains all the paths deduced of the existing relationships between entities. In our example, only one path can be exploited: Failure mode -> Cause. The user cannot select another path than this one.

By selecting the 'Cause' entity model in the **Entity path used** area, the user defines the used path as ('Failure mode' -> 'cause'). The following figure shows the result, that all of the chosen path nodes are selected.



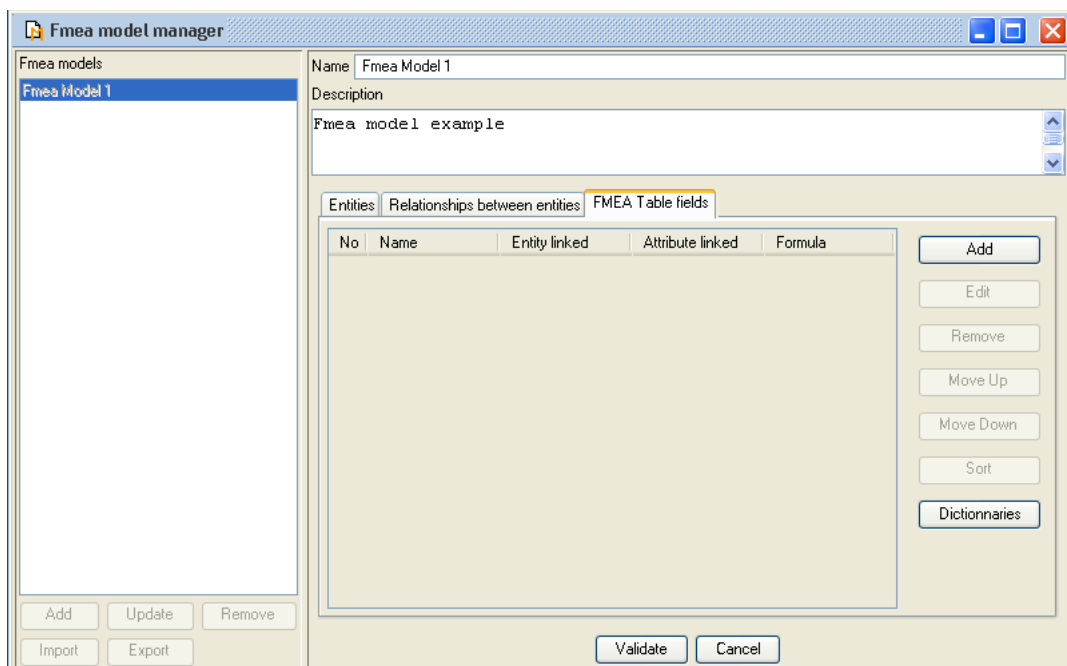
## 8.2.2.4. Defining the FMEA table model associated to the FMEA model

Remind that the FMEA tables are composed of columns, these columns are placed in a orderly way. Each of the table columns is linked to an entity model attribute created previously or used to display the result of an arithmetic formula taking as entries other column cells values.

When the column is linked to an entity model attribute, its type correspond to the attribute one and its value directly reflect the attribute one.

In the case of the use of a formula, the only one columns that can participate as input variables are necessarily of numerical type. The number of input variables is not open. The use of formula is described below in the document.

The definition of the structure of the FMEA tables associated to the model is the last step of an FMEA model creation. This step is done throughout the **FMEA Table fields** tab of the FMEA model manager window as shown on the following figure:





This tab is composed of a table containing the set of the FMEA table columns associated to the model with their properties. The table is composed of the following columns:

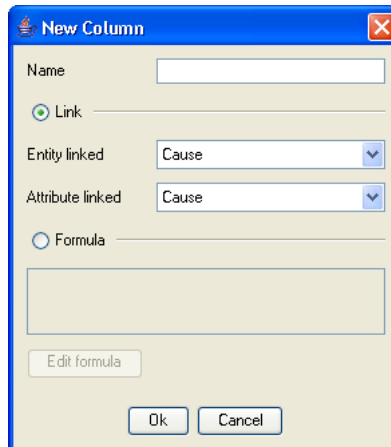
- **Position:** position of the column in the table (from 0 and left to right),
- **Name:** name of the column,
- **Entity linked:** entity the column is link to which,
- **Attribute linked:** attribute the column is linked to which,
- **Formula:** formula applied to obtain the result displayed in the column cells.

The following actions are provided to the user:

- **Add** a new column the FMEA table model definition
- **Edit** the selected FMEA table model column definition
- **Remove** the selected FMEA table model column definition
- **Move Up** the column to a position up in the FMEA table model
- **Move Down** the column to a position down in the FMEA table model
- **Sort** the FMEA tables based on this model and merge cells
- **Dictionaries:** to manage dictionaries

 In the column description table, the position of these columns is sorted in ascendant order.

To create and add a new column definition to the FMEA table model definition, one entity model at least must exist. The user presses the  in order to show the following dialog window:



The dialog window titled "New Column" has a blue header bar with a close button. It contains two main sections: "Link" and "Formula". The "Link" section is selected with a radio button. It includes a "Name" text field, an "Entity linked" dropdown menu (set to "Cause"), and an "Attribute linked" dropdown menu (set to "Cause"). The "Formula" section is unselected and contains a large text area and an "Edit formula" button. At the bottom are "Ok" and "Cancel" buttons.

The user fills the **Name** field in with the new column name and then select the chosen column type with the **Link** and **Formula** options.

If the user chooses the **Link** option, the column is linked to an entity model attribute. The user then chooses the entity model and the attribute by using the provided fields.

If the user chooses the **Formula** option, he then has to press the **Edit formula** button in order to create a formula. We will see the formula edition later.

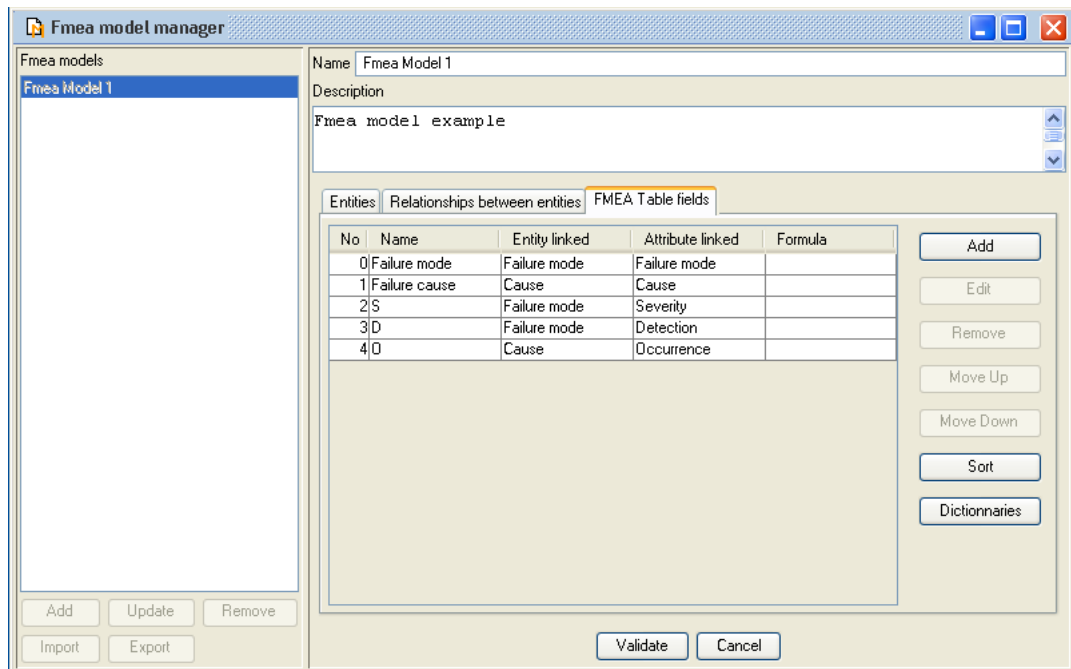
When all fields of the **New Column** dialog window are filled, the user confirms its modifications by pressing on the **Validate** button. If he prefers to cancel all the modifications, he then presses on the **Cancel** button.

Using the previous example, let us admit that the user has created the five following columns:

Name	Entity linked	Attribute linked
Failure mode	Failure mode	Failure mode
Failure cause	Cause	Cause
S	Failure mode	Severity

D	Failure mode	Detection
O	Cause	Occurrence

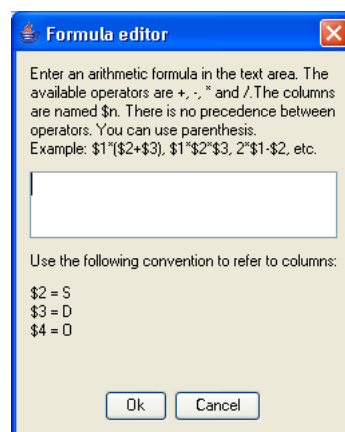
The FMEA model manager window then appears like this:



The user can notice that the **Formula** column stay empty as the column principal type is not the result of an arithmetic computation defined by a formula.

### 8.2.2.5. Create a formula associated to a column

To create a formula associated to an FMEA table model column, the user opens the edition dialog window by pressing onto the **Add** button or onto the **Edit** button, selects the **Formula** option, finally presses onto the **Edit formula** button. The following dialog window then appears:



The edition area is located in the center of the window with a white background. The lower part of the window shows the columns that can be used as input variables in the formula.

Concerning the formula syntax, the available operators are the following ones: +, -, \* and /. There is no precedence between operators so you can use parenthesis to define the order of the calculation. The input variables must correspond to columns of numerical type (Integer or Double). Literal numerical values can be used.

An input value is written as follows:

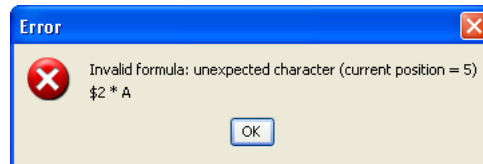
\$n

Where n is the column position in the edited table model.

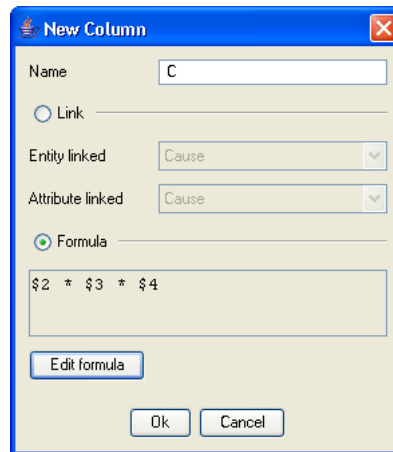
To confirm its formula, the user presses onto the **Validate** button. To cancel them, he presses on the **Cancel** button.

When the user presses onto the **Validate** button, a syntax check is done on the formula. An incorrect formula cannot be saved. When a formula is incorrect, a message is shown.

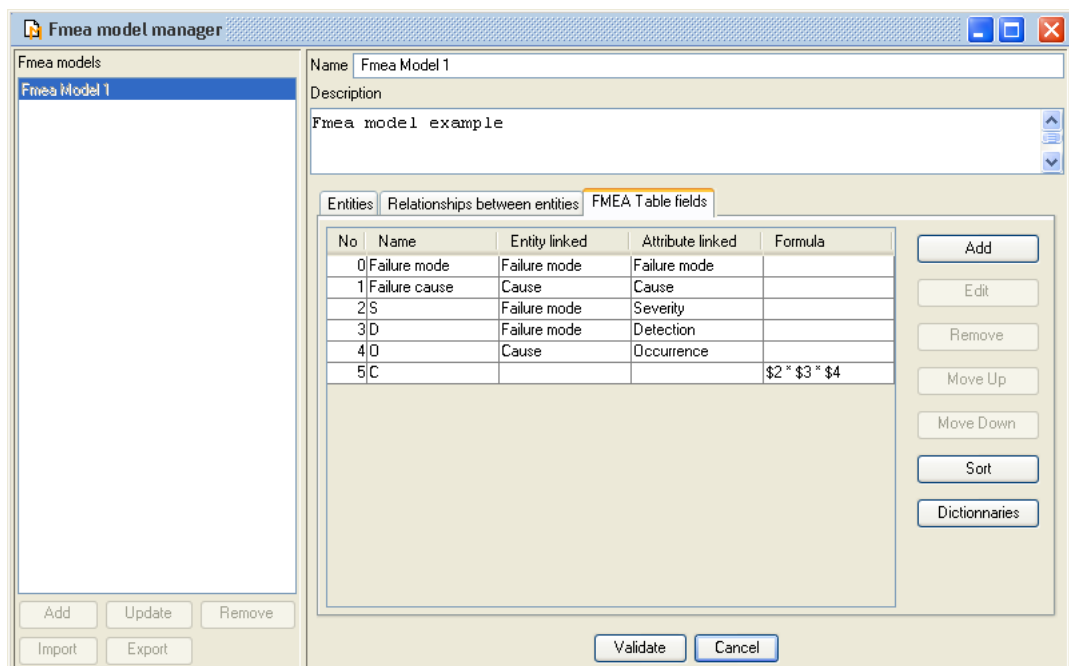
For instance, if the user enters the following formula: \$2 \* A and presses onto the **Validate** button, the following message is then shown:



If the user enters the following formula: \$2 \* \$3 \* \$4 and presses onto the **Validate** button, the formula is then accepted and the edition window appears as follows:



The user confirms the modification by pressing the **Validate** button. The FMEA model manager window appears as follows:

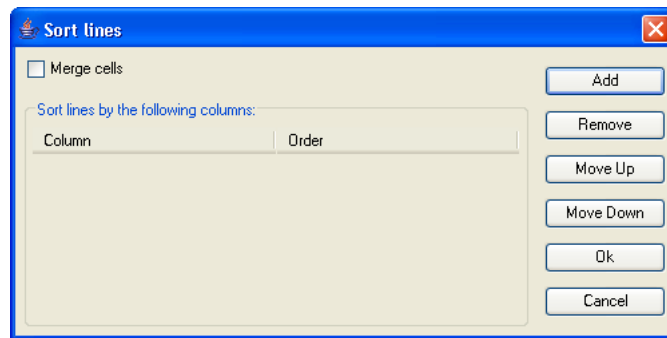


With our example, we show how to create a column containing the criticality (C) of a failure cause resulting of the product of the failure mode severity (S), the failure cause occurrence(O) and the failure mode detection (D).

## 8.2.2.6. Defining a sort set and merging cells

The user can define a sort at the FMEA table model level. Furthermore, the user can achieve an operation called cells merge which will be explained below. The sort and the cells merge operations are defined at the FMEA table model level so apply on all the FMEA tables which are based on.

These operations are accessible by using the **Sort** button. When the user presses onto this button, the following dialog window appears:



This window is based on a table containing the columns used to do the sort of the FMEA table lines. The functionality of cells merge exist as an option of the sort processing with the following label **Merge cells**.

The main actions available on this window are the following ones:

- **Add** a new sort column,
- **Remove** the selected sort column,
- **Move Up** a sort column,
- **Move Down** a sort column,
- **Validate** all the actions done in the dialog window,
- **Cancel** all the actions done in the dialog window.

The sort is done on the first column of the table, then on the second column and so on. The **Move Up** and **Move Down** commands allow the user to change the order of the multi-column sort using order. The sort on a column is done in ascending or in descending order.

Depending on the **Merge cells** checkbox button, the number of columns of the sort definition table change: If the option is not activated, the table columns are the following ones:

- **Column**: name of the column used in the sort,
- **Order**: sort order for the column (ascending or descending).

If the option is activated, the table columns are the following ones:

- **Column**: name of the column used in the sort,
- **Order**: sort order for the column (ascending or descending),
- **Formula**: indique l'emploi d'une formule pour obtenir les valeurs de cette colonne,
- **Operation**: formula applied on the source column merge cells,
- **Source**: name of the column containing the cells implied as effective parameters of the formula.

The **Operation** and **Source** columns cells are editable by double- clicking on the cells and at the condition that the corresponding **Formula** cell is selected.

The cells merge during the sort definition is an action that allows to merge cells which are placed at contiguous cells by the sort operation.

Once the cells grouped by the sort operation, it is possible to use a formula in order to obtain the values of a column from the values of the grouped source cells. The conditions requested to make things work are the following ones:

- the source column type must be numerical,
- the column containing the result of the formula application must be of numerical type,
- the cells merge must be realized by an adequate choice of columns to be used for the sort operation.

Let us take an example. Let us assume that the user has changed the FMEA table model as shown on the following figure:

No	Name	Entity linked	Attribute linked	Formula
0	Failure mode	Failure mode	Failure mode	
1	FM - Rate	Failure mode	Rate	
2	Failure cause	Cause	Cause	
3	Fail cause - Rate	Cause	Rate	
4	Effect	Failure mode	Effect	
5	S	Failure mode	Severity	
6	D	Failure mode	Detection	
7	O	Cause	Occurrence	
8	C			\$5 * \$6 * \$7

Let us now assume that the user has created an FMEA table based on this model and that he has entered data in this table. The following figure shows the FMEA table edition window with the data entered by the user:

Failure mode	FM - Rate	Failure cause	Fail cause - Rate	Effect	S	D	O	C
FM_1		new Cause 3	1.000E-01	Effect 0				0.000E00
FM_1		new Cause 1	2.000E-01	Effect 0				0.000E00
FM_2		new Cause 4	5.000E-01	Effect 1				0.000E00
FM_2		new Cause 2	4.000E-01	Effect 1				0.000E00
FM_0		new Cause 0	3.000E-01	Effect 2				0.000E00

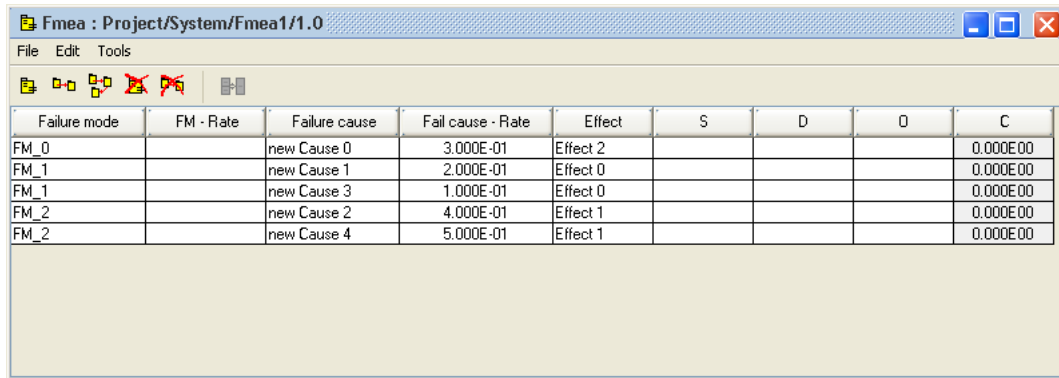
In the table, lines are not sorted. They are organized in an undefined order.

Let us assume that the user defines the sort shown on the following figure:

Column	Order
Failure mode	Ascending



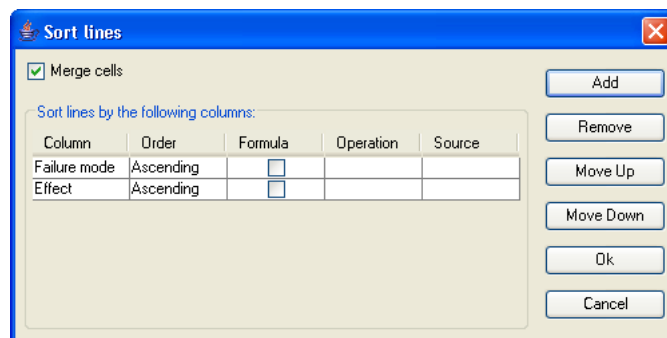
The following figure shows the change on the FMEA table edition window:



Failure mode	FM - Rate	Failure cause	Fail cause - Rate	Effect	S	D	O	C
FM_0		new Cause 0	3.000E-01	Effect 2				0.000E00
FM_1		new Cause 1	2.000E-01	Effect 0				0.000E00
FM_1		new Cause 3	1.000E-01	Effect 0				0.000E00
FM_2		new Cause 2	4.000E-01	Effect 1				0.000E00
FM_2		new Cause 4	5.000E-01	Effect 1				0.000E00

We can here notice that the lines are well sorted on the 'Failure mode' column in ascending order. We can also notice the grouping of cells with the same value in this column.

We assume now that the user would like to merge cells that have been grouped by the sort operation by the following way:



**Sort lines**

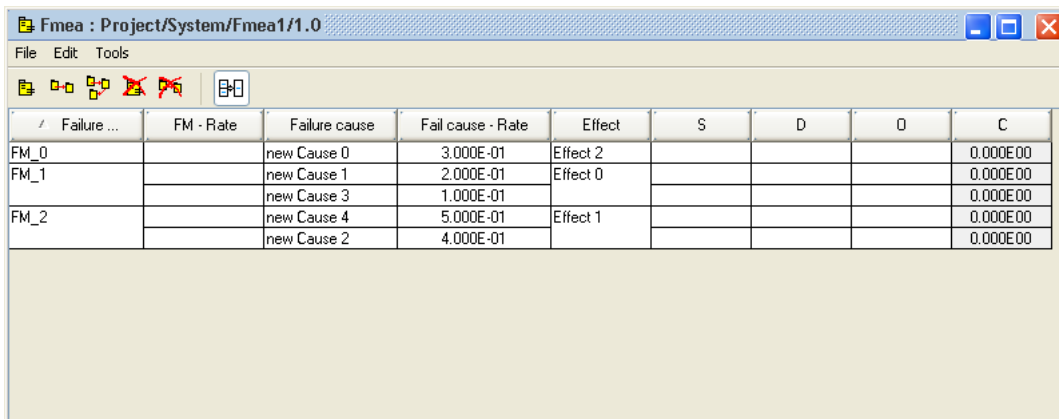
☒ Merge cells

Sort lines by the following columns:

Column	Order	Formula	Operation	Source
Failure mode	Ascending	<input type="checkbox"/>		
Effect	Ascending	<input type="checkbox"/>		

Buttons: Add, Remove, Move Up, Move Down, Ok, Cancel

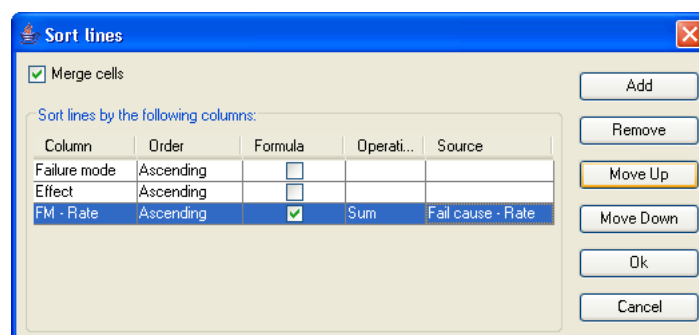
The result appears on the following figure:



Failure ...	FM - Rate	Failure cause	Fail cause - Rate	Effect	S	D	O	C
FM_0		new Cause 0	3.000E-01	Effect 2				0.000E00
FM_1		new Cause 1	2.000E-01	Effect 0				0.000E00
		new Cause 3	1.000E-01					0.000E00
FM_2		new Cause 4	5.000E-01	Effect 1				0.000E00
		new Cause 2	4.000E-01					0.000E00

The both concerned columns are 'Failure mode' and 'Effect'.

Let us assume that the user would like to fill the 'FM - Rate' column in with the result of the sum of 'Failure cause - rate' column cell values. The definition window includes the following information:



**Sort lines**

☒ Merge cells

Sort lines by the following columns:

Column	Order	Formula	Operati...	Source
Failure mode	Ascending	<input type="checkbox"/>		
Effect	Ascending	<input type="checkbox"/>		
FM - Rate	Ascending	<input checked="" type="checkbox"/>	Sum	Fail cause - Rate

Buttons: Add, Remove, Move Up, Move Down, Ok, Cancel

The 'FM - Rate' column was added with a formula specification.

The result of the formula application is shown on the following figure:

Failure ...	FM - Rate	Failure cause	Fail cause - Rate	Effect	S	D	O	C
FM_0	3.000E-01	new Cause 0	3.000E-01	Effect 2				0.000E00
FM_1	3.000E-01	new Cause 3	1.000E-01	Effect 0				0.000E00
		new Cause 1	2.000E-01					0.000E00
FM_2	9.000E-01	new Cause 2	4.000E-01	Effect 1				0.000E00
		new Cause 4	5.000E-01					0.000E00

## 8.2.2.7. Dictionary manager

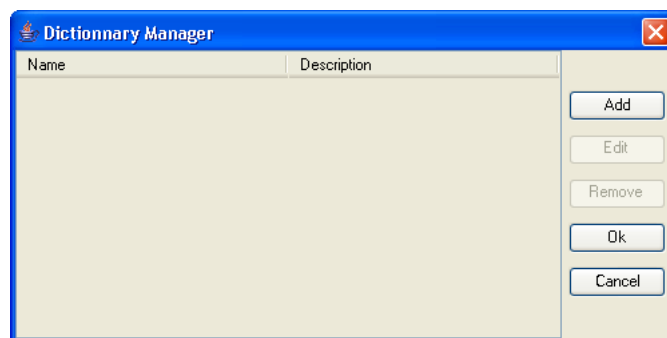
A data dictionary gather a set of threshold - value couple. A value interval is associated to a name, the value affected to the threshold corresponding to the lower bound. These notions allow the user to create for instance a dictionary of criticities, of severities as well as of phases.

At the condition of defining at least one dictionary, the user can then associate a dictionary to a column. The cells of the corresponding column will then display the name of the interval containing their value.

The dictionary manager is used to manage the dictionary set of an FMEA model.

For all concerning the dictionaries use, the user is invited to read the 'Using data dictionaries' sub-chapter of the 'FMEA tables' chapter.

The dictionary manager window can be accessed using the **Dictionaries** command. By pressing the corresponding button, the following dialog window appears:



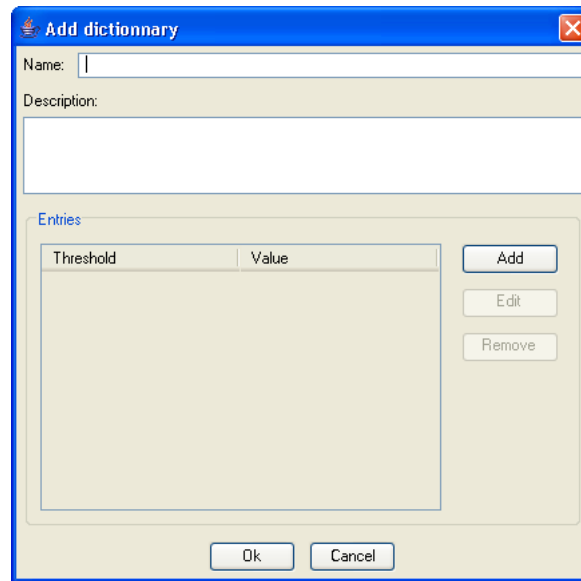
This window contains the dictionary list on the left part of the pannel as well as a set of command buttons on the right part of the pannel. The existing dictionary list contains two columns: a column containing the dictionary names and a column containing the dictionary descriptions.

The available commands are the following ones:

- **Add** a new dictionary to the liste of existing ones,
- **Edit** a dictionary which has been previously selected,
- **Remove** dictionaries which have been previously selected,
- **Validate** all the modifications and close the dialog window,
- **Cancel** all the modifications and close the dialog window.

All the commands are described in the rest of the chapter.

The **Add** command allows the user to create a new dictionary and to add it the list of existing ones. When the user presses onto the **Add** button, the following dialog window appears:



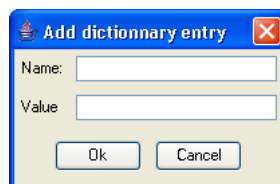
This window allows the edition of a dictionary description and content either during its creation or its modification (by using the **Edit** command).

This window contains several fields as the **Name** and **Description**.

In addition, the window comprises an area named **Entries** containing the list of threshold - value couples of the edited dictionary. This area allows the user to do the following actions:

- **Add** an entry to the list of existing ones,
- **Edit** the entry selected in the list,
- **Remove** the entries selected in the list.

To add an entry in the current dictionary, the user presses onto the **Add** button. The following window is bringing forward to the user:



The user enters the entry name and associated numerical value and validates his modification by pressing onto the **Validate** button or cancels all his modification by pressing onto the **Cancel** button.

Assuming that the user has entered the following data:

Threshold	Value
Minor	0
Major	3
Extreme	6
Disastrous	9

and filled the **Name** and **Description** fields in respectively with the 'Criticity' and 'This dictionary contains the criticity thresholds' values, the dictionary edit window then will appear as shown on the following figure:

Threshold	Value
Disastrous	9
Extreme	6
Major	3
Minor	0

As we will see in the chapter entitled 'Using data dictionaries' while editing an FMEA table, assuming that the user has linked a column to the present dictionary, all the cells of this column which value is included in the  $[0, 3[$  range will display the text 'Minor', all those included in the  $[3, 6[$  range will display the text 'Major' and so on until the highest threshold. For this last, the corresponding range in our example is  $[9, \text{Infinite}[$ . For all the values included in this range, the cells will display the text 'Disastrous'.

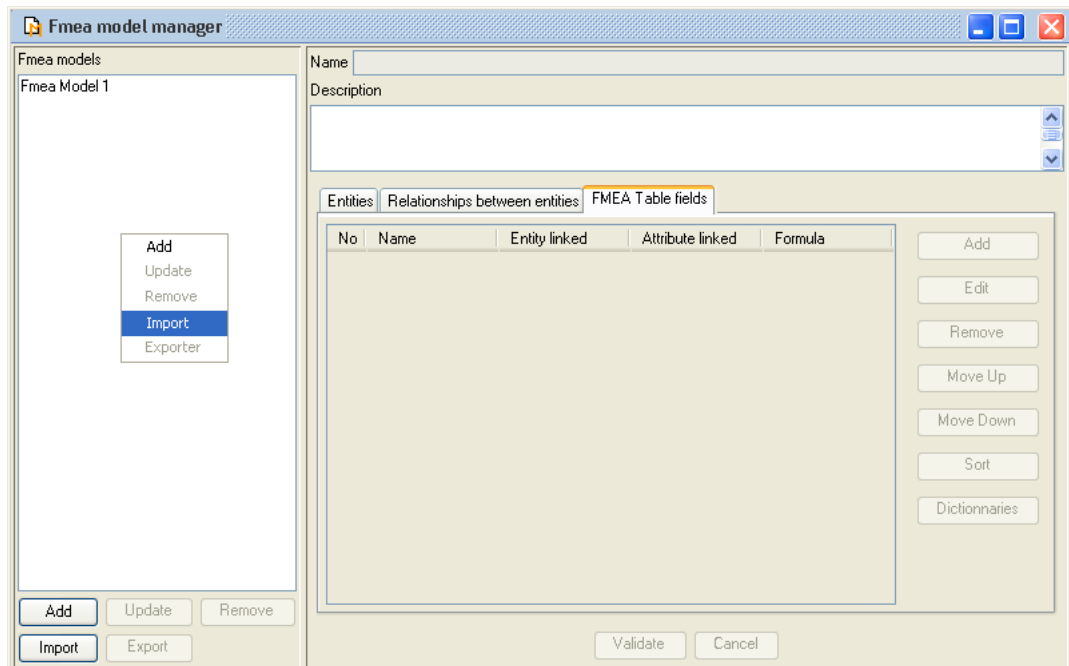
By pressing onto the **Validate** button of the dictionary edit window, the dialog window will close and the dictionary manager window will then appears as on the following figure:

Name	Description
Criticities	Criticity threshold dictionary

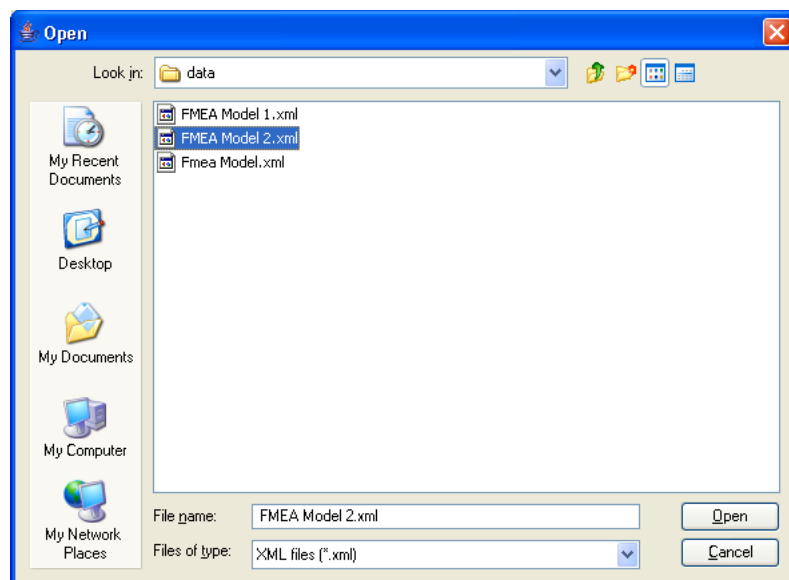
If the user would like to modify an existing dictionary, he will use the **Edit** button.

### 8.2.3. XML import of an FMEA model

The XML import of an FMEA model is realized by using the **Import** command accessible from the popup menu of the existing FMEA model list or the corresponding button under the list as shown on the following figure:

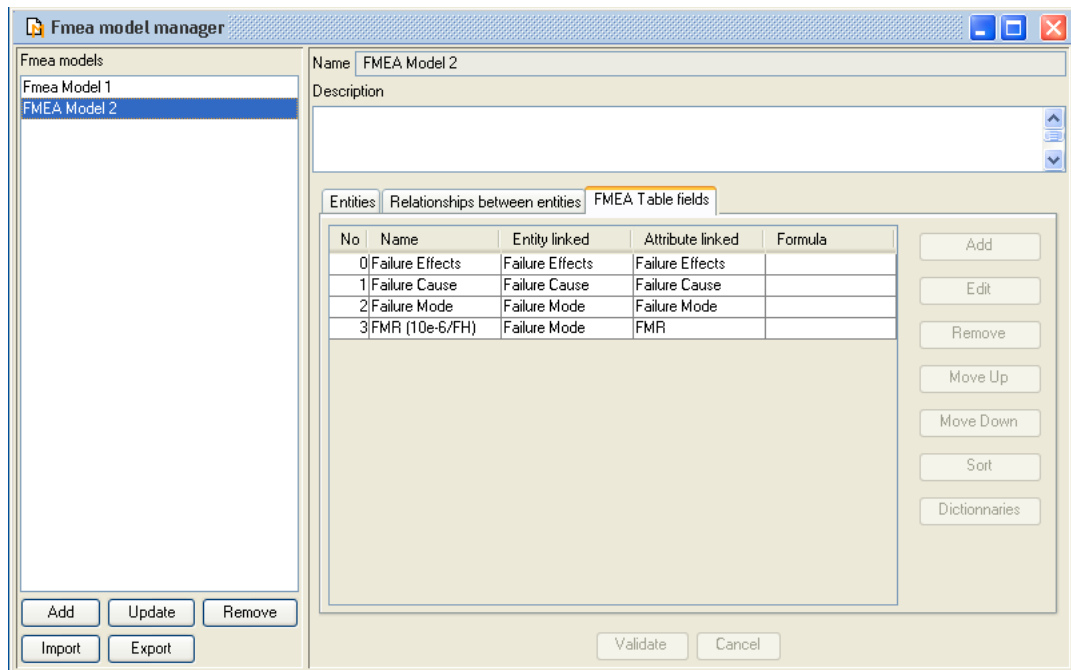


When activating this command, the following dialog window appears:



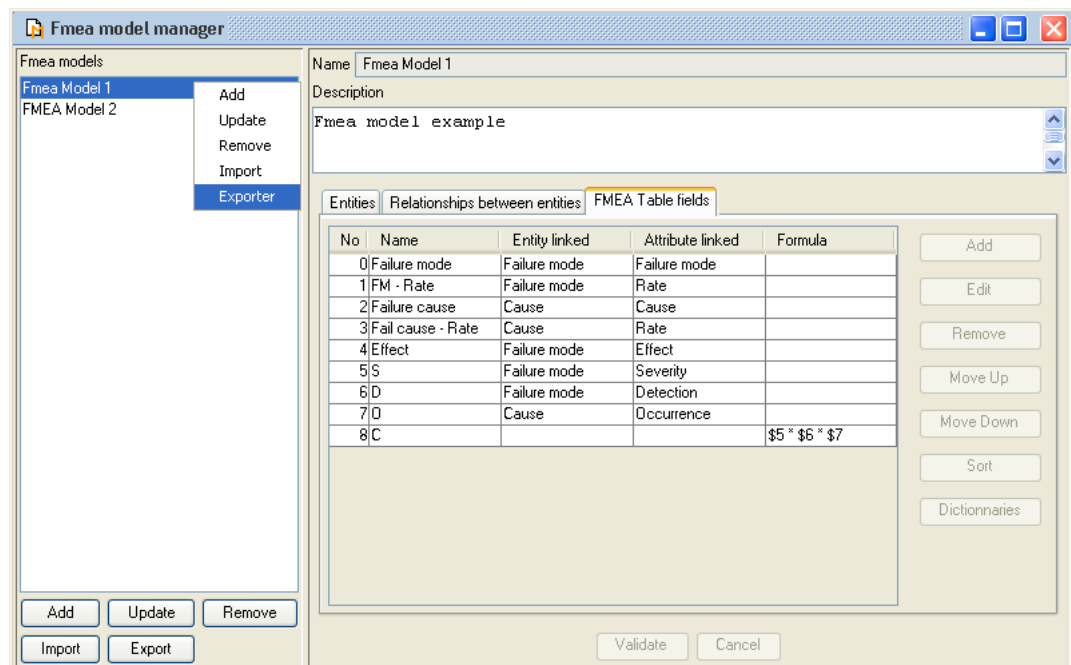
This window allows the user to select the XML file containing the FMEA model to import in to the database. If the user presses the **Open** button, the dialog window closes and the file data is imported into the database.

The new model is appended to the FMEA model list. In our case, the FMEA model manager window then appears like this:

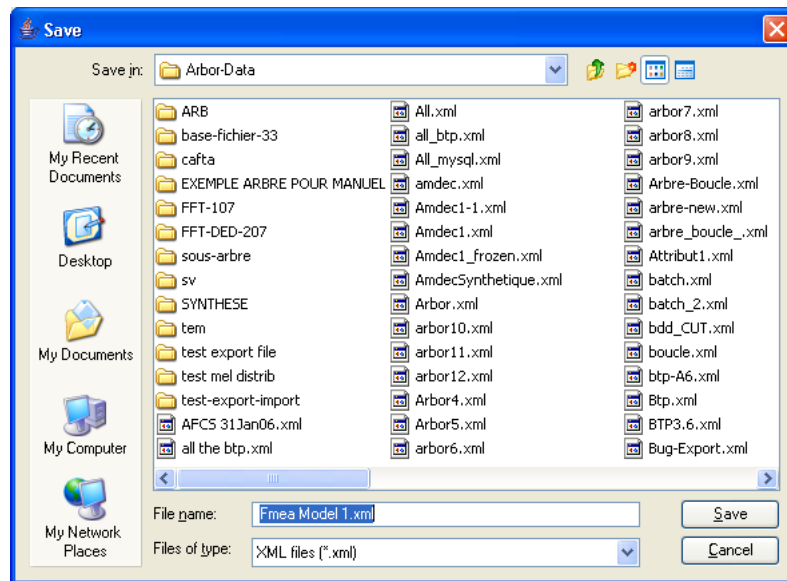


## 8.2.4. XML export of an FMEA model

The XML export of FMEA model is done using the **Export** command available from the popup menu of the existing FMEA model list or from the corresponding button under the list as shown on the following figure:



The user selects the FMEA model to export among the model list and then activate the **Export** command. The following dialog window appears:



This window allows the user to select or enter the name of the XML file into which he would like to export the FMEA model. By default, the **File name** field is filled in with the name of the model to export. If the user presses the **Save** button, the dialog window closes and the model is exported into the chosen file.

## 8.3. The FMEA tables






### 8.3.1. Fmea tables management

The FMEA table's management is done under the application document manager the same way as for the fault trees. An FMEA table is associated to one and only one system. However, a system can contain several FMEA tables. As for the fault trees, the FMEA tables are managed in versions and subject to access rights.

The available actions on the FMEA tables are introduced in the following table:

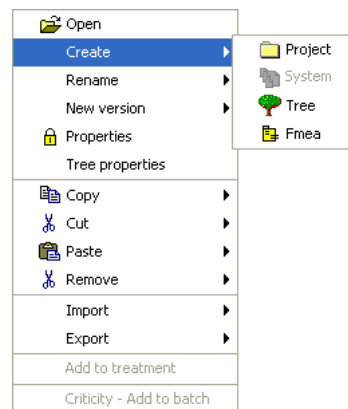
Create - Fmea	When selecting a system node	Creation of a new FMEA table and addition of this table under the selected system
Paste - Fmea	When selecting a system node	Copy or move the previously selected FMEA tables under the presently selected system
Open	When selecting an FMEA node or FMEA version node	Edition of the selected FMEA tables
Rename - Fmea	When selecting an FMEA node	Rename of an FMEA table
New version - Fmea	When selecting an FMEA node	Creation of a new version of FMEA table and addition of this version under the selected FMEA table
Properties	When selecting an FMEA node	Display of the selected FMEA table
Copy - Fmea	When selecting some FMEA nodes	Copy of the selected FMEA tables
Cut - Fmea	When selecting some FMEA nodes	Mark the selected FMEA tables as to be moved
Remove - Fmea	When selecting some FMEA nodes or FMEA version nodes	To remove the selected FMEA tables
Export - XML file (*.xml)	When selecting some FMEA nodes or FMEA version nodes	Export of the selected FMEA tables into an XML formatted file
Import - XML file (*.xml)	Do not need nodes selection	Import data from XML formatted files including FMEAs and FMEA models

In addition to these actions, the application main menu bar and toolbar contain some actions available on FMEA tables which are the following ones:

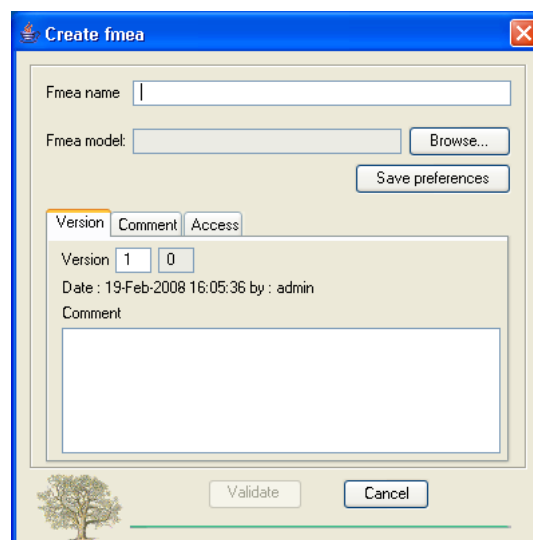
Menu	Icon	Functionnality	Use
File		New - Fmea	Start the creation of a new FMEA table into the document manager
		Open	Edit the FMEA table selected in the document manager
		Rename - Fmea	Rename the FMEA table selected in the document manager
		Save	Save the currently edited FMEA table wich is active
		Save all	Save all the currently edited documents (fault trees and FMEA tables)
		Close	Close the currently edited FMEA table which is active
		Print	Print the currently edited FMEA table which is active
		File report	Produce a report on the currently edited FMEA table which is active

## 8.3.2. Creating an FMEA table

The FMEA table creation is done from the application document manager using the popup menu as shown on the following figure.



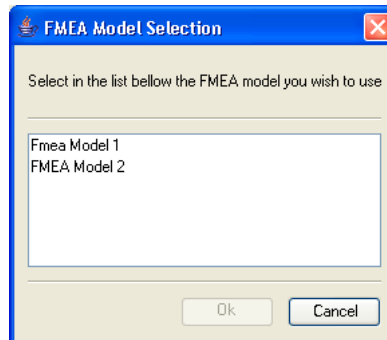
When the user activates the **Create - Fmea** command, the following dialog window is bring forward:





The user must fill the **Name** and **FMEA model** fields. The **FMEA model** field correspond to the FMEA model to which the FMEA table is referred.

To fill the **FMEA model** field, the user accesses to the existing model list by pressing on the **Browse** button. The following dialog window then appears:



Here there is only one available FMEA model which is 'Fmea Model 1'.

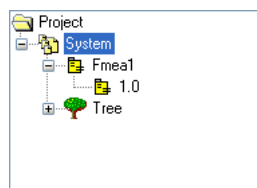
The user selects the FMEA model by double-clicking on the model chosen in the list or by selecting it and by pressing on the **Validate** button. When validating, the window closes and the selected model fills in the creation window **FMEA model** field.

The other fields are optional. The user can change the number of the version which is fixed to the 1.0 value by default as well as the comment inside the **Version** tab.

The **Comment** tab allows the user to edit the FMEA table attached comment which is common to all the FMEA table versions. Finally, the **Access** tab allows the user to edit the FMEA table access rights which are by default set to 'read only' for the group to which it is attached and 'No access' for the others.

When the user has ended to fill the chosen fields, he confirms the creation by pressing on the **Validate** button or cancels all the creation in progress by pressing on the **Cancel**.

If the user confirms the FMEA table creation, this one is created into the database and appears in the document manager under two different nodes: a node with the name of the FMEA table as label and a node with the FMEA table version as label, as shown on the following figure:

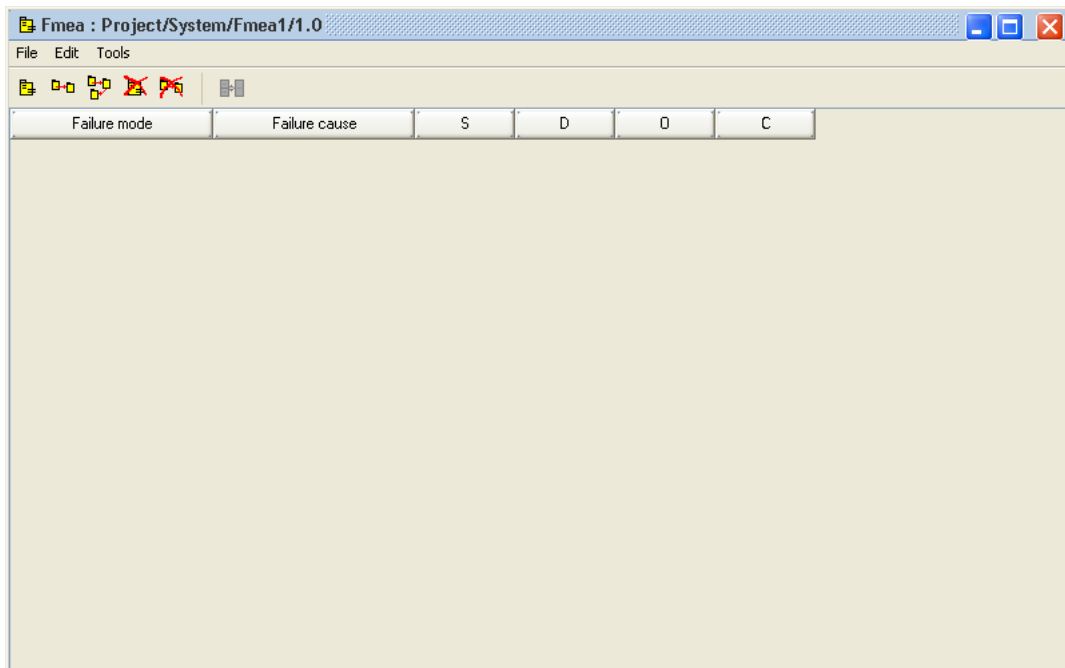


The figure above shows the creation in the document manager of the FMEA table named 'Fmea1' with its first version '1.0'.

The FMEA table is assigned to the user that has logged onto the application and in its default group. The access rights for the FMEA table owner are 'Read/Write'.

### 8.3.3. Editing an FMEA table

To edit an FMEA table, the user activates the **Open** command of the document manager popup menu. When the command is activated, the following editor window is opened:



This edit window is an internal frame inside the application workspace. All the FMEA table edition is done throughout this window.

The edit window comprises a menu bar and a toolbar.

The available commands on the menu bar and possibly on the toolbar are the following ones:

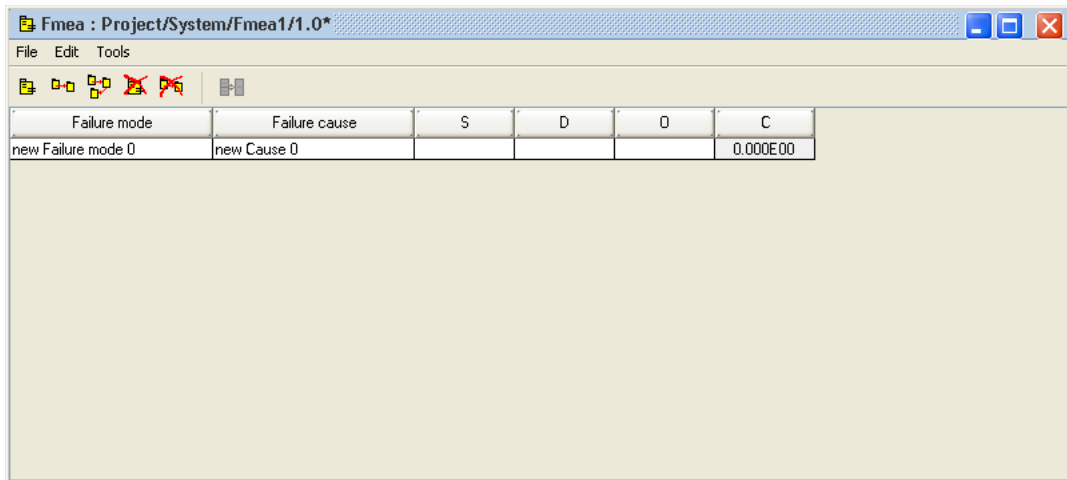
Menu	Icon	Functionnality	Use
File		Import - Text File...	Import a text file
Edit		New entity	Create a new entity and insert it into the FMEA table
		New relation	Create a relationship from the selected entity and insert the new objects into the FMEA table
		New relations	Create a complete line of data into the FMEA table
		Remove entity	Remove the entity selected in the FMEA table
		Remove relation	Remove the relationship which extremity is the entity selected in the FMEA table
		Merge cells	Merge vertically cells which are abut
Tools		Dictionnaires...	Edit the dictionnary manager window

All these commands are described in detail thereafter in the chapter.

In the FMEA table, the cells appearing in gray color correspond to cells which column is linked to an entity model for which no entities is linked to one of the source entity model on the used entity path in the table.

### 8.3.3.1. Adding a new line

To create a new line in the FMEA table, the user needs to create all the entities of the entity path (defined at the FMEA model level) and all the relationships between these entities. In order to do it, the user launches the **New entity** command. A new line appears in the FMEA table as shown on the following figure:



The screenshot shows a window titled 'Fmea : Project/System/Fmea1/1.0\*'. It contains a table with the following structure:

Failure mode	Failure cause	S	D	O	C
new Failure mode 0	new Cause 0				0.000E00

At the time of creation of a new line inside the FMEA table, the names identifying the new entities are automatically generated. These names have as suffix, the name of the entity definition (entity model). This suffix is followed by a number auto-incremented.

The user is invited to modify the automatically generated names by significant names in the studied domain. If the user tries to rename an entity by editing one corresponding cell, a check on the unicity of the names is done in order to validate the modification.

### 8.3.3.2. Removing an entity

To remove an entity and all the resulting lines in the FMEA table, the user proceeds as follows:

- he selects in the table a cell which column is linked to an attribute of the entity to remove,
- and launches the **Remove entity** command.

The entity is then removed from the underlying model which leads to the removal of all the associated FMEA table lines.

### 8.3.3.3. Adding a new relationship

To add a new relationship, the user proceeds as follows:

- he selects in the table a cell which column is linked to one attribute of the entity from which the relationship has to be created,
- then launches the **New relation** command.

The relationship is then created into the underlying model which leads to the appearance of new lines in the FMEA table. There is creation of only one relationship direct from the selected entity and not of all the entity in cascade from this entity.

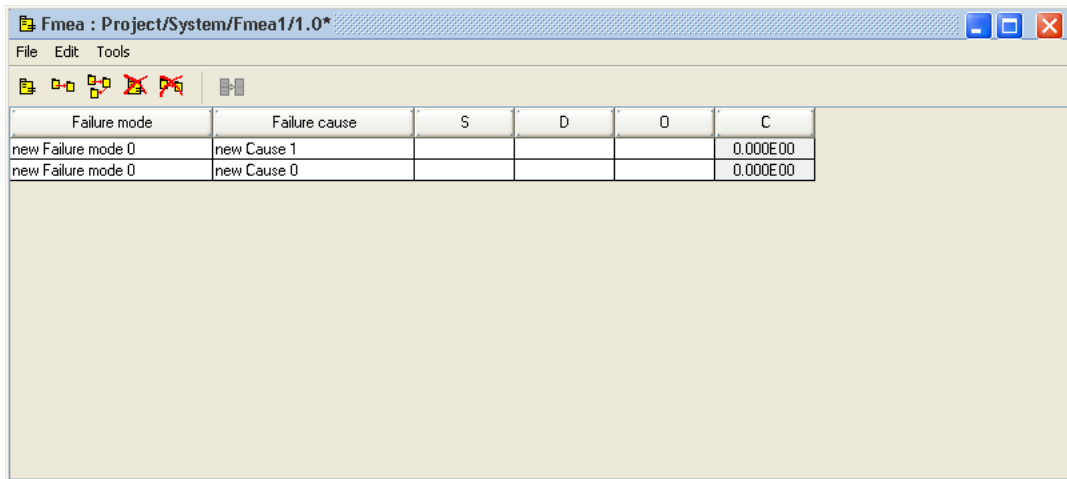
### 8.3.3.4. Decline a part of a line

The user would like to create a relationship between two given entities which results, in the FMEA table, in the creation of a new line of data declining the one that served as reference. On the new line, the data is identical for all the columns linked to entities upstream on the entity path defined in the edited FMEA model. In the other hand, from the selected entity, the data differs in the way that new entities (as well as relationships between them) are created.

To decline a part of a line, the user proceeds like this:

- he selects in the table a cell which column is linked to one of the attributes of the entity from which the line must decline,
- then uses the **New relations** command.

The figure that follows shows the declension of a part of line.



Failure mode	Failure cause	S	D	O	C
new Failure mode 0	new Cause 1				0.000E00
new Failure mode 0	new Cause 0				0.000E00

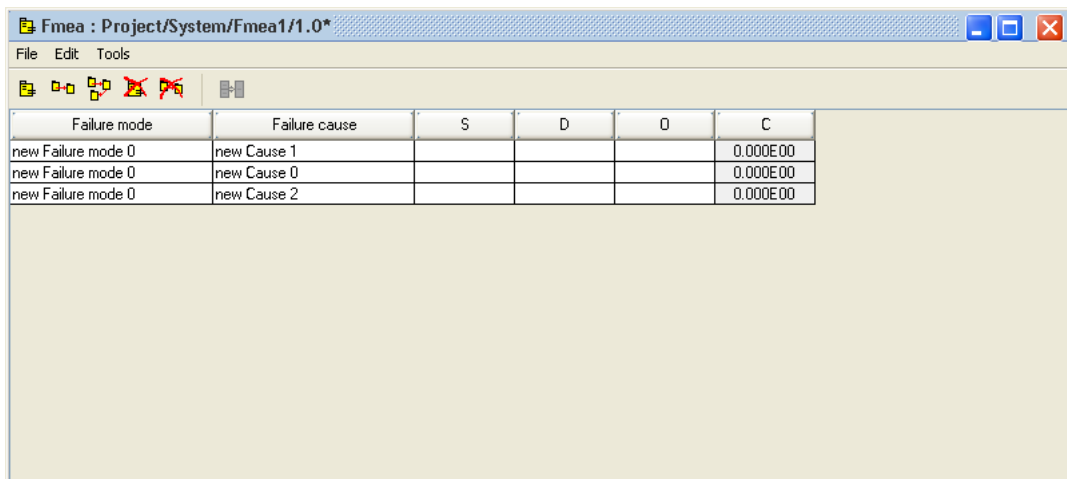
The example mentioned on this figure is obtained after selecting the cell containing the text "new Failure mode 0" then using the command. In the underlying model, a new entity of 'Cause' type and "new Cause 1" name is created as well as one relationship between the "new Failure mode 0" and "new Cause 1" entities. Thereafter, there is creation of the new line in the FMEA table located at the first position in the table of the example.

### 8.3.3.5. Removing a relationship

To remove a relationship, the user proceeds like this:

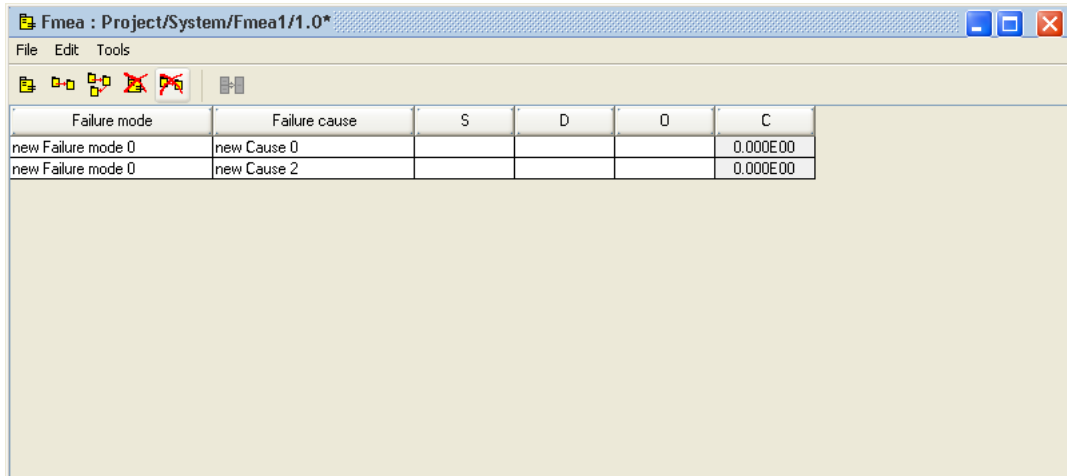
- he selects in the FMEA table the cell which column is linked to one of the attribute of the entity destination of the relationship to remove,
- then uses the **Remove relation** command.

The figure that follows shows the content of the FMEA table before removal of one relationship.



Failure mode	Failure cause	S	D	O	C
new Failure mode 0	new Cause 1				0.000E00
new Failure mode 0	new Cause 0				0.000E00
new Failure mode 0	new Cause 2				0.000E00

When the user selects the cell containing the text "new Cause 1" and uses the **Remove relation** command, then the FMEA table appears as shown on the following figure:



Failure mode	Failure cause	S	D	O	C
new Failure mode 0	new Cause 0				0.000E00
new Failure mode 0	new Cause 2				0.000E00

We can see on this figure that the removal of the line which manifested the existence of the entity of 'Cause' type and "new Cause 1" name.

### 8.3.3.6. Sorting lines and merge cells

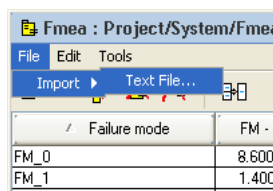
Line sort and cells merge has been well developed in the chapter titled 'FMEA models'. Go to this chapter to know the principles and use during the FMEA table edition.

The FMEA table editor own a special command named **merge cells**. This command is only enabled when a sort with the merge cells option was defined at the model creation step. It allows the user to activate/deactivate the cells merge option.

### 8.3.3.7. Importing textual data

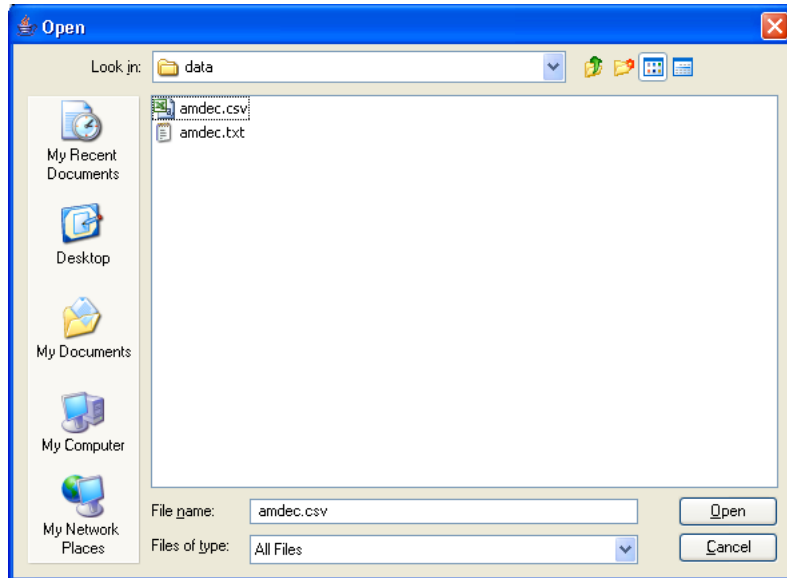
The text import allows the user to import textual data. The data feed the FMEA table and the underlying FMEA model. The import file format must at least use one character to trim the lines into fields in order to separate the fields from ones to ones. The file can for instance be of the cvs format (comma separated) or still of a simple text format with separator.

The textual import is done using the **Import - Text File...** command available on the FMEA table editor menu bar as shown on the following figure:



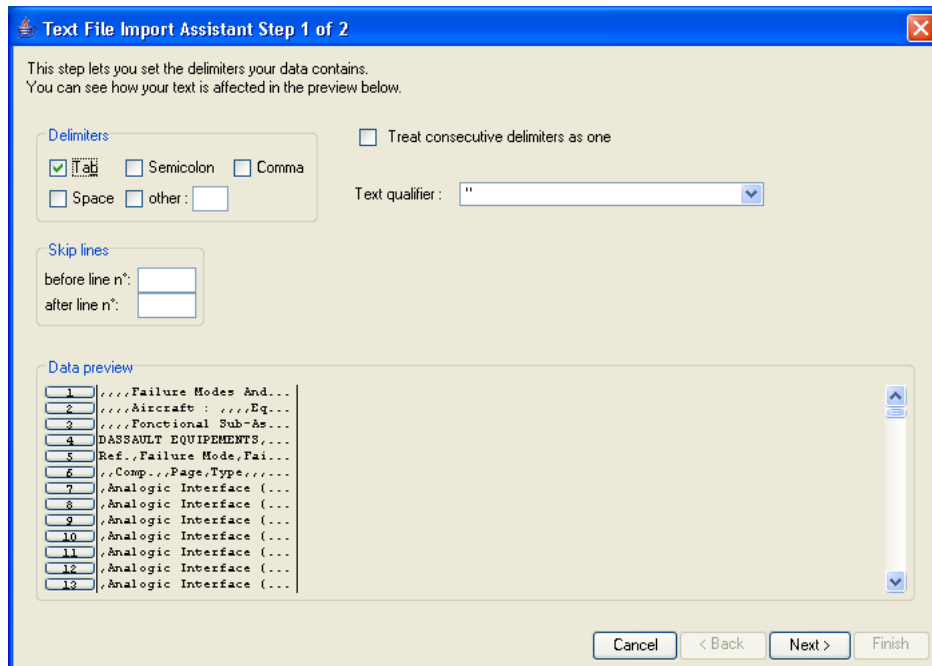
Failure mode	FM - f
FM_0	8.6000
FM_1	1.400

To import a text file, the user activates this command to make the following dialog window appear:



From this window, the user selects the file to import. An optional filter on the txt and csv file extensions is available.

When the file has been chosen, a second dialog window appears:



This second dialog window is the Text File Import Assistant. This assistant contains two steps which we will detail.

The first step allows the user to fill the information that will serve to interpret the file data.

The **Delimiters** area allows the user to select the field's delimiters. These delimiters are used to cut the file in several fields which at last will correspond to columns in the edited FMEA table. This area contains several checkboxes allowing to select the right delimiter. The **Other** checkbox allows the user to choose a different delimiter as the others.

The text qualifier allows to inform that there is a character in the file that is used to enclose the text. This character has priority on the field delimiters so all characters corresponding to a field delimiter located inside two text qualifiers, will not be interpreted as field delimiter but as simple character (belonging to the enclosed character string).

The **Skip lines** area allows the user to indicate the lines to consider.

Finally, the **Data preview** area give to the user a preview of the data such as they will be interpreted during the import with the chosen parameters.

In the example shown on the following figure, the delimiter to use is the comma. In addition, the sixth first line of the file correspond to the file header. Finally, the user wants to do an import test on the fifth first line. So, he will use the following setting:

**Text File Import Assistant Step 1 of 2**

This step lets you set the delimiters your data contains.  
You can see how your text is affected in the preview below.

**Delimiters**

☐ Tab ☐ Semicolon ☒ Comma  
☐ Space ☐ other :

☐ Treat consecutive delimiters as one

Text qualifier :

**Skip lines**

before line n°:   
after line n°:

**Data preview**

	Ref.	Failure Mode	Failure causes
2			
3			
4	DASSAULT EQUIPEMENTS		
5			
6			
7		Analogic Interface (1)	R226
8		Analogic Interface (1)	R226
9		Analogic Interface (1)	R227
10		Analogic Interface (1)	R227
11		Analogic Interface (1)	R228
12		Analogic Interface (1)	R228
13		Analogic Interface (1)	R229

Buttons: Cancel < Back Next > Finish

When the user has done its choices, he presses the **Next** button to go to the next step. The following window then appears:

**Text File Import Assistant Step 2 of 2**

This step lets you set the correspondences between the imported file columns and the edited FMEA columns

**Column correspondences**

File	Edited FMEA
Column 0	
Column 1	
Column 2	
Column 3	
Column 4	

Sauver preferences

**Data preview**

Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
	Analogic Interface (1)	R226		37	open circuit	0.136
	Analogic Interface (1)	R226		37	short circuit	
	Analogic Interface (1)	R227		37	open circuit	
	Analogic Interface (1)	R227		37	short circuit	
	Analogic Interface (1)	R228		37	open circuit	

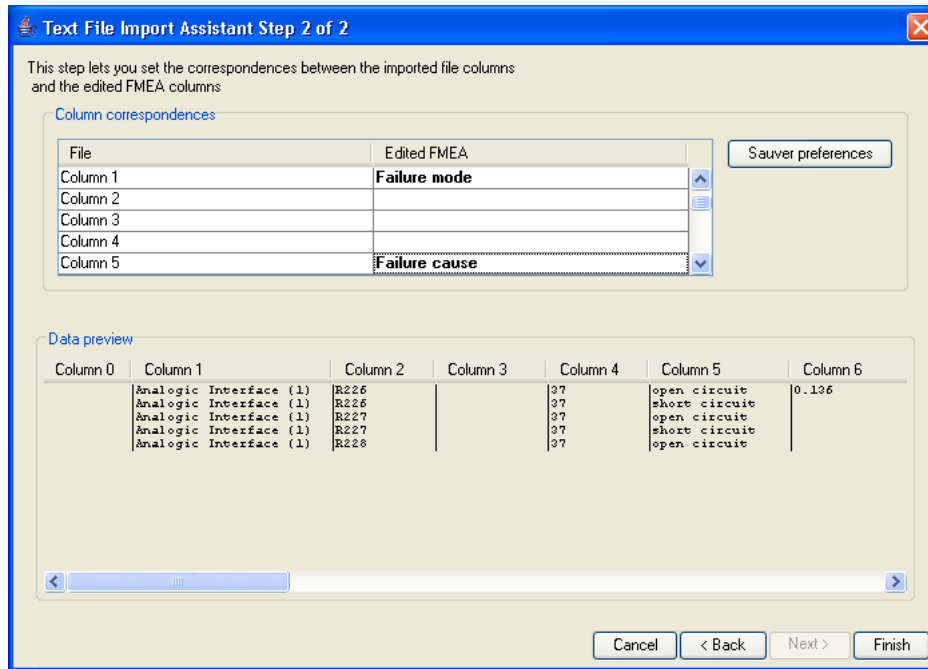
Buttons: Cancel < Back Next > Finish

The second step of the text file import consists in defining the correspondences between the fields extracted in the file and the FMEA table columns. Doing the correspondence is not an insignificant action because it must take account of the relationships between entities. It conduct to the obligation to match at least the columns identifying the entities with the file fields.

The window corresponding to the second step of the import contains two areas:

- **Column correspondences**, area allowing to match the columns of the FMEA table with fields in the file.
- **Data preview**, area showing the set of available fields in the file with the data that it contains.

The user proceeds as follows. He visualizes on the data preview the fields that he is interested with in the file (shown in columns on the data preview) and then defines the correspondences in the **Column correspondences** area. In our example, the fields corresponding to the columns 1 and 5 are to import because they contains respectively the identifiers for the 'Failure mode' model entities and for the 'Failure cause' model entities in our model. The user will set the correspondences shown on the following figure:



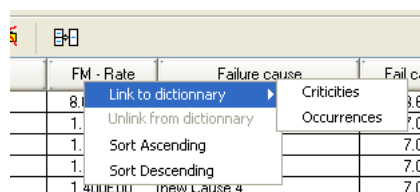
The user can save the correspondences in its preferences by pressing the **Save preferences** button.

When he has finished its setting, the user presses the **Finish** button. The data import is then initiated. When it is finished, the edited FMEA table contains the data that has been imported.

### 8.3.3.8. Using the data dictionaries

While editing FMEA tables, the dictionaries can be used to display in the cells the names of the threshold associated to the numerical intervals instead of the real numerical values themselves.

Providing that the user has first defined at least a dictionary, the user can associate a dictionary to a column by using the **Link to dictionary** command accessible in the column popup menu. The user clicks with the right button on column header to show the popup menu as on the following figure:

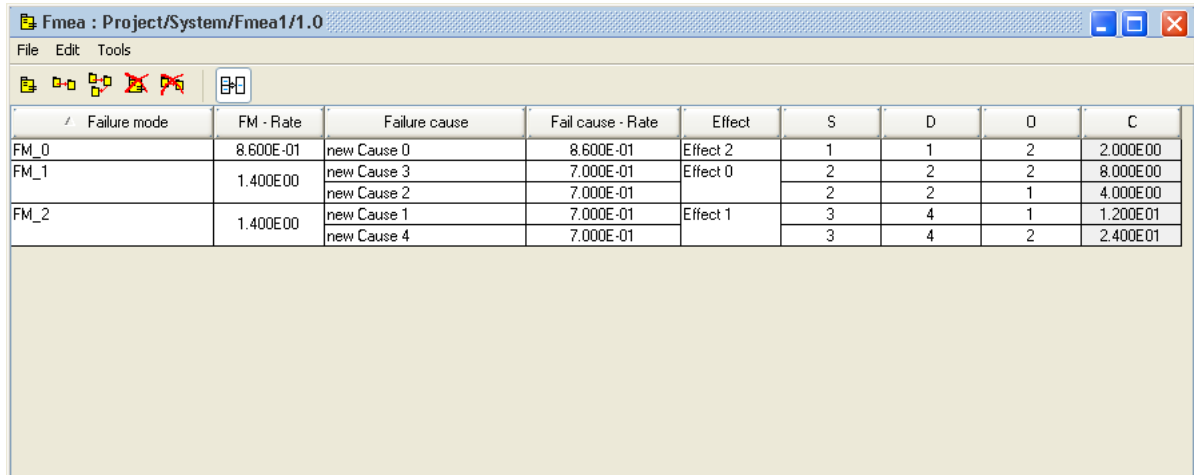


The user then can choose the wanted dictionary. The previous figure shows the two following dictionary : 'Criticalities' and 'Occurrences'.

To remove the link between the column and the dictionary, the user launches the **Unlink from dictionary** command also located in the column popup menu.

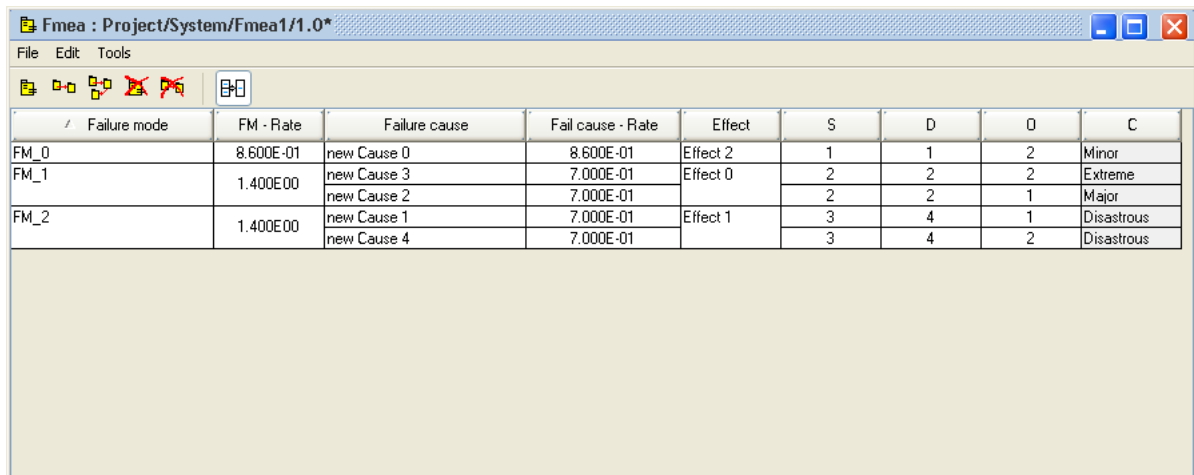


Let us take an example. Let us suppose that the user has entered the following information in the FMEA table:



Failure mode	FM - Rate	Failure cause	Fail cause - Rate	Effect	S	D	O	C
FM_0	8.600E-01	new Cause 0	8.600E-01	Effect 2	1	1	2	2.000E00
FM_1	1.400E00	new Cause 3	7.000E-01	Effect 0	2	2	2	8.000E00
		new Cause 2	7.000E-01	Effect 0	2	2	1	4.000E00
FM_2	1.400E00	new Cause 1	7.000E-01	Effect 1	3	4	1	1.200E01
		new Cause 4	7.000E-01	Effect 1	3	4	2	2.400E01

If the user decides to associate to the 'C' column (last column of the table) the criticities dictionary mentioned in the chapter concerning the Data Dictionary Manager, the FMEA table will appear as following:

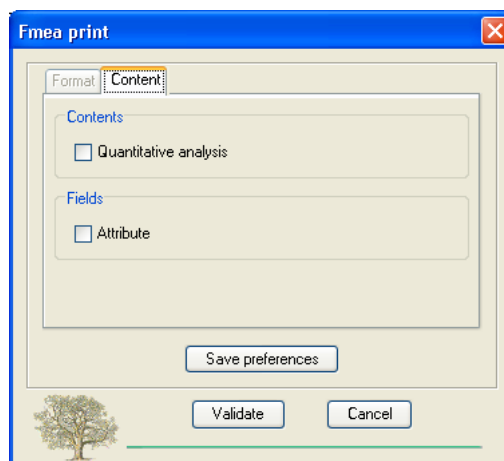


Failure mode	FM - Rate	Failure cause	Fail cause - Rate	Effect	S	D	O	C
FM_0	8.600E-01	new Cause 0	8.600E-01	Effect 2	1	1	2	Minor
FM_1	1.400E00	new Cause 3	7.000E-01	Effect 0	2	2	2	Extreme
		new Cause 2	7.000E-01	Effect 0	2	2	1	Major
FM_2	1.400E00	new Cause 1	7.000E-01	Effect 1	3	4	1	Disastrous
		new Cause 4	7.000E-01	Effect 1	3	4	2	Disastrous

### 8.3.3.9. Printing an FMEA table

The printing FMEA table functionality considers the possible external references from FMEA table attributes. To know more about it, the user is invited to go and look at the 'Referring to a set of data in an FMEA table from another table' chapter.

The FMEA table printing is done by using the **File - Print** command located in the application main menu bar or from the corresponding button on the toolbar. The command activation shows the following dialog window:



This window allows the user to choose the printing options before its effective launching. The options are the following ones:

- **Quantitative analysis:** print the section containing the FMEA table properties
- **Attribute:** print the 'Attribute' column in the tables showing the external references

When all the options are set, the user launches the printing by pressing the **Validate** button.

The following figures show an example the printing of an FMEA table containing attributes with external references:

**DASSAULT AVIATION** 2/19/08 page 1/2

**FMEA TABLE**

1. QUANTITATIVE ANALYSIS

\* FMEA reference  
 Base : DataBase  
 Project : Project (1.0)  
 System : System (1.0)  
 FMEA : FMEA1 (1.0)

\* Dates  
 Creation : 01-Feb-2008 10:59:58 by admin  
 Modification : 01-Feb-2008 10:59:58 by admin

2. FMEA TABLE

FMEA								
Failure mode	FH ~ Rate	Failure cause	Fail cause ~ Rate	Effect	S	D	O	C
FM_0	8.600E-01	new Cause 0	8.600E-01	Effect 2	1.000E00	1.000E00	2.000E00	Minor
FM_1	1.400E00	new Cause 0	7.000E-01	Effect 0	2.000E00	2.000E00	2.000E00	Extreme
		new Cause 2	7.000E-01		2.000E00	2.000E00	1.000E00	Major
FM_2	1.400E00	new Cause 1	7.000E-01	Effect 1	3.000E00	4.000E00	1.000E00	Disastrous
		new Cause 4	7.000E-01		3.000E00	4.000E00	2.000E00	Disastrous

3. EXTERNAL REFERENCES LIST

\* Entity : Cause.new Cause 0  
 Value : 8.600E-01

External references			
FMEA path	Entity	Attribute	Value
Project/System/FMEA2/1.0	Cause.Cause_0	Rate	8.000E-01
Project/System/FMEA3/1.0	Cause.Cause7	Rate	8.000E-01

This example was generated with the **Quantitative analysis** and **Attribute** options selected. The second print page shows the existence of an attribute linked to two external references listed in the corresponding table.

### 8.3.3.10. Generating a file report on the FMEA table

The FMEA table generation functionality considers the possible external references from FMEA table attributes. To know more about it, the user is invited to go and look at the 'Referring to a set of data in an FMEA table from another table' chapter.

The file report generation on an FMEA table is done using the **File report** command accessible from the corresponding button on the application toolbar:



The activation of this command shows the following dialog window:

**Fmea file report**

Format Content

☐ SCRIBE  
☐ TEXTE  
☐ HTML  
☐ XML  
☒ RTF

File name: E:\Arbor-Data\Fmea1(1.0).rtf

Save preferences

Validate Cancel

The **Content** tab allows the user to choose the options of file report generation before its effective launching.

- When all the options are set, the user launches the file report generation by pressing the **Validate** button.

**DASSAULT AVIATION**

DATE: 13/02/08 PAGE: 1/1

### FMEA TABLE

**QUANTITATIVE ANALYSIS**

- Fmea reference:
  - Base: DataBase
  - Project: Project (1.0)
  - System: System (1.0)
  - Fmea: Fmea1 (1.0)
- Dates:
  - Creation: 01-Feb-2008 10:59:58 by admin
  - Modification: 01-Feb-2008 10:59:58 by admin

**FMEA TABLE**

Failure mode		FM - Rate	Failure cause	FMEA Fail cause - Rate	Effect	S	D	O	C
FM_0	8.600E-01	new Cause 0	8.600E-01	Effect 2	1.000E00	1.000E00	2.000E00	Minor	
FM_1	1.400E00	new Cause 3	7.000E-01	Effect 0	2.000E00	2.000E00	2.000E00	Extreme	
		new Cause 2	7.000E-01		2.000E00	2.000E00	1.000E00	Major	
FM_2	1.400E00	new Cause 1	7.000E-01	Effect 1	3.000E00	4.000E00	1.000E00	Disastrous	
		new Cause 4	7.000E-01		3.000E00	4.000E00	2.000E00	Disastrous	

**EXTERNAL REFERENCES LIST**

- Entity: Cause.new Cause 0
- Value: 8.600E-01

**External references**

Fmea path	Entity	Attribute	Value
Project/System/Fmea2/1.0	Cause.Cause_0	Rate	8.000E-01
Project/System/Fmea3/1.0	Cause.Cause0	Rate	2.000E-01

#### 4. Referencing data in FMEA tables

- from an event model in the failure rate bank,
- from another FMEA table.

In addition, the referenced data must be a numerical value. The column cells of any other type than the numerical one (Double or Integer) cannot be referenced.

- the target Fmea table identifier,
- the identifier of the entity containing the attribute,
- the attribute identifier.

In this case, the user can link an event model law parameter to an FMEA table entity attribute. When the link is created, all modification of the attribute is sent back to the linked law parameter.

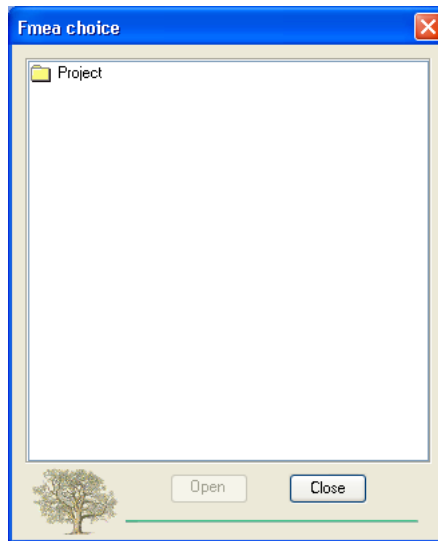
To create a link, the user must first edit the concerned event model. Remind that the event model edition is done in the following internal window:

From this point, the user clicks on the (...) button to access to the law parameter assignment window and selects the **Link to Fmea** tab.

The **Link to FMEA** tab contains the following fields:

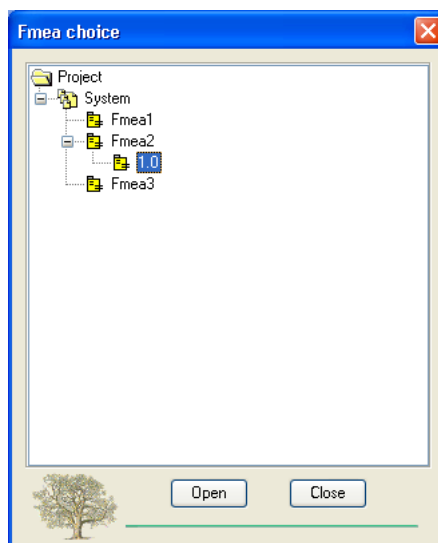
- **FMEA** : the FMEA table containing the attribute,
- **Entity** : the entity containing the attribute,
- **Attribute** : the attribute,
- **Value** : the attribute value.

To put in place the link, the user clicks on the (...) button located at the right of the **FMEA** field. The following dialog window then appears:



This first step allows the user to select the FMEA table in the existing document tree. The user double clicks on the tree nodes in order to make their sons appear. Then the user browses the tree structure until the version node of the FMEA table that has interest to him.

The following figure shows what the window can look like when the user expands a branch until an FMEA table version node:



You will notice that the tree shows only the nodes corresponding to FMEA tables and not those that correspond to fault trees. In addition, the **Open** button is enabled only when the selected node is the version node of an FMEA table.

At this time, to edit the selected FMEA table, the user double clicks on the version node or presses the **Open** button which leads to the opening of the following window:

Cause	Rate
Cause_1	4.000E-01
Cause_2	3.000E-01
Cause_0	8.000E-01
Cause_3	1.000E-01

To select the attribute of interest, the user must select a corresponding cell displaying this attribute. He then presses the **Validate** button to confirm its choice. Alternatively he can also double click on the chosen cell.

Once the validation done, the selection windows close and the **Assignment** window fields are filled with the selected data as the following figure shows:

To confirm its action, the user must at last press **Validate** button, otherwise he presses the **Cancel** button.

#### 8.4.2. Referencing an FMEA table data set from an FMEA table

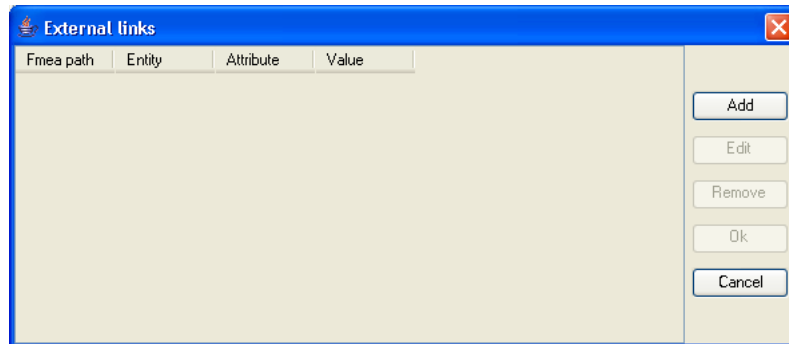
The user can reference several entity attributes of numerical type in order to compute the value of another FMEA table entity attribute being edited. the formula to compute the reference attribute is the Pointcarré approximation formula:

$$V = (v1+v2) - (v1 \times v2)$$

Applied on two input variables.

To calculate the value on more input, we calculate successively the V intermediate values. For instance, for the three input variables v1, v2 and v3, we calculate the V value from v1 and v2 then from V and v3 to obtain the final value of V.

To define the input variables of the formula from the edited FMEA table, the user clicks on the computed cell (which must get the result of the Pointcarré formula), then clicks with the right button to show the popup menu and selects the **Link to Fmeas** command. The following window appears:



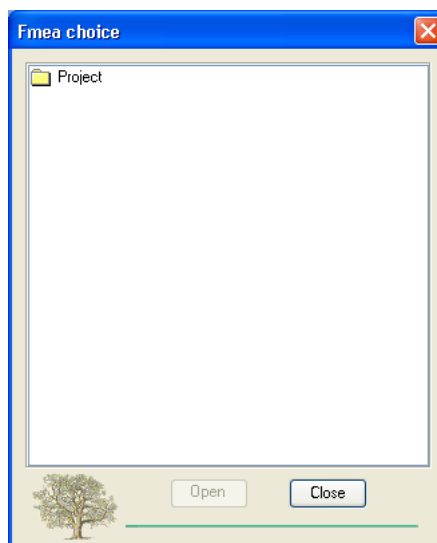
This window shows all the referenced attributes from the attribute on which the formula is defined, in a table which of the signification of the columns is the following one:

- **Fmea** : the FMEA table containing the attribute,
- **Entity** : the entity containing the attribute,
- **Attribute** : the attribute,
- **Value** : the attribute value.

Furthermore, the available command from this window and using buttons are the following ones:

- **Add** a reference,
- **Edit** a reference,
- **Remove** a reference,
- **Validate** the modifications,
- **Cancel** the modifications.

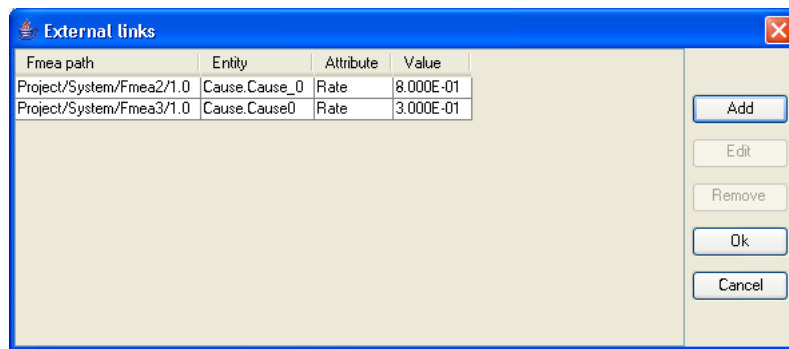
To add a new reference, the user presses on the **Add** button. The following window appears:



The process of selection of the referenced attribute is then the same as for the creation of a reference from an event model: first, the user selects the FMEA table from the document manager then the target attribute in this FMEA table.

We will notice that at the FMEA table selection stage, the user is not authorized to select an FMEA table that would lead to a circular reference ring between FMEA tables.

Assuming that the user has referenced two attributes from distinct FMEA tables, the **External links** window could appear like this:



To finish, the user confirms its modification by pressing the **Validate** button.

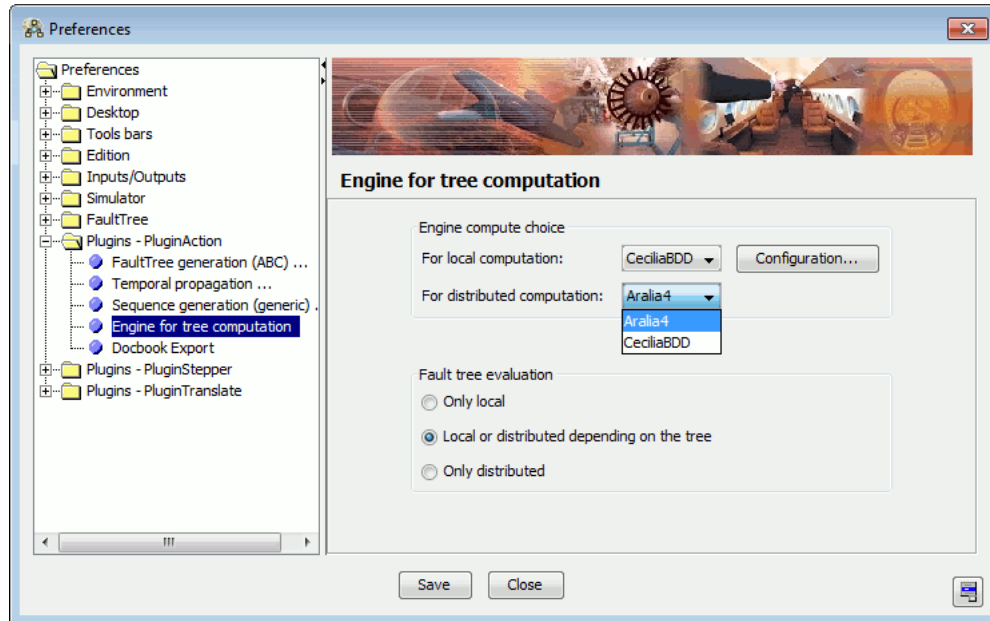


## 9. Grid computing

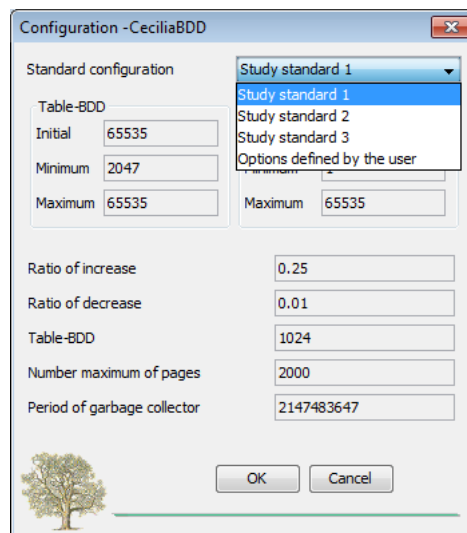
### 9.1. Introduction

The aim of the **Grif computing** is to launch computation a part or all the calculations in another computer. User can use **Grif computing** for fault tree computation or for sequence generation.

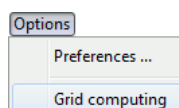
In **Options-Preference**, user can choose the type of calculation wanted in **Plugins - PluginAction - Engine for tree computation**.



For local computation, user can configure the memory allocated using **Configuration** button. In general, more the the tree is important, more it is necessary to allocate memory using **Study standard i**.



Grid computing window can be opened with the **Options-Grid computing** menu.

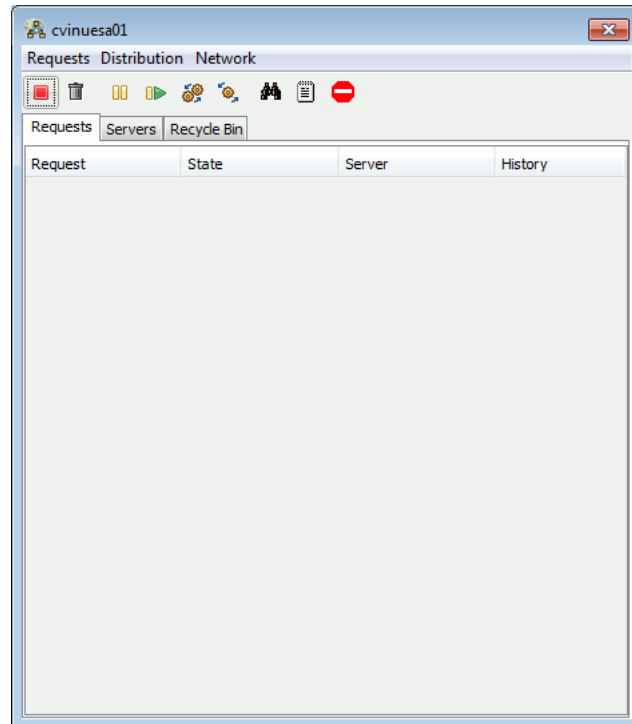


## 9.2. Grid computing interface

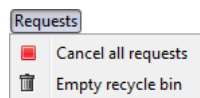
### 9.2.1. Presentation of interface

The interface is made up of 5 parts:

- A menu bar : Contains all actions that can be performed
- An icon bar : Enables quick access to actions
- A requests tab : Contains requests table (number, state, server which compute, history)
- A servers tab : Contains servers table (server name, stock size, current computed request, checkbox to say if server have to be used or not)
- A recycle-bin tab : Contains requests that are finished (request, state, last server having computed it) (a double-click on a request opens detailed history)



### 9.2.2. Requests menu



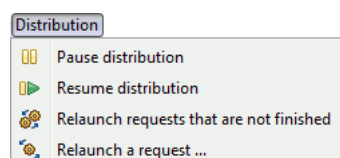
#### 9.2.2.1. Cancel all requests

The **Cancel all requests** menu cancels all requests. Servers are warned that requests are cancelled. Then they are put in the recycle-bin.

#### 9.2.2.2. Empty recycle bin

The **Empty recycle bin** menu remove all requests from recycle bin.

### 9.2.3. Distribution menu



### 9.2.3.1. Pause distribution

The **Pause distribution** menu enables to stop sending requests to servers. Requests stay in NEW state. Distribution can be resume with **Resume distribution**.

### 9.2.3.2. Resume distribution

The **Resume distribution** menu enables to resume requests sending.

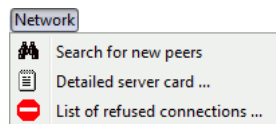
### 9.2.3.3. Relaunch requests that are not finished

The **Relaunch requests that are not finished** menu enable to relaunch all request that are not yet fully threatred. These requests are cancelled on servers, their state are set to NEW, and then they are redistributed.

### 9.2.3.4. Relaunch a request

The **Relaunch a request** menu does the same action that the one described before, but for only one request whom number has to be specified in the dialog box.

## 9.2.4. Network menu

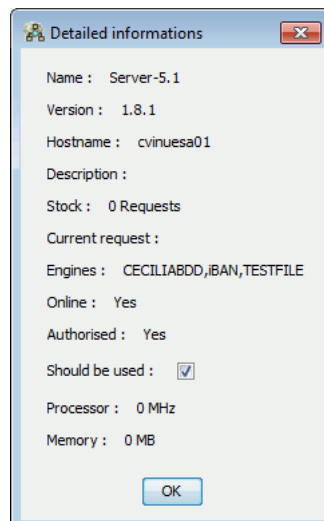


### 9.2.4.1. Search for new peers

The **Search for new peers** menu enables to search new peers on network. New detected peers will be added to list. This action can take a long time (10 - 20 seconds), a waiting window is displayed during this time.

### 9.2.4.2. Detailed server card

The **Detailed server card** displays detailed information about selected server. If no server is selected, the **Servers** tab will be displayed, and a message will inform user that a server must be selected in list. This server card can also be displayed by double-clicking on a server in the list.



This detailed card contains:

- The name of the peer
- The hostname of the computer hosting the peer
- The description of the server
- The number of requests that are in stock on server

- The current computed request
- The list of engines that are available on server
- The online status
- The authorization to communicate with server or not
- A checkbox **should be used** to choose if server should be used or not, for computations.
- The processor frequency (not yet implemented)
- The amount of memory available on the server (not yet implemented)

## 10. Others functionalities

### 10.1. Event global list

#### 10.1.1. Introduction

The event global list is a list of which the event is the reference data. An event can exist in several trees. In that case, the event appears in several lines in the data extraction result table if the corresponding trees are part of the extractions criteria.

The event global list display is composed of the two following steps:

- Data generation in the database;
- Data extraction and display.

These steps are described in the following sections.

#### 10.1.2. Data generation

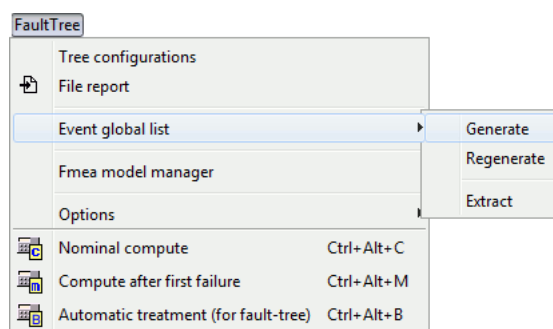
The data generation consists in collecting information from all events contained in all trees from the database and stores it in a central table in order to get a fast access while extracting it. The first time this operation can cost a lot of time because it depends directly of the total number of events in the database. It can be costly in space for the same reason. Inside, the data generation proceeds as follows:

- Open each tree;
- Read and store information relatives to all events.

The data generation take account of the last modified time stamp and the likely reading error. The cost is decreased because only the trees modified after a given generation are treated during the next generation. The trees that could not have been read during a given generation are read again during the next generation.

Be aware: during the data generation, all the strings are truncated to 255 characters.

To start the data generation, the user select the command **Fault Tree - Event global list - Generate**.



When the data generation is done, the user can extract a subset and display it in the Event global list window.

#### 10.1.3. Data extraction

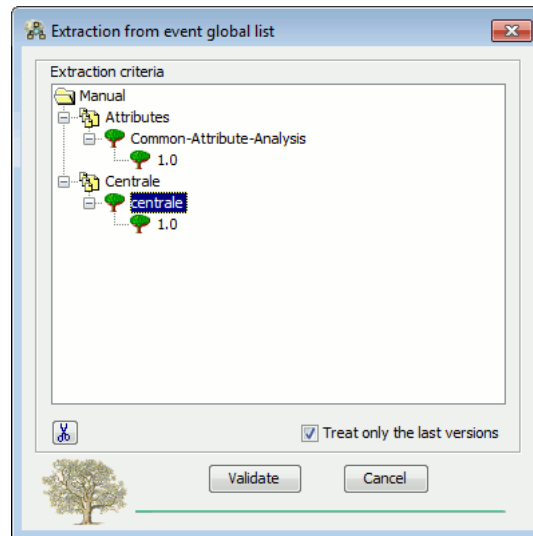
The data extraction consists to select the data the user would like to display. Using this principle the time cost is decreased and the displayed data are more useful for the user.


The data extraction follows the steps:

- Display Extract from event global list window;
- Add projects, systems or trees to the extract criteria using a drag and drop of the project, systems or trees;

- Data extraction.

To show the Extraction from event global list window, the user apply the command **Fault Tree - Event global list - Extract**. The Extraction from event global list window appears as follows:



This window shows the selected extraction criteria. Criteria constitute the application field of the extraction. Information is organized in a Project - System - Tree paths. The user can add or remove criteria. To add a criteria, the user select the project and drag and drop the folder. To remove a criterion, the user presses the icon .

When the all the criteria are selected, the user click on the **Validate** button to extract the data or on the **Cancel** button to close without doing anything.

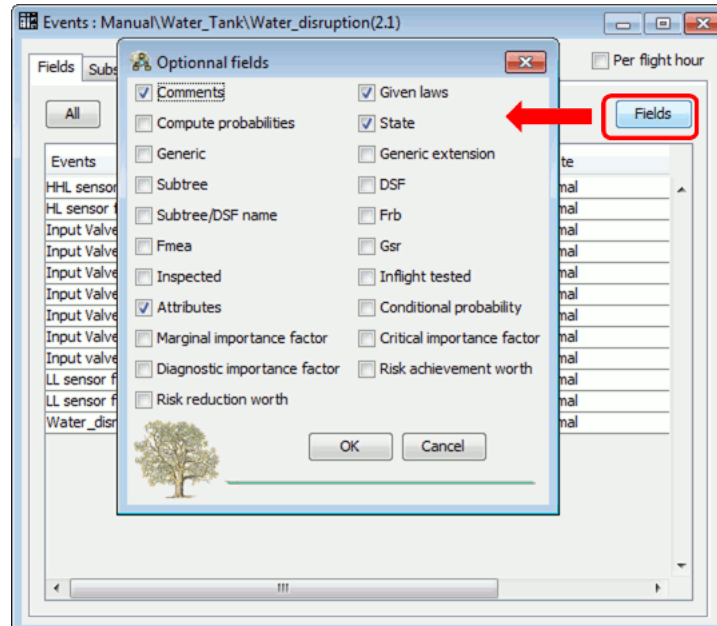
#### 10.1.4. Data display

At the end of the extraction process, the Event global list window appears as follows:

Projects	Systems	Trees	Events	Given laws	Inspec...	Attributes
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	A	exponential 1.000E-03	-	'Zone[1]'=[1]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	B	exponential 1.000E-03	-	'Zone[1]'=[1]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	C	exponential 1.000E-03	-	'Zone[1]'=[2]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	Common-Attribute-Analysis	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	D	exponential 1.000E-03	-	'Zone[1]'=[2]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	E	exponential 1.000E-03	-	'Zone[1]'=[2]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	F	exponential 1.000E-03	-	'Zone[1]'=[3]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	G	exponential 1.000E-03	-	'Zone[1]'=[3]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	H	exponential 1.000E-03	-	'Zone[1]'=[4]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	I	exponential 1.000E-03	-	'Zone[1]'=[4]
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	J	exponential 1.000E-03	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS1	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS2	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS3	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS4	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS5	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS6	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS7	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS8	-	-	-
Manual	Attributes	Common-Attribute-Analysis ( 1.0 )	MCS9	-	-	-
Manual	Centrale	centrale ( 1.0 )	And11	-	-	-
Manual	Centrale	centrale ( 1.0 )	And21	-	-	-
Manual	Centrale	centrale ( 1.0 )	And22	-	-	-
Manual	Centrale	centrale ( 1.0 )	And23	-	-	-

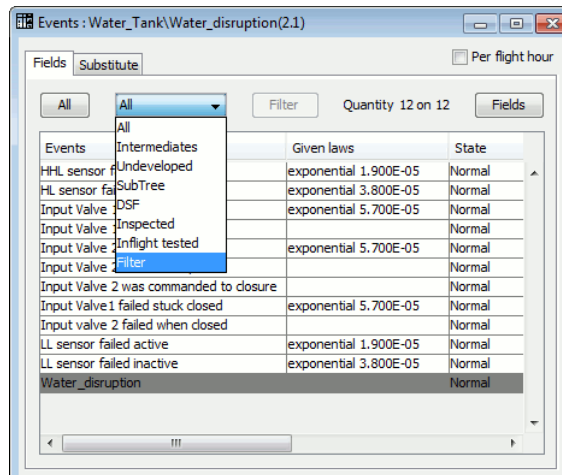
The **List tab** contains the data extracted using the extraction criteria. Each table line contains all the information relatives to an event. The user can select the event type he would like to be appearing in the table (undeveloped, intermediate, etc.) by using the combo box.

He can also display the fields he would like to be visible in the table (table columns) by selecting them in the **Fields tab**:

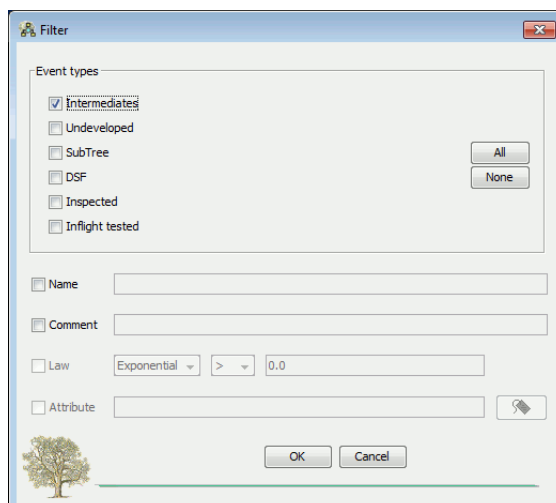


Finally, the user can apply a filter on the data by selecting the Filter option in the combo box and fill in the different fields as needed in the Filter tab:

- In the drop-down list, select the filter criteria;
- Or directly filter to create a personal filter.



- If **Filter** is selected, the following window appears to create the filter.

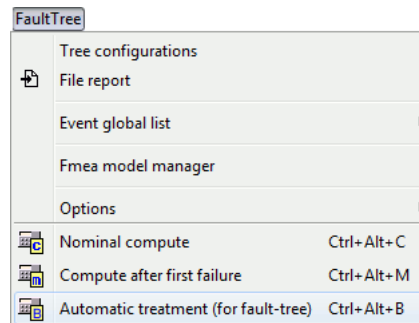


## 10.2. Batch computation

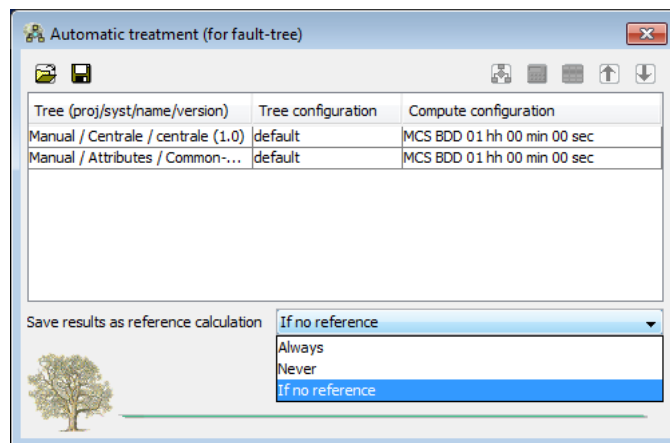
The aim is launch all the calculation automatically in the same time.

To configure the automatic treatment:

Click on the following icon:  or in the menu **FaultTree - Automatic treatment (for fault tree)** :






The following window appears in order to set-up the calculation:

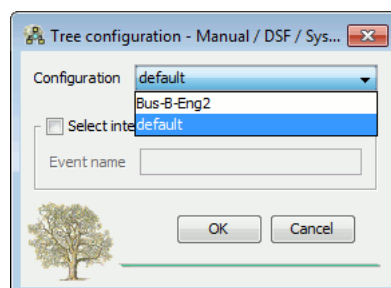


The Automatic treatment window contains the tree list on which a computation has to be done.


To add a new tree in the list, use a drag and drop of the tree, project to the Automatic treatment window.

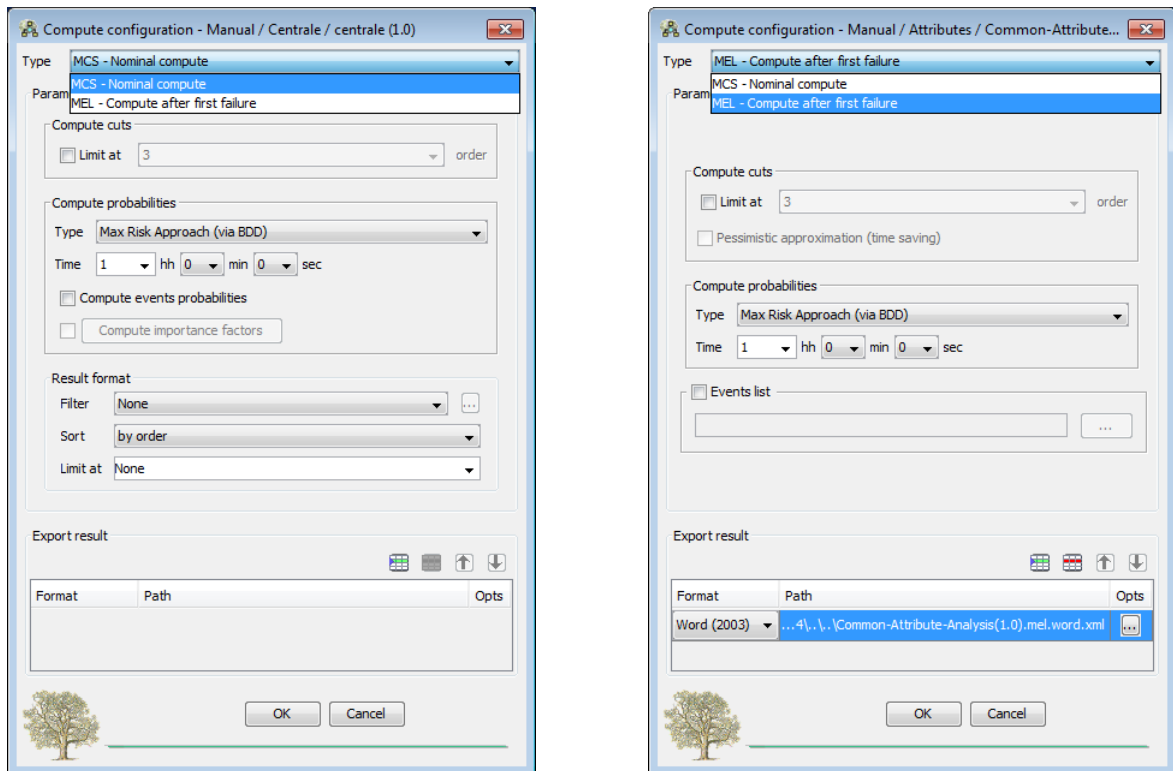
The Automatic treatment window contains the following buttons:

-  **Open XML fil containing compute list** : loads a list of calculation already performed and previously saved;
-  **Save list in XML file** : allows storing of the computation informations,
-  **Edit tree configuration** : allows to configure the tree. For example if the calculation used default configuration or DSF or Sub-trees or CCF configurations. This option is also accessible with a double-clicks on the Tree configuration column. The following dialog box appears:





-  **Edit compute configuration:** allows to configure the tree. For example if the calculation used default configuration or DSF or Sub-trees or CCF configurations. This option is also accessible with a double-clicks on the Compute configuration column. The following dialog box appears:







In this dialog box in the first part, user select the different type of computation and parameters for the calculation in the Section 5.1.4, “Minimal cut set calculations (MCS)” and Section 5.1.6, “Minimum equipment list computations (MEL)”

The second part corresponds to the result format and export.

For MCS calculation, in the **Filter** field it is possible to filter result by Standard, Relative weight or Latent Specific Risk. All these filters/ sorts and limitations are detailed in the Section 5.1.4, “Minimal cut set calculations (MCS)”

The export results parts permits to select the file format and the path where the file is saved. The following formats are available:

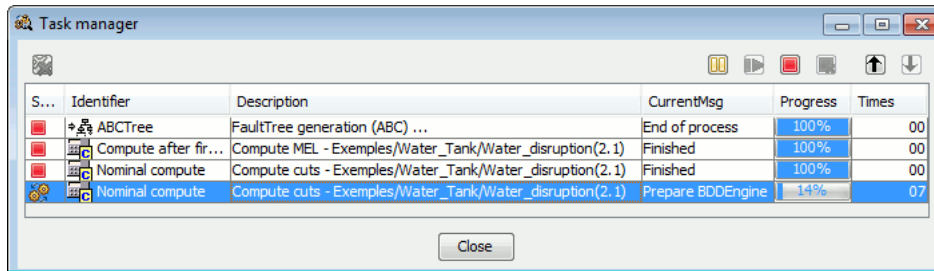
Format	Path	Opts
Word (2003)	...4\...\Common-Attribute-Analysis(1.0).mcs.word.xml	
XML		
Docbook		
Word (2003)		
CAA Docbook		
CAA Word		
Tree View		

-  **Delete selection :** delete the selected tree,
-  **Move up :** to move up in the list of trees,
-  **Move down :** to move down in the list of trees,

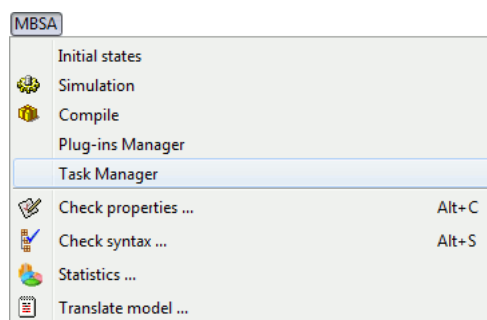
**Save results as reference calculation** permits to save results as reference or not.

## 10.3. Task manager

**Task Manager** shows the calculations, fault tree generation, sequence generation,... That are currently running or already performed.



**Task Manager** is automatically displayed when calculation or generation are performed. User can display the window in **MBSA** menu.



This tab is made of 6 columns:

- Status: finished, in progress, ... ;
- Identifier: type of calculation or generation;
- Description: type of calculation or generation and file name;
- CurrentMsg;
- Progress: progress bar;
- Time: time calculation.

In **Task Manager** some actions are available:

- : Clear all compute;
- : suspend selected compute;
- : resume compute in suspend;
- : stop selected compute;
- : remove selected compute;
- : up the compute;
- : down the compute;

These actions are also available by a right click in a compute line.

When a task is added to TaskManager, user is not blocked until the task is ended. He can continue to work. He can even relaunch a calculation or a generation of result. The various tasks accumulate and are treated sequentially. It is possible to pause the current task to launch the following task because the user considers it as more priority..

## A. Laws and uncertainties

### 1. Description of the laws

A total of 10 laws are available in the Cecilia-Workshop module. Each of these laws has one or more corresponding parameters. Here is a list of the different "types" possible:

- Probability: value between 0 and 1 inclusive.
- Rate: value greater than or equal to corresponding to a failure rate.
- Duration: value greater than or equal to 0 corresponding to a duration or to a time.
- Factor: value strictly greater than 0.
- NatInt: integer value greater than or equal to 0.
- Boolean: can take a value of 0 or 1 corresponding to an option parameter.
- Other: any value.

In the remainder of this chapter, the parameter "types" will be specified for each law.

#### 1.1. Constant probability

This law has one parameter: the probability **q** of the event. Whatever the time, the probability of the component failing is constant.

Parameter:

- **q** : constant (Probability)

The law is defined as follows:

$$Q(t) = q$$

This law generally corresponds to the case where the only failure considered for the components is that of a refusal to change state (e.g.: Fails to start/stop, etc.).

#### 1.2. Exponential probability

This law only has a one parameter: the component's failure rate (supposed to be constant over time). It describes the time interval before the first failure for a non-repairable component.

Parameters:

- **Lambda** (Rate) = failure rate

The law is defined as follows:

$$Q(t) = 1 - e^{-\lambda t}$$

This law is widely used since it is almost the only one to make it possible to obtain analytical results. In addition, it describes the lifetime of a non-repairable component very well (at least when there are a large number of components) when the component is no longer young.

#### 1.3. Constant mission time

It corresponds to an exponential law with a fixed time given as parameter.



This law does not depend on the time, it is a short version of a constant law.

Parameters:

- **Lambda** (Rate) = failure rate
- **T** (Duration) = mission time
- **Q** (Probability) = optional law

The law is defined as follows:

$$Q(t) = Q + 1 - e^{-\lambda t}$$

### 1.4. Weibull probability

This law has three parameters: **alpha** , **beta** and **t0** . It describes the behavior of a component which is not repairable and which does not fail to start. Its specific feature is that it takes account of the component's young and old periods.

Parameters:

- **Alpha** (Factor) = scale parameter
- **Beta** (Factor) = form parameter
- **T0** (Time) = location parameter

The law is defined as follows:

$$Q(t) = 1 - \exp \left[ - \left( \frac{t - t_0}{\alpha} \right)^\beta \right]$$

The significance of this law is that new distributions can be tested by varying the **beta** factor:

- If **Beta** is less than 1, the failure rate decreases and the law then allows the period when the component is young to be taken into account.
- If **Beta** is greater than 1, the failure rate increases and the law then allows the component's ageing period to be taken into account.
- If **Beta** is equal to 1, the Weibull law is equivalent to the exponential law.

### 1.5. Periodic testing probability

This law allows a component which fails to be represented according to an exponential distribution law and whose failure is found during a periodic test. The repair is then carried out instantaneously.

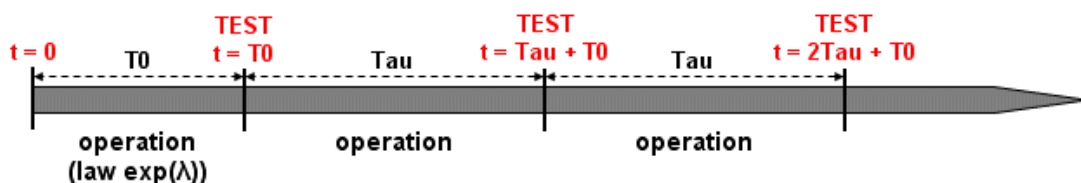
Parameters:

- **Lambda** (Rate) = failure rate
- **Tau** (Duration) = test period (time interval between two consecutive tests)
- **T0** (Time) = date of first test

The law is defined as follows:

$$Q(t) = \begin{cases} 1 - e^{-\lambda t} & \text{if } t < t_0 \\ 1 - e^{-\lambda[(t-t_0) \bmod \tau]} & \text{otherwise} \end{cases}$$

Here is a small graph representing the different phases of the component's "life":



This law is a simplified version of the Complete "Periodic testing" probability" law.

### 1.6. Complete "Periodic testing" probability

This law allows a periodically tested component to be represented as completely as possible. There are many parameters in play.

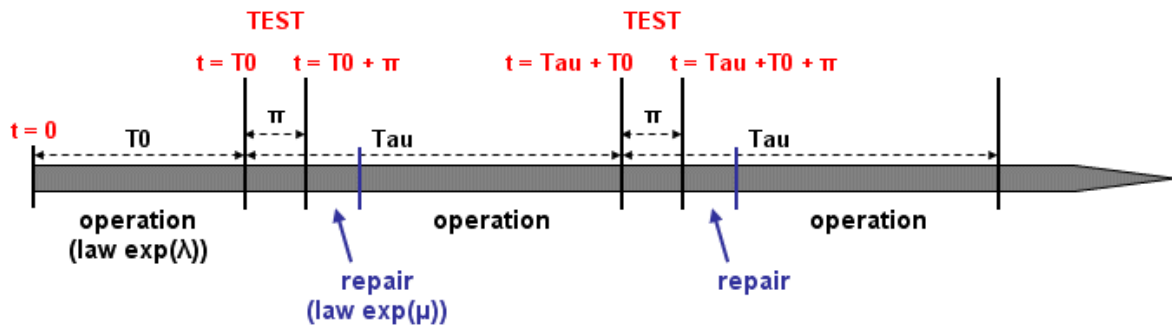
Parameters:

- **Lambda** (Rate) = failure rate during operation or on standby
- **Lambda\*** (Rate) = failure rate during the test
- **Mu** (Rate) = repair rate (once the test has shown up the failure)
- **Tau** (Duration) = test period (time interval between two consecutive tests)
- **Theta** (Time) = date of first test (ignore parameter value: **Tau** )
- **Gamma** (Probability) = probability of failure due to starting the test (ignore parameter value: 0 = starting the test does not cause a failure)
- **Pi** (Duration) = duration of test (ignore parameter value: 0 (instantaneous test))
- **X** (Boolean) indicator of component availability during the test (0 = component unavailable during the test; 1 = component available) (ignore parameter value: 1 = available during the test)
- **Sigma** (Probability) = test cover rate (probability that the component failure is detected during the test) (ignore parameter value: 1 = the test covers all the possible failures)
- **Omega** ((Probability) = probability of omission of reconfiguration (after the test and after repair) (ignore parameter value: 0 = no reconfiguration problem)



The "ignore parameter value" is the value to type if you want parameter to do not affect component availability.

Here is a small graph representing the different phases of the component's "life":



## 1.7. Dormant law

Components that form part of a standby system, which may lie dormant for long periods of time, may not have failures revealed until they are required to operate or are tested during maintenance or a start-up inspection. Only at these critical times can repairs be performed. It is during the dormant time that the system's components exhibit characteristics similar to non-repairable components. Variations of unavailability with time are periodic because of the different lengths of time that components may lie dormant.

This law has three parameters: a failure rate, a mean repair time and a delay. In addition, it does not depend on the time.

Parameters:

- $\lambda$  (Rate) = failure rate
- **MTTR** (Duration) = average repair time
- $\tau$  (Duration) = delay

The law is defined as follows:

$$Q_{mean} = \frac{\lambda \cdot \tau - (1 - e^{-\lambda \cdot \tau}) + \lambda \cdot MTTR \cdot (1 - e^{-\lambda \cdot \tau})}{\lambda \cdot \tau + \lambda \cdot MTTR \cdot (1 - e^{-\lambda \cdot \tau})}$$

for MTTR = 0:

$$Q_{mean} = \frac{\lambda \cdot \tau - (1 - e^{-\lambda \cdot \tau})}{\lambda \cdot \tau} = 1 - \frac{(1 - e^{-\lambda \cdot \tau})}{\lambda \cdot \tau}$$

## 1.8. NRD probability

This law takes two parameters: a repair rate **Mu** and a recovery time **D**. For non-repairable components, it gives the probability of not succeeding to recover the component before a delay Delay.



This law does not depend on the time, it is a short version of a constant law.

Parameters:

- **Mu** (Rate) = repair rate
- **d** (Duration) = recovery time

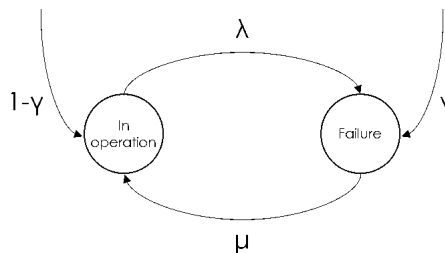
The law is defined as follows:

$$Q(t) = e^{-\mu d}$$

## 1.9. GLM probability

This law describes the behavior of a component (repairable or not), with (or without) failure to start, using exponential expressions. It generalizes the exponential law with the **Lambda** parameter (failure rate).

It is equivalent to the following Markov diagram:



Parameters:

- **Gamma** (Probability) = probability of initial start failure (at t = 0)
- **Lambda** (Rate) = failure rate
- **Mu** (Rate) = repair rate

The law is defined as follows:

$$Q(t) = \frac{\lambda}{\lambda + \mu} - \frac{\lambda - \gamma(\lambda + \mu)}{\lambda + \mu} \times e^{-(\lambda + \mu)t}$$

The **Gamma** and **Mu** parameters are optional. Depending on the case, they can be zero.

- If the component is not repairable, set **Mu** to zero.
- If the component cannot fail to start, set **Gamma** to zero.

The failure to start is only taken into account at t = 0.

## 1.10. GLM Asymptotic law

This law corresponds to the probability of a Gamma-Lambda-Mu law calculated at t = infinity.

NB: This law does not depend on the time, it is a short version of a constant law.

Parameters:

- **Lambda** (Rate) = failure rate

- **Mu** (Rate) = repair rate

The law is defined as follows:

$$Q(t) = \frac{\lambda}{\lambda + \mu}$$

## 2. Modifier coefficient of the law

It is possible to modify the law using a coefficient called Bound time.

**Bound time** indicates the duration where the law is considered. Before **t<sub>0</sub>** it is considered the probability for the selected law at 0. Between **t<sub>0</sub>** and **t<sub>0</sub> + delay** the selected law at **t = t - t<sub>0</sub>** and after the law considered for **t<sub>0</sub> + delay**.

## 3. Uncertainties on the parameters

For each probability law used in the model, it is possible to introduce an uncertainty on each of the parameters. There are two distributions available::

- Uniform distribution;
- Lognormal distribution;

The principle underlying sensitivity studies is to run a Monte-Carlo simulation on the calculation of the probability of the top event.

Each history of the simulation comprises three stages:

- history of the simulation comprises three stages: First, the value of the parameters of the probability laws are drawn pseudo-randomly according to the distributions set;
- For each time point considered, you then calculate the probabilities of the terminal events on the basis of the parameter values;
- Finally, you calculate the value of the top event on the basis of the probabilities of the terminal events and the binary decision diagram coding the top event.

The same operation is then repeated and naturally delivers a second result, different from the first. You recommence 5 times, 10 times, 100 times ... N times until you obtain a sample considered statistically representative.

The resulting sample is then statistically processed, by calculating:

The mean **m** of this collection of values:

$$m = \frac{\sum_{i=1}^N Valeurs}{N}$$

A standard deviation **σ**

A confidence interval at 90%:

$$[m - q, m + q] \text{ with } q = \frac{1.645 \sigma}{N}$$

Sensitivity studies can be performed on all the risk measurements which can be calculated by Aralia, i.e. not only the probability of the top event, but also the various importance factors and the magnitudes for calculating approximate reliability.

### 3.1. Uniform distribution

This law has two parameters: and upper limit and a lower limit.

Parameters:

- **a** = upper limit
- **b** = lower limit

The law is defined as follows:

$$Q(t) = \frac{(t-a)}{(b-a)}$$

### 3.2. Lognormal distribution

This law has two parameters: the mean and the standard deviation.

Parameters:

- **Mu** = mean
- **Sigma** = standard deviation

A random variable is distributed according to a lognormal law if its logarithm is distributed according to a normal law.

If  $\mu$  and  $\sigma$  are respectively the mean and the standard deviation of the relevant normal law, the probability density of the random variable is as follows:

$$Q(t) = 1 - \int_0^t f(t) dt \quad f(t) = \frac{1}{t\sigma\sqrt{2\pi}} e^{-\left(\frac{(\ln t - \mu)^2}{2\sigma^2}\right)}$$



## B. Probabilities computations

### 1. Introduction / Aeronautic Context

Cecilia-Workshop software is developed to perform reliability and availability studies for aeronautic domain.

In this context, whether it is the authorities of validation / certification or whether it is the standards to apply, the quantitative objective is in link with a probabilistic risk for an average flight. Mission time corresponds to average flight duration.

In a plane, components can be grouped in various types of component.

There are components:

- Structural which are often considered as non-repairable over the plane life.
- Periodically tested: equivalent of components tested / replaced during the big and small maintenances on an industrial equipment.
- Systematically tested after or before each flight.
- Used only during a certain phase of the plane flight (take-off, landing, ...)

Furthermore, except very particular case, during the flight, components are not repairable.

During flight duration, Failure probability of a component not reparable is given by an exponential law (failure rate is constant). On the other hand the test duration or repair time of a component can be considered as null because they are realized on the ground. Besides, the inspections taking place on the ground, the various tested components have all a period of inspection which is a multiple of the flight duration of the plane.

If the temporal reference corresponds to plane life, components can be considered as periodically tested component.

If the temporal reference corresponds to the next flight, components can have already lived a number flight (since the previous maintenance) and potentially with not detected failure. We speak then about hidden failure or dormant failure.

In all cases, aeronautic reliability engineers link to failure mode:

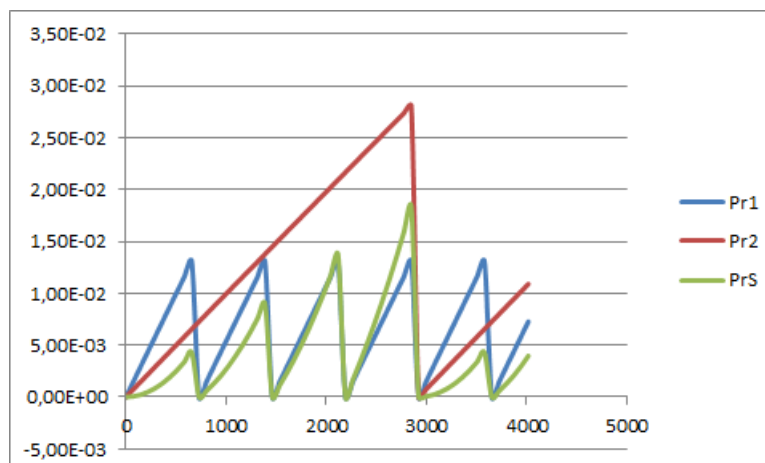
- a constant failure rate:  $\lambda_i$ ,
- a test periodicity  $T_i$ .

How in this context, to calculate the probability of occurrence of an undesired event?

### 2. Notion of maximum risk and average risk

Thereafter, example are based on a system made up of 2 components  $C_1$  and  $C_2$  with failure rate  $\lambda_1$  and  $\lambda_2$  and inspection periods such as  $T_2 = n.T_1$  ( $n$  is an integer greater than 0).

It is possible to represent the failure probability of the system  $S$  ( $PrS$ ) to  $t = 0$  from  $t > T_2$  ( $t$  is the time) with the following curve (on this graph  $n = 4$ ).



There is a divisor factor on Pr1 (Probability of failure of component C<sub>1</sub>) and Pr2 (Probability of failure of component C<sub>2</sub>) in order to show the 3 curves on the same graphic.

The green curve represents the immediate risk that the system is down. The maximum is reached at time  $t = T_2$ . This time corresponds at the last moment of the last flight before the inspection of components C<sub>1</sub> and C<sub>2</sub>. It adequately represents the maximum risk of the system S.

In case of hidden failures or dormant failures, the notion of maximum risk can be too strongly penalizing.

It is authorized, within the standard ARP-4761 and under certain conditions, to calculate an average risk instead of the maximum risk.

### 3. Calculation of the average risk for 2 hidden failures

Mathematically, this average risk can be calculated using an integral calculation from 0 to  $T_2$  of PrS.

Compared with the previous curve, the average risk can be obtained from the following equation:

$$Q_{avg} = \frac{1}{T_2} \left[ \int_0^{T_1} ((1 - \exp^{-\lambda_1 t}) \cdot (1 - \exp^{-\lambda_2 (0.T_1 + t)})) \cdot dt \right. \\ \left. + \int_0^{T_1} ((1 - \exp^{-\lambda_1 t}) \cdot (1 - \exp^{-\lambda_2 (1.T_1 + t)})) \cdot dt \right. \\ \left. + \int_0^{T_1} ((1 - \exp^{-\lambda_1 t}) \cdot (1 - \exp^{-\lambda_2 (2.T_1 + t)})) \cdot dt \right. \\ \left. + \int_0^{T_1} ((1 - \exp^{-\lambda_1 t}) \cdot (1 - \exp^{-\lambda_2 (3.T_1 + t)})) \cdot dt \right]$$

This equation can be to simplify by considering that products  $\lambda_1.T_1$  and  $\lambda_2.T_2$  are very small front 1 ( $\lambda_1.T_1 \ll 1$ ,  $\lambda_2.T_2 \ll 1$ ) and to generalize for n different to 4:

$$Q_{avg} = \frac{1}{T_2} \left[ \int_0^{T_1} (\lambda_1 \cdot t \cdot \lambda_2 \cdot (0.T_1 + t)) \cdot dt \right. \\ \left. + \int_0^{T_1} (\lambda_1 \cdot t \cdot \lambda_2 \cdot (1.T_1 + t)) \cdot dt \right. \\ \left. + \int_0^{T_1} (\lambda_1 \cdot t \cdot \lambda_2 \cdot (2.T_1 + t)) \cdot dt \right. \\ \left. + \int_0^{T_1} (\lambda_1 \cdot t \cdot \lambda_2 \cdot (3.T_1 + t)) \cdot dt \right] = \frac{1}{T_2} \sum_{p=1}^n \left[ \int_0^{T_1} (\lambda_1 \cdot t \cdot \lambda_2 \cdot ((p-1).T_1 + t)) \cdot dt \right]$$

Knowing that the sum of n consecutive terms of an arithmetic progression is worth n times the half a sum of the first and the last term and that  $T_2 = n.T_1$ , we obtain (by developing and by simplifying the formula):

$$Q_{avg} = \frac{\lambda_1 \cdot \lambda_2}{2 \cdot T_1} \left[ \int_0^{T_1} t^2 \cdot dt + \int_0^{T_1} (t^2 \cdot dt + (T_2 - T_1) \cdot t \cdot dt) \right] = \frac{\lambda_1 \cdot \lambda_2 \cdot (3.T_2 + T_1)}{12}$$

The formula proposed in this appendix corresponds to a simplified formula which is valid only for the following hypotheses:  
 $\lambda_1.T_1 \ll 1$ ,  $\lambda_2.T_2 \ll 1$ .

There is in Cecilia-Workshop algorithms which calculate this average risk using not simplified formula.

Starting with the same principle, mathematical formulae were developed to calculate the average risk for the conjunction of 1, 2 or 3 components in hidden failure. Beyond 3 components, the approach risks maximum was kept.

## 4. Probability calculation within Cecilia-Workshop

Within Cecilia-Workshop, user associates with the terminal events:

- Either an inspection period  $T$  and a failure rate  $\lambda$  (via an exponential law).

In this case, it will be possible to take into account for this event an average risk or a maximum risk.

The maximum risk, for a flight duration or mission time  $t$ , consists in associating with this event the following probability:

$$Q(t) = \begin{cases} 1 - \exp^{-\lambda.T} & \text{if } T \geq t \\ 1 - \exp^{-\lambda.t} & \text{if } T < t \end{cases}$$

If the flight duration is upper to the inspection period, then the calculation is performed on the flight duration.

- Either an exposition time  $X$  and a failure rate  $\lambda$  (via an exponential law).

In this case, the maximum risk can be considered. For a flight duration or a mission time  $t$ , probability is calculated using the following formula:

$$Q(t) = \begin{cases} 1 - \exp^{-\lambda.X} & \text{if } X \leq t \\ 1 - \exp^{-\lambda.t} & \text{if } X > t \end{cases}$$

If the flight duration is lower to the exposition time, then the calculation is performed on the flight duration.

- Or a probability law

In this case also, only the maximum risk is considered.

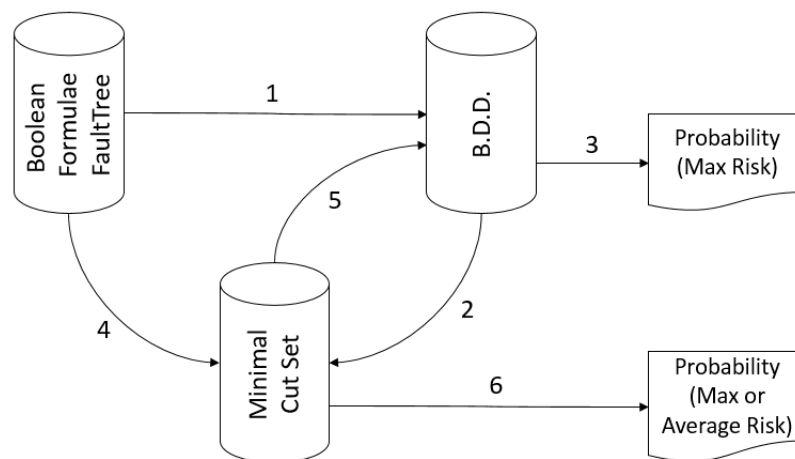
The probability associated in the flight duration  $t$  depends on the probability law of the event. These laws are described within the Appendix A, *Laws and uncertainties*.

For probability calculation (As well as for the calculations of minimal cut sets) for the undesired event (or top event), Cecilia-Workshop uses a library B.D.D (Binary Decision Diagram).

By means of this library, Cecilia-Workshop can:

1. To create a data structure (called exactly B.D.D.) which codes the Boolean formula corresponding to the fault tree seized.
2. To calculate, from the B.D.D. (the data structure created previously), the minimal cut sets of the fault tree.
3. To calculate, from the B.D.D., the exact probability of the undesired event according to the probability of the terminal events.
4. To calculate directly the minimal cut sets limited at the order given by the fault tree.
5. To create the B.D.D. from the minimal cut sets limited at the order given.
6. To calculate from the minimal cut sets the probability of the undesired event (average risk or maximum risk).

We find all these functionalities in the diagram below:



Cecilia-Workshop proposes several methods to calculate the top event probability (see Section 5.1.4, “Minimal cut set calculations (MCS)”).

- **Max Risk Approach (via BDD):**

This option consists to perform a probability calculation from the B.D.D.. This B.D.D. is obtained either directly from the Boolean formula (way 1), or indirectly from the minimal cuts sets for a calculation limited at an order given (way 4, then 5).

At each basic event is associated an occurrence probability. For the events defines by an inspection period, the formula of the maximum risk is used to calculate this occurrence probability.

- **Using sum of minimal cuts:**

This option consists to perform a probability calculation from the minimal cuts. These minimal cuts are obtained either via the way 1 then 2, or via the way 4 as the user limits or not the given order for his calculation.

The probability associated with each terminal event is realized in the same way as the previous method. The probability of each minimal cuts is calculated by making the product of the probability of the different events which compose the minimal cut. The probability of the top event is calculated by making the sum of the probability of minimal cuts. This calculation is a pessimistic approximation of the probability of the top event (Theorem of Poincaré limited to the order 1).

In the same way as the previous method, the result is thus considered as an approach in maximum risk.

- **Average risk approach:**

This option also consists to perform a calculation of probability from the minimal cuts. The difference compared with the previous option consists to calculate the probability of each minimal cut differently.

For a given minimal cut set, the events which compose it are split into 2 categories:

- The events of type average risk: only the events with an inspection period can be in this category.
- The events of type maximum risk: by default, all the events with no inspection period are for type maximum risk.

The probability of the conjunction of the events of type average risks is calculated from the developed formulae presented previously. At this average risk is multiplied the product of the probability of the events of type maximum risk.

- **Average risk approach for critical system:**

This last option is similar to the previous one in a notable difference: for a critical system, a minimal cut cannot consist that of hidden failures. Because if it was the case, the system would already down before taking off!

If a minimal cut consists only of event of type average risk, then one of the events (the one who has the smallest period inspection) is chosen to associate it a maximum risk (notion of obvious failures), the other events will be classically treated as average risk.

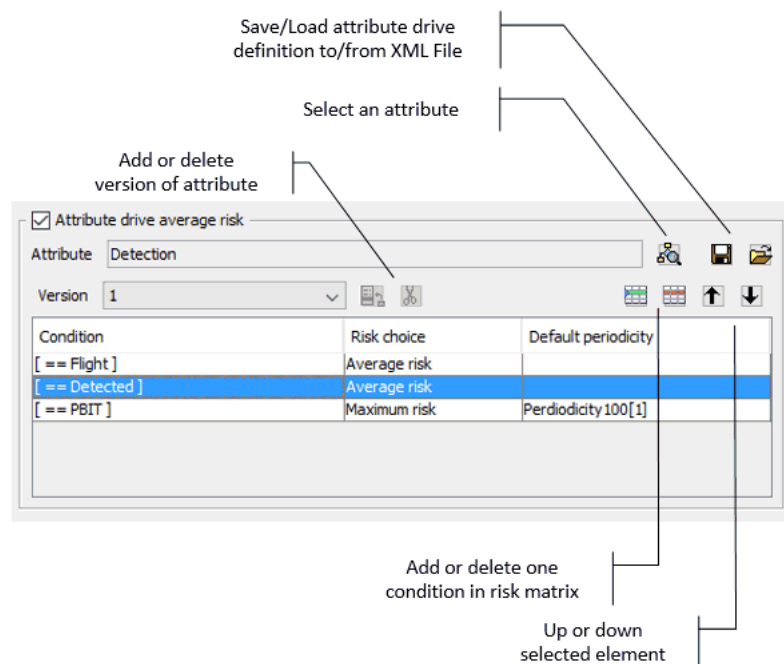
## 5. Risks matrixes defines using an attribute

In the probability calculation by an approach of type average risk, it was specified that only the events with inspection period can be events of type average risk. The users did not wish forcing that an average risk is systematically allocated to all the events with an inspection period. It is the reason which urged to be able to pilot the calculation of average risk using an attribute.

The main idea consists to associate each terminal event at an attribute (generally of enumerated type) which the value specifies how it is taking into account for the average risk calculation. This matrix will also allow to verify the coherence between the different elements seized by the user.

The risk matrix is defined in the project properties. Only an administrator can modify this risks matrix because it affects the way of calculating the average risk for all the trees of all the systems of the same project.

To modify (or show for a user non administrator), it is necessary to go to the project properties in the tab **Compute**.



Check box permits to specify that administrator wants to use an attribute to manage the average risk approach. An attribute (As well as its version for the attributes with several versions) must be selected. For each possible value of the attribute (in the case of an enumerated attribute), it will be necessary to specify the type of risk (**By default** , **Average risk** or **Maximum risk** ) and a periodicity by default.

The rules to check, which applies when the user asks for a calculation with an approach of average risk, are the following:

- From the moment where an attribute pilots the approach average risk, each event must have been associated with the selected attribute.
- Each event has to answer at least at one condition of the risks matrix (the risks matrix being read in the order, the selected condition is the first one met).
- Each event must be compatible with the **Risk choice** and the **Default periodicity** define by the selected condition.
- If **Risk choice** = **Default periodicity** then the event cannot be have a inspection period, else the event has to forcing have an inspection period.
- If **Default periodicity** was defined, then the inspection period of the event must be equal to the value of **Default periodicity**.

If at least one of these checks is not validated, an error message displays.

This matrix thus allows to differentiate among the events with inspection period those which will be handled by an approach of average risk of those which will be handled by an approach of maximum risk.

## C. Importance factors

This section reviews the different importance factors available in Cecilia-Workshop. Importance factors are for evaluating the role of a component in the reliability or availability of a system. The word component as used here should be taken in the broad sense. A component can be described by any Boolean function. The calculations run by Aralia are therefore not limited to just basic events or groups of basic events.

Throughout the rest of this section, the following convention applies:

- S the function describing system failures;
- e the function describing component failures.

And also:

- $\bar{e}$  the negation of the function e;
- $p(F)$  the probability of the function F;
- $p(F|G)$  the conditional probability of F given G (where G is any function).

### 1. Conditional probabilities

The first of the importance factors available is of course conditional probability  $CPr(S,e)$ , that is the probability of the system S failing, given that component e has failed.

$$CPr(S,e) = p(S|e) = \frac{p(S \wedge e)}{p(e)}$$

### 2. Marginal Importance Factor (MIF)

The marginal importance factor  $MIF(S,e)$  was introduced by Birnbaum. It is the rate at which the availability of the system S increases when availability of the component e increases.

$$MIF(S,e) = \frac{\partial p(S)}{\partial p(e)}$$

The marginal importance factor may also be interpreted as the probability of system S being in a state such that if, all other things being equal, the component e has failed, then the system fails, if not it works.

Effectively, we write:

$$MIF(S,e) = \frac{\partial p(S)}{\partial p(e)} = \frac{\partial [p(e) \cdot [p(S|e) - p(S|\bar{e})] + p(S|\bar{e})]}{\partial [p(e)]} = p(S|e) - p(S|\bar{e})$$

### 3. Critical Importance Factor

The critical importance factor  $CIF(S,e)$  which was apparently introduced by Lambert, is for evaluating the relative importance of the components. It is, by definition, highly dependent on the marginal importance factor, since:

$$CIF(S,e) = \frac{p(e)}{p(S)} \times MIF(S,e)$$

## 4. Diagnostic importance factor (DIF)

The diagnostic importance factor  $DIF(S,e)$ , which was introduced by Vesely and Fussel, is the probability of component  $e$  having caused the failure of the system  $S$ , given that system  $S$  has effectively failed. Its definition is as follows:

$$DIF(S,e) = p(e|S)$$

## 5. Risk achievement worth

The risk achievement worth  $RAW(S,e)$  or risk increase factor is, an indicator of the importance of maintaining the level of reliability of the component  $e$  in maintaining the reliability of the system  $S$ . It should be interpreted with extreme caution as it is only a rough approximation.

$$RAW(S,e) = \frac{p(S|e)}{p(S)}$$

## 6. Risk Reduction Worth (RRW)

The risk reduction worth  $RRW(S,e)$  or risk decrease factor represents the maximum decrease in risk which can be expected by improving the reliability of a component  $e$ . It is important for identifying those components whose improved reliability is most likely to increase that of the system  $S$ .

$$RRW(S,e) = \frac{p(S)}{p(S|e)}$$

## D. Minimal Cut Set utilities

This chapter deals with all Minimal Cut Set utilities (translator, comparator, ...) of Cecilia-Workshop which enable easy use.

### 1. Translation of MCS into *Aralia* equation

Translation of minimal cuts set into *Aralia* Boolean formula

#### 1.1. Introduction

MCS is abbreviation of Minimal Cut Set.

The *Aralia* is reference software in Safety community. This enables processing of Boolean formulae with probability.

Each minimal cut is translated into boolean conjunction (AND gate). The set of minimal cuts is considered as a disjunction (OR gate).

In order to ease reading by *Aralia* or other software that can import MCS, minimal cuts are checked in fixe size packets.

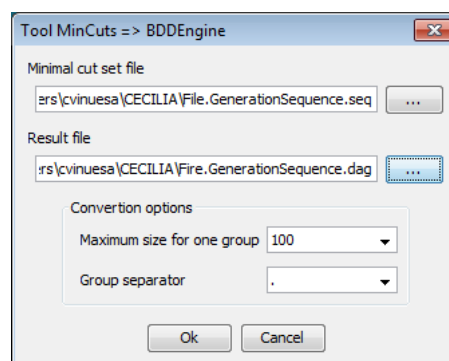
A separator character is used in order to create variables for each minimal cut and for each packet of minimal cuts.

#### 1.2. Launching of translation



In order to launch translation, use the **Tools MinCuts => Aralia** command.

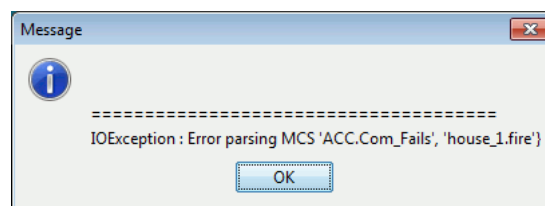
The following window enables to select an input file containing minimal cuts, and a result file.



The input file containing minimal cuts must be in one known format described in previous chapter.

Options enable to define maximum size of each group (packet) of minimal cuts and separator character.

If an issue occurs during input file reading, an error message is displayed.



#### 1.3. Command line

This utility can be launched without user interface with the following command line:



```
<JavaPath>/java.exe -classpath <ToolsPath>/plugins/MCSUtil.jar
dassault.mcslib.McsToAralia <input-file> <output-file>
```

where

- <JavaPath> : Java installation directory
- <ToolsPath> : Cecilia-Workshop installation directory
- <input-file> : input file containing minimal cuts
- <output-file> : result file

Default separator is the dot '.' and group size is set to 100.

## 2. Compare two set of minimal cuts

### 2.1. Introduction

MCS is abbreviation of Minimal Cut Set.

diff is the name of historical program that compare 2 files.

Diff MCS allows comparison of 2 minimal cut set.

A simple file comparison doesn't allow efficient MCS comparison, for two reasons:

- There are many different save/display formats.
- Order inside MCS is not unique (no order, lexicographic order, increasing probability, ... )

This utility has been improved in last version, it can also compare Minimal Sequence Set. Files format specify if data must deal with minimal cuts or minimal sequences.

### 2.2. Different formats

Sets of minimal cuts can come from different sources.

This utility can read following formats:

- Cecilia ARBOR - Export of minimal cuts into Text format [ \*.txt ]

#### Example D.1.

```
Document produit par CECILIA ARBOR
COUPES MINIMALES
<DATE>07/03/06
...
COUPES MINIMALES D'ORDRE ( 1 2 )
COUPES MINIMALES
Numéro Occurrence Nom Commentaire Loi Périodicité Temps d'exposition Propriétés Attributs
1 0.000E00 bt_2.panne exponential 1.000E-06 - - U
    equ1.cst10.panne exponential 1.000E-06 - - U
2 0.000E00 equ1.cst10.panne exponential 1.000E-06 - - U
    equ2.cst10.panne exponential 1.000E-06 - - U
...
```

- Cecilia ARBOR - Export of minimal cuts into XML format [ \*.xml ]

#### Example D.2.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE EXPORT SYSTEM "file:///R:\Site\App\Arbor\V3.7\data\properties.dtd">
<EXPORT>
```

```

<MINIMAL_CUT> <IDENTIFICATION_LIST .../>
<PROJECT> <IDENTIFICATION .../>
  <SYSTEM> <IDENTIFICATION .../>
    <TREE CREATION_DATE="06-Mar-2006 19:16:45"> <IDENTIFICATION .../>
      <CUT>
        <EVENT NAME="bt_2.panne">
          <EXPONENTIAL> <VALUE_PARAMETER VALUE="1.000E-06"></VALUE_PARAMETER>
        </EXPONENTIAL>
      </EVENT>
      <EVENT NAME="equ1.cst10.panne">
        <EXPONENTIAL> <VALUE_PARAMETER VALUE="1.000E-06"></VALUE_PARAMETER>
      </EXPONENTIAL>
    </EVENT>
  </CUT>
  <CUT>
    <EVENT NAME="equ1.cst10.panne">
      <EXPONENTIAL> <VALUE_PARAMETER VALUE="1.000E-06"></VALUE_PARAMETER>
    </EXPONENTIAL>
  </EVENT>
  <EVENT NAME="equ2.cst10.panne">
    <EXPONENTIAL> <VALUE_PARAMETER VALUE="1.000E-06"></VALUE_PARAMETER>
  </EXPONENTIAL>
</EVENT>
</CUT>
</TREE>
</SYSTEM>
</PROJECT>
</MINIMAL_CUT>
</EXPORT>

```

- *Aralia* - Command result display products ... [ \*.mcs | \*.seq ]

With MCS format, **Sequences generation (Generic)** products a result file in same syntax.

### Example D.3.

```

products(MRS('cst12.s.false')) =
{'equ2.cst10.panne', 'bt_2.panne', 'equ1.cst10.panne'}
{'equ2.cst10.panne', 'equ1.cst10.panne'}
{'bt_2.panne', 'equ1.cst10.panne'}
end

```

- **Sequences generation (Generic)** with XML format for result file [ \*.xml ]

### Example D.4.

```

<?xml version='1.0' encoding='iso-8859-15'?>
<seqgen>
  <define>
    <target name="cst12.s" value="false">
      <param name="resultset" value="minseqs"/>
    ...
  </target>
</define>
<abstract>
/* Order of products :
  ...
*/
</abstract>
<result>
  <seq><tr id="1" evt="equ2.cst10.panne"/>
    <tr id="16" evt="bt_2.panne"/>
    <tr id="19" evt="equ1.cst10.panne"/></seq>
  <seq><tr id="1" evt="equ2.cst10.panne"/>
    <tr id="19" evt="equ1.cst10.panne"/></seq>
  <seq><tr id="16" evt="bt_2.panne"/>
    <tr id="19" evt="equ1.cst10.panne"/></seq>
</result>
<model>

```

```
<flow name="cstl2.s" domain="bool" orientation="out"></flow>
<state name="cstl2.stt" domain="{no_ok, ok}">
  <init value="ok"/></state>
...
<event name="equ2.cstl0.panne">
  <law value="exponential(1.0E-4)" aralia="exponential 1.0E-4" moca="exp 1.0E-4"/></
event>
...
</model>
</seqgen>
```

The `resultset` parameter in XML file enables to distinguish between minimal cuts and minimal sequences.

- The *alta-seq* sequences generation with control automaton creates XML file compatible with this utility. [ \*.xml ]

### Example D.5.

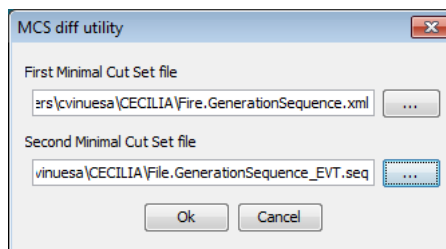
```
<?xml version="1.0"?>
<!--
#sequences: 3
running time: ...
-->
<sequences name="sequences.main.automaton1" >
  <sequence >
    <transition event="equ2.cstl0.panne" />
    <transition event="bt_2.panne" />
    <transition event="equ1.cstl0.panne" />
  </sequence>
  <sequence >
    <transition event="equ2.cstl0.panne" />
    <transition event="equ1.cstl0.panne" />
  </sequence>
  <sequence >
    <transition event="bt_2.panne" />
    <transition event="equ1.cstl0.panne" />
  </sequence>
</sequences>
```

## 2.3. Launch a comparison



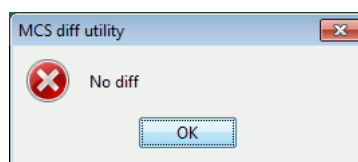
In order to launch a comparison between two sets of minimal cuts, use the **Diff. MCS** command.

The following window enables to select two files containing MCS to be compared.

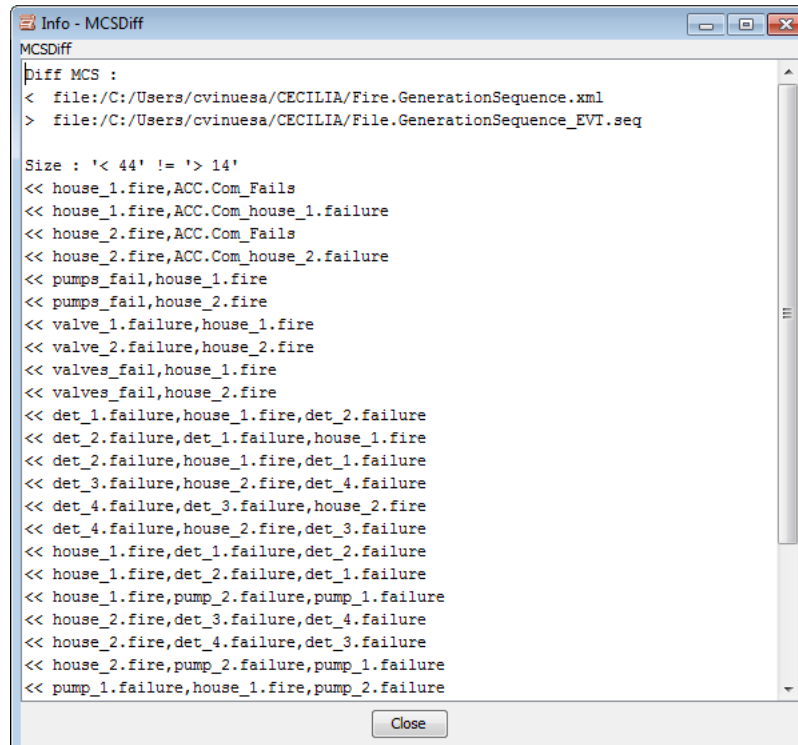


## 2.4. Comparison result

If the two sets are equivalent, the following message is displayed:



If they are not equivalent, a window displays differences:



Cuts that are in the first file but not in the second one are noticed with a << sign.

Cuts that are in the second file but not in the first one are noticed with a >> sign.

## 2.5. Command line

This utility can be launched without user interface with the following command line:

```
<JavaPath>/java.exe -classpath <ToolsPath>/plugins/MCSUtil.jar
dassault.mcslib.McsDiff <file1> <file2>
```

where

- <JavaPath> : Java installation directory
- <ToolsPath> : Cecilia-Workshop installation directory
- <file1> <file2> : the 2 files to be compared

## ***E. Language AltaRica***

### **1. Introduction**

#### ***1.1. Objectives of this document***

AltaRica Extended language is reference language of Cecilia-Workshop.

It comes from confluence works on different AltaRica dialects from 2000 to 2003 :

- AltaRica Dassault (reference language of Cécilia OCAS 2.7 Béta) [4]
- AltaRica DataFlow (reference language of Alta-X tools from ARBoost Technologie) [3]
- Original AltaRica and its extensions from LaBRI (reference language or LABRI tools) [1,2]

The objective of this language is to enables complex grammatical constructions, like common cause failures or structured flows in order to be compliable in elementary construction. Then, it's possible to add new functions without modifying computation core which can still use a language that doesn't handle this structure.

Two languages have been defined:

- A minimalist language that doesn't handle structured flows, operators, common cause failures synchronizations, and some specificities of AltaRica Dassault language:

This language relies essentially on current version of Altarica DataFlow language. The main difference with this language is syntax of assertions which is closer than the one of original Altiraca language.

- AltaRica Extended mainly relies on functionalities expected of Cecilia-Workshop:

So, it has been built from AltaRica Dassault language and has been improved with AltaRica DataFlow notions (init and extern clause, parting component and equipment at language level).

In addition of original language functionalities, these two languages handle arithmetic operators (+, -, \*, /) and "Integer" and "Real" domains (even if tools using minimal language don't take it into account).

A translator enables to translate from AltaRica Extended language to minimal language.

This document deals with grammar rules of AltaRica Extended language.

#### ***1.2. References***

[1] : Projet Altarica Phase III - Le Langage Altarica - Manuel de Référence, par G. Point, 01/06/2000, réf. IXI-AltaRica/03/T01/DT00-V0

[2] : AltaRica – Manuel méthodique, par A. Arnold, G. Point, A. Griffault, A. Rauzy, 05/10/2001

[3] : The AltaRica Data-Flow Language – Syntax, par A. Rauzy, 09/10/2002, réf. ARBoost Technologies/AltaRica/NT02-1, version 7

[4] : Atelier Cécilia OCAS version 2.7 Béta, couplé au moteur de calcul JMoteur version 4.59

### **2. Syntax of AltaRica Extended language**

#### ***2.1. Lexical conventions***

##### ***2.1.1. Lexemes***

Lexemes of Altarica are identifiers, keywords, integer or real constants, operators, and separators.

White-spaces, tabulations, end-line characters and comments are ignored unless they separate lexemes.

### 2.1.2. Comments

Two type of comments are allowed:

1. Characters `/*` mark the beginning of a multi-line comment which must ends with `*/`. Comments can't be imbricated.

```
/*
 * This is comment over many lines
 */
```

2. Characters `//` mark the beginning of a one-line comment. Comment ends with End-Of-Line or End-Of-File character.

```
// this is one-line comment
// This is another comment
```

### 2.1.3. Identifiers

An identifier is a sequence of letters, digits or `'_'`. A simple identifier always begins with a letter and can have an unlimited lenght. Uppercase and lowercase are differentiated.

Example : Engine, Engine\_9, engine\_9 (The last two are different identifiers).

```
<identifier> ::= '[a-z][a-zA-Z0-9_-]*'
```

### 2.1.4. Keywords

Following Identifiers are reserved keyword:

and	assert	bool	case	cnuf	const
domain	edon	else	event	extern	false
float	flow	func	if	imply	in
init	int	inverse	knil	link	local
max	min	node	not	or	out
private	state	sub	sync	term	then
trans	true				

### 2.1.5. Constants

There are different types of constants, Integer constants, Real constants, enumerated constants and Boolean constants:

1. An integer constant is :
  - either 0 ;
  - either a succession of digits which doesn't begin with 0.

```
<integer> ::= '0 | ([1-9][0-9]*)'
```

2. A real constant

```
<float> ::= '[0-9]*.[0-9]+([eE][+-]?[0-9]+)?'
```

3. A Boolean constant in one of the two key-word: true or false.
4. An enumerated constant is an identifier declared beforehand in an enumeration domain (cf. Section 2.5.3, “Enumeration of symbolic constants”).

### 2.1.6. Strings

A string is a sequence beginning and ending with double-quote `'"`.

```
<string> ::= '"[^"]+"'
```

## 2.2. Structure of an AltaRica file

An altarica description is a set of declarations of constants, domains or component models. These declarations can't be imbricated, the range of objects so declared is global from the declaration point.

```
<altarica-description>
  ::= <global-declaration-list>
<global-declaration-list>
  ::= <global-declaration> ((';'?)? <global-declaration>)*
  ::=
<global-declaration>
  ::= <constant-declaration>
  ::= <domain-declaration>
  ::= <function-declaration>
  ::= <model-declaration>
```

## 2.3. Path in hierarchy

State variables, flow variables or events of a node can be specified by a path in hierarchy of AltaRica models. Such a path is a succession of identifier separate with points (.).

```
<hierarchy-path> ::= <identifier> ('.' <hierarchy-path>)*
```

Depending on context, a path can be construed as either a variable identifier, or an event identifier, or an enumerated constant identifier, or a declared constant identifier (cf. 0). In this condition, such an identifier cannot be used for a declared constant, a variable or an enumerated constant

## 2.4. Expressions

This section deals with syntax of AltaRica expressions that can be found in assertions, guards or transitions of a node.

Expressions can be regrouped into different categories depending on the returned type: Boolean (<bool-expr>), numerical (<num-expr>), symbolic (<symb-expr>) or structured (<struct-expr>). Expression type is not syntactically verified as made in AltaRica DataFlow language, but it's made after, during lexical verification of data.

AltaRica Extended syntax takes operators priority into account (like former version of AltaRica Dassault or AltaRica LaBRI).

```
<bool-expr>   ::= <expression>
<num-expr>    ::= <expression>
<symb-expr>   ::= <expression>
<struct-expr> ::= <expression>
```

### 2.4.1. "if ... then ... else ..." and "(if ... then ...)" expressions

Expressions *If-then*, without *Else* has been introduced in AltaRica Dassault. These expressions are equivalent to implication. Their syntaxes have been kept for compatibility reasons and user habitude. But they are translated in imply operator => in low level language. Brackets must surround expressions.

Moreover, in former version of AltaRica language, *If-Then-Else* expression was surrounded with brackets in order to facilitate understanding. Now, brackets are no more mandatory.

```
<expression>
  ::= <match-expr>
  ::= <unmatch-expr>
<match-expr>
  ::= <other-expr>
  ::= if <bool-expr> then <match-expr> else <match-expr>
<unmatch-expr>
  ::= '(' if <bool-expr> then <bool-expr> ')'
  ::= if <bool-expr> then <match-expr> else <unmatch-expr>
<other-expr>
  ::= <or-expression>
```

For an *If-Then-Else* expression, the first sub-expression must be Boolean (predicate); it is called condition of operation. The two following sub-expressions are *Then* and *Else* of *If-Then-Else*.

### 2.4.2. Atomic expressions

Atomic expressions are either access path of variables or constants or expressions between brackets, or functions of the language (min, max, cardinality '#') or defined by user, or case operator of AltaRica DataFlow.

```
<atomic-expression>
  ::= <integer>
  ::= <float>
  ::= true
  ::= false
  ::= <id-variable>
  ::= '(' <expression> ')'
  ::= min '(' <num-expr> ',' <num-expr> ')'
  ::= max '(' <num-expr> ',' <num-expr> ')'
  ::= '#' '(' <bool-expr> ',' <bool-expr> ')'
  ::= <identifier> '(' <expression> ',' <expression> ')'
  ::= <identifier> '(' ')'
  ::= case '{'
      (<bool-expr> ':' <expression> ',' )*
      else <expression>
    '}'
<id-variable>
  ::= <hierarchy-path> '^' <identifier>
  ::= <hierarchy-path>
```

In order to access to a variable of a sub-node, point (.) must be used as separator of the path. In order to access to a field of a structured flow, circumflex accent (^) is used and followed by field name.

### 2.4.3. Unary operators

Unary operations are evaluated from right to left.

```
<unary-expression>
  ::= - <unary-expression>
  ::= <operator-not> <unary-expression>
  ::= <atomic-expression>
<operator-not>
  ::= not
  ::= '~'
```

Operand of arithmetic negation – must be from arithmetic type (integer or real) and the result of this negation is the opposite of its operand.

Boolean negation is described thanks to the two following identifiers: not word or tilde (~). Operand of a Boolean expression must be Boolean.

### 2.4.4. Multiplying Operators

Multiplying operators \*, / and % are evaluated from left to right. Their operands must be arithmetic. \* is multiplication, / is division and % is remainder of integer division (in this case, operands must be integer).

```
<multiplicative-expression>
  ::= <multiplicative-expression> '*' <unary-expression>
  ::= <multiplicative-expression> '/' <unary-expression>
  ::= <multiplicative-expression> '%' <unary-expression>
  ::= <unary-expression>
```

### 2.4.5. Additive operators

Additive operators + and – are evaluated from left to right. Operands of these operators must be arithmetic. + operator gives the sum of values of operands. + operator gives the values of the first operand minus the value of the second one.



```

<additive-expression>
    ::= <additive-expression> '+' <multiplicative-expression>
    ::= <additive-expression> '-' <multiplicative-expression>
    ::= <multiplicative-expression>

```

### 2.4.6. Relational operators

Relational operators are valuated from left to right. Operands must be arithmetic because - in opposition to some programming language - neither enumerated constants nor Boolean are assimilated with integer.

< (less than), > (greater than), <= (less than or equal to) et >= (greater than or equal to) give the value `false` if the specified comparison is not verified with the operands; otherwise the value of expression is `true`.

```

<relational-expression>
    ::= <relational-expression> '<' <additive-expression>
    ::= <relational-expression> '>' <additive-expression>
    ::= <relational-expression> '<=' <additive-expression>
    ::= <relational-expression> '>=' <additive-expression>
    ::= <additive-expression>

```

### 2.4.7. Equality operators

These operators are = (equal to), != or # (different from), imply or => (imply).

Operands of = and != are Boolean, arithmetic or enumerates. Operands of = operator can also be structured. The value of these expressions is `true` if the equality is verified by operands, and `false` otherwise. In Boolean case, equality matches equivalence of operands, difference matches *Exclusive or* (XOR)

Operands of imply and => must be Boolean. These operateur matches logical implication.

```

<equality-expression>
    ::= <relational-expression> '=' <relational-expression>
    ::= <relational-expression> <operator-neq> <relational-expression>
    ::= <relational-expression> <operator-imply> <relational-expression>
    ::= <relational-expression>
<operator-neq>
    ::= '!='
    ::= '#'
<operator-imply>
    ::= imply
    ::= '=>'

```

### 2.4.8. Logical AND operator

Logical AND operator has got two identifiers: & and and. Its operands must be Boolean. It is valuated from left to right. The value is `true` if the two operands are `true`, `false` otherwise.

```

<and-expression>
    ::= <and-expression> <operator-and> <equality-expression>
    ::= <equality-expression>
<operator-and>
    ::= and
    ::= '&'

```

### 2.4.9. Logical OR operator

Logical OR operator has got two identifiers: | and or. Its operands must be Boolean. It is valuated from left to right. The value is `false` if the two operands are `false`, `true` otherwise.

```

<or-expression>
    ::= <or-expression> <operator-or> <and-expression>
    ::= <and-expression>
<operator-or>
    ::= or

```

```
 ::= ' | '
```

## 2.5. Domains statement

### 2.5.1. General points

Language has predefined domains: `bool` (for Boolean variables), `int` (for integer variables), `float` (for real variables).

New domain can be created using following syntax:

```
<domain-declaration>
 ::= domain <identifier> = <domain-definition>
<domain-definition>
 ::= <link-domain>
 ::= <domain>
<domain>
 ::= <range-domain>
 ::= <enumeration-domain>
 ::= <predefined-domain>
 ::= <identifier>
<predefined-domain>
 ::= bool
 ::= int
 ::= float
```

An existing domain name cannot be redefined. The declared domain can be used in the rest of description if a domain of value is required. Domain alias can be created associating a domain name to another domain name (predefined or declared).

Three domains construction are allowed: relative integer interval, symbolic constant enumeration, and structured domain.

### 2.5.2. Integer interval

Integer interval are specified between square bracket (`[ , ]`), bounds are separate with comma. Bounds are integer. The value of left bound must be less than the value of right bound.

```
<range-domain>
 ::= '[' <integer> ',' <integer> ']'
```

### 2.5.3. Enumeration of symbolic constants

An enumeration is a list of identifiers which are between brace and separate with commas (`{...}`). These identifiers must not be associated with a global constant name.

```
<enumeration-domain>
 ::= '{' <identifier-list> '}'
```

### 2.5.4. Structured domain

Structured domain replaces type def of AltaRica Dassault language. They are essentially used to simplify entry of connection beam between components (bus, serial cable, informations transmission, ...). Only flow variables (in or out) can be declared with structured domain (private or state variables can't).

The `flow` clause allows to define a set of structured domain fields. Each field is defined with an identifier and a domain (which can't be structured).

A structured domain defines two plugs: the in plug and the out plug. Each connection plug is composed of `n` fields defined with `flow` clause. Usually, a field has the same orientation as its reference plug. The `inverse` clause defines plug fields having reversed orientation.

The `assert` clause defines equality relations between in plugin and out plug. In order to simplify understanding, equality relations are oriented and defined with assignment.

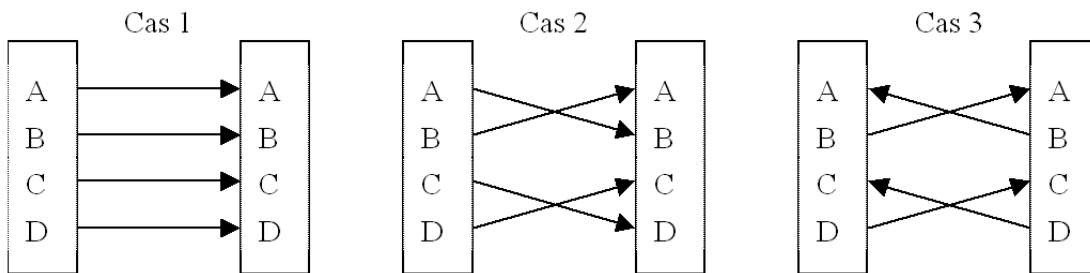
```
<link-domain>
```

```

::= link
    flow <link-flow-list> (';')?
    ( inverse <inverse-list> (';')? )?
    assert <connect-list> (';')?
    knil
<link-flow-list>
    ::= <link-flow-decl> (';' <link-flow-decl>)*
<link-flow-decl>
    ::= <identifier-list> ':' <domain>
<inverse-list>
    ::= <id-flow-link> (';' <id-flow-link>)*
<connect-list>
    ::= <connect-decl> (';' <connect-decl>)*
<connect-decl>
    ::= <id-flow-link> ':= ' <id-flow-link>
<id-flow-link>
    ::= in '^' <identifier>
    ::= out '^' <identifier>

```

Examples :



```

domain cas1 = link
    flow A,B,C,D : int ;
    assert
        in^A := out^A;
        in^B := out^B;
        in^C := out^C;
        in^D := out^D;
    knil;

```

```

domain cas2 = link
    flow A,B,C,D : int ;
    assert
        in^A := out^B;
        in^B := out^A;
        in^C := out^D;
        in^D := out^C;
    knil;

```

```

domain cas3 = link
    flow A,B,C,D : int ;
    inverse out^A; out^C;
    in^B; in^D;
    assert
        in^A := out^B;
        out^A := in^B;
        in^C := out^D;
        out^C := in^D;
    knil;

```

## 2.6. Declaration of functions or operators

Operators (or functions : func) enable definition of output variable functions of variables in parameters.

Description of a function is close to description of a component (node). The difference is function doesn't describe a behavior. A function is a component model having an output flow, some input flows and assertions enabling definition of output flow functions of input flows.

```

<function-decl>
    ::= func <identifier>
        <function-elem>
        <function-body>
    cnuf
<function-elem>
    ::= <flow-cls> <function-elem>
    ::=
<function-body>
    ::= <assert-cls> <function-body>
    ::=

```

Domains of flow variables can be any domains, even structured domain. In this last case, structured flows having inverse clause are forbidden. Moreover, equality between structured flows takes into account crossing that are defined in assert clause of structured domain.

A function can be used only in expression of model assertions. It cannot be used in guards of assignments. Moreover, recursive function cannot be written.

## 2.7. Declaration of models

A model is declared using the above syntax. Declaration begins with `node` keyword, followed by the model name, a list of fields and ends with `edon`.

```
<model-declaration>
  ::= node <identifier>
      <node-elem>
      <node-body>
      <extern-1st>
  edon
```

The first part enables declaration of data manipulated in this component model, that is to say flow variables (`flow`), state variables (`state`), events (`event`) and nodes (`sub`).

```
<node-elem>
  ::= <node-elem-field> <node-elem>
  ::=
<node-elem-field>
  ::= <flow-cls>
  ::= <state-cls>
  ::= <event-cls>
  ::= <sub-cls>
```

The second part describes behavior and connections between data thanks to assertions (`assert`), transitions (`trans`), synchronizations (`sync`) and initialization (`init`).

```
<node-body>
  ::= <node-body-field> <node-body>
  ::=
<node-body-field>
  ::= <trans-cls>
  ::= <assert-cls>
  ::= <sync-cls>
  ::= <init-cls>
```

The last part (one or many `extern` clauses) enables addition of information that are external to language. It's mainly used for tools.

```
<extern-1st>
  ::= <extern-cls> <extern-1st>
  ::=
```

The `sync` field is allowed only for hierarchical component model, that is to say if at least one sub-node has been declared in `sub` field.

`state` fields and `trans` fields are allowed only if model is not hierarchical.

### 2.7.1. Declaration of flow variables

Flow variables serve as interface with other components.

Declaration of flow variables must respect following syntax.

```
<flow-cls>
  ::= flow <flow-decl-list> (';')?
```

The `flow` keyword is followed by an unempty list of variables declaration separate with semicolon. Declarations list may end with semicolon.

```
<flow-decl-list>
  ::= <flow-decl> ( ';' <flow-decl> ) *
```

Every declaration conforms to following syntax : A list of names of variables, a separator :, a value domain, a second separator : and the flow orientation (input flow : in, output flow : out, local flow or private component: local or private). Variables identifiers must not be followed by a declared constant, an enumerated constant or an already declared variable of the model (in the same or in another variable class).

```
<flow-decl>
  ::= <identifier-list> ':' <domain> ':' <orientation>
<orientation>
  ::= in
  ::= out
  ::= private
  ::= local
```

### 2.7.2. Declaration of state variables

State variables are used to imitate component state. Component changes its state by changing the value of one or many state variables thanks to transitions.

Declaration of state variables must respect following syntax.

```
<state-cls>
  ::= state <state-decl-list> ( ';' ) ?
```

The `state` keyword is followed by an unempty list of variables declaration separate with semicolon. Declarations list may end with semicolon.

```
<state-decl-list>
  ::= <state-decl> ( ';' <state-decl> ) *
```

Every declaration conforms to following syntax : A list of names of variables, a separator : and a value domain. Variables identifiers must not be followed by a declared constant, an enumerated constant or an already declared variable of the model (in the same or in another variable class).

```
<state-decl>
  ::= <identifier-list> ':' <domain>
```

### 2.7.3. Declarations of events

Declarations of events must respect following syntax.

```
<event-cls>
  ::= event <event-decl-list> ( ';' ) ?
<event-decl-list>
  ::= <event-decl> ( ';' <event-decl> ) *
<event-decl>
  ::= <identifier>
```

Declaration of event begins with `event` keyword, followed by an unempty list of event specifications separate with semicolon. Declarations list may end with semicolon.

An event is defined with its name that is to say with an identifier. If an event is declared, a model must contain at least one transition labeled with this event.

### 2.7.4. Declaration of sub-components

Declaration of sub-nodes of an Altarica model begins with the `sub` keyword. For the rest, syntax is the same as the one used for state variables, except the fact that domain is replaced by a name of a previously declared model.

```

<sub-cls>
  ::= sub <sub-list> (';')?
<sub-decl-list>
  ::= <sub-decl> (';' <sub-decl>)*
<sub-decl>
  ::= <identifier-list> ':' <identifier>

```

### 2.7.5. Definition des transitions

Definition of model transitions begins with `trans` keyword. This keyword is followed by an unempty list of transitions specifications separate with semicolon. Declarations list may end with semicolon.

```

<trans-cls>
  ::= trans <trans-list> (';')?
<trans-list>
  ::= <transition> (';' <transition>)*

```

Transition definition always begins with a boolean expression which is the transition guard.

```

<transition>
  ::= <bool-expr> (<transition-target>)+

```

Guard of a transition is followed by an unempty list of actions linked to events.

An action begins with a separator `|` – followed by an unempty list of events previously declared in the model.(cf. 2.8.1) The list ends with `->` separator, followed by a possibly empty list of assignments of variables.

```

<transition-target>
  ::= '|' <event-name-list> '->' <assignment-list>
  ::= '|' <event-name-list> '->'

```

Assignments of variables are separate with commas.

```

<assignment-list>
  ::= <assignment> (',' <assignment>)*

```

An assignment is composed of a state variable identifier, followed by assignment separator `:=` and ends with an expression from the same type as state variable.

```

<assignment>
  ::= <identifier> ':=' <expression>

```

### 2.7.6. Definition of assertions

Definition of component assertions begins with `assert` keyword. This keyword is followed by an unempty list of Boolean expressions separate with semicolon. Declarations list may end with semicolon.

List of assertions is construed as conjunction (logical AND) of assertions of the list.

```

<assert-cls>
  ::= assert <assert-list> (';')?
  ::= assert
<assert-list>
  ::= <bool-expr> (';' <bool-expr>)*

```

### 2.7.7. Definition of synchronizations

Declaration of synchronizations vectors begins with `sync` keyword. Declaration of synchronizations vectors is allowed if model contains at least one sub-component.

```
<sync-cls>
  ::= sync <vector-list> (';')?
```

Declaration of synchronizations vectors consists in an unempty list of vectors separate with semicolon. Declarations list may end with semicolon.

```
<vector-list>
  ::= <vector> (';' <vector>)*
```

Synchronization vector consist in an event vector beginning with < and ending with >. Events can be events of the model, or events of sub-component for hierarchical models, in this case events are prefixed with the name of the sub-component and followed by a dot.

Discussions about synchronizations have shown that there are 3 types of synchronizations:

1. Synchronization like mec : events can only appears at the same time

```
<vector-sync> ::= '<' <hierarchie-path> (';' <hierarchie-path>)+ '>'
```

2. Broadcast : all events that can appear at the same time appear at the same

```
<vector-diff> ::= '<' <hierarchie-path> ('|' <hierarchie-path>)+ '>'
```

3. Common cause : either events appears individually (failure without common cause), or events that can appear at the same time appear at the same time (common cause strictly speaking ~= broadcasting)

```
<vector-ccf> ::= '<' <hierarchie-path> ('?' <hierarchie-path>)+ '>'
```

A vector can be written these ways.

```
<vector>
  ::= <vector-sync>
  ::= <vector-diff>
  ::= <vector-ccf>
```

## 2.7.8. Definition of initialization

The init keyword enables specifications of initialization of model state variables, and initialization of sub-component state variables for hierarchical models.

Declaration of component initialization begins with init keyword. This keyword is followed by an unempty list of state variable assignments separate with semicolon. Declarations list may end with semicolon.

List of assertions is construed as conjunction (logical AND) of assertions of the list.

```
<init-cls>
  ::= init <init-list> (';')?
<init-list>
  ::= <init-def> (';' <init-def>)*
```

An initialization consists of a path to a state variable. If the length of path is 1, then the referenced variable is a variable of the currently defined model, else it is a variable of a son-component of the model. Each variable is associated with a constant expression which must have the same type as that of variable.

```
<init-def>
  ::= <hierarchy-path> ':' <expression>
```

Initialization specified in a model override the ones specified in sub-components (these ones can be considered as value for default initialization).

### 2.7.9. Definition of external directives

The `extern` clause of a model has been introduced in order to give information to tools using AltaRica language.

Declaration of directives begins with `extern` keyword. This keyword is followed by an unempty list of directives separate with semicolon. Declarations list may end with semicolon.

```
<extern-cls>
    ::= extern <extern-list> (';')?
<extern-list>
    ::= <extern-decl> (';' <extern-decl>)*
```

Contrary to previous versions of AltaRica Dassault (and AltaRica LaBRI), the syntax of external clause is specified. It takes and extends the one defined in AltaRica DataFlow language.

```
<extern-decl>
    ::= <identifier> <extern-term> '=' <extern-term>
    ::= <identifier> <extern-term>
<extern-term>
    ::= true
    ::= false
    ::= <integer>
    ::= <float>
    ::= <string>
    ::= <identifier>
    ::= <identifier> '(' <extern-term> (';' <extern-term>)* ')'
    ::= '{' <extern-term> (';' <extern-term>)* '}'
    ::= '<' flow <hierarchy-path> '>'
    ::= '<' state <hierarchy-path> '>'
    ::= '<' event <hierarchy-path> '>'
    ::= '<' sub <hierarchy-path> '>'
    ::= '<' local <hierarchy-path> '>'
    ::= '<' term '(' <expression> ')' '>'
```

The second way to declare `extern` clause enables description of a set of term having specific priority (defined by identifier before this set).

Thus, a term (of `extern` clause) is a number, a string (surrounded with double-quotes), an identifier, a function using some parameters (external type), or a set of term (of `extern` clause).

A term (of `extern` clause) can also refer to a part of the described component (flow variables, state variables, events or sub-components).

The `< local ... >` term enables definition of a parameter linked to current component, but only usable in external clauses. On the contrary, a term composed by only one identifier is defined as a global parameter.

The `< term ... >` term enables the use of expression inside external clauses.

Following chapter deals with wanted syntax of usual `extern` directives.

#### 2.7.9.1. Comments

It's possible to add a comment, that is to say a string, to a part of a component (flow variable, state variable, event or sub-component).

This comment can be used by tools in order to display extra information. For example, let be a comment linked to events representing failure of the component. This comment can be used by fault-tree generator for commenting basic event of tree for more traceability.

```
<remark-decl>
    ::= remark <objects> '=' <string>
<objects>
    ::= <object>
    ::= <objects-set>
<objects-set>
    ::= '{' <object> (';' <object>)* '}'
```



```

<object>
  ::= '<' flow <hierarchy-path> '>'
  ::= '<' state <hierarchy-path> '>'
  ::= '<' event <hierarchy-path> '>'
  ::= '<' sub <hierarchy-path> '>'
  ::= '<' local <hierarchy-path> '>'

```

### 2.7.9.2. Named parameters

Named parameters can be used for parameters of probability laws. It links identifier (local or global) to a law parameter, that is to say a number or a density function.

```

<param-decl>
  ::= parameter <identifier> '=' <param>
  ::= parameter '<' local <hierarchy-path> '>' '=' <param>

<param>
  ::= <float>
  ::= <identifier>
  ::= <identifier> '(' <extern-term> (',' <extern-term>)* ')'
  ::= '<' local <hierarchy-path> '>'

```

The identified density functions are :

- uniform(min-value, max-value)
- normal(mean, standard-deviation)
- lognormal(mean, error-factor)

### 2.7.9.3. Probability laws

Model events can be linked to probability law of occurrence. This is mainly used by tools generating fault-trees. These probability laws are described with functions using one or many parameters.

```

<law-decl>
  ::= law <events> '=' <law>

<events>
  ::= '<' event <hierarchy-path> '>'
  ::= <events-set>

<events-set>
  ::= '{' <events> (',' <events>)* '}'

<law>
  ::= <identifier> '(' <param> (',' <param>)* ')'

```

Identified probability laws are :

- exponential(lambda)
- Weibull(alpha, beta)
- Dirac(delay)
- constant(probability)
- asymptotic\_exponential(lambda, mu)
- GLM(gamma, lambda, mu)
- periodic\_test(lambda, period, t0)
- CMT(lambda, mission-time, Q)
- uniform(min, max)
- periodic(T, t0)

### 2.7.9.4. Attributes

Attributes can be linked to model events. These attributes are couples (name, value) mainly used by tools generating fault-trees.

Syntax used to define attributes on base events is the following:

```

<attribute-decl>

```

```

::= attribute <identifier> '(' <events> ')' '=' <attribute-value>
<attribute-value>
::= true
::= false
::= <integer>
::= <float>
::= <string>
::= <identifier>

```

<events> is defined in chapter dealing with laws.

For example, `attribute type(<event def>)= "CircuitBreaker"` means value of "type" attribut for "def" event is equal to "CircuitBreaker".

### 2.7.9.5. Priority

In case of deterministic events, two events (instantaneous or time delay) may appear at the same time. In order to choose the firing order of transition, a priority can be defined as shown below:

```

<priority-decl>
::= priority <events> '=' <integer>

```

<events> is defined in chapter dealing with laws and <integer> is an integer greater or equal to zero.

Events with high priority will be the more priority events.

### 2.7.9.6. Conditional events

Conditional transition are a specific case of immediate transitions. They are merged and defined with a bucket external clause. Every transition of a bucket must have the same guard. When transitions of a bucket become fireable, only one transition is randomly chosen (fired) depending on their probability law.

In order to say that an event is conditional, we have to link it to a constant probability, and we have to declare it in a bucket with the following syntax:

```

<bucket-decl>
::= bucket <events>

```

<events> is defined in chapter dealing with laws.

Sum of event probabilities of a bucket must me equal to 1.

### 2.7.9.7. Predicates and properties

Predicates and properties are quantity computed from model and can be observed with different tools. These quantities can be defined from state or flow variable, and generally from expression define with `<term (expr)>` clause. Predicated are Boolean quantity, properties are numerical quantity (that is to say integer or real).

The role of this quantity is to give a way to observe model to external tools.

They are defined with the following syntax:

```

<predicate-decl>
::= predicate <identifier> '=' '<' term '(' <expression> ')>' '>'
::= predicate '<' local <hierarchy-path> '>' '=' '<' term '(' <expression> ')>' '>'
<property-decl>
::= property <identifier> '=' '<' term '(' <expression> ')>' '>'
::= property '<' local <hierarchy-path> '>' '=' '<' term '(' <expression> ')>' '>'

```

### 2.7.9.8. Events with memory (preemptible)

When transition is valid, a delay is defined functions of probability law assigned to event. Delay is randomly fired for stochastic transitions, or fire with a determinate way for deterministic transitions. Transition will be fired at the current

time of simulation plus this delay if transition stay valid until firing time. If transition doesn't stay valid (because of other transition firing) until this time, there are two way to handle transitions when they become valid again:

- The timed elapsed when transition was valid is forgotten. When transition becomes valid again, a new delay is computed. It's AltaRica normal behavior.
- The timed elapsed when transition was valid is memorized. When transition becomes valid again, remaining time is used instead of a new one.

This property of events is declared as shown below:

```
<preemptible-decl>
 ::= preemptible <events>
```

<events> is defined in chapter dealing with laws.

### 2.7.9.9. Properties of component (nodeproperty)

It can be useful to memorize information. It is useful to keep information about components in results files, for example their name, their creation date, their version or owner.

It can be made thanks to node property external clause as shown below:

```
<nodeproperty-decl>
 ::= nodeproperty <identifier> '=' <attribute-value>
 ::= nodeproperty '<' local <hierarchy-path> '>' '=' <attribute-value>
```

<attribute-value> is defined in chapter dealing with attributes.

It's also possible to memorize information about generation of AltaRica file, and other information (user who generated it ...), adding external clause `nodeproperty` to main node of AltaRica file.

## F. AltaRica Extended Grammar

```

<identifier> ::= '[a-z][a-zA-Z0-9_-]*'

<integer> ::= '0 | ([1-9][0-9]*)'

<float> ::= '[0-9]*.[0-9]+([eE][+-]?[0-9]+)?'

<string> ::= '"[^"]+"'

<altarica-description>
  ::= <global-declaration-list>
<global-declaration-list>
  ::= <global-declaration> ((';'? <global-declaration>)*
  ::=
<global-declaration>
  ::= <constant-declaration>
  ::= <domain-declaration>
  ::= <function-declaration>
  ::= <model-declaration>

<hierarchy-path> ::= <identifier> ('.' <hierarchy-path>)*

<bool-expr> ::= <expression>
<num-expr> ::= <expression>
<symb-expr> ::= <expression>
<struct-expr> ::= <expression>

<expression>
  ::= <match-expr>
  ::= <unmatch-expr>
<match-expr>
  ::= <other-expr>
  ::= if <bool-expr> then <match-expr> else <match-expr>
<unmatch-expr>
  ::= '(' if <bool-expr> then <bool-expr> ')'
  ::= if <bool-expr> then <match-expr> else <unmatch-expr>
<other-expr>
  ::= <or-expression>

<atomic-expression>
  ::= <integer>
  ::= <float>
  ::= true
  ::= false
  ::= <id-variable>
  ::= '(' <expression> ')'
  ::= min '(' <num-expr> (',' <num-expr>)+ ')'
  ::= max '(' <num-expr> (',' <num-expr>)+ ')'
  ::= '#' '(' <bool-expr> (',' <bool-expr>)+ ')'
  ::= <identifier> '(' <expression> (',' <expression>)* ')'
  ::= <identifier> '(' ')'
  ::= case '{'
    (<bool-expr> ':' <expression> ',')*
    else <expression>
  '}'
<id-variable>
  ::= <hierarchy-path> '^' <identifier>
  ::= <hierarchy-path>

<unary-expression>
  ::= - <unary-expression>
  ::= <operator-not> <unary-expression>
  ::= <atomic-expression>
<operator-not>
  ::= not
  ::= '~'

<multiplicative-expression>
  ::= <multiplicative-expression> '*' <unary-expression>
  ::= <multiplicative-expression> '/' <unary-expression>
  ::= <multiplicative-expression> '%' <unary-expression>

```

```

::= <unary-expression>

<additive-expression>
::= <additive-expression> '+' <multiplicative-expression>
::= <additive-expression> '-' <multiplicative-expression>
::= <multiplicative-expression>

<relational-expression>
::= <relational-expression> '<' <additive-expression>
::= <relational-expression> '>' <additive-expression>
::= <relational-expression> '<=' <additive-expression>
::= <relational-expression> '>=' <additive-expression>
::= <additive-expression>

<equality-expression>
::= <relational-expression> '=' <relational-expression>
::= <relational-expression> <operator-neq> <relational-expression>
::= <relational-expression> <operator-impl> <relational-expression>
::= <relational-expression>

<operator-neq>
::= '!='
::= '#='

<operator-impl>
::= 'impl'
::= '=>'

<and-expression>
::= <and-expression> <operator-and> <equality-expression>
::= <equality-expression>

<operator-and>
::= 'and'
::= '&'

<or-expression>
::= <or-expression> <operator-or> <and-expression>
::= <and-expression>

<operator-or>
::= 'or'
::= '|'

<domain-declaration>
::= domain <identifier> = <domain-definition>

<domain-definition>
::= <link-domain>
::= <domain>

<domain>
::= <range-domain>
::= <enumeration-domain>
::= <predefined-domain>
::= <identifier>

<predefined-domain>
::= bool
::= int
::= float

<range-domain>
::= '[' <integer> ',' <integer> ']'

<enumeration-domain>
::= '{' <identifier-list> '}'

<link-domain>
::= link
    flow <link-flow-list> (';')?
    ( inverse <inverse-list> (';')? )?
    assert <connect-list> (';')?
    knit
<link-flow-list>
::= <link-flow-decl> (';' <link-flow-decl>)*
<link-flow-decl>
::= <identifier-list> ':' <domain>
<inverse-list>
::= <id-flow-link> (';' <id-flow-link>)*
<connect-list>
::= <connect-decl> (';' <connect-decl>)*

```

```

<connect-decl>
  ::= <id-flow-link> ':' <id-flow-link>
<id-flow-link>
  ::= in '^' <identifier>
  ::= out '^' <identifier>

<function-decl>
  ::= func <identifier>
    <function-elem>
    <function-body>
  cnuf
<function-elem>
  ::= <flow-cls> <function-elem>
  ::=
<function-body>
  ::= <assert-cls> <function-body>
  ::=

<model-declaration>
  ::= node <identifier>
    <node-elem>
    <node-body>
    <extern-1st>
  edon

<node-elem>
  ::= <node-elem-field> <node-elem>
  ::=
<node-elem-field>
  ::= <flow-cls>
  ::= <state-cls>
  ::= <event-cls>
  ::= <sub-cls>

<node-body>
  ::= <node-body-field> <node-body>
  ::=
<node-body-field>
  ::= <trans-cls>
  ::= <assert-cls>
  ::= <sync-cls>
  ::= <init-cls>

<extern-1st>
  ::= <extern-cls> <extern-1st>
  ::=

<flow-cls>
  ::= flow <flow-decl-list> (';')?

<flow-decl-list>
  ::= <flow-decl> (';' <flow-decl>)*

<flow-decl>
  ::= <identifier-list> ':' <domain> ':' <orientation>
<orientation>
  ::= in
  ::= out
  ::= private
  ::= local

<state-cls>
  ::= state <state-decl-list> (';')?

<state-decl-list>
  ::= <state-decl> (';' <state-decl>)*

<state-decl>
  ::= <identifier-list> ':' <domain>

<event-cls>
  ::= event <event-decl-list> (';')?
<event-decl-list>
  ::= <event-decl> (';' <event-decl>)*
<event-decl>

```

```

    ::= <identifier>

<sub-cls>
    ::= sub <sub-list> (';')?
<sub-decl-list>
    ::= <sub-decl> (';' <sub-decl>)*
<sub-decl>
    ::= <identifier-list> ':' <identifier>

<trans-cls>
    ::= trans <trans-list> (';')?
<trans-list>
    ::= <transition> (';' <transition>)*

<transition>
    ::= <bool-expr> (<transition-target>)+

<transition-target>
    ::= '|' <event-name-list> '->' <assignment-list>
    ::= '|' <event-name-list> '->'

<assignment-list>
    ::= <assignment> (',' <assignment>)*

<assignment>
    ::= <identifier> ':' <expression>

<assert-cls>
    ::= assert <assert-list> (';')?
    ::= assert
<assert-list>
    ::= <bool-expr> (';' <bool-expr>)*

<sync-cls>
    ::= sync <vector-list> (';')?

<vector-list>
    ::= <vector> (';' <vector>)*

<vector-sync> ::= '<' <hierarchie-path> (',' <hierarchie-path>)+ '>'
<vector-diff> ::= '<' <hierarchie-path> ('|' <hierarchie-path>)+ '>'
<vector-ccf>  ::= '<' <hierarchie-path> ('?' <hierarchie-path>)+ '>'

<vector>
    ::= <vector-sync>
    ::= <vector-diff>
    ::= <vector-ccf>

<init-cls>
    ::= init <init-list> (';')?
<init-list>
    ::= <init-def> (';' <init-def>)*

<init-def>
    ::= <hierarchy-path> ':' <expression>

<extern-cls>
    ::= extern <extern-list> (';')?
<extern-list>
    ::= <extern-decl> (';' <extern-decl>)*

<extern-decl>
    ::= <identifier> <extern-term> '=' <extern-term>
    ::= <identifier> <extern-term>

<extern-term>
    ::= true
    ::= false
    ::= <integer>
    ::= <float>
    ::= <string>
    ::= <identifier>
    ::= <identifier> '(' <extern-term> (',' <extern-term>)* ')'
    ::= '{' <extern-term> (',' <extern-term>)* '}'

```

```

::= '<' flow <hierarchy-path> '>'
::= '<' state <hierarchy-path> '>'
::= '<' event <hierarchy-path> '>'
::= '<' sub <hierarchy-path> '>'
::= '<' local <hierarchy-path> '>'
::= '<' term '(' <expression> ')' '>'

<remark-decl>
  ::= remark <objects> '=' <string>
<objects>
  ::= <object>
  ::= <objects-set>
<objects-set>
  ::= '{' <object> (',' <object>)* '}'
<object>
  ::= '<' flow <hierarchy-path> '>'
  ::= '<' state <hierarchy-path> '>'
  ::= '<' event <hierarchy-path> '>'
  ::= '<' sub <hierarchy-path> '>'
  ::= '<' local <hierarchy-path> '>'

<param-decl>
  ::= parameter <identifier> '=' <param>
  ::= parameter '<' local <hierarchy-path> '>' '=' <param>
<param>
  ::= <float>
  ::= <identifier>
  ::= <identifier> '(' <extern-term> (',' <extern-term>)* ')'
  ::= '<' local <hierarchy-path> '>'

<law-decl>
  ::= law <events> '=' <law>
<events>
  ::= '<' event <hierarchy-path> '>'
  ::= <events-set>
<events-set>
  ::= '{' <events> (',' <events>)* '}'
<law>
  ::= <identifier> '(' <param> (',' <param>)* ')'

<attribute-decl>
  ::= attribute <identifier> '(' <events> ')' '=' <attribute-value>
<attribute-value>
  ::= true
  ::= false
  ::= <integer>
  ::= <float>
  ::= <string>
  ::= <identifier>

<priority-decl>
  ::= priority <events> '=' <integer>

<bucket-decl>
  ::= bucket <events>

<predicate-decl>
  ::= predicate <identifier> '=' '<' term '(' <expression> ')' '>'
  ::= predicate '<' local <hierarchy-path> '>' '=' '<' term '(' <expression> ')' '>'
<property-decl>
  ::= property <identifier> '=' '<' term '(' <expression> ')' '>'
  ::= property '<' local <hierarchy-path> '>' '=' '<' term '(' <expression> ')' '>'

<preemptible-decl>
  ::= preemptible <events>

<nodeproperty-decl>
  ::= nodeproperty <identifier> '=' <attribute-value>
  ::= nodeproperty '<' local <hierarchy-path> '>' '=' <attribute-value>

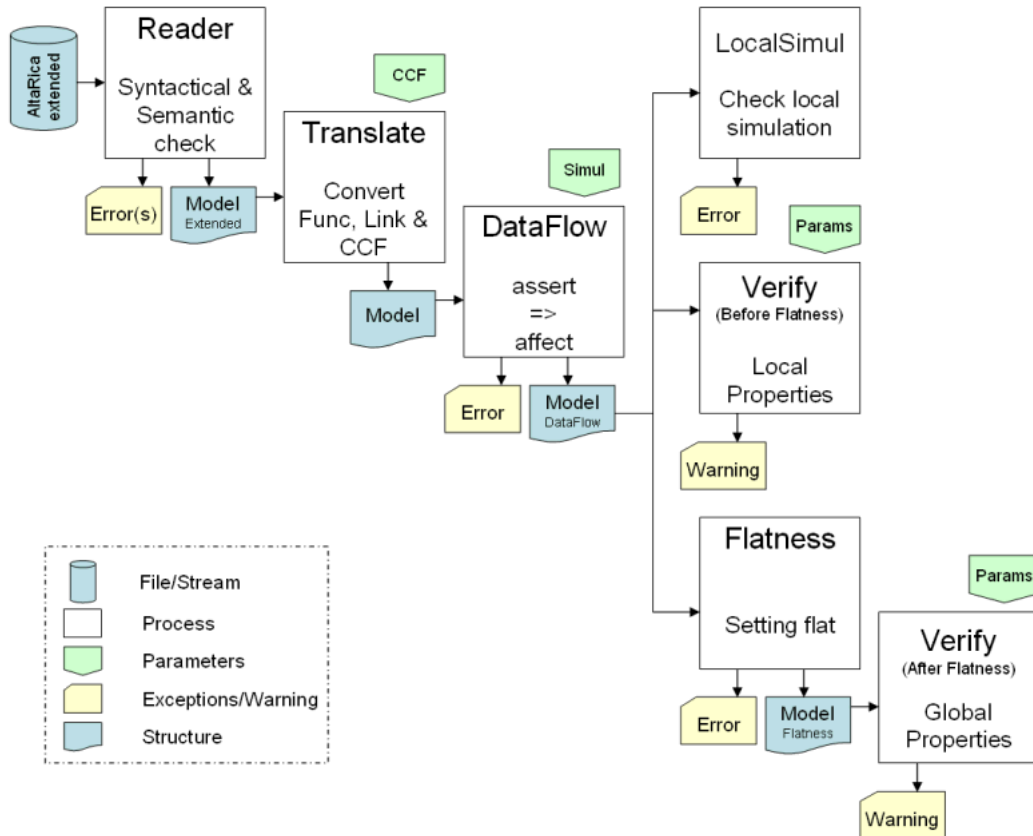
```



## G. AltaRica model verification

AltaRica model verification requires a certain number of steps. Each step will detect potential errors on model.

The linking of these steps is described as follows:



Overall principle consists in:

- reading file (or flow) in extended AltaRica format. Extended format adds elements to manage functions, structured links and common cause failure synchronization.
- translating extended model in 'standard' AltaRica.
- converting assertions to assignments (or convert to 'dataflow' format)
- verify node having behavior with local simulation
- setting flat a model (that's to say removing hierarchy in order to use only one component standing for the system)
- verifying properties like validity of some predefined external clauses, or like loop presence in assertions ...

Each transformation generates a data structure standing for system, if no error is detected. On the contrary case, an exception is generated so that user can modify his model. Some transformations need one or more input-parameters.

The following part deals with each transformation principle in details and all possible error messages.

### 1. Syntactical check

Syntactical check uses lexical and syntactic analyzer like Lex&Yacc. AltaRica Extended's syntax is defined in BNF format (e.g. LggOcas-02-0.pdf).

If there is at least one error, a message will display line and position of the found error.

```
1 | node syntax
2 | state State:bool;
```

```

3 | event Evt;
4 | trans
5 |   State |- Evt -> State=false;
6 | edon;

```

```

AltaRica : 5 : Parser error on token = : syntax error
Event : 4 : Event (Evt) is orphan (No transition use it).

```

## 2. Semantic check

Semantic check aims at model consistency validation. That's to say, model uses known and defined data, data are compatible, model seems to be 'logic', ...

Errors on domains :

- Unknown domain

```

1 | node Semantic
2 |   flow Out : Power;
3 | edon;

```

```

UndefDomain : 2 : Domain (Power) unknown
AltaRica : 2 : Construct domain

```

- Impossible interval-domain: Min > Max

```

1 | node Semantic
2 |   flow Out : [3,2];
3 | edon;

```

```

UndefDomain : 2 : Domain : Min (3) > Max (2) in range

```

- Already declared domain

```

1 | domain Power = {Pos, Null, Neg};
2 | domain Power = [0,2];

```

```

DomainRange : 2 : Name (Power) already used for another domain
AltaRica : 2 : Defined domain (Power)

```

Errors on structured domains ([link](#)):

- Not structured domain made of fields with structured domain

```

1 | domain First = link
2 |   flow A,B:int;
3 |   assert
4 |     in^A := out^A;
5 |     in^B := out^B;
6 | knil;
7 |
8 | domain Second = link

```

```

9 |   flow
10 |     A:int;
11 |     B:First;
12 |   assert
13 |     in^A := out^A;
14 |     in^B := out^B;
15 | knil;

```

```

Link : 11 : Link : Struct domain not allowed
AltaRica : 11 : Construct link
Link : 15 : Flow (B) unknown
AltaRica : 15 : Construct link

```

- 'inverse' and 'assert' clauses are not compatible

```

1 | domain First = link
2 |   flow A,B:int;
3 |   inverse
4 |     in^A; out^A;
5 |   assert
6 |     in^A := out^A;
7 |     in^B := out^B;
8 | knil;

```

```

Flow : 2 : inverse and assert clauses are incompatible
AltaRica : 8 : Construct link

```

- Flow variable already assigned

```

1 | domain First = link
2 |   flow A,B:int;
3 |   assert
4 |     in^A := out^A;
5 |     in^A := out^B;
6 | knil;

```

```

Flow : 2 : Flow (in^A) already assigned
AltaRica : 6 : Construct link

```

Errors on operators/functions (func):

- Structured domain with 'inverse' clause are not allowed in function

```

1 | domain Connect = link
2 |   flow A,B:int;
3 |   inverse in^A; out^A;
4 |   assert
5 |     out^A := in^A;
6 |     in^B := out^B;
7 | knil;
8 |
9 | func Operation
10 |   flow
11 |     Operation:int:out;
12 |     Arg1:Connect:in;
13 |   assert
14 |     Operation = Arg1^B;
15 | cnuf

```

```
Flow : 12 : Struct inverse not possible
AltaRica : 15 : Defined function (Operation)
```

- Output variable already defined

```
1 | func Operation
2 |   flow
3 |     Operation:int:out;
4 |     Arg1,Arg2:int:in;
5 |     Add:bool:out;
6 |   assert Operation =
7 |     (if Add then Arg1+Arg2 else Arg1-Arg2);
8 | cnuf
```

```
Flow : 5 : Out variable already exists
AltaRica : 8 : Defined function (Operation)
```

- Wrong number of argument

```
1 | func Operation
2 |   flow
3 |     Operation:int:out;
4 |     Arg1,Arg2:int:in;
5 |     Add:bool:in;
6 |   assert Operation =
7 |     (if Add then Arg1+Arg2 else Arg1-Arg2);
8 | cnuf
9 |
10 | node Args
11 |   flow
12 |     In1,In2:int:in;
13 |     Out:int:out;
14 |   assert
15 |     Out=Operation(In1,In2);
16 | edon
```

```
Expr : 15 : arguments number
AltaRica : 16 : Defined node (Args)
```

- Function already declared

```
1 | func Operation
2 |   flow
3 |     Operation:int:out;
4 |     Arg1,Arg2:int:in;
5 |     Add:bool:in;
6 |   assert Operation =
7 |     (if Add then Arg1+Arg2 else Arg1-Arg2);
8 | cnuf
9 |
10 | func Operation
11 |   flow
12 |     Operation:int:out;
13 |     Arg:int:in;
14 |   assert Operation = -Arg;
15 | cnuf
```

```
Fct : 8 : Name (Operation) already used for another function
```

```
AltaRica : 15 : Defined function (Operation)
```

- Unknown function

```
1 | node Node
2 |   flow
3 |     Out:int:out;
4 |     In1,In2:int:in;
5 |   assert
6 |     Out = Fct(In1,In2);
7 | edon
```

```
Expr : 6 : Fct (Fct) unknown
AltaRica : 6 : Construct expression
Expr : 6 : Undef expression
AltaRica : 7 : Defined node (Node)
```

Errors on nodes/components (node):

- Component already declared

```
1 | node Node
2 |   flow
3 |     Out:int:out;
4 |     In1,In2:int:in;
5 |   assert
6 |     Out = In1+In2;
7 | edon
8 |
9 | node Node
10 |   flow
11 |     Out:int:out;
12 |     In:int:in;
13 |   assert
14 |     Out = -In;
15 | edon
```

```
Node : 7 : Name (Node) already used for another node
AltaRica : 15 : Defined node (Node)
```

- Name-conflict with flow variable

```
1 | node Node
2 |   flow
3 |     Out:int:out;
4 |     In1,In1:int:in;
5 |   assert
6 |     Out = In1+In2;
7 | edon
```

```
Flow : 4 : Name-conflict (In1) with flow variable
AltaRica : 7 : Defined node (Node)
```

- Name-conflict with state variable

```
1 | node Node
2 |   state Out:int;
```

```

3 |   flow  Out:int:out;
4 |       In:int:in;
5 |   assert
6 |       Out = In;
7 | edon

```

Flow : 3 : Name-conflict (Out) with state variable  
AltaRica : 7 : Defined node (Node)

- Name-conflict with symbolic constant

```

1 | node Node
2 |   flow  Out:{ST,SF,SB}:out;
3 |   state ST:int;
4 |   assert
5 |       Out = (if ST>3 then SF else SB);
6 | edon

```

State : 3 : Name-conflict (ST) with symbolic constant  
AltaRica : 6 : Defined node (Node)

- Unknown component

```

1 | node Unit
2 |   flow
3 |       Out:bool:out;
4 |       In:bool:in;
5 |   state OK:bool;
6 |   assert
7 |       Out = (if OK then In else false);
8 | edon
9 |
10 | node Equip
11 |   flow
12 |       Out:bool:out;
13 |       In:bool:in;
14 |   sub
15 |       A,B:unit;
16 |   assert
17 |       A.In = In;
18 |       B.In = In;
19 |       Out = (A.In | B.In);
20 | edon

```

AltaRica : 15 : Node (unit) unknown  
Expr : 17 : Symbol (A.In) not found in current node  
AltaRica : 20 : Defined node (Equip)

Errors on flow variables (flow):

- Unknown flow variable

```

1 | domain Connect = link
2 |   flow A,B:int;
3 |   inverse in^C; out^A;
4 |   assert
5 |       out^A := in^A;
6 |       in^B := out^B;
7 | knil;

```

```
Link : 7 : Flow (C) unknown
AltaRica : 7 : Construct link
```

- Flow variable already declared

```
1 | domain Connect = link
2 |   flow A,A:int;
3 |   inverse in^A; out^A;
4 |   assert
5 |     out^A := in^A;
6 |     in^B := out^B;
7 | knil;
```

```
Flow : 2 : Flow (A) already exists
AltaRica : 7 : Construct link
```

- Local flow variables are not allowed with structured domains having 'inverse' clauses.

```
1 | domain Connect = link
2 |   flow A,B:int;
3 |   inverse in^A; out^A;
4 |   assert
5 |     out^A := in^A;
6 |     in^B := out^B;
7 | knil
8 |
9 | node Unit
10 |   flow
11 |     Mem:Connect:local;
12 |     Our:Connect:out;
13 |   assert
14 |     Out = Mem;
15 | edon
```

```
Link : 12 : Flow : Inverse struct domain not allowed for local flow
Expr : 14 : Symbol (Out) not found in current node
AltaRica : 15 : Defined node (Unit)
```

Errors on state variables (state) :

- State variables are not allowed with structured domains.

```
1 | domain Connect = link
2 |   flow A,B:int;
3 |   inverse in^A; out^A;
4 |   assert
5 |     out^A := in^A;
6 |     in^B := out^B;
7 | knil
8 |
9 | node Unit
10 |   flow Out:Connect:out;
11 |   state State:Connect;
12 |   assert
13 |     Out = State;
14 | edon
```

```
Link : 11 : State : Struct domain not allowed
Expr : 13 : Symbol (State) not found in current node
AltaRica : 14 : Defined node (Unit)
```

Errors on events (event):

- Unknown event

```
1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |   state OK:bool;
6 |   trans
7 |     OK |- def -> OK := false;
8 |   assert
9 |     Out = (if OK then In else false);
10 |   init OK := true;
11 | edon
```

```
Tree : 7 : Event (def) unknown
AltaRica : 11 : Defined node (Unit)
```

- Event already declared

```
1 | node Unit
2 |   event def;
3 |   flow
4 |     Out:bool:out;
5 |     In:bool:in;
6 |   state OK:bool;
7 |   event def;
8 |   trans
9 |     OK |- def -> OK := false;
10 |   assert
11 |     Out = (if OK then In else false);
12 |   init OK := true;
13 | edon
```

```
Event : 8 : Event (def) already exists
AltaRica : 13 : Defined node (Unit)
```

- Orphan event (used by none of transitions)

```
1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |   state OK:bool;
6 |   event def;rep;
7 |   trans
8 |     OK |- def -> OK := false;
9 |   assert
10 |     Out = (if OK then In else false);
11 |   init OK := true;
12 | edon
```

```
Event : 7 : Event (rep) is orphan (No transition use it).
```



Errors on sub-components (sub):

- Sub-component already declared

```

1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |   state OK:bool;
6 |   assert
7 |     Out = (if OK then In else false);
8 |   edon
9 |
10 | node Equip
11 |   sub
12 |     A,A:Unit;
13 |   edon

```

```

Sub : 12 : Sub (A) already exists
AltaRica : 13 : Defined node (Equip)

```

Errors on assertions/assignments (assert):

- Always false assertion

```

1 | node Unit
2 |   flow True:bool:in;
3 |   state OK:bool;
4 |   assert
5 |     (if OK then not(true) else false);
6 |   init OK := true;
7 |   edon

```

```

Expr : 5 : Assert always false
AltaRica : 7 : Defined node (Unit)

```

- Assertion directly defined with function

```

1 | func Fct
2 |   flow
3 |     Fct:int:out;
4 |     Arg:int:in;
5 |   assert
6 |     Fct = Arg+1;
7 |   cnuf
8 |
9 | node Unit
10 |   flow In:bool:in;
11 |   assert
12 |     Fct(In);
13 |   edon

```

```

Expr : 12 : Assert not define with function
AltaRica : 13 : Defined node (Unit)

```

- Assertion without constant boolean expression

```

1 | node Unit

```

```

2 |   flow In:int:in;
3 |   assert
4 |     10 + 5;
5 | edon

```

Expr : 4 : Assert with no boolean constant expression  
 Expr : 4 : Attach data  
 AltaRica : 5 : Defined node (Unit)

Errors on expressions (assertions/guards/assignments) :

- Unknown identifier for current node (neither a variable, nor possible value of enumerate)

```

1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |   state State:{OK,KO,SB};
6 |   event def;rep;
7 |   trans
8 |     State=OK |- def -> State:=Ko;
9 |   assert
10 |     Out = (if State=ok then In else false);
11 |   init State := OK;
12 | edon

```

Expr : 8 : Symbol (Ko) not found in current node  
 AltaRica : 12 : Defined node (Unit)

- Non-boolean Argument(s)

```

1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |   state State:{OK,KO,SB};
6 |   assert
7 |     Out = (if State then In else false);
8 |   init State := OK;
9 | edon

```

Expr : 7 : No boolean args : State  
 AltaRica : 9 : Defined node (Unit)

- Non-numeric argument(s)

```

1 | node Unit
2 |   flow
3 |     Out:int:out;
4 |     In:int:in;
5 |   state State:{OK,KO,SB};
6 |   assert
7 |     Out = min(In, State);
8 |   init State := OK;
9 | edon

```

Expr : 7 : No numeric args : State  
 AltaRica : 9 : Defined node (Unit)

- Non-structured argument(s)

```

1 | domain Connect = link
2 |   flow A,B:int;
3 |   assert
4 |     in^A := out^A;
5 |     in^B := out^B;
6 | knil;
7 |
8 | node Unit
9 |   flow
10 |    Out:bool:out;
11 |    In1, In2:Connect:in;
12 |   assert
13 |     Out = (In1 != In2);
14 | edon

```

```

Expr : 13 : No structured args : In1
AltaRica : 14 : Defined node (Unit)

```

- Equality between arguments from domains that aren't compatible

```

1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In1:bool:in;
5 |     In2:int:in;
6 |   assert
7 |     Out = (In1 = In2);
8 | edon

```

```

Expr : 7 : Equality args
AltaRica : 8 : Defined node (Unit)

```

- Assignment with two input flows (in = in)

```

1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In1, In2:int:in;
5 |   assert
6 |     In1 = In2;
7 | edon

```

```

Expr : 6 : Assignment args (in = in)
AltaRica : 7 : Defined node (Unit)

```

- Assigne Assignment with enumerate domains that aren't equivalent

```

1 | node Unit
2 |   flow
3 |     Out:{OK,KO,SB}:out;
4 |     In:{OK,KO}:in;
5 |   assert
6 |     Out = In;

```

```
7 | edon
```

```
Expr : 6 : Assignment enums not equivalent
AltaRica : 7 : Defined node (Unit)
```

- Assignment with two constants (cst = cst)

```
1 | node Unit
2 |   flow
3 |     Out:{OK,KO,SB}:out;
4 |     In:{OK,KO}:in;
5 |   assert
6 |     OK = KO;
7 | edon
```

```
Expr : 6 : Assignment args (cst = cst)
AltaRica : 7 : Defined node (Unit)
```

- Division by zero

```
1 | node Unit
2 |   flow
3 |     Out:int:out;
4 |     In:int:in;
5 |   state OK:bool;
6 |   assert
7 |     Out = (if OK then In else In/0);
8 | edon
```

```
Expr : 7 : Division by zero
AltaRica : 8 : Defined node (Unit)
```

- Function are not allowed into transitions (guard or affectation), or into extern clause.

```
1 | func Add
2 |   flow
3 |     Add:int:out;
4 |     Arg1,Arg2:int:in;
5 |   assert
6 |     Add = Arg1+Arg2;
7 | cnuf
8 |
9 | node Unit
10 |   flow
11 |     Out:int:out;
12 |     In1,In2,Chk:int:in;
13 |   state OK:bool;
14 |   event fail;
15 |   trans
16 |     OK |- fail -> OK := (Add(In1,In2)!=Chk);
17 |   assert
18 |     Out = OK;
19 | edon
```

```
Expr : 16 : Function are not allowed in this context (transition or extern clause)
AltaRica : 19 : Defined node (Unit)
```

Errors on transitions (trans):

- Guard of a non-boolean transition.

```

1 | node Unit
2 |   flow In:int:in;
3 |   state OK:bool;
4 |   event chg;
5 |   trans
6 |     In |- chg -> OK := not(OK);
7 |   init OK := true;
8 | edon

```

```

Expr : 6 : Not boolean guard
AltaRica : 8 : Defined node (Unit)

```

- Guard always false

```

1 | node Unit
2 |   state State:{OK,KO,SB};
3 |   event def;
4 |   trans
5 |     OK=KO |- def -> State := KO;
6 | edon

```

```

Expr : 5 : Guard always false
AltaRica : 6 : Defined node (Unit)

```

- Variable already assigned in transition

```

1 | node Unit
2 |   state OK:bool;
3 |   event def;
4 |   trans
5 |     OK |- def ->
6 |       OK := false,
7 |       OK := not(OK);
8 | edon

```

```

Trans : 7 : State (OK) already assigned
AltaRica : 8 : Defined node (Unit)

```

- Domain not compatible for variable assignment

```

1 | node Unit
2 |   state OK:bool;
3 |   event def;
4 |   trans
5 |     OK |- def -> OK := 10;
6 | edon

```

```

Expr : 5 : Conflict domain assignment for state (OK)
AltaRica : 6 : Defined node (Unit)

```

Errors on synchronizations (sync):

- First event of a synchronization must belong to current component.

```

1 | node Unit
2 |   state OK:bool;
3 |   event def;rep;
4 |   trans
5 |     OK |- def -> OK := false;
6 |     not(OK) |- rep -> OK := true;
7 |   init OK := true;
8 | edon
9 |
10 | node Equip
11 |   sub A,B : Unit;
12 |   sync <A.def ? B.def>;
13 | edon;

```

Sync : 12 : First event must belong to current model  
 AltaRica : 13 : Defined node (Equip)

- Events following the first event must belong to sub-components

```

1 | node Unit
2 |   state OK:bool;
3 |   event def;rep;
4 |   trans
5 |     OK |- def -> OK := false;
6 |     not(OK) |- rep -> OK := true;
7 |   init OK := true;
8 | edon
9 |
10 | node Equip
11 |   sub A,B : Unit;
12 |   event rep;
13 |   sync <rep, A.rep, rep>;
14 | edon;

```

Sync : 13 : Other event must belong to models component  
 AltaRica : 14 : Defined node (Equip)

Errors on initializations (init):

- Impossible initialization: Usually, initial value is not compatible with the domain of state variable.

```

1 | node Unit
2 |   state OK:bool;
3 |   init OK := 10;
4 | edon

```

Expr : 3 : Init [OK := 10] is not possible  
 AltaRica : 4 : Defined node (Unit)

### 3. Translation into 'standard' AltaRica

AltaRica Extended language has added some constructions helping model entry. It's so useful to convert to 'standard' AltaRica.

They are tree specific constructions:

1. Structured flows: they allow easy representation of complex connection between two components.
2. Operators/Functions can be considered as components without behavior (no state variable, no event). They are used directly in component assertions.
3. Synchronizations of Common Cause Failures (CCF) enrich model. They allow to consider that many events (failures) can be fired at the same time, but without deleting each event presence (as it would be done in case of broadcast synchronizations).

If syntactic and semantic check didn't display errors, this processing should not generate errors.

## 4. Convert assertions to assignments

### Conversion to Dataflow format

In AltaRica, it's possible to write assertions that are not implicitly assignments. Some users use to write assertions like `if [condition] then [var]=[value]`. It's the same thing as the following boolean implication `[condition] => ([var]=[value])`.

'AltaRica DataFlow' language view assertions as assignments on output flow variables. That is to say `out = [fct(ins, states)]`.

Goal is to convert usually used assertions of type implication to dataflow assignments.

In order to do that, transformation algorithm need parameters which are all component assertions to generate equivalent assignments.

This algorithm is made of two steps:

A. For each flow (local or output), it search an equation like `flow = fct(flows, states>`.

This step is made of three sub-steps:

1. Is there a dataflow equation on considered flow ?
2. Is there a dataflow equation hide in clauses `if ... then ... else ...` on considered flow ?
3. On the contrary, all clauses assigning variable are retrieved in couple (condition, assignment) and equivalent equations are generated.

Consistency and completeness of assertions are checked thanks to simple flow-simulator.

B. For every assertion:

1. It verifies there is no assignment with output-variable if at least one assertion has been generated in step A.3.
2. It verifies that there is no circular definition of variables. Assignment is tidied up to avoid ambiguities.

List of A.3 errors:

- No assignment for a given variable

```

1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |   state
5 |     State:{OK,KO,SB};
6 |   /* ... */
7 |   assert
8 |     Out;
9 | edon

```

```
Node : 9 : DataFlow(DataFlow, Out) - No assignment operation for variable : Out
```

- Presence of operator avoiding processing. Inside non-dataflow-equation, the only allowed operators are : implication if [condition] then [affects] and test-operators if [condition] then [affects] else [affects] ; condition is boolean expressions and assignment is conjunction (operator &) of assignment ([Var]=[Value]).

```
1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |     icone:[1,3]:out;
6 |   state
7 |     State:{OK,KO,SB};
8 |   /* ... */
9 |   assert
10 |     if State=OK then icone=1 | Out=true;
11 |     if State=KO then icone=2 | Out=false;
12 |     if State=SB then icone=3 | Out = false;
13 | edon
```

```
Node : 13 : DataFlow(DataFlow, Out) - Operator not dataflowisable : ((icone = 1) or (Out = true))
```

- Output flow not connected.

```
1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |     icone:[1,3]:out;
6 |   state
7 |     State:{OK,KO,SB};
8 |   /* ... */
9 |   assert
10 |     (if State=OK then Out=In else Out=false);
11 | edon
```

```
Node : 11 : DataFlow(DataFlow, icone) no connected variable
```

List of B errors:

- Assignment of output flow is not allowed.

```
1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |     icone:[1,3]:out;
6 |   state
7 |     State:{OK,KO,SB};
8 |   /* ... */
9 |   assert
10 |     if State=OK then icone=1;
11 |     if State=KO then icone=2;
12 |     if State=SB then icone=3;
13 |     if icone=1 then Out=In else Out=false;
14 | edon
```

```
Node : 14 : DataFlow(DataFlow, Out) assignment with output flow invalid
```



- At the end of 'dataflowisation' process, there are still assertions that are not taken into account. It usually comes from useless assignment of a variable.

```

1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |     icone:[1,3]:out;
6 |   state
7 |     State:{OK,KO,SB};
8 |   /* ... */
9 |   assert
10 |     if State=OK then Out=In else Out=false;
11 |     if State=OK then icone=1;
12 |     if State=KO then icone=2;
13 |     if State=SB then icone=3 & Out=false;
14 | edon

```

```

Node : 14 : DataFlow(DataFlow, ???) - Already assert exist
((State = SB) => (Out = false))

```

- Set of assignments creating a loop

```

1 | node DataFlow
2 |   flow
3 |     a,b,c:bool:in;
4 |     v,w,x,y,z:bool:out;
5 |   /* ... */
6 |   assert
7 |     w = (v & b);
8 |     x = (y & z);
9 |     y = (w & a);
10 |     z = (v & c);
11 |     v = (a|b|x);
12 | edon

```

```

Node : 12 : DataFlow(DataFlow, ???) - No DAG equation.
Current loop :
v
x
y
w
v

```

Possible errors during check with simulation of flows:

- It isn't possible to verify completeness and/or consistency of a model, if one of its input flows has an infinite domain definition (integer or float)

```

1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In:int:in;
5 |   state
6 |     State:{OK,KO,SB};
7 |   /* ... */
8 |   assert
9 |     if State=OK & In > 10 then Out=true;
10 |     if State=KO & In < 20 then Out=false;
11 |     if (State=SB | In <=10 | In >=20) then Out = true;

```

12 | edon

Node : 12 : DataFlow(DataFlow, Out) - Simul - Infinity domain from variable : In

- Simulation is only possible if component isn't too-complex. On the contrary, this simulation will take too much time and too much memory.

```

1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In1,In2,In3,In4,In5,In6:[0,9]:in;
5 |   state
6 |     State:{OK,KO,SB};
7 |   /* ... */
8 |   assert
9 |     if State=OK & (In1+In2+In3+In4+In5+In6) > 26 then Out=true;
10 |    if State!=OK then Out=false;
11 | edon

```

Node : 11 : DataFlow(DataFlow, Out) - Simul - Too complex component

- The simulation engendered an affectation of a variable outside its domain of definition.

```

1 | node DataFlow
2 |   flow
3 |     Out:[1,2]:out;
4 |   state
5 |     State:{OK,KO,SB};
6 |   /* ... */
7 |   assert
8 |     if State=OK then Out=1;
9 |     if State=KO then Out=2;
10 |    if State=SB then Out=3;
11 | edon

```

Node : 11 : DataFlow(DataFlow, Out) - Simul - Affectation outside domain of definition  
Value 3 (Domain [1,2]) with valuation :  
State = SB

- Consistency error during simulation: a valuation of input variables can generate assignments having different values.

```

1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |   state
6 |     State:{OK,KO,SB};
7 |   /* ... */
8 |   assert
9 |     if State=OK | In then Out=true;
10 |    if State=KO & In then Out=false;
11 |    if State=KO & not(In) then Out=true;
12 |    if State=SB then Out=false;
13 | edon

```

Node : 13 : DataFlow(DataFlow, Out) - Simul - Incoherence error  
Value (true, false) with valuation :

```
State = KO
In = true
```

- Completeness error during simulation: there is an input variable valuation that has no assignment.

```
1 | node DataFlow
2 |   flow
3 |     Out:bool:out;
4 |     In:bool:in;
5 |   state
6 |     State:{OK,KO,SB};
7 |   /* ... */
8 |   assert
9 |     if State=OK & In then Out=true;
10 |    if State=KO & In then Out=false;
11 |    if State=KO & not(In) then Out=true;
12 |    if State=SB then Out=false;
13 | edon
```

```
Node : 13 : DataFlow(DataFlow, Out) - Simul - Uncompleteness error
No value with valuation :
  State = OK
  In = false
```

## 5. Local simulation of components

Local simulation of components enables detection of potentials default like:

- transition conflicts: Two transitions - having equivalent guards and associated with the same event - may be valid simultaneously.
- variable assignment with a value from a different domain of definition.
- a too much complex component (too many state variables, too many input variable) which could generate a combinative explosion with some tools.
- A dynamic component can be an issue during some fault tree generation, and for comparisons between tree generation and sequence generation. A component is dynamic if from the same initial state, 2 event-permutations lead to 2 different states.

Principle is to do a local simulation of each component having behavior (presence of transition or state variable). This simulation assumes that component input flows have the same value during simulation time. This simulation is made for every value of state variables and input flows.

Messages associated with too complex components.

- Local simulation can't be done if a variable (state or flow) has an infinite domain (Integer or Float).

```
1 | node Test
2 |   flow
3 |     I : int : in ;
4 |   state S : bool ;
5 |   event a;
6 |   init S := false ;
7 |   trans
8 |     (I > 25) & not(S) |- a -> S := true;
9 | edon
```

```
Node : 9 : LocalSimul(Test) - Infinity domain from variable : I
```

- Local simulation can't be done if component is too complex. Otherwise, this simulation would take too many times and memory.

```

1 | node Test
2 |   flow
3 |     O : bool : out ;
4 |     I1,I2,I3,I4,I5 : [0,9] : in ;
5 |     state S : bool ;
6 |     event a;
7 |     init S := false ;
8 |     trans
9 |       ((I1+I2+I3+I4+I5) > 25) & not(S) |- a -> S := true;
10 |    assert
11 |      O = (if S then I1>I2 else I3>I4);
12 | edon

```

```

Node : 12 : LocalSimul(Test) - Too complex component
Cardinal = 200000 [ > 100000]

```

- Local simulation can't be done if there are too many dependent events. Otherwise there are too many permutations, and simulation takes too many time.

```

1 | node Test
2 |   flow
3 |     In : [0,9] : in ;
4 |     state S : [0,9] ;
5 |     event a; b; c; d; e; f; g; h;
6 |     init S := 0 ;
7 |     trans
8 |       S = In+0 |- a -> S := In+1;
9 |       S = In+1 |- b -> S := In+2;
10 |      S = In+2 |- c -> S := In+3;
11 |      S = In+3 |- d -> S := In+4;
12 |      S = In+4 |- e -> S := In+5;
13 |      S = In+5 |- f -> S := In+6;
14 |      S = In+6 |- g -> S := In+7;
15 |      S = In+7 |- h -> S := In+8;
16 | edon

```

```

Node : 16 : LocalSimul(Test) - Too complex component
Cardinal = 4032000 [ > 100000]

```

A fault tree is a static view of a system. During fault tree generation, we assume that generated system is static. Local simulation enables static component checking. That's to say its state doesn't depend on events order leading to it.

- Example 1

```

1 | node Test
2 |   state S : [0,4] ;
3 |   event a; b;
4 |   init S := 0 ;
5 |   trans
6 |     S = 0 |- a -> S := 1;
7 |     S = 1 |- b -> S := 2;
8 |     S = 0 |- b -> S := 3;
9 |     S = 3 |- a -> S := 4;
10 | edon

```

```

Node : 10 : LocalSimul(Test) - Final state component depend on failures order fire in
scenario
MemComb(a, b) => {S=2}

```

```
CurPerm(b, a) => {S=4}
from the following conditions
S = 0
```

- Example 2

```
1 | node Test
2 |   state S : [0,3] ;
3 |   event a; b; c;
4 |   init S := 0 ;
5 |   trans
6 |     S = 0 | - a -> S := 1;
7 |     S = 1 | - b -> S := 3;
8 |     S = 0 | - c -> S := 3;
9 |     S = 0 | - b -> S := 2;
10 | edon
```

```
Node : 10 : LocalSimul(Test) - Final state component depend on failures order fire in
scenario
MemComb(a, b) => {S=3}
CurPerm(b, a?) => {S=2}
from the following conditions
S = 0
```

Two transitions are in conflict at a given time, if they have valid guards and are associated with the same event. Transition in conflict can be detected during a local simulation.

- Transition in conflict

```
1 | node Test
2 |   flow I : [0,4] : in;
3 |   state S : [0,4] ;
4 |   event a; b;
5 |   init S := 0 ;
6 |   trans
7 |     S = 2 | - a -> S := 3;
8 |     S = 3 & I = 0 | - b -> S := 4;
9 |     S = 3 | - b -> S := 1;
10 | edon
```

```
Node : 10 : LocalSimul(Test) - Conflict transition fire : Seq(a, b)
from the following conditions
S = 2
I = 0
```

During the local simulation, a number of error can arise as an affectation except a domain of definition or a division by zero.

- Affectation outside domain of definition included assert

```
1 | node Test
2 |   flow
3 |     Out : [0,4] : out ;
4 |     In : [0,2] : in ;
5 |   state S : [0,3] ;
6 |   event a; b;
7 |   init S := 0 ;
8 |   trans
9 |     S = 0 | - a -> S := 1;
10 |    S = 1 | - b -> S := 3;
11 |    S = 0 | - b -> S := 2;
12 |    S = 2 | - a -> S := 3;
```

```

13 |   assert
14 |     Out = In+S;
15 | edon

```

```

Node : 15 : LocalSimul(Test) - Affectation outside domain of definition
  Out : value=5, domain=[0,4]
from the following conditions
  In = 2
  S = 3

```

- Affectation outside domain of definition included transition's affect

```

1 | node Test
2 |   state
3 |     S : [0,3];
4 |   event a; b;
5 |   init S := 0;
6 |   trans
7 |     S = 0 | - a -> S := 1;
8 |     S = 1 | - b -> S := 4;
9 |     S = 0 | - b -> S := 2;
10 |    S = 2 | - a -> S := 4;
11 | edon

```

```

Node : 11 : LocalSimul(Test) - Affectation outside domain of definition
  S : value=4, domain=[0,3]
after sequence : Seq(a, b)
from the following conditions
  S = 0

```

- Error during the evaluation of an expression (division by zero)

```

1 | node Test
2 |   state
3 |     div : [0,1];
4 |   event dec;
5 |   init div := 1;
6 |   trans
7 |     div > 0 | - dec -> div := div / (div-1);
8 | edon

```

```

Node : 8 : LocalSimul(Test) - Division by zero
after sequence : Seq(dec)
from the following conditions
div = 1

```

## 6. Setting flat of model

To set a model flat is to remove hierarchy in order to use only one component standing for the system.

The principle consists in making instances recursively for sub-components, in order to add flow-variables, states, events, transitions, assertions and external clauses inside current model.

The only difficulty is the synchronizations processing.

Possible error during setting flat:

- Unknown event for a parent node

```

1 | node Flat1
2 |   /* ... */
3 |   state OK:bool;
4 |   event chg;
5 |   trans
6 |     OK      | - chg -> OK := false;
7 |     not(OK) | - chg -> OK := true;
8 |   /* ... */
9 | edon
10 |
11 | node Flat2
12 |   sub U1,U2:Flat1;
13 |   event chg;
14 |   sync <chg, U1.chg, U2.chg>;
15 |   /* ... */
16 | edon
17 |
18 | node Flat3
19 |   sub
20 |     U:Flat1;
21 |     E:Flat2;
22 |   event chg;
23 |   sync <chg, U.chg, E.U1.chg>;
24 |   /* ... */
25 | edon;

```

```

Node : 25 : Flatness Event (E.U1.chg) unknown for node (Flat3)
Node : 25 : Attach data

```

Indeed, model is set flat recursively, the component Flat2 will be set flat before component Flat3.

During Flat2 setting flat, the events U1.chg and U2.chg will be replaced by chg.

During Flat3 setting Flat, during processing of <chg, U.chg, E.U1.chg> synchronization, E.U1.chg event is searched, but it has disappeared and has been replaced by E.chg, that's why there is an error.

- Too complex component - Too high number of transition having to be generated.

```

1 | node Flat1
2 |   /* ... */
3 |   state OK:[0,9];
4 |   event chg;
5 |   trans
6 |     OK = 0 | - chg -> OK := 1;
7 |     OK = 1 | - chg -> OK := 2;
8 |     OK = 2 | - chg -> OK := 3;
9 |     OK = 3 | - chg -> OK := 4;
10 |    OK = 4 | - chg -> OK := 5;
11 |    OK = 5 | - chg -> OK := 6;
12 |    OK = 6 | - chg -> OK := 7;
13 |    OK = 7 | - chg -> OK := 8;
14 |    OK = 8 | - chg -> OK := 9;
15 |    OK = 9 | - chg -> OK := 0;
16 |   /* ... */
17 | edon
18 |
19 | node Flat2
20 |   sub
21 |     U1,U2,U3,U4:Flat1;
22 |   event chg;
23 |   sync <chg, U1.chg, U2.chg, U3.chg, U4.chg>;
24 |   /* ... */
25 | edon;

```

```

Sync : 23 : Flatness Sync (chg) for node (Flat2)

```

```
generate so large number of transition (10000)
Sync : 23 : Attach data
```

## 7. Properties control

Properties control consists in a local properties checking on each component or an overall properties checking on the flat model.

These properties aren't considered as AltaRica errors. Nevertheless, some tools can have difficulties to process AltaRica models having properties like float variables or looped assertions.

This processing displays possible errors/issues.

### 1. Control of 'parameter' external clause

'parameter' external clause allows to define law parameters that can be used in external clauses. These parameters are named either in a overall way or with a clause `<local ID>`.

Overall syntax for this external clause is: `parameter [ID] = [param]; avec [param] ::= [FLOAT] | [ID] | [FCT]([param]+)` where [ID] are identifiers, [FLOAT] is a float and [FCT] is a incertitude-propagation-law (also called propagation-law or incertitude-law) among {lognormal, uniform, normal}.

A parameter is either a float (parameter value), or a name referencing a named parameter, or an incertitude-law. In the last case, parameter of the incertitude-law can't be defined with an incertitude-law.

Possible errors are:

- Syntax error in parameter statement

```
1 | node main
2 |   state OK:bool ;
3 |   event def; rep;
4 |   trans
5 |     OK      |- def -> OK := false;
6 |     not(OK) |- rep -> OK := true;
7 |   init OK := true;
8 |   extern
9 |     parameter {lbd,mu} = 1e-3;
10 |     law <event def> = exponential(lbd);
11 |     law <event rep> = exponential(mu);
12 | edon;
```

```
ExternParameter : 9 : file : Syntax error for 'parameter' clause :
parameter [id] = [param];
=> parameter {lbd, mu} = 0.001
```

- Syntax error in parameter definition

```
1 | node main
2 |   state OK:bool ;
3 |   event def ;
4 |   trans OK |- def -> OK := false;
5 |   init OK := true;
6 |   extern
7 |     parameter lbd = "1e-3";
8 |     law <event def> = exponential(lbd);
9 | edon;
```

```
ExternParameter : 7 : file : Syntax error for term [param] used in 'law' or 'parameter'
clause :
[param] ::= [float] | [id] | fct([param]+);
```



```
=> parameter lbd = "1e-3"
```

- Unknown incertitude-function

```
1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   init OK := true;
6 |   extern
7 |     parameter lbd = lognormale(1e-3, 3);
8 |     law <event def> = exponential(lbd);
9 |   edon;
```

```
ExternParameter : 7 : file : Unknown propagation function for [param] () :
fct = {lognormal|uniform|normal}
=> parameter lbd = lognormale(0.001, 3)
```

- Recursive incertitude-function

```
1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   init OK := true;
6 |   extern
7 |     parameter lbd = lognormal(uniform(0.8e-3, 1.2e-3), 3);
8 |     law <event def> = exponential(lbd);
9 |   edon;
```

```
ExternParameter : 7 : file : Recursive propagation functions are forbidden ...
=> parameter lbd = lognormal(uniform(8.0E-4, 0.0012), 3)
```

## 2. Control of 'law' external clause

The 'law' external clause allows to define delay and/or probability laws associated with events of model.

Overall syntax for this external clause is: `law <event [ID-EVT]> = [FCT]([param]+);` where [ID-EVT] is an event identifier and [FCT] is a law recognized by Aralia and/or Moca12 (for [param], e.g. previous paragraph)

Possible errors:

- Syntax error in law definition

```
1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   init OK := true;
6 |   extern
7 |     law <event def> = 1e-3;
8 |   edon;
```

```
ExternLaw : 7 : file : Syntax error for 'law' clause :
law <event [id]> = fct([param]+);
=> law <event def> = 0.001
```

- Unknown law

```

1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   init  OK := true;
6 |   extern
7 |     law <event def> = dirac(1e-3);
8 | edon;

```

```

ExternLaw : 7 : file : Unknown function for 'law' clause : must be a known law for Aralia
or Mocal2 compute engine
=> law <event def> = dirac(0.001)

```

- Managed Aralia laws are: exponential, constant, Weibull, Dirac, GLM, asymptotic\_exponential, periodic\_test.

```

1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   init  OK := true;
6 |   extern
7 |     law <event def> = ifa(10, 100);
8 | edon;

```

```

ExternLawAralia : 7 : file : Unknown law for Aralia
=> law <event def> = ifa(10, 100)

```

- Managed Mocal2 laws are: exponential, constant, Weibull, Dirac, ifa (planned instant), nlog, unif.

```

1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   init  OK := true;
6 |   extern
7 |     law <event def> = GLM(0, 1e-3, 1e-2);
8 | edon;

```

```

ExternLawMoca : 7 : file : Unknown law for Mocal2
=> law <event def> = GLM(0, 0.001, 0.01)

```

- Number of parameters is defines for each law.

```

1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   init  OK := true;
6 |   extern
7 |     law <event def> = exponential(1e-3, 1e-2);
8 | edon;

```

```

ExternParameter : 7 : file : The number of law parameters isn't correct.
=> law <event def> = exponential(0.001, 0.01)

```

### 3. Control of 'attribute' external clause

The 'attribute' external clause allows to associate attributes to events.

Overall syntax for this external clause is: `attribute [ID-ATTR](<event [ID-EVT]>) = [value];` where [ID-ATTR] is name of attribute, [ID-EVT] an event identifier and [value] is value of the attribute for the considered event.

Possible errors:

- Syntax error

```

1 | node main
2 |   state OK:bool ;
3 |   event def;
4 |   trans OK | - def -> OK := false;
5 |   init  OK := true;
6 |   extern
7 |     law <event def> = exponential(1e-3);
8 |     attribute Type(<event def>) = pompe(2);
9 | edon;

```

```

ExternAttribute : 8 : file : Syntax error for 'attribute' clause : [value]
attribute [name](<event [id]>) = [value];
=> attribute Type(<event def>) = pompe(2)

```

### 4. Control of 'nodeproperty' external clause

The 'nodeproperty' external clause allows to associate properties to components (nodes).

Overall syntax for this external clause is: `nodeproperty [ID] = [value];` where [ID] is name of property and [value] is value of the property.

```

1 | node main
2 |   state OK:bool ;
3 |   extern
4 |     nodeproperty date = format("2007/06/12");
5 | edon;

```

```

ExternNodeProperty : 4 : file : Syntax error for 'nodeproperty' clause :
nodeproperty [id] = [value];
=> nodeproperty date = format("2007/06/12")

```

### 5. Control of 'priority' external clause

The 'priority' external clause allows to define priority between events.

Overall syntax for this clause is: `priority <event [ID-EVT]> = [INT];` where [ID-EVT] is an event identifier and [INT] a non negative integer specifying priority level (the higher the number, the higher the event priority).

Possible errors:

- syntax error

```

1 | node main
2 |   flow In:bool:in;

```

```

3 | state Mem:bool ;
4 | event chg;
5 | trans Mem!=In |- chg -> Mem := In;
6 | init Mem := true;
7 | extern
8 |     law <event chg> = Dirac(0);
9 |     priority <event chg> = High;
10 | edon;

```

```

ExternPriority : 9 : file : Syntax error for 'priority' clause :
priority <event [id]> = [integer];
=> priority <event chg> = High

```

- Only instantaneous events can have priority.

```

1 | node main
2 |     flow In:bool:in;
3 |     state Mem:bool ;
4 |     event chg;
5 |     trans Mem!=In |- chg -> Mem := In;
6 |     init Mem := true;
7 |     extern
8 |         law <event chg> = exponential(0.001);
9 |         priority <event chg> = 1;
10 | edon;

```

```

ExternPriorityDirac : 8 : file : A untemporised event (chg) can not have a priority.
=> law <event chg> = exponential(0.001)

```

## 6. Control of 'remark' external clause

The 'remark' external clause allows to document events (flow variables, state variables events, sub-components, local parameters) of an Altarica model.

Overall syntax for this clause is: `remark [OBJ] = [STRING];` where [OBJ] is an element of the model. (`<event [ID]>|<flow [ID]>|<state [ID]>|<sub [ID]>|<local [ID]>`) and [STRING] is string (between quotation marks).

Possible errors:

- Syntax error

```

1 | node main
2 |     state OK:bool ;
3 |     event def;
4 |     trans OK |- def -> OK := false;
5 |     extern
6 |         remark <event def> = defaillance;
7 | edon;

```

```

ExternRemark : 6 : file : Syntax error for 'remark' clause :
remark [obj] = "<String>"; with [obj] ::= <event [id]>|<flow [id]>|<state [id]>|<sub
[id]>|<local [id]>
=> remark <event def> = defaillance

```

## 7. Control of 'preemptible' external clause

The 'preemptible' external clause allows to define events such as preemptible events.

Overall syntax for this clause is: preemptible { (<event [ID-EVT]>)+ }; where [ID-EVT] are events.

```

1 | node main
2 |   flow Call:bool:in;
3 |   state St:{OK,KO,SB} ;
4 |   event Def; Rep; SOK; SSB;
5 |   trans
6 |     St=OK          | - Def -> St := KO;
7 |     St=KO          | - Rep -> St := SB;
8 |     St=SB & Call    | - SOK -> St := OK;
9 |     St=OK & ~Call   | - SSB -> St := SB;
10 |  extern
11 |    law <event SOK> = Dirac(0);
12 |    law <event SSB> = Dirac(0);
13 |    preemptible {<event Rep>} = false;
14 |  edon;

```

```

ExternPreemptible : 13 : file : Syntax error for 'preemptible' clause :
  preemptible '{' (<event [id]>)+ '}'
=> preemptible {<event Rep>} = false

```

## 8. Control of 'bucket' external clause

The 'bucket' external clause allows to associate events in order to consider that there are disjunctive in probability (can't appear at the same time). It allows to imitate trigger with realistic solicitation.

Overall syntax for this clause is: bucket { (<event [ID-EVT]>)+ }; where [ID-EVT] are events.

Possible errors:

- Syntax error

```

1 | node main
2 |   flow Call:bool:in;
3 |   state St:{OK,KO,SB} ;
4 |   event Def; Rep; SOK; SKO; SSB;
5 |   trans
6 |     St=OK          | - Def -> St := KO;
7 |     St=KO          | - Rep -> St := SB;
8 |     St=SB & Call    | - SOK -> St := OK;
9 |     St=SB & Call    | - SKO -> St := KO;
10 |    St=OK & ~Call   | - SSB -> St := SB;
11 |  extern
12 |    law <event Def> = exponential(0.001);
13 |    law <event Rep> = exponential(0.01);
14 |    law <event SOK> = constant(0.98);
15 |    law <event SKO> = constant(0.02);
16 |    law <event SSB> = Dirac(0);
17 |    bucket {<event SOK>, <event SKO>} = false;
18 |  edon;

```

```

ExternBucket : 17 : file : Syntax error for 'bucket' clause :
  bucket '{' (<event [id]>)+ '}'
=> bucket {<event SOK>, <event SKO>} = false

```

- Each event must be associated with one and only one transition.

```

1 | node main
2 |   flow Call:bool:in;
3 |   state St:{OK,KO,SB} ;
4 |   event Def; Rep; SOK; SKO;

```

```

5 | trans
6 |   St=OK          |- Def -> St := KO;
7 |   St=KO          |- Rep -> St := SB;
8 |   St=SB & Call   |- SOK -> St := OK;
9 |   St=SB & Call   |- SKO -> St := KO;
10 |  St=OK & ~Call  |- SOK -> St := SB;
11 | extern
12 |   law <event Def> = exponential(0.001);
13 |   law <event Rep> = exponential(0.01);
14 |   law <event SOK> = constant(0.98);
15 |   law <event SKO> = constant(0.02);
16 |   bucket {<event SOK>, <event SKO>} = true;
17 | edon;

```

ExternBucket : 16 : file : The event (SOK) is used at different transition.  
=> bucket {<event SOK>, <event SKO>} = true

- The guards of transition must be equal.

```

1 | node main
2 |   flow Call:bool:in;
3 |   state St:{OK,KO,SB} ;
4 |   event Def; Rep; SOK; SKO; SSB;
5 |   trans
6 |     St=OK          |- Def -> St := KO;
7 |     St=KO          |- Rep -> St := SB;
8 |     St=SB & Call   |- SOK -> St := OK;
9 |     St=KO & Call   |- SKO -> St := KO;
10 |    St=OK & ~Call  |- SSB -> St := SB;
11 |   extern
12 |     law <event Def> = exponential(0.001);
13 |     law <event Rep> = exponential(0.01);
14 |     law <event SOK> = constant(0.98);
15 |     law <event SKO> = constant(0.02);
16 |     bucket {<event SOK>, <event SKO>} = true;
17 | edon;

```

[warning] : 9 : file : Transition guards of events (SOK, SKO) are not equal (exactly).  
=> ((St = KO) and Call) |- SKO -> St := KO

- Law associated to each event must be a constant law with parameter between 0 and 1.

```

1 | node main
2 |   flow Call:bool:in;
3 |   state St:{OK,KO,SB} ;
4 |   event Def; Rep; SOK; SKO; SSB;
5 |   trans
6 |     St=OK          |- Def -> St := KO;
7 |     St=KO          |- Rep -> St := SB;
8 |     St=SB & Call   |- SOK -> St := OK;
9 |     St=SB & Call   |- SKO -> St := KO;
10 |    St=OK & ~Call  |- SSB -> St := SB;
11 |   extern
12 |     law <event Def> = exponential(0.001);
13 |     law <event Rep> = exponential(0.01);
14 |     law <event SOK> = constant(2);
15 |     law <event SKO> = constant(0.02);
16 |     law <event SSB> = Dirac(0);
17 |     bucket {<event SOK>, <event SKO>} = true;
18 | edon;

```

ExternBucket : 8 : file : Gamma not between 0 and 1 : Event (SOK)  
=> ((St = SB) and Call) |- SOK -> St := OK

- The sum of parameters for event-laws must be equal to 1.

```

1 | node main
2 |   flow Call:bool:in;
3 |   state St:{OK,KO,SB} ;
4 |   event Def; Rep; SOK; SKO; SSB;
5 |   trans
6 |     St=OK      | - Def -> St := KO;
7 |     St=KO      | - Rep -> St := SB;
8 |     St=SB & Call | - SOK -> St := OK;
9 |     St=SB & Call | - SKO -> St := KO;
10 |    St=OK & ~Call | - SSB -> St := SB;
11 |   extern
12 |     law <event Def> = exponential(0.001);
13 |     law <event Rep> = exponential(0.01);
14 |     law <event SOK> = constant(0.998);
15 |     law <event SKO> = constant(0.02);
16 |     law <event SSB> = Dirac(0);
17 |     bucket {<event SOK>, <event SKO>} = true;
18 |   edon;

```

```

ExternBucket : 17 : file : Sum of gamma not equal 1.
=> bucket {<event SOK>, <event SKO>} = true

```

## 9. Control of 'observer' external clause

The 'observer' external clause allowed to define statistic observers that can be use with Combava stochastic simulator and with Mocal2.

The 'property' and 'predicate' external clauses replace it.

A 'deprecated' message is display if this clause is used.

```

1 | node main
2 |   state St:int;
3 |   event chg;
4 |   trans
5 |     St >= 0 | - chg -> St := St*4/5;
6 |   init St := 100;
7 |   extern
8 |     observer EndValueOfSt = <term (St)>;
9 |   edon;

```

```

ExternObserver : 8 : file : 'observer' clause is deprecated.
=> observer EndValueOfSt = <term (St)>

```

## 10. Control of 'predicate' external clause

The 'predicate' external clause allows to define boolean observers that can be use with Combava tools and with Mocal2.

Right syntax for this external clause is: predicate [ID] = <term ([boolean-term])> where [ID] is an identifier and [boolean-term] is an AltaRica boolean expression between brackets.

An error is display if expression is not boolean.

```

1 | node main
2 |   state cpt:int ;

```

```

3 |   extern
4 |     predicate failed = <term (cpt*2)>;
5 |   edon;

```

```

ExternPredicate : 4 : file : Syntax error for 'predicate' clause :
  predicate [id] = <term ([boolean-term])>;
    => predicate failed = <term ((cpt * 2))>

```

## 11. Control of 'property' external clause

The 'property' external clause allows to define numeric (integer ou real) observers that can be use with Combava tools and with Mocal2.

Right syntax for this external clause is: `predicate [ID] = <term ([numeric-term])>` where [ID] is an identifier and [numeric-term] is an AltaRica numeric expression between brackets.

An error is display if expression is not numeric.

```

1 | node main
2 |   state S : {ok, ko, hs} ;
3 |   extern
4 |     property failed = <term (S)>;
5 |   edon;

```

```

ExternProperty : 4 : file : Syntax error for 'property' clause :
  property [id] = <term ([numeric-term])>;
    => property failed = <term (S)>

```

## 12. Other controls

Instead of `<event [ID-EVT]>`, there usually can be a list of events. External clause is defined for every event in the list. In this case, the list mustn't be empty and mustn't have double (two times the same event).

List of possible errors in a definition of set of events.

- Empty list

```

1 | node main
2 |   state OK:bool;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   extern
6 |     law def = exponential(0.001);
7 |   edon;

```

```

ExternLaw : 6 : file : No define event for current clause
    => law def = exponential(0.001)

```

- List with double

```

1 | node main
2 |   state OK:bool;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 |   extern
6 |     law {<event def>, <event def>} = exponential(0.001);

```



```
7 | edon;
```

```
[warning] : 6 : file : Clause set with redefine event.
=> law {<event def>, <event def>} = exponential(0.001)
```

AltaRica event are either instantaneous or temporized or stochastic. Tools may not manage instantaneous or temporized events.

- Some tools may not manage instantaneous events.

```
1 | node main
2 |   flow In:bool:in;
3 |   state OK:bool;
4 |   event chg;
5 |   trans OK & In |- chg -> OK := false;
6 |   extern law <event chg > = Dirac(0);
7 | edon;
```

```
EventInstantaneous : 6 : file : Event (chg) has instantaneous (Dirac(0)).
=> law <event chg> = Dirac(0)
```

- Some tools may not manage temporized events.

```
1 | node main
2 |   flow In:bool:in;
3 |   state OK:bool;
4 |   event chg;
5 |   trans OK & In |- chg -> OK := false;
6 |   extern law <event chg > = Dirac(10);
7 | edon;
```

```
EventTemporised : 6 : file : Event (chg) has time delay (Dirac(x)).
=> law <event chg> = Dirac(10)
```

- Some tools consider that events without law are instantaneous events. Existence of events without law must be checked.

```
1 | node main
2 |   state OK:bool;
3 |   event def;
4 |   trans OK |- def -> OK := false;
5 | edon;
```

```
EventLaw : 4 : file : Event (def) has nothing define law.
=> def
```

Possible errors on guards and transitions:

- FaultTree generation with inference engine isn't safe when there are flows in guard of a transition.

```
1 | node main
2 |   flow In:bool:in;
3 |   state OK:bool;
4 |   event def;
```

```
5 |   trans OK&In |- def -> OK := false;
6 | edon;
```

GuardWithFlow : 5 : file : Guard of transition with flow variable (In).  
=> (OK and In) |- def -> OK := false

- Besides this particular cases, it is inadvisable to have transitions that are always valid. In the case below, when failure happens, event must be no more fireable. The guard of this transition have to be modified.

```
1 | node main
2 |   state OK:bool;
3 |   event def;
4 |   trans true |- def -> OK := false;
5 | edon;
```

GuardTrue : 4 : file : Always valid transition (guard always true)  
=> true |- def -> OK := false

- Two transitions - having equivalent guards and associated with the same event - may be in conflict.

Because they have equivalent guards, they always be valid in the same time. Because they are associated with the same event, they will be fireable in the same time.

Currently, two guards are considered as equivalent if they are strictly equal (same order in arguments of operators)  $A+B+C$  doesn't equals  $C+B+A$ .

```
1 | node main
2 |   state Etat:{Ouvert,Ferme};
3 |   flow CC:bool:in;
4 |   event chg;
5 |   trans
6 |     Etat=Ouvert & CC |- chg -> Etat:=Ferme;
7 |     Etat=Ouvert & CC |- chg -> Etat:=Ouvert;
8 | edon
```

TransConflict : 7 : file : Warring transitions (same guard, same event) for (chg) event.  
=> ((Etat = Ouvert) and CC) |- chg -> Etat := Ouvert

Some type of synchronization are not fully compatible with some tools. For example, tree generation of type: inference engine, works properly only with CCF synchronization. In addition, it is possible to use synchronization (other CCF type) with events belong to same sub component. This synchronization can generate affectation conflict (Two transition who affect some state variable with different value).

- Presence of Synchronization of type "synchronization".

```
1 | node Unit
2 |   event def;
3 |   state OK:bool;
4 |   init OK := true;
5 |   trans
6 |     OK |- def -> OK := false;
7 | edon;
8 |
9 | node main
10 |   sub A,B:Unit;
11 |   event synk;
12 |   sync <synk, A.def, B.def> ;
```

```
13 | edon;
```

```
SyncSync : 12 : file : Synchronization type of synk is Synchronization.
=> <synk , A.def , B.def>
```

- Presence of synchronization of type Diffusion (BroadCast)

```
1 | node Unit
2 |   event def;
3 |   state OK:bool;
4 |   init OK := true;
5 |   trans
6 |     OK |- def -> OK := false;
7 |   edon;
8 |
9 | node main
10 |   sub A,B:Unit;
11 |   event synk;
12 |   sync <synk | A.def | B.def> ;
13 | edon;
```

```
SyncDiff : 12 : file : Synchronization type of synk is Diffusion (BroadCast).
=> <synk : A.def or B.def>
```

- Presence of synchronization of type CCF (Common Cause Failure)

```
1 | node Unit
2 |   event def;
3 |   state OK:bool;
4 |   init OK := true;
5 |   trans
6 |     OK |- def -> OK := false;
7 |   edon;
8 |
9 | node main
10 |   sub A,B:Unit;
11 |   event synk;
12 |   sync <synk ? A.def ? B.def> ;
13 | edon;
```

```
SyncCCF : 12 : file : Synchronization type of synk is CCF (Common Cause Failure).
=> <synk : A.def or B.def>
```

- Presence of synchronization with events belong to same sub component

```
1 | node Unit
2 |   event chg1; chg2;
3 |   flow I:bool:in;
4 |   state
5 |     OK:bool;
6 |     Mem:[0,2];
7 |   init
8 |     OK := true;
9 |     Mem := 0;
10 |   trans
11 |     OK & I |- chg1 -> OK := false, Mem := 1;
12 |     OK & ~I |- chg2 -> OK := false, Mem := 2;
13 |   edon;
14 |
15 | node main
```

```

16 |   sub A,B:Unit;
17 |   event synk;
18 |   sync <synk | A.chg1 | B.chg1 | A.chg2 | B.chg2> ;
19 | edon;

```

```

SyncSomeSub : 18 : file : Synchronization synk with events (A.chg1, A.chg2) belong to
same sub component.
=> <synk : A.chg1 or B.chg1 or A.chg2 or B.chg2>

```

Most step by step simulator and some tools can't manage systems with loops in their assertions.

- Loop presence in assertions

```

1 | node Unit
2 |   flow
3 |     Out:bool:out;
4 |     In :bool:in;
5 |   event def;
6 |   state OK:bool;
7 |   init OK := true;
8 |   trans
9 |     OK |- def -> OK := false;
10 |   assert
11 |     Out = (if OK then In else false);
12 | edon;
13 |
14 | node main
15 |   sub A,B:Unit;
16 |   assert
17 |     B.In = A.Out;
18 |     A.In = B.Out;
19 | edon;

```

```

Loop : 3 : file=>Instance : Loop assert : A.Out [ B.In ]
  <= B.Out [ A.In ]
  <= A.Out [ B.In ]
=> A.Out:bool:out

```

Altatica code entered by user can be too complex to be generated into a compilable Java language. Actually, in some cases, the generated java code contains too large methods, so Java compiler can not manage compilation.

- The generated java code contains too large methods, so Java compiler can not manage compilation.

```

node complex
  flow
    Ssw:bool:in;
    FmAct:bool:in;
    DefRv:bool:in;
    In1:bool:in;
    In2:bool:in;
    In3:bool:in;
    In4:bool:in;
    Val1:bool:in;
    Val2:bool:in;
    Val3:bool:in;
    Val4:bool:in;
    Rv:bool:out;
  assert
    Rv = case {
      (Val1 and Val2 and Val3 and Val4 and In1 and In2 and In3 and In4) : true,
      (Val1 and Val2 and Val3 and Val4 and (!In1) and In2 and In3 and In4) : false,
      ...
      /* More line */
      ...
    }

```

```

    (Val1 and Val2 and Val3 and Val4 and not In1 and In2 and (!In3) and (!In4)) : true,
  else DefRv
};
edon

```

```

GenerateJava1 : 103 : file : Likely error during Java compilation
The 'assert' AltaRica Code of 'complex' component is too big.
=> node complex
    ...
edon

```

Currently, float/integer variables and some operators are not always supported by Altarica model processing tools.

- Integer variable presence

```

1 | node Expr
2 |   flow
3 |     Out:int:out;
4 |   event def;
5 |   state Prod:int;
6 |   init Prod := 100;
7 |   trans
8 |     Prod>0 |- def -> Prod := 0;
9 |   assert
10 |     Out = Prod;
11 | edon;

```

```

ExprInt : 3 : file : Variable with integer domain : Out
=> Out:int:out
ExprInt : 5 : file : Variable with integer domain : Prod
=> Prod:int

```

- Float variable presence

```

1 | node Expr
2 |   flow
3 |     Out:float:out;
4 |   event def;
5 |   state Prod:float;
6 |   init Prod := 100;
7 |   trans
8 |     Prod>0 |- def -> Prod := 0;
9 |   assert
10 |     Out = Prod;
11 | edon;

```

```

ExprFloat : 3 : file : Variable with float domain : Out
=> Out:float:out
ExprFloat : 5 : file : Variable with float domain : Prod
=> Prod:float

```

- Presence of unwanted operators

```

1 | node KOf3
2 |   flow
3 |     Out:bool:out;
4 |     In1,In2,In3:bool:in;
5 |   state K:[1,3];
6 |   init K := 2;

```

```
7 |   assert
8 |     Out = #(In1,In2,In3)>=K;
9 |   edon;
```

```
ExprCrd : 8 : file : Operator of type : #(...)
=> #(In1, In2, In3)
```