2, rue du Marché - 92160 Antony
tel and fax: 01 46 74 54 10, e-mail: ariste@wanadoo.fr

FIGAROIDE USER MANUAL

	Name	Visa	Date
Editor	Ms. Anne Flori		December 20, 2021
Approver	Mr. Jean Christophe Houdebine		December 20, 2021

TRACK CHANGES TABLE

Version	Date	Description of the amendment	Written by	Approved by
V1.0	03/05/16	First version	Ms A Flori	Mr. Jean Christophe Houdebine
V2.0	29/06/16	Version adapted to FigaroIDE version 1.0.1.2	Ms A Flori	Mr. Jean Christophe Houdebine
V2.1	13/02/18	Improvements in or relating to detail	Mr. Bouissou	
V3.0	29/10/21	Version adapted to version 1.0.3.1	Mr. Houdebine	Ms A Flori

SOMMARY

1 INTRODUCTION

This manual follows the logic of building a knowledge base. The operation described here corresponds to version 1.0.3.1 of FigaroIDE. This guide is structured around chapters presenting successively:

- basic concepts and terminology,
- the principles of file storage,
- the launch of FigaroIDE,
- the general structure of the interface,
- the creation of a new knowledge base,
- writing the knowledge base,
- the creation of the KB3 GUI settings file,
- verification of the writing of the knowledge base,

2 REFERENCE DOCUMENTS

[1] KB3 V3 User Manual, H-T52-2012-01065-EN, EDF, September 17, 2012.

[2] Figaro probabilistic modeling language reference manual, EDF6125-1612-2017-00624-EN, April 2017.

[3] Syntax of the Figaro probabilistic modeling language, EDF 6125-1612-2016-15549-EN, October 2016.

3 ACRONYMS

KB	Knowledge Base
UE	Undesirable Event
IDE	Integrated Development Environment
GUI	Graphical User Interface
GMC	Graphical Macro Components
DO	Deducted Object
PC	PreCompilation
MR	Manual Rule
XML	Extensible Markup Language

4 BASIC CONCEPTS AND TERMINOLOGY

In order to perform a safety study of a system with KB3[1] a knowledge base dedicated to the study of this type of system is required. The knowledge bases (KBs) are written in FIGARO, a specific language developed at EDF[2][3].

FigaroIDE is a tool to help develop and verify knowledge bases.

5 FILES OF A KNOWLEDGE BASE

A KB is described by a set of several files:

- a .fi file describing in Figaro language the classes of objects associated with the category of

systems modeled in the KB,

- a .bdc file describing in XML language the parameters of the graphical interface of the database in KB3 V3 and thus consisting of the graphical representations associated with each object, the authorized processing modes, the color changes during simulations (graphical variants of the same object)...
- an "icons" directory [1]containing .ico files to represent the objects of the KB3 V3 palette and associated .sym files with the same name representing the same objects on the KB3 V3 system diagram. An .ico file may not be associated with a .sym file; this is the case if it is associated with a link or a so-called non-graphical object (not displayable on the system scheme in KB3 V3). An .ico file can be associated with several .sym files corresponding to the graphical variants of the object when certain conditions are met (for example, to locate a faulty object).

6 LAUNCH OF FIGAROIDE

FigaroIDE can be launched by double-clicking on the executable itself or a shortcut pointing to it.

7 GENERAL STRUCTURE OF THE INTERFACE

The FigaroIDE GUI consists of a main window with the following elements (see Figure 1):

- a menu bar (cf. 7.1),
- a toolbar with some of the main functions of the menus and identified by icons,
- a set of widgets (dockable panels) (cf. 7.2),
- a workspace in which the Figaro editor (.fi file editor) and the XML file window associated with the knowledge base are displayed.

[1] This directory must be named "icons" or the same name as the .fi file without the extension to be readable in KB3 V3 and be located next to the .bdc and .fi extension files.

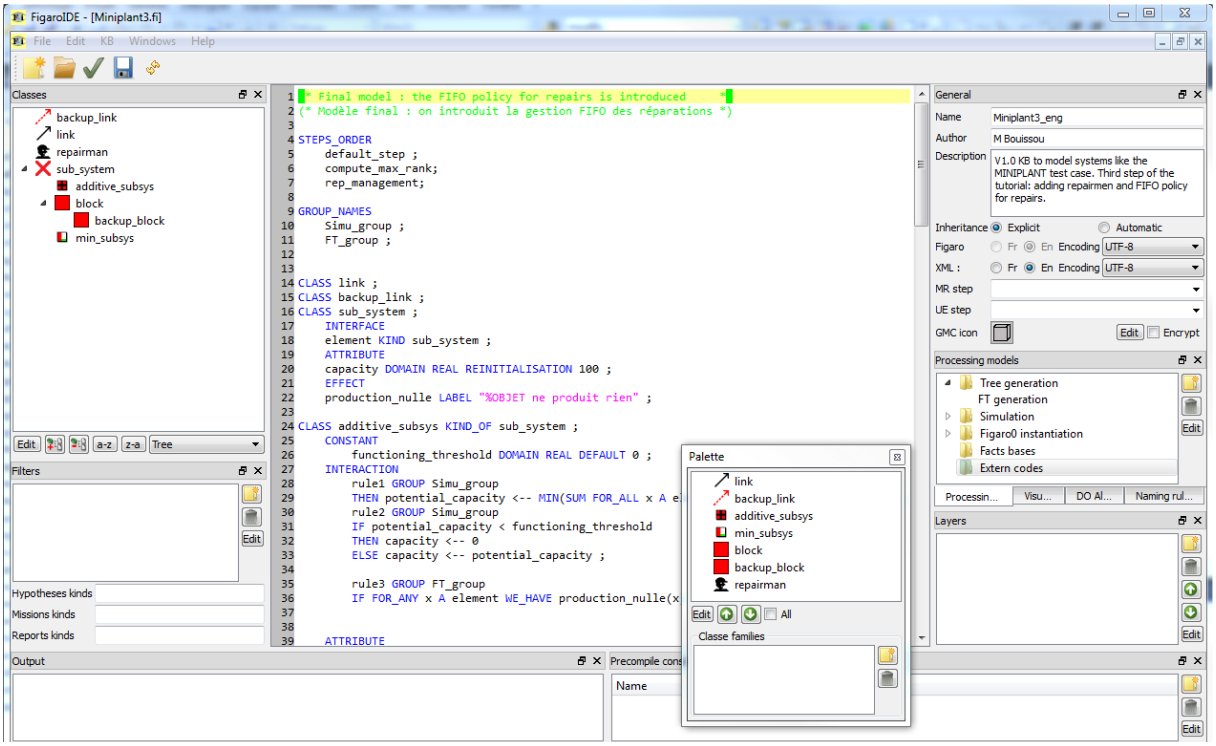


Figure 1: Overview of the FigaroIDE main interface

7.1 MENUS

The menus and their main functions are as follows:

- The File menu allows a set of management actions on a KB (creation, opening, closing, saving) and to exit FigaroIDE,
- The Edit menu allows you to perform the classic editing operations (copy, cut, paste), to access the search tool (Search) and to define some editing parameters (Options...),
- the KB menu allows certain actions to be performed on the KB opened in the workspace (Verify: cf. 12, Synchronize allows you to make the .fi and .xml files of the database consistent with the modifications made from the widgets, Precompile: see § 11.14, Translate...: see §. 14, Inherit allows to display in the workspace the "inherited KB" showing for each type the inherited interfaces, constants, attributes, occurrences…),
- The Windows menu offers different presentations of the files open in the workspace (Mosaic, Cascade or Tabs) and presents the list of open files. The name of the checked file corresponds to the active file in the workspace. Simply check another name in the list to make it active.

The right click of the mouse gives access to contextual menus regrouping a set of active commands on the current selection.

7.2 WIDGETS

There are 10 widgets:

- The Classes, DO Algorithms, General, Palette, Visualizations, Processing Templates, Precompilation Constants, Naming Rule Groups, Filters and Layers widgets are intended to help the user build the xml file with the .bdc extension that will define the parameters of

the KB graphical interface in KB3 V3.

- The Classes widget also provides quick search tools by sorting and selecting on types and a hierarchical presentation of types (presentation of inheritance between types).
- Finally, the Output widget allows you to view the writing errors in the KB following a grammatical and syntactic check of the Figaro file and the inconsistencies in the .bdc file (cf. 12).

The user chooses which widgets to display by right-clicking on the menu bar, which brings up the list of widgets. The displayed widgets are checked in the list. Simply uncheck them to remove them from the list.

The user can resize the widgets and place them anywhere on the screen.

7.2.1 WIDGET TYPES

The Classes widget allows you to view all the classes defined in the KB and to access certain operations that can be performed on the classes.

A right click of the mouse on one of the classes in the list displays the actions to be performed. The Select fi and Select xml commands lead to the display of the class in the .fi file and in the .xml file respectively.

The Edit command causes the opening of the window of the selected class in which is presented all the properties of the class: description (text), graphic instance whose icon if it exists is presented, the graphic variants and the points of connection to the object (cf. 11.1, § 11.2).

7.2.2 DO ALGORITHMS

The DO Algorithms widget provides the tools necessary to create deduced objects (DO) (cf. 11.12).

7.2.3 GENERAL

The General widget allows you to define the name, author, description and language of the KB, and to specify:

- the language of the Figaro keywords cannot be chosen; it is deduced from the parsing of the Figaro file
- the desired encodings for .fi and .bdc files (it is recommended to use the UTF-8+BOM encoding, which is automatically recognized by most text editors and allows to transfer the files without change from a Windows environment to Linux)
- the language of the tags in the .bdc file,
- the graphical representation of the GMC (see § 11.7),
- the Figaro step in which the manual rules are placed (see § 11.10),
- the Figaro step in which the undesirable events are placed (see § 11.11).

7.2.4 PALETTE

The Palette widget presents the classes that will appear in the KB3 palette in the order defined by the user (see § 11.2). When All is unchecked, only the types appearing in the palette are displayed. All allows to see all classes declared as node or link although they do not appear in the KB3 palette.

As with the Classes widget, it is possible to access certain operations that can be performed on types by selecting a class and clicking on the Edit button. The Edit command opens the window of the selected class in which all the properties of the class are presented: description (text), graphical instance whose icon, if it exists, is presented, graphical variants and connection points to the object (cf. 11.1, § 11.2).

The Palette widget can also be used to specify class families (see § 11.6).

7.2.5 VISUALIZATIONS

The Visualizations widget allows you to specify the different visualizations proposed by the knowledge base (see §1111.7).

7.2.6 TREATMENT MODELS

The Processing Templates widget allows you to define templates for the processes that will be performed in KB3 (see § 11.13). 11.13).

7.2.7 PRE-COMPILATION CONSTANTS

The Precompilation constants widget allows you to define precompilation constants that will modify the operation of the KB in KB3 (cf. 11.14).

7.2.8 FILTERS

This widget allows you to build the search filters that form the predefined reports proposed by the knowledge base.

7.2.9 NAMING RULE GROUPS

The Naming Rule Groups widget allows you to define rule groups to modify the names and descriptions of fault tree gates to suit the quantization software used.

7.2.10 LAYERS

The purpose of this widget is to build layers that allow to filter the display of objects instances of the classes proposed by the knowledge base and to define a page background image in KB3.

7.2.11 OUTPUT

The Output widget is a window for viewing errors contained in the KB (see § 12).

8 OPEN A KNOWLEDGE BASE

The Open... command in the File menu allows you to access the directory hierarchy and select the knowledge base of interest. Selecting either the .fi or .bdc file will open both files in the workspace.

The Recent command allows you to have direct access to the directory of the last opened databases.

9 CREATION OF A NEW KNOWLEDGE BASE

Selecting the New command from the File menu creates two new pages .fi and .bdc in the workspace. The corresponding files have not been created and will not be created until the user saves the database (Save or Save As... command from the File menu).

***Warning**: the .bdc file will only be saved if a name is given to the knowledge base.*

10 WRITING THE FIGARO KNOWLEDGE BASE

For an overview of the concepts and syntax of the FIGARO language, the user may refer to the "FIGARO Language Reference Manual [2][3].

The user writes the base in the text editor corresponding to the file with extension .fi.

FigaroIDE offers several writing aids for the database[2]:

- autocomplete,
- colorization :

  keywords of the Figaro language (blue)

  enumerated values of constants and attributes (dark green)

  description and label strings (pink)

  comments (light green).
- tools for entering the main elements of a knowledge base: see below.

10.1 COLORIZATION

Below is an example of knowledge base content in Figaro language.

```
KB_DESCRIPTION "Small educational knowledge base.
 It is used to describe telecommunication networks
   meshed with failures on nodes and links.
   It has two possible uses: simulation (with the
   group_simu rules) and generation of
   failures (with rule group_add)";
(* Author: M. Bouissou, 20 Dec 2007 *)
GROUP_NAMES
group_simu ;
group_add ;

CLASS noeud GR_NOEUD ;
CONSTANT
function DOMAIN 'source' 'purpose' 'intermediate'DEFAULT 'intermediate' ;
lambda DOMAIN REAL DEFAULT 1e-5 ROLE DESIGN;
mu DOMAIN REAL DEFAULT 0.1 ROLE DESIGN;
EFFECT
linked ;
OCCURRENCE
group_simu
MAY_OCCUR
DEFAULT def
DIST EXP(lambda);
INTERACTION
rule1
IF WORKING AND function = 'source
THEN connect;
FAILURE def
RELIABILITY DATA
GROUP add group
MODEL_GLM
```

[2] Keywords can be in French or English. FigaroIDE recognizes the language automatically and proposes completions in the language of the database.

```
GAMMA 0.
LAMBDA lambda
MU mu ;

CLASS Source KIND_OF noeud;
CONSTANT
function DEFAULT 'source';

CLASS but EXTENDS noeud;
CONSTANT
function DEFAULT 'goal';

CLASS link GR_LIEN;
CONSTANT
lambda_arete DOMAIN REAL DEFAULT 1e-5 ROLE DESIGN ;
mu_arete DOMAIN REAL DEFAULT 1 ROLE DESIGN ;
FAILURE
outage
RELIABILITY DATA
GROUP add group
MODEL_GLM
GAMMA 0.
LAMBDA lambda_arete
MU mu_arete ;
OCCURRENCE
group_simu
MAY_OCCURE
FAILURE cut-off
DIST EXP(lambda_arete);

CLASS arete_uni_dir EXTENDS link ;
INTERFACE
start KIND node CARDINAL 1;
arrival KIND noeud CARDINAL 1 ;
INTERACTION
rule1
IF WORKING AND linked(start) AND WORKING(arrival)
THEN linked(arrival) ;

CLASS arete_bi_dir EXTENDS link ;
INTERFACE
extremity KIND node CARDINAL 2 ;
INTERACTION
rule1
IF WORKING AND
(FOR_ANY x AN extremity WE_HAVE WORKING(x)) AND
THERE_EXISTS x AN extremity SUCH_AS linked(x)
THEN FOR_ALL z AN extremity DO linked(z);

CLASS counter_def ;
ATTRIBUTE
nb_def DOMAIN ENTEGER DEFAULT 0;
INTERACTION
rule1
GROUP simu
THEN nb_def <-- SUM FOR_ALL x AN OBJECT OF_TYPE node OF_TERMS (1 * def(x)) +
SUM FOR_ALL y AN OBJECT OF_TYPE link OF_TERMS (1 * cut(y)) ;
OBJECT_SYSTEM Failure_Counter EXTENDS counter_def ;
```

10.2 CLASS CREATION/MODIFICATION TOOL

The tool can be accessed via the Create type context menu or, when the cursor is on the definition line of a type, via the Edit element context menu.

The tool allows you to define the name of the class and the classes it inherits from as well as the order of inheritance.

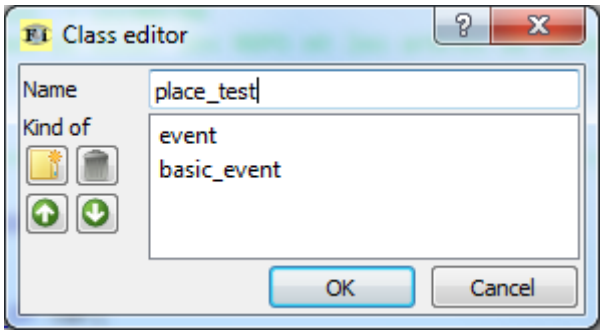


Figure 2: tool for creating/modifying a class

10.3 INTERFACE CREATION/MODIFICATION TOOL

The tool can be accessed via the Create interface context menu or, when the cursor is on the definition line of an interface, via the Edit element context menu.

The tool allows to define the name of the interface, its edition mode, its label (description), its kind (class of objects accepted in the interface) and its cardinality (limits of the number of objects accepted in the interface).

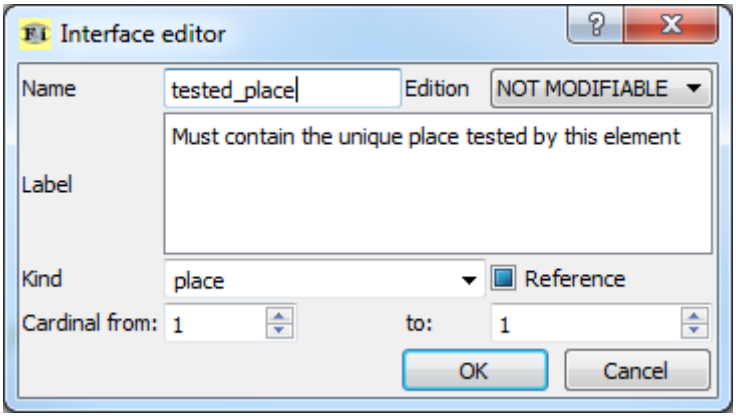


Figure 3: Interface creation/modification tool

10.4 TOOL FOR CREATING/MODIFYING A VARIABLE

The tool is accessible from the Create variable context menu or, when the cursor is on the definition line of a variable, from the Modify element context menu.

The tool allows to define the name of the variable, its edition mode, its label (description), its type (Constant, Attribute, Effect, Failure), its domain (Boolean, Integer, Real, Enumerated), its initial value and its reset mode. In the case of an enumerated variable, the tool allows to define the different accessible values.

Attention! The initial value is a FIGARO expression; the enumerated values must be written there surrounded by a single quote.

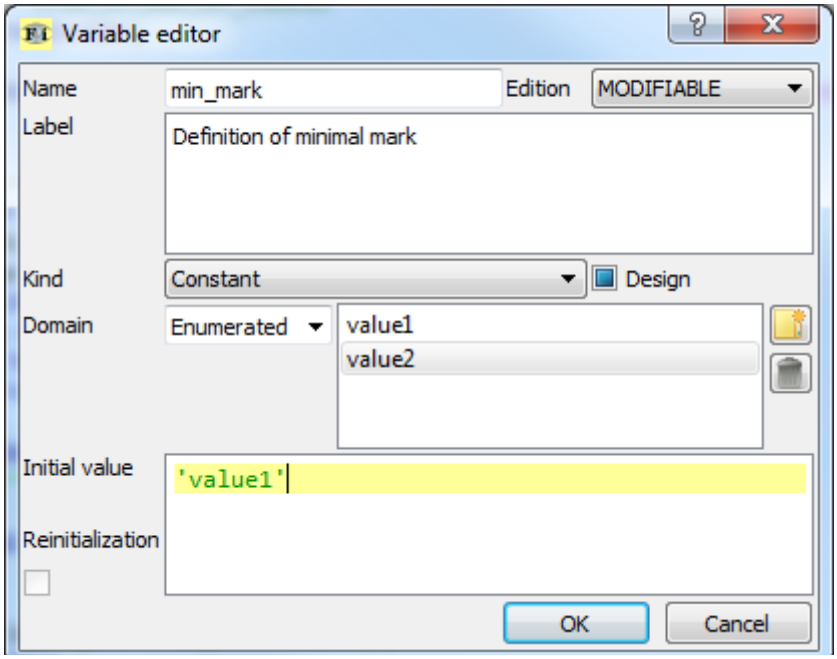


Figure 4: Tool for creating/modifying a variable

10.5 TOOL FOR CREATING/MODIFYING AN INTERACTION RULE

The tool can be accessed via the Create interaction context menu or, when the cursor is on the definition line of an interaction rule, via the Edit element context menu.

The tool allows you to define the name of the rule, its label (description), its different facets (Given, If, Then, Else) and the groups and steps to which it belongs.

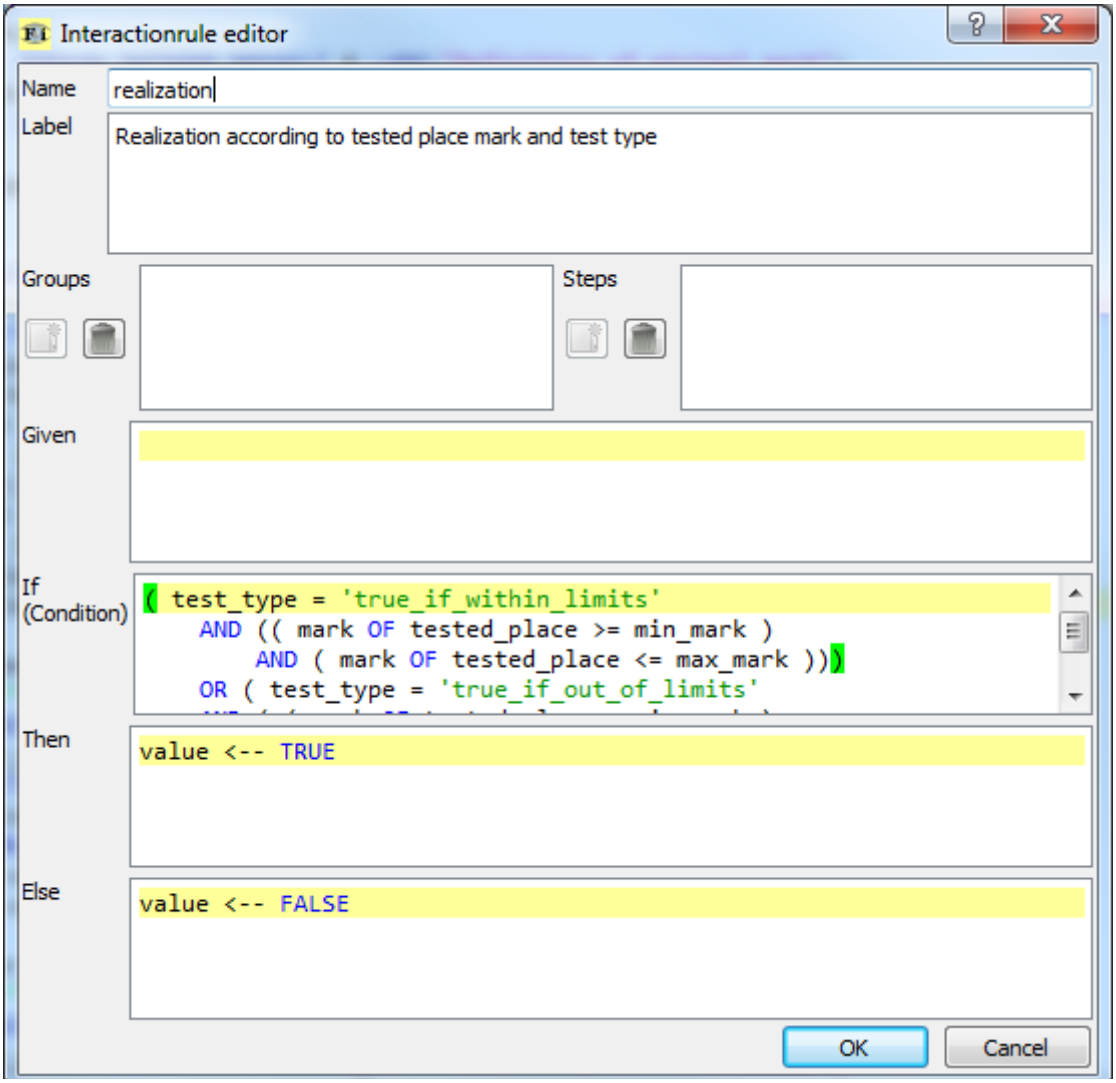


Figure 5: tool for creating/modifying an interaction rule

10.6 TOOL FOR CREATING/MODIFYING A RULE OF OCCURRENCE

The tool can be accessed via the Create occurrence context menu or, when the cursor is on the definition line of an occurrence rule, via the Edit element context menu.

The tool allows you to define the name of the rule, its label (description), its different facets: its condition, its different transitions and the groups to which it belongs.

For each transition (only one for exponential and constant time rules) it is possible to define a name, a label, the value of the law parameter and an action.

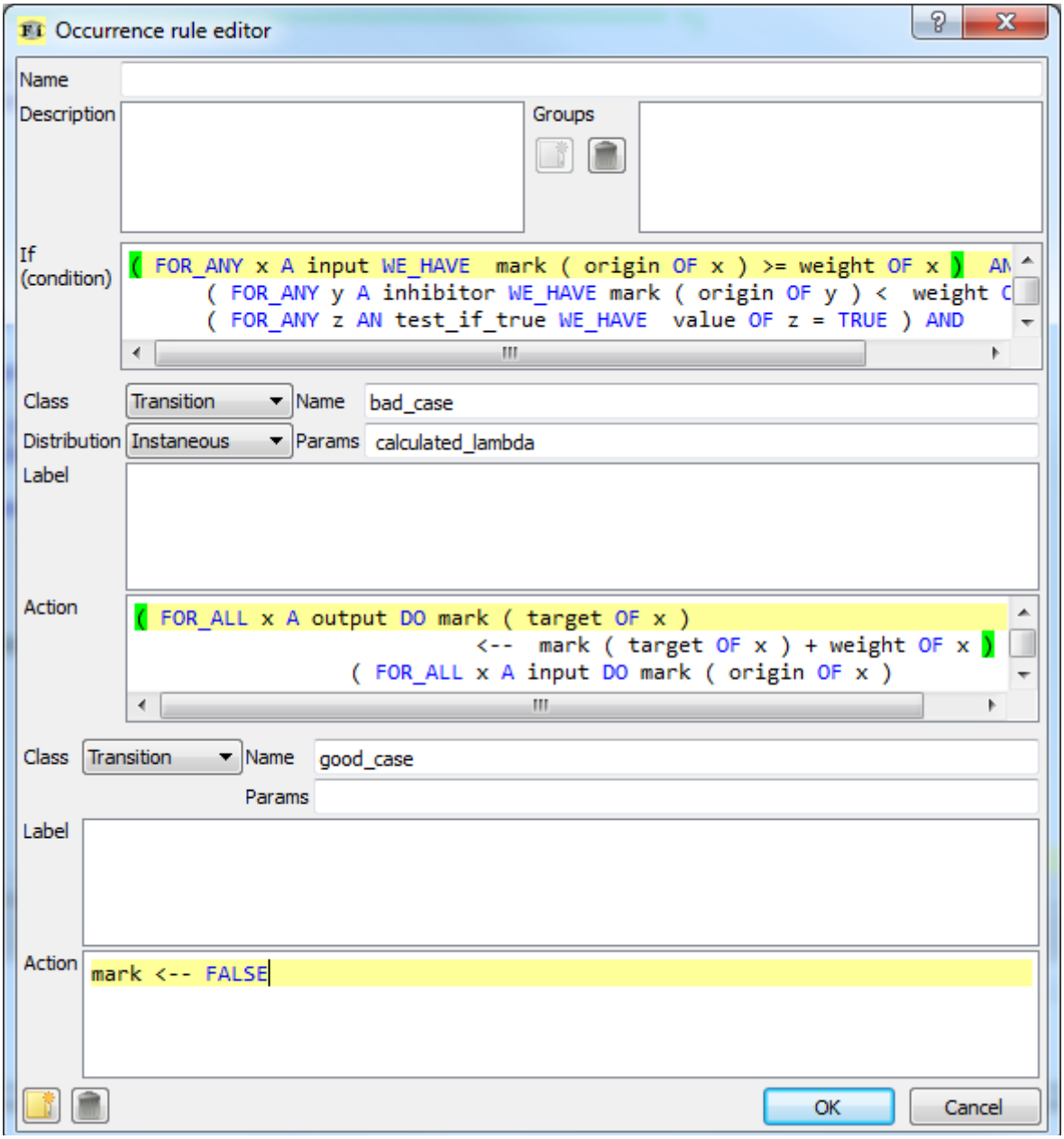


Figure 6: tool for creating/modifying a rule of occurrence

11 CREATING THE KB3 GUI SETTINGS FILE

11.1 CREATE GRAPHICAL REPRESENTATIONS OF OBJECTS IN KB3

The creation of a KB3 GUI settings file (.bdc file) requires to define the graphical parameters associated to each CLASS.

To do this, in the Classes widget or the Palette widget, select the class of interest and click on the Edit button (or on the Edit command of the contextual menu). This opens the window of the selected class (cf. Figure 7) in which the user will enter the graphic properties of the objects corresponding to the selected class.

First, the user chooses the shape to give to the objects of the selected class. An object corresponding to a given CLASS can have a Link shape[3] or a Node shape. If it does not have a shape, it does not correspond to a graphic object and will thus not be usable for the construction of the system diagram. The user also specifies if he wants to see objects of this type appear in the KB3 object palette.

Warning: any type inheriting from a node is a node and any type inheriting from a link is a link. Consequently, the user will not be able to modify the shape of a child.

If the user has already placed the graphic icon that will represent the objects of the type of interest in the icons directory, it will appear in the window. If not, or if the user wishes to modify the icon, he/she will be able to access directly to the graphic editor installed on his/her workstation by clicking on the "Edit default graphic representation" button (cf. Figure 7).

A Link or Node object does not need to have a graphical instance if it is never placed on the system diagram. In this case the user will choose the graphic instance PROHIBITED and in the opposite case, he will choose MANDATORY.

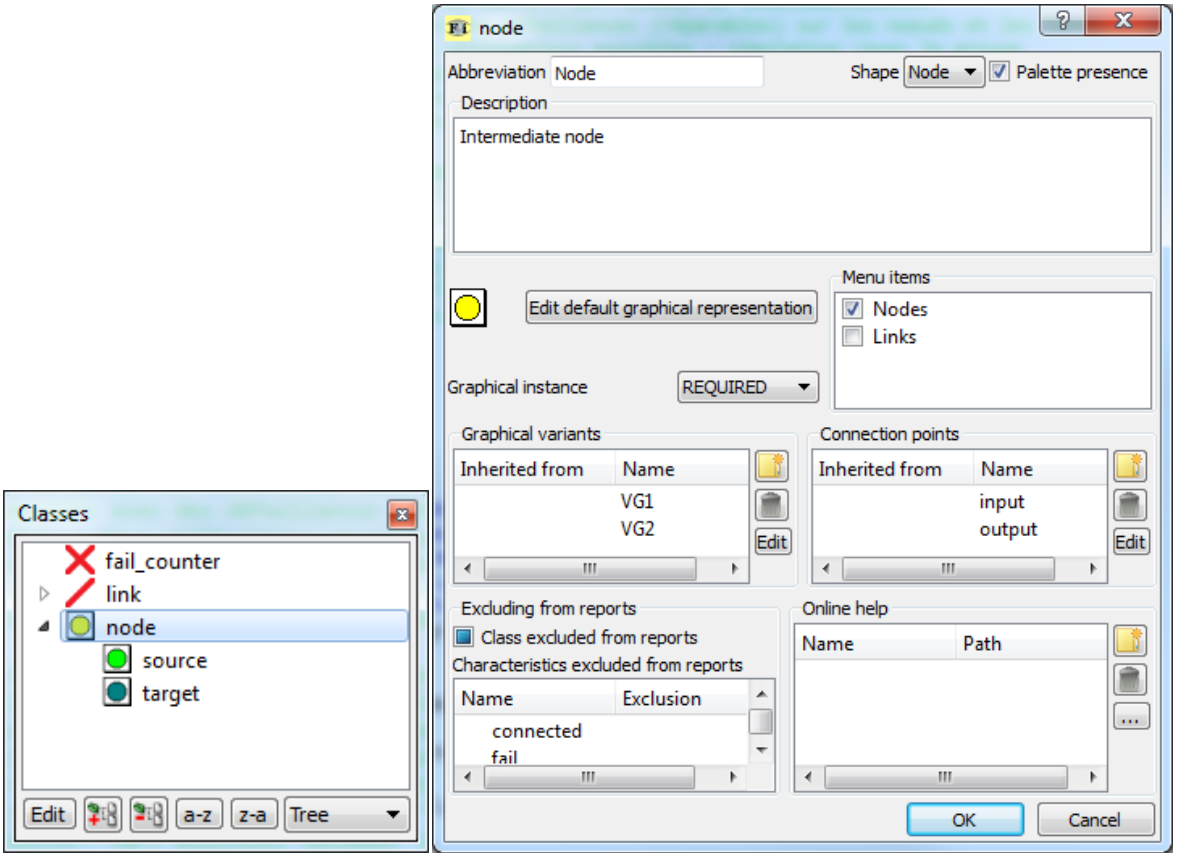


Figure 7: Example of the creation of the graphical representation of node objects

An object may need to have several graphical representations. They are called graphical variants. This is often the case, for example, when you want to visualize the state of a component during a simulation in KB3 (cf. 11.7). These variants are defined by the user and will correspond to different .sym files or to different aspects of the initial graphical instance. For the latter case, the user creates a new graphical variant by clicking on the new icon and then on the "Edit" button which proposes to the user in a dedicated window different ways to modify the icon of an object under condition and for a visualization whose name has been

[3] A link connects two nodes. In practice, a link is mainly intended to facilitate the entry of interfaces in KB3.

defined in the Visualizations space of the General widget (cf. § 11.7 and example on the Figure 8).

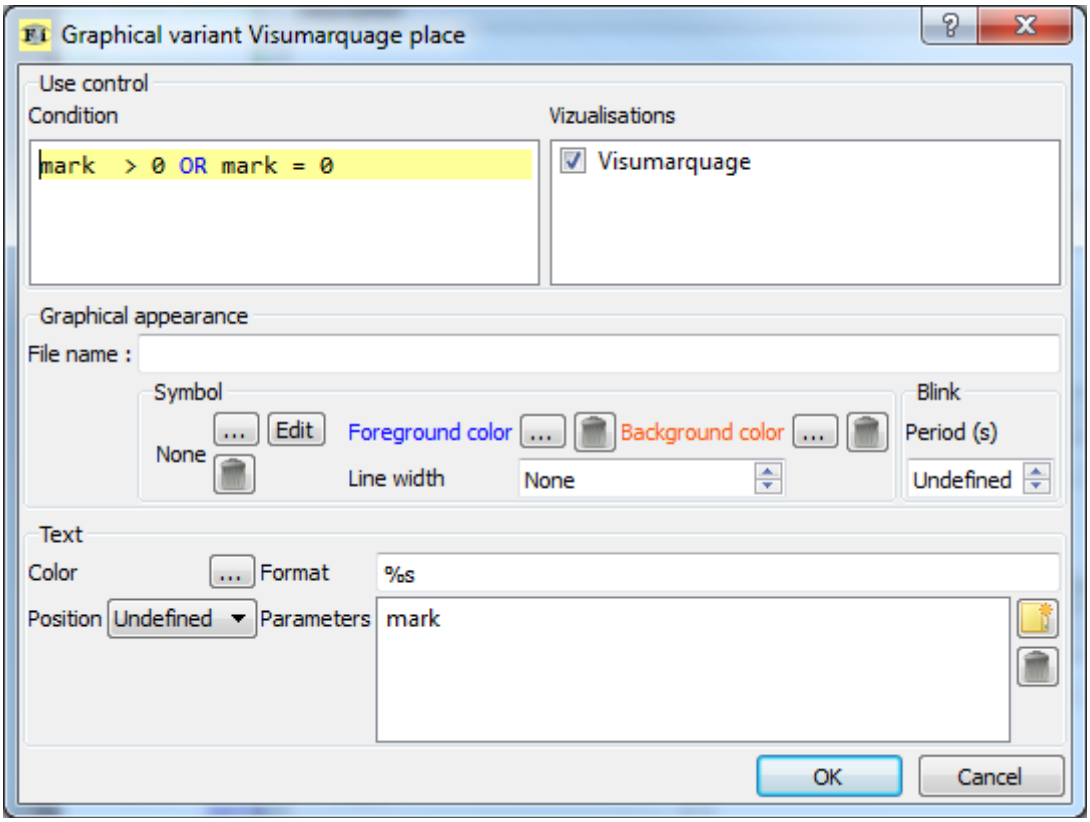
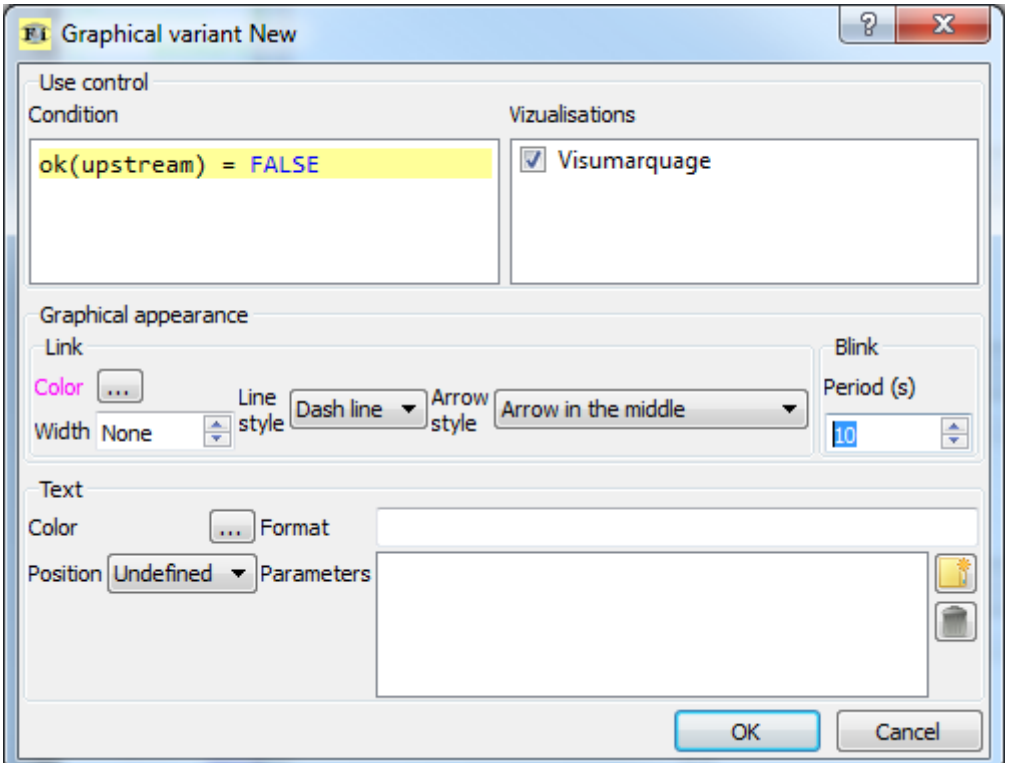


Figure 8: Example of how to create a graphical variant of node objects

The possible changes for a node object are as follows:

- Replace the symbol by clicking on the "..." button; you must then enter the name of the file storing the associated image
- Modification of the background color (fill color of the closed elements) and foreground color (color of the lines of the elements having a black line)
- Changing the line thickness
- Changing the flashing period of the icon
- Adding a text consisting of a format and parameters. The format is made of text and insertion markers " %s " which will be replaced by the parameter values. The text can be positioned around the icon and colored.



The possible modifications for a link object are the following:

- Change the color and thickness of the line
- Modification of the line style (plain line, dotted line, dashed line, ...)
- Modification of the symbols on the link (arrow or diamond at the beginning, middle or end of the link)
- Changing the flashing period of the icon
- Adding a text consisting of a format and parameters. The format is made of text and insertion markers " %s " which will be replaced by the parameter values. The text can be positioned around the icon and colored.

11.2 CREATE CONNECTION POINTS ON OBJECTS OF A TYPE

If the object is a Link, the connection points are always STARTING and TARGET (the names cannot be modified and no connection point can be added or deleted). FigaroIDE proposes the list of connection points already created in the database and at least the connection points defined for its parents, which cannot be modified (cf.Figure 7).

To define the connection points of nodes to which a link can be connected, the user must edit the connection point properties by clicking the Edit button.

The user can then add connection permissions where he has to choose the type of node accepted to the connection and eventually define the accepted connection point. If the name of the connection point on the node is not defined, the link will be able to be connected to all the connection points of the node (case of the Figure 9 If the name of the connection point on the node is not defined, the link will be able to be connected to all the connection points of the node (case of the "Connection point name accepted").

In the example of the Figure 9 the connection point STARTING can be connected to objects of type Component.

A link connection to a node can automatically fill the interfaces of the connected nodes and the link. To do this, the user clicks on the "Add interface filling" button, which adds an interface filling rule that can be modified by double-clicking on the rule (cf. Figure 9).

The principles of filling the interfaces are based on the two keywords contained in the rule. The filled interface is the one of the object corresponding to the first keyword. Thus, the fills for the different rules are as follows:

- RULE_STARTING_LINK: fills in the interface named in the "Value" column of the node at the link START, with the name of the link,
- RULE_LINK_STARTING: fills the interface named in the "Value" column of the link with the name of the node at the link's START,
- RULE_TARGET _LINK: fills in the interface named in the "Value" column of the link ARRIVAL node with the name of the link,
- RULE _LINK_TARGET: fills the interface named in the "Value" column of the link with the name of the node at the link's ARRIVAL,
- RULE_TARGET_STARTING: filling of the interface named in the "Value" column of the node at the lin's arrival, by the name of the node at the link's departure,
- RULE_STARTING_TARGET: filling of the interface named in the "Value" column of the node at the link's departure, by the name of the node at the link's arrival.

For the example of the Figure 9, the RULE_STARTING_TARGET implies that the name of the node connected to the TARGET of the link will be added in the interface named "place" of the node connected to the STARTING.

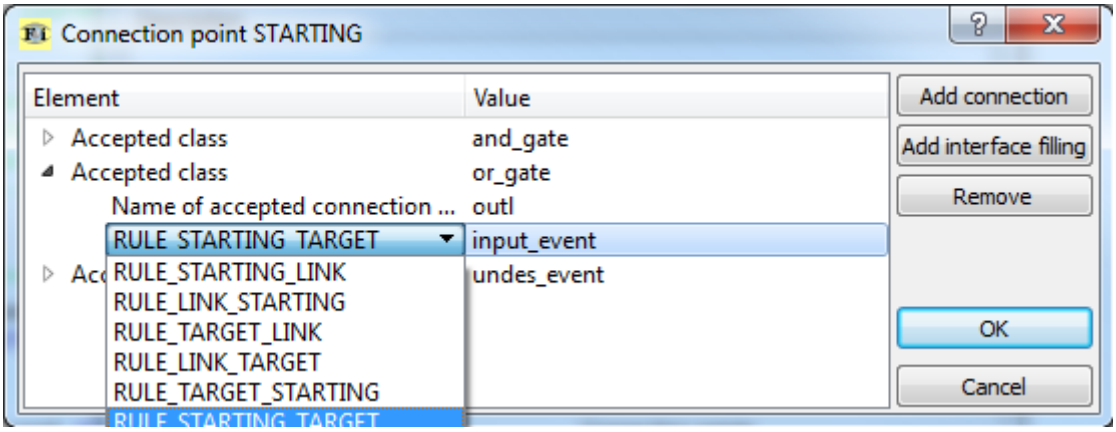


Figure 9: Example of the creation of connection points on link objects and the filling of interfaces

If the object is of Node form, several connection points can be added to the node by the user who defines their positions on the graphical object (cf. Figure 10).

The addition of a connection point is obtained by the "Add connection" button. FigaroIDE then proposes the list of all the link types of the KB. The user chooses the type of link and the name of the connection point.

If the type inherits from a type of form Node which itself has connection points, it is possible to inherit it by choosing the parent type in the Inherit from column and the inherited connection point in the Name column. In this case, the inherited connection point cannot be modified.

As for the links, the user must define the connections allowed on each connection point. To do this, the user selects the connection point to be modified and clicks on the Edit button.

The user can then add connection permissions where they must choose the type of link accepted on the connection and optionally set the accepted STARTING or TARGET connection point. If the connection point name on the link is not set, the node will be able to connect to all ends of the link.

If the user wants to fill an interface of the connected node or link, he clicks on the "Add interface fill" button and chooses the interface to fill according to the rule :

- RULE_NODE_LINK: fills the interface of the node named in the "Value" column with the name of the link,
- RULE_LINK_NODE: fills the interface of the link named in the "Value" column with the name of the node.

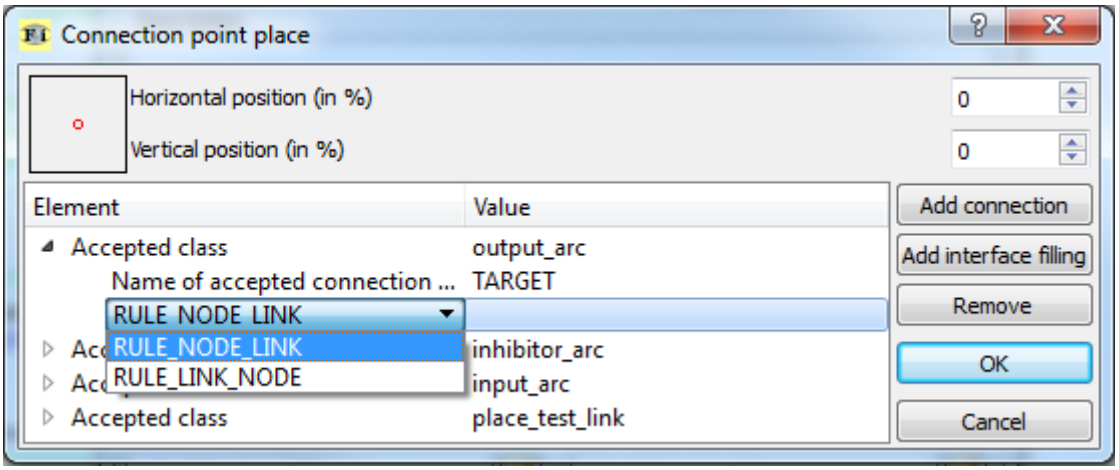


Figure 10: Example of the creation of connection points on node objects and the filling of interfaces

11.3 DEFINE THE ONLINE HELP OF A KB3 TYPE

The online help consists of files that can be opened on the target machine when using the KB with KB3.

The path to these files must therefore generally be a network path so that they can be accessed from all target machines.

Each file is associated with a name allowing the KB3 user to choose the documentation to display.

To add a document, click on the new icon.

The name and path of the document can be changed directly in the list shown. The path to the document can also be defined by clicking on the "..." button, which opens a window for choosing a file.

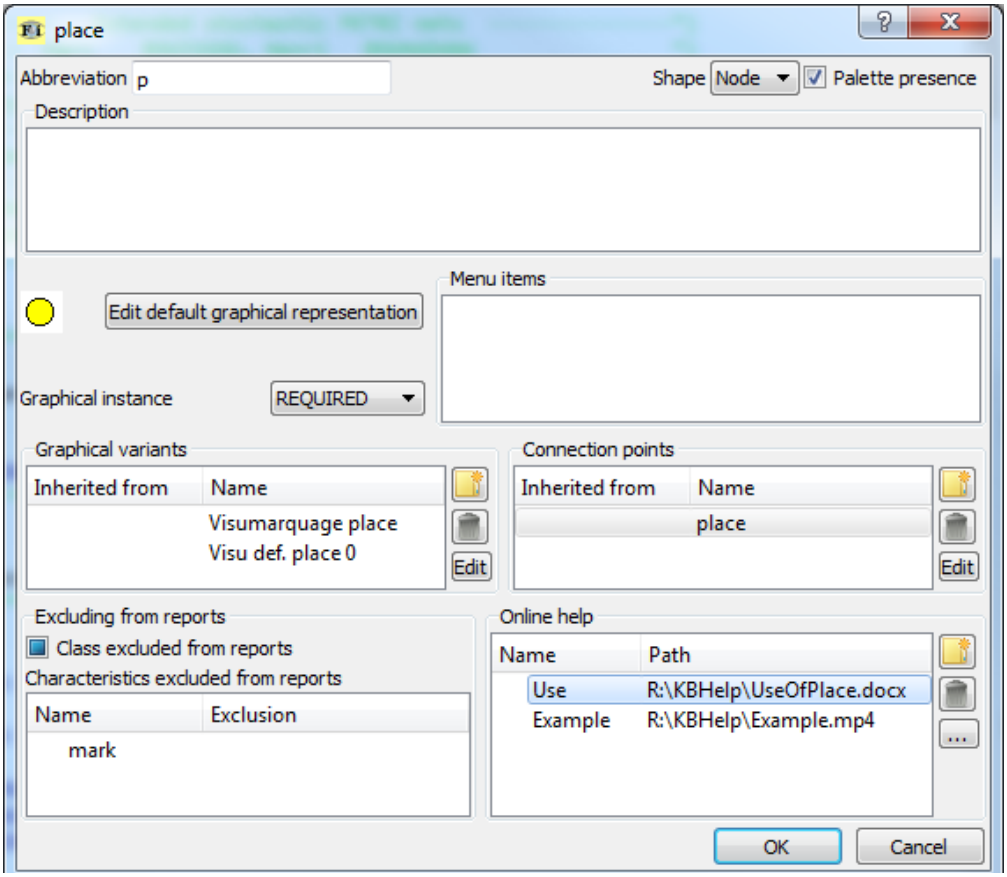


Figure 11: Example of an online help entry

11.4 MANAGE THE INCLUSION OF A KB3 TYPE IN STUDY NOTES

KB3 allows building automatic reports where the system entered by the user is described in a synthetic way.

It is possible to exclude certain types or variables from the reports when they are not relevant.

The "Class excluded from reports" checkbox (see below) allows you to set the exclusion or inclusion of the class of study notes. By default, the check box is grayed out, which corresponds to an exclusion defined by inheritance or by default (no exclusion by default). Checking the box excludes objects of this class from the notes. Unchecking the box includes objects of this type in the notes.

The "Characteristics excluded from reports" table presents the list of characteristics of the class. The right-hand column allows you to define how the characteristics are taken into account in the study notes using 5 choices:

- " " : (empty choice) exclusion is set by inheritance or by default (no default exclusion)
- "Complete": the characteristic is excluded from all paragraphs of the notes
- "From § House events": the characteristic is excluded from the paragraph House events in the notes
- "From § Profiles: The characteristic is excluded from the Profiles section of the notes
- "None": The characteristic is not excluded from the study notes

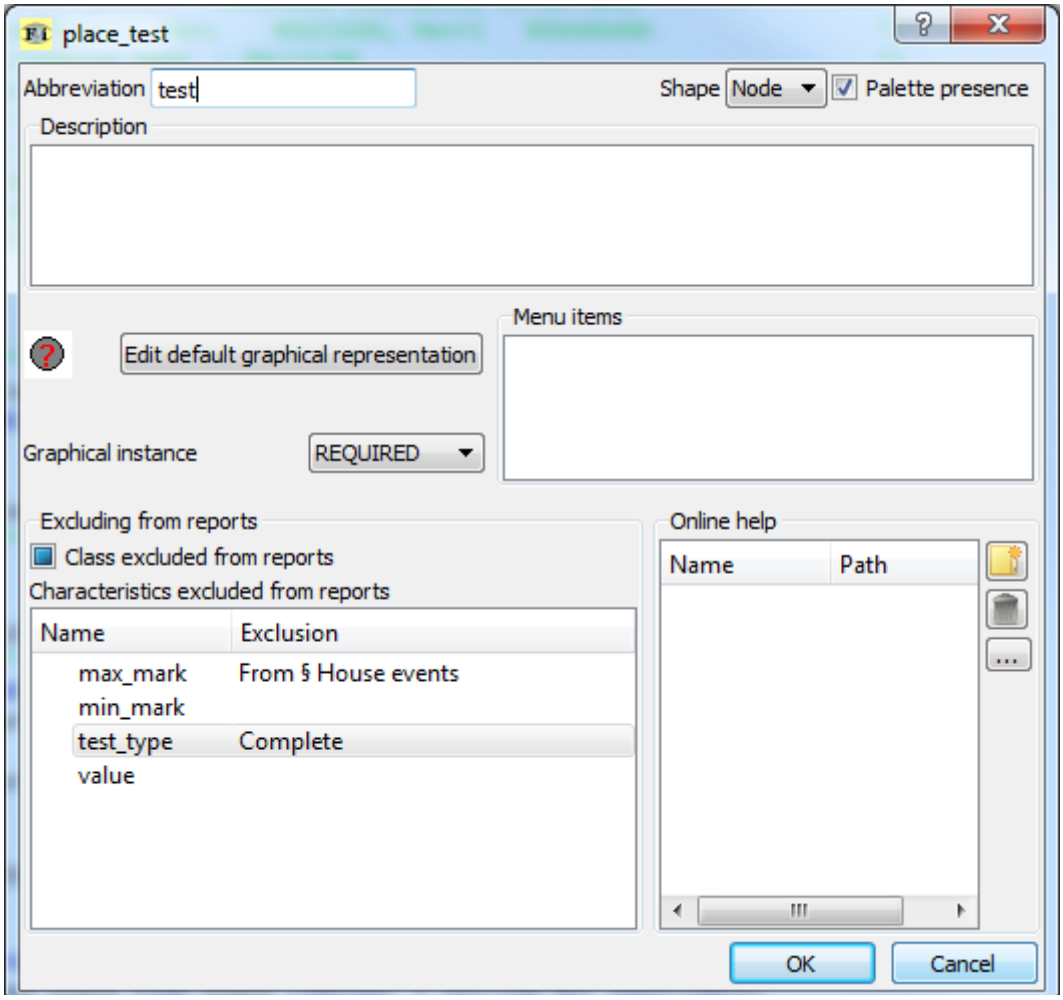


Figure 12: Example of excluding items from study notes

11.5 DEFINE THE PALETTE OF CLASSES

The user can choose to present the classes in the KB3 palette in an order that is not necessarily alphabetical (for example, nodes first, then links). To do this, the user selects a type from the list and moves it up or down using the arrows .

11.6 CREATE GROUPINGS OF OBJECT TYPES

For some large knowledge bases, it can be interesting for the user to present in the KB3 palette groups of object classes (for example, grouping nodes and links in two separate directories). It is in the "Class families" area of the Palette widget that the user will define these groupings.

The membership of a class to one of these groups can be defined in the class editing tool (see 11.1) in the Menu items frame. The type belongs to each of the groups whose checkbox is checked.

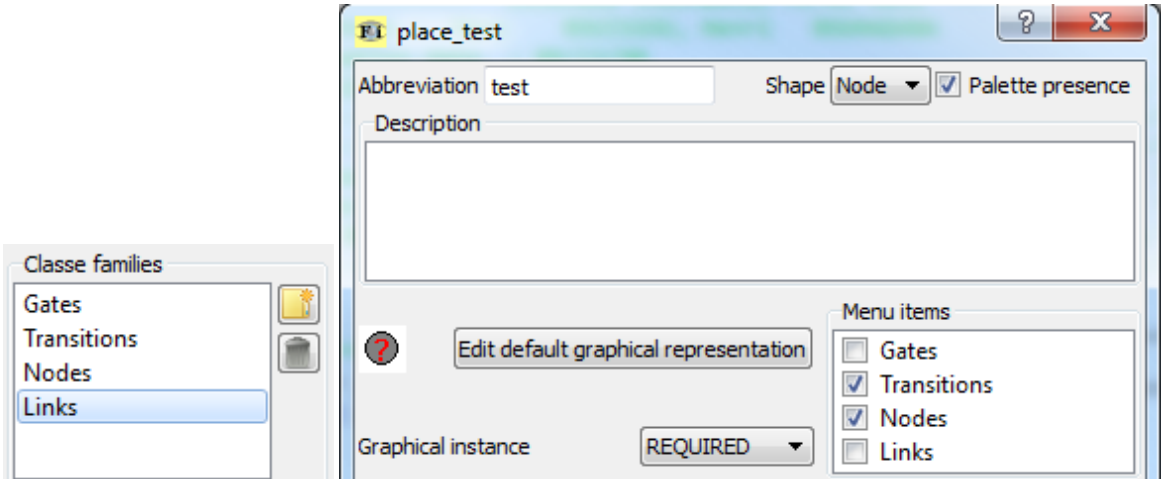


Figure 13: Example of creating and using type grouping

11.7 CREATE VISUALIZATIONS

The designer of a BDC can specify several visualizations that will allow to display in KB3 some results of the simulation on the system scheme (for example, the visualization of the components in failure, the visualization corresponding to the initial configuration of the system by a representation of the states of the components). These different visualizations suppose that the user has defined in the database graphical variants for the classes which will make the objects appear in a different way according to the realization of certain conditions (cf. 11.1). The user creates a new visualization in the Visualizations widget.

11.8 DEFINE LAYERS

Layers allow you to filter the display of objects in KB3 by selecting the classes of objects to display. An object is displayed if its class is in the list of one of the layers selected in KB3 or if no layer has been selected.

Each layer allows you to define a background image that will be displayed as the main page background in KB3 if the layer is selected by the user.

Layers are added by clicking on the new button of the Layers tool. The name of the layer can be changed in the list next to it. To modify the content of the layer, click on the Edit button which opens the window below.

To add or remove a class whose objects are to be displayed, click on the Add and Remove buttons on the right. The class is then selected using a drop-down list.

It is possible to define the background image of the layer by clicking on the " ... " button. It is then necessary to define the name of the file used to store this image (name without extension).

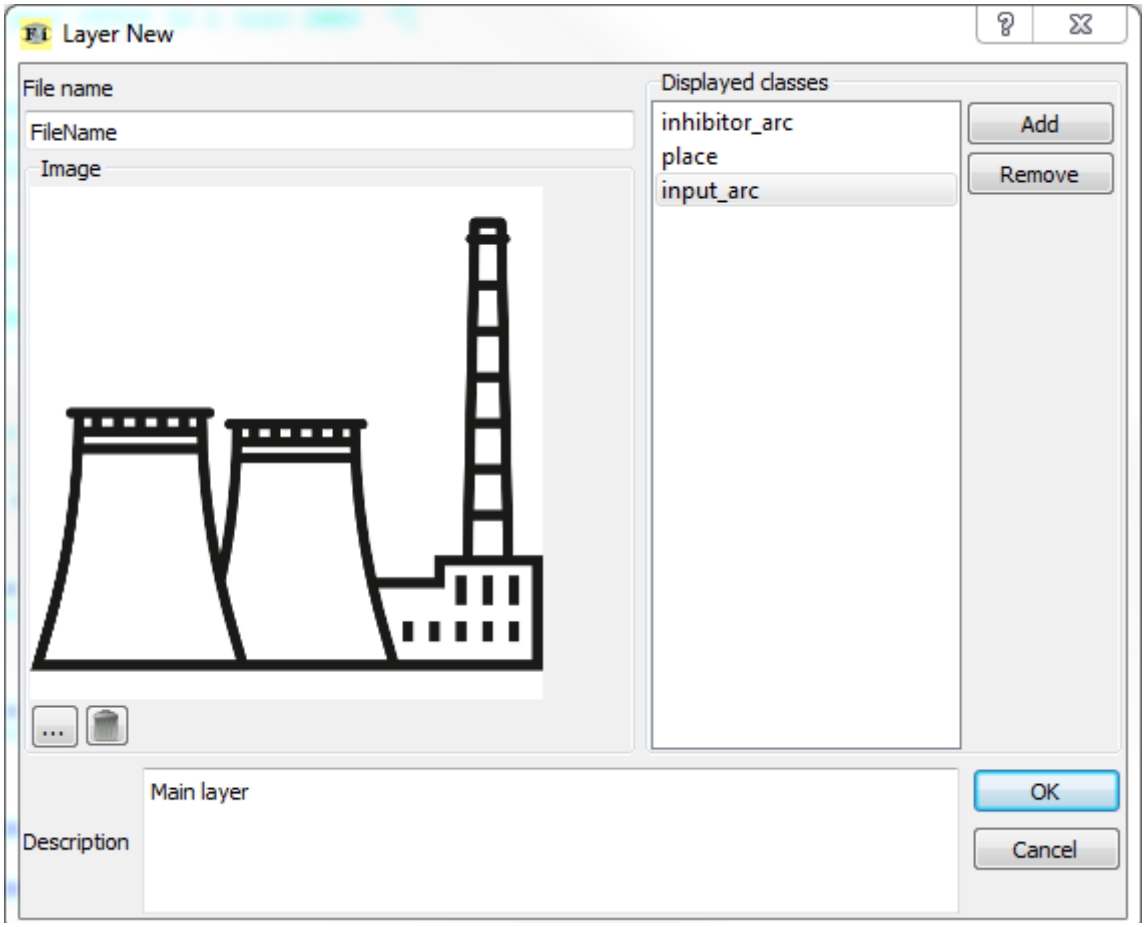


Figure 14: example of a layer definition

11.9 CHANGE THE GRAPHIC REPRESENTATION OF GMCS

In addition to the icons corresponding to the types defined in the bdc, it is systematically necessary to have an icon with the name GMC (thus the two files GMC.ico and GMC.sym) which will be used to represent the graphic macro-components (GMC). By default, the GMC icon is the following:  .

To change the icon, the user clicks on the Edit button of the "General" Widget in front of the GMC icon. This opens the window shown in the Figure 15 in which FigaroIDE gives access to two drawing tools by :

- the Edit button on the left which launches the graphical tool defined by default on the user's workstation for .ico files,
- the Edit button on the right that launches the Symbol Designer graphic tool provided with the FigaroIDE software.

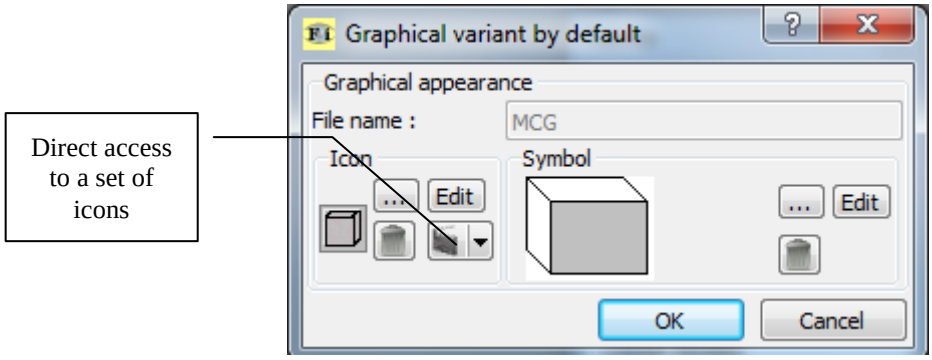


Figure 15: MCG icon modification window

11.10 ADD MANUAL RULES

It may happen that the KB3 user wants to add rules to specify a specific malfunction or knowledge. For this purpose, he will use manual rules (MR) which will be described graphically by a tree.

The KB designer can force the KB3 user to create his manual rules in a specific Figaro step of the knowledge base. To do so, he defines in the General widget the step in which the MRs will be defined.

Note: If no step is specified for MRs, manual rules built in KB3 will be added in the default step created: default_stage.

11.11 ADDING UNDESIRABLE EVENTS

Undesirable events (UEs) are mainly used for fault tree generation in KB3. They could also be used to build indicators for system model exploitation, but this functionality is already provided by Yams.

As with the manual rules, the KB designer can force the KB3 user to create his undesirable events in specific Figaro step of the knowledge base. To do so, he defines in the General widget the step in which UEs will be defined.

Note: In the absence of a step specification for UEs, undesirable events built in KB3 will be added in the default step created: default_stage.

11.12 CREATE DEDUCED OBJECT ALGORITHMS

A deduced object is an object automatically built by KB3 when launching a tree generation, a simulation or an instantiation.

The deduced object algorithms are intended to define the construction of these objects. These algorithms can also be used to complete interface fills, compute initial values of constants, or to perform checks on the user's input to the system in KB3.

An algorithm can be defined in two forms:

- A configurable algorithm for building obstruction blocks
- A rule-based algorithm using an extension of the Figaro language

The first form is proposed by FigaroIDE to ensure compatibility with older knowledge bases, but it is best to avoid using it.

The rules and equations of the DO algorithms are defined in the KB types chosen by the user. The rules and equations have the same facets as the Figaro1standard interaction rules and

equations but can use specific operators to modify objects or interface contents and display error messages.

By default, the algorithms are executed in the order of declaration in the DO Algorithms widget and the rules corresponding to an algorithm in the order of declaration chosen by the user within each class in the algorithm definition window (cf. example of the Figure 16 the ControlNumSat rule will be executed before the ControlDatePlan rule).

It is possible to modify the execution order of the algorithms by specifying within an algorithm the algorithm to be executed after (Loop/jump field). This makes it possible to build loops of algorithms which will be carried out until stabilization of the calculations.

The execution of an algorithm will consist in scanning the objects of the system to find the objects with rules to execute. For each of these objects, KB3 executes the *Start* rules and then the *Exit* rules. If in these objects, there are TRANSFER_TO actions, the algorithm executes the rules of the object designated by TRANSFER_TO starting with the *Entry* rules then the *Exit* rules then recursively applies the TRANSFER_TO.

The principle of building an algorithm is as follows:

1. In the DO Algorithms widget, add a name to the list and click the Edit button,
2. In the algorithm window, add a class by clicking on the "Add Class" button, then select the class of interest on the line thus created in the list of classes,
3. Add a rule by clicking the Add Rule button applicable to either *Start*, *Input* or *Output* depending on the user's choice, or Add an equation by clicking the Add Equation button,
4. For an equation, specify the name of the system to which the equation belongs in the System name field that appears on the right side of the window when the equation is selected,
5. Describe in FIGARO language the rule or the equation by entering them in the spaces If PC, Given, If, Then, Otherwise for a rule and in the spaces If PC, If, Formula for an equation.
6. Specify the order of declaration of a rule or equation using the green arrows,
7. Add to the list of Constructed classes all the classes of objects that will be constructed by the algorithm.

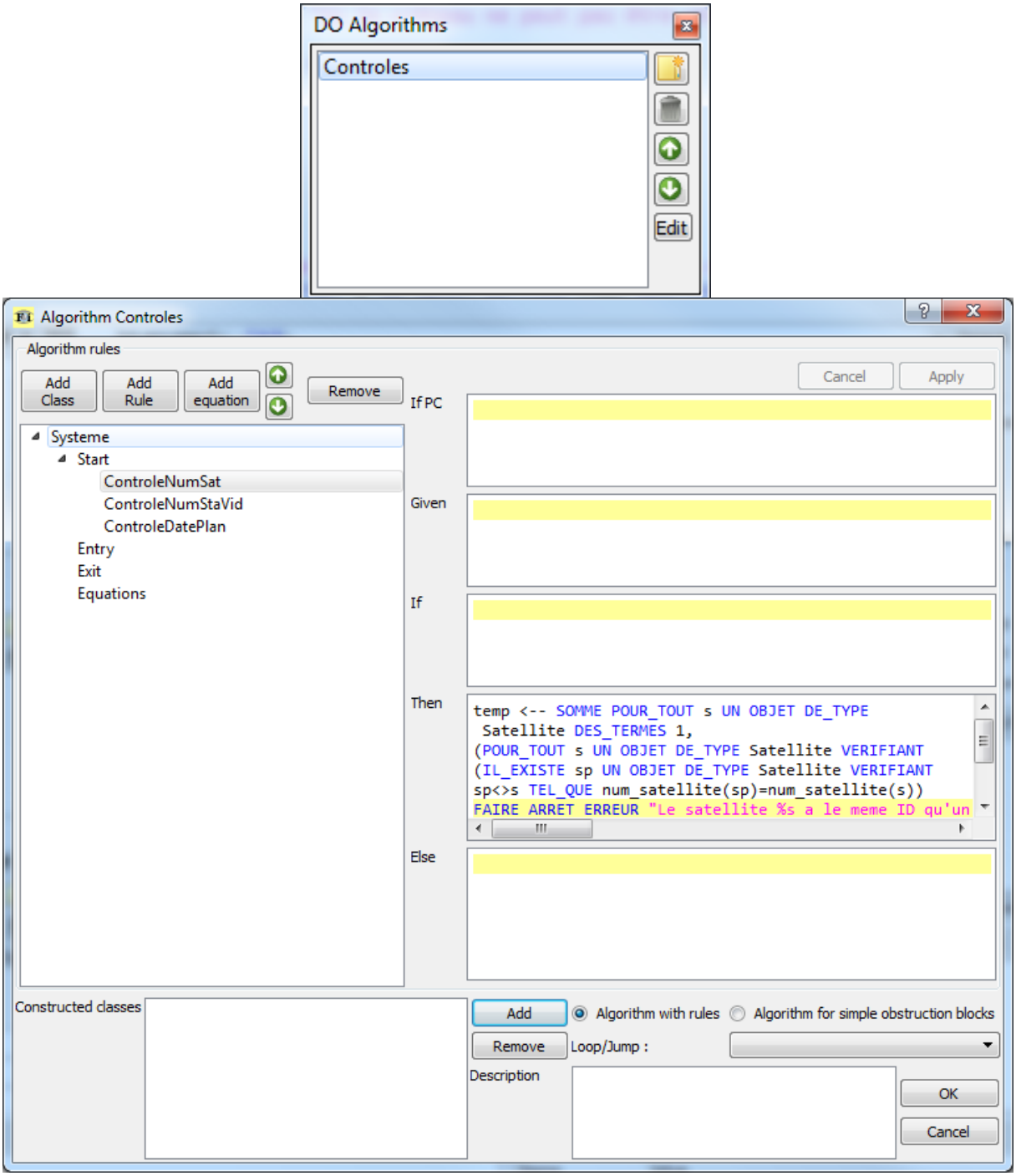


Figure 16: DO Algorithms widget and Controle algorithm construction window (display of the ControleNumSat rule)

11.13 CREATE PROCESSING MODELS

KB3 proposes five categories of processing models to be performed on a study (≡model of a system) in KB3 (cf.Figure 17):

- **Generation of fault trees**,
- **Interactive simulation**,
- **Figaro0 instantiation** corresponding to the creation of a file in Figaro0 language following the instantiation of the KB rules,

- **Fact base** allows to extract the definition of objects, filled interfaces and modified variables (constants, attributes),
- **External codes** allows the launching of calculation codes on data generated by KB3.

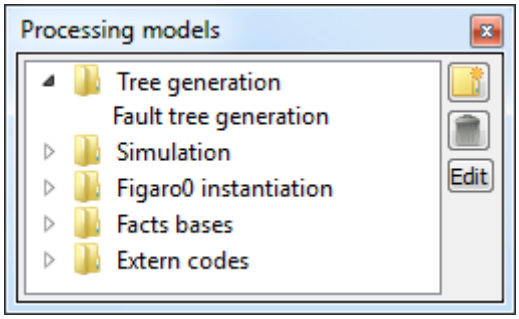


Figure 17: Processing models widget

For a study to have access to these treatments, the treatment models must be specified in the KB linked to the study. It is possible to create several treatments of a given category.

All these templates will appear in the menus of the "Treatments" tab for the knowledge base user under KB3.

FigaroIDE provides default templates for tree generation, simulation and Figaro0 instantiation (cf. Figure 18).

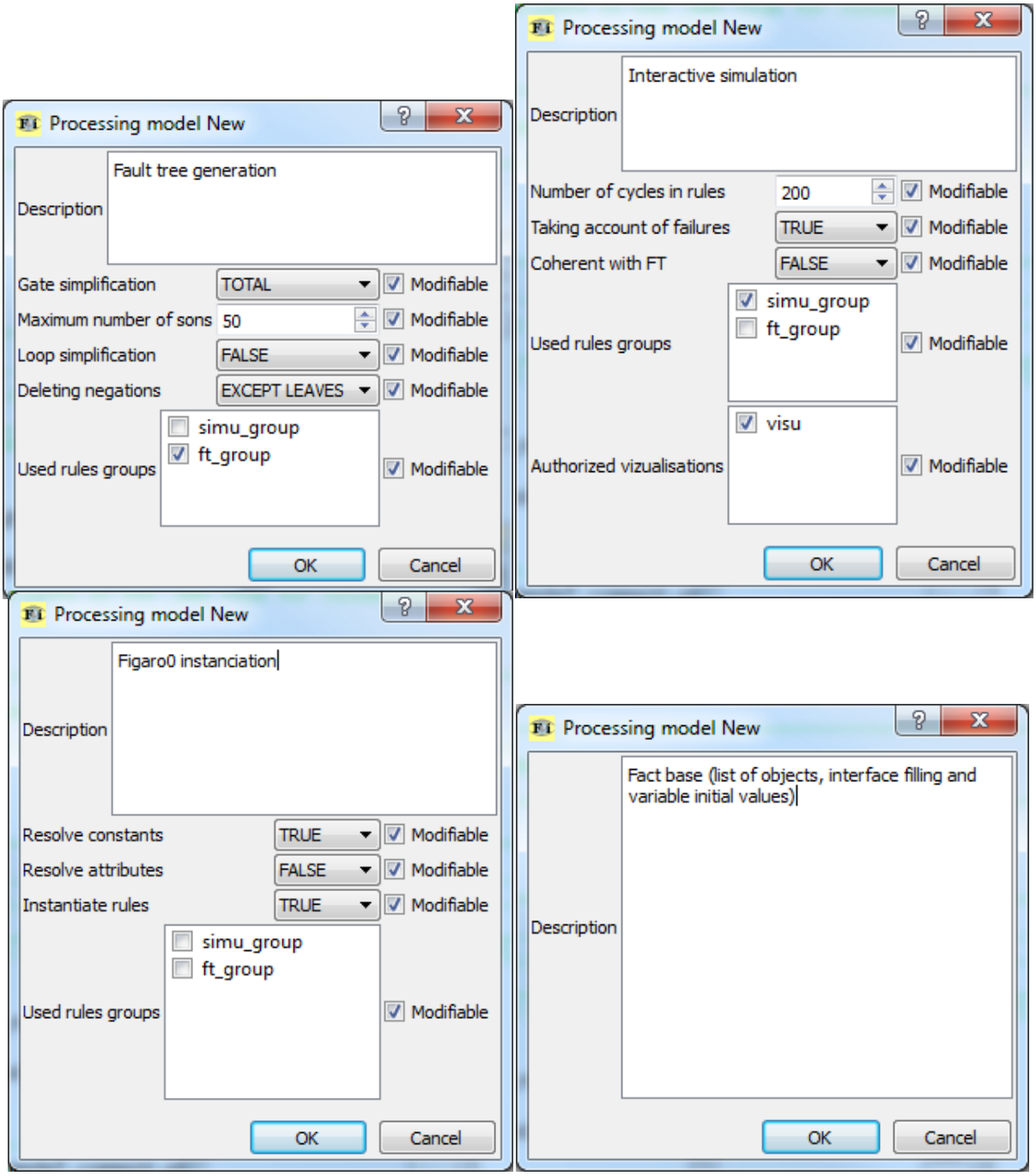


Figure 18: Default processing template settings

For external code processing models, the following must be specified (cf. Figure 19):

- the necessary input information for the external software: Figaro 0 checkboxes, FT (fault tree), FB (fact base, i.e. list of model objects);
- the hierarchy of the subdirectories used to store the necessary Figaro 0 and ADD files: check boxes Knowledge base, Study, Model ;
- usable external software (software installed in conjunction with KB3): check box All or add software in the list (YAMS and FIG0DEBUG are usually available). The names of the external codes must match the names in the list accessible via the KB3 option panel, otherwise the knowledge base user will get a "code does not exist" message when he tries to run this process.

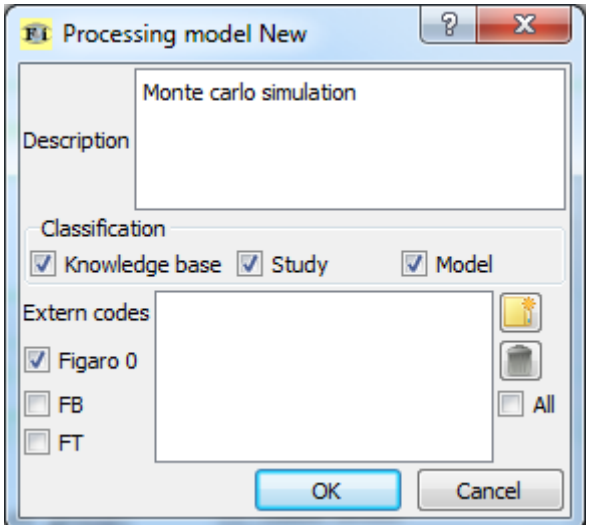


Figure 19: Setting up the processing templates External codes

11.14 CREATE PRECOMPILATION CONSTANTS

The precompilation constants allow creating options for the operation of the knowledge base (for example, to allow seeing or not the attributes of the objects of the study, to control or not the constants, to pass in debugging mode...). In KB3, the user will be able to choose the operating options of the KB that he wants.

Precompilation constants are defined from the Precompilation Constants widget. The principle is as follows: create a new name and click on the Edit button. This opens the constant definition window (cf. Figure 20). The application of the precompilation constant is conditioned by a value ("Debug mode" in the case of Figure 20) which can be integer if Integer value is checked and/or can be mandatory if Mandatory value is checked. In this last case, the use of the KB for which the precompilation constant has been created will only be possible if the KB3 user gives a value (choice of database design).

As an example, let's take the case of the constant __DEBUG__ of the Figure 20. The database designer has defined a use for the constant in the KB (.fi file) by:

```
#ifdef __DEBUG__
#define _VISIBILITY_ EDITION NOT MODIFIABLE
#else
#define _VISIBILITY_ EDITION NON VISIBLE
#endif
```

and assigned `_VISIBILITY_` to all constants and/or attributes that he wanted to make visible only when the constant __DEBUG__ is defined, for example :

```
capacity DOMAIN REAL DEFAULT 100 _VISIBILITY_;
```

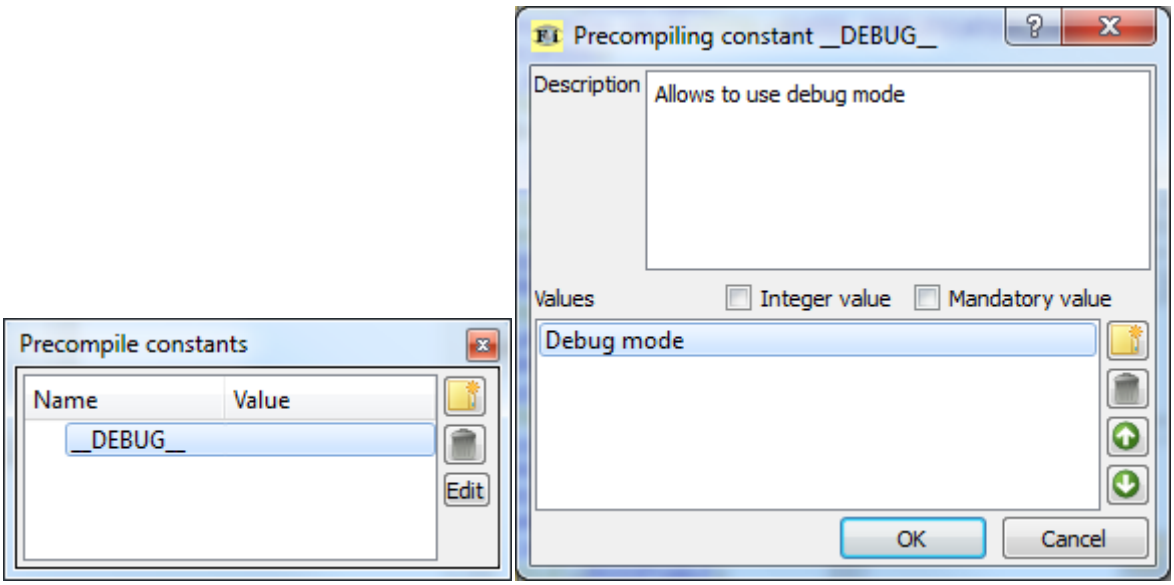


Figure 20: Creating a precompilation constant

The user can visualize in the database (.fi file), the application of the constant. To do this, select the Precompile command from the KB menu after choosing a value for the constant, which displays the precompiled KB in the workspace. For the previous example, we will have in the precompiled KB :

If the associated value in the widget is undefined

```
capacity DOMAIN REAL DEFAULT 100 EDITION NOT VISIBLE;
```

If the associated value in the widget is defined (any value)

```
capacity DOMAIN REAL DEFAULT 100 EDITION NOT MODIFIABLE;
```

11.15 CREATE FILTERS

A filter is the expression of a search within a system built from the knowledge base. Filters are used to build reports on the system.

A filter is defined by the type of element searched for (Object, Constant, Attribute, Failure, Effect, Interface, Reverse Interface); filter masks for the name and description of the element, expressed as regular expressions and a list of classes within which the elements are searched for.

Filters are created in the Filters widget and defined in a dedicated tool.

In the example below, the "Capacities" filter searches for attributes whose name begins with "capa" within objects derived from the sub_system type. This filter will therefore find the attributes capacity and capa_potential.

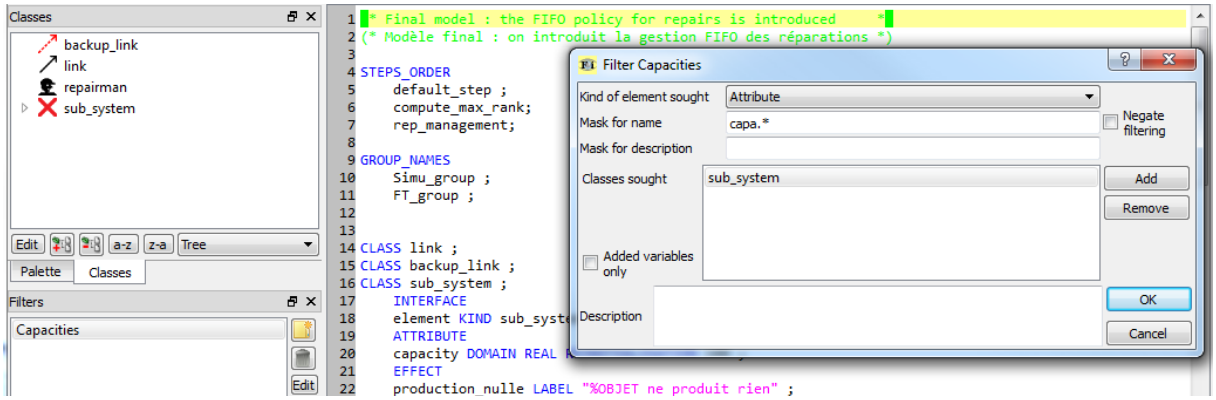


Figure 21: Creating a filter

11.16 CREATE NAMING RULE GROUPS

Naming rule groups are a set of rules allowing modifying the names and descriptions of fault tree elements (gates, flags, base events...) before their export to a quantization code.

Each rule in a group is responsible for modifying one type of tree element.

The modification of the element name consists of:

- to concatenate the following strings: prefix, tree name, separator, element name, suffix
- to apply the sed substitution rule on the string obtained previously.

Modifying the description consists of applying the sed substitution rule to the element's description.

In the case of a house event, basic event or external reference rule, the naming rule may also contain a type modification sed rule. This rule is applied to the concatenated name obtained previously to obtain a string designating the new type of the element:

- #BE: designates a transformation into a basic event
- #HE : designates a house event transformation
- #ET: designates a transformation to external referral

Naming rule groups are created in the Naming Rule Groups widget and defined in the dedicated tool.

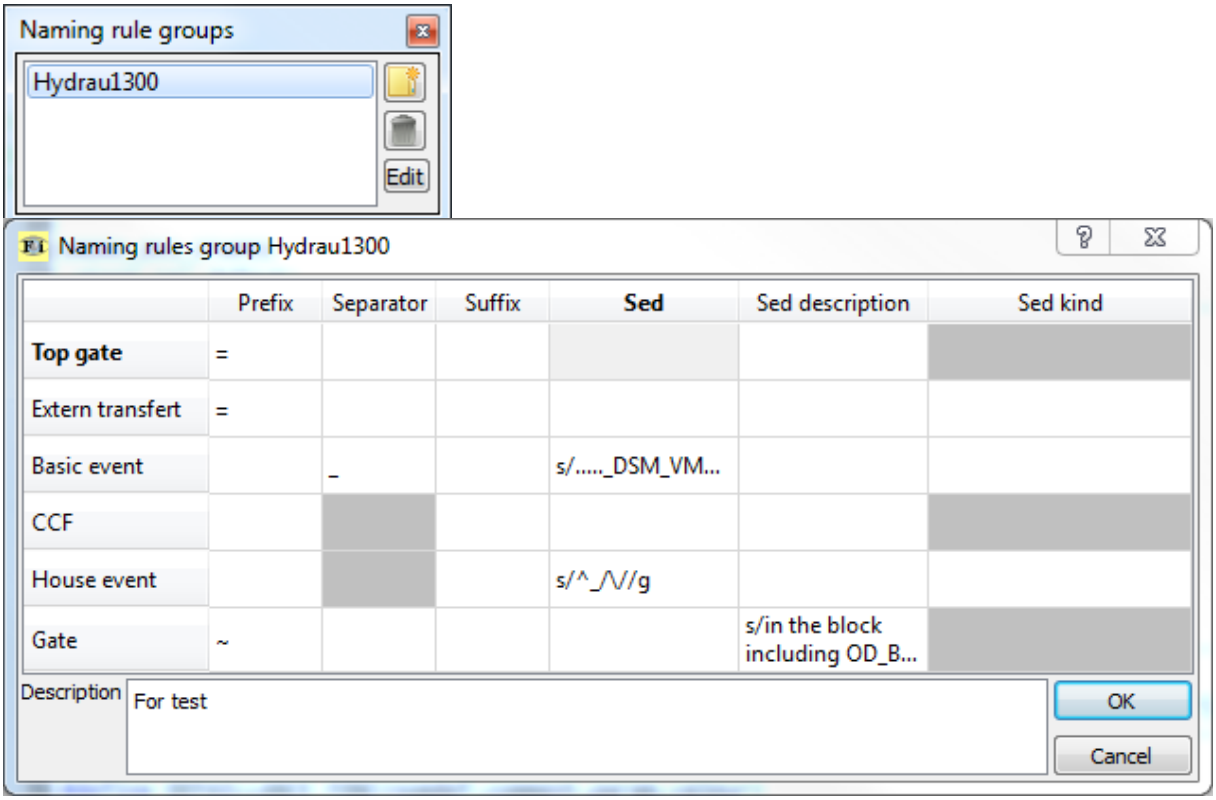


Figure 22: Creating a group of naming rules

12 MANUAL MODIFICATION OF THE KB3 GUI SETTINGS FILE

The paragraph 11 describes how to build or modify this file using the tools dedicated to each of the parameterizable properties, but it is possible, when the user masters the syntax of the file or wishes to make a small correction, to directly modify the text of the XML file.

To activate the manual editing mode of the XML file, you must check the menu item Edit/Edit XML. In this mode, the other tools and the Figaro file are disabled.

Once the modification is done, to validate this modification, you must click on the synchronization button. If there are any errors, the synchronization will not be performed and the errors will be reported in the output panel.

To exit the XML file editing mode, uncheck the Edit/Edit XML menu item.

13 CHECK THE WRITING OF THE KNOWLEDGE BASE

The writing of the KB can be checked at any time by choosing the Verify command in the KB menu or by clicking on the corresponding icon. FigaroIDE then displays in the Output widget the errors detected in the .fi file (cf. Figure 23) and inconsistencies in the .bdc file.

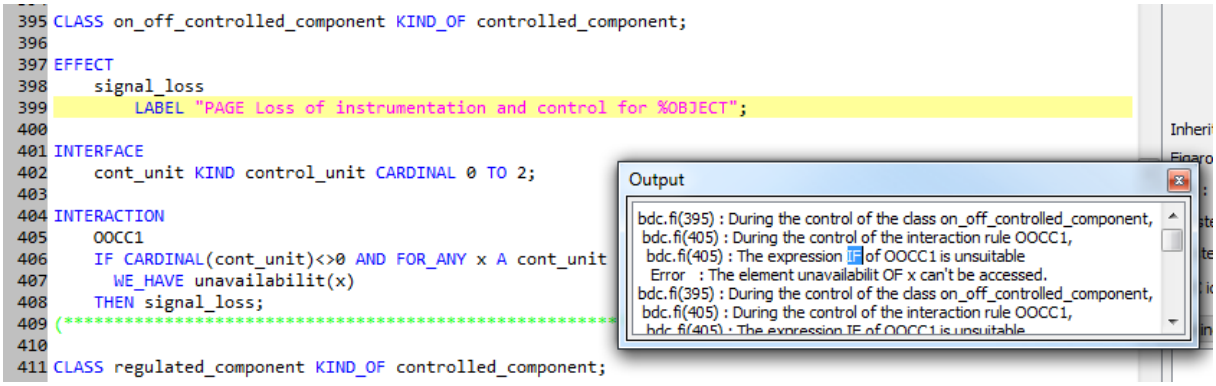


Figure 23: Example of checking the vocabulary and syntax of a KB

A double-click on a line beginning with bdc displays the text reported in either the .fi or .bdc file.

14 TRANSLATE A KNOWLEDGE BASE

The translation tool of the knowledge base being edited can be opened using the menu KB/Translate ...

The translation tool presents 2 lists of texts to translate:

- A list of texts and names present in the Figaro file,
- A list of texts and names present in the XML file.

By default, no translation of these texts is loaded. To load a file containing translations, use the "..." button at the top.

The translations are displayed in the right column of the lists and can be modified directly in the lists.

Erroneous translations (list not enclosed in apostrophes or description not enclosed in quotation marks) are shown with a red background. Translations from the translation file, but which do not correspond to a text present in the knowledge base, are presented with a grey background in the left-hand column and can be deleted using the Delete button.

Once translations have been defined, they can be saved to a translation file using the Save As and Save buttons at the top right of the dialog.

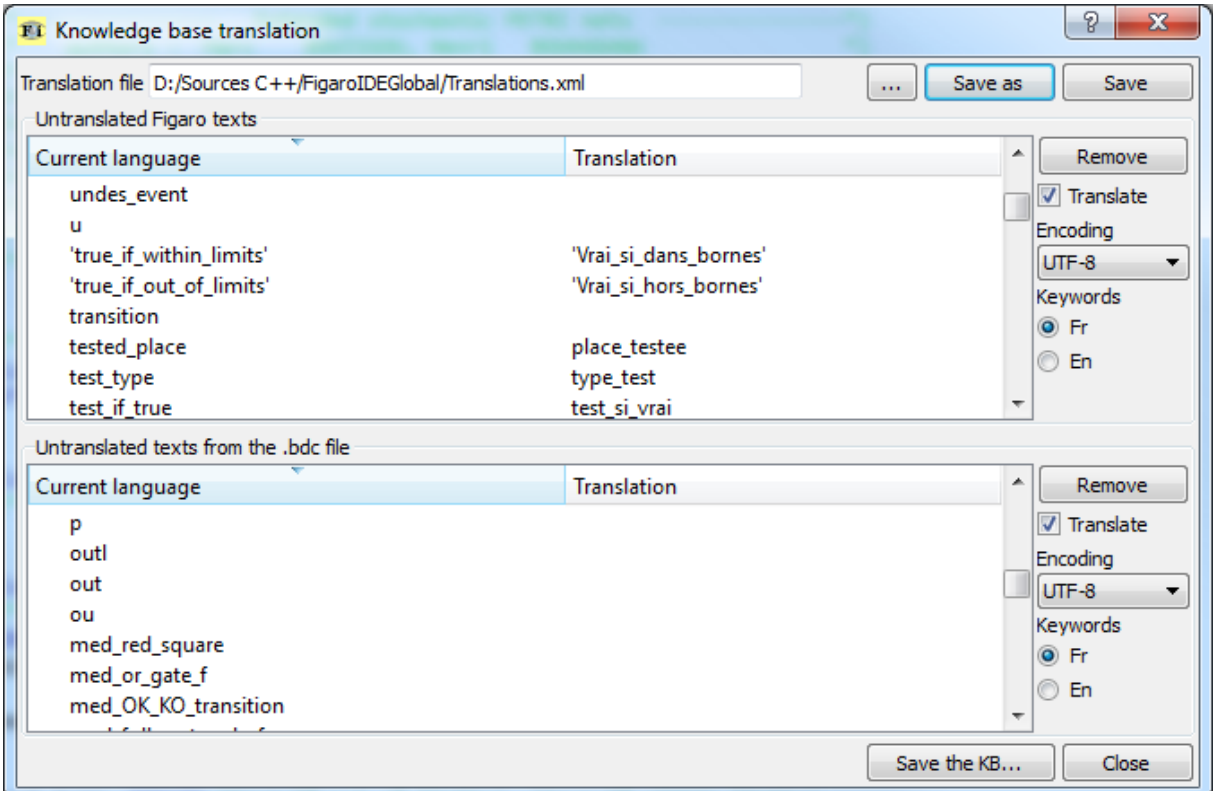


Figure 24: Knowledge base translation tool

To save the translated knowledge base, you must click on the "Save the KB" button located at the bottom right of the KB translation tool. The tool systematically asks in which file the knowledge base should be saved.

The Translate checkboxes allow you to choose the elements whose texts must be translated: Figaro file and/or XML file.

Before saving, it is possible to choose the language of the Figaro or XML keywords (radio buttons " Fr " and " En ") and to choose the encoding of the Figaro and XML files. These modifications have no influence on the initial knowledge base.