

# Generic tools

## *Sequences generator and a step by step simulator*

---

Copyright © 2008 Dassault Systemes

Version 1.0

## *Abstract*

The chapter deals with plugins needing a step by step simulator to work. This simulator can be select in order to choose the one to be used to do the action of plugin.

These plugins use step by step simulator for specific uses:

- Sequences generator search for every sequences below a given order (number of events in sequences) leading to a non-wanted system state (feared event), from an initial state of a system. This search is made with a depth-first search or with a specific strategy. To do that, generator will use step by step simulator to fire transitions, to validate feared-event presence, to go back, ...
- FMEA generation help aims at synthesizing consequences of unitary failures on models. Overall principle consists in starting from an initial state and firing each model transition, and then display every model variables.
- Temporal propagation visualization help function allows user to have information about the progression of the propagation of a functional or dysfunctional flow in a system following an unitary failure. Overall principle consists in starting from an initial state and firing a stochastic event, then let the step by step simulator to fire transitions according the Dirac values. After each step, we will collect the flows whose values have changed and increment the rank of the propagation.
- Sequences checking enables to replay result of a sequence generation on current model. There are many utilities. First, it enables to validate that sequences found with a sequences generator (from BPA DAS or other tools) lead to a top event of the system. Secondly, it can be used to verify that a slightly modified model behaves the same way with a set of sequences.

# Table of Contents

---

<b>Sequences generation (Generic) .....</b>	<b>4</b>
Sequences generation launching .....	4
Result of sequences generation .....	8
User preferences .....	8
<b>FMEA generation help .....</b>	<b>9</b>
FMEA generation help launching .....	9
Result of FMEA generation .....	10
<b>Temporal propagation help .....</b>	<b>12</b>
Temporal propagation help launching .....	12
Result of temporal propagation .....	13
User preferences .....	14
<b>Verification of sequences .....</b>	<b>16</b>
Launch verification of sequences .....	16
Results of verification of sequences .....	16

# 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).

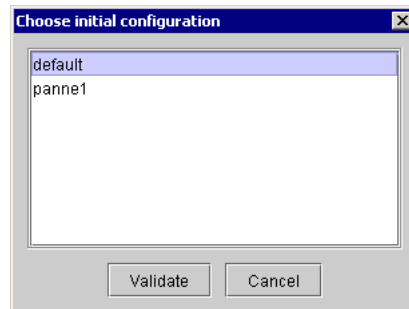
Processing time is proportional to the system size, but exponential to the depth of the ridding through. It implies a long processing time, it's the reason why this method have to be used only when tree generation can't be used (dynamic system and/or looped system).

## Sequences generation launching



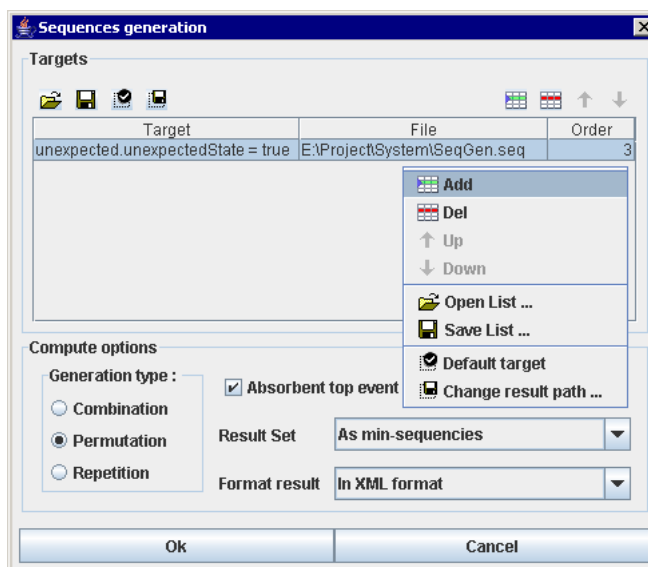
In order to launch sequences generation, use the **Sequences generation (Generic)** command

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 exemple, 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.



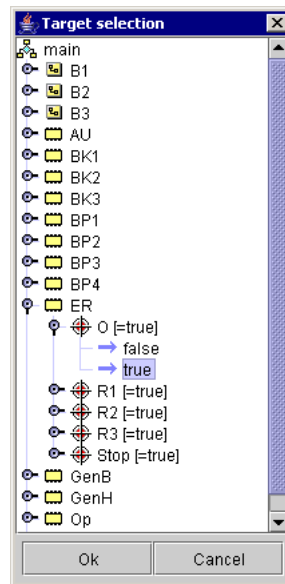
This window manage a list of computation-targets. When this window is validate, every typed information are validate to ensure that there is no incomplete line, no wrong line, targets correspond to a variable of processed model, file can be written ....

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

Some icons/actions enables 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 choose 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.

- **Combinaison:** 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 memorised.
- **As min-sequences :** Only minimal sequences will be memorised.
- **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
```

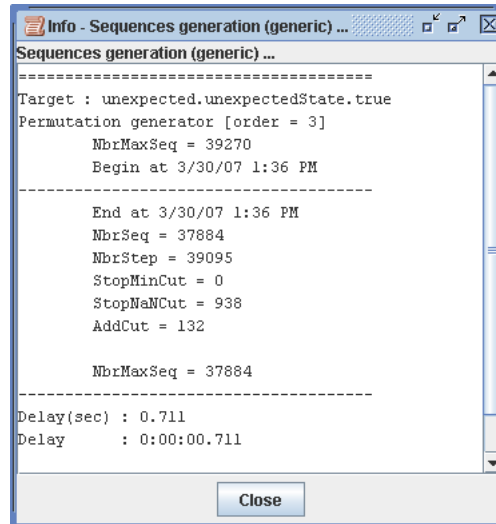
- **In Aralia format** : Result file contains found sequences in Aralia equation syntax (with laws, parameters and attributes).
- **In XML format** : Result file contains found sequences in XML format included parameters computation, abstract of result, model informations, ...

```
<?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 BPA DAS

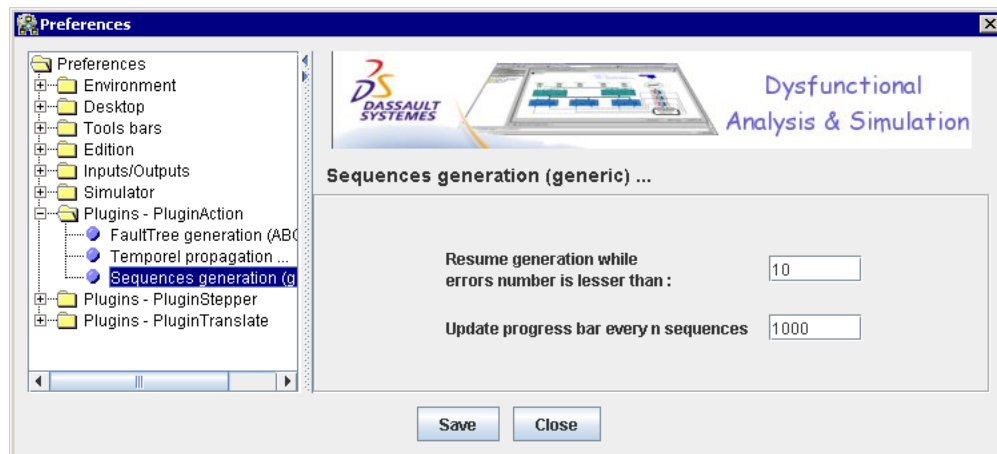
## Result of sequences generation

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 transition)

## 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'.



# 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.

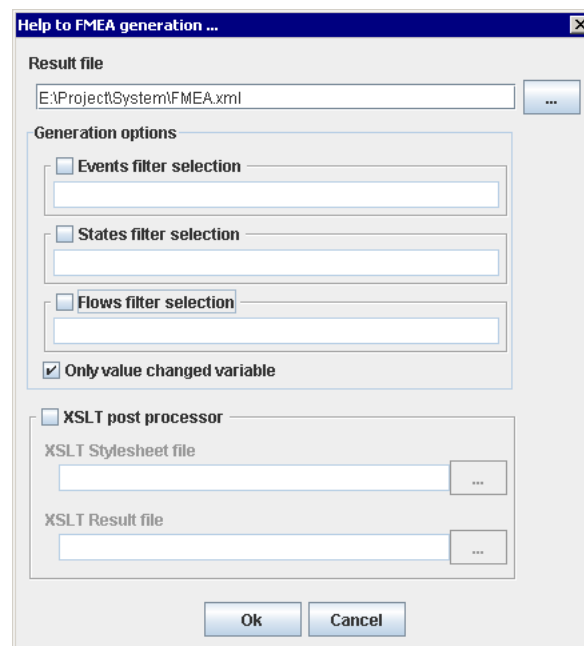
Overall principle of algorithm consists in starting from an initial state and firing each model transition, and then display every model variables.

## 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 et 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)
'$Equi\.' => Everything beginning whit 'Equi'. (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.

A post processing can be ask by ticking **XSLT Post-processor**. If this box if ticked, a XSLT style sheet and a result file must be specified.

If no important error appears during generation, style-sheet is applied on generated file, and the result of this post-processing is stored in the specified result file.

When this window is correctly filled and validated, **GenFMEA** task is added to the task manager of BPA DAS. On a standard system, this task shouldn't take a long time.

## 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>
```

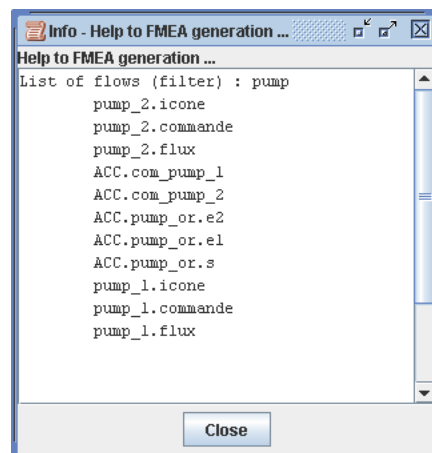
```
<flow name="B1.BNeg" domain="KTest_Potentiel" orientation="in">
  <init value="Neg"/></flow>
...
<state name="AU.Def" domain="bool">
  <init value="false"/></state>
<state name="B1.EA.Def" domain="bool">
  <init value="false"/></state>
...
<event name="AU.def">
  <law value="constant(1.0)" aralia="constant 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.

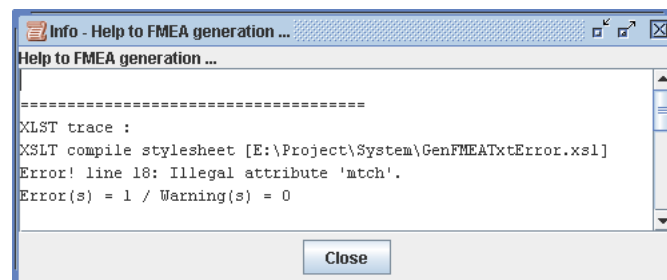
It displays event name of each transition followed by the value of each system variable (here, having their value changed).

To end, it displays informations 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:



If XSLT post-processing has been selected, warnings and/or errors may appear. In this case, the following window displays these informations:



# Temporal propagation help

Temporal propagation help aims at synthesizing ordered transient values of flows consequences of component-elementary-failure.

These changes of values are done through the use of deterministic event in models in order for the simulator to do inventory of changed values at each step.

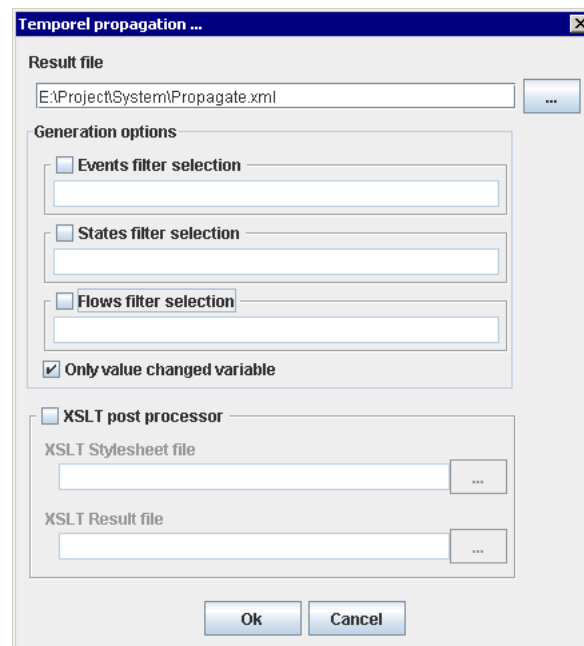
The rule used to order these transient values is based on the notion of rank. The rank is increased after each step of the simulator.

## Temporal propagation help launching



In order to launch temporal propagation, use the **Temporel propagation ...** command.

The following dialog window enables to define temporal propagation 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 et 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)
'$Equl\.' => Everything beginning whit 'Equl'. (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.

A post processing can be ask by ticking **XSLT Post-processor**. If this box if ticked, a XSLT style sheet and a result file must be specified.

If no important error appears during generation, style-sheet is applied on generated file, and the result of this post-processing is stored in the specified result file.

In case of temporal propagation, the **Only value changed variables** check-box filters results in order to hide variables whose transient values have not changed compared to the value at initial state. This check-box must be selected.

When this window is correctly filled and validated, **Propagate** task is added to the task manager of BPA DAS. On a standard system, this task shouldn't take a long time.

## Result of temporal propagation

If everything is all right, task ends without error.

Result file is like:

```
<?xml version="1.0"?>
<propagategen>
  <define>
    <param name="select.state" value="\s" />
    <param name="onlychange" value="true" />
    <param name="limit.delay" value="1.0" />
    <param name="limit.rank" value="100" />
  </define>
  <result>
    <tr id="9" evt="equ7.evt">
      <rank idx="1" time="0.0" >
        <flow name="equ2.b" value="true"/>
        <flow name="equ3.a" value="true"/>
        <flow name="equ7.a" value="true"/>
        <flow name="equ7.b" value="true"/>
      </rank>
      <rank idx="2" time="1.0" >
        <flow name="equ11.g" value="true"/>
        <flow name="equ2.d" value="true"/>
        <flow name="equ2.g" value="true"/>
        <flow name="equ3.c" value="true"/>
        <flow name="equ4.c" value="true"/>
        <flow name="equ4.d" value="true"/>
      </rank>
      <rank idx="3" time="2.0" >
        <flow name="equ11.f" value="true"/>
        <flow name="equ18.e" value="true"/>
        <flow name="equ4.e" value="true"/>
        <flow name="equ4.f" value="true"/>
      </rank>
      <rank idx="4" time="3.0" >
        <flow name="equ11.h" value="true"/>
        <flow name="equ11.i" value="true"/>
      </rank>
    </tr>
  </result>
</propagategen>
```

```

    <flow name="equ18.h" value="true"/>
  </rank>
<rank idx="5" time="4.0" >
  </rank>
</tr>
</result>
<model>
  <flow name="equ11.f" domain="bool" orientation="in">
    <init value="false"/></flow>
  <flow name="equ11.g" domain="bool" orientation="in">
    <init value="false"/></flow>
  <flow name="equ11.h" domain="bool" orientation="in">
    <init value="false"/></flow>
  <flow name="equ11.i" domain="bool" orientation="out">
    <init value="false"/></flow>
  ...
  <event name="equ7.evt"></event>
</model>
</propagategen>

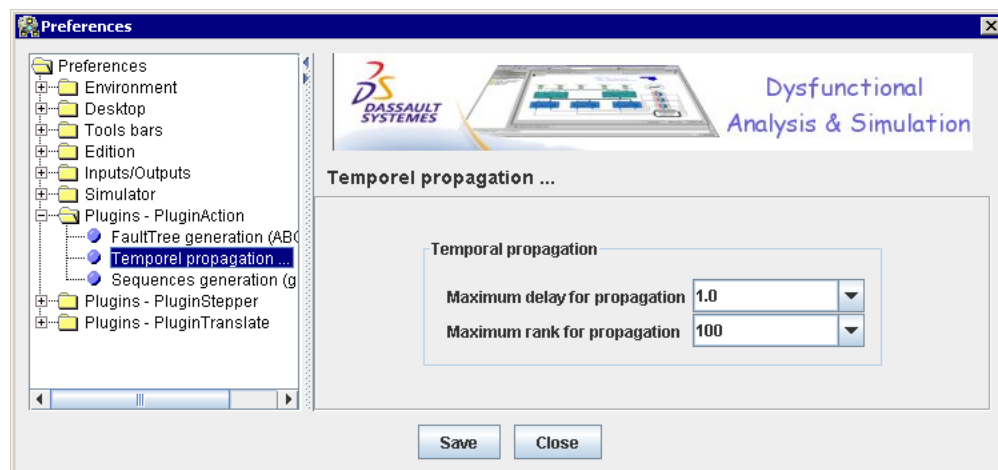
```

It displays first the parameters of the study.

Then, for each event, it displays by rank the temporal value and a list of variables (whose value has changed) with its value.

To end, it displays informations about system variables and events implied in these results.

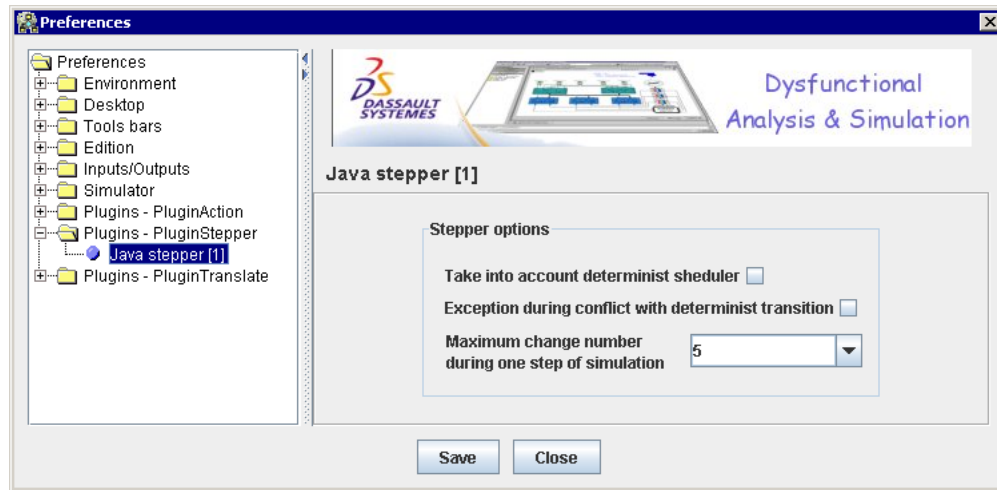
## User preferences



Some user preferences allows to configure the temporal propagation.

- Maximum delay for propagation allows to filter on deterministic events whose value is greater than this maximum delay.
- Maximum rank for propagation prevents from infinite deterministic event firing.

It is highly recommended to unchecked the option about raising exception when there is a conflict between deterministic events. This exception is raised when there is no priority on these events when there is a conflict. In our study, these priorities are not necessary.



# 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).

Informations 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 initialised 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.

## Launch verification of sequences

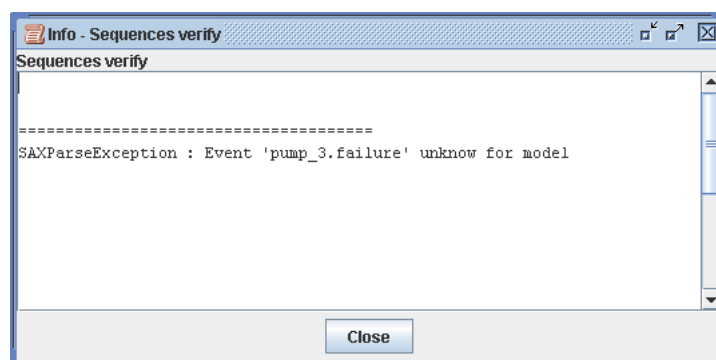


In order to launch sequences checking, use **Verification of sequences ...** command.

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 BPA DAS.

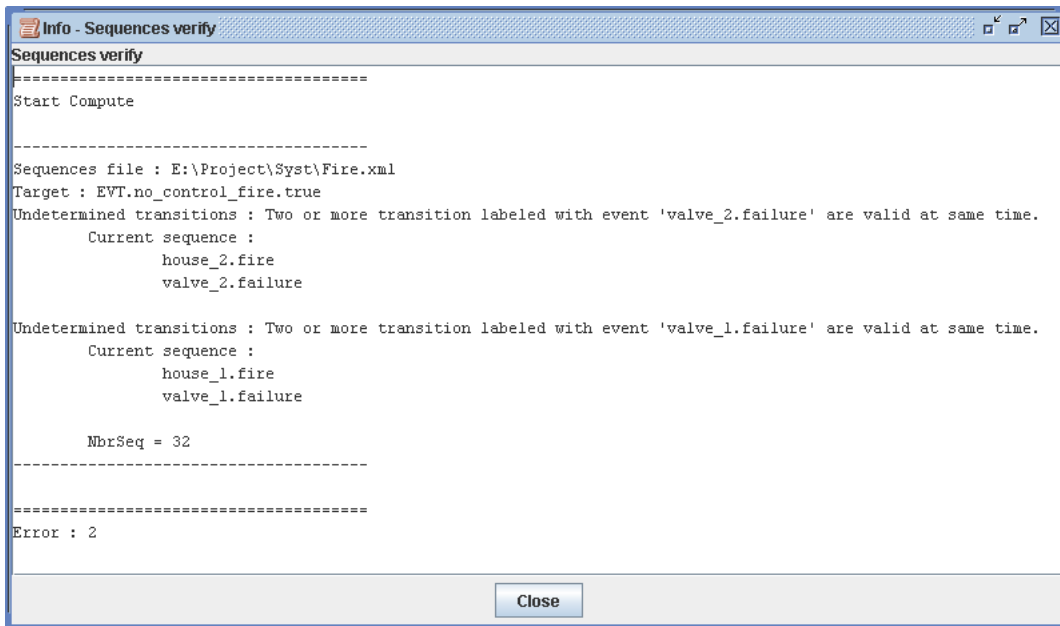
## Results of verification of sequences

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.

