MODICA Manual

MODICA Version 5.4.0

What's MODICA?

MODICA is an Eclipse-based tool for creating usage models of a test object (**modeling**).

From these usage models, test cases can be automatically deduced based on the possible sequences of interaction with the test object (**test case generation**).

The generated test cases can be transferred to an automatic test execution environment and are immediately ready for use in the target system (**test execution**).

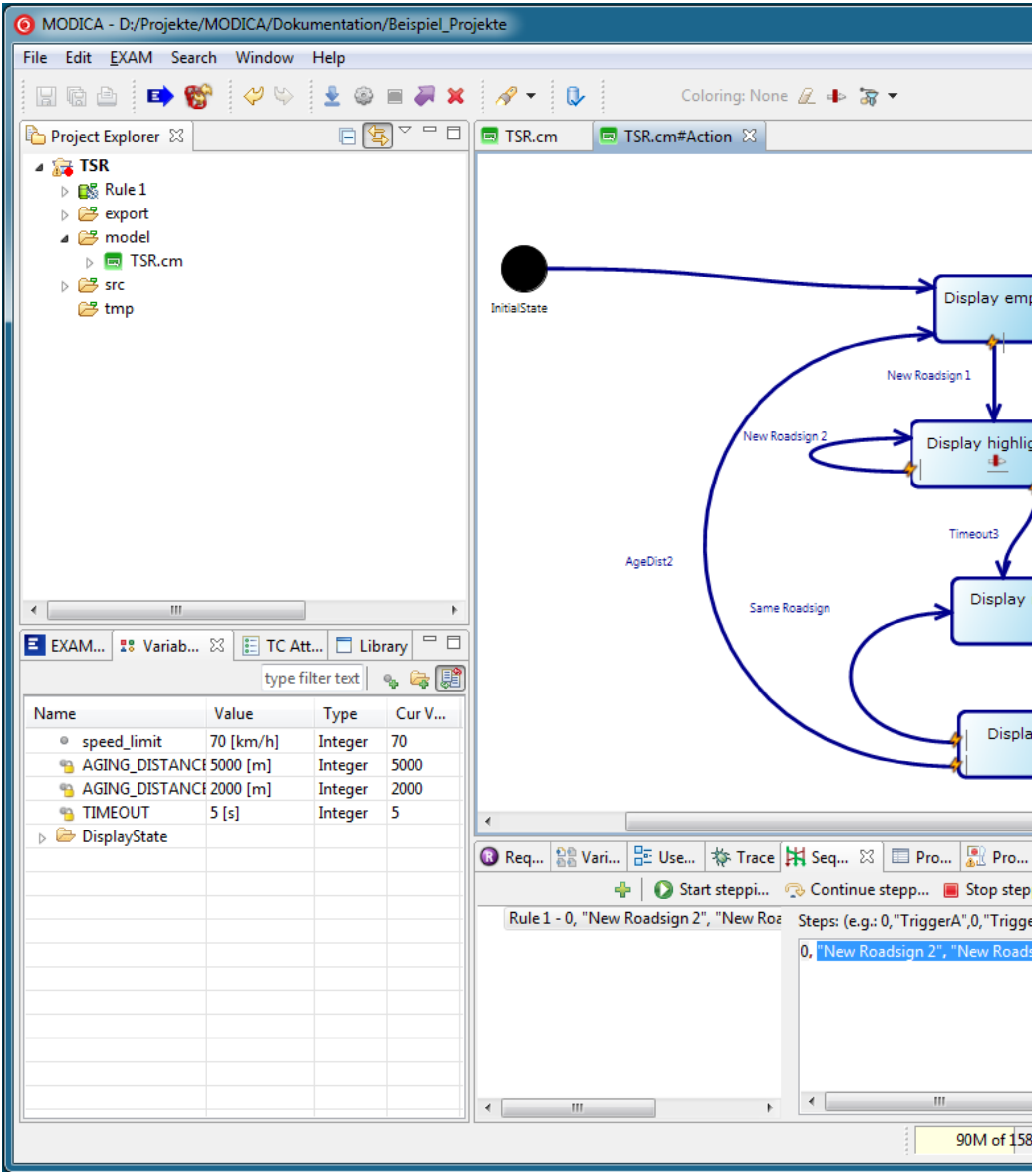
Important!

All functions listed in this manual are only available in the full version of MODICA. Other MODICA versions may lead to limitations in functionality.

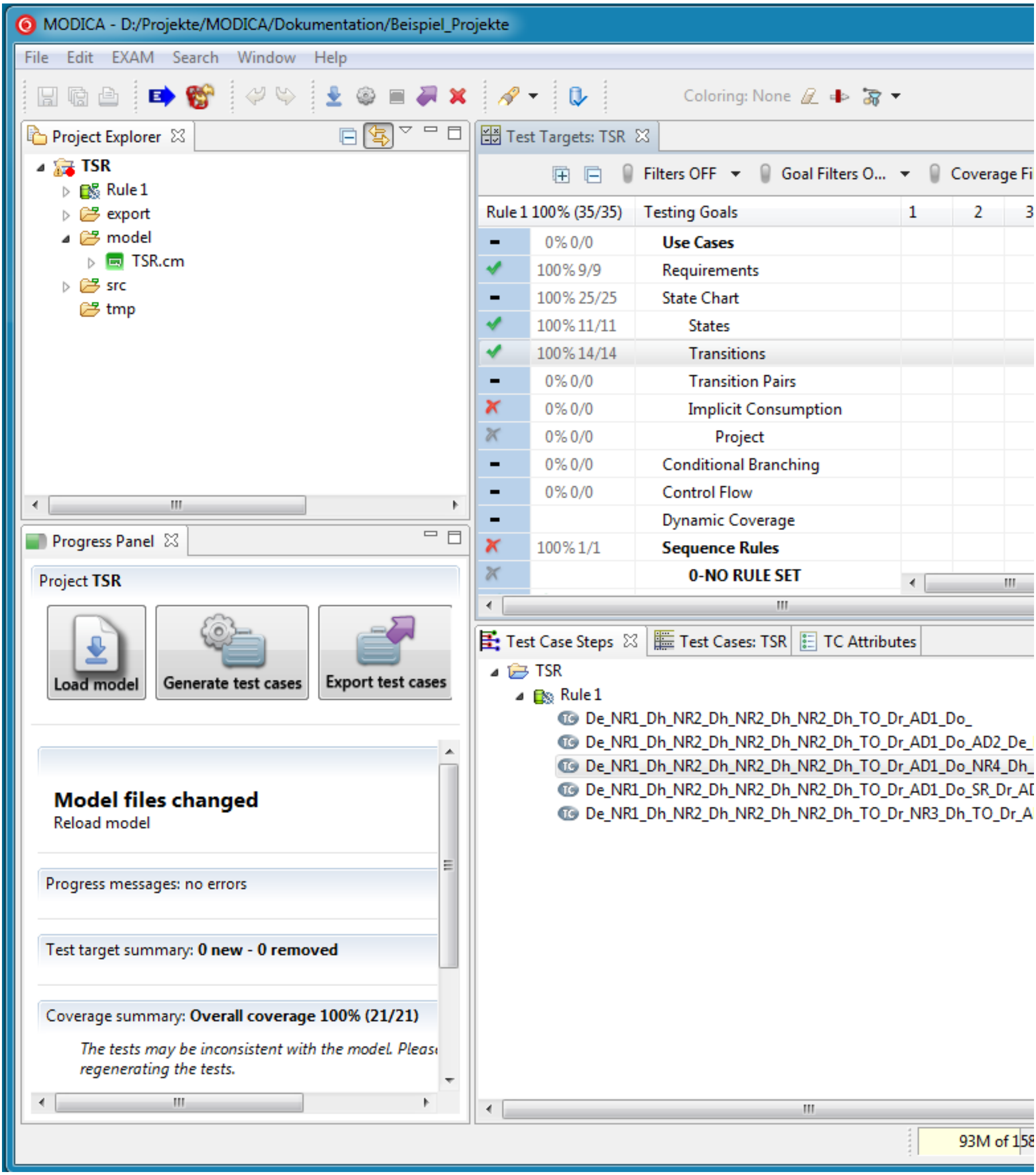
Overview MODICA Perspectives

MODICA basically consists of perspectives that can be used:

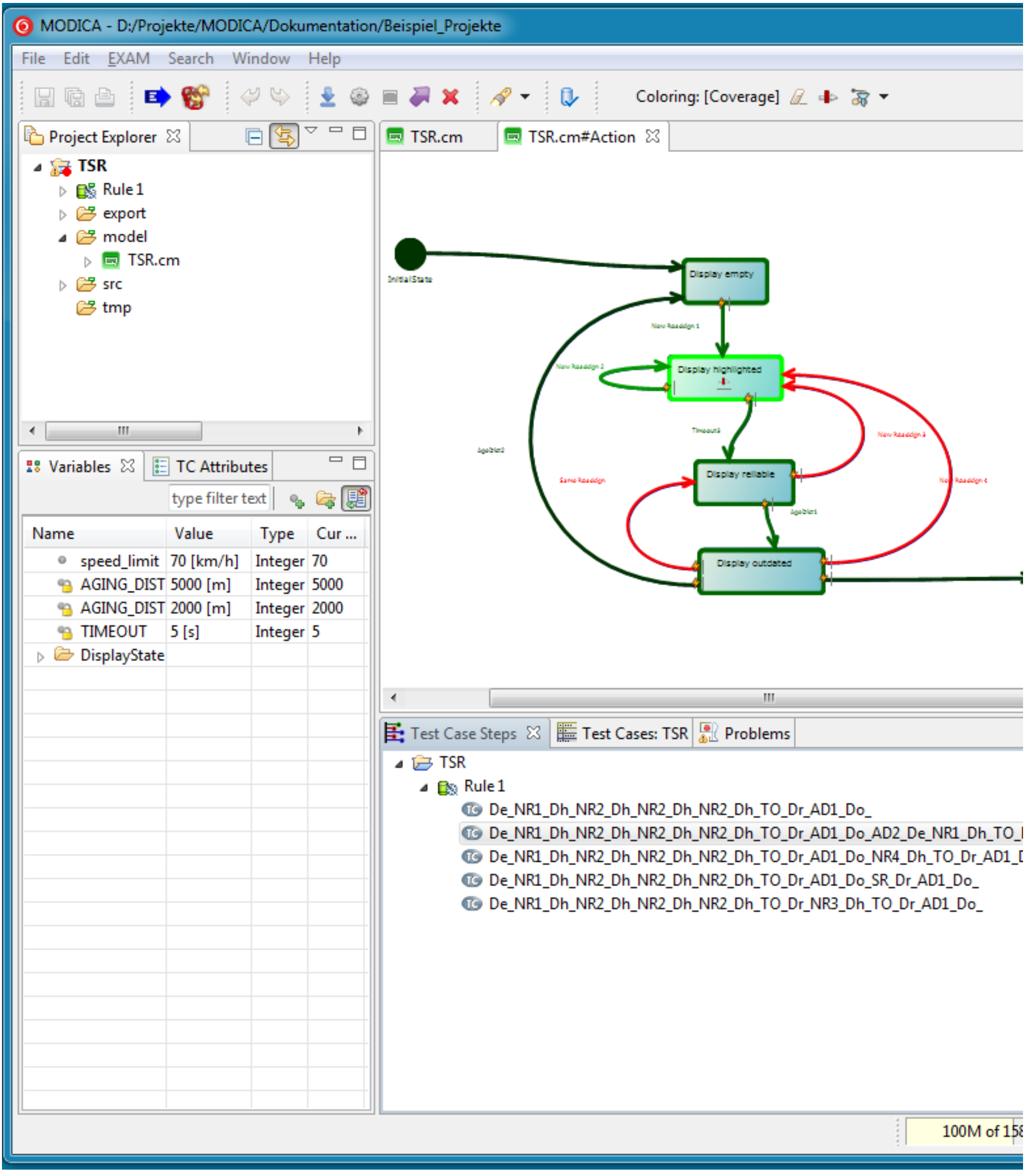
- The **Modeling Perspective**, in which
  - the system to be tested is modeled as a usage model
  - the requirements are assigned
  - the requirements used are analyzed
  - the variables are managed
  - the objects of the test-framework are integrated and used
  - the objects used are analyzed in the usage model
  - variants can be defined and managed



- The **Test Generation Perspective**, in which
  - the generation strategies are defined
  - the created model is checked syntactically
  - test cases are created
  - test cases are exported to the target test system



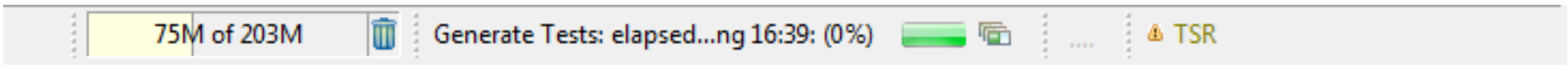
- The **Analysis Perspective**, in which
  - the generated test cases can be analyzed
  - the test results can be compared to the model



Switching between the perspectives can be done via the two switches in the top-right corner of MODICA:

The status bar at the bottom displays additional information about the state of the current project and MODICA in general.

- Current RAM/Heap usage (excluding test case generation)
- Currently running tasks such as testcase generation
- Errors or warnings that may have happened in MODICA
- The currently active project and if it contains any problems or warnings.



Modeling Perspective

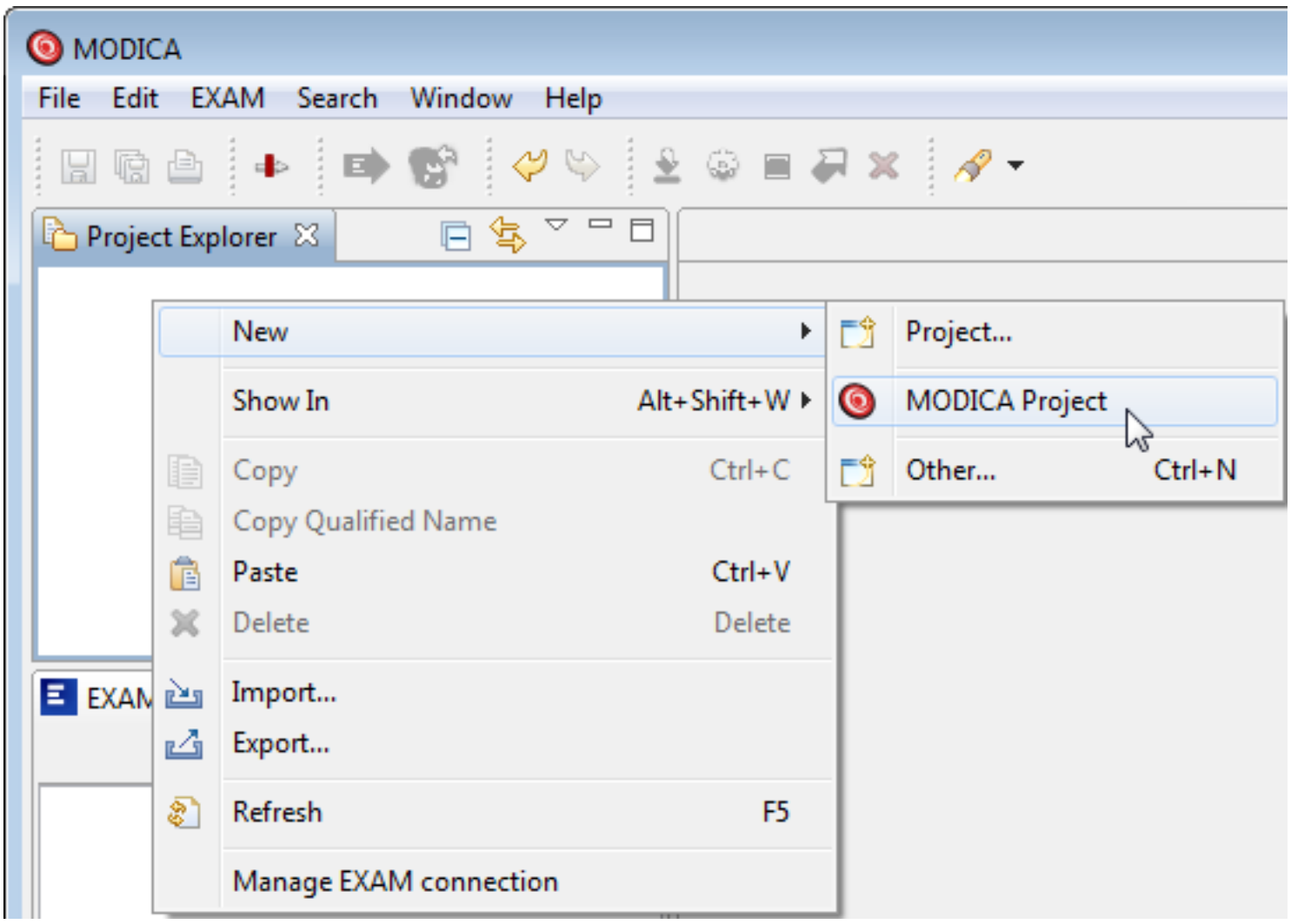
The following pages describe the most important views of the *modeling perspective:*

Project Explorer (en)

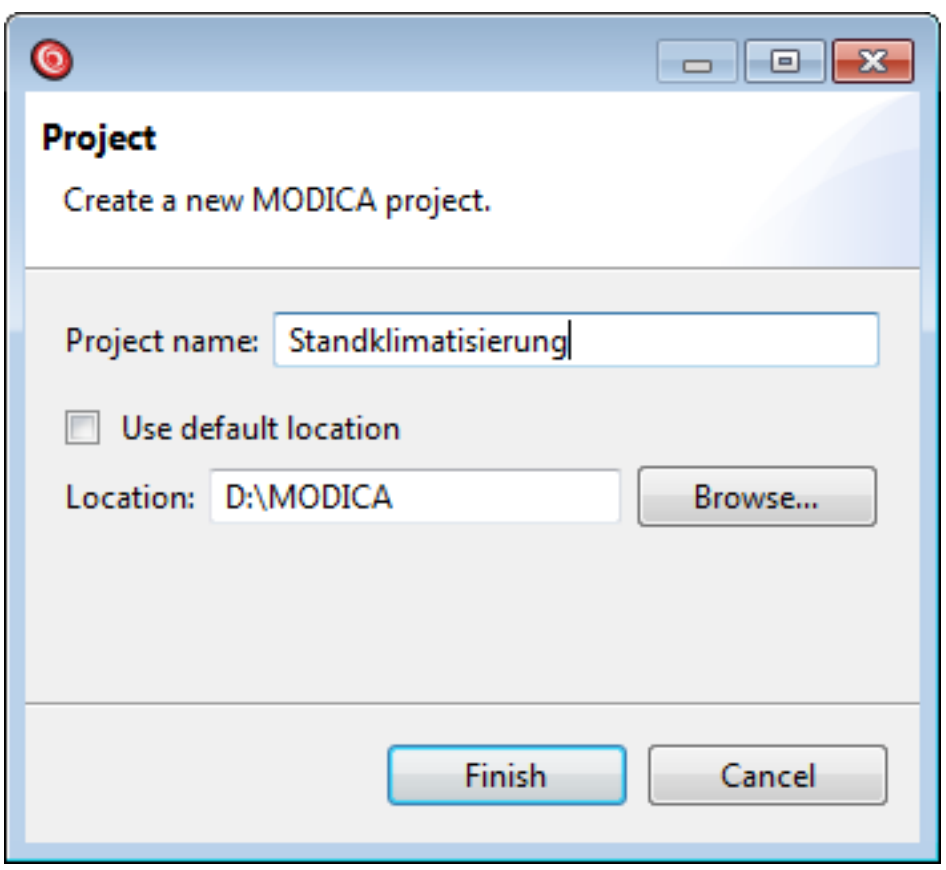
The *Project Explorer* lists all available projects and offers the possibility to create new projects.

Create a new project

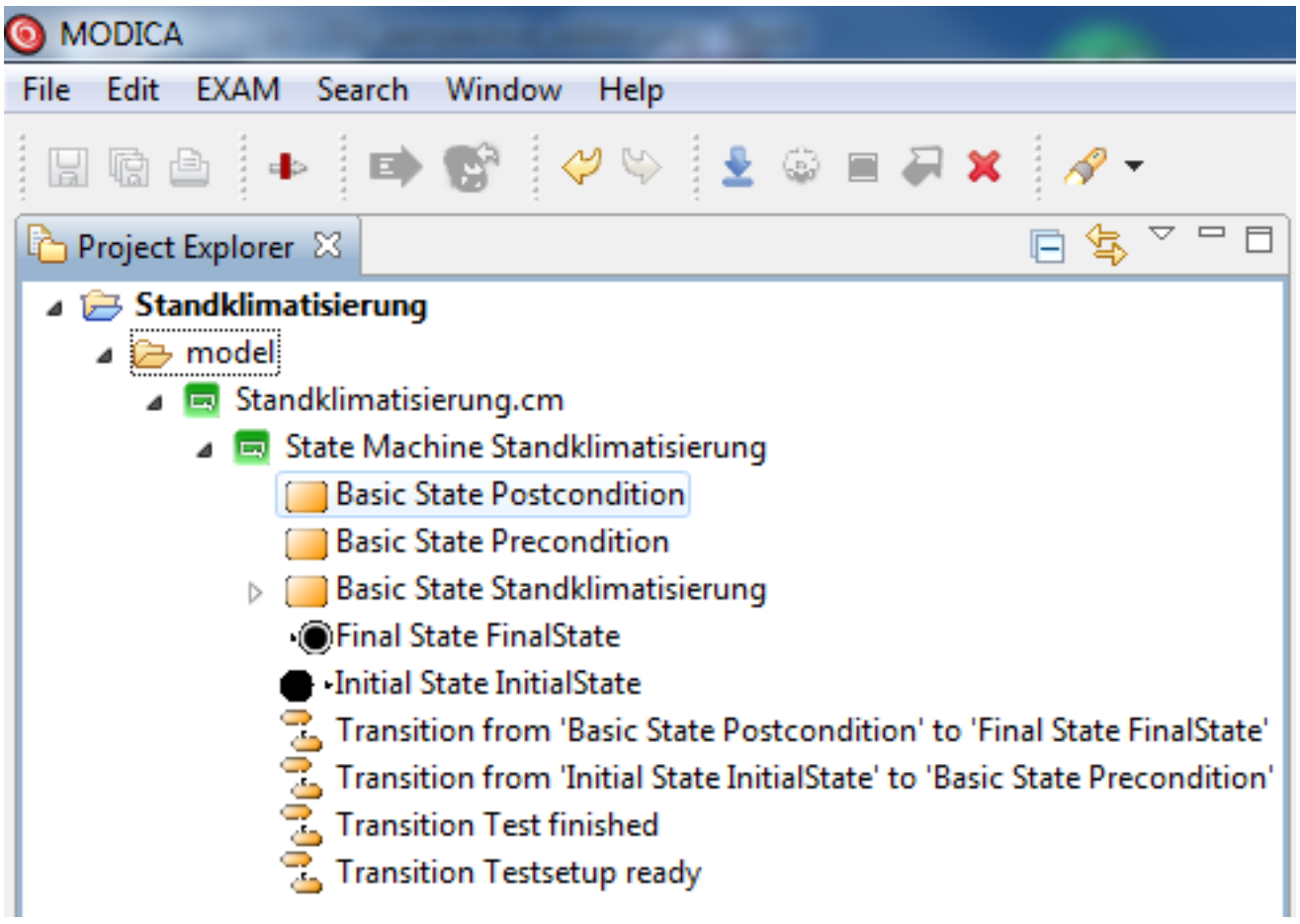
To create a new MODICA project, right-click on *Project Explorer* and select *New → MODICA Project* in the context menu.



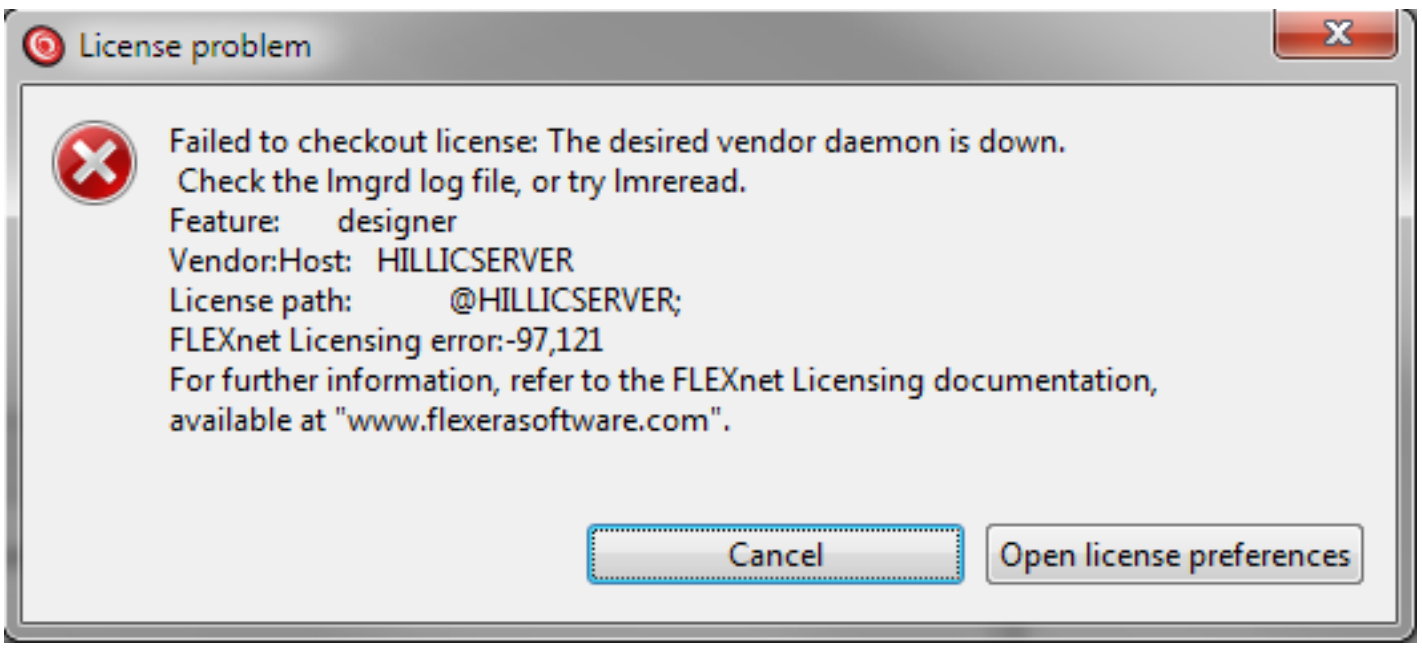
In the following dialog, the name of the project as well as the location of the workspace, in which the project will be stored on the drive, can be specified.



When everything is set as desired, the dialog can be confirmed with *Finish* and thereupon the project is created in the selected workspace. By clicking on the white triangle at the side of the newly created project, all objects of this project will be displayed in the project tree:



If an object in the *Project Explorer* is selected for the first time after opening MODICA, a license for MODICA is required. If no license can be obtained, an error message, similar to the following, appears:



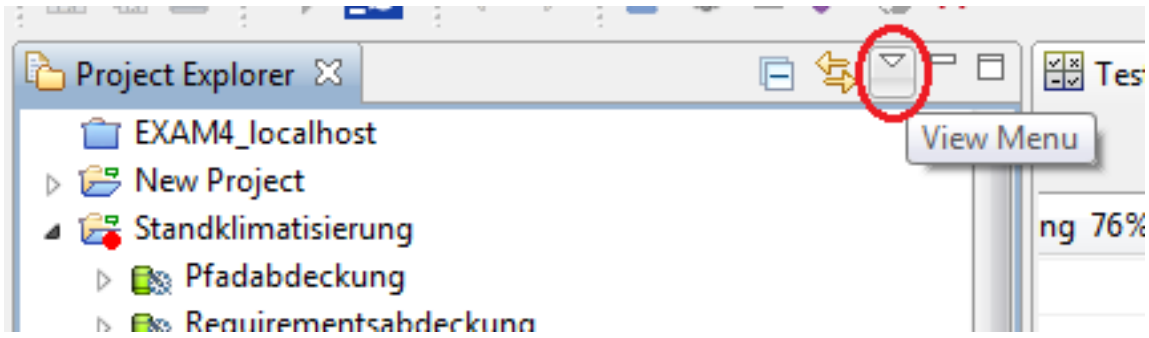
Filter the displayed items in the Project Explorer

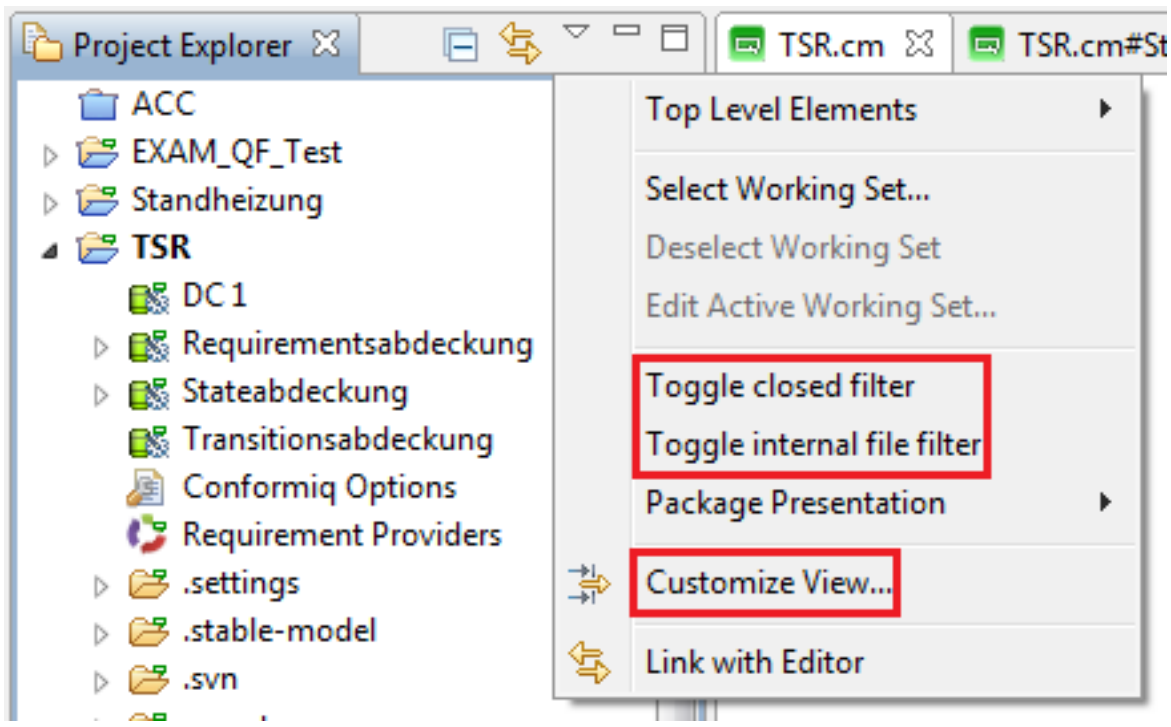
There are several ways to tailor the contents of the *Project Explorer* to the user-relevant information.

On the one hand, this is done via the View Menu  and

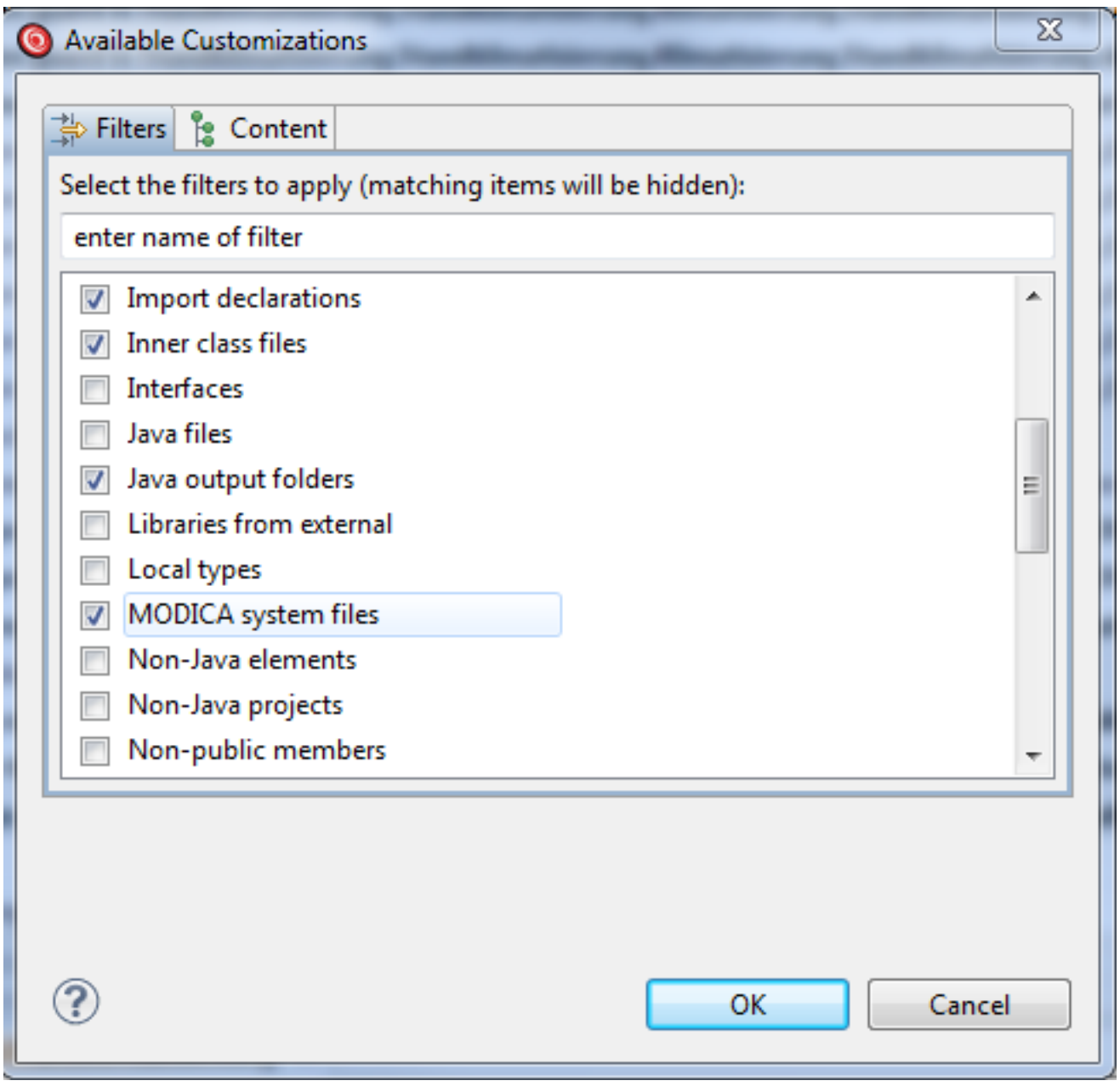
- *Toggle closed filter*: closed projects are shown/hidden
- *Toggle internal file filter*: MODICA-internal elements are shown/hidden

On the other hand, you can specify the display more precisely using *View Menu → Customize View*.





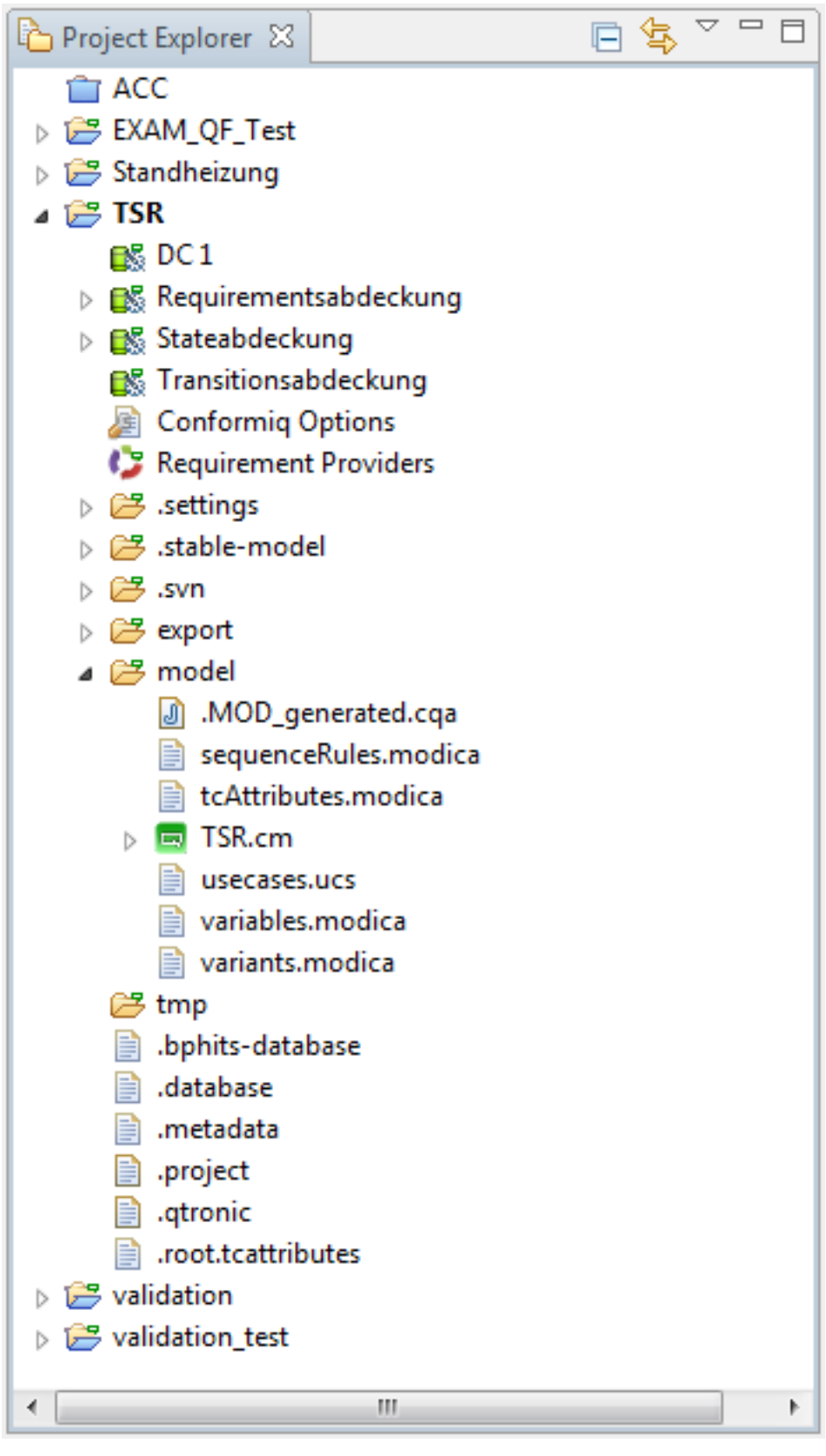
The *Filters* section of the appearing dialog shows all the specific filters (see picture below). For example, activating the filter *MODICA system files* has the same effect as *Toggle internal file filter*.



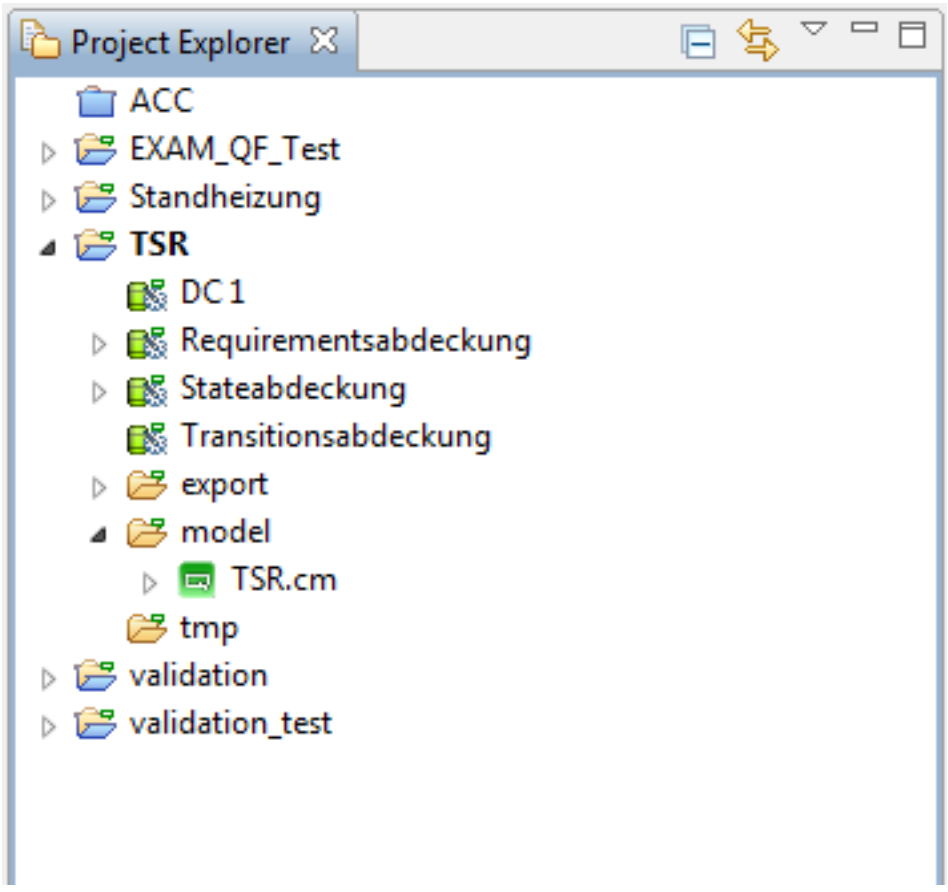
This will filter all MODICA system files (`*.cqa, *.modica and .*`) out of *Project Explorer*. The difference between a

non-filtered *Project Explorer* and a filtered *Project Explorer* is as follows:

non-filtered:



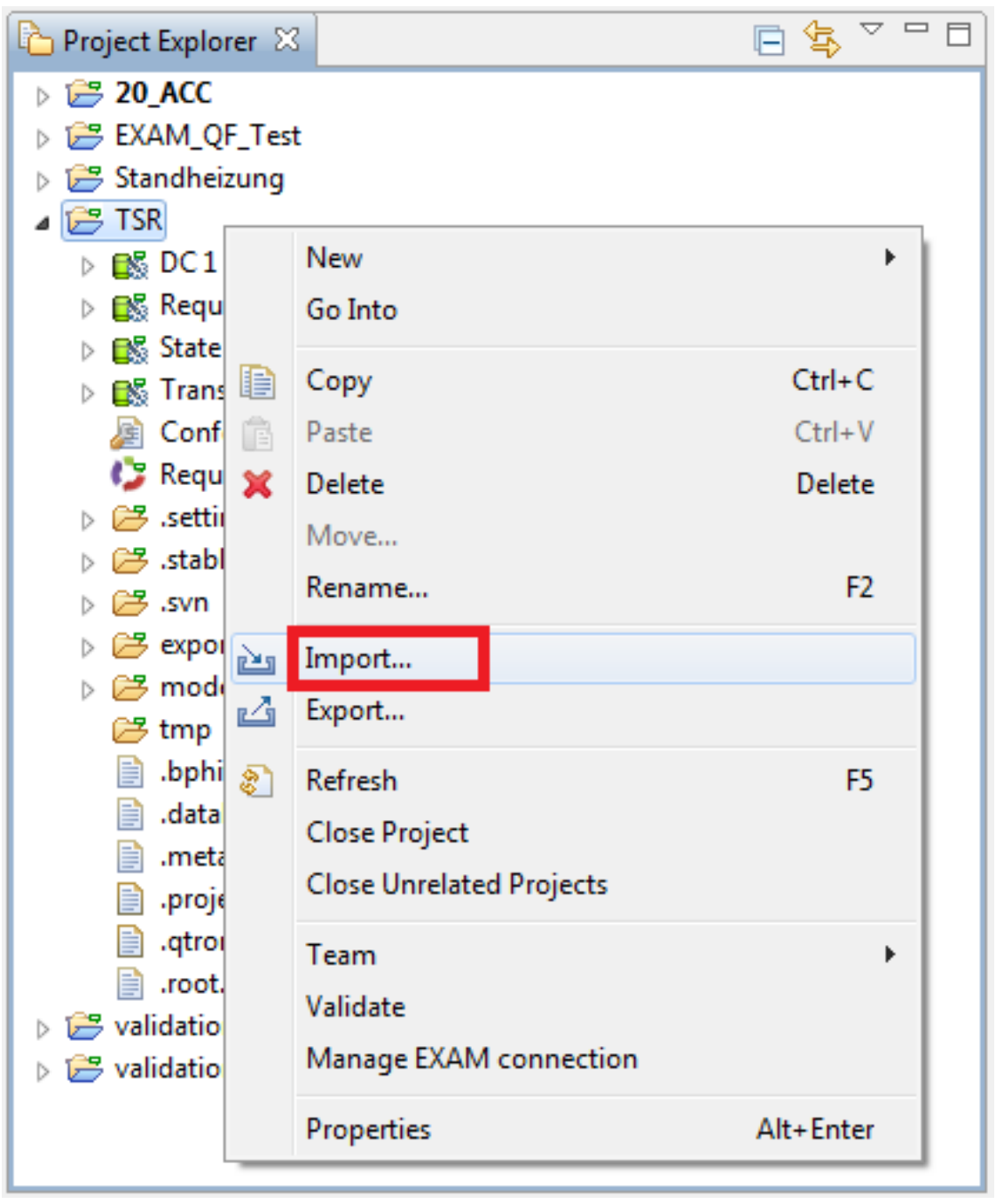
filtered:



Open and import additional projects

By *right-clicking* → *Import*, additional MODICA projects can be loaded into the workspace.

The release notes show the version number to which older projects are supported.



Project administration

By *right-clicking → Delete / Rename / Close Project* the project can be deleted, renamed or closed.

To perform these actions, all *Editor*-tabs belonging to the project must be closed.

The active project will be highlighted **bold** in the *Project Explorer*. The main criterion for project activation is an open statechart.

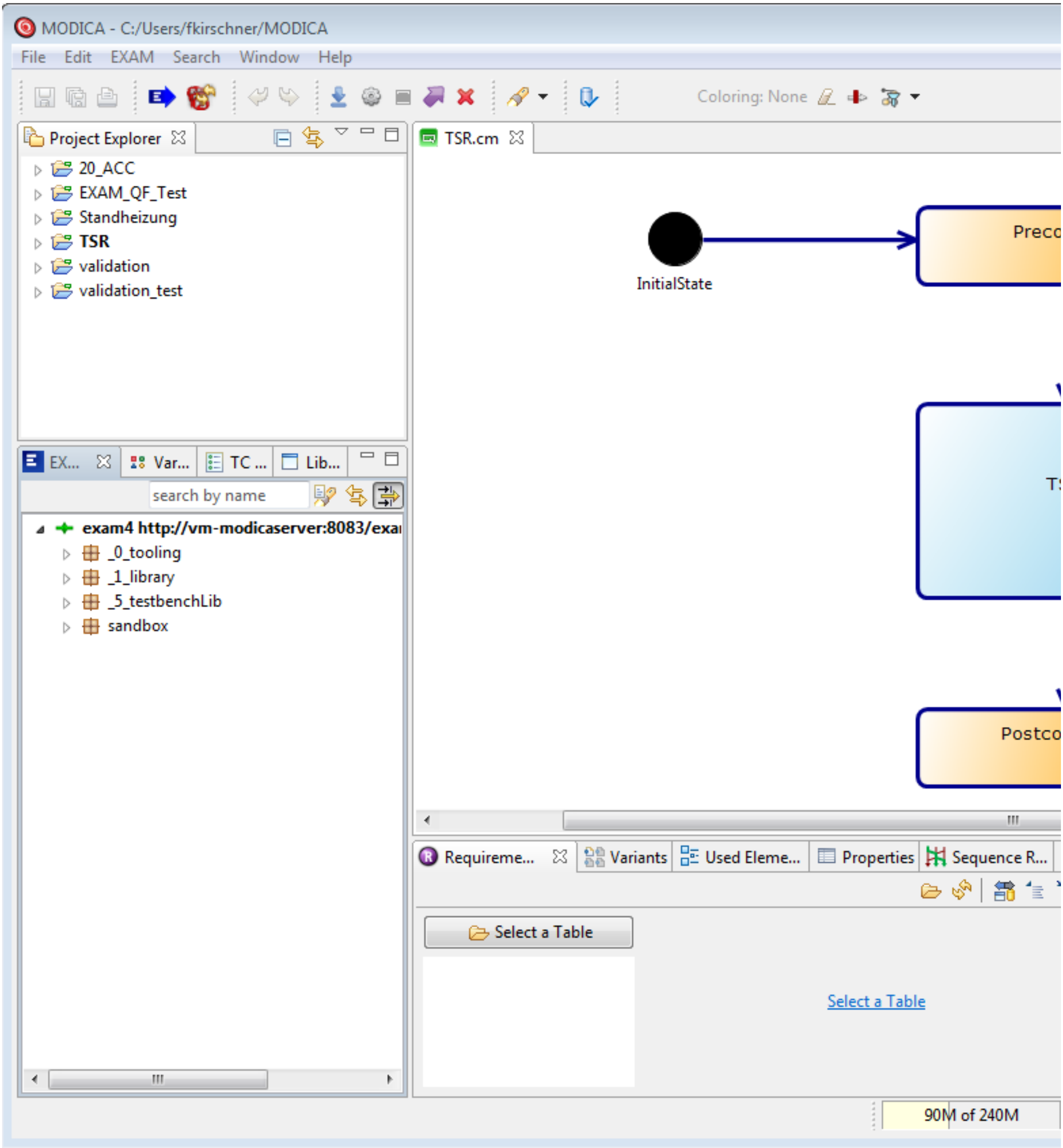
Editor (en)

This chapter discusses the *Editor* and the modeling tools in MODICA.

General

To display the modeling view of a project, the `.cm` file must be opened in the *Project Explorer*. In the sample project, this is the file TSR.cm.

This can be opened with a *double-click* or *right-clicking* it and selecting *Open* in the context menu. The modeling editor opens and displays the open model.

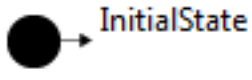


Elements of the *Editors* are:

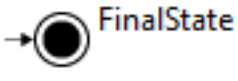
1. The modeling area
2. The palette with modeling tools

If you click on one of the modeling tools in the palette, the corresponding object will be added to the modeling area the next time you click. **HINT**: To add several identical objects one after the other, press the `CTRL` key while clicking in the modeling area. This will keep the tool after the insertion.

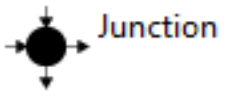
The most important modeling tools for modeling with MODICA are:



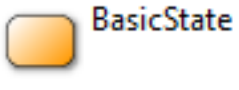
The InitialState is always the starting state of a model



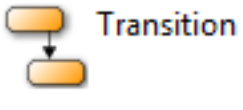
The FinalState is the ending state of a model



The JunctionState is an auxiliary state for simplifying the model

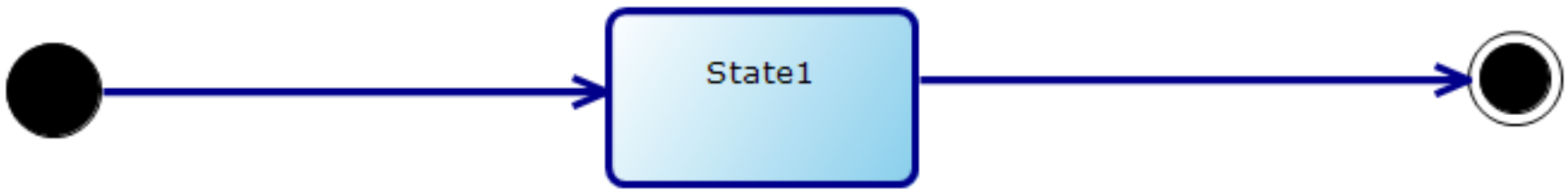


BasicState is the default state in a model



A transition from one state to another

The following model can be created by using these tools.



This is also the minimal expression of a MODICA statechart. In every MODICA statechart there must be:

- **Exactly one InitialState**
- **one or more BasicStates**
- **one ore more FinalStates** (**exception**: When using triggers on a higher level, FinalStates can be dispensed with)
- as well as transitions, which can connect the states to each other

States

A state is created in the model by activating the item for *BasicState* in the tool palette. Then the mouse is clicked where a *BasicState* should be created.

InitialState

In every MODICA statechart, there must be exactly one *InitialState*. It is the starting point of each model (or substates).

If you create a new *InitialState*, the following symbol appears in the model diagram:

A descriptive name can be assigned by double-clicking the state, for example:

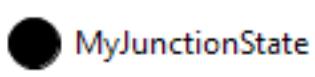
FinalState

In each MODICA-statechart, there may be several *FinalStates*.
It is the end point of each model (or substates).
If you create a new *FinalState*, the following icon appears in the model diagram:

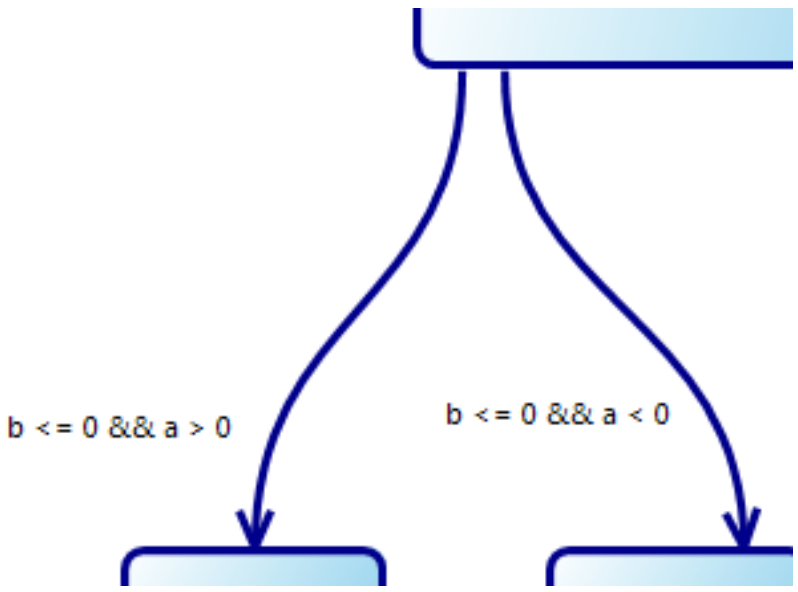
A descriptive name can be assigned by double-clicking the state, for example:

JunctionState

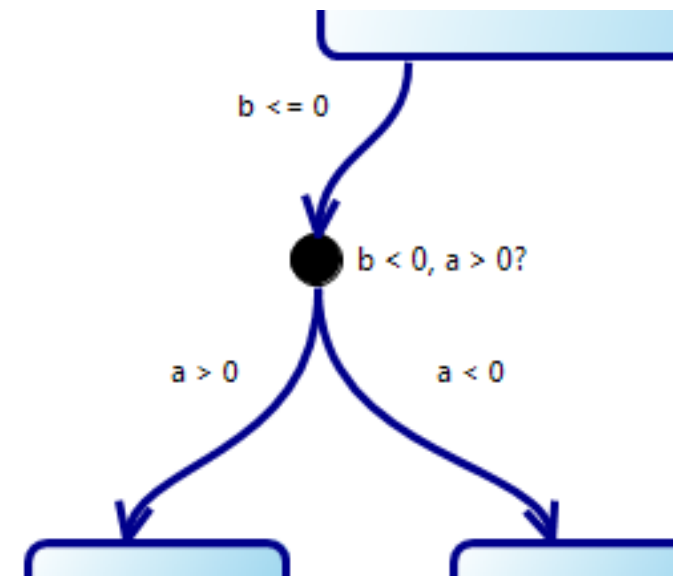
In a MODICA statechart, there may be *JunctionStates*. However, their use is optional. The *JunctionState* is a special state in which a system can not stay permanently. It has no inner behavior, but can be named. In the diagram inserted it looks like this:



The JunctionState simplifies the modeling of complex diagrams with many *transitions* and *guards*. The following figure shows an exemplary situation in which a *JunctionState* is used to split complex *guards* into simple ones.



without *JunctionState*



with *JunctionState*

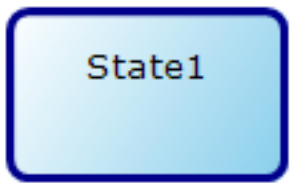
A special feature of JunctionStates is that no triggers can be used for outgoing transitions.

Apart from this aspect, *JunctionStates* are usable in many cases equivalent to normal states. The main difference is that the system can not stay in a *JunctionState*. If none of the outgoing *transitions* of a *JunctionState* can be taken, the system remains in the *BasicState* preceding it.

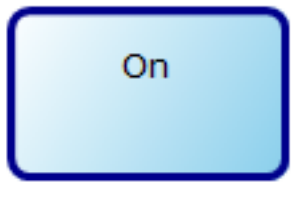
BasicState

In each MODICA statechart there must be at least one *BasicState*. There may be several *BasicStates* in a statechart and model.

If you create a new *BasicState*, the following symbol appears in the model diagram:



By double-clicking State1, a descriptive name can be given to this state:

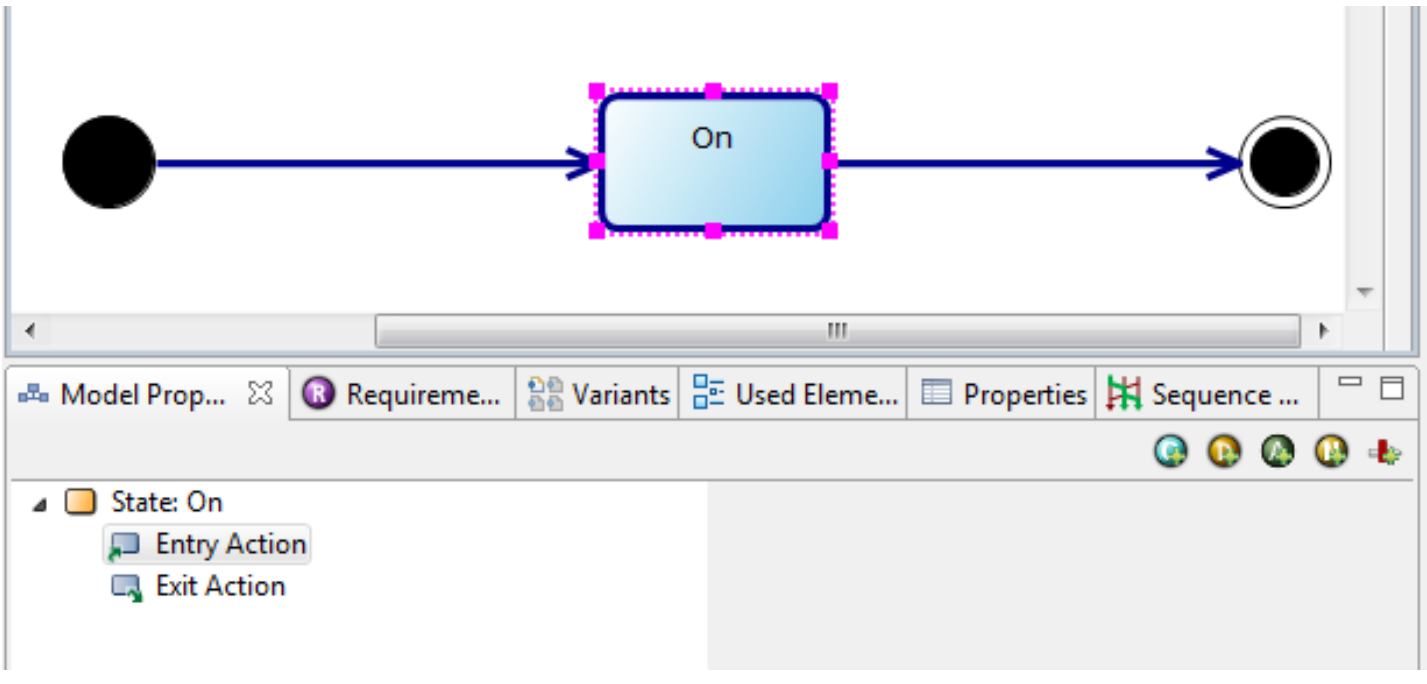


Compared to *Initial, Final,* and *JunctionStates, BasicStates* have properties that describe the behavior of the system in this state.

Each *BasicState* has an entry and an exit action.

- **Entry Action:** All actions in the entry action are performed when this state is entered.
- **Exit Action:** All actions in the exit action are performed when this state is exited.

Entry and exit actions of a *BasicState* can be accessed via the *Model Properties View* as soon as you select it in the model:



For detailed information have a look at the chapter Model Properties View.

Substates

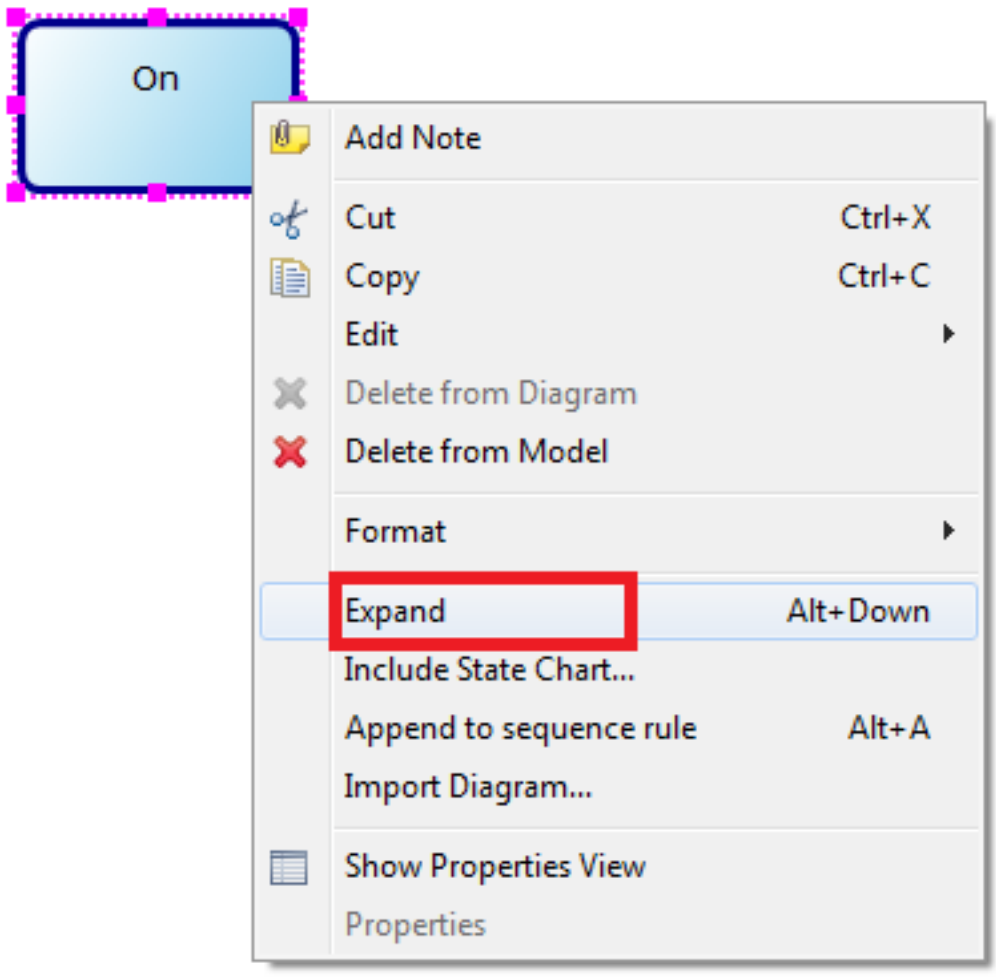
Each *state* in MODICA is a *Simple State* at the time of creation.

Any *Simple State* in MODICA can be expanded to a *Composite State.*

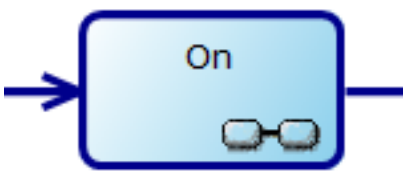
A *Composite State* is a state whose internal behavior is expanded by receiving / containing an internal state - a so-called **substate**.

A *Simple State* is turned into a *Composite State* by following these steps:

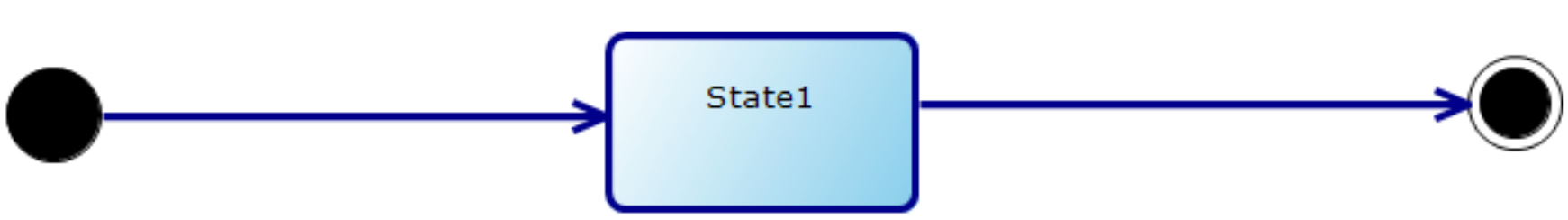
- The state which should receive a substate, is right-clicked and the *Expand*-entry is selected in the context menu



- This will turn the state On to a *Composite State* in this example and create a substate diagram within that *state*. The State2 is henceforth characterized by a spectacle-like symbol as a state with a substate diagram:



- This substate diagram can be opened by clicking on the spectacle symbol of the *state* On. This opens another diagram, parallel to the existing TSR.cm, named TSR.cm#State2,
- To make a *substate* syntactically correct, its state diagram also requires an *InitialState* and a *FinalState*. Therefore, each s*ubstate* diagram must be modeled at least in this form:



Transitions

Transitions in state diagrams are connections from one *state* to another. They are shown as arrows. In these *transitions*, actions can happen or be carried out.

A *transition* is created as a link between two already existing *states*. To do this, select the symbol for the *transition* in the tool palette and, with the mouse button pressed, connect two *states* in the sequence "State_from_which_the_transition_comes" -> "State_into_which_the_transition_leads".

Labels

Each *transition* has a name, called label, similar to those of *States*. This label is fixed to a *transition* and can also be empty. When the *transition* is created, this name is empty.

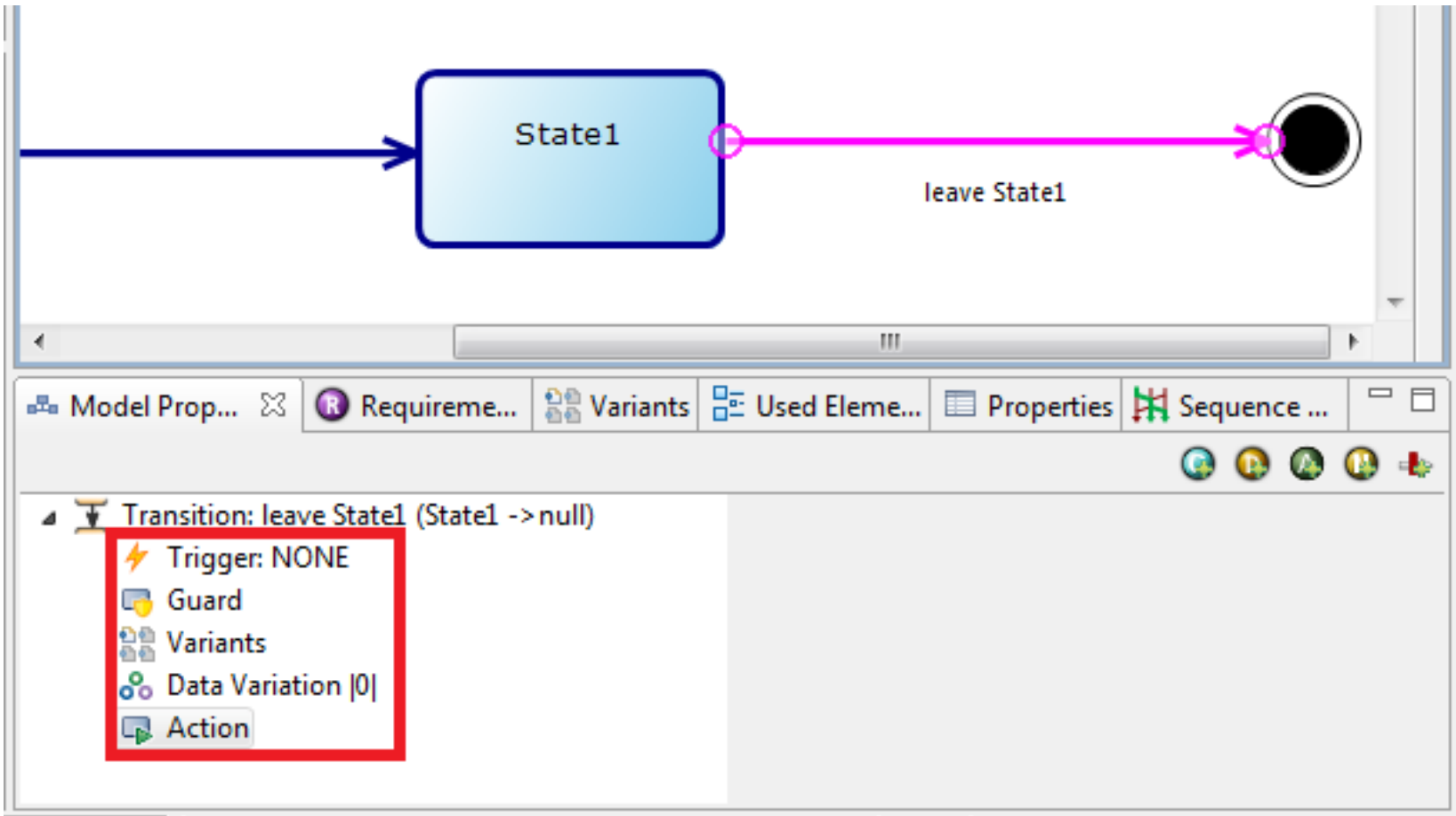
By selecting the transition and double-clicking or pressing the F2 key, it can be renamed in a descriptive name, e.g., leave State1:

Components of a transition

Each *transition* in MODICA consists of five parts:

- Trigger
- Data
- Guard
- Variants
- Action

These five components can be accessed via the *Model Properties View* after selecting a *transition* in the model:



In the following, different elements of a *transition* are briefly explained and their use is described in an exemplary manner.

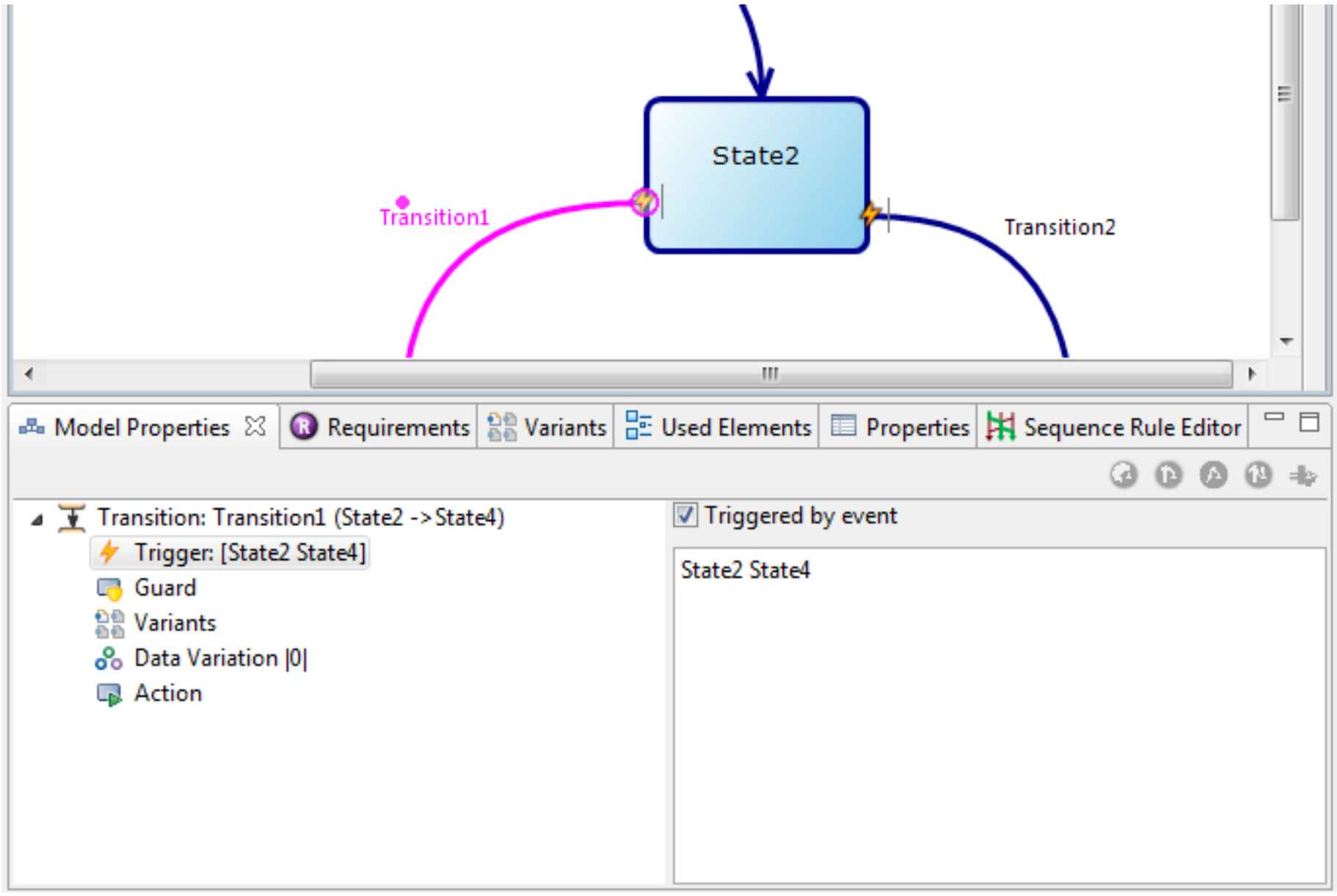
Trigger

In UML, a *trigger* is the event, which causes a state to be exited.

To generate test cases, the test case generator runs through the usage model. Actions which influence the modeled system and transfer it into other states are modeled on *transitions*. In order for the test case generator to be able to make the selection of a *transition* in case of several outgoing *transitions* from a *state*, the conditions of the execution of the *transitions* must be unique. This uniqueness can be achieved either by means of different *guards* or *triggers* (events which affect the system). Therefore, it must be ensured that this uniqueness is achieved.

Triggers in MODICA are specified as texts that symbolize the event that leads to this *transition*. The *triggers* are used by the test case generator to generate the test sequences from the usage model. Actions performed on *transitions*, e.g. interactions with the test environment in the form of EXAM actions, changes of a *naming fragment*s, setting of a variant, etc. are specified in the action.

The basic rule is the following: IF A STATE CAN BE EXITED BY SEVERAL TRANSITIONS AND THESE ARE NOT EXCLUSIVE BASED ON THEIR GUARDS, TRIGGERS MUST BE SET FOR THESE TRANSITIONS!



If there is only one outgoing *transition* from a state, it is usually redundant to set a *trigger*.

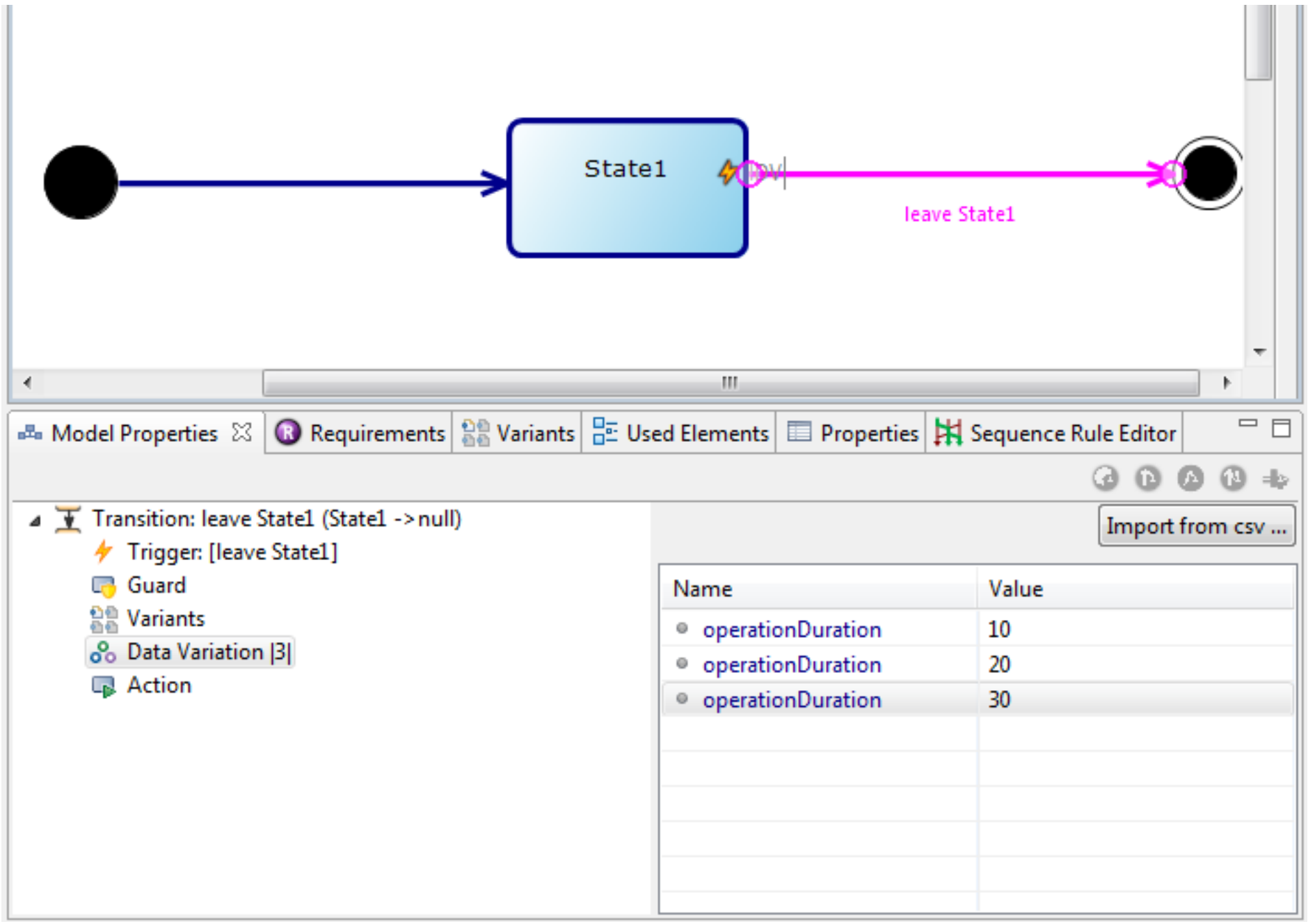
Data Variation

The data field at a *transition* makes it possible to generate data variations with which, for example, a variable of an EXAM function is filled when this *transition* is passed.

For example, suppose that a function of the test automation `set_operating_duration` needs a parameter of the type Integer which is used to set the operating time of a function (example: `set_operating_duration(int operationDuration)`)

In the model the operating time should be set to 10, 20 or 30 minutes.

Example:

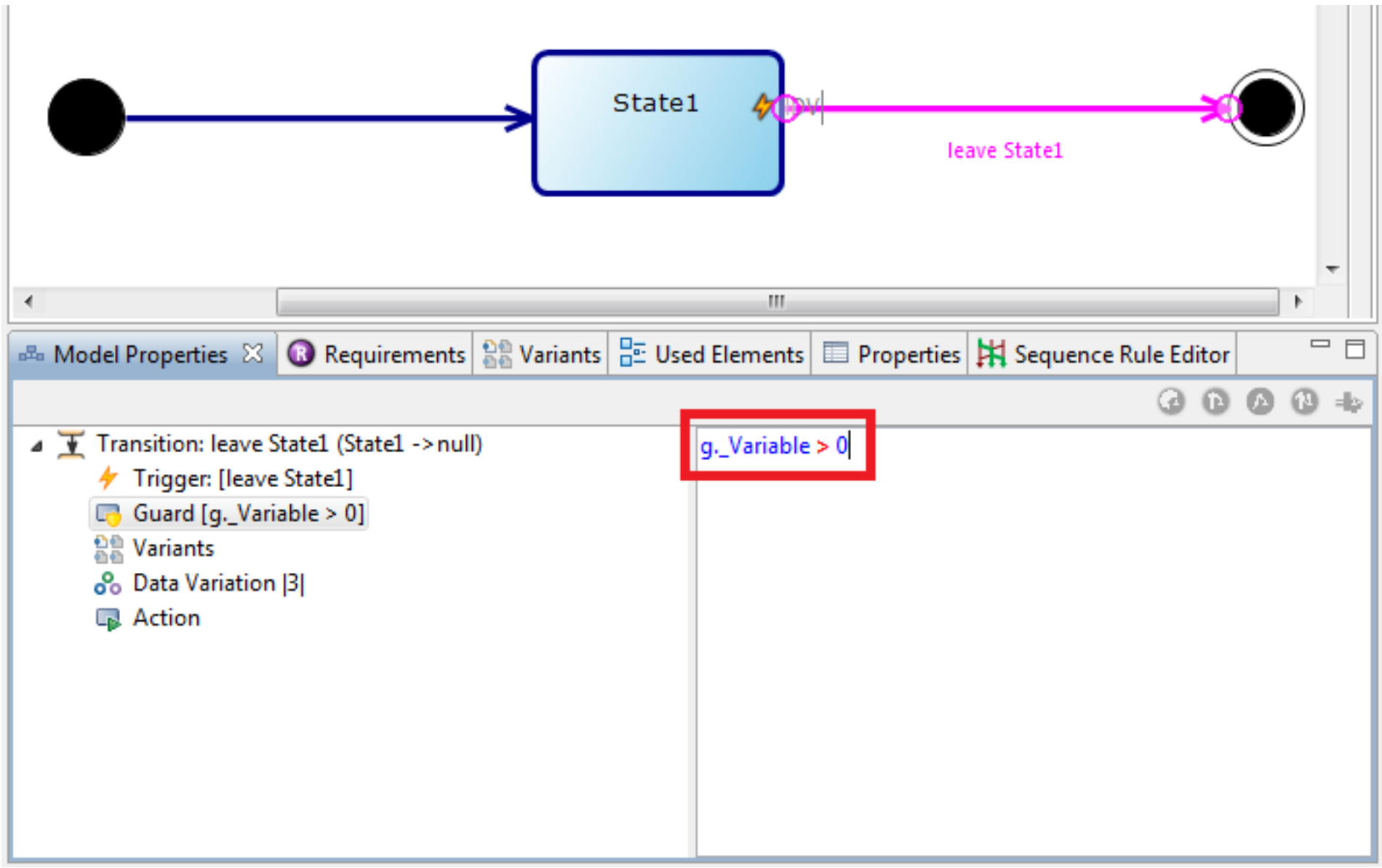


One should consider that a long chain of multiple data variations leads to a large number of possible test cases which, depending on the chosen generation strategy, influence not only the MODICA performance or the number of test cases generated. If a more explicit control is desired here, it's a good idea to explicitly model the variance by creating a separate *transition* (with *trigger*!) for each value.

The more detailed use and syntax of the fields of the *Model Properties View* is explained in the chapter Model Properties View.

Guard

The *guard* can be understood as the guardian of a *transition*.
In the *Guard*-field, conditions are set under which a *transition* may be traversed. For example, in the following, the *transition* 'leave State1' can only to be traversed if the variable 'Variable' is greater than 0:



The more detailed use and syntax of the fields of the *Model Properties View* is discussed in the section Model Properties View.

Else Guard

The *Else Guard* has a special and very useful role. This allows a *transition* to be taken if no other *transition* can be taken from the previous state. To do this, enter the text `else` as Guard. There must be no other characters in the *guard*. **The *Else Guard* can not be used together with *triggers*.** It is possible to use inbound or outbound transitions of a *JunctionState*.

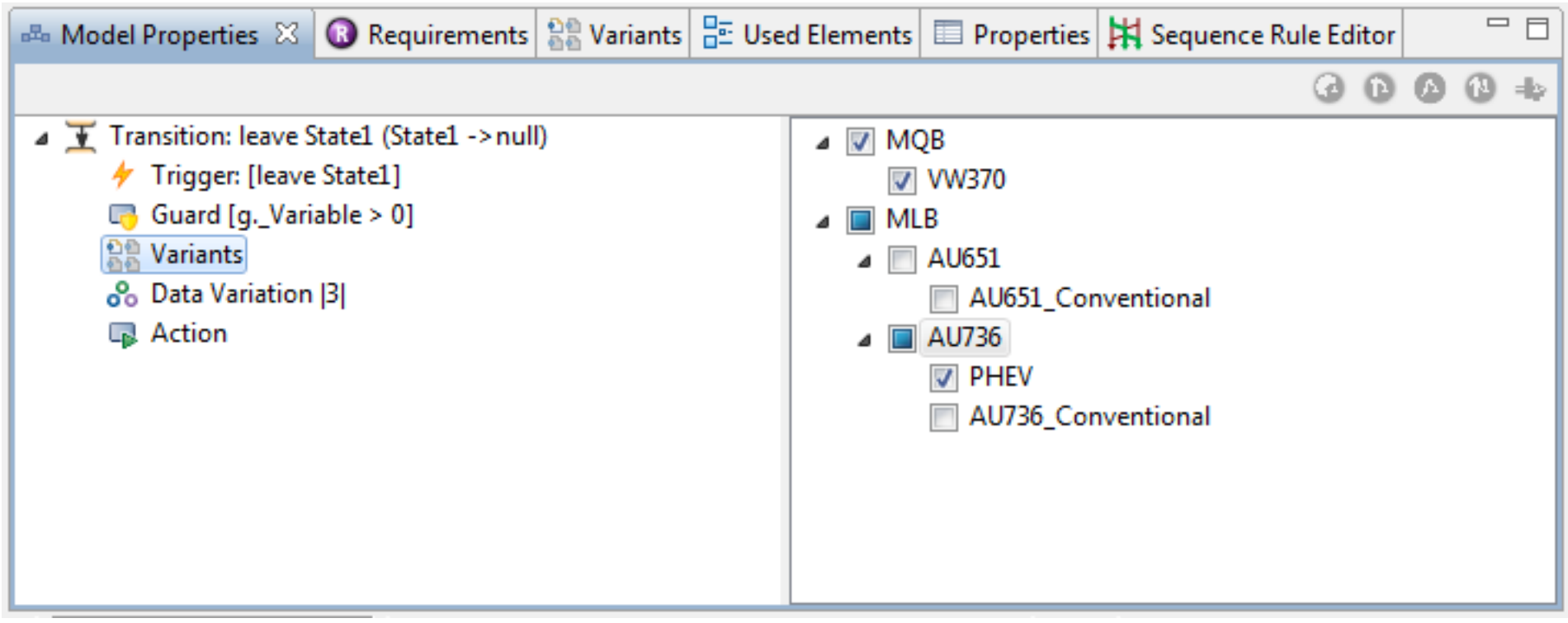
Variants

In the *Variants* area variant handling can be included in a MODICA model. Thereby, it is possible to generate test cases from a model for different variants of a system, which basically have the same behavior, but differ in some points and thus shouldn't create the same quantity / type of test cases.

Variations are explained in detail in the chapter Variant Management. It should be mentioned in advance that in the *Variants*-section all variants are displayed which are available in the current MODICA project.

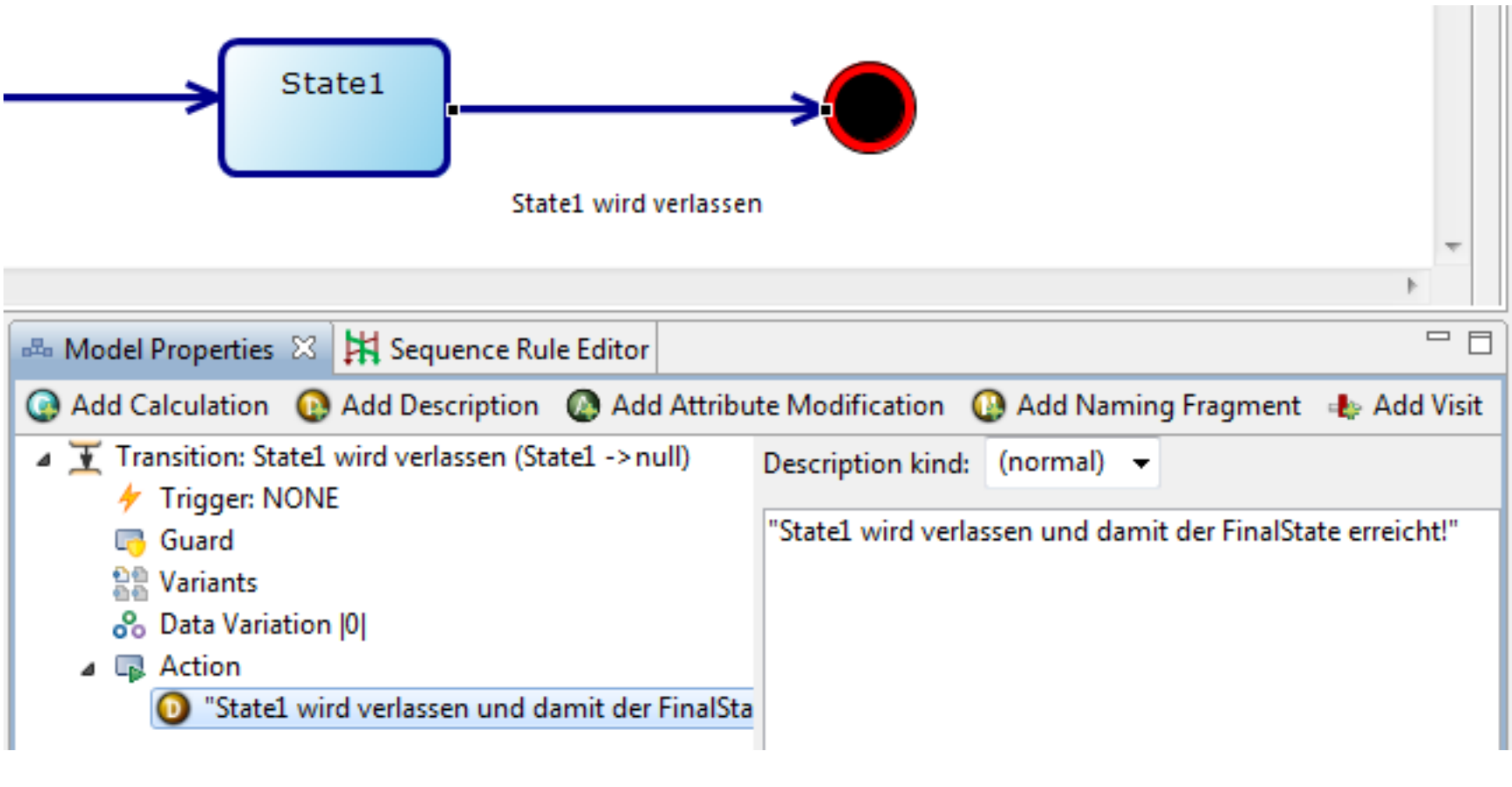
By activating / deactivating an entry in the variant tree, this transition is marked as part of this system variant or explicitly deactivated as a function / property not allowed for certain system variants.

Example:



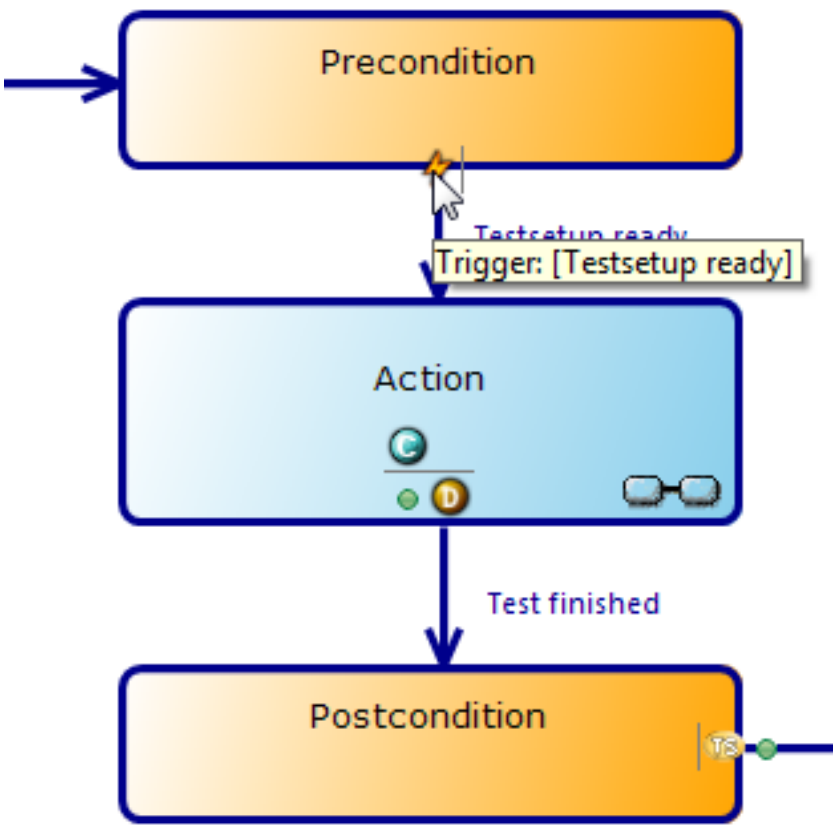
Action

The action of a transition describes what is to be done when this *transition* is traversed, e.g. EXAM function call, fulfillment of a requirement or creation of a description text for test cases or test case names, if the transition is passed, etc.

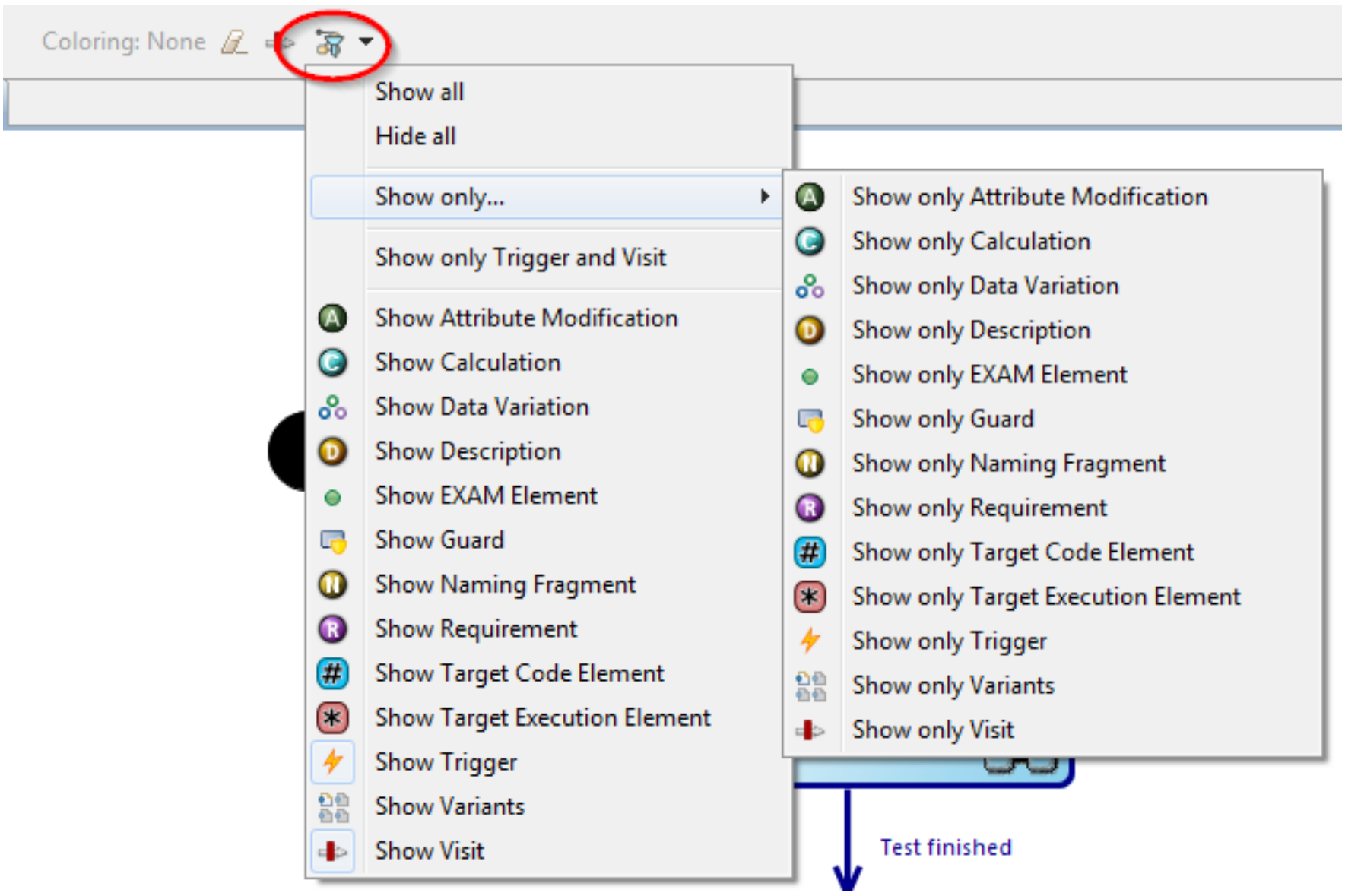


Decoration symbols

In the state diagram, decorating symbols can be used to show if objects are contained in *transitions* and *states*.

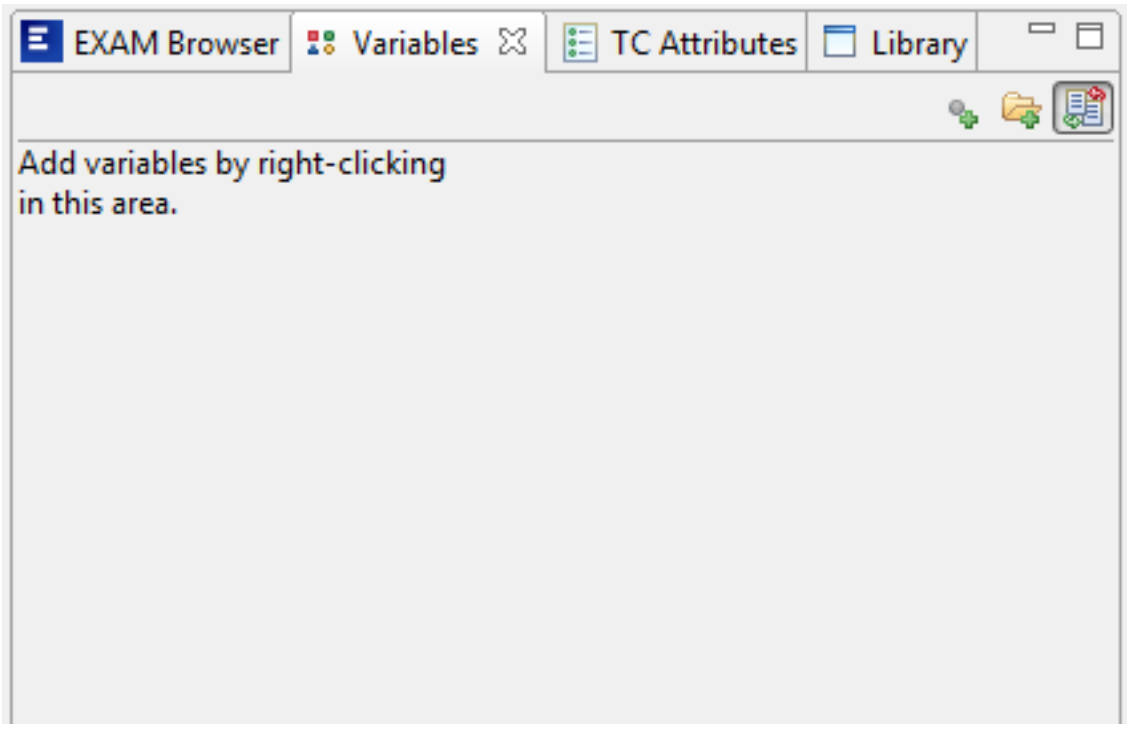


The configuration is done via the down arrow in the *toolbar*.



Variables (en)

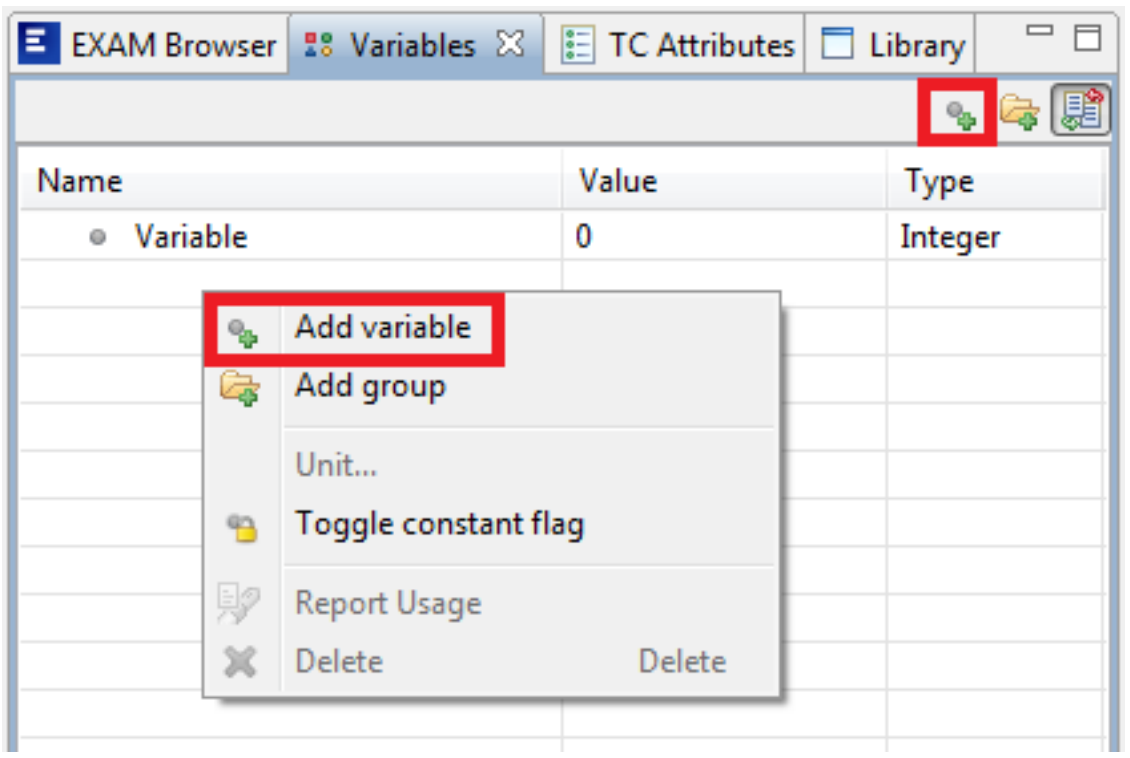
The *Variables View* provides the ability to create and manage variables that can be used in the *states* and *transitions*, whether as parameters to function calls or to control the test case generator.



Variables

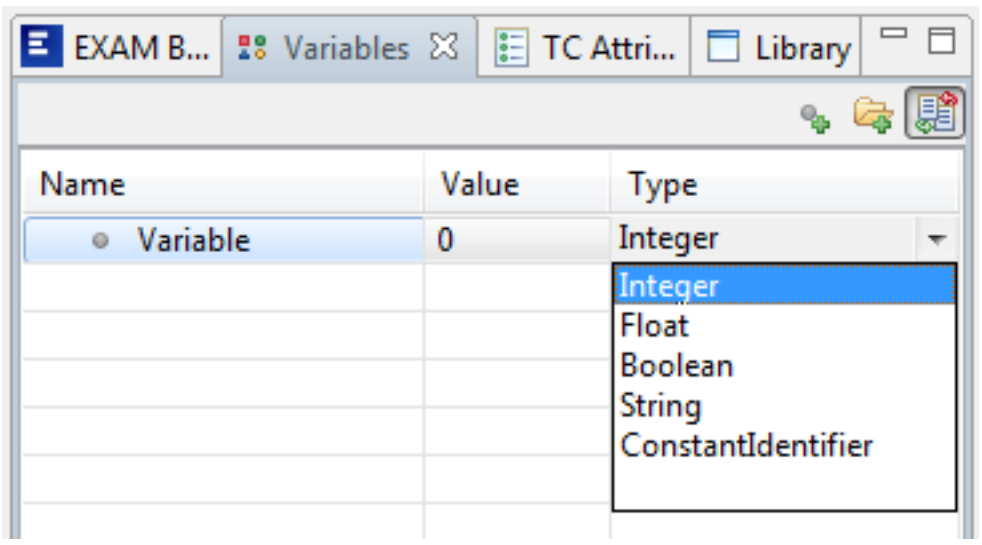
Create variables

A new variable is created by *right-clicking* in the *Variables View* and selecting *Add Variable* in the context menu. You can also use the  button in the toolbar of the view. A standard variable of the *type integer* with the default value of 0 and the name Variable is created.



Change variables

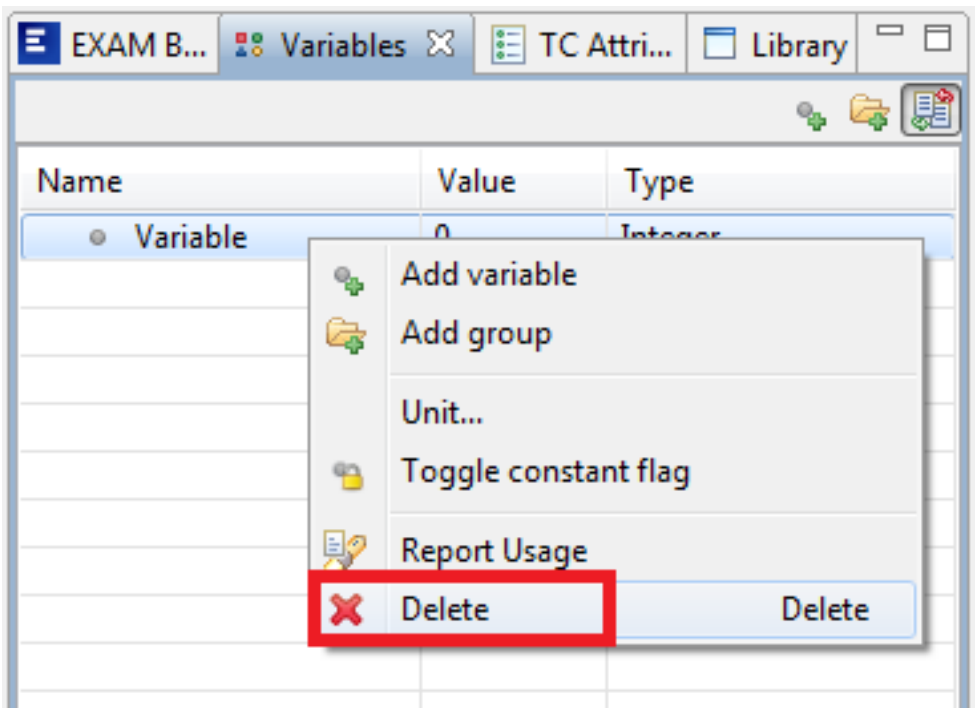
The name and the default value of the variable can be adjusted by *double-clicking* and changing the respective value. The type of the new variable is selected from the available list of variable types in MODICA, which opens by clicking on the current variable type.



Only if the refactoring symbol on the right side of the *Variables View* is activated, changes of variables are immediately adapted in the model and taken into account during the next test generation. Note: The refactoring button must be activated **BEFORE** changing the variable(s)!

Delete variables

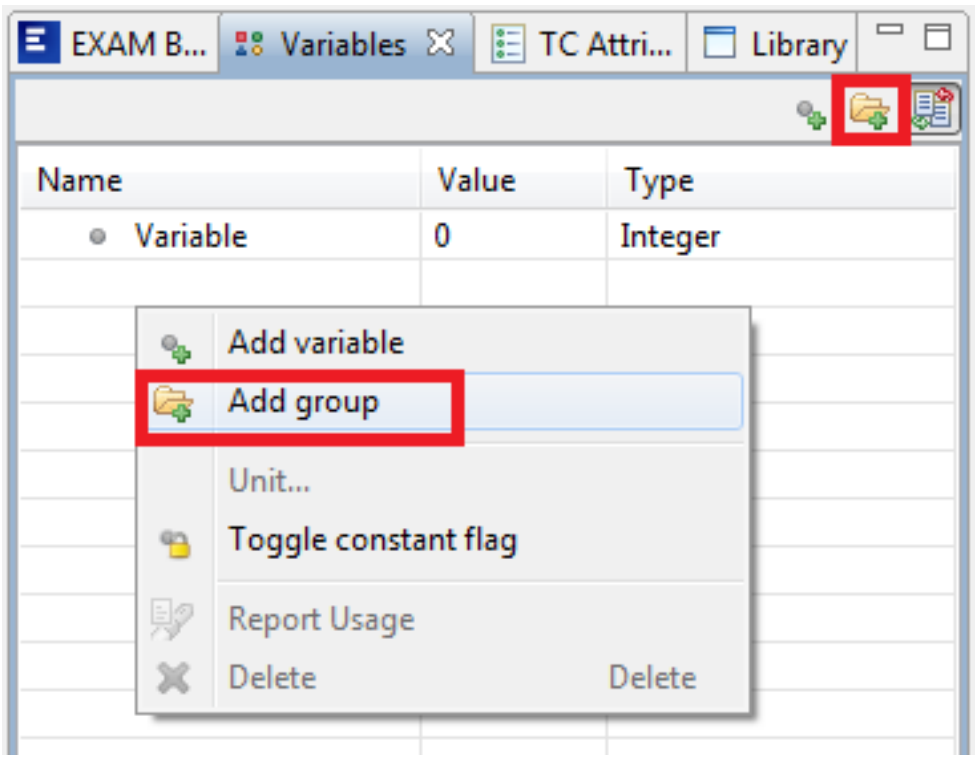
If a variable should be deleted, it must be marked in the *Variables View* and either deleted by pressing the `Del`-key or *right-clicking → Delete.*



Group variables

In MODICA, it is possible to group *variables* to get a better overview or to find related *variables* more quickly.

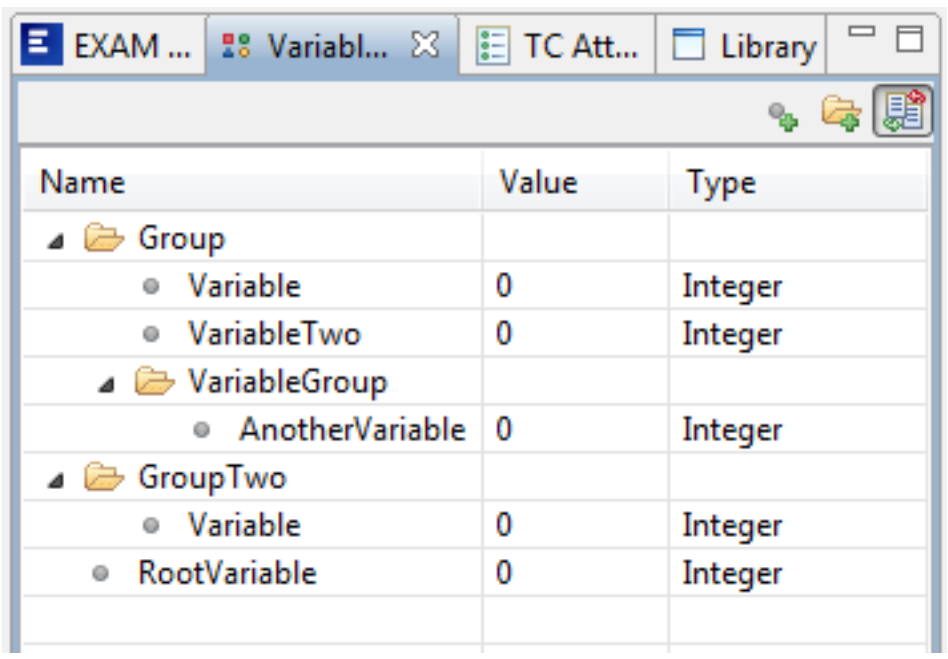
In order to be able to group variables, a variable group must be created first. To do this, *right-click* in the *Variables View* and select *Add Group* from the context menu. You can also use the button in the toolbar of the view:



This creates a group with a default name. The name of the group can be changed by *double-clicking*.

Variables can now be moved to the new group by drag & drop to sort them under this group name.

Example:



Types of variables

The following variable types are currently available in MODICA:

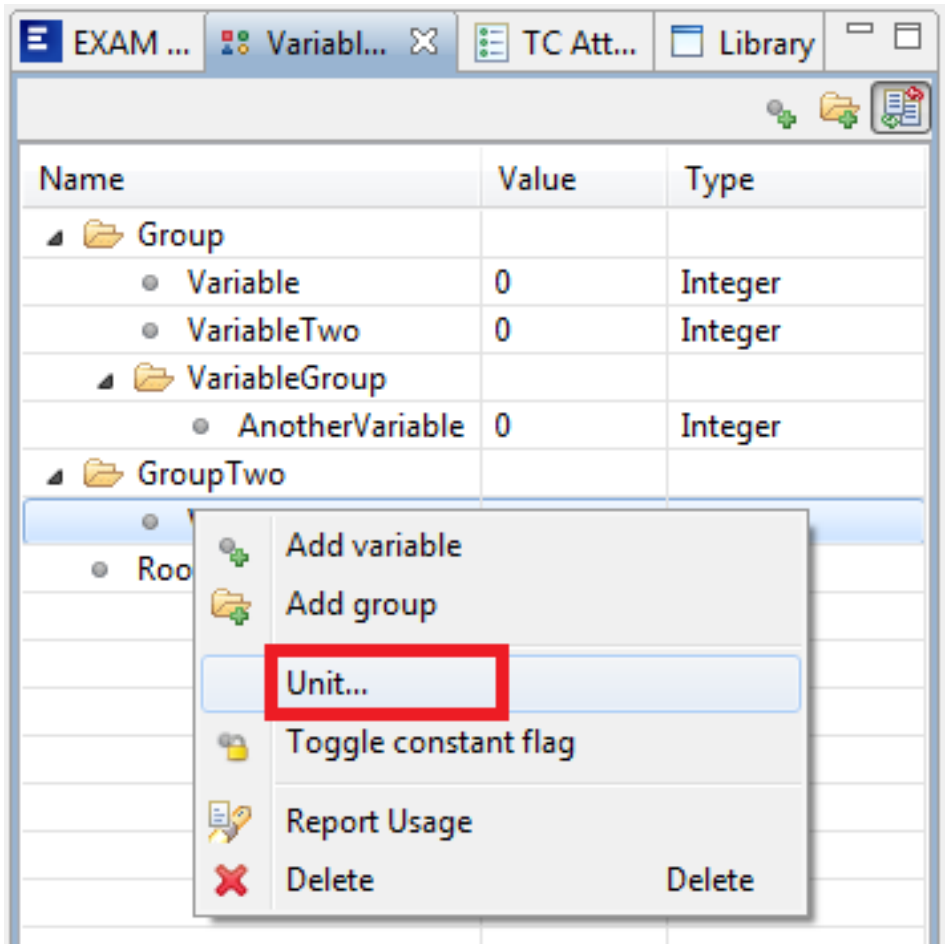
Variable type	Range of values
Integer	− 2147483648 bis + 2147483648 (− 0x8000000 bis + 0x80000000 if declared as hexadecimal number)
Float	Floating-point numbers (Must be a string that can be converted to a valid float value)
Boolean	*true*, *false*

String	Text of any length. If you want to represent an EXAM string, single quotes are to be used at the beginning and at the end. (E.g., 'I am a string in EXAM')
ConstantIdentifier	Special function to globally manage the names of local variables (→ out parameters in EXAM) for the entire MODICA project. The value can not be edited because the name of the variable 1: 1 is used in EXAM. The group (s) in which the variable can be located are not visible on the EXAM side.

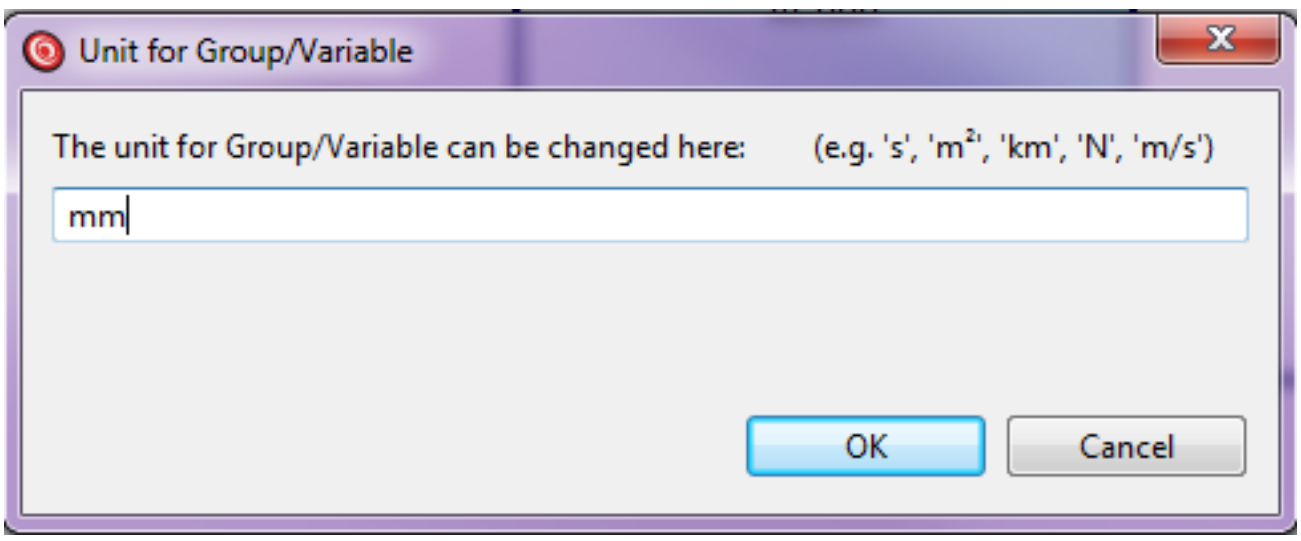
Units of variables

In order to specify the *variables*, it is possible to provide them with units. These are next to the value. However, the units are not evaluated.

To add a unit, right-click on the respective *variable* and select *Unit...* from the context menu.



The window for entering the desired unit opens.

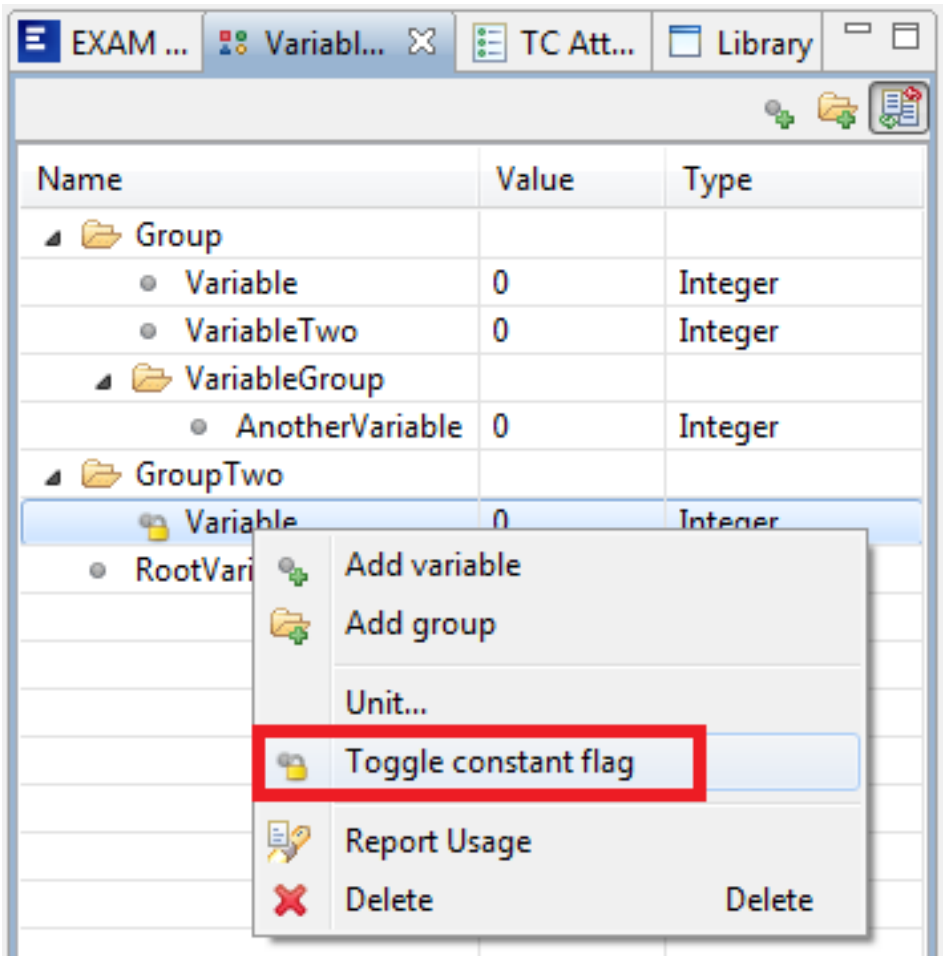


A string of up to 30 characters can be selected, which can not start or end with a space.

Constants

A *variable* can be converted to a *constant*. To do this, *right-click* on the *variable* and select *Toggle constant flag* in the context menu.

The *constant* is then provided with a small lock symbol.



Report Usage

The *Usage Report* provides the ability to display all locations in the project where the selected object is used. To do this, *right-click* the *variable* and choose *Report Usage* from the context menu. The results are displayed in the search window.

Access to variables

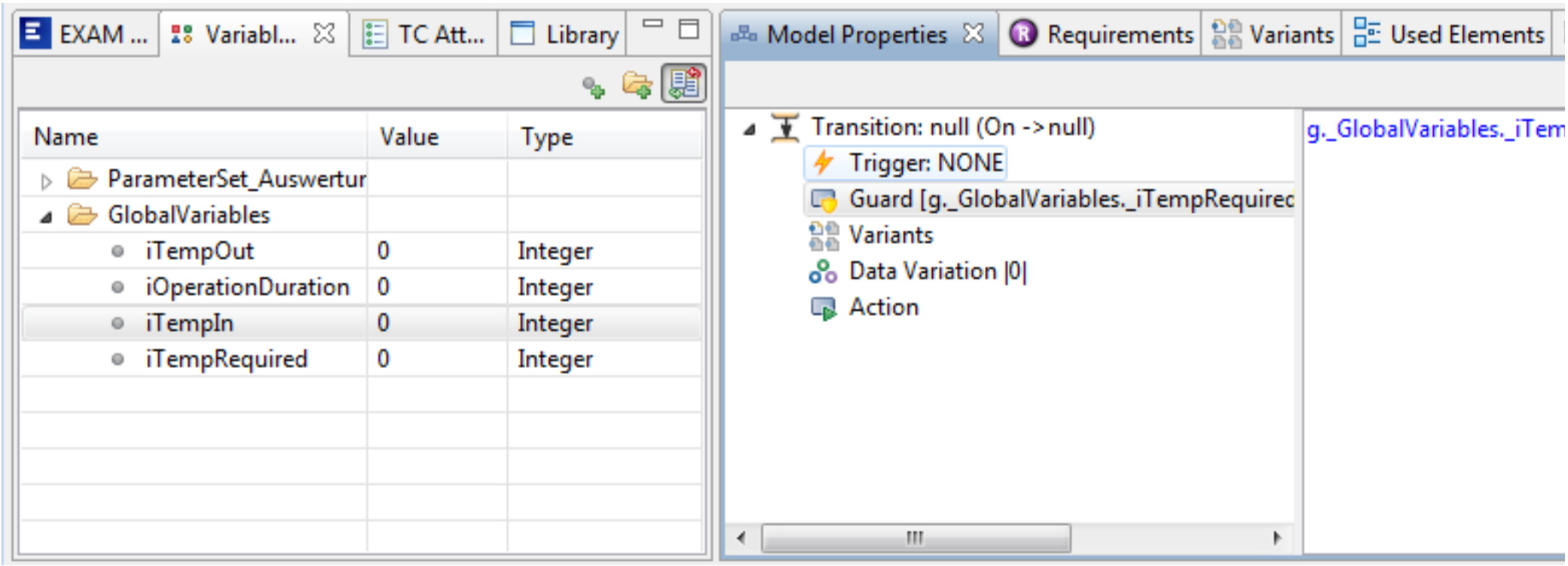
The following syntax must be used in MODICA wherever there is no graphical alternative to access a variable (for example, *Guard* to *Transitions*). To access variables in a MODICA project: **g._groupname._variableName**

Explanation:

- **g:** Internal class managed in the variable
- **_groupname:** Name of the group from the *Variables View* with a "_" in front of the name
- **_variableName**: Name of the variable from the *Variables View* with a "_" in front of the name

Example:

Variables and call in the *Guard of a transition*:



Alternatively, the *variable* can also be dragged directly from the *Variable View* to the required location by drag & drop.

Requirements View (en)

Requirements that are submitted to the SUT are a major input for the modeling. The *requirements* describe the expected behavior to be checked in the tests.

Most of the following examples are shown here using DOORS as requirements source. The steps are very similar using other supported requirements sources (such as PTC Integrity, or using an Excel-Export)

Connection to DOORS

Usually, all *requirements* are stored in a database called DOORS. The steps for producing a DOORS connection are described in the tutorial under Step 2 - Connection to Requirements (DOORS).

Assignment of requirements

After including one or more DOORS modules into a project, the *requirements* are visible in the *Requirements View* and can be used from here. The *requirements* are transferred by assigning them to *transitions* or *states* using Drag & Drop from the *Requirements View* to the *Model Properties View*. By assigning the *requirements* in the model, it is possible to create test cases using the model elements to which the *requirements* are assigned.

The right column shows the concrete element in the model to which this *requirement* is assigned. If you click on it, the item is automatically selected and displayed in the editor.

ID	Object Header	
10	**1 Single traffic sign recognition**	
1	When the system starts, the display may not show a traffic sign (empty display)	InitialState -> Display e When the System starts display may not show a sign (empty display)
2	If no traffic sign is shown and a new traffic sign is recognized the new traffic sign must be shown as highlighted	New Roadsign 1
3	When showing as highlighted and after five seconds the traffic sign must be shown as normal (reliable, no longer highlighted)	Timeout
4	When showing as reliable and after passing the configured distance (Aging Distance 1) the traffic sign must be shown as outdated	AgeDist1
5	When outdated and after passing the configured distance (Aging Distance 2) the traffic sign must disappear from the display (empty display)	AgeDist2
11	**2 Multi traffic sign recognition**	
6	If, while showing a traffic sign highlighted, a new traffic sign is recognized this must be shown as highlighted.	New Roadsign 2
7	If, while showing a traffic sign as reliable, a new traffic sign is recognized this must be shown as highlighted.	New Roadsign 3
8	If, while showing a traffic sign as outdated, a new traffic sign is recognized this must be shown as highlighted.	New Roadsign 4
9	If, while showing a traffic sign as reliable, the same traffic sign is recognized this must be shown as reliable again.	Same Roadsign
		InitialState -> Display e deleted requirement

Requirements View - Overview

Meaning of colors

- *Requirement* is used in the model
- *Requirement* is **not** used in the model
- *Requirement* has been changed in the DOORS database
  - In the right column you can read the original version of the requirements, on the left is the current version. In the screenshot, for example, the large `'S'` at `'System'` was replaced by a small `'s'`.
- *Requirement* no longer exists in the DOORS database
  - In the right column you can read the text of the original *requirements*, in the screenshot `deleted requirement`

Update

In order to always work with the most recent *requirements* catalog, the refresh button should always be pressed in the toolbar of the *Requirements View* in MODICA to be up to date.

Filter

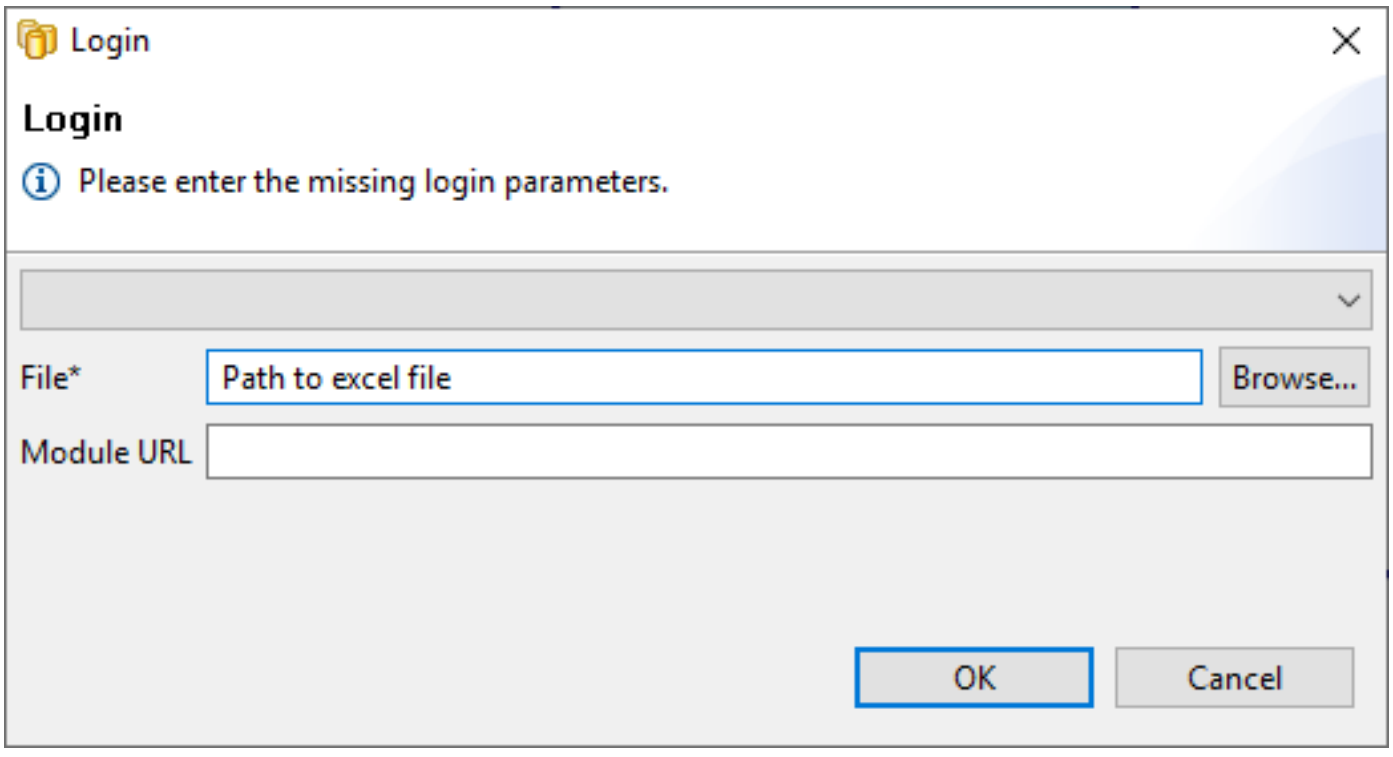
By means of filters in the upper toolbar of the *Requirements View*, only certain *requirements* can be displayed:

- (1) *Show changed requirements only*: Only requirements whose properties are different (and / or not used in the model) compared to the DOORS database are displayed
- (2) *Show linked requirements only:* Only the requirements that are used in the model are displayed
- (3) *Show non linked requirements only:* Only requirements that are not used in the model are displayed

Requirements View - Filter

Excel Integration

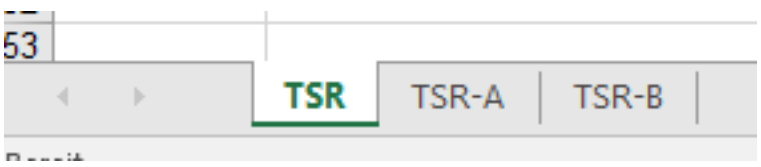
Unlike DOORS, EXCEL only needs the path to a file for its configuration.



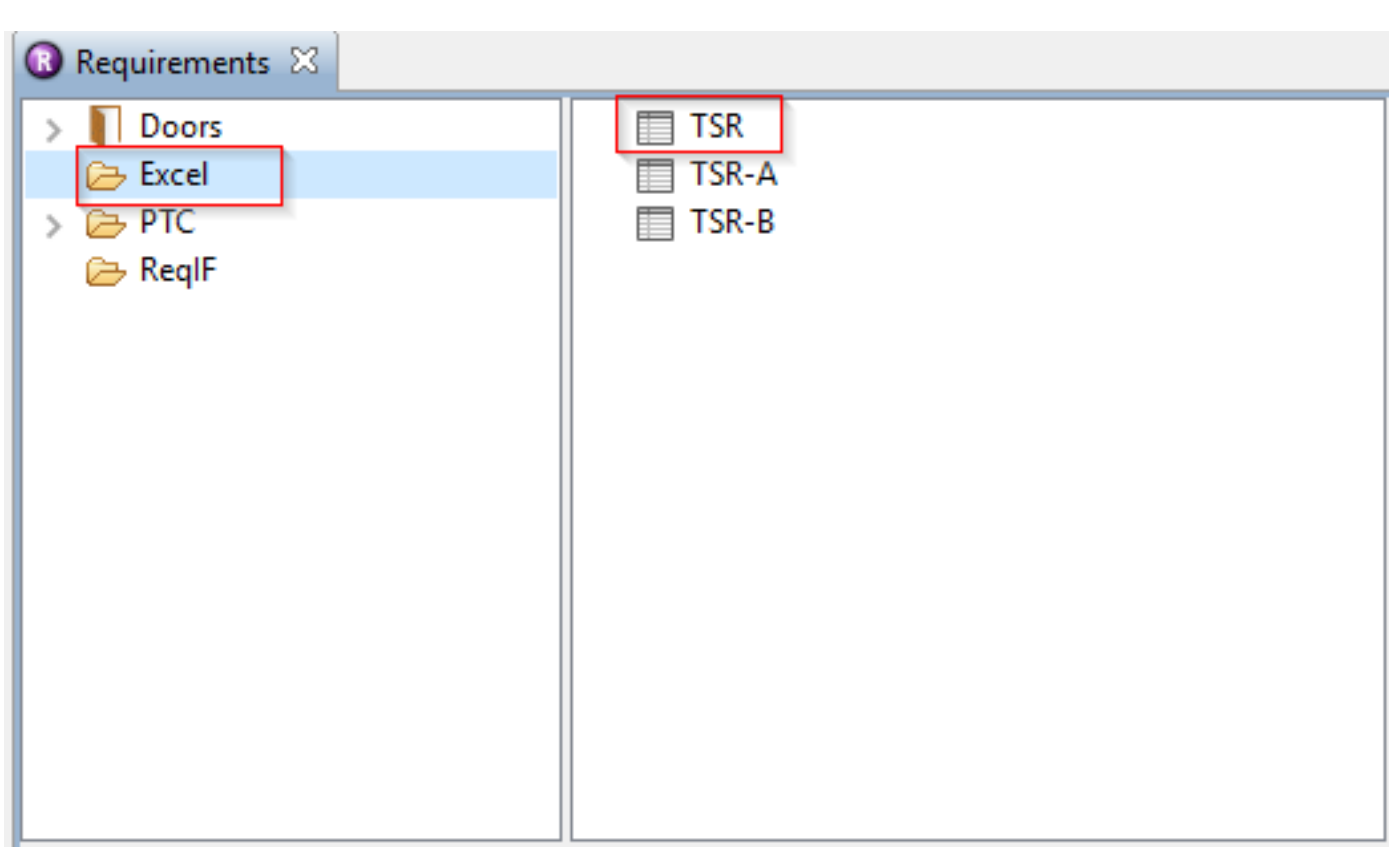
The field "Module URL" is optional and serves for later exporting of the requirements linked in the test case in the case of excel table was exported from DOORS.
The URL has the following format: "doors://DoorsServer:36677/?version=2&prodID=0&urn=urn:telelogic::1-1234abcd1234abcd-M-00000060".
This URL can be found in the DOORS database in the properties of the module to be exported. The module can be copied in the "General" tab.

As second step, the worksheet must be selected.

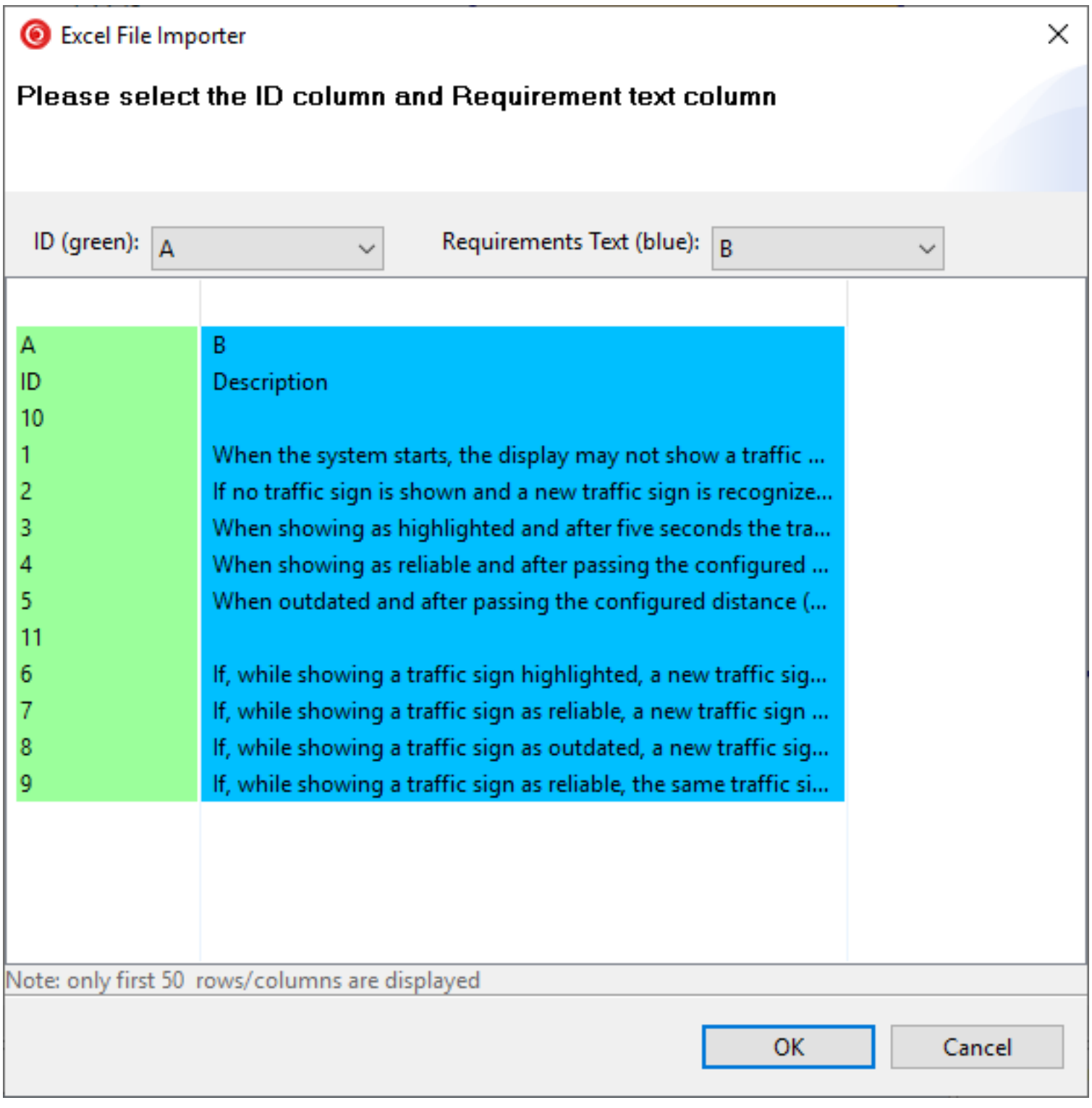
In Excel:



In MODICA:

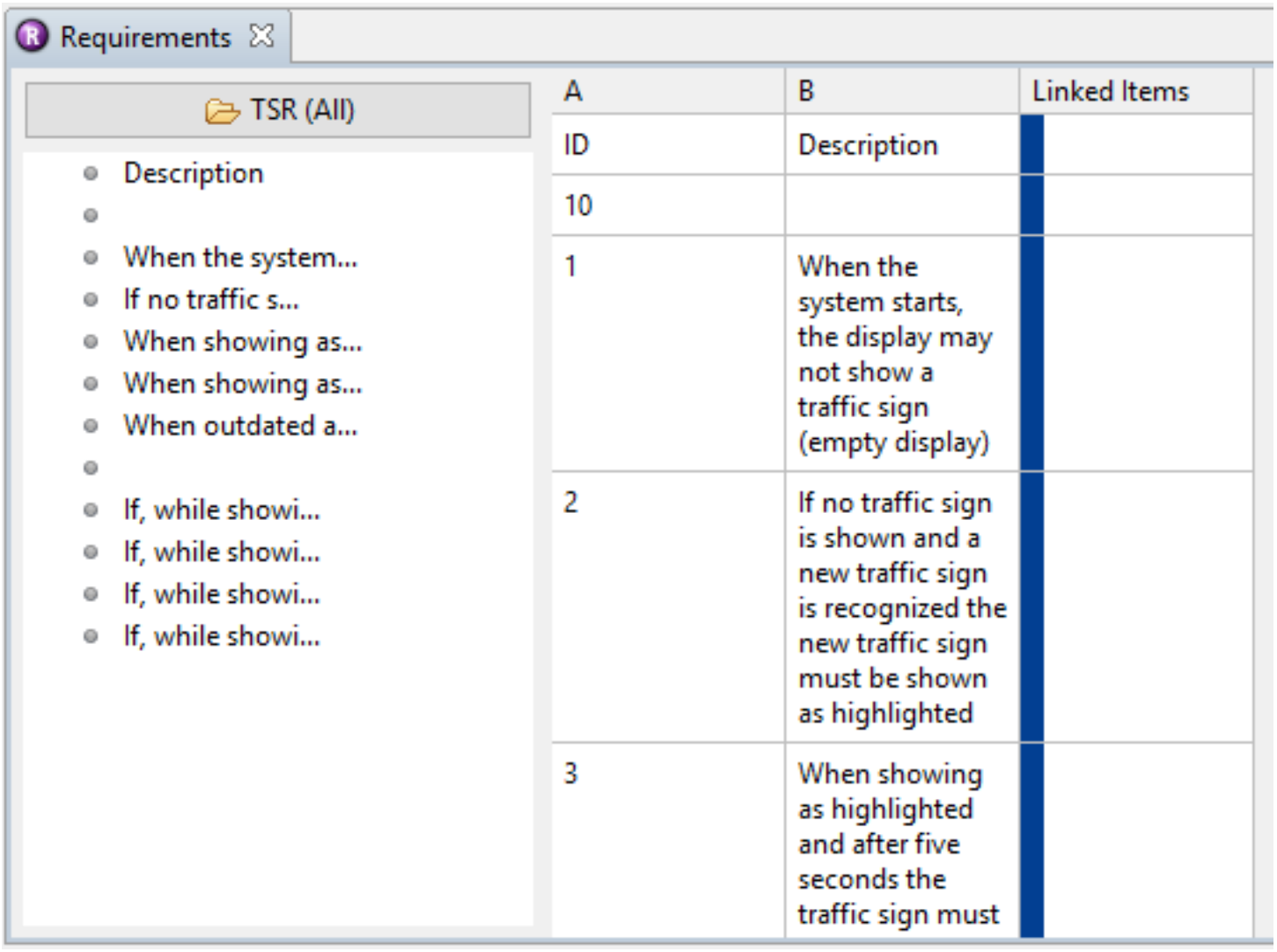


After a double click on the worksheet, a configuration dialog will be displayed where the column of the ID and the description of the requirements should be selected using selection fields.



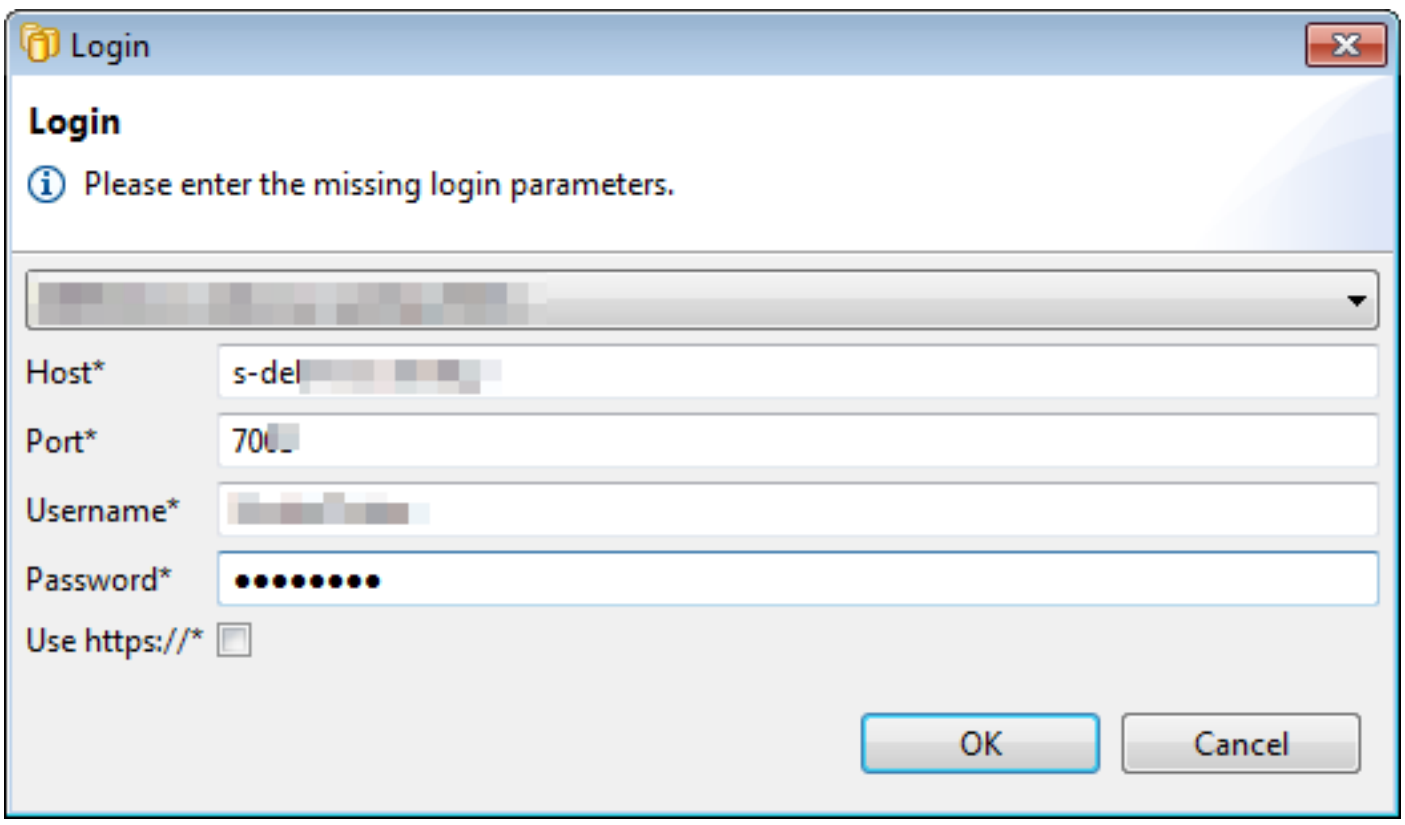
In the above example, Column "A" is selected as source for the IDs and "B" is used for the descriptions.

After confirming the dialog the worksheet is loaded in selected configuration in MODICA.

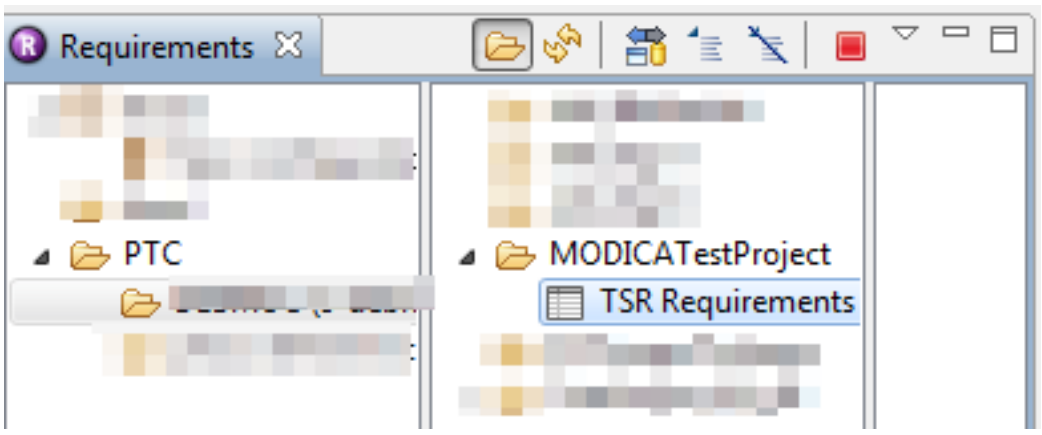


PTC Integrity Integration

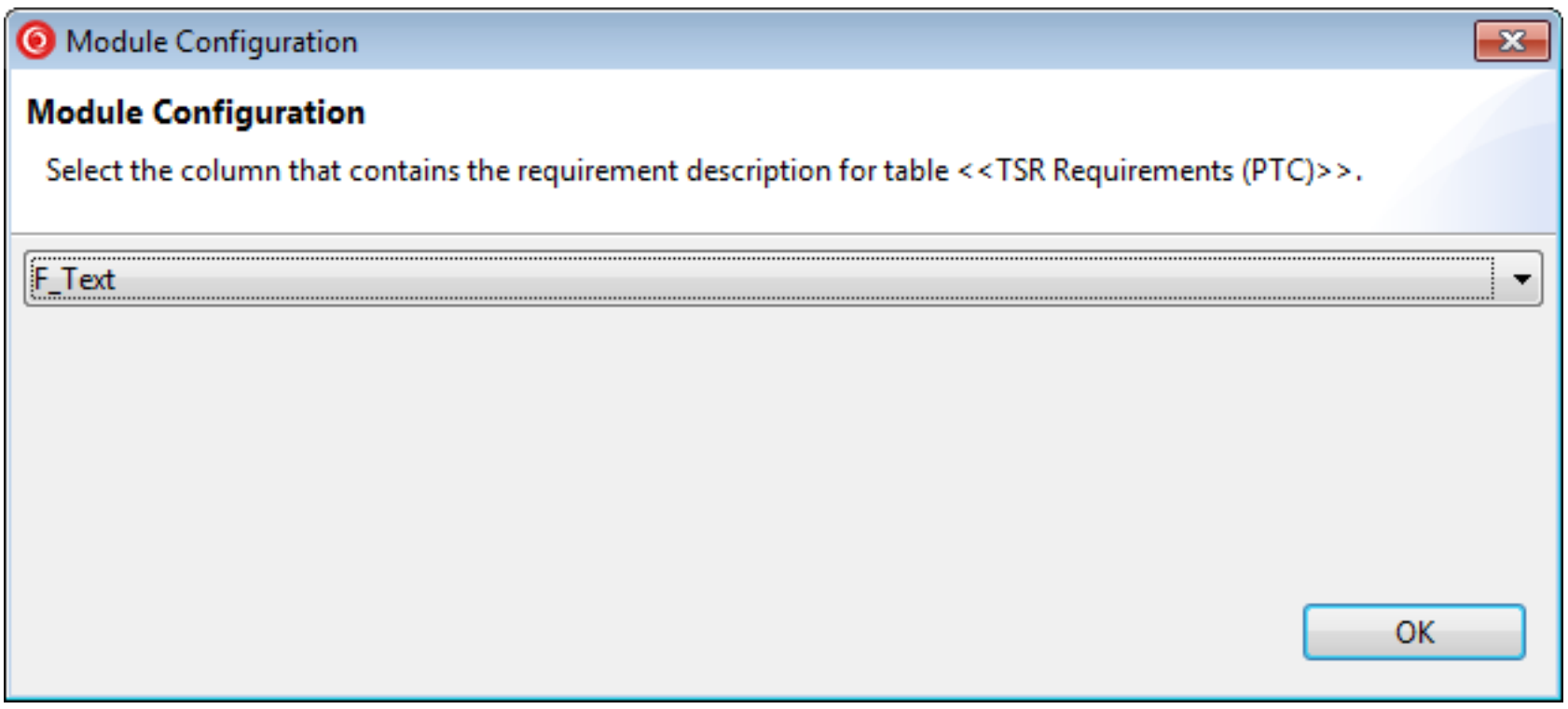
Similarly to DOORS, the integration with PTC-Integrity requires valid connection settings for a PTC-Server. Possibly, additional permissions are required to be able to connect (server-configuration). On the technical level, MODICA uses the mksapi.jar available from PTC.



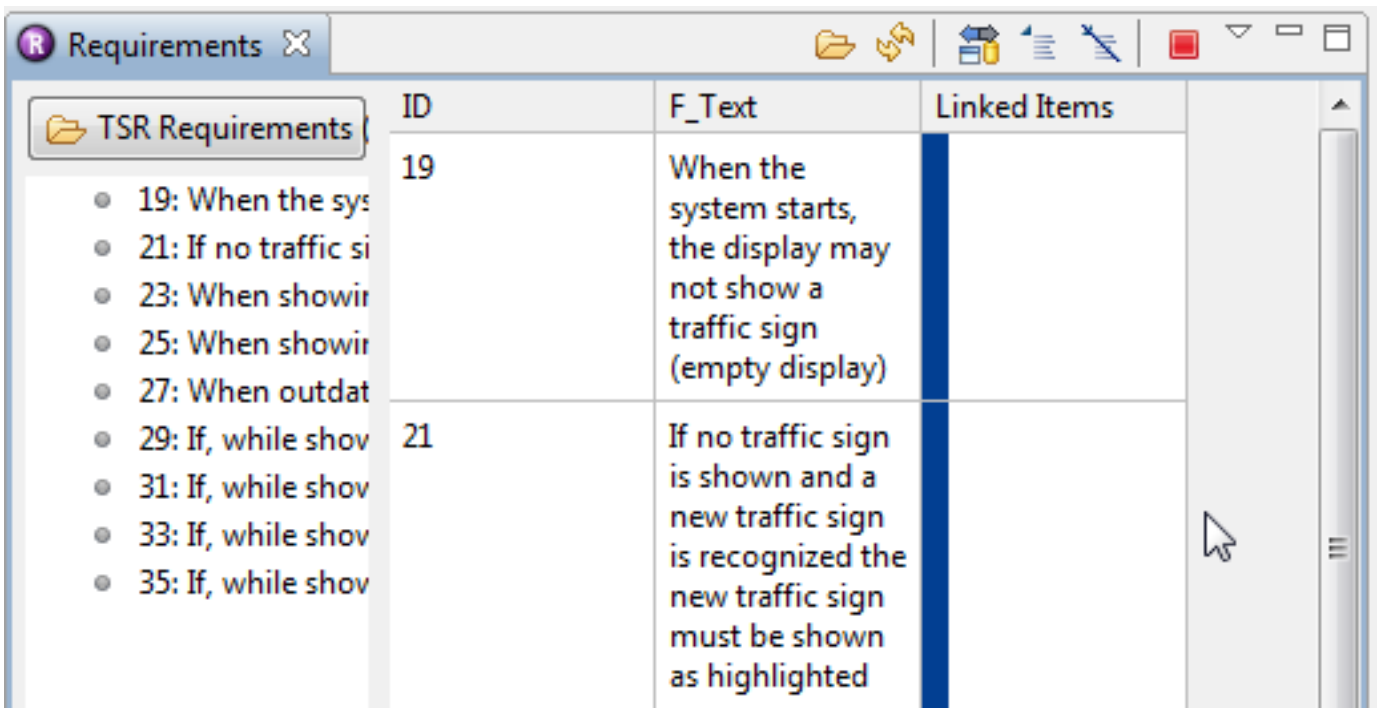
After entering valid connection parameters, the list of projects and documents is loaded from the server.



By double clicking on a (requirements-) document it will be loaded. As part of the loading, the user must select a field defined in PTC-Integrity that should be used as description in MODICA.

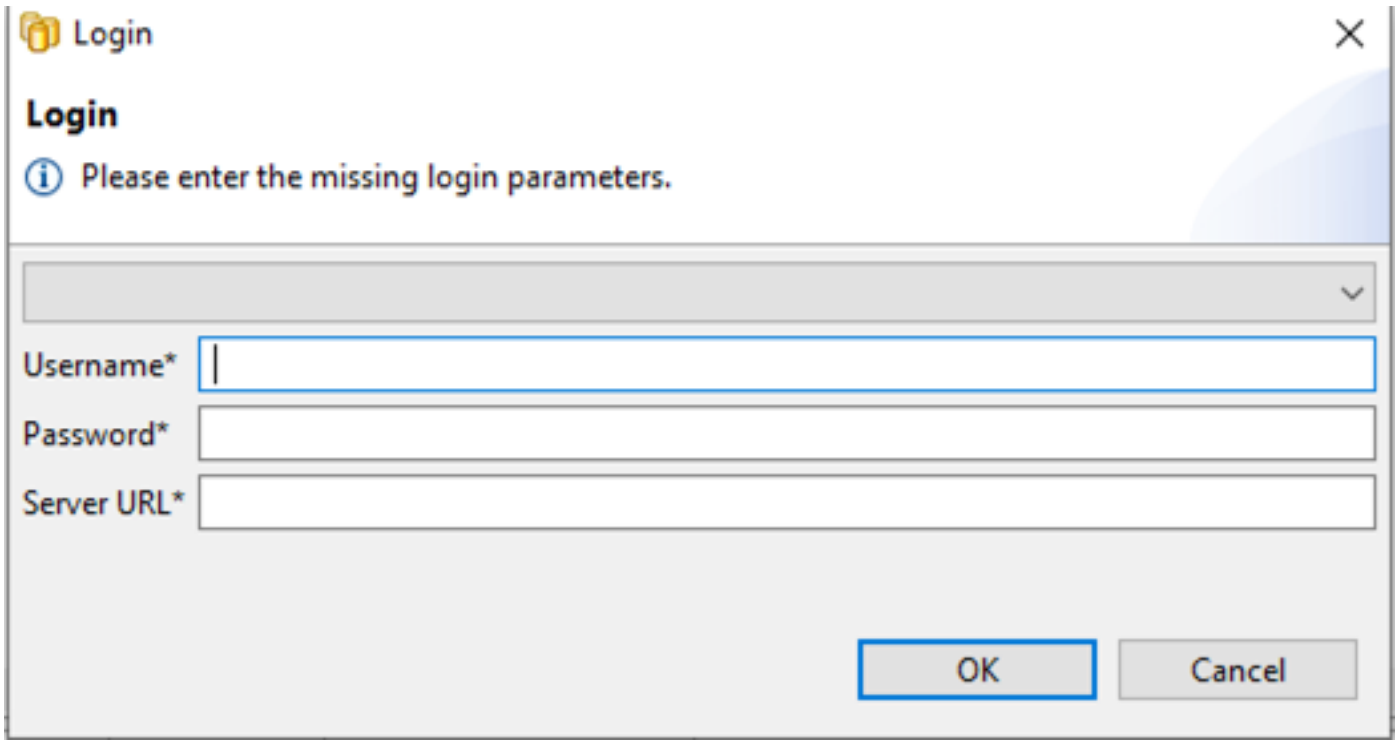


After confirmation, the Requirements are loaded from the server and can be attached to diagram elements using drag&drop to the *Model Properties View*.

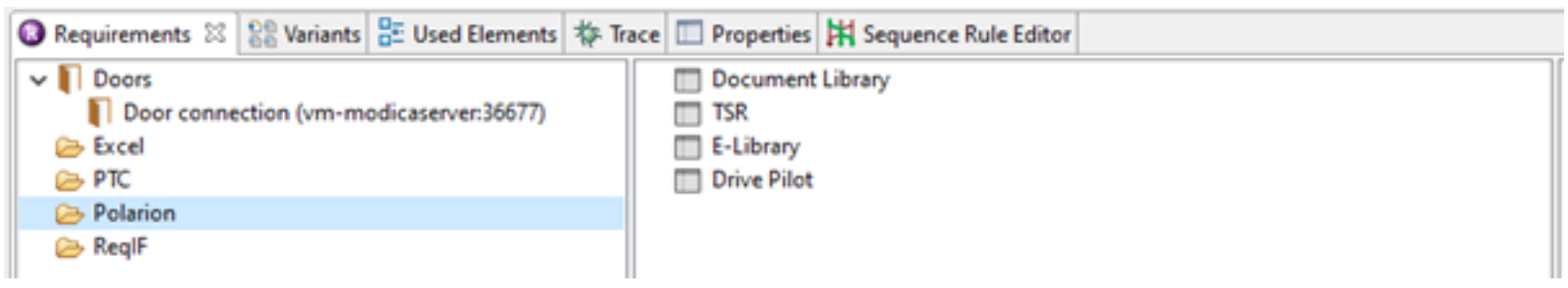


Polarion

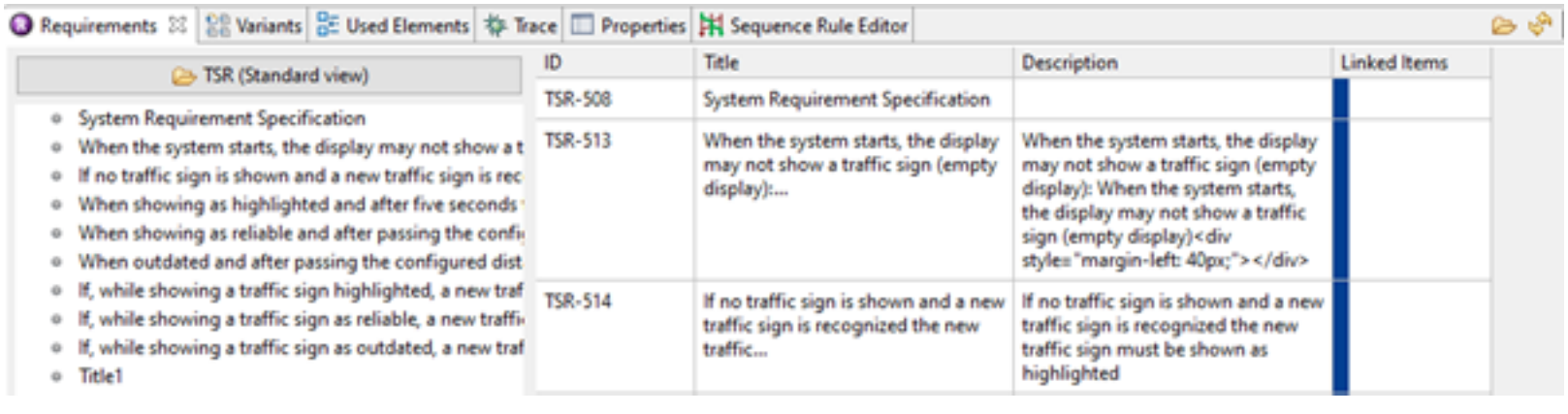
Similar to DOORS, the integration with Polarion requires valid connection settings for a Polarion Web Server. Additionally, the Polarion service has to be installed on the system to use the Polarion Web Services. A user cannot login using the browser-based login credentials.



The username, password and the server URL on which web services are accessed are required to login. After entering valid connection parameters, the list of projects available to the user is loaded from the server.



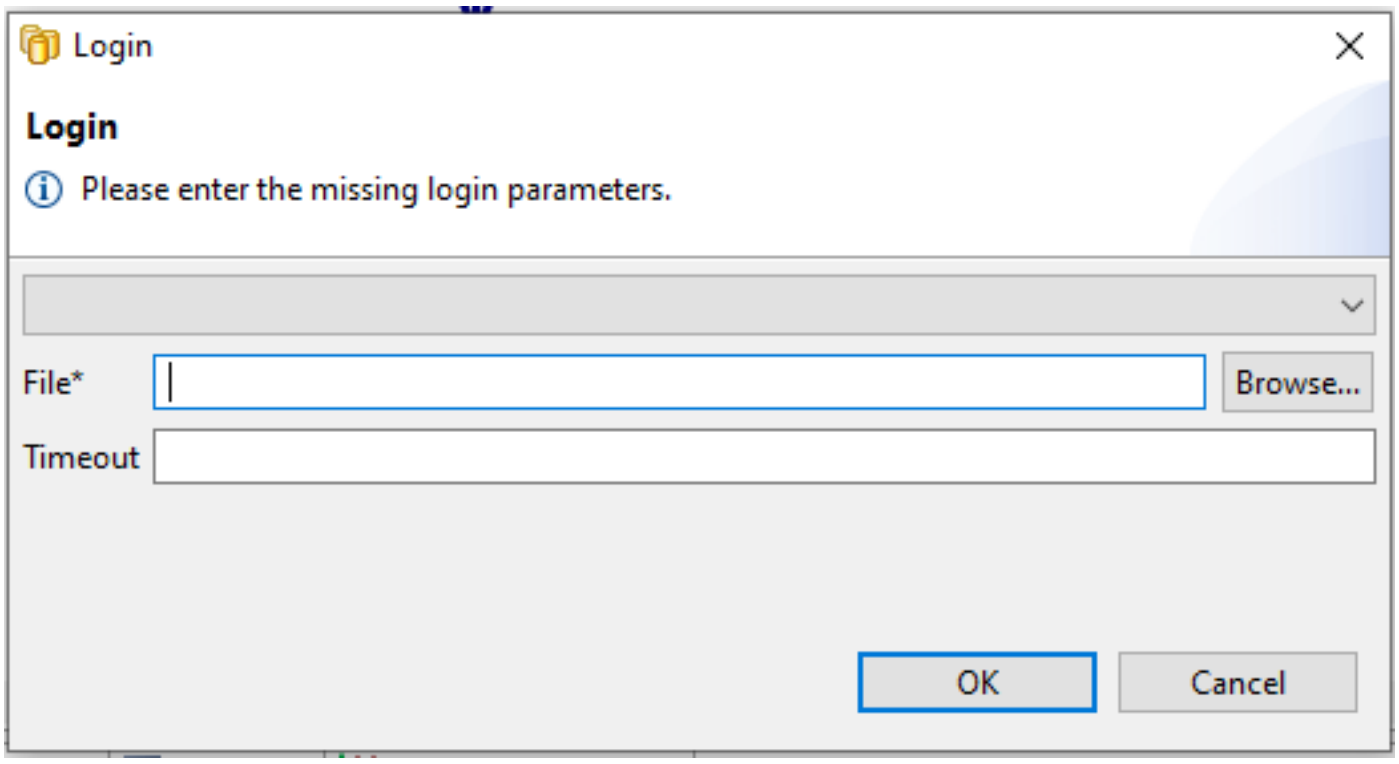
By double clicking on a project, the work items associated with that project will be displayed.



These work items can be attached to diagram elements using drag&drop to the Model Properties View.

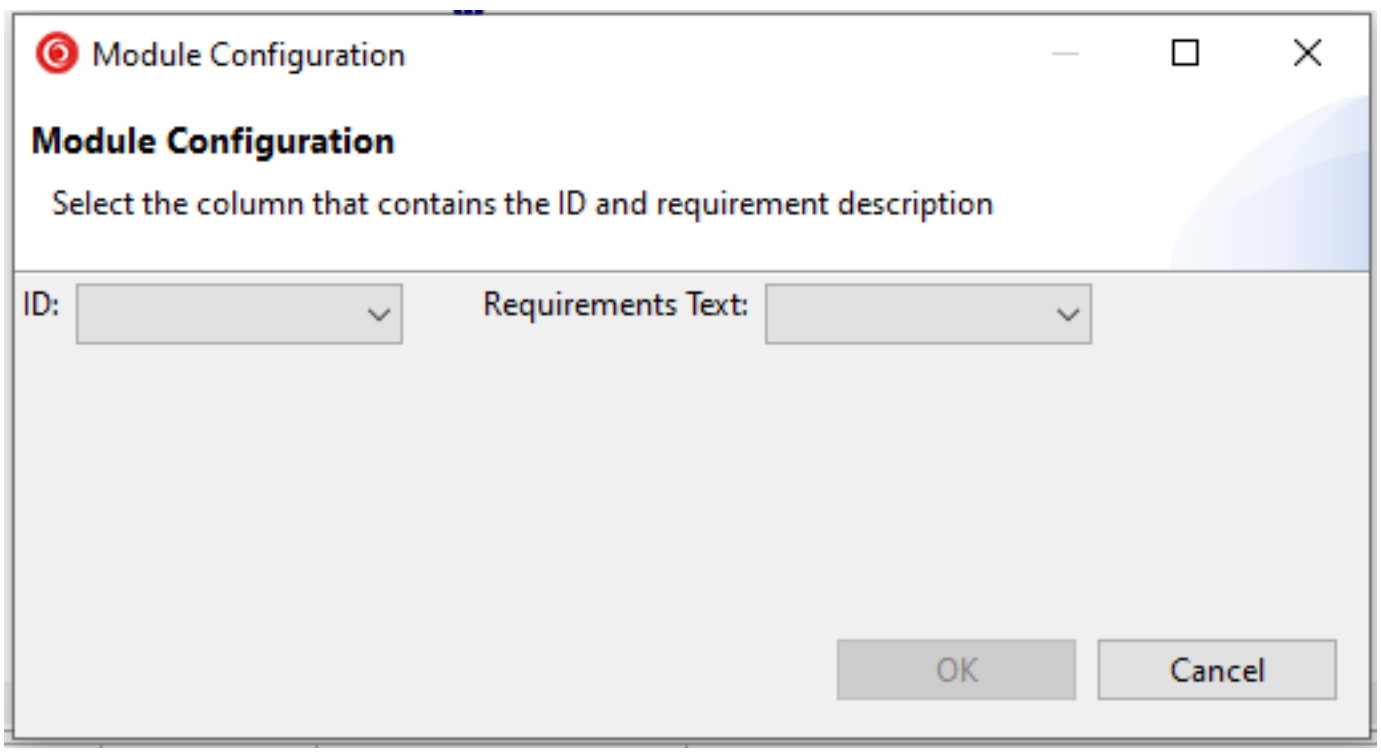
ReqIF

ReqIF only requires the path to a ReqIF or RIF file for its configuration.

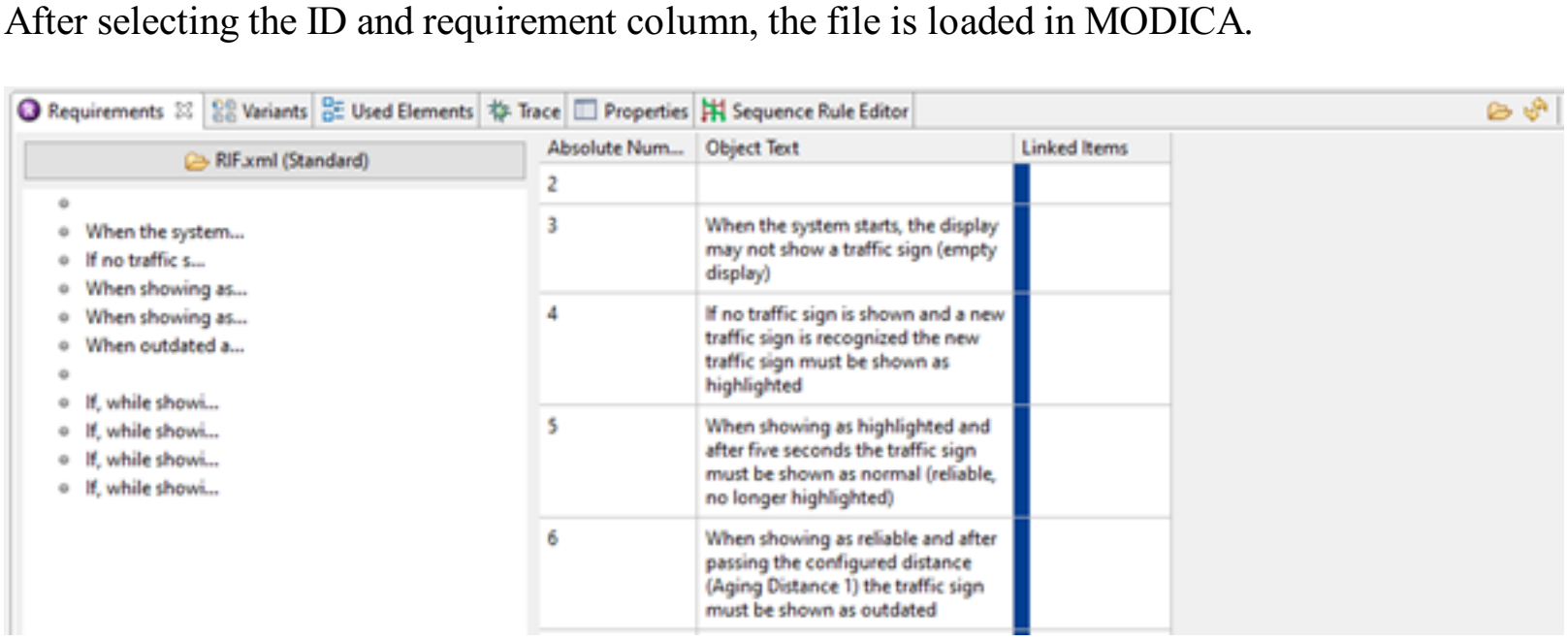


The field “Timeout” is optional.

After a double click on the worksheet, a configuration dialog will be displayed where the column of the ID and the description of the requirements should be selected using selection fields.



After selecting the ID and requirement column, the file is loaded in MODICA.



Model Properties View (en)

A central function of modeling in MODICA is the *Model Properties View*. This view is used to configure *transitions* and *states* as well as the assignment of values, functions, requirements, and description texts.

The available elements of the *Model Properties View* depend on which model object is selected.

States have an *entry action* and an *exit action*.

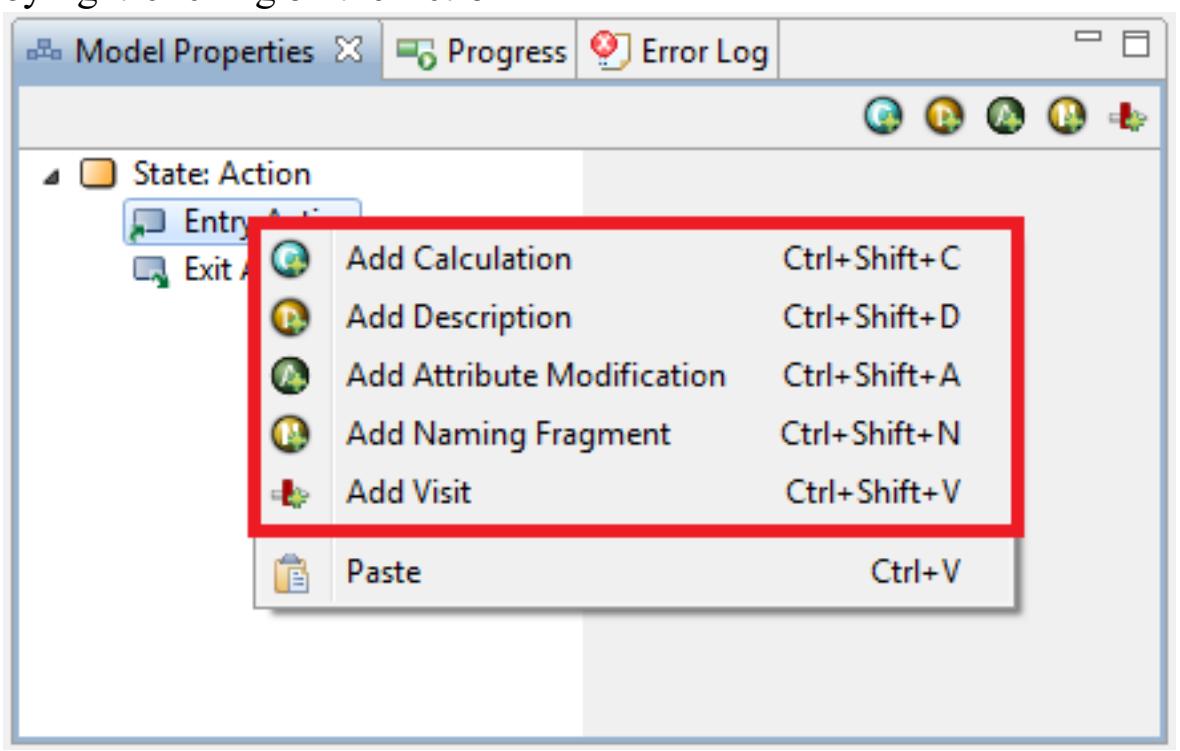
Transitions can contain *triggers, guards, variants, data variations*, and also *actions*.

Actions (*Entry* and *Exit Action*s as well as *Transitions-Actions*) consist of the following elements: *Requirements, Calculations, Descripitions, Naming Fragments, Visits* and *Attribute Modifications*.

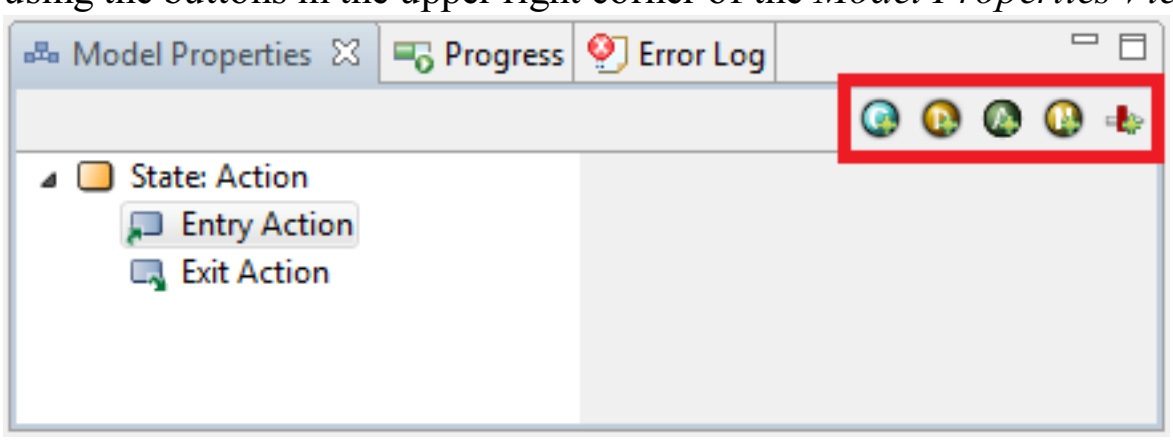
Add an Action

To add an *action* to a *state* or a *transition*, there are different options depending on the type of *action*.

- *Calculations, Descriptions, Attribute Modifications, Naming Fragments* and *Visits* belong to the same group an can be added
  - by right-clicking on the Action



  - using the buttons in the upper right corner of the *Model Properties View*



  - using a keyboard shortcut: *Ctrl + Shift + Initial letter of the action* (e.g. Calculation: *Ctrl + Shift + C*)

- *Requirements* and *(EXAM) Calls* are dragged and dropped from the respective View (*Requirements View* or *EXAM Browser*) to the *action*.

Assignment of Actions, Triggers, Guards, Variants and Data Variations

The various types of actions and the assignment of *triggers*, *guards*, *variants* and *data variations* are explained in more detail below.

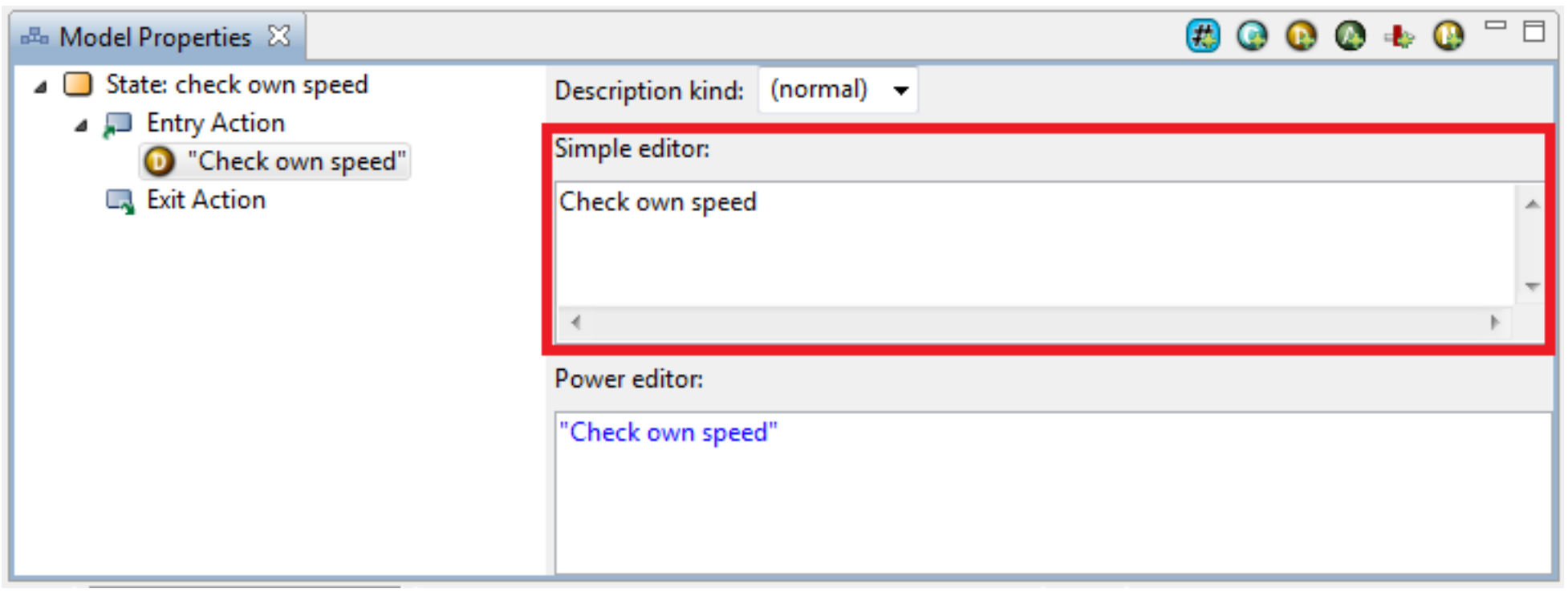
Descriptions

In order to generate a description text for a test case, MODICA offers the possibility to store individual parts of the entire test case description at different points in a model. In the final test case, this is combined into an entire description text. To do this, the *entry* or *exit action* (depending on where you want to add a description text) must be selected and the description added as described above. In the selected area, a new entry *Description* is displayed, which is indicated by the icon 

The corresponding text field can be used to define a text that describes the current events in the test case or in the system. There are two text editors available: The *Simple editor* and the *Power editor*.

Simple Editor

The *Simple editor* can be used if only pure text is to be added to a *description* - ie. without the use of *variables* or calculations. Simply enter the desired text directly into the editor - without quotes.



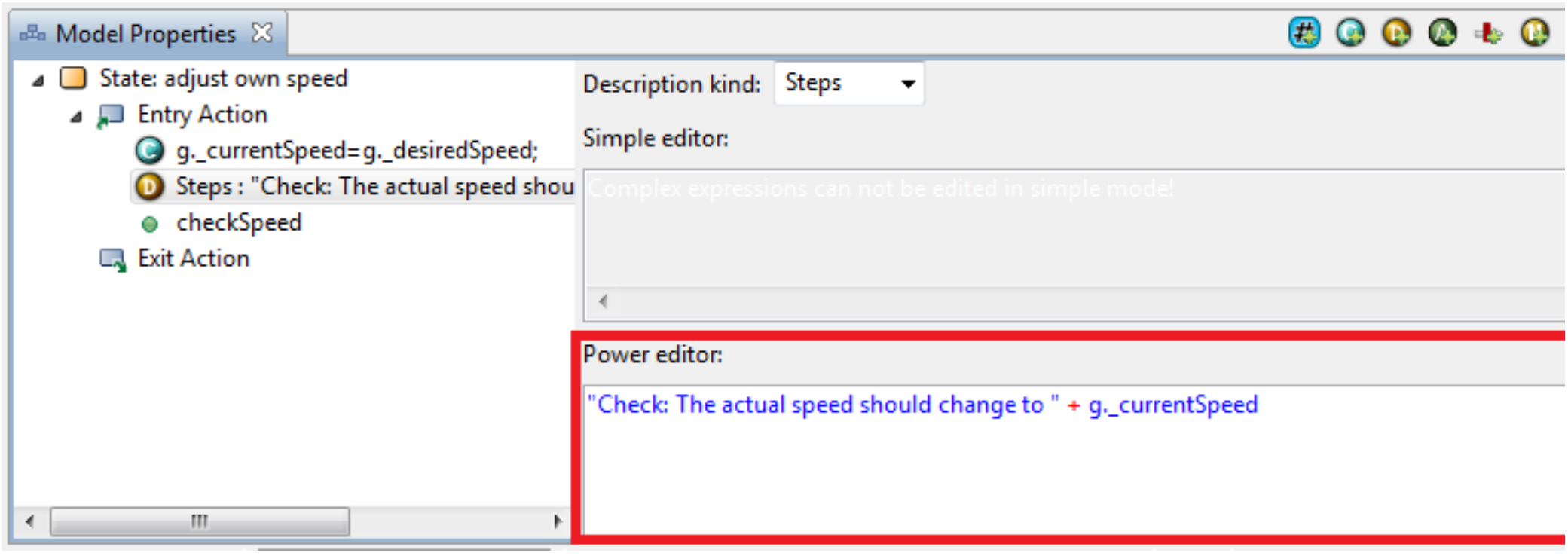
Model Properties View, Description - Simple Editor

Power Editor

The *Power Editor* is used if calculations and variables are to be added to the *descriptions*.

The following rules apply to the spelling of the *description* text in the *Power editor*:

- pure text must be written in quotation marks
- Variables can be dragged from the *Variables View* to the desired location in the description text
- To use the values of *variables* in a description text, they must be enclosed with quotation marks in the corresponding place and "added" to the text



Model Properties View, Description - Power Editor

Assuming the variable currentSpeed has the value `55`, the result of the *description* text looks as follows:

> **The actual speed should be changed to 55**

Special characters:

- The following characters are not allowed in a description text:
  - {}
  - []

Naming Fragments

Naming fragment is a fragment of the test case name.

Similar to *descriptions*, test case names can be composed in this way. For this purpose, an element *Naming Fragment* must be added at all points in the model where a part is to be added to the test case name.

The **rules** for the use of a *naming fragment* are the same as for the *description* element.

Special characters:

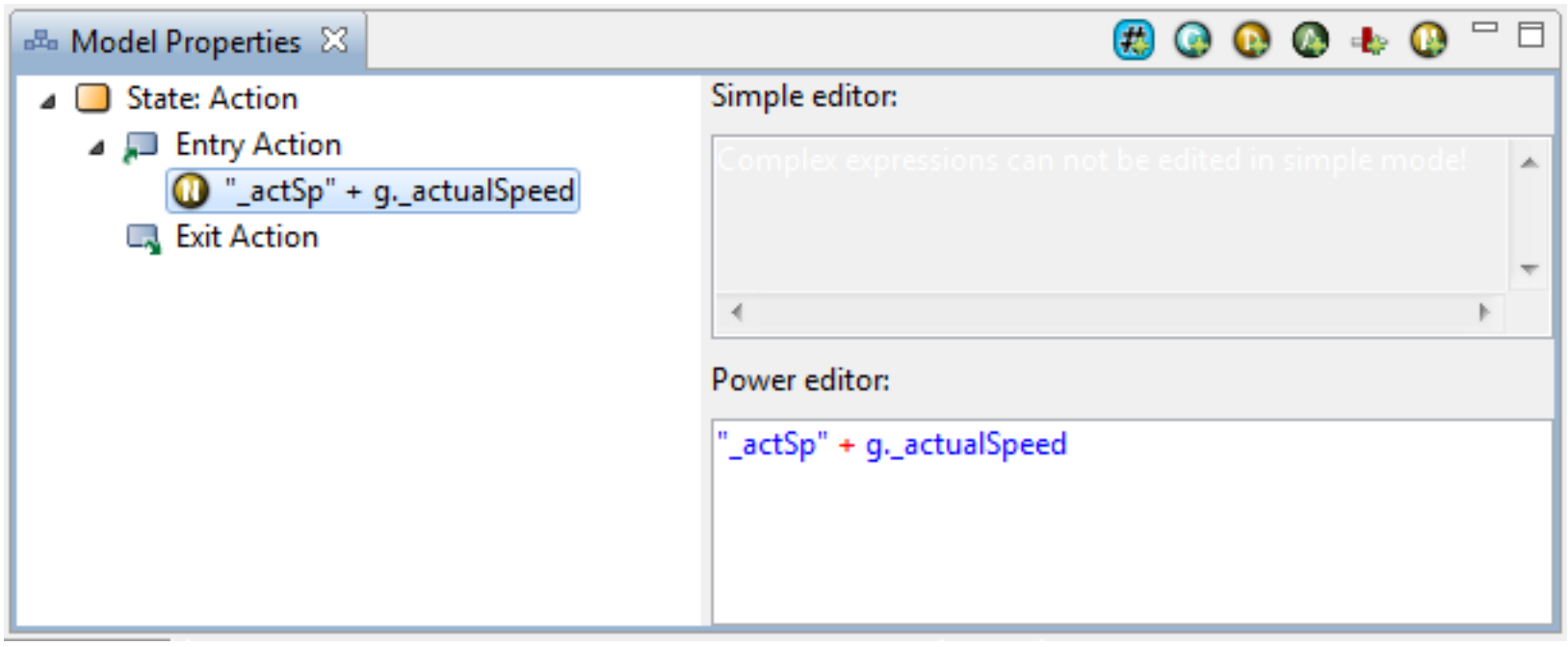
- Test case names may only consist of numbers [0 - 9] and large [A - Z] as well as small [a - z] letters and underscores. The name must begin with a letter.

The *NamingFragment* element is identified by a  in the *Model Properties View*.

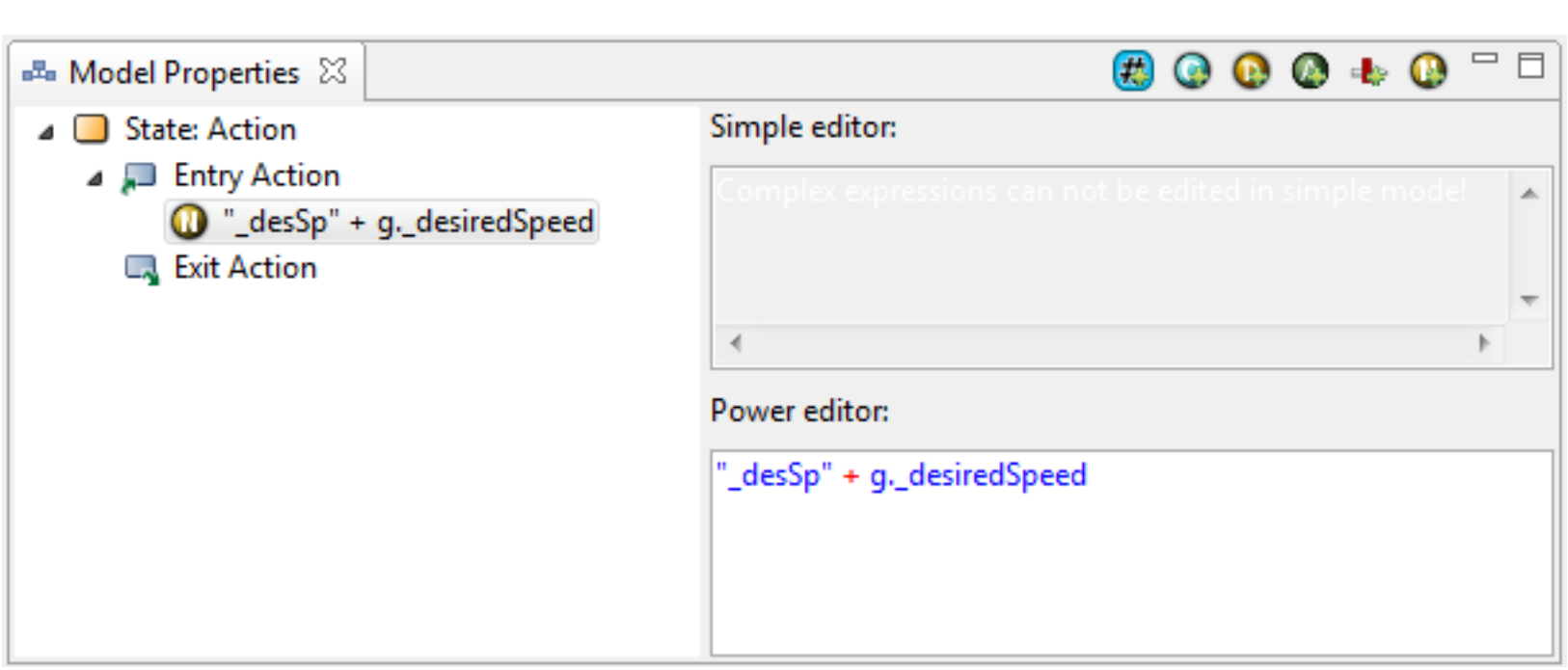
Example:

In the following, two *naming fragments* are shown at two different points in the model, from which a test case name is created.

Naming fragment at point 1:



Naming Fragment at point 2:



If actualSpeed is `50` and desiredSpeed is `70`, the test case name would look like this:

_actSp50_desSp70

IMPORTANT: As soon as a *naming fragment* is assigned in the model, it is used for the naming of the test cases. In this case, the modeler must ensure that the names of the test cases differ by the naming fragments. Otherwise, the names may not be exported to the test environment (for example, EXAM).

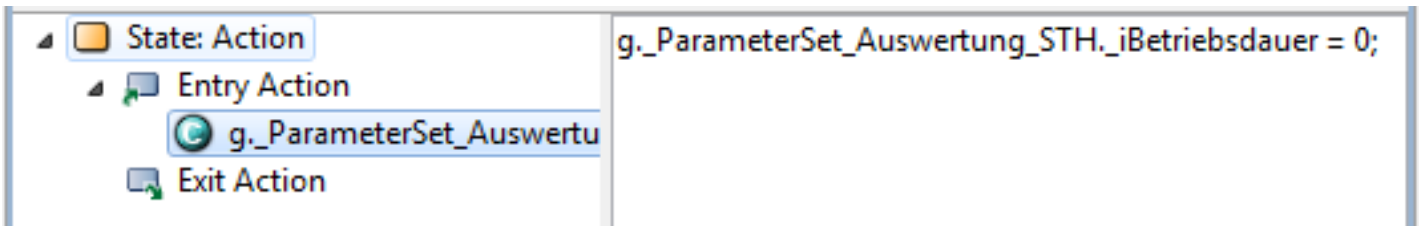
Calculations

In a *calculation*, simple computations and value assignments to MODICA *variables* can be performed.

Via Drag & Drop, *variables* can be added to the writing area of a *calculation* and used there.

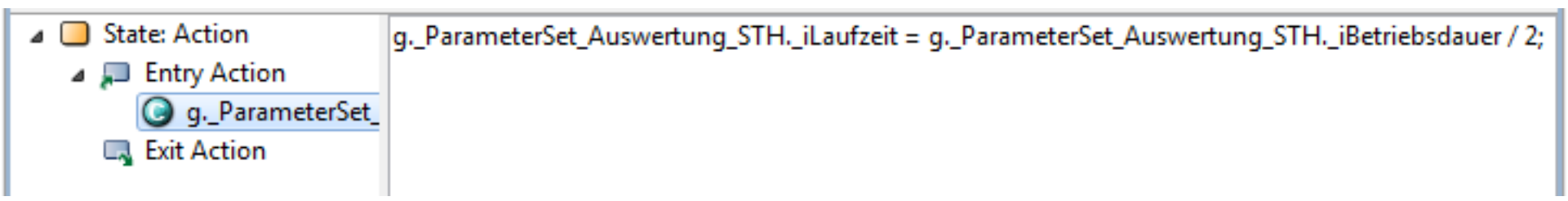
The following actions can be performed in a *calculation*:

- simple value assignments of variables



- simple calculations

- simple calculations and use of variables



The following simple calculation modes are supported in MODICA:

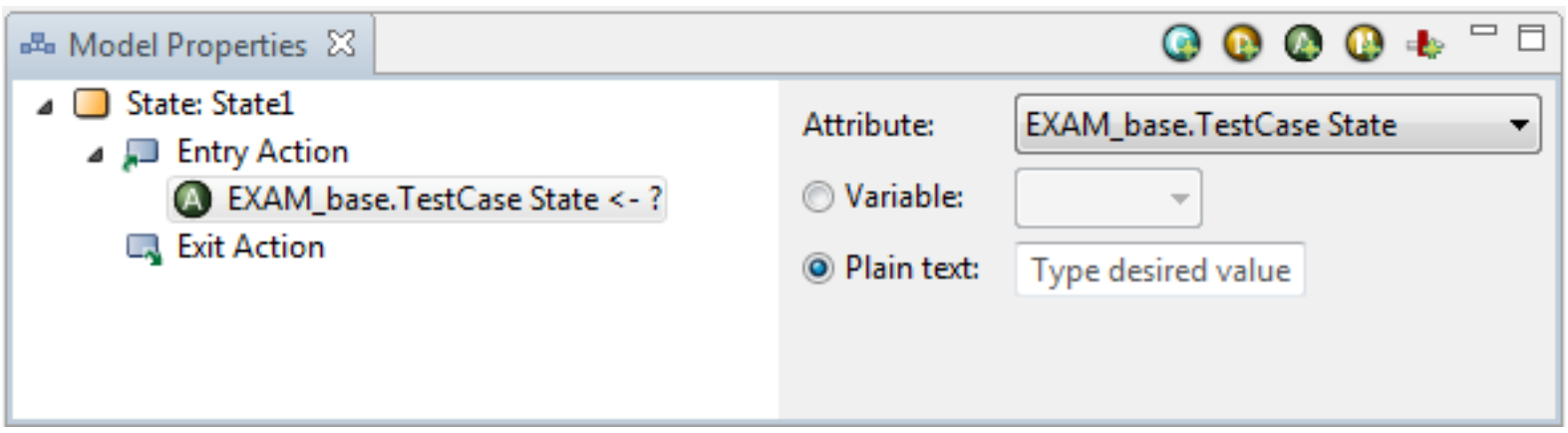
- Addition (a = b + c)
- Subtraction (a = b – c)
- Multiplication (a = b * c)
- Division (a = b / c)

The name of each *calculation* field can be changed by right-clicking on it and selecting *Rename* in the context menu.

Attributes

Attributes are used to set meta data for a test case.

In the right-hand editor field, an *attribute* is selected via the drop-down menu. Below you can assign the value of a *variable* or a specific text to this *attribute*.

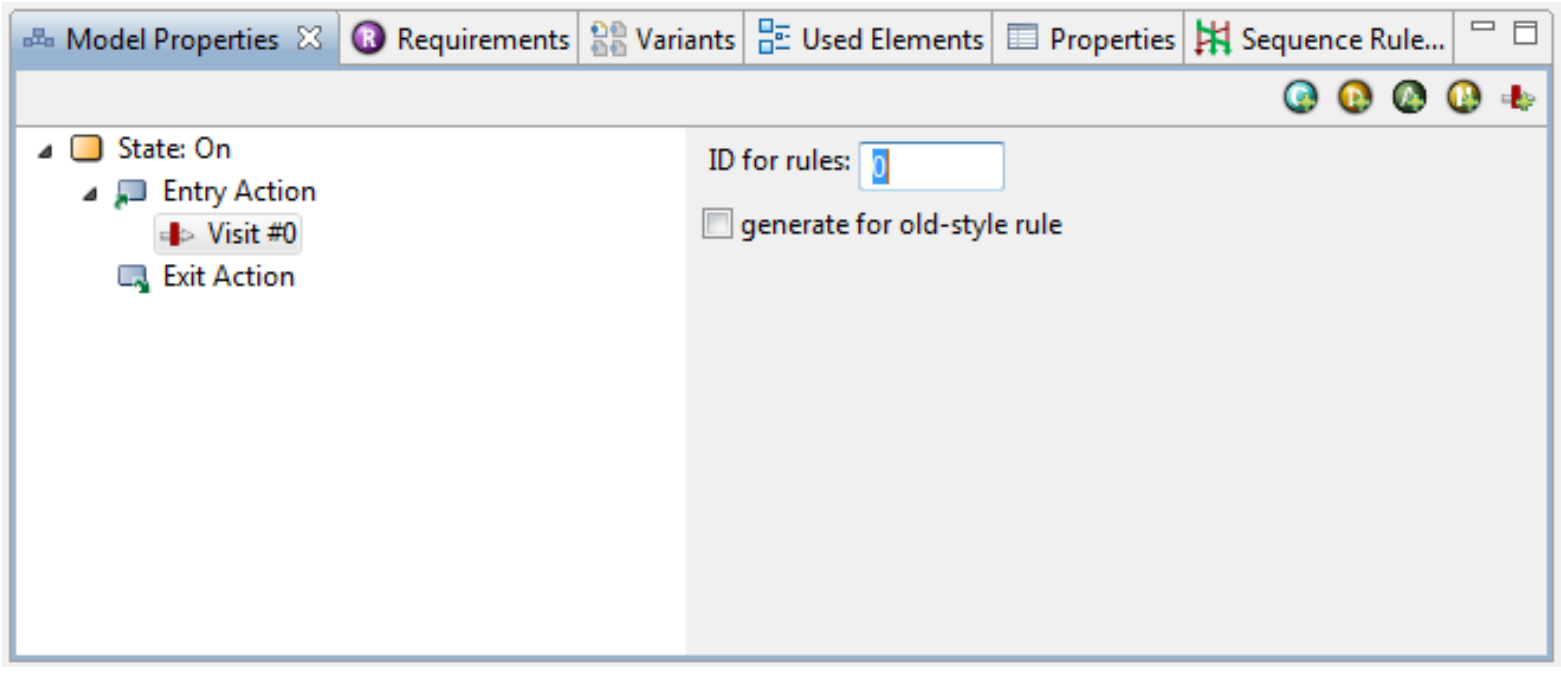


Visits

Visit elements are used to describe predefined paths through a model by means of sequence rules.

With the help of the *visit* elements, markers (IDs) are set on model elements (*states, transitions*). Via these markers you can use the related *states* or *transitions* to define sequence rules.

Example of creating a *visit* object:



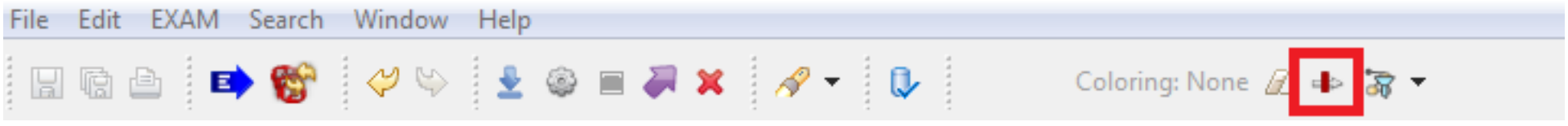
After creating a *visit*, it must have a unique ID to use it in seqeunce rules. This ID must be assigned by the user in the "*ID for rules:*" text field. An ID that has not yet been used is preselected for convenience.

IMPORTANT: Each visit ID must be unique in a MODICA project. The user has to take care of himself that each visit is unique!

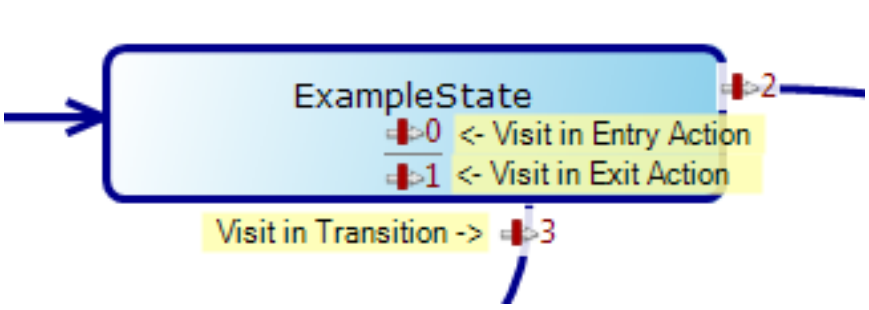
As long as no sequence rules are defined or these are not selected for the generation, *visits* have no influence on the test case generation.

Display Visit IDs and find them

To facilitate the work with *Visit*-IDs, there are some usability features. A basic function to get an overview of the given *visit* IDs is available via the button in the main toolbar:

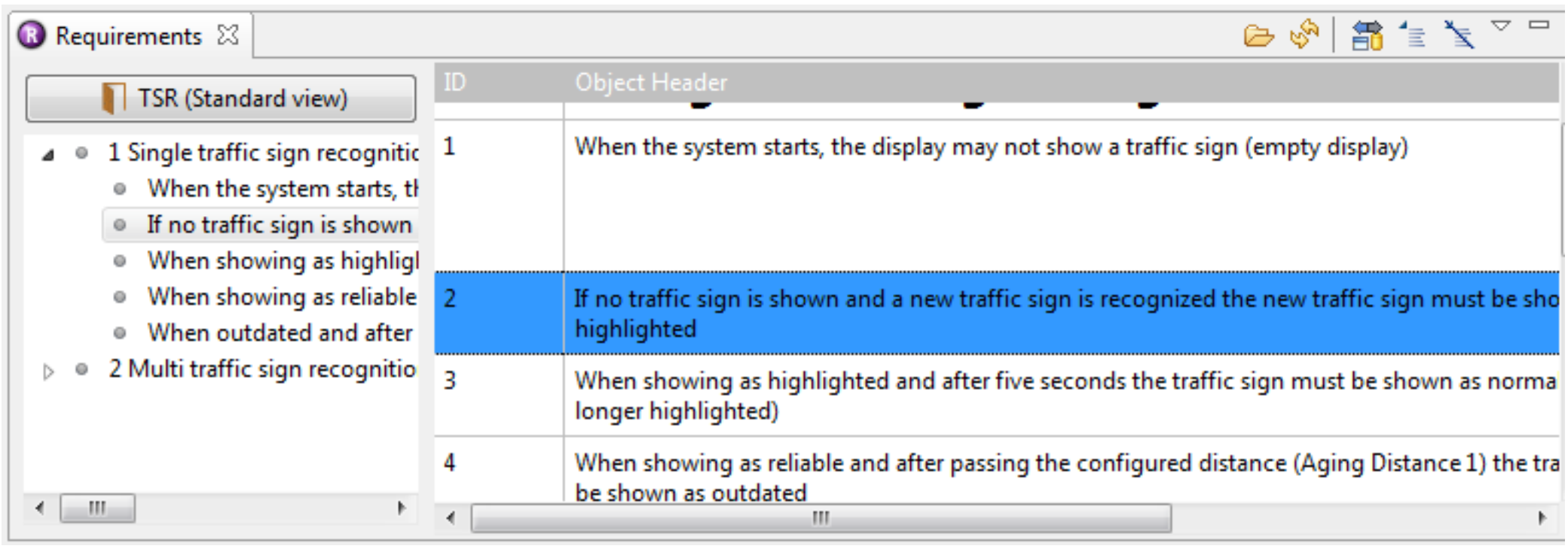


If this is selected, Visit IDs appear in the Statechart editors at the respective *transitions* or *states*:



Requirements

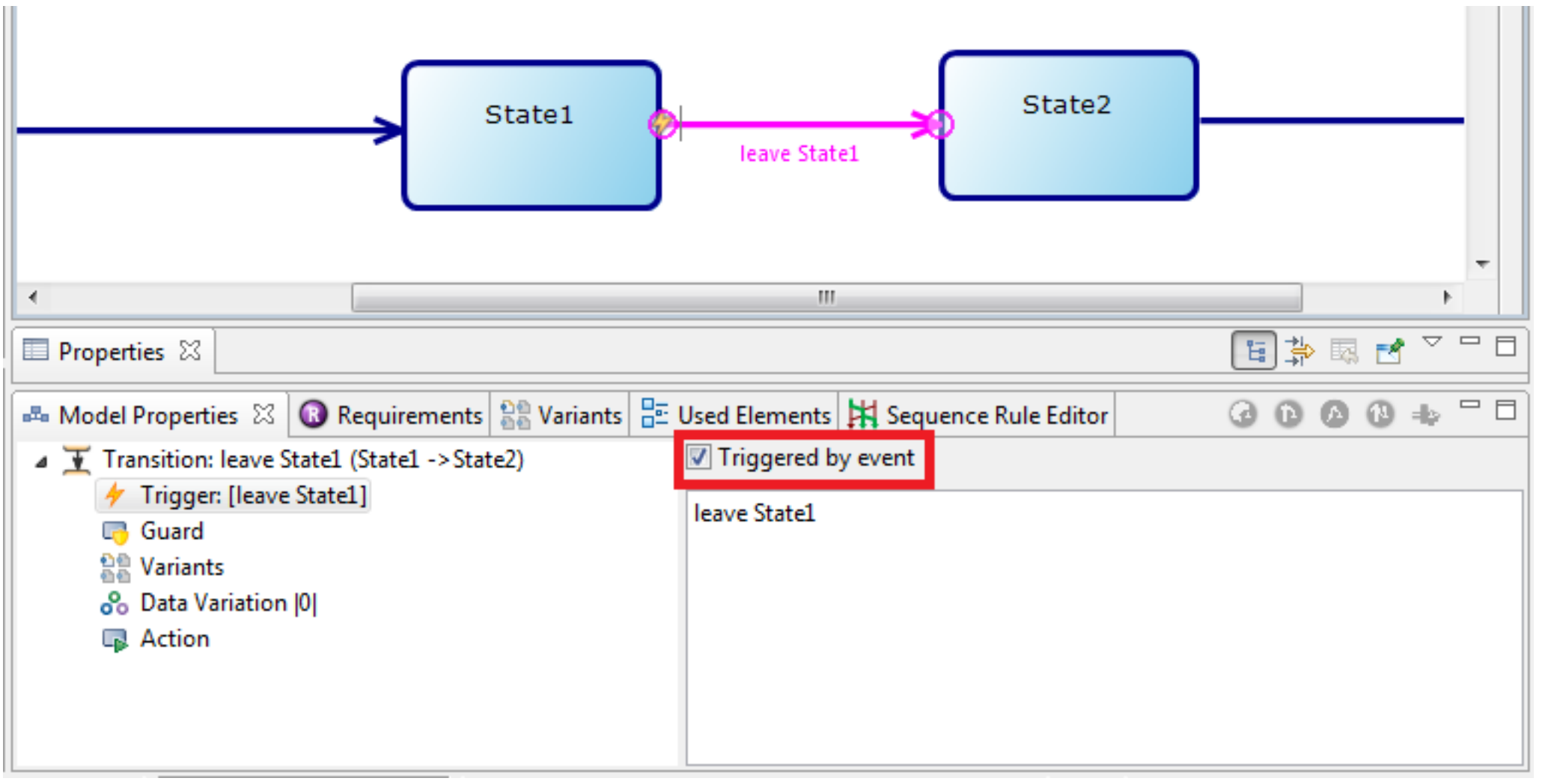
Requirements are dragged and dropped from the *Requirement Explorer* into the *entry* or *exit action*. Thus, a *requirement* can be marked and assigned in a *state* (*entry* or *exit action*) or in a *transition* (*action*).



A *requirement* is indicated by the icon in the *Model Properties View*.

Triggers

The *trigger* is activated by clicking the *Triggered by event* box. By default, [auto] is used. It signifies an automatic, ID-based trigger. Expert feature: It is possible to manually change the trigger by double clicking on the trigger box (in the example leave State1 below).



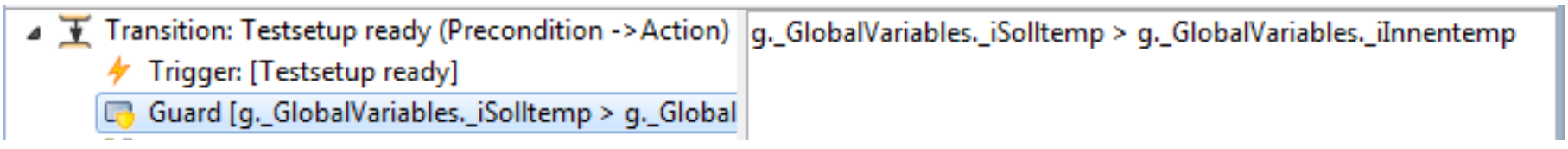
Guards

The *guard* is responsible for either allowing or preventing a *transition*. The screenshot shows the *guard* of a *transition* that is to be executed only if the variable `iSolltemp` is larger than the variable `iInnentemp`.

The *Guard* (Guard) is filled by hand like a *calculation*. Variable references can be inserted by dragging and dropping the corresponding *variable*.

Attention:

- An expression must not be terminated with a semicolon
- Comments '//' are not allowed in the input field for *guards*



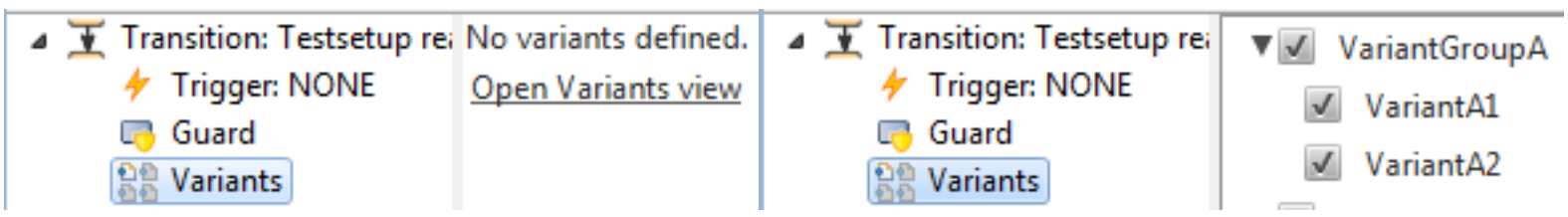
A *guard* must not have a content.

- If a *guard* is not filled, the *transition* is always traversed.
- If a *guard* is filled, the respective *transition* is only traversed when the condition is fulfilled.

If a condition is present in a *guard*, this is graphically indicated by the first characters of the condition being displayed behind the Guard icon.

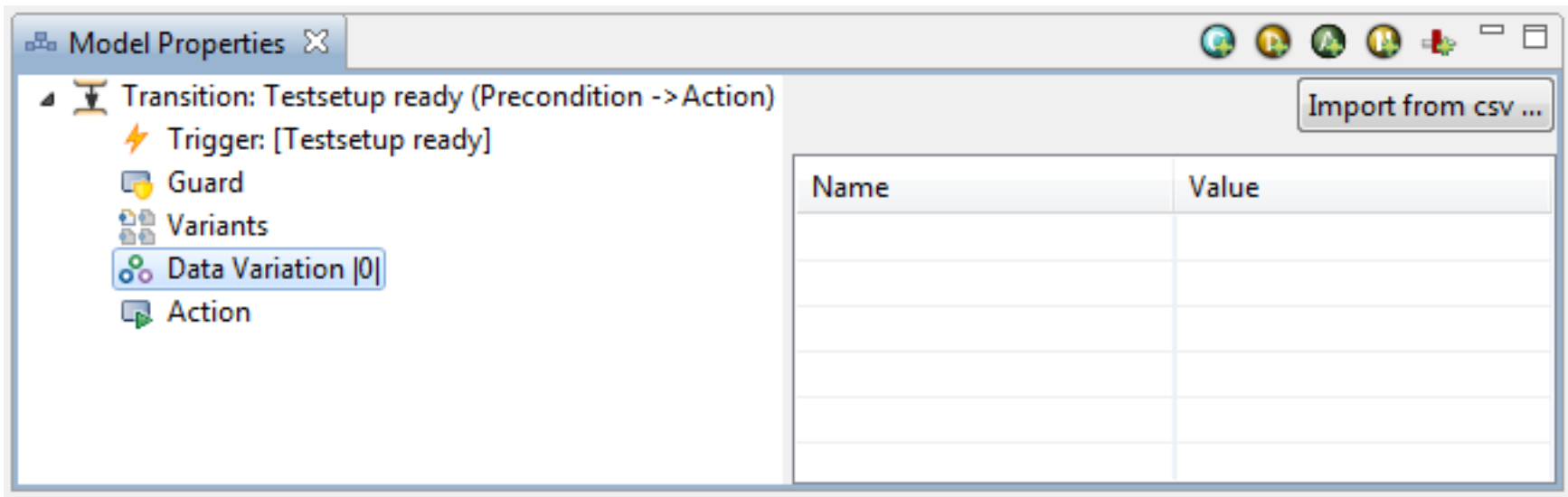
Variants

By means of the variant entry, certain *variants* can be assigned to the selected *transition*.



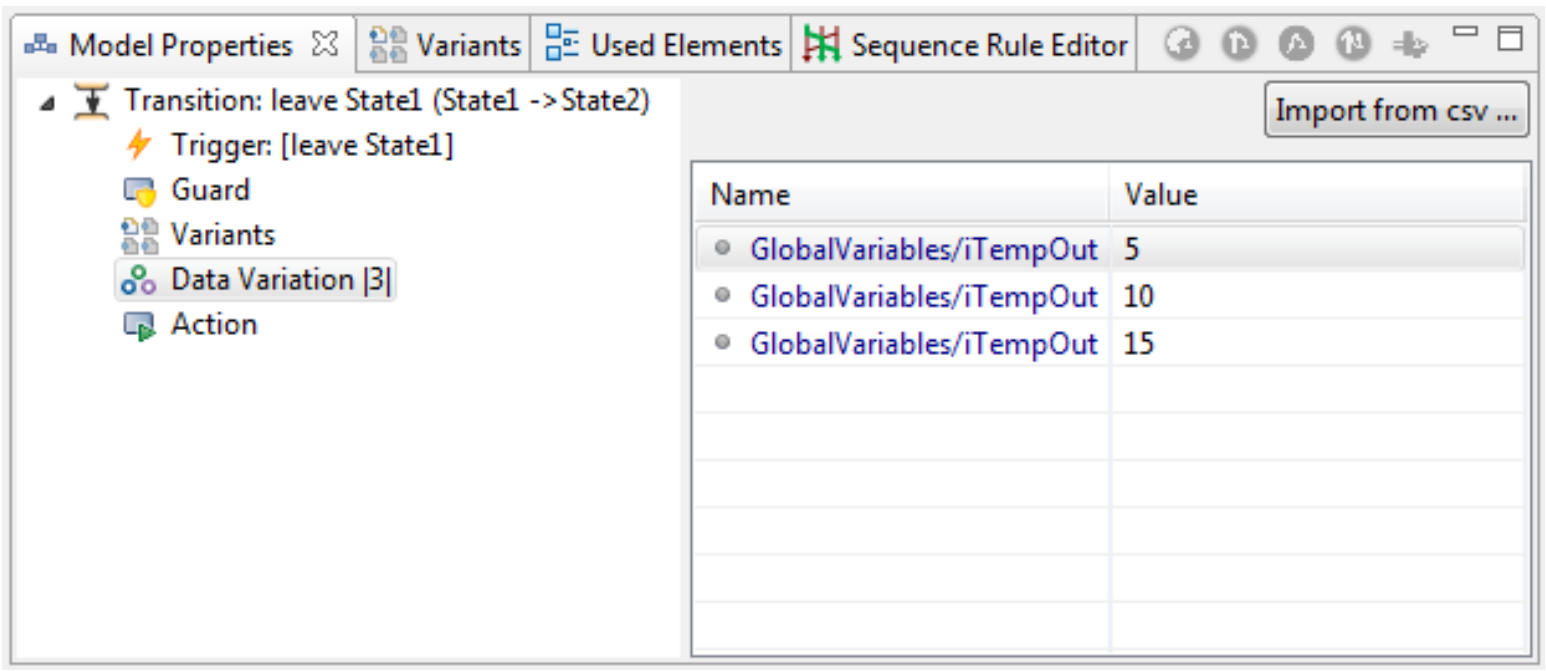
Data Variations

Data Variations are used to assign different values to a *variables* when a *transition* is passed. *Data Variations* are created by adding one or more variables to the Data Variation |0| - node of a *transition* by drag-and-drop.



The *Data Variation* node always displays the amount of variations of *variable* values. It can also be renamed by right-clicking and selecting *Rename* in the context menu. To assign a *variable* to the *data variation*, simply drag a *variable* to the right column of the *Model Properties View*. To assign different values, add them several times and change the values by double-clicking the value column. In the displayed name of the entries, the group names and the *variable* name are separated by "/".

In the following example, the variable `iTempOut` is to be assigned to one of the values 5, 10 or 15 when generating a test case.



An entry within a variation can be deleted by selecting and pressing the `Del`-key or by *right-clicking* → *Delete*.

It is also possible to import *data variations* from an external csv file. In the csv file, the variable name must be separated by a semicolon from its associated value, and groups can be used by prefixed `'group name/'`. The used *variables* must already exist in the *Variables View* **before** the import.

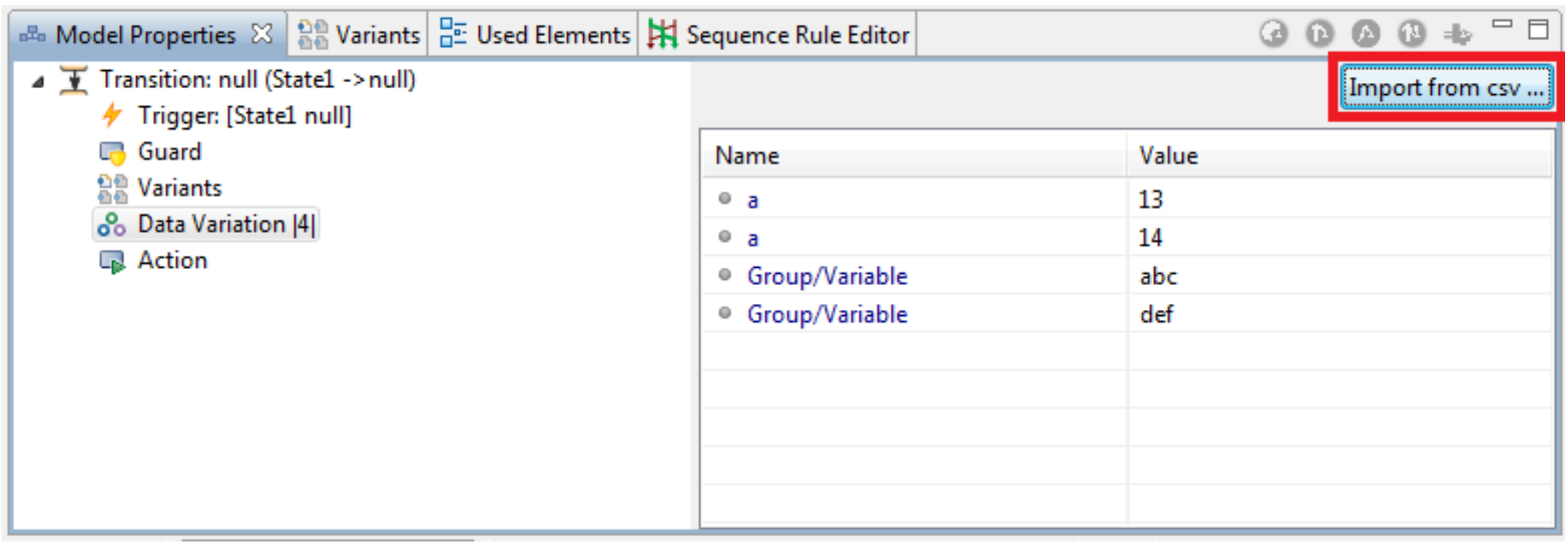
example.csv

a;13

a;14

Group/Variable;abc

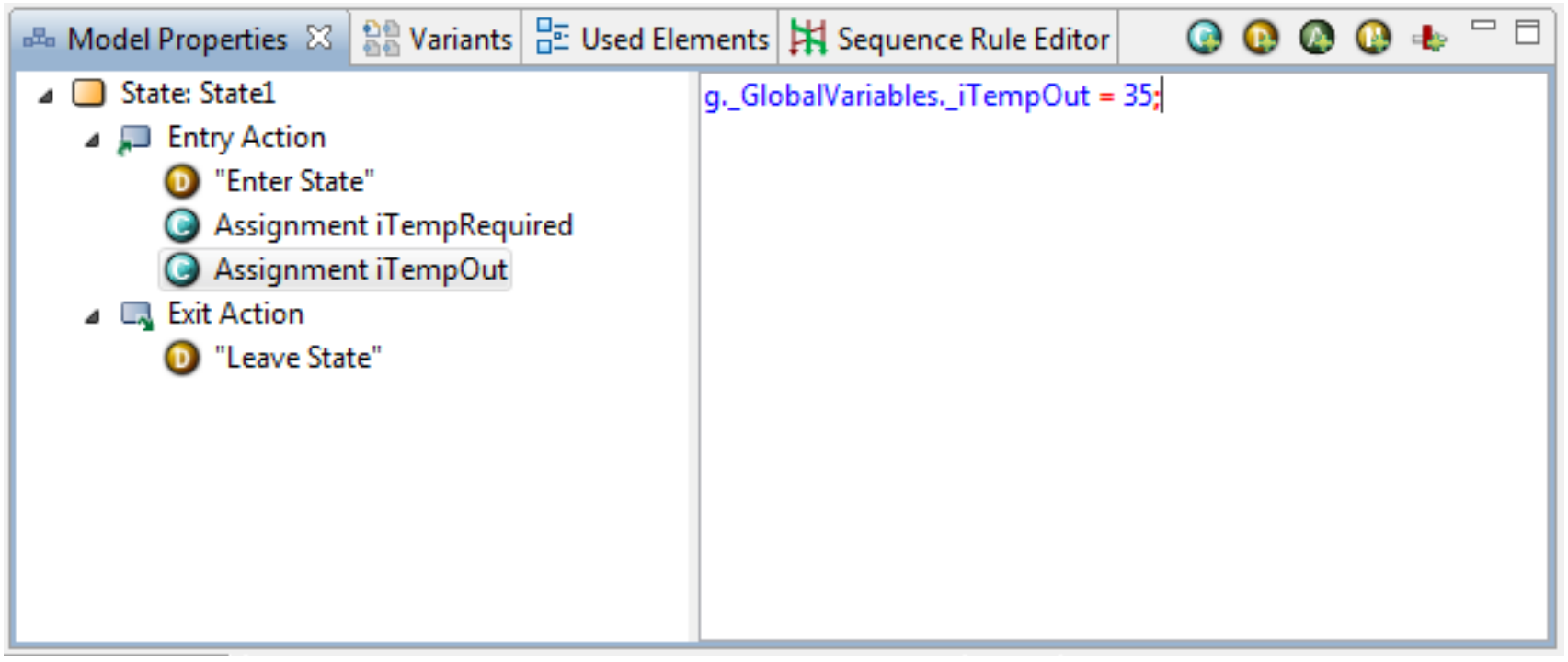
Group/Variable;def



In the course of the test case generation, test targets can be created on the basis of the *Data Variations* in the *Test Target View*.

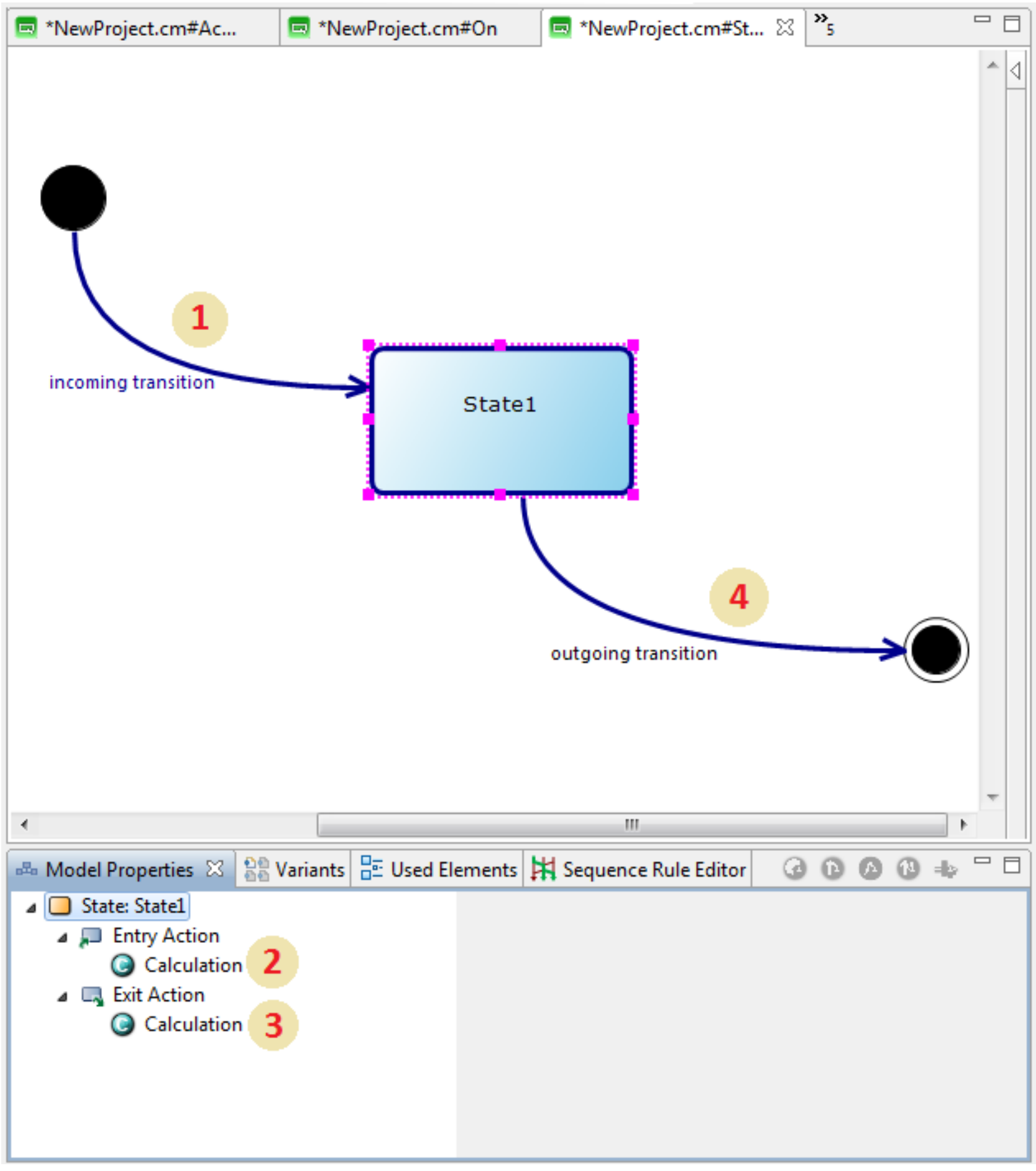
Order of execution

If there are several elements in an *action*, they are executed in the order as they are shown in the *Model Properties View.*



In the above example, this means that the `iTempRequired` assignment would occur before the `iTempOut` assignment. The same applies with regard to order if, for example, two EXAM functions are called one after the other.
The basic sequence of *actions* in a MODICA state chart is as follows:

1. First, the incoming *transition* (and its *actions*) is traversed.
2. The following is the processing of the *entry Action* of the *state* `State`.
3. After this, the *exit action* of the *state* `State` is executed.
4. Finally, the outgoing *transition* is passed.


*NewProject.cm#Ac...
*NewProject.cm#On
*NewProject.cm#St...
5
1
incoming transition
State1
4
outgoing transition
Model Properties
Variants
Used Elements
Sequence Rule Editor
State: State1
Entry Action
Calculation
2
Exit Action
Calculation
3

Test Generation Perspective

The following pages describe the most important views of the *test generation perspective*:

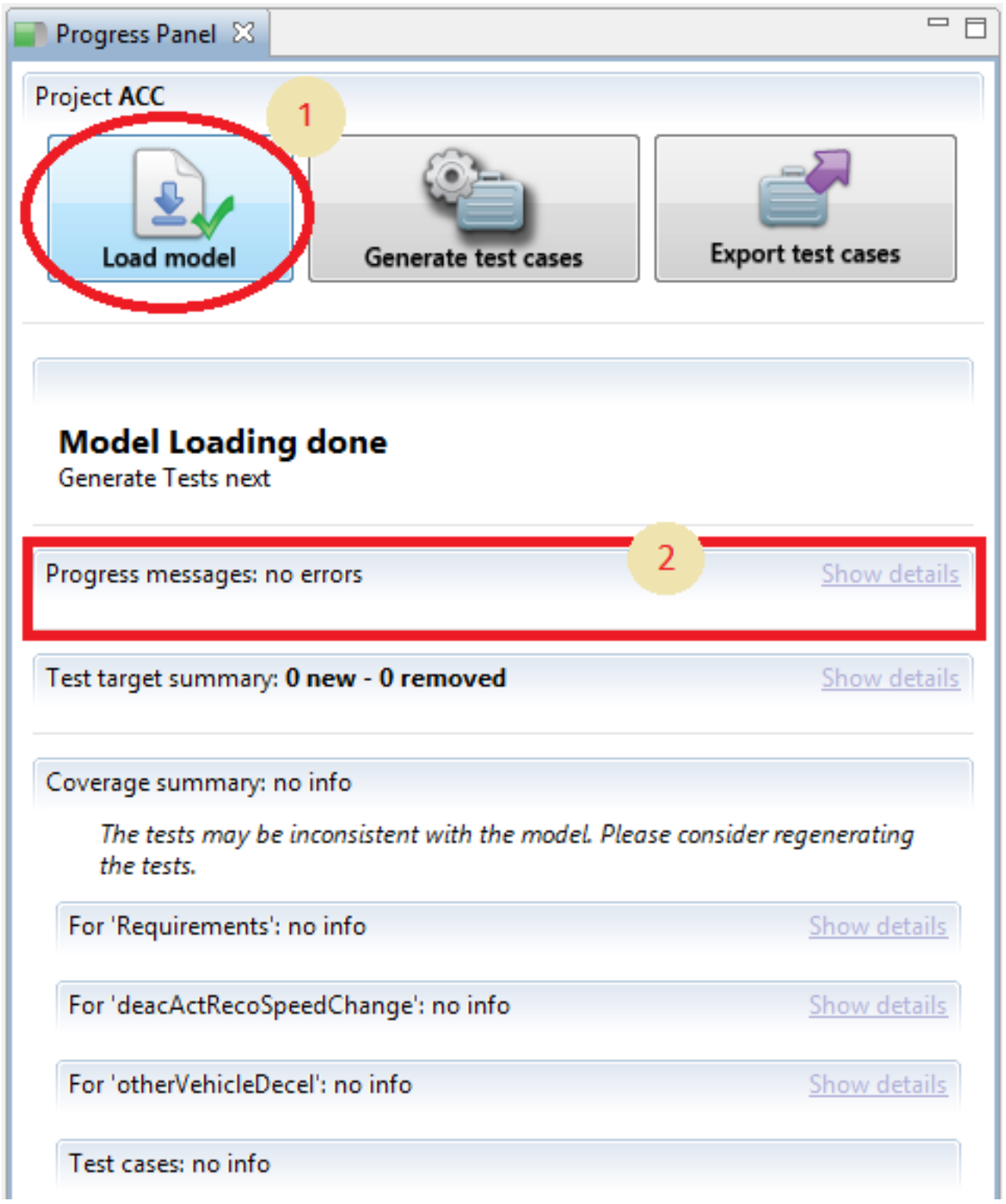
Progress Panel (en)

After switching from the *Modeling Perspective* to the *Test Generation Perspective*, the *Progress Panel* is the first view to continue.

Here, the created model is loaded, test cases are generated according to specific settings and then exported.

Load Model

With a click on *Load-Model* (1), the model is loaded and checked for (syntax) errors, which are displayed below the three large buttons in a log overview (2).



Hint

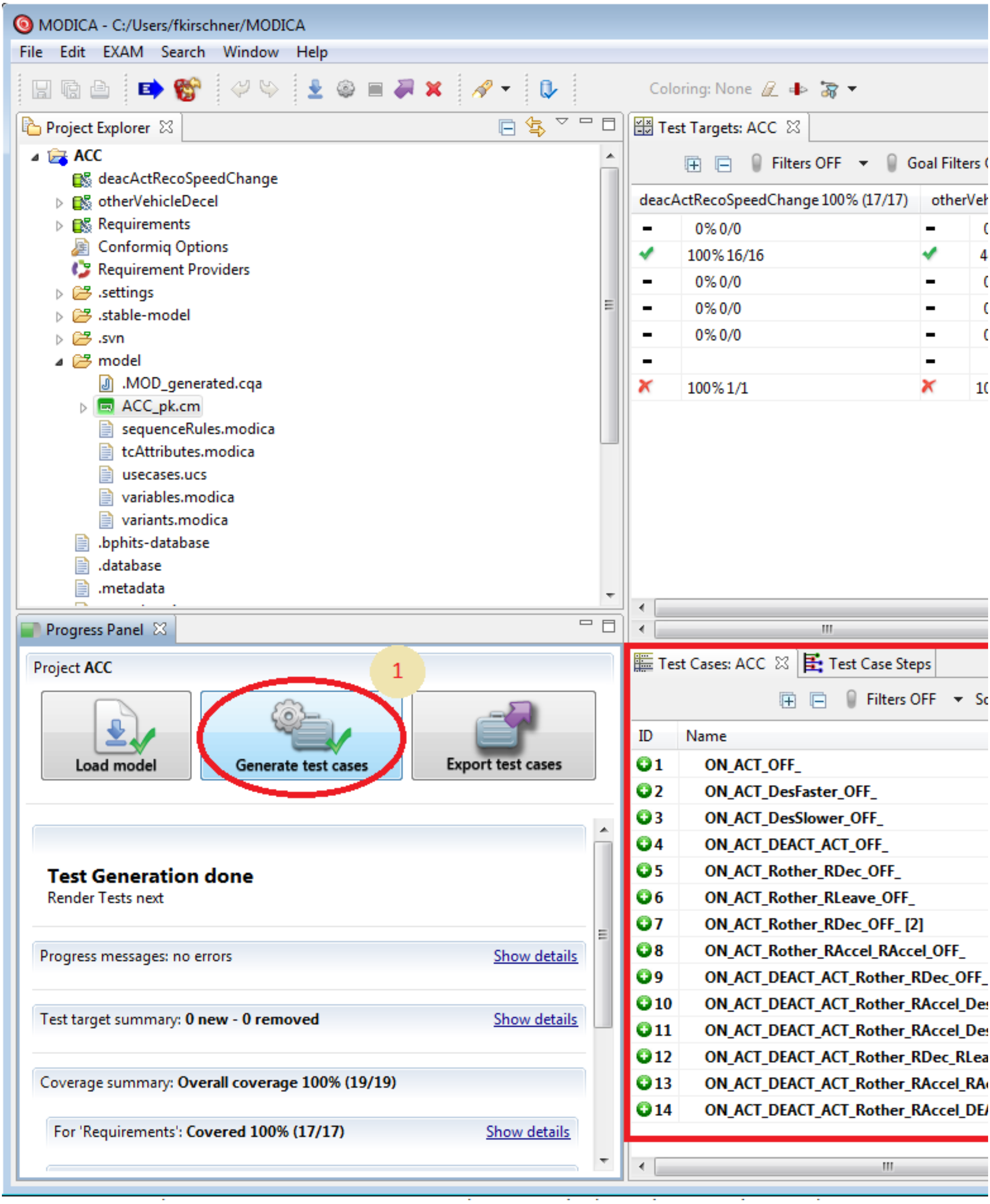
In order to avoid the error messages in the Progess panel, the model should be validated before loading.

To do this, you must switch to the *modeling perspective* and click on the  button in the upper toolbar. If errors are present, they are displayed in the *Problems View*. The localization and elimination of the errors is here much easier and more understandable than in the *Progress Panel* itself. A more detailed documentation on project validation can be found in the chapter Validation of the modeling.

Generate Test Cases

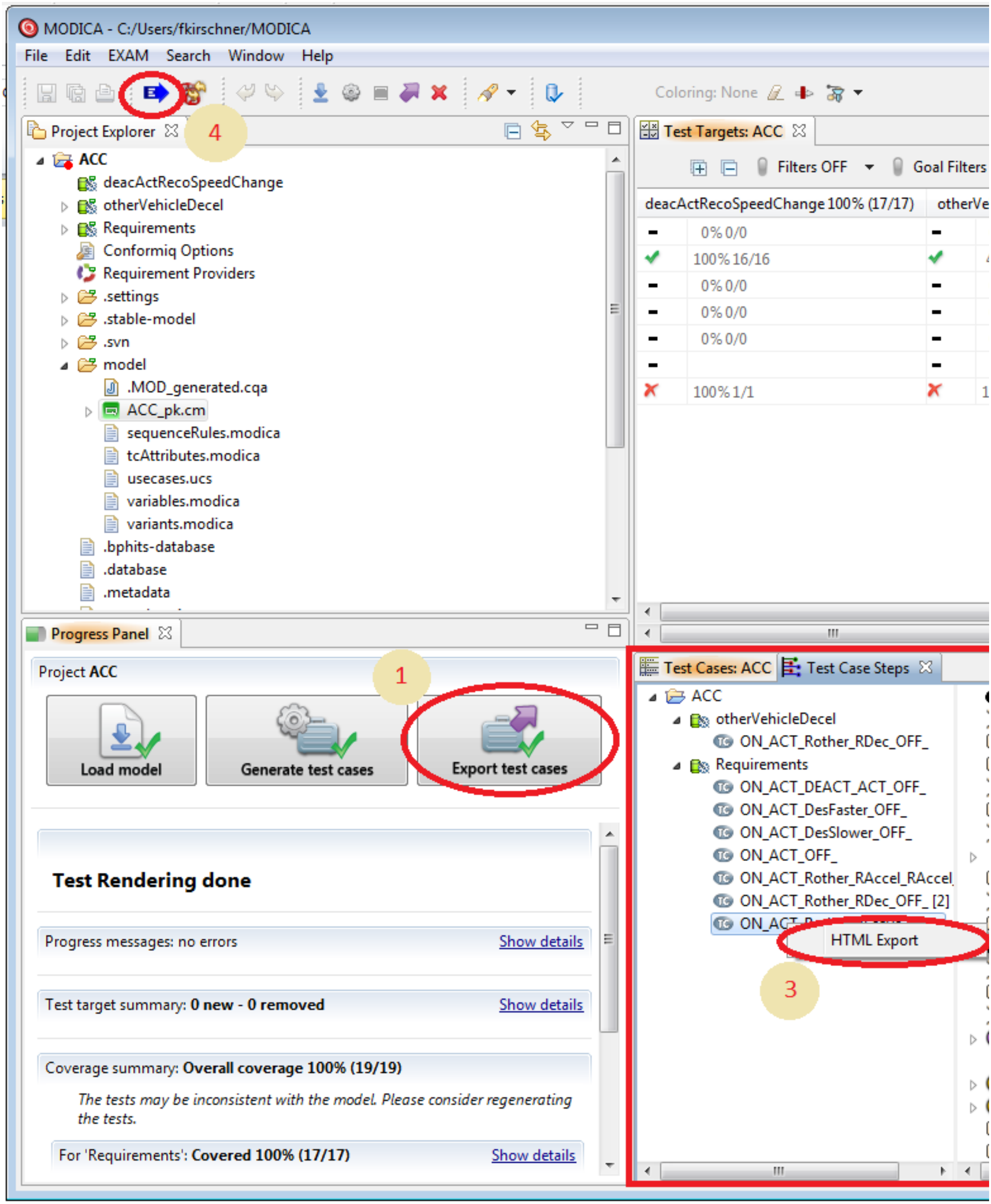
If you press the *Generate test cases button* (1), the corresponding test cases are generated according to the *design configuration* and the *Test Targets View* and displayed in the *Test Cases View* (2).

Note: The button *Generate test cases* can only be pressed if a *design configuration* has been added to the project!



Export Test Cases

When the *Export Test Cases* (1) button is pressed, the generated test cases (displayed in the *Test Cases View*) are put into a readable form and grouped according to the *design configurations* in the *Test Case Steps View* (2). From here, further steps for final exporting into an HTML format (3) or into the environment of a test execution program (4) can now be carried out.

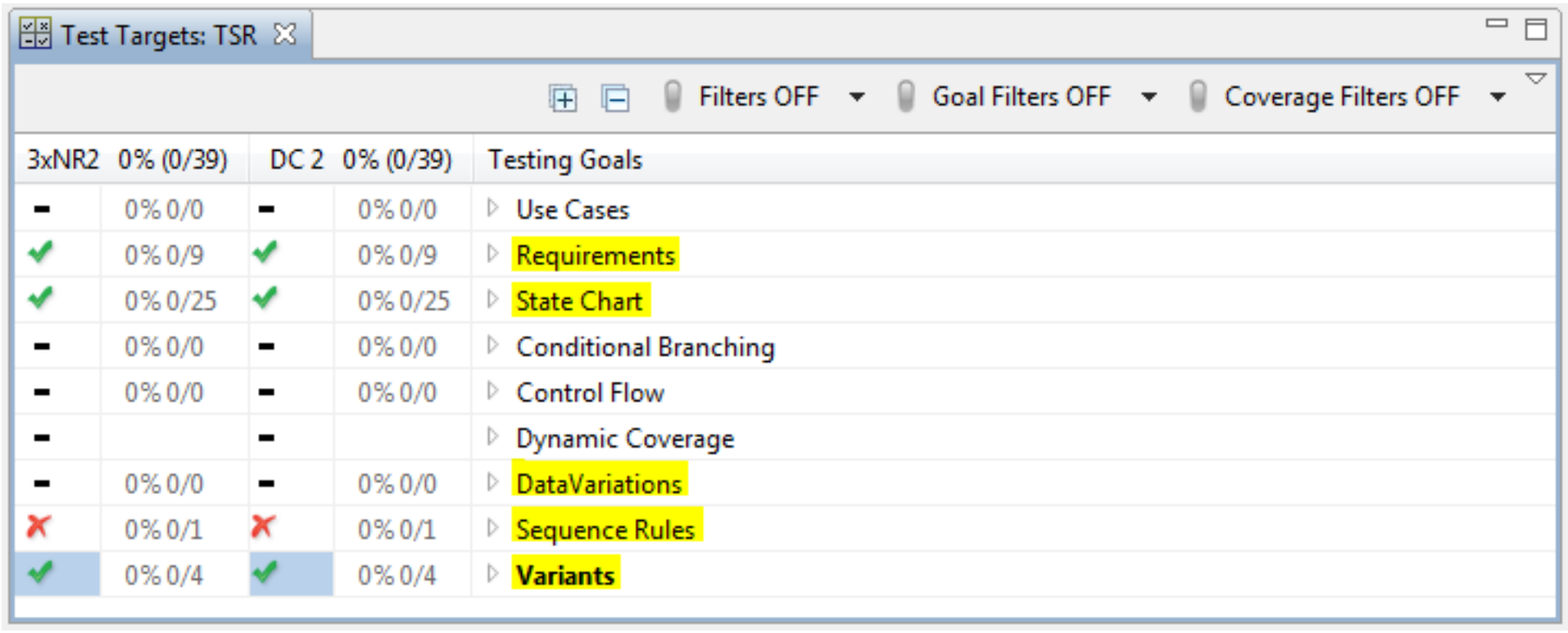

MODICA - C:/Users/fkirschner/MODICA
File Edit EXAM Search Window Help
Coloring: None
Project Explorer
4
ACC
deacActRecoSpeedChange
otherVehicleDecel
Requirements
Conformiq Options
Requirement Providers
.settings
.stable-model
.svn
model
.MOD_generated.cqa
ACC_pk.cm
sequenceRules.modica
tcAttributes.modica
usecases.ucs
variables.modica
variants.modica
.bphits-database
.database
.metadata
Test Targets: ACC
Filters OFF
Goal Filters
deacActRecoSpeedChange 100% (17/17)
otherVe
0% 0/0
100% 16/16
0% 0/0
0% 0/0
0% 0/0
100% 1/1
Progress Panel
Project ACC
1
Load model
Generate test cases
Export test cases
Test Rendering done
Progress messages: no errors
Show details
Test target summary: 0 new - 0 removed
Show details
Coverage summary: Overall coverage 100% (19/19)
The tests may be inconsistent with the model. Please consider regenerating the tests.
For 'Requirements': Covered 100% (17/17)
Show details
Test Cases: ACC
Test Case Steps
ACC
otherVehicleDecel
ON_ACT_Rother_RDec_OFF_
Requirements
ON_ACT_DEACT_ACT_OFF_
ON_ACT_DesFaster_OFF_
ON_ACT_DesSlower_OFF_
ON_ACT_OFF_
ON_ACT_Rother_RAccel_RAccel
ON_ACT_Rother_RDec_OFF_ [2]
HTML Export
3

Test Target (en)

Structure

In the *Test Target View*, the different groups of testing goals are displayed - marked yellow in the screenshot. By default, this view does not always look like this, *Requirements, DataVariations, SequencesRules*, and *Variants* are only displayed if they are actually present in the model.

Once a model has been loaded, the test goals are filled with the available test goals of the active model. Thus, e.g. you can find all states of the current model in the group *State Chart → States*.



Test Target View - Aufbau

For example, if a recently added *Sequence Rule* is not displayed as a Testing Goal, this may be because the *Test Target View* is still referring to the old model. Simply press the *LoadModel* button again in the *Progress-Panel* and the *Sequence Rule* will appear as a test goal.

Setting options for the testing goals

A criterion in the hierarchical tree of the testing goals can be activated or deactivated in the first column of the *Test Target View*. The following setting options are available for each testing goal which is located in the MODICA model and can be reached via the *Test Target View*:

✔ TARGET	Indicates a test target to be **reached**. A test target marked in this way causes the test case generator to attempt to create test cases in the test suite that meet this criterion (these positions / constructs in the model).
▬ DONT_CARE	Indicates a test target to be **ignored**. A test target marked in this way leads to a disregard of the test case generator. Such a test target can be either contained or missing in the generated test suite, it does not affect how the generator determines the test sequences through the model and compiles the test suite.

BLOCK	Indicates a test destination which has to be **blocked** and thus **not fulfilled**. A test target marked in this way results in the test case generator blocking this construct in the model. That is, points in the model that describes this criterion, are explicitly not included in the generated test cases since none of these criteria must occur in the test suite created.
ASSERT	Indicates a test target which **should not be reached logically** and, if so, causes the test case generation to be aborted. An assert test target is used to find errors in the model, e.g. To identify locations that should not be logical (given by the modeling and definition of the remaining test targets in this generation strategy). If the generator nevertheless manages to get to such a location, the generation is terminated and the user is informed that in this generation strategy with its settings it is possible to reach this location.
INHERIT	This option on a test destination will **inherit its setting** from the group it is in. If a state A in the group statechart is set to INHERIT, set the StateChart group to TARGET, this means that the A inherits the setting from StateChart and is thus also marked with TARGET for the test case generator.

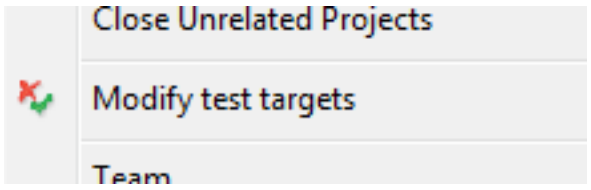
For variants and sequence rules the setting options for the test targets have a special meaning:

	✔	– (expert-settings)	✘
Variants	ON and OFF – Entries must be set contraty!		
• Entries in the ON-tree	Variant should be **selected at least once**.	Variant **may** be **selected**.	Variant **must never** be **selected**
• Entries in the OFF-tree	Variant should be **deselected at least once**.	Variant **may** be **deselected**.	Variant **can never** be **deselected**.
Sequence Rules	Only one rule set can occur in a test case. However, generation for several is possible, these are processed one after the other.		
• Entries 0-NO RULE SET	At least one test case **without** sequence rules is **to be created.**	Test cases **without** sequence rules **may** be generated.	**No test case without** sequence rules **must** be generated.

• Normal Entry (z.B. Rule Set1)	At least one test case **with** this sequence rule is **to be created**.	Test cases **with** this sequence rule **may** be generated.	**No test case with** this sequence rule **must** be generated.

Modify Test Targets Wizard

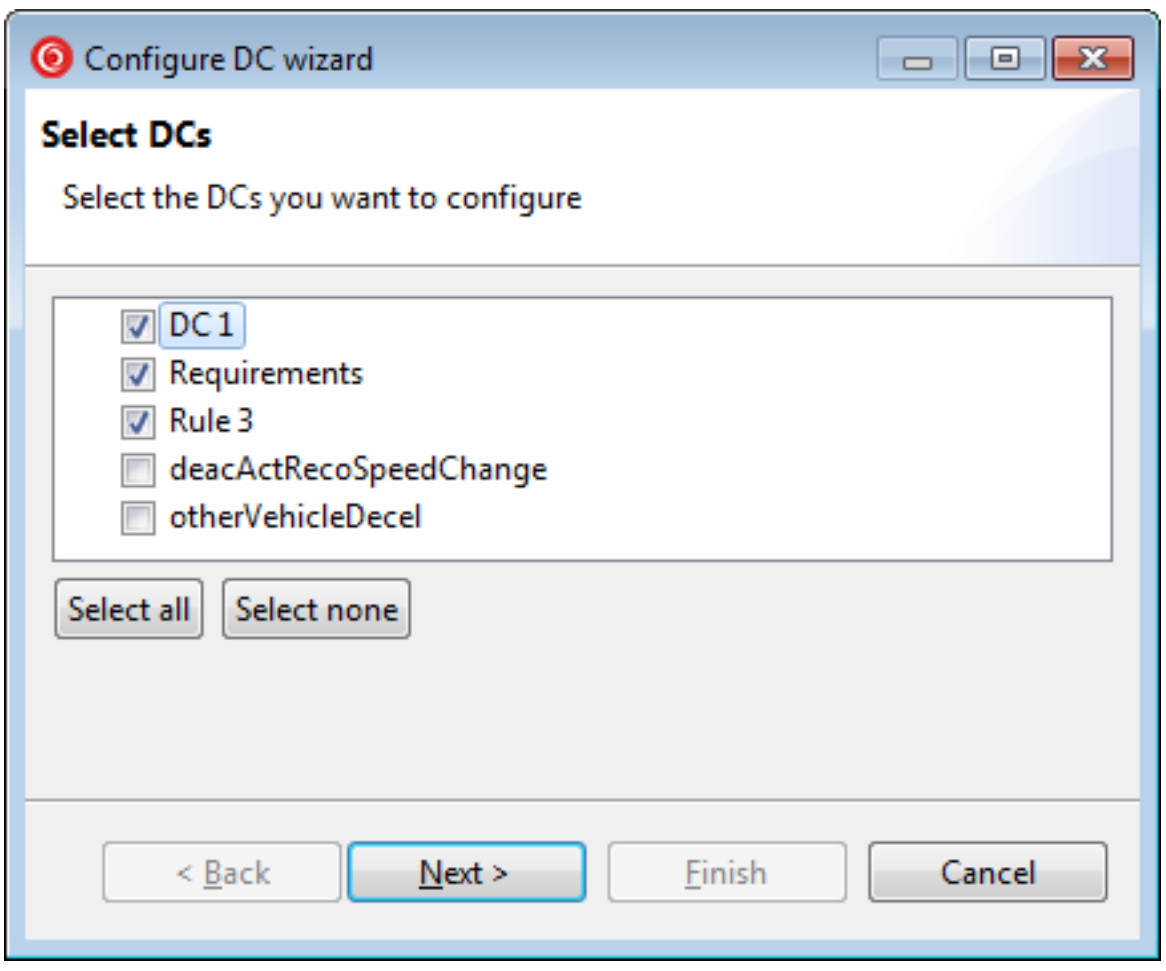
In order to simplify the settings of Test Targets and for applying the settings for multiple design configurations, offers MODICA a Test Target Wizard. The wizard can be startet by choosing "*Modify Test Targets*" in the context menu of a project or design configuration:



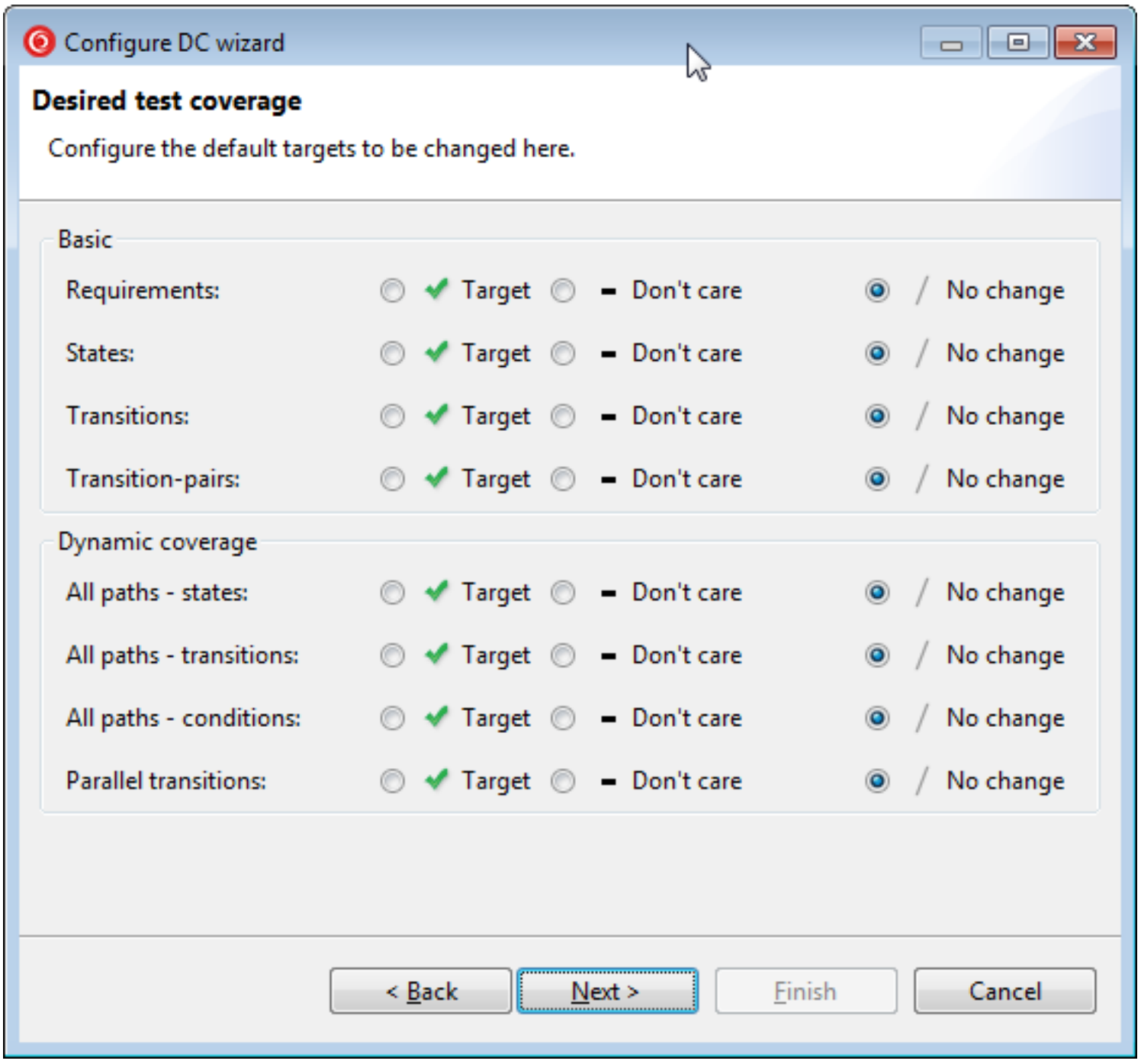
Modify Test Target Menu Entry

As a prerequisite, the current state of the model must be loaded ("Load Model"). If not already loaded, a dialog offers the possibility to do so.

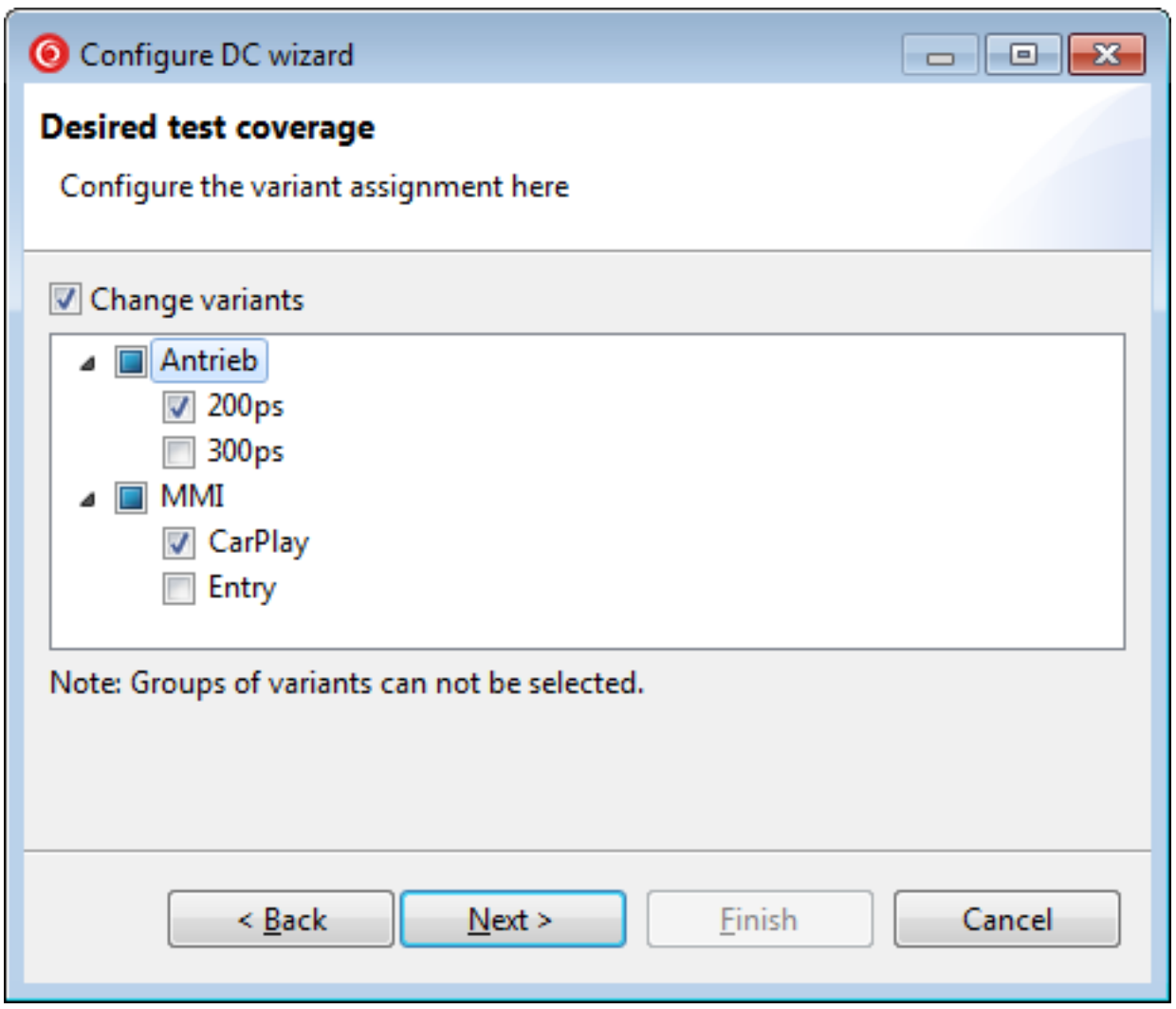
On the first page of the assistent the design configurations are choosen to which the configuration changes should apply



On the next page the basic test targets are set. The entries can be set to "Target" or "Don't Care". Alternativly the actual settings can remain unchanged ("No change").

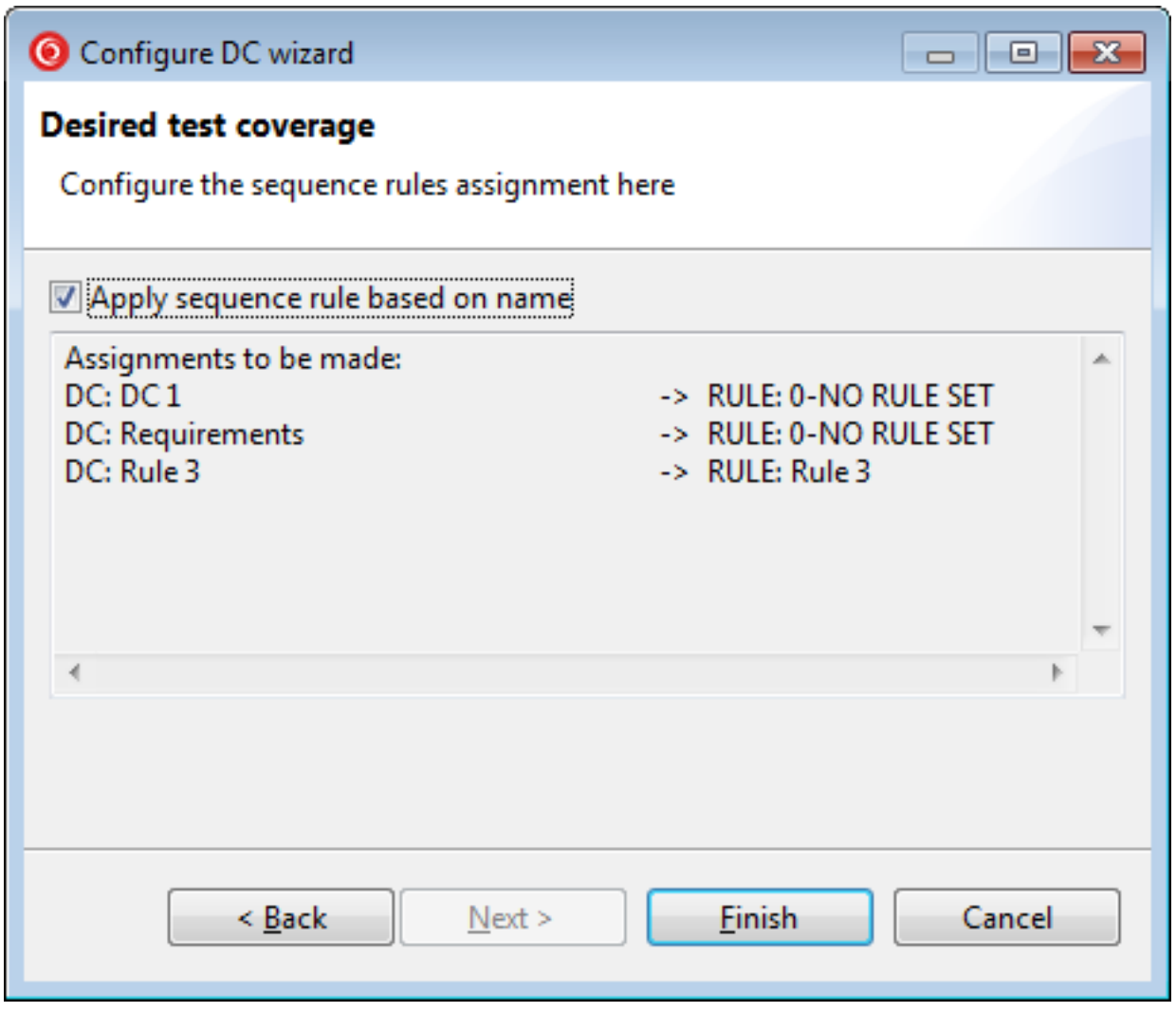


On the next page the settings for variants can be changed. If "Change Variants" is checked, the settings will be done according to the table above.



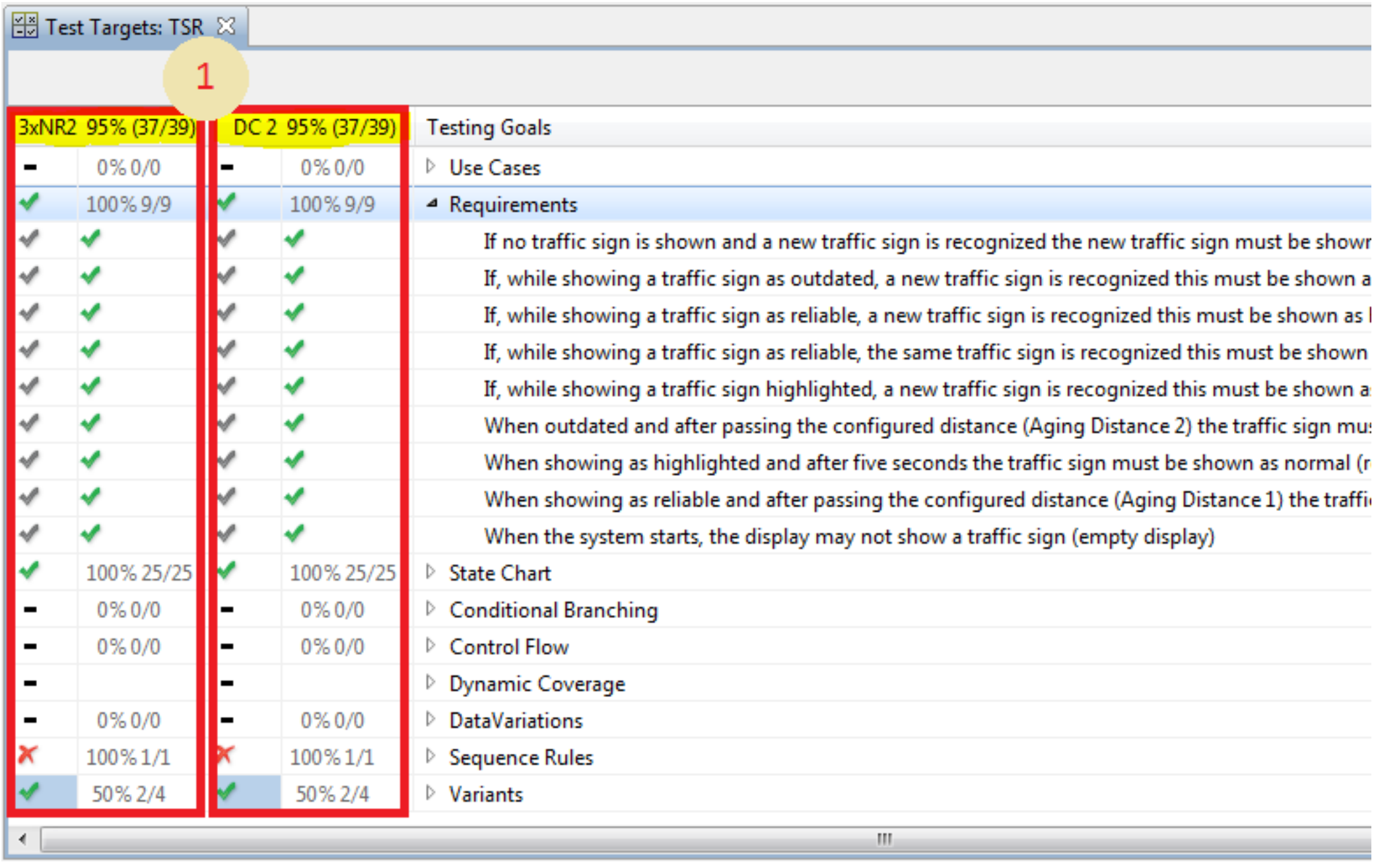
The next page allows the mapping of design configurations to sequence rules. If "Apply sequence rule based on name" is set,

the sequence rule with the same name will be set. If there is no such a rule, no rule will be set.



Finish stores the changes to the test targets. Cancel finishes the dialog without any changes.

Test case generation analysis



Test Target View - Test case generation analysis

(1)

- The different *design configurations* are displayed on the left side of the *Test Target View* - in the example the two configurations `3xNR2` and `DC2`.
- The percentage after the name indicates the percentage of the test objectives that have been covered (coverage)
- In brackets behind the name, this percentage is displayed depending on the individual test targets.
- Underneath, you can see in the same column how much of a test target could be covered.

Example

- In the *design configuration* `3xNR2`, all test cases totaled covered 95% of the required test objectives.
- 95% correspond to **37** covered test targets from the total of **39** required test targets. The 39 required test targets consisted of **9** individual *requirements*, **25** different statecharts, **1** *sequence rul*e and **4** different *variants*.
- Each individual *requirement* was covered at least once in the course of all test cases, so the coverage for the requirements group corresponds to 100%.

If you want to view all *design configurations* side by side, the project must be active in the *Project Explorer*. If, however, only one *design configuration* should be displayed, the respective *design configuration* can be clicked in *Project Explorer.*

(2)

- On the right hand side, the percentage coverage of a specific test target is shown for the individual test cases. A column is displayed for each generated test case. The percentages appear when the mouse pointer is placed on the group name of the test target (eg *Requirements*).
- If you open all the individual test targets of a group, you can see on the basis of the crosses which test target was covered explicitly in which test case (`X` is `covered`).

Example

- In test case 1, 67% of all required *requirements* were covered.
- The second *requirement* (`If, while showing a traffic sign as outdated, a new traffic sign is recognized this must be shown as highlighted.`) is only covered in the 3rd and 9th test cases.

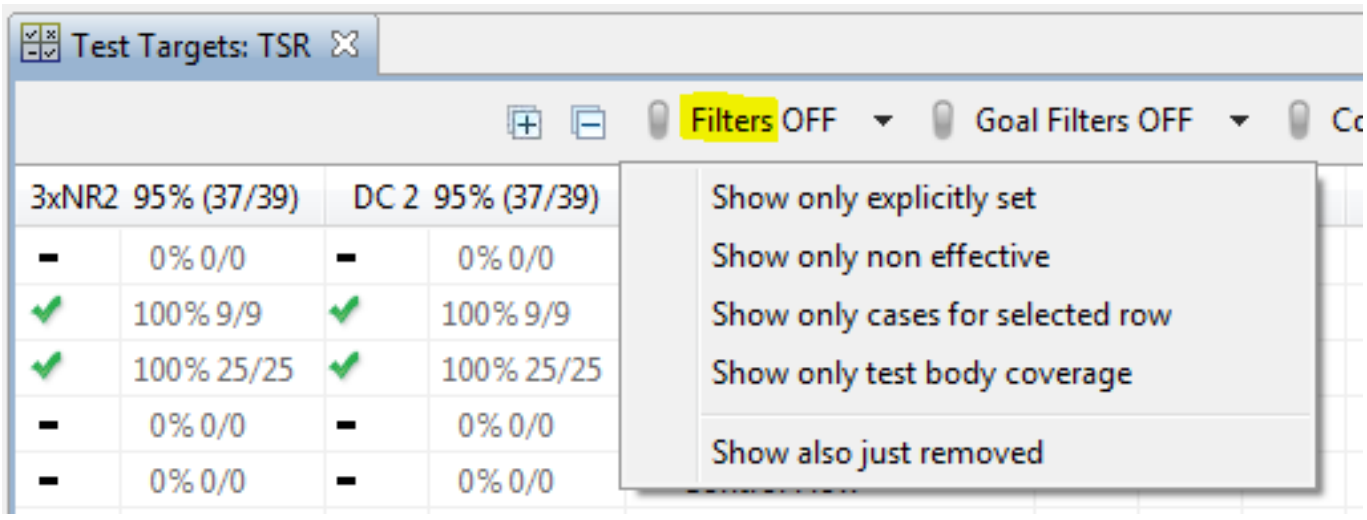
Filter

The view of the test targets can be filtered according to different criteria:

- **Filters**: General filters such as. explicitly configured test targets by the user
- **Goal Filters**: Filter according to test objectives
- **Coverage Filters**: Filter according to coverage criteria

The most important filters are explained in more detail below.

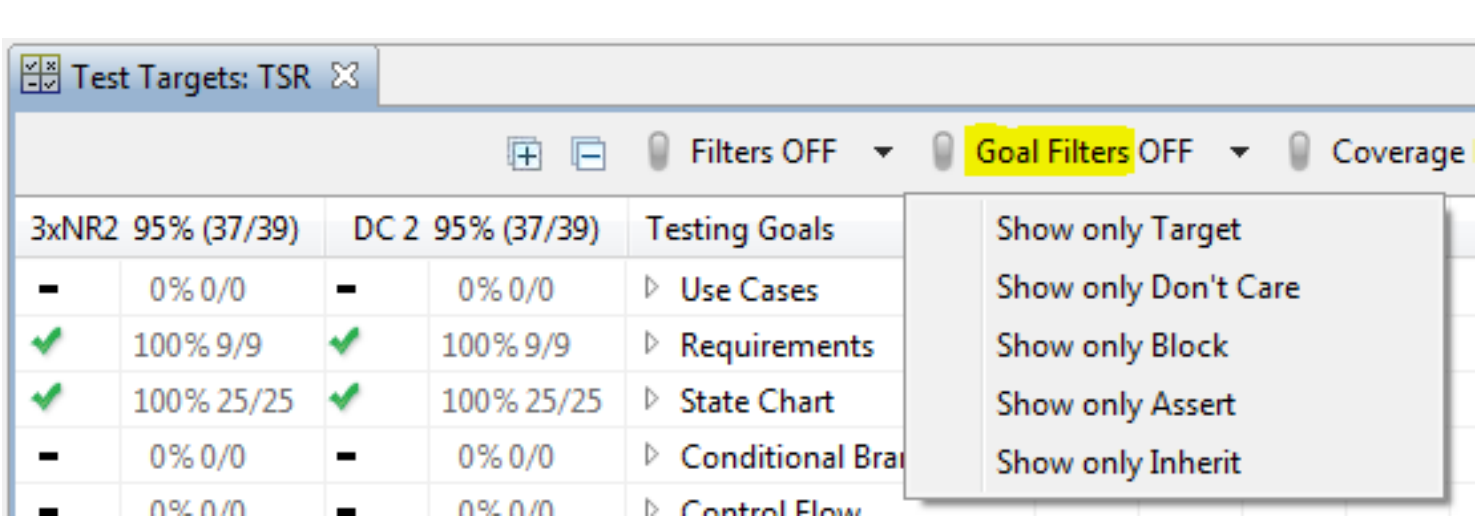
Filters



Test Target View - Filters

- *Show only explicitly set*: Only those test targets are displayed whose test target property was explicitly set (all test targets that do not have a gray symbol)
- *Show only cases for selected row*: Only the selected row is displayed

Goal Filters

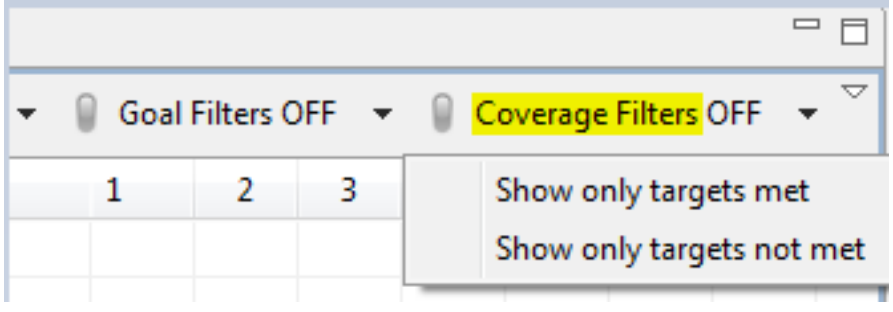


Test Target View - Goal Filters

- *Show only Target*: Only those test targets that are marked with ✔ are displayed
- *Show only Don't Care*: Only those test targets that are marked with - are displayed
- *Show only Block*: Only those test targets that are marked with ✘ are displayed
- *Show only Assert*: Only those test targets that are marked with ϟ are displayed

- *Show only Inherit*: Only those test targets are displayed that inherit the test target property from another group

Coverage Filters



Test Target View - Coverage Filters

- *Show only targets met*: Only those test targets that were covered by at least one test case are displayed
- *Show only targets not met*: Only those test targets are displayed that were not covered by a test case

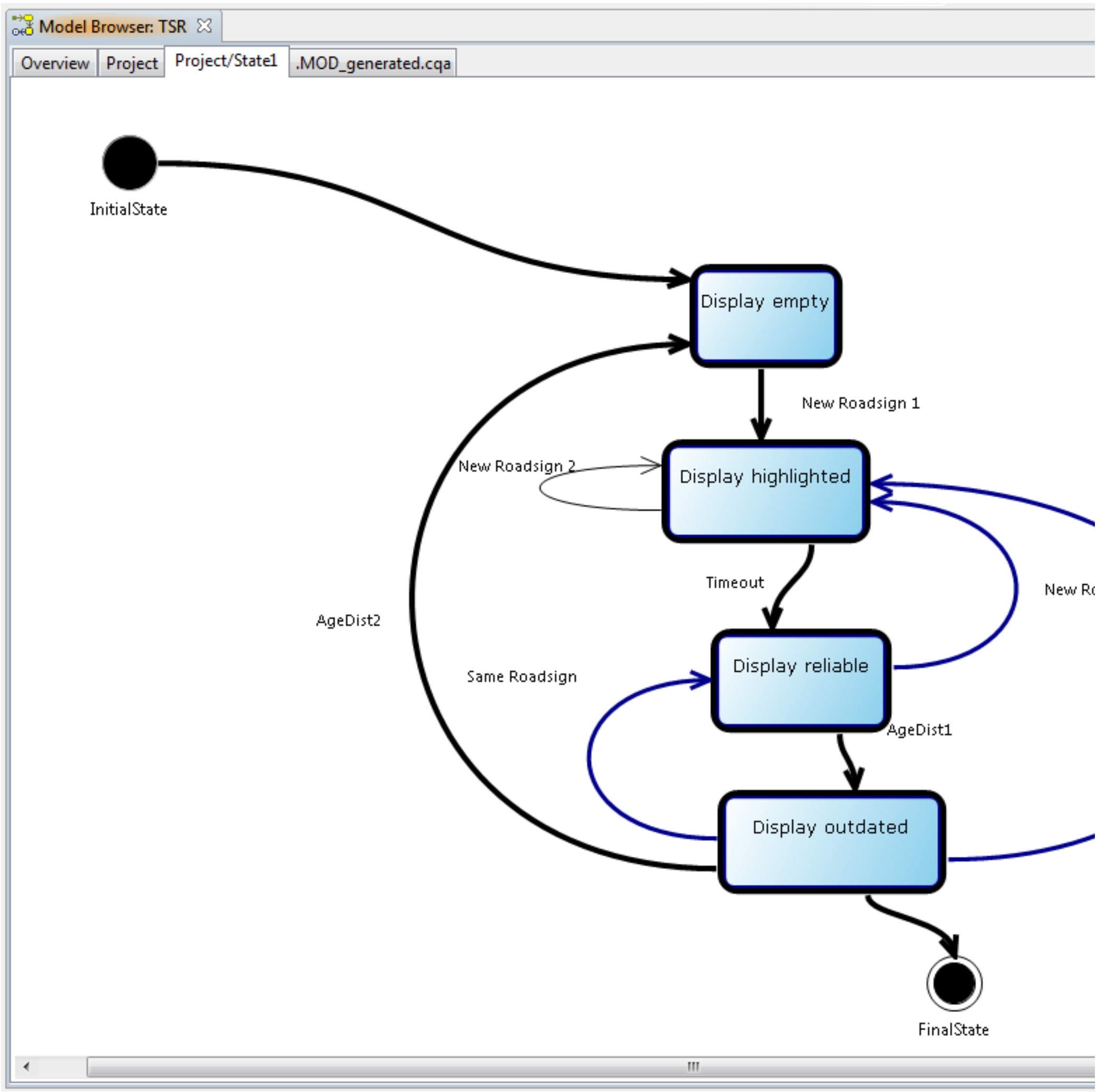
Model Browser (en)

The *Model Browser View* is used to graphically visualize a single test case in the state diagram.

If a test case is selected in the *Test Cases List*, it is automatically activated in the *Model Browser*.

All *states* and *transitions* that this test case has passed are marked **bold black** in the *Model Browser View*.

With the help of the tabs at the top, you can switch between the different *substates* and an overview view for the entire model.



Model Browser View

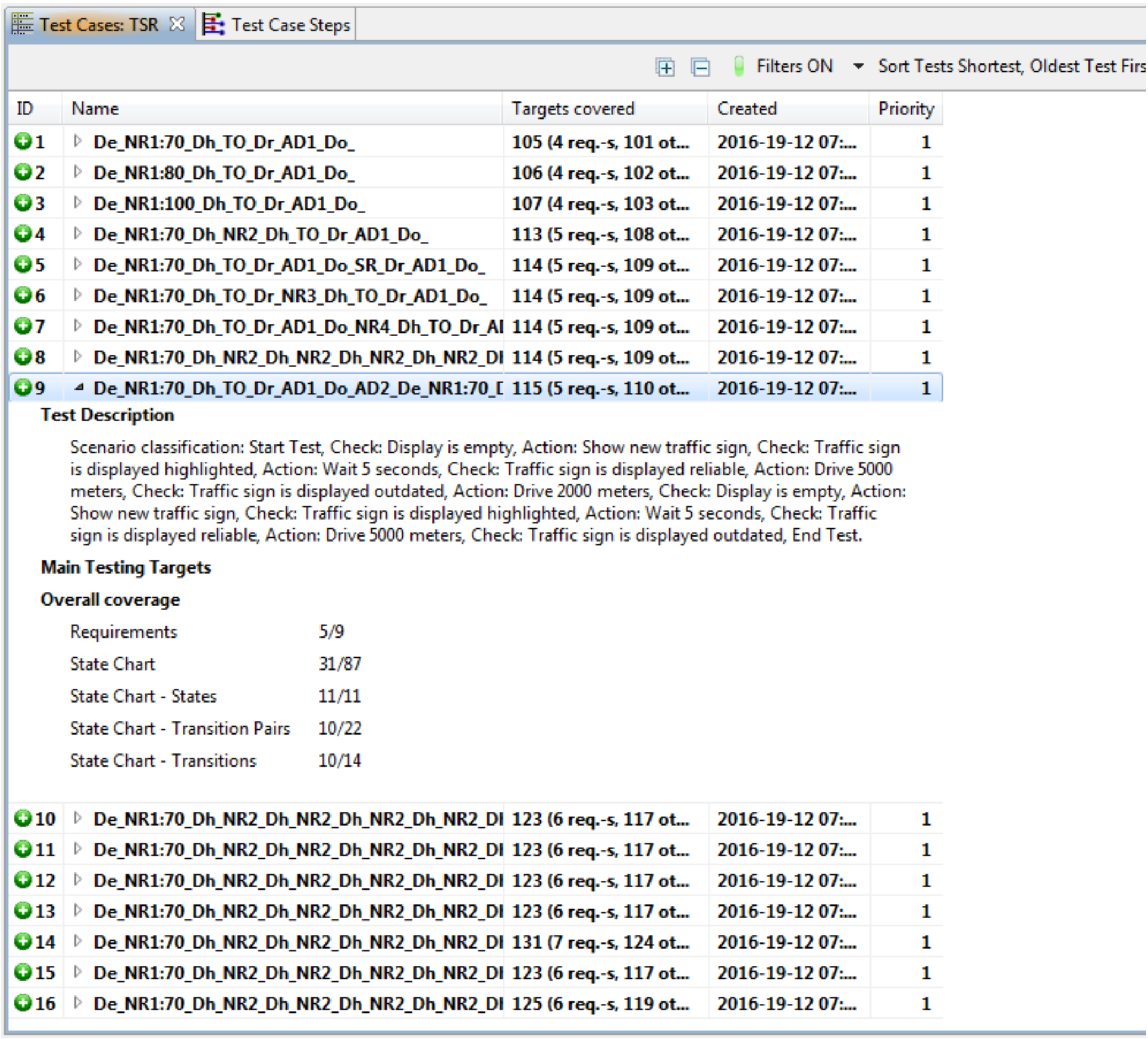
Test Cases (en)

In the *Test Cases View*, all generated test cases are listed.

Each test case can be opened, so you can see the test case description.

Depending on what is selected in the *Project Explorer*, different test cases are displayed in the *Test Case View*:

- **Project** marked: all generated test cases are displayed
- A **design configuration** is selected: only the test cases belonging to the design configuration are displayed



Filtering and Sorting

To sort the test cases, click on the *Filters* button to switch it to ON. A further button, including a drop-down menu, will

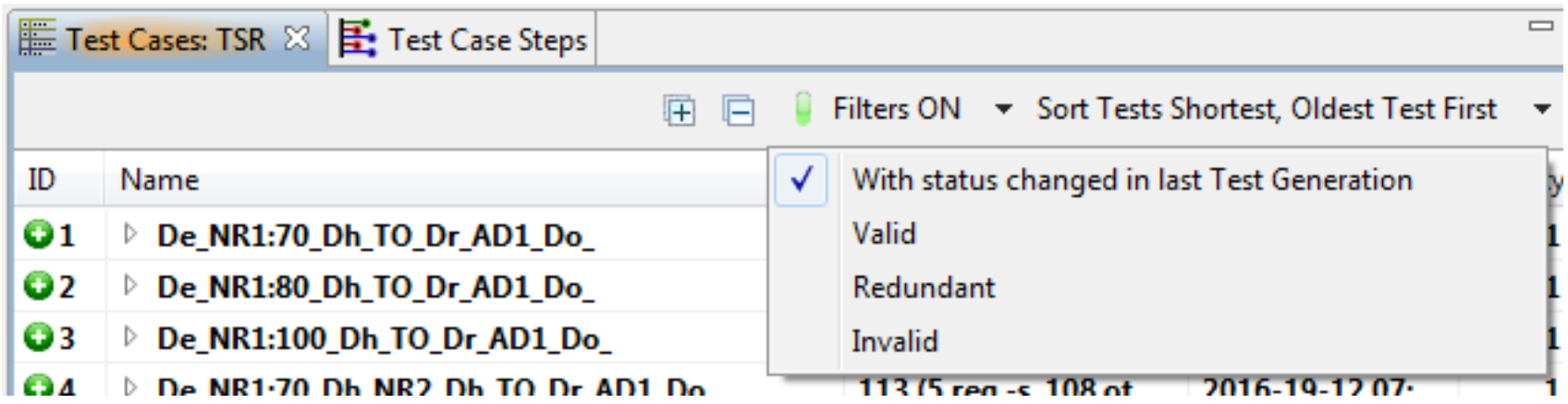
appear on the right, with which the test cases can be sorted according to various options.

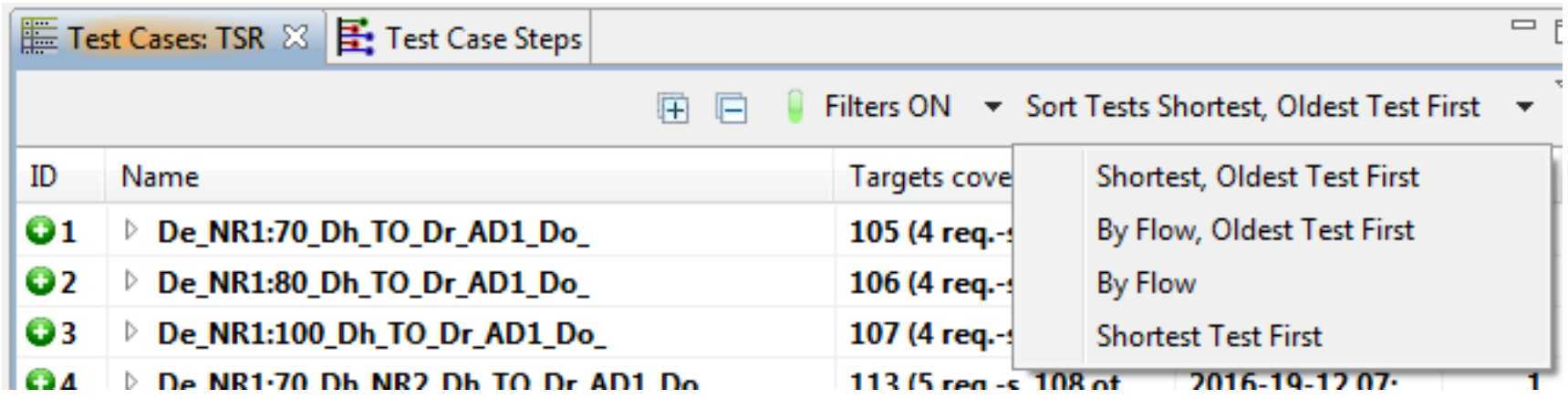
MODICA does not automatically delete old test cases if a further test case generation is carried out with an altered model. As a result, old test cases may have a different meaning in the new model.

In order to be able to check this better, the following filters are available.

Filter

- *With status changed in last Generation:* Only test cases whose current status differs from the status from the previous generation run are displayed
- *Valid:* Only test cases which are still valid after another generation run are displayed
- *Redundant:* Only test cases, which are covered by a new generation process from a new test case, are displayed
- *Invalid:* Only test cases which are no longer valid after a further generation run are displayed



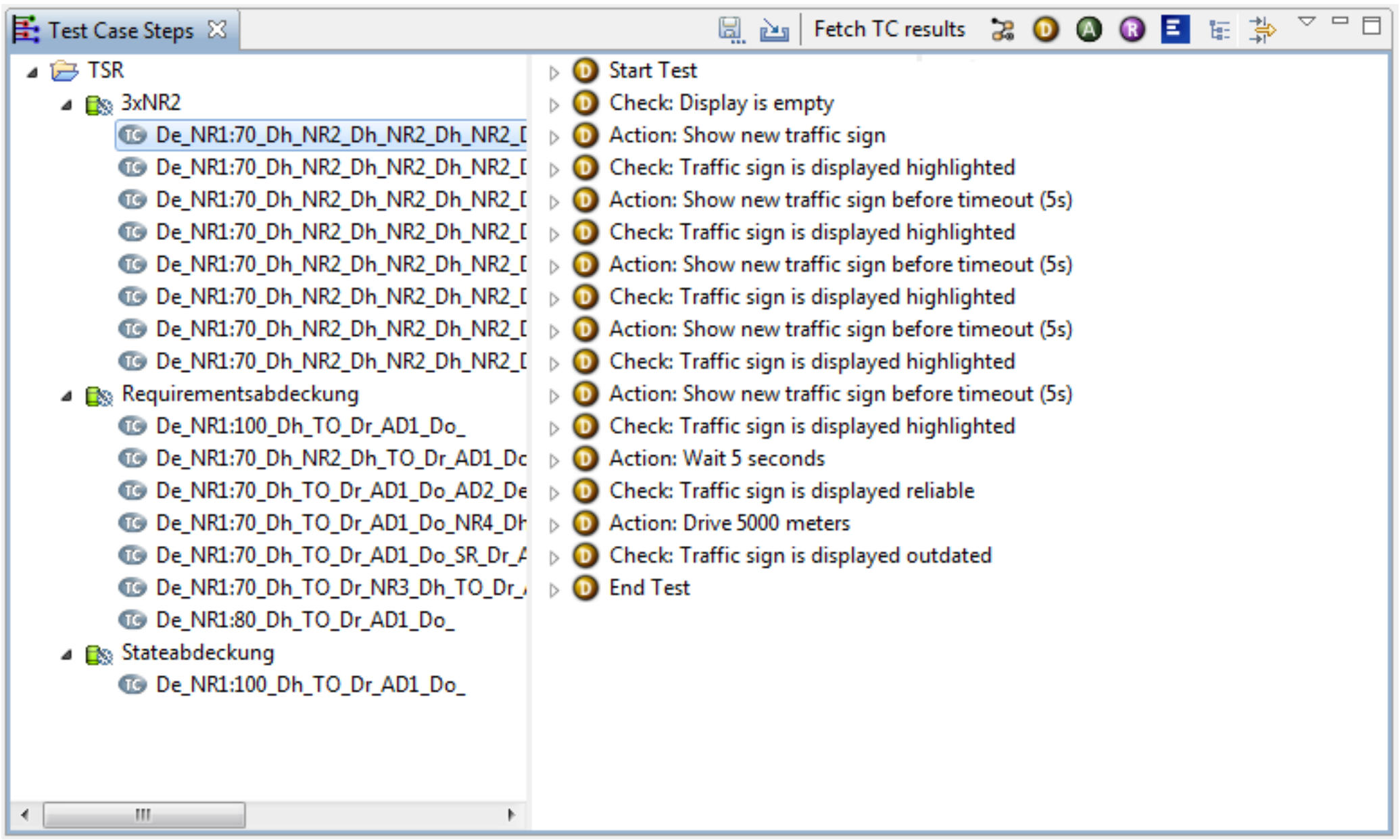


Test Case Steps (en)

The *Test Case steps Steps View* can be used to examine the individual steps of the test cases.

Here you can see all created test cases, grouped in the *design configurations* in which they were created.

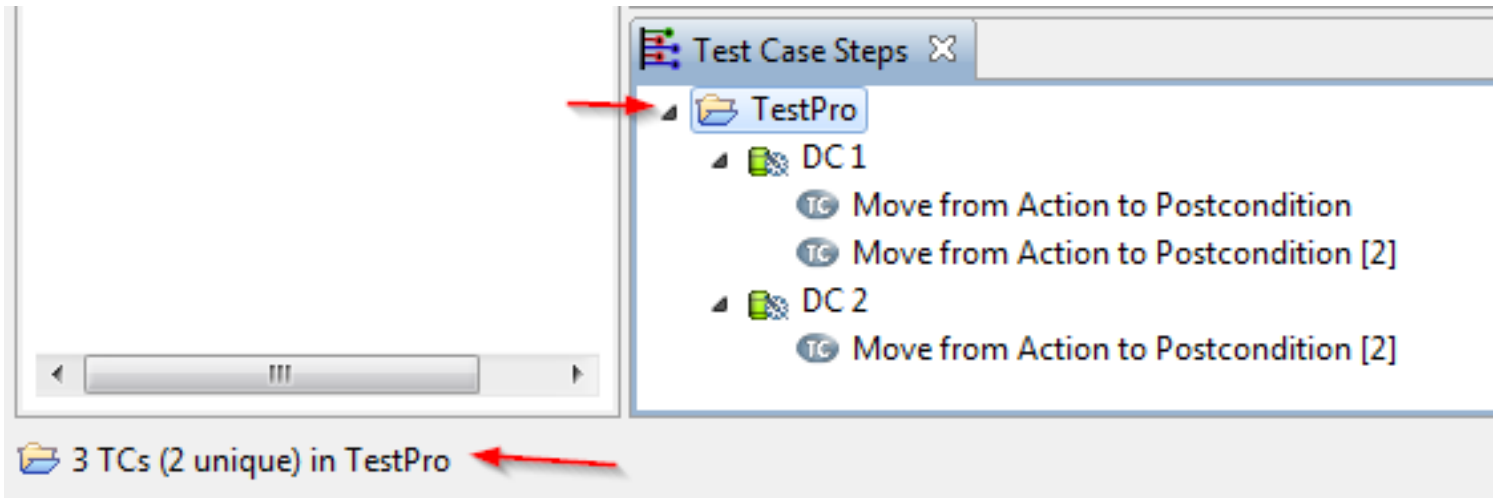
On the right side of the view, all steps of a test case are displayed as tree or list. The display mode can be toggled using a toolbar button: 



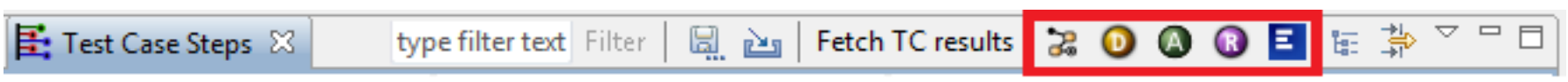
Statistics

When selecting the project or a DC on the left side of the view, some simple statistics about selection can be seen in the status bar of MODICA.

- For projects, the total count of generated tests is shown as well as how many of those are unique (the same test can appear in more than one DC)
- For DCs, the number of Tests is shown. (A DC cannot contain any duplicates)



Filtering steps



The colored buttons in the upper right corner of the view allow you to quickly filter the view. When selected, certain types of test case steps can be hidden.
The available filters are:

- Structural elements (states and transitionen)
- Descriptions
- Requirements
- Attribut modifications
- Elements of the test automation environment (here: EXAM)

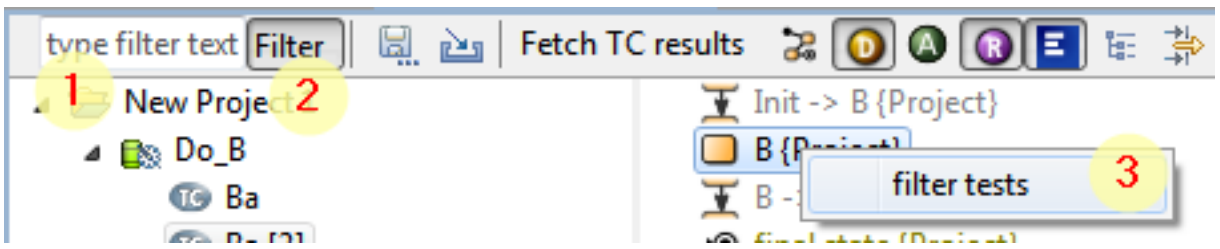
Additional toolbar items are available:

- To configure other, less used filters
- Allows switching to a tree based view of the test steps (reflecting hierarchies in the diagrams - not available for parallel statemachines)

In the screenshot, all elements except of the *descriptions* were filtered. Now a pure description of the test case is displayed in the *Test case steps view*.

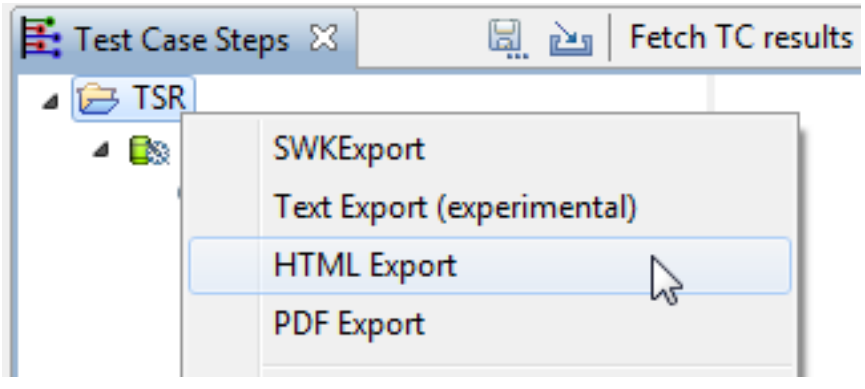
Filtering test cases

The list of test cases can be filtered using a quick-filter based on the test name using the quick-filter (1). Besides this, it is possible to select a structural step (state/transition) from the list of tests steps to only display tests that contain the step (3). Both kinds of filters can be cleared by clicking on the Filter button in the toolbar (2):



HTML-Export

By right-clicking on the project or a *design configuration*, the test cases can be exported to an HTML format.



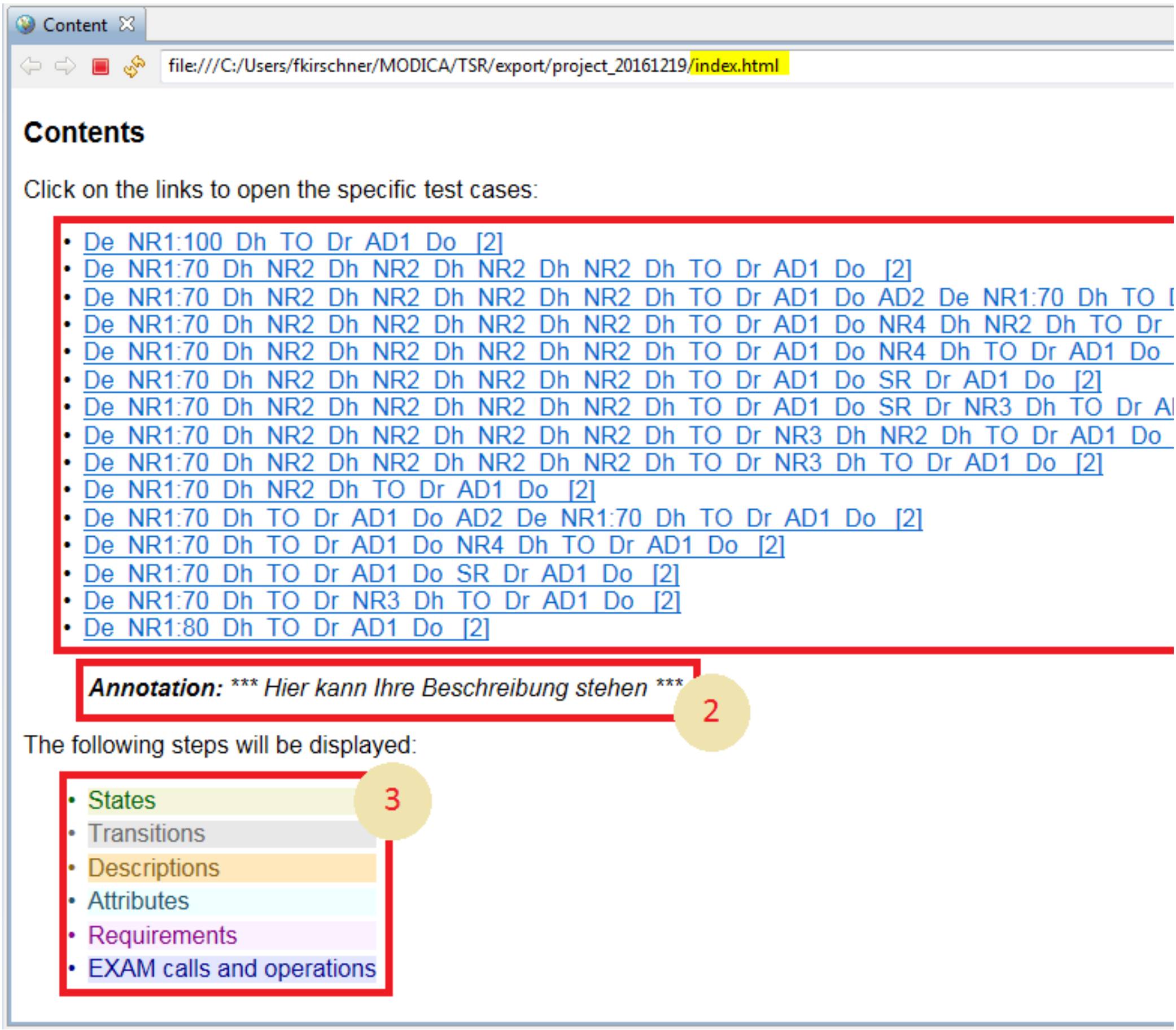
In the following dialog you can make some settings:

- *Destination Folder:* all exports are stored in the workspace in the export folder. The name of the new subfolder can be defined here. By default, when clicking on a project, it has the name `project_time stamp`, when clicking on a *design configuration* it has the name `name-of-the-design-configuration_time stamp`.
- *Select Filters:* Items activated here are NOT displayed in the HTML export. The symbols have the same meaning as the previously described filters in the *Test Case Steps View*. refers to all elements of the used test execution environment
- *Description*: add a annoation which will be later displayed in the index-file
- *Export to one document*: By default, a document is created for each test case. If this is not desired, enable this option and the program creates a single document in which all test cases are listed vertically.

⚠ For experts

Click to copy templates to project for manual override: The HTML export is based on templates. If the HTML document is to be modified, these templates can be adapted manually. By clicking on the link, all templates are loaded into the project directory under *ExportTemplates*, where the user can now make manual changes, for example, of the css file. Variables that are later replaced in the program by data are embedded between dollar and paragraph signs ($ and §). These variable names, including the encapsulation symbols, must not be deleted or altered, as otherwise a fault-free generation of the HTML document can no longer be guaranteed.

The *index* file contains links to the html pages of each test case (1), the description (2), if applicable, and highlights which elements are displayed in which color on the individual test pages (3).



By clicking on one of the links you get to the corresponding test case.

file:///C:/Users/fkirschner/MODICA/TSR/export/project_20170405/tc_De_NR1100_Dh_TO_Dr_AD1_Do_.html

De_NR1:100_Dh_TO_Dr_AD1_Do_ [2]

Transition	Action
transition (from *InitialState* to *Precondition*)	
Testsetup ready (from *Precondition* to *State1*)	
transition (from *InitialState* to *Display empty*)	deleted requirement
	When the System starts, the display may not show a traffic sign (empty display)
	Start Test
	Check: Display is empty
New Roadsign 1 (from *Display empty* to *Display highlighted*)	Adaptive Cruise Control (ACC) is an optional cruise control system for road vehicles that automatically adjusts the vehicle speed to maintain a safe distance from vehicles ahead.
	If no traffic sign is shown and a new traffic sign is recognized the new traffic sign must be shown as highlighted
	Action: Show new traffic sign
	Check: Traffic sign is displayed highlighted
Timeout (from *Display highlighted* to *Display reliable*)	When showing as highlighted and after five seconds the traffic sign must be shown as normal (reliable, no longer highlighted)
	Action: Wait 5 seconds
	MessHandler_STH_AU736_AU49X:
	Argument: onlyParameterSet; Value: ; Type: ParameterSet
	Check: Traffic sign is displayed reliable
AgeDist1 (from *Display reliable* to *Display outdated*)	Action: Drive 5000 meters
	When showing as reliable and after passing the configured distance (Aging Distance 1) the traffic sign must be shown as outdated
	Check: Traffic sign is displayed outdated
transition (from *Display outdated* to *FinalState*)	End Test
Test finished (from *FinalState* to *Postcondition*)	
transition (from *Postcondition* to *FinalState*)	

PDF-Export

Preconditions

The PDF export requires an external program for converting HTML to PDF, such as **wkhtmltopdf**

At the beginning, the path to the exe file of the converter program must be specified under *Window → Preferences → PDF-Export.*

Optionally, additional parameters can be specified to customize the output.

A converter programm only works if it can be called via the command line in the given order: `Path_to_exe [Parameter]* input-file output-file`

Export

By right-clicking on the project or a design configuration, the test cases can be exported to a PDF format.

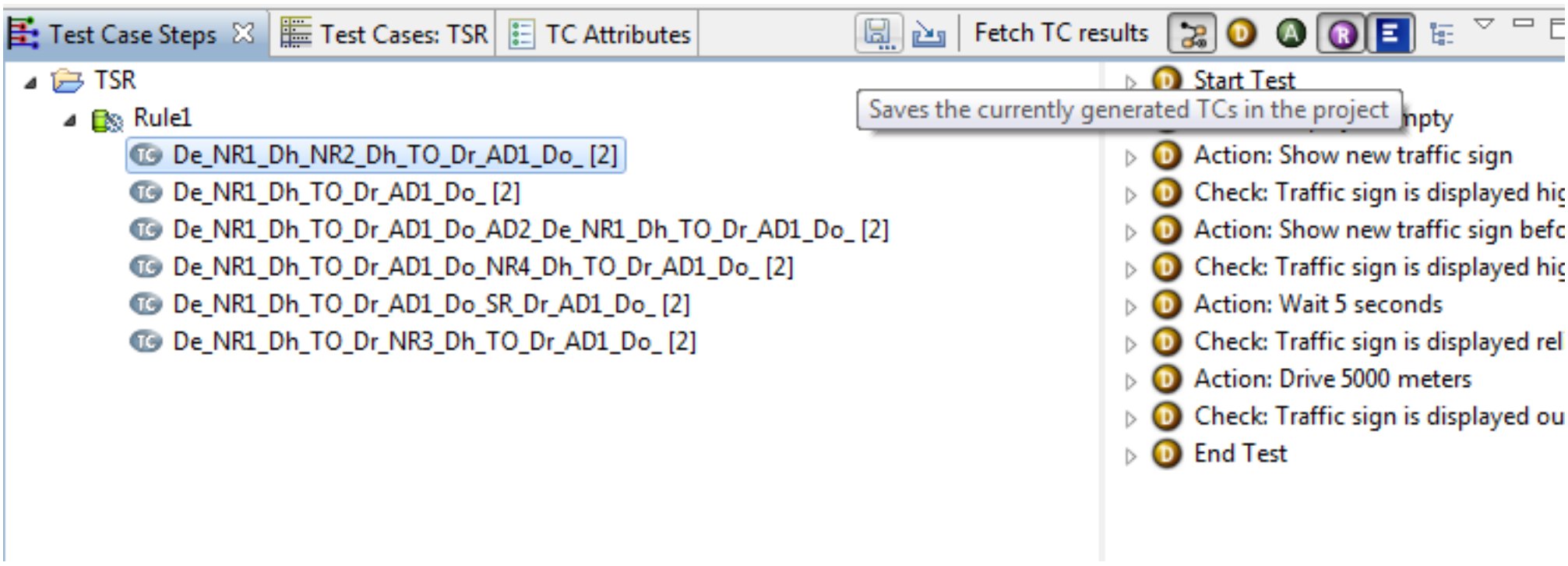
The settings in the dialog are similar to the HTML export (see above).

The test cases are saved in the project directory under *export → pdf.*

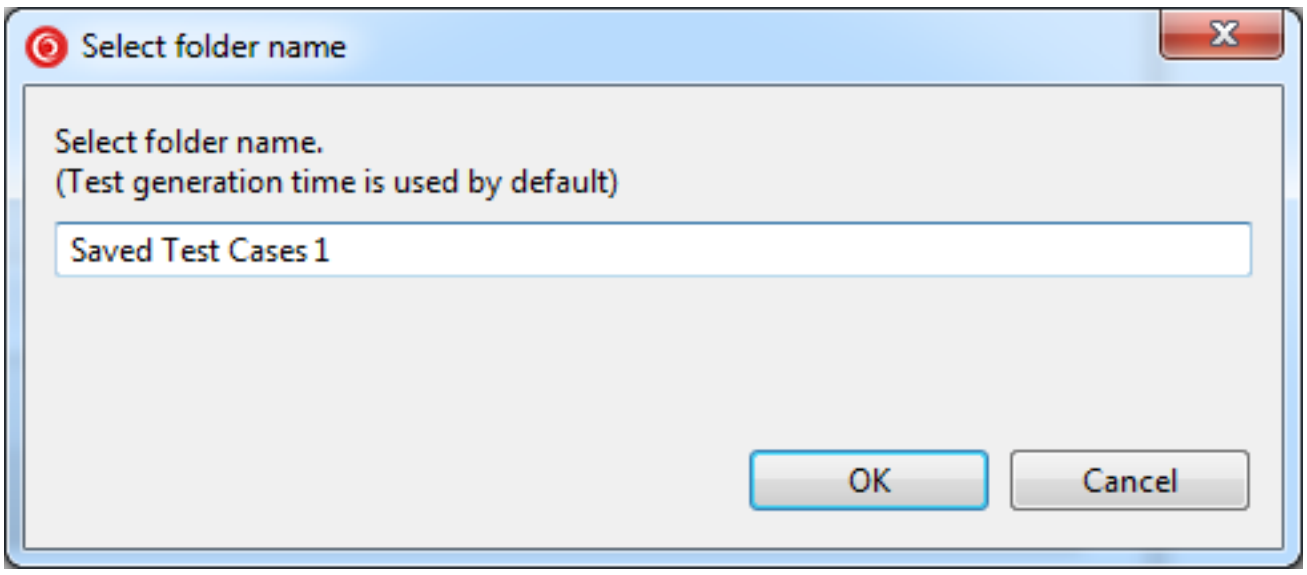
Saving Test Cases

Generated Test Cases can be saved in MODICA. The goal is to allow the user to review previously generated test cases in MODICA. Also, test results from the test execution environment can be linked to these test cases.

In the Test Case Steps View there is a toolbar button

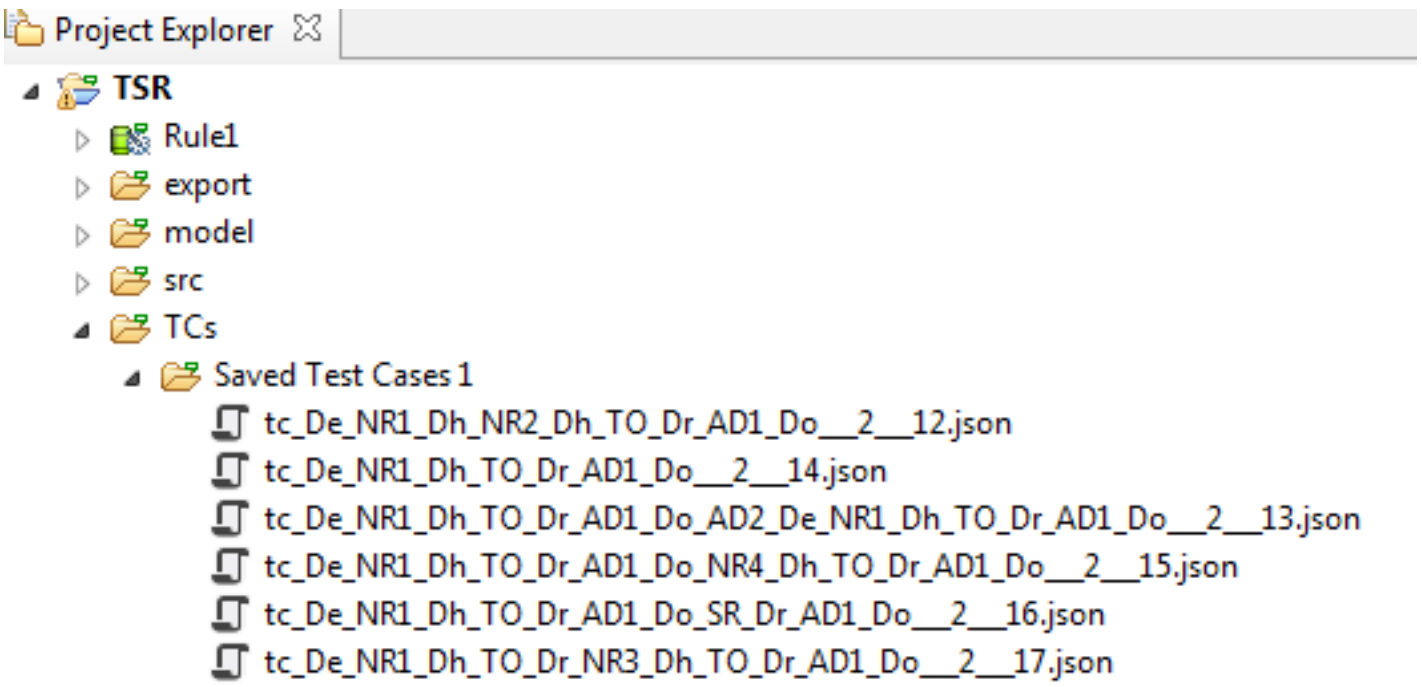


Saving Test Cases



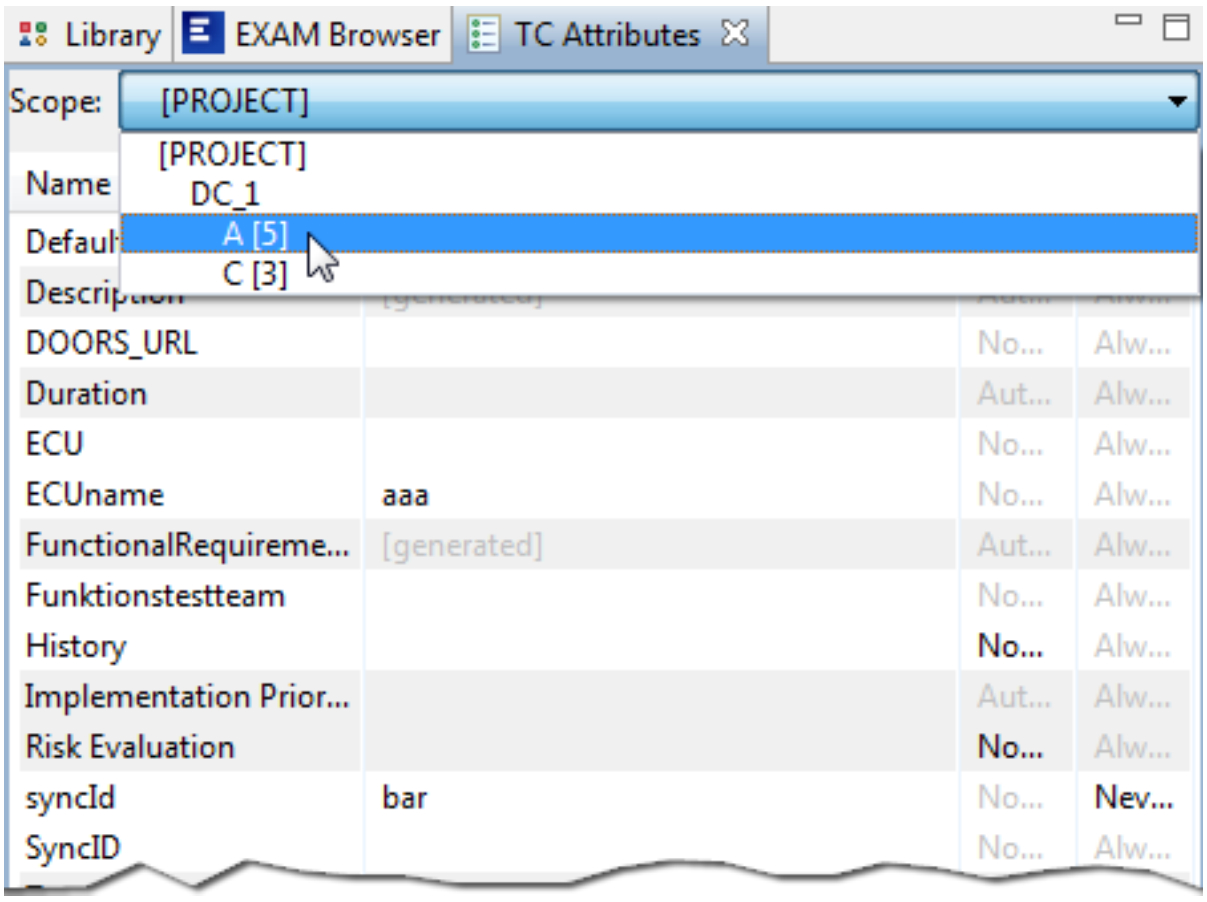
Save Test Case Dialog

The test cases are stored in the Project Explorer in the folder TCs:



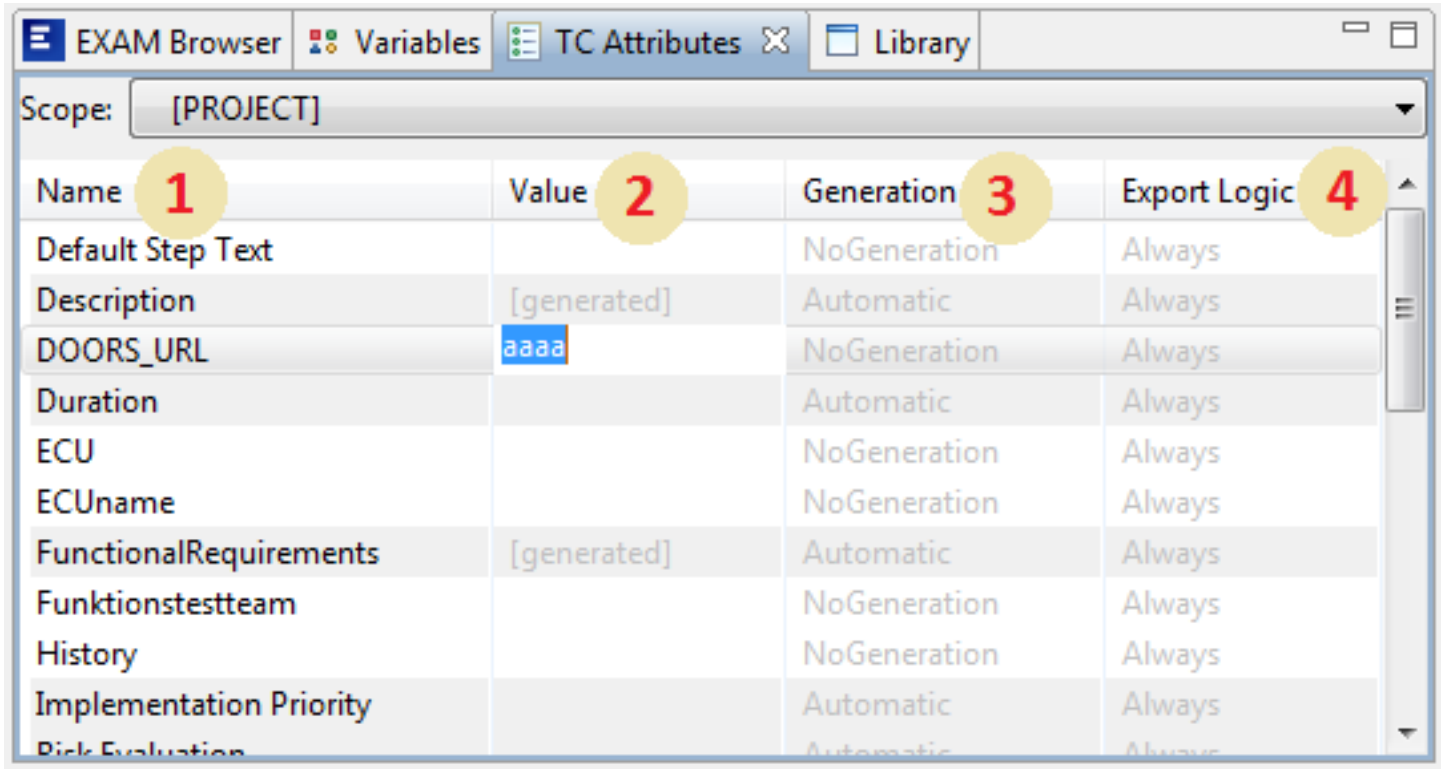
TC Attributes View (en)

The *TC Attributes View* is used to configure test case attributes such as *DefaultStepText, History* and others. In addition, automatically generated attributes can be viewed without a full export of the test cases into a target system.
The View offers a hierarchical overview of the project, the included *design configurations* and their test cases. In the following screenshot, for example, the test cases `A[5]` and `C[3]` in the *Design Configuration* `DC_1`.



The selection of an entry determines the level at which the attributes are displayed or edited in the main part of the view. The project is always selected by default. Attributes are therefore pre-assigned for the entire project. If you choose a more concrete level, such as *Design Configuration* or a specific test case, values can also be predefined for these levels.

In the attribute list, global attributes like *TestCaseName* and attributes which are defined using a *TestCase Stereotype* in a concrete test automatisation model appear. The list can be manually (re-)loaded from associated servers (e.g. EXAM) by clicking on the *Load/Refresh definitions* button at the bottom of the view. The names of the attributes are listed in the column *Name* (1). Some of the attributes are generated automatically or are not to be edited by MODICA (gray background). For the other attributes, a value can be stored in the *Value* (2) column:



To the right of the *Value* column there are two more columns *GenerationStrategy* (3) and *ExportLogic* (4). The entries in

these columns can only be changed at the project level.
The *Export Logic* column specifies whether

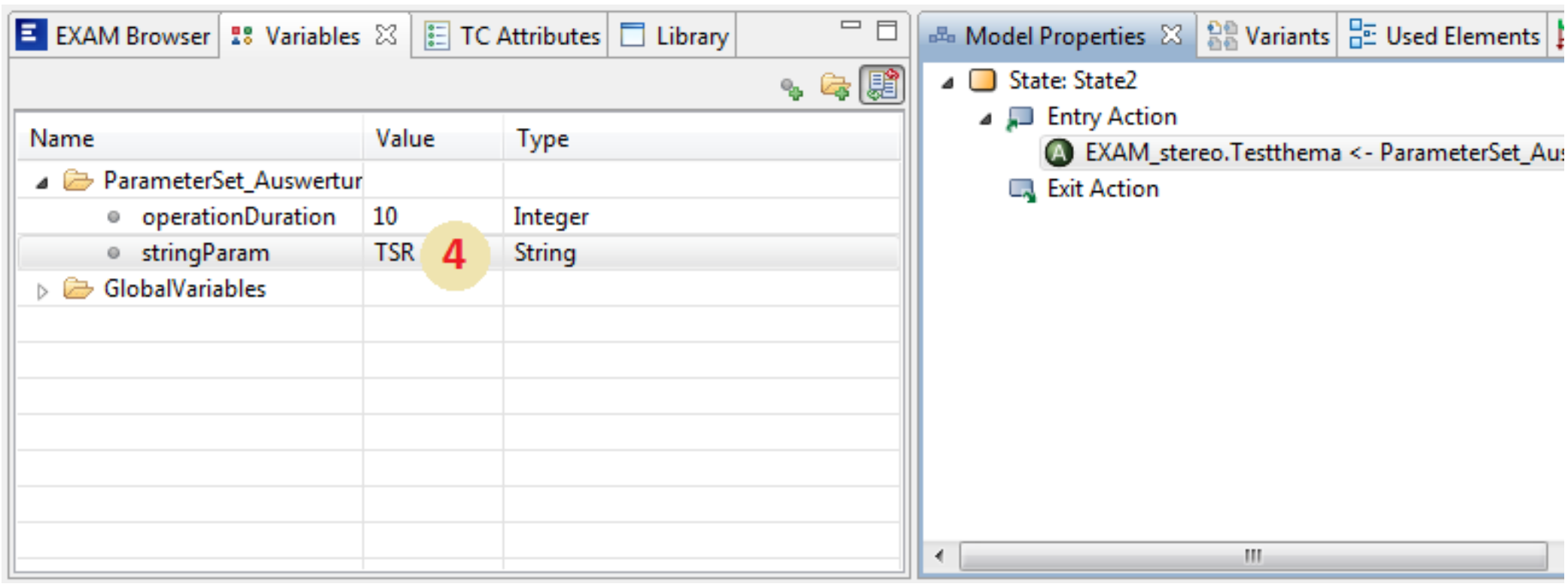
- The value should be written for each export (*Always*)
- The value should be written only in new test cases (*IfNew*)
- The value should not be changed (*Never*)

For some of the special fields like *UUID*, the selection in this column is not relevant.
The *Generation Strategy* column specifies whether

- The value of the attribute is static (*NoGeneration*)
- The value of the attribute should be generated from the model (*FirstValue*Only, *LastValueOnly, Concatenate, SetConcatenate*)
- The value of the attribute is subject to a non-variable logic (*Automatic*, Not selectable by the user)

If one of the four options *FirstValueOnly, LastValueOnly, Concatenate* or *SetConcatenate* is selected, the value of the attribute is generated dynamically for each test case through the model.

- To mark places in the model, *Modification Elements* can be inserted in the *Model Properties View* by clicking on (1).



- (2): select the attribute to be generated
- At (3), a global variable can be selected from the Variables View (4) as a source for the value, while (5) allows a static string to be specified as a value.

What happens with these values now depends on the selected generation strategy:

- *FirstValueOnly:* Only the modification occurring first in the test case is taken into account.
- *LastValueOnly*: Only the modification occurring last in the test case is taken into account.
- *Concatenate*: Multiple values for the same attribute are appended to each other and line breaks are inserted between them.
- *SetConcatenate*: Multiple values for the same attribute are appended to each other and line breaks are inserted between them - duplicates are deleted.

In the above scenario, all generation strategies would behave the same and set the ACC value in the *Stereotype Attribute* `Testthema` as soon as State1 is entered.

Attention: If *NoGeneration* is selected as the generation strategy, *Attribute Modifications* for this attribute are not taken into account!

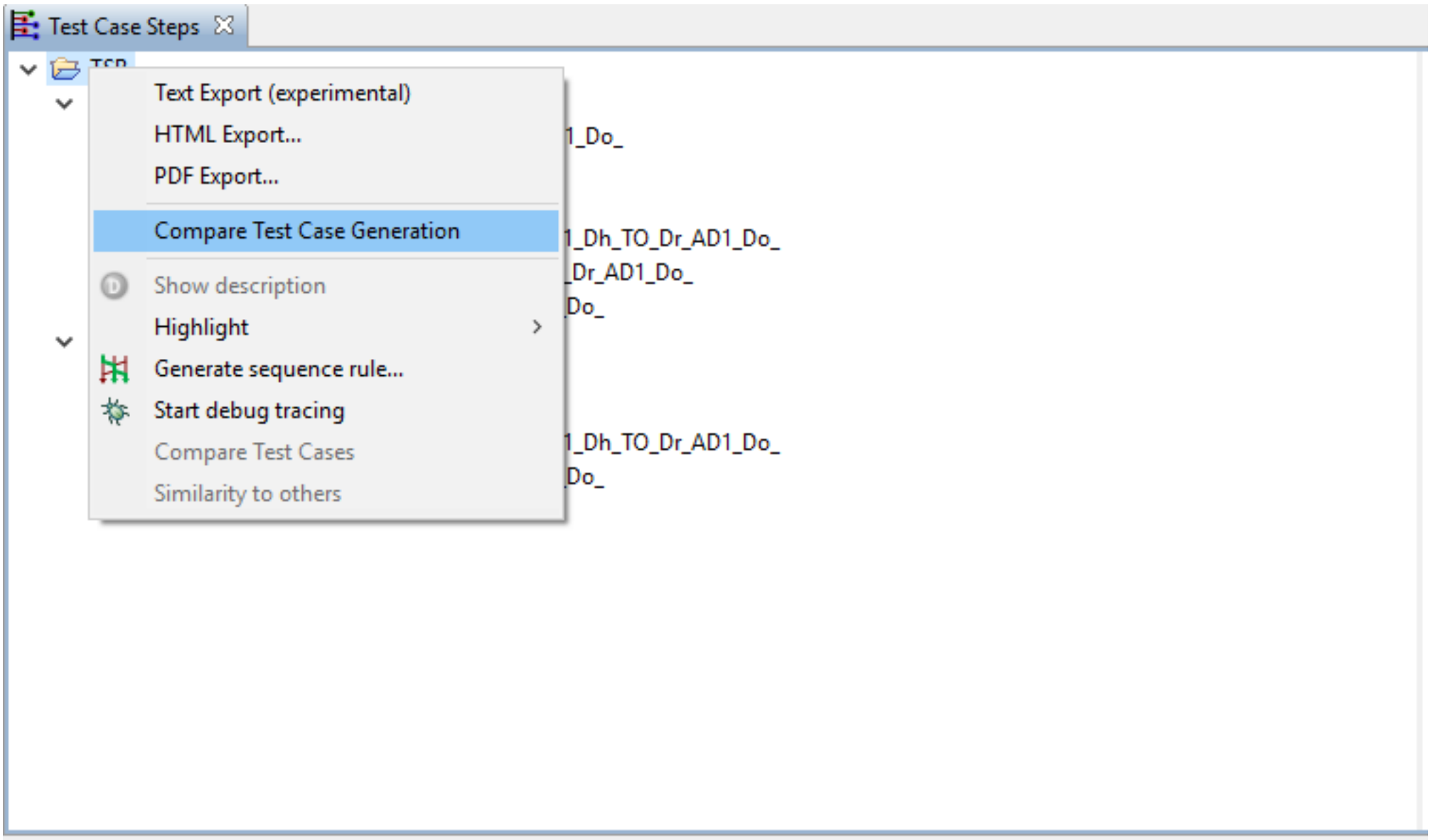
Analysis Perspective

This perspective is for analyse genereted testcases and obtained coverage.

Compare Test Generation Dialog (en)

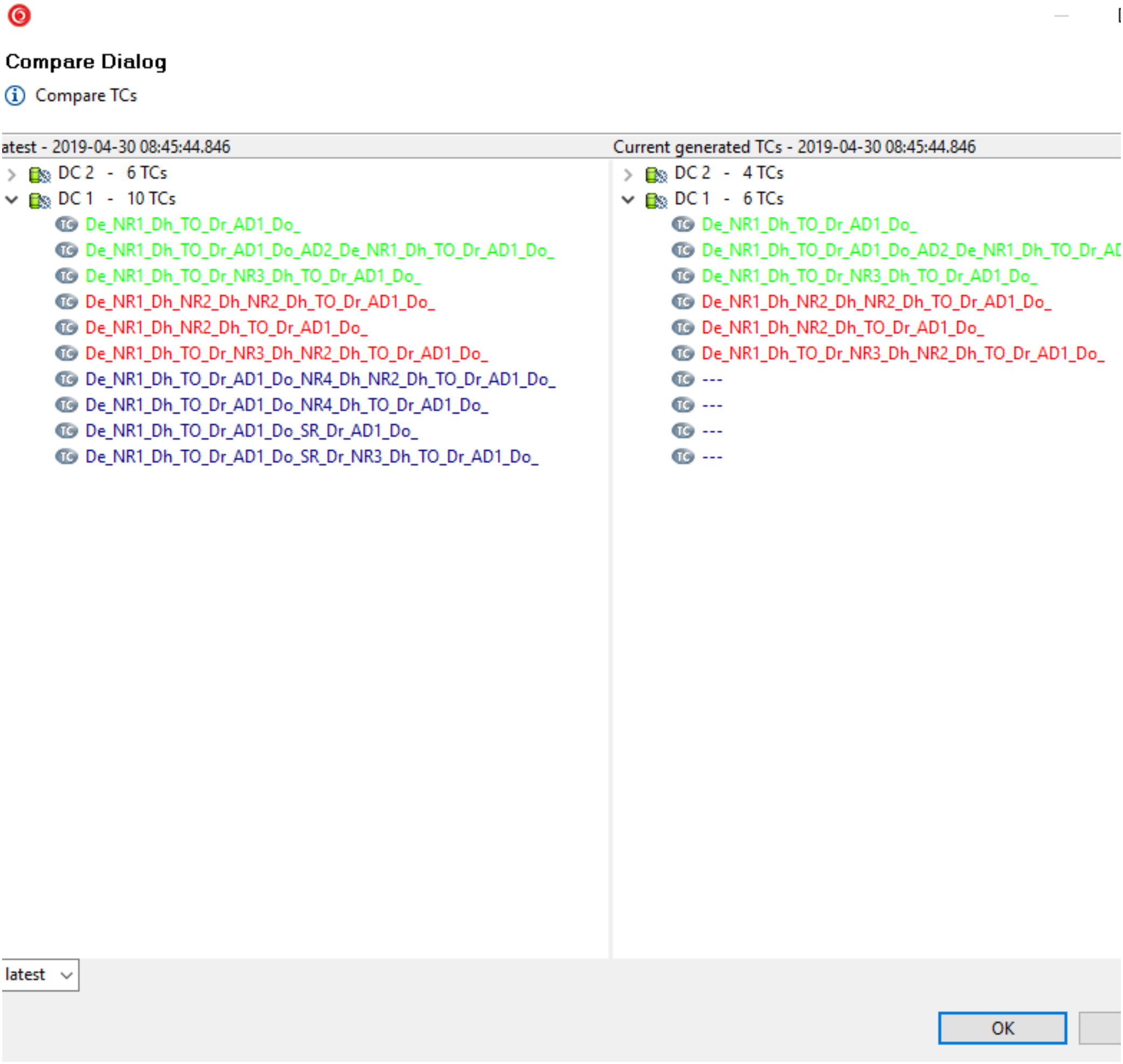
The dialog will be shown at the end of the test case generation and will be used for comparison with previous test case generation. The user has the opportunity to see the differences of the test cases compared to the last generation or a selected set of generated test cases.

The compare dialog can also be opened via the context menu of the project in the Test Case Steps View.

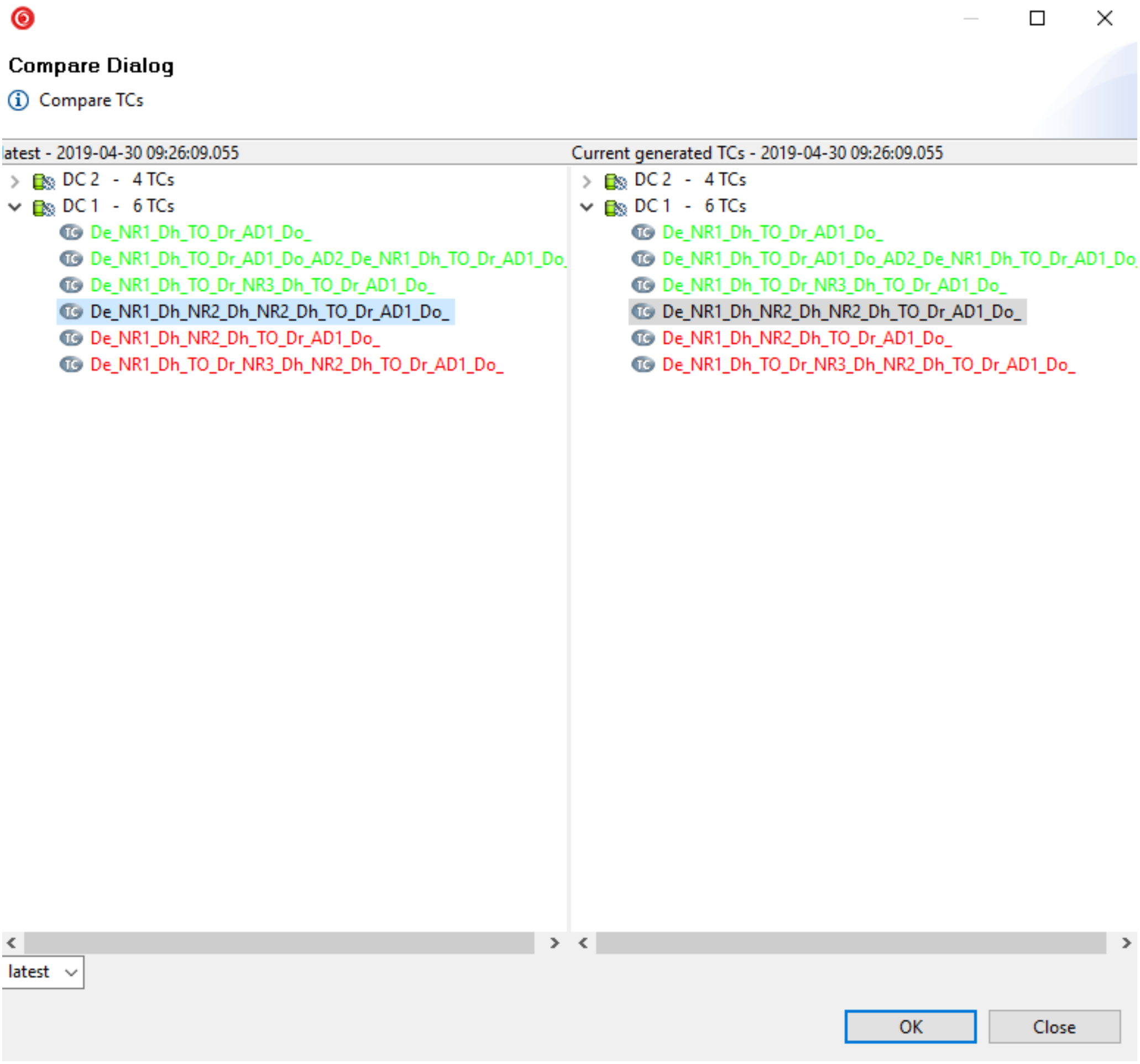


After the dialog has been opened and the comparison has been executed, test cases will be colored accordingly**:**

- Greed→ Test cases are the same
- Red→ Test cases have the same name but differ in content
- Blue→ Test cases have been deleted or have been added



Moreover, each test case is clickable and can be compared with the opposite test case for exact differnces. Double-click on colored test case opens a comparison view and shows the differences.


Compare Dialog
Compare TCs
atest - 2019-04-30 09:26:09.055
Current generated TCs - 2019-04-30 09:26:09.055
DC 2 - 4 TCs
DC 1 - 6 TCs
De_NR1_Dh_TO_Dr_AD1_Do_
De_NR1_Dh_TO_Dr_AD1_Do_AD2_De_NR1_Dh_TO_Dr_AD1_Do_
De_NR1_Dh_TO_Dr_NR3_Dh_TO_Dr_AD1_Do_
De_NR1_Dh_NR2_Dh_NR2_Dh_TO_Dr_AD1_Do_
De_NR1_Dh_NR2_Dh_TO_Dr_AD1_Do_
De_NR1_Dh_TO_Dr_NR3_Dh_NR2_Dh_TO_Dr_AD1_Do_
latest
OK
Close

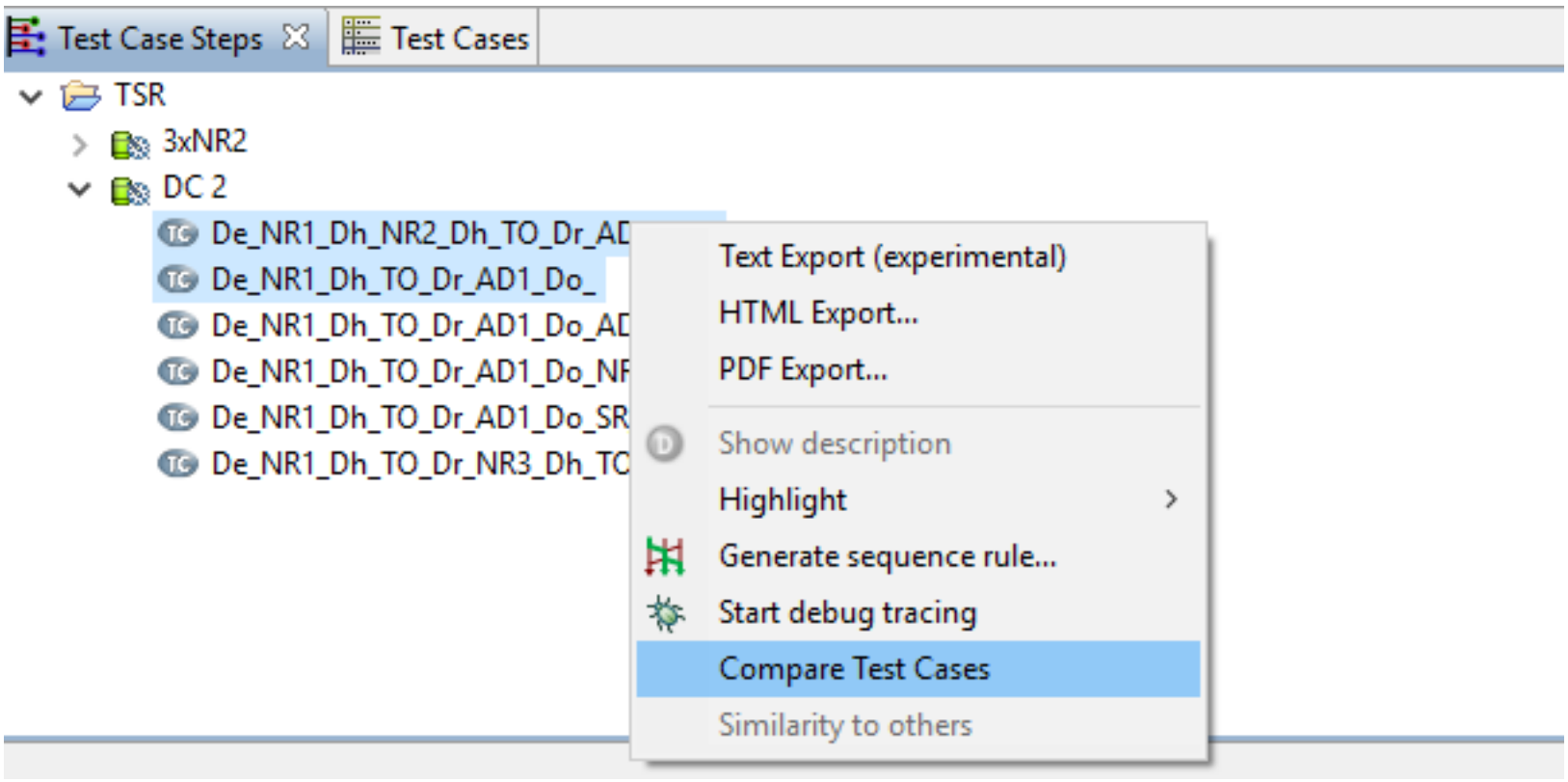
Compare Test Cases

Text Compare

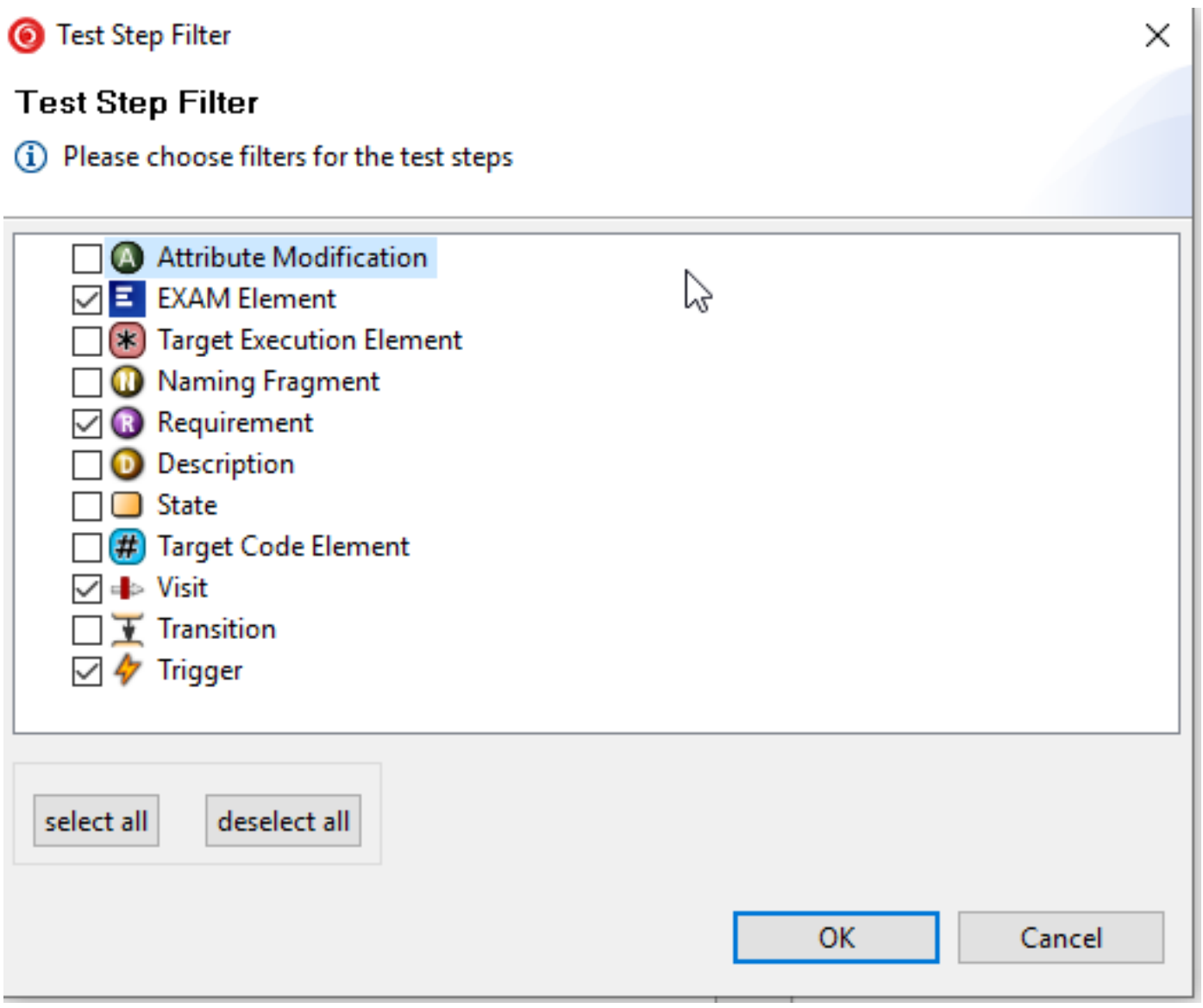
Test Case: De_NR1_Dh_NR2_Dh_NR2_Dh_TO_Dr_AD1_Do_	Test Case: De_NR1_Dh_NR2_Dh_
com.berner_mattner.modica.common.model.testcase.RequirementURLStep@63	com.berner_mattner.mo
Start Test	Start Test
Display empty {Project/Action}	Display empty {Proje
Check: Display is empty	Check: Display is emp
NamingFragmentStep: De_	NamingFragmentStep: D
Display empty {Project/Action}	Display empty {Proje
TriggerStep: New Roadsign 1	TriggerStep: New Roa
New Roadsign 1 (Display empty -> Display highlighted) {Project/Actior	New Roadsign 1 (Displ
com.berner_mattner.modica.common.model.testcase.RequirementURLStep@56	com.berner_mattner.mo
NamingFragmentStep: NR1_	NamingFragmentStep: D
Action: Show new traffic sign	Action: Show new tra
Display highlighted {Project/Action}	Display highlighted
NamingFragmentStep: Dh_	NamingFragmentStep: D
Check: Traffic sign is displayed highlighted	Check: Traffic sign
Display highlighted {Project/Action}	Display highlighted
TriggerStep: New Roadsign 2	TriggerStep: New Roa
New Roadsign 2 (Display highlighted -> Display highlighted) {Project/	New Roadsign 2 (Displ
com.berner_mattner.modica.common.model.testcase.RequirementURLStep@3f	com.berner_mattner.mo
NamingFragmentStep: NR2_	NamingFragmentStep: D
Action: Show new traffic sign before timeou (5s)	Action: Show new tra
Display highlighted {Project/Action}	Display highlighted

Compare View (en)

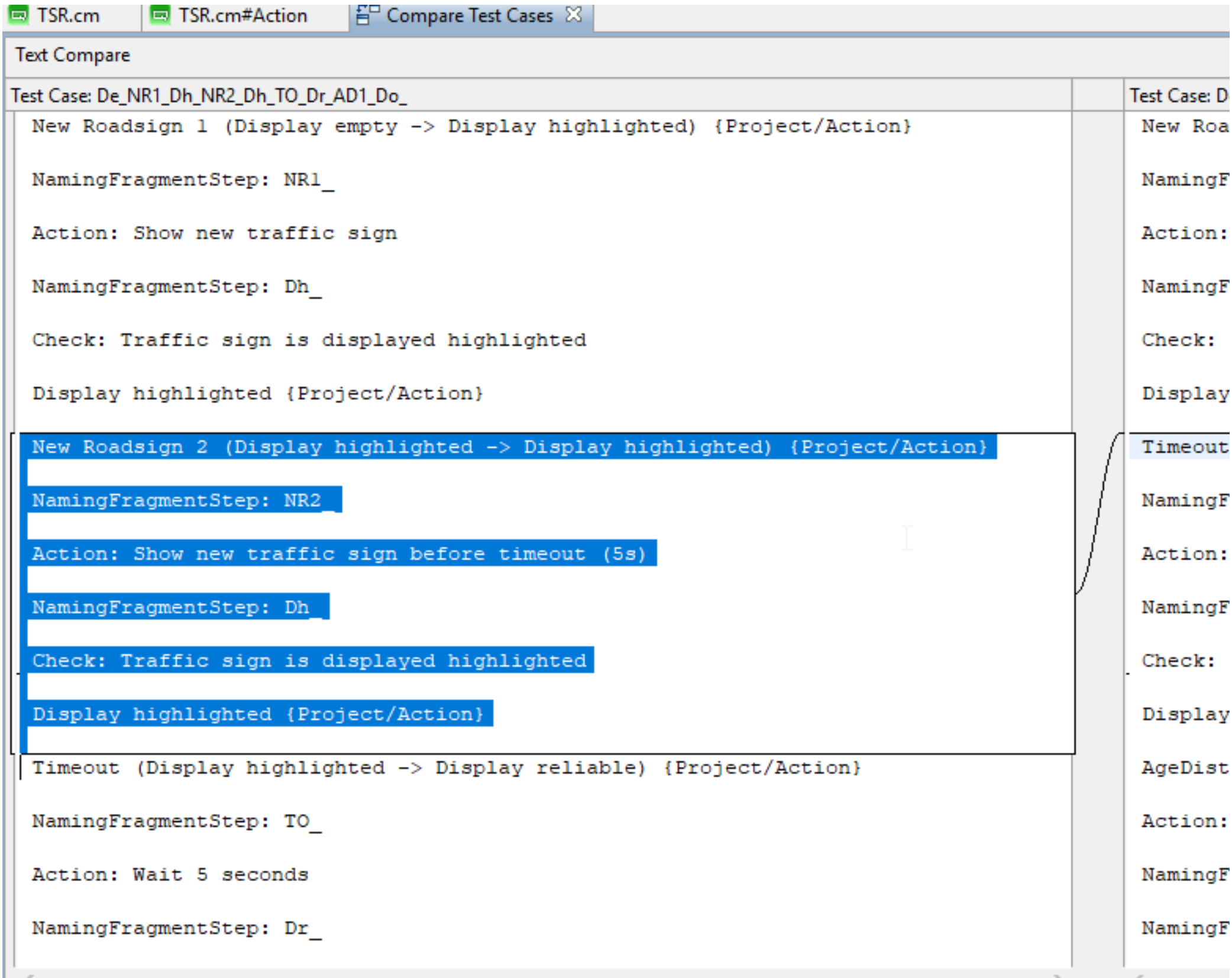
This view shows the user the differences between **two** test cases. Before a comparison can be started, there is the possibility to select the filters to specify the comparison.



The filter selection.



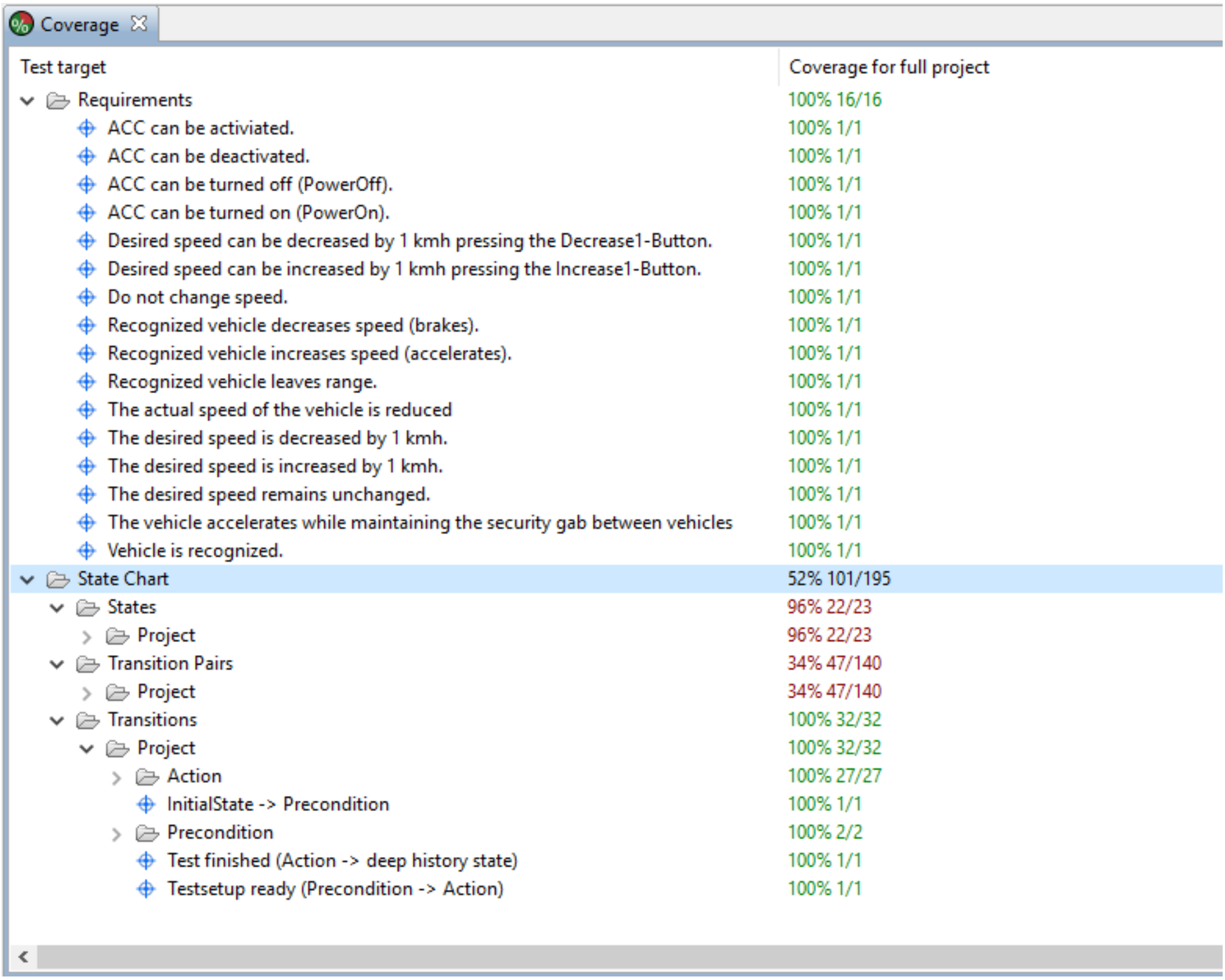
The Compare View consists of two views, where the test steps to be compared are displayed. If there is a difference, the position is marked and also shown where it could belong.



Coverage View (en)

After the creation of test cases, the coverage view shows which part of test goals (for example: Requirements, States, ...) were covered by the test cases.

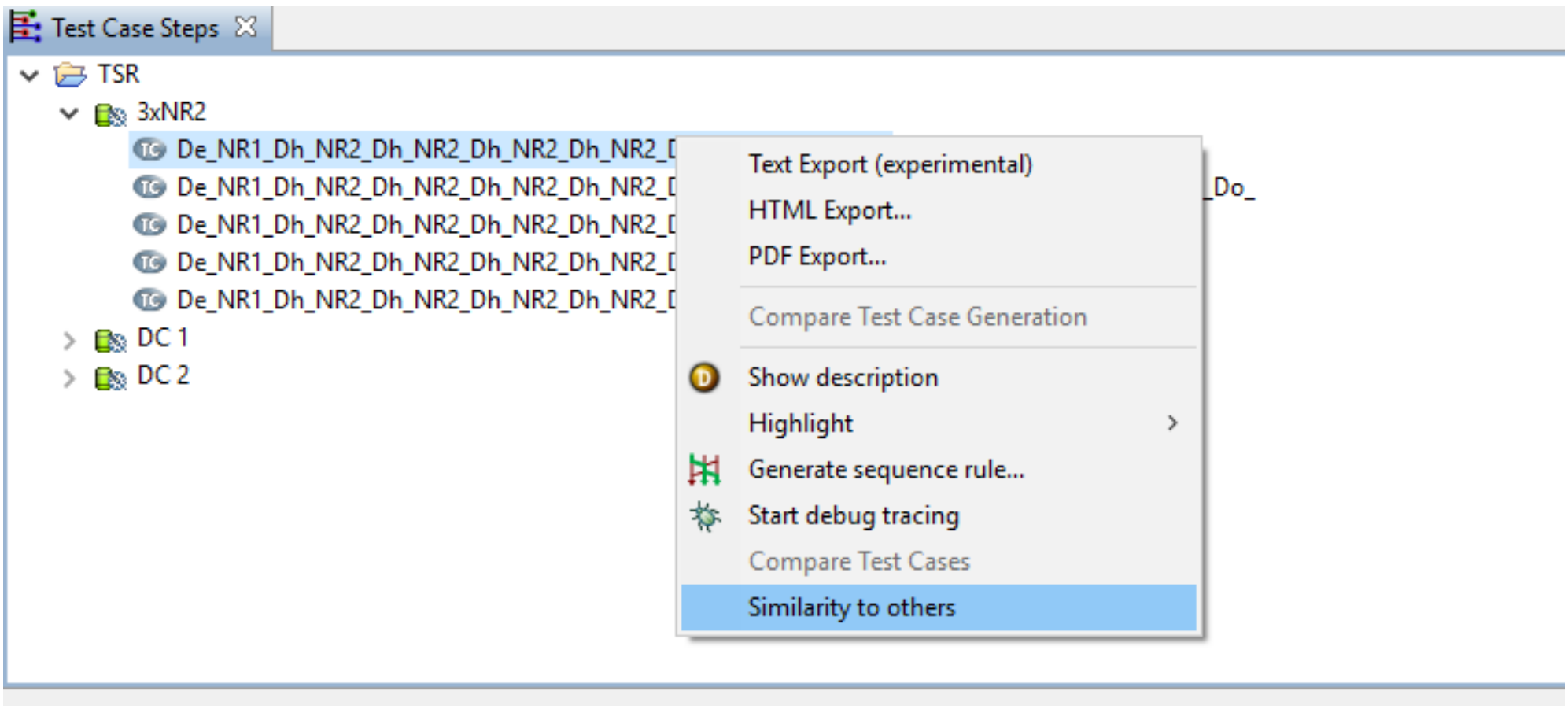
The Coverage view consists of two columns. "Test target" column shows specific groups such as Requirements, State Chart, Transition Pairs und Transitions and single test goals. The "Coverage ..." column shows the coverage for whole project or for selected DC.



Similarity View (en)

Similarity View gives users the opportunity to perform a similarity measure.

The test cases will be investigated and evaluated together. The goal is to be able to recognize test cases that are similar to other test cases, e.g. to avoid or prioritize their execution.



The Similarity View is a table and consists of several columns. The following information will be shown:

- Test case name
- Number of steps in the test case
- Number of equal steps (over the whole test case)
- Percentage of equal steps (over the whole test case)
- Number of equal steps without interruptions
- Percentage of equal steps without interruptions

At the top left is the name of the test case, which will be compared to others.

Similarity View

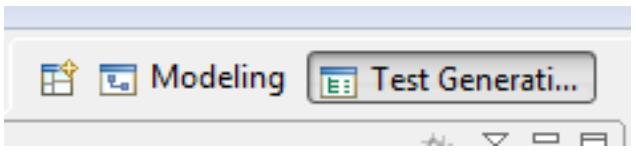
Test Case Name: De_NR1_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_Dh_TO_Dr_AD1_Do_ Steps: 84

Test Case Name	Test Steps Count	Same Steps #	Same Steps %	Sam
De_NR1_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_Dh_TO_...	102	76	74%	7
De_NR1_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_Dh_TO_...	102	76	74%	7
De_NR1_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_Dh_TO_...	111	76	68%	7
De_NR1_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_Dh_TO_...	120	76	63%	7

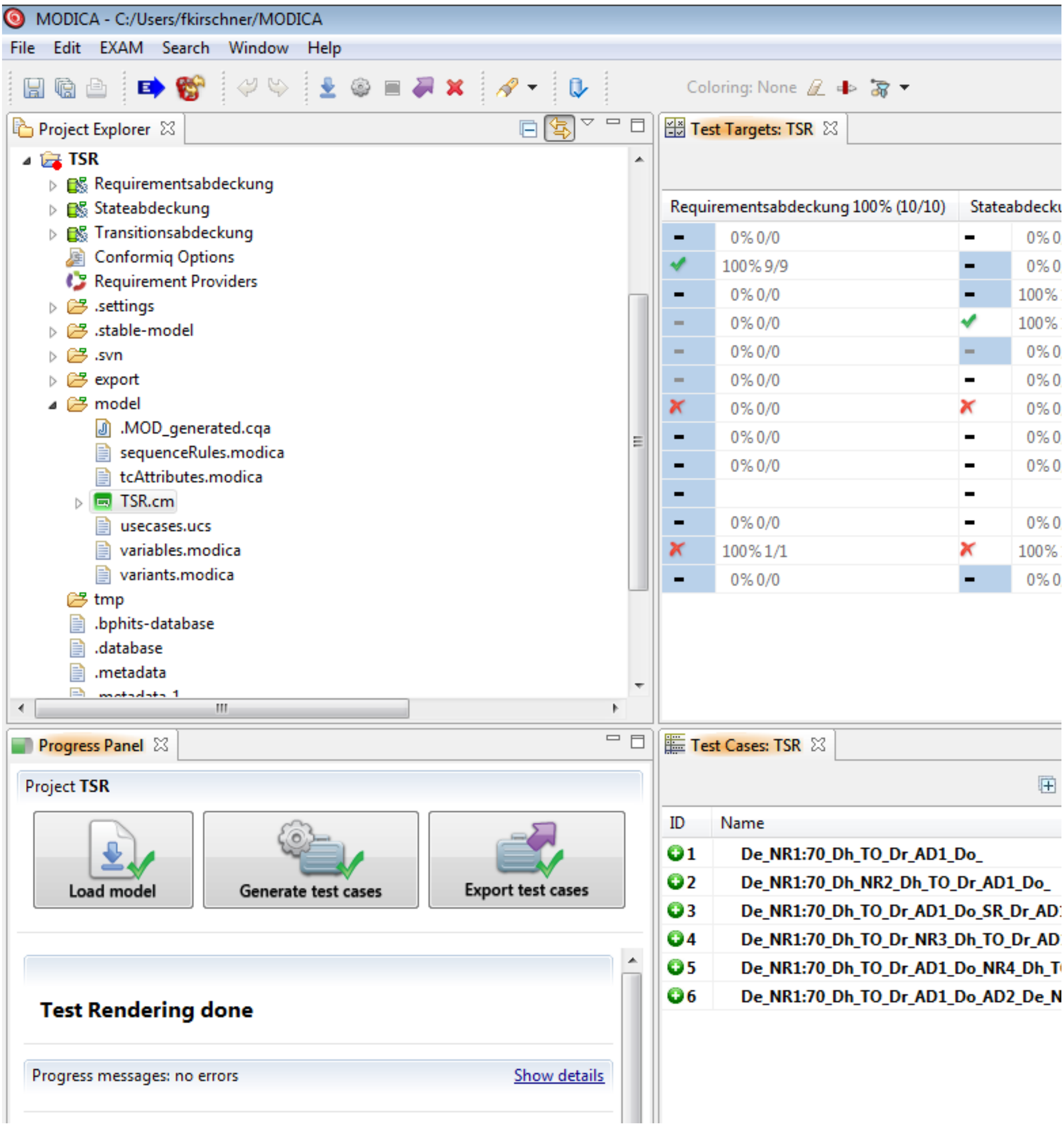
Use Cases

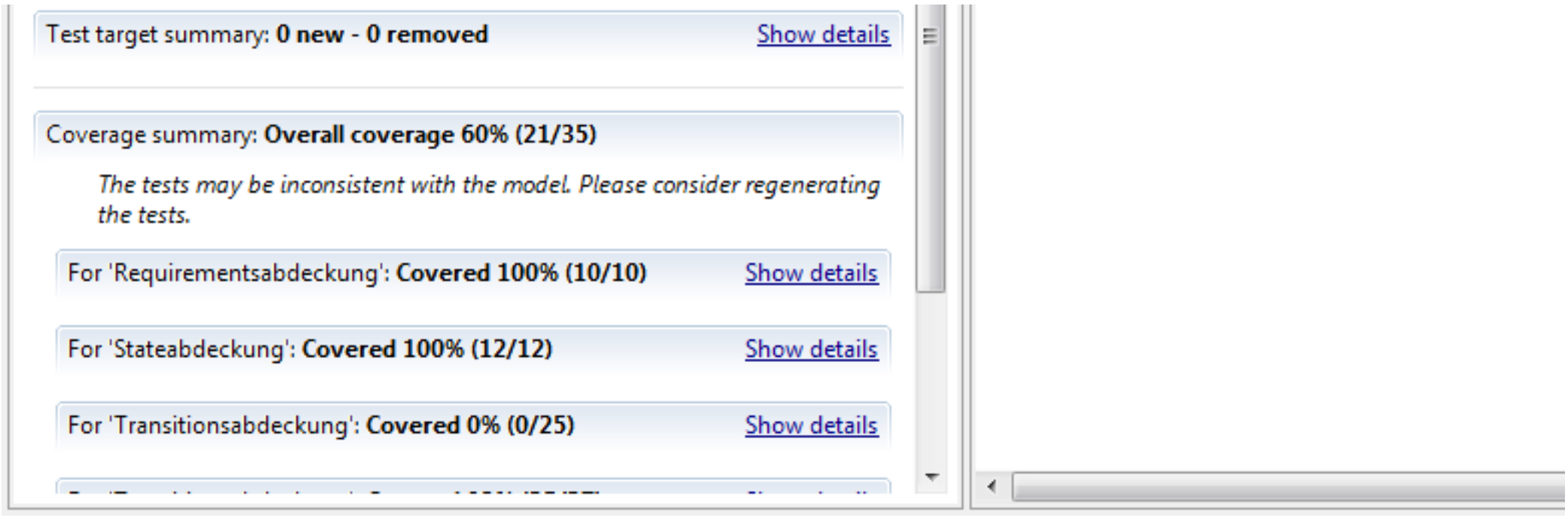
Test Case Generation

In order to generate test cases, you have to switch to the *test generation perspective*.



The *generation perspectiv*e looks something like this.





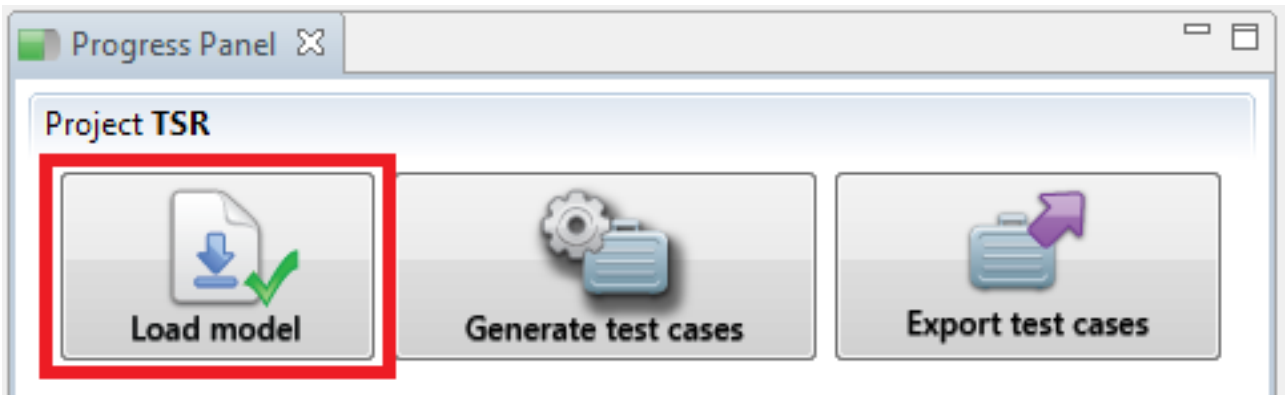
The basic procedure for the test case generation can be subdivided into the following points:

1. **Load model**
2. **Create design configuration**
3. **Set global test parameters**
4. **Set specific test targets**
5. **Generate test cases**
6. **Visualization and analysis of the test cases**
7. **Export of the test cases**

Load Model

Before test cases can be generated, the MODICA model must be checked for correct syntax or modeling errors.

This is done via the button *Load model* in the *Progress Panel View*:



Depending on the use of *variants* and *sequence rules*, one or more warnings under *Progress messages* can appear at this point. "*Failed to make the following request statement true [...]*"

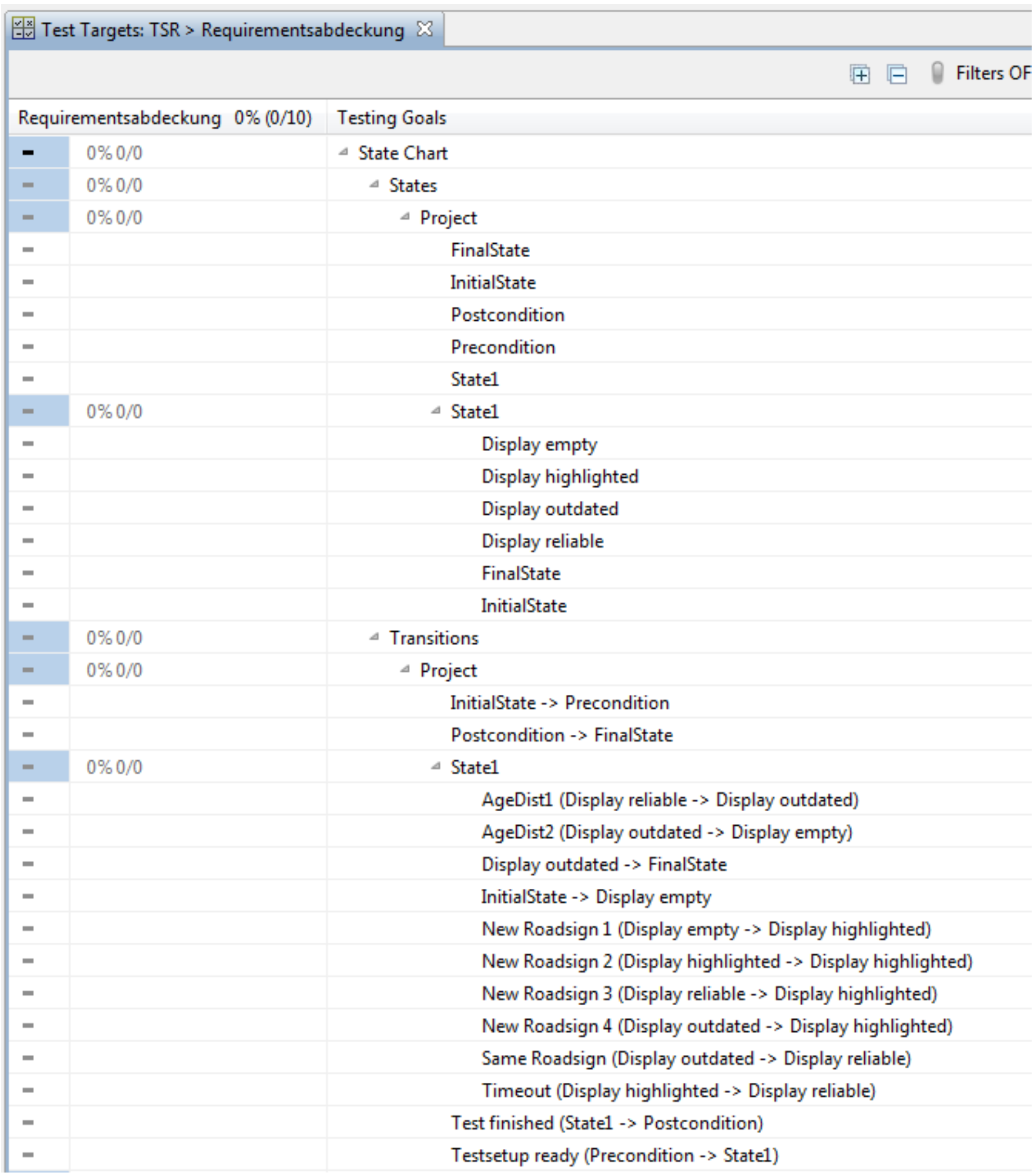
This kind of warning is, however, completely normal if *variants* and *sequence rules are used*, and you do not have to pay further attention to them.

In addition, error messages can occur which can not be ignored, as test cases can only be generated with a completely error-free model. In order to recognize such errors at an early stage, a project validation should be carried out before the

LoadModel step, in order to better localize and correct the errors.

To do this, you must switch to the *modeling perspective* and click on the button. If errors are present, they are displayed in the *Problems View*. A more detailed documentation on the project validation can be found in the chapter Test Case Generation.

Once a model has been successfully loaded, the groups of testing goals are filled with the available testing goals of this MODICA model, e.g. all states, transitions, requirements, branches, queries, function calls, etc. For example, In the *State Chart* group under *States*, you can find all the different states of the related MODICA model.



Create design configuration

For the creation of test cases in MODICA so-called *design configurations* are created, in which specific test parameters and

test targets are defined.

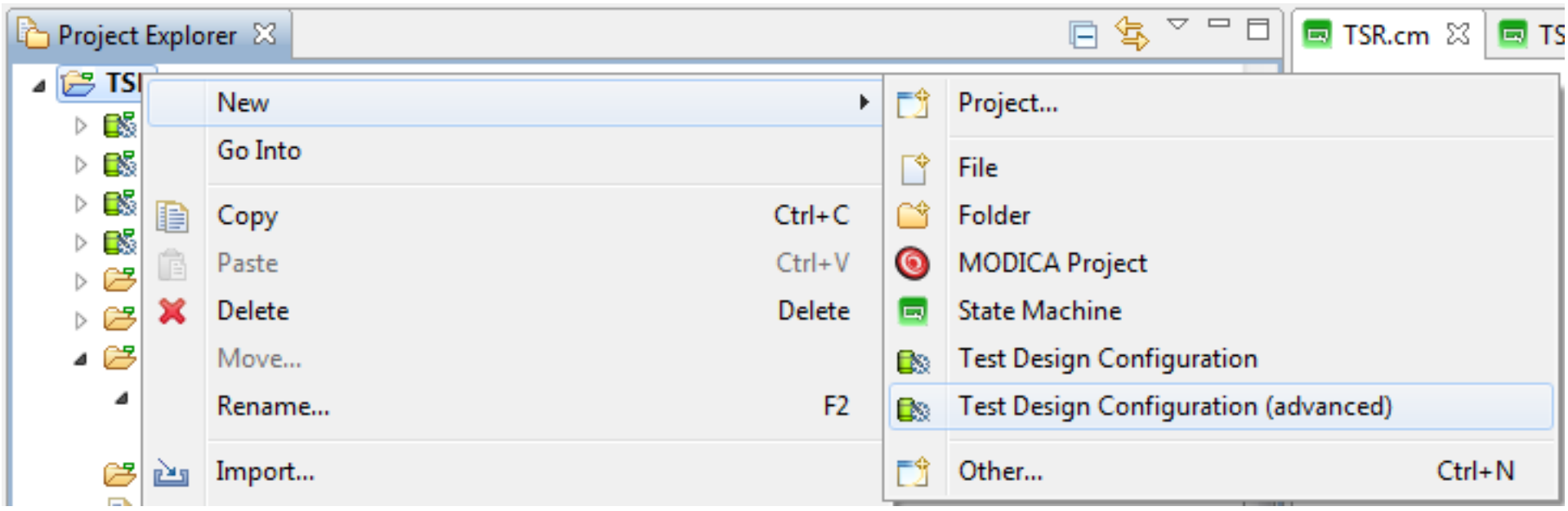
In a MODICA project several *design configurations* can be created and different test suites can be generated from one and the same model.

A *design configuration* is also called the generation strategy and describes which goals are fulfilled with this strategy. Generation strategies are project-related and therefore have to be individually created and adapted for each MODICA project.

For each of these *design configurations*, a set of model-based coverage criteria can be defined, which are used by the MODICA test case generator to create appropriate and meaningful test cases that meet the test objectives of this generation strategy.

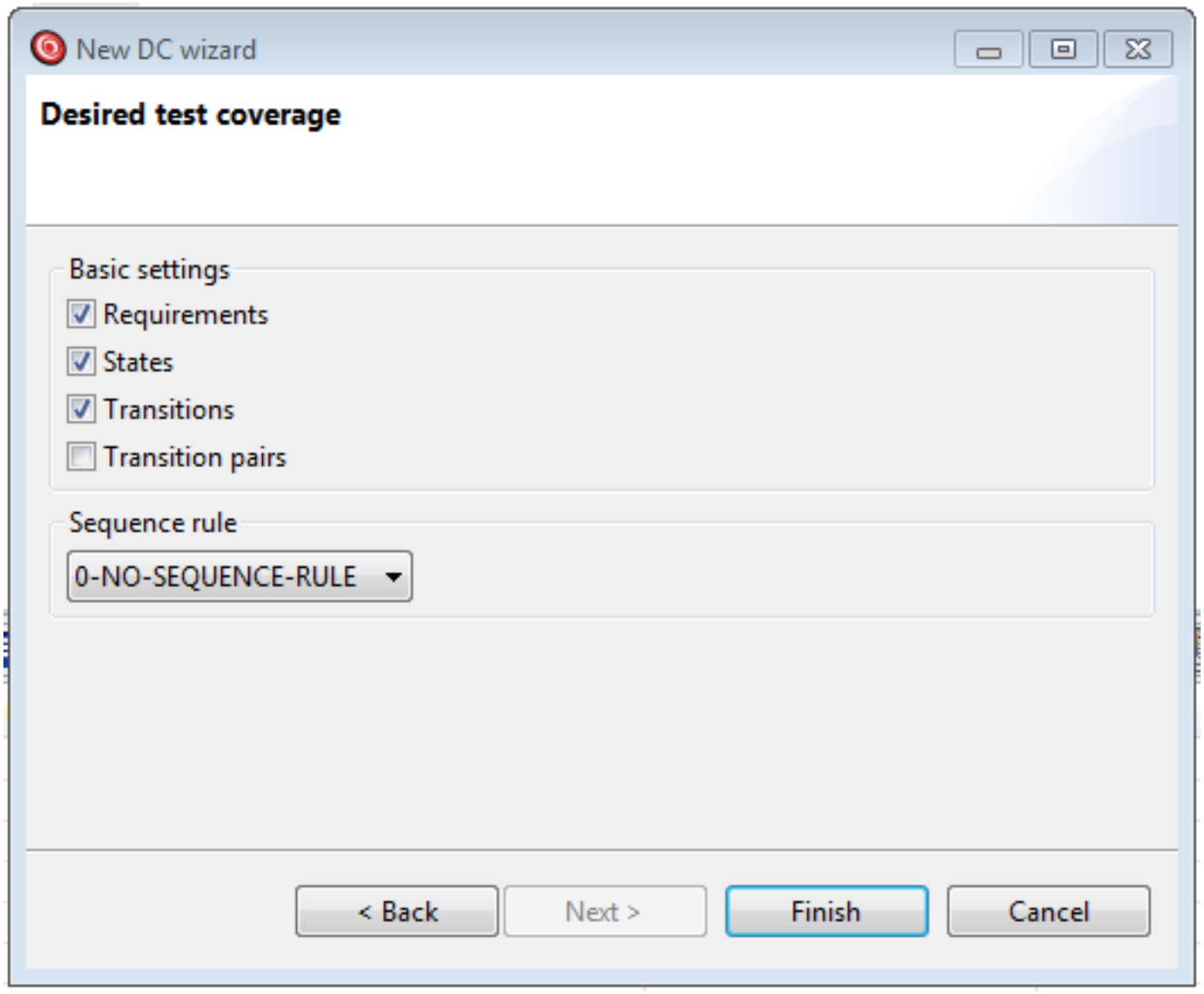
Define generation strategies

A new generation strategy is created by right-clicking on the desired MODICA project in the *Project Explorer* and selecting *New → Test Design Configuration (advanced)*.



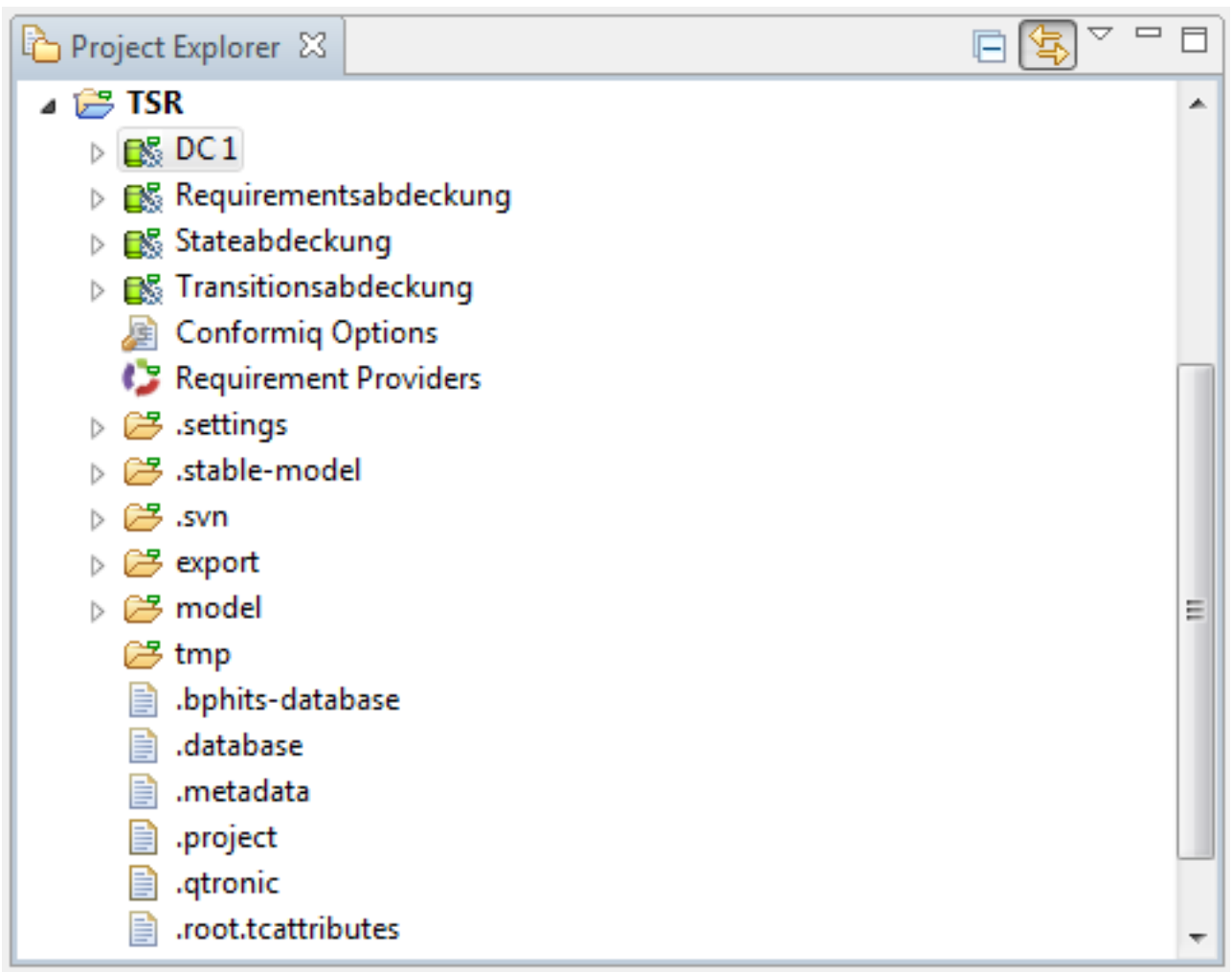
Test case generation, Define generation strategies

A name for the generation strategy can be specified in the dialog *New DC Wizard*. After clicking on *Next*, there is a pre-selection of prefabricated test targets. Besides the possibilities offered here, you can further refine your test targets in the *Test Target View*. If one or more *sequence rules* have been created in the current model, it is possible to define one of them as an additional test target in the section *Sequence rule*.



Test case generation, Create new design configuration

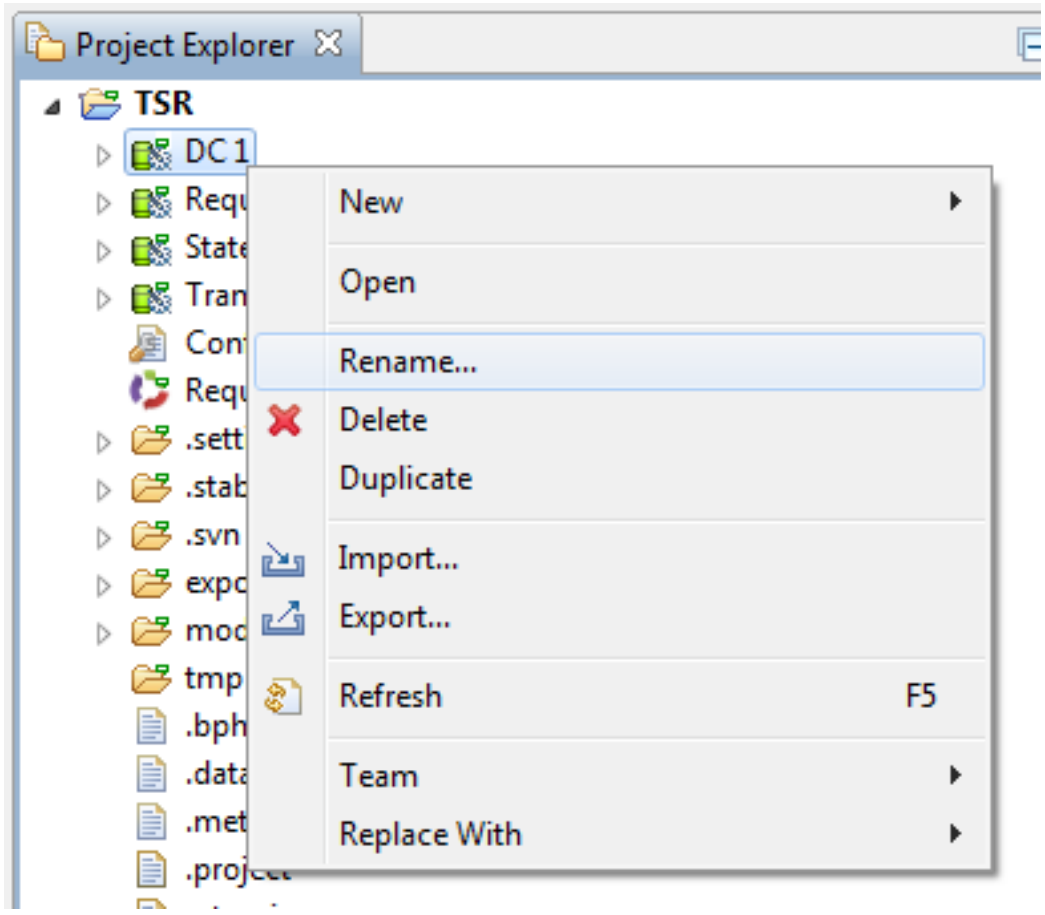
After confirming with *Finish*, the new generation strategy is created and can be found in the project tree.

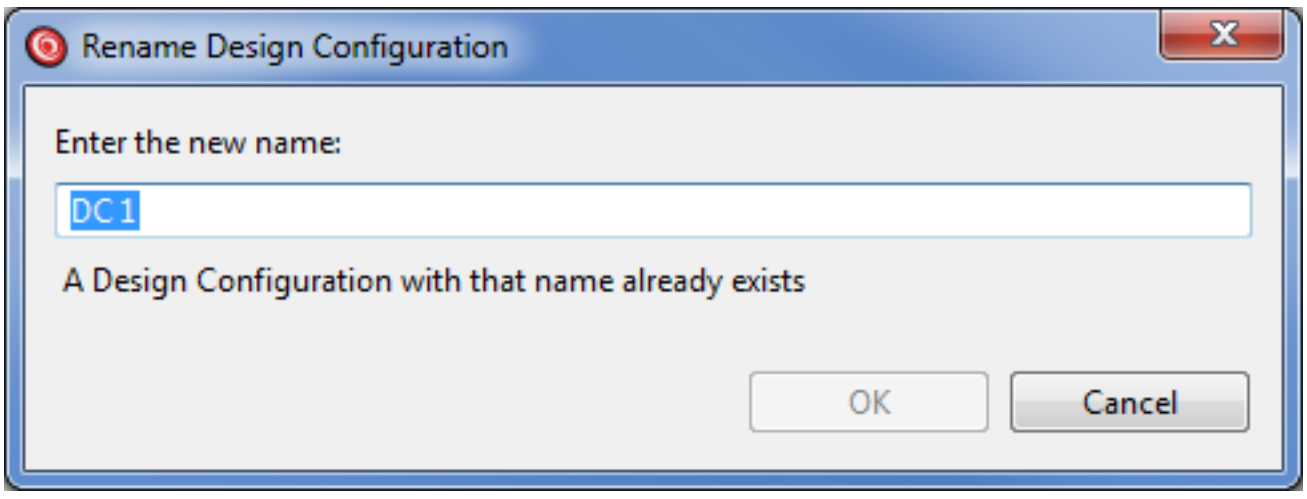


Rename design configuration

The name of a generation strategy can be changed at any time by *right-clicking* on the strategy in the *Project Explorer* and

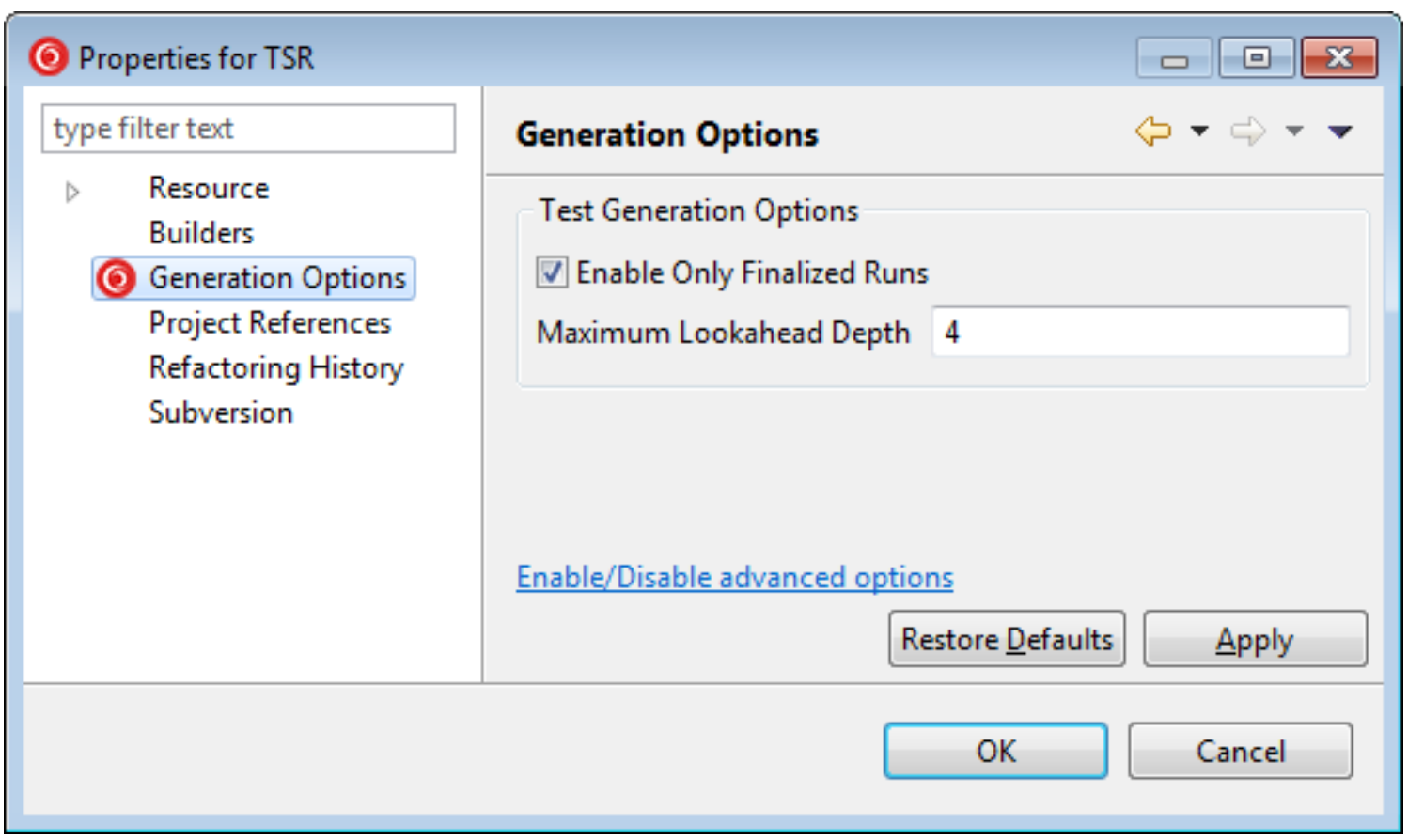
selecting *Rename*.





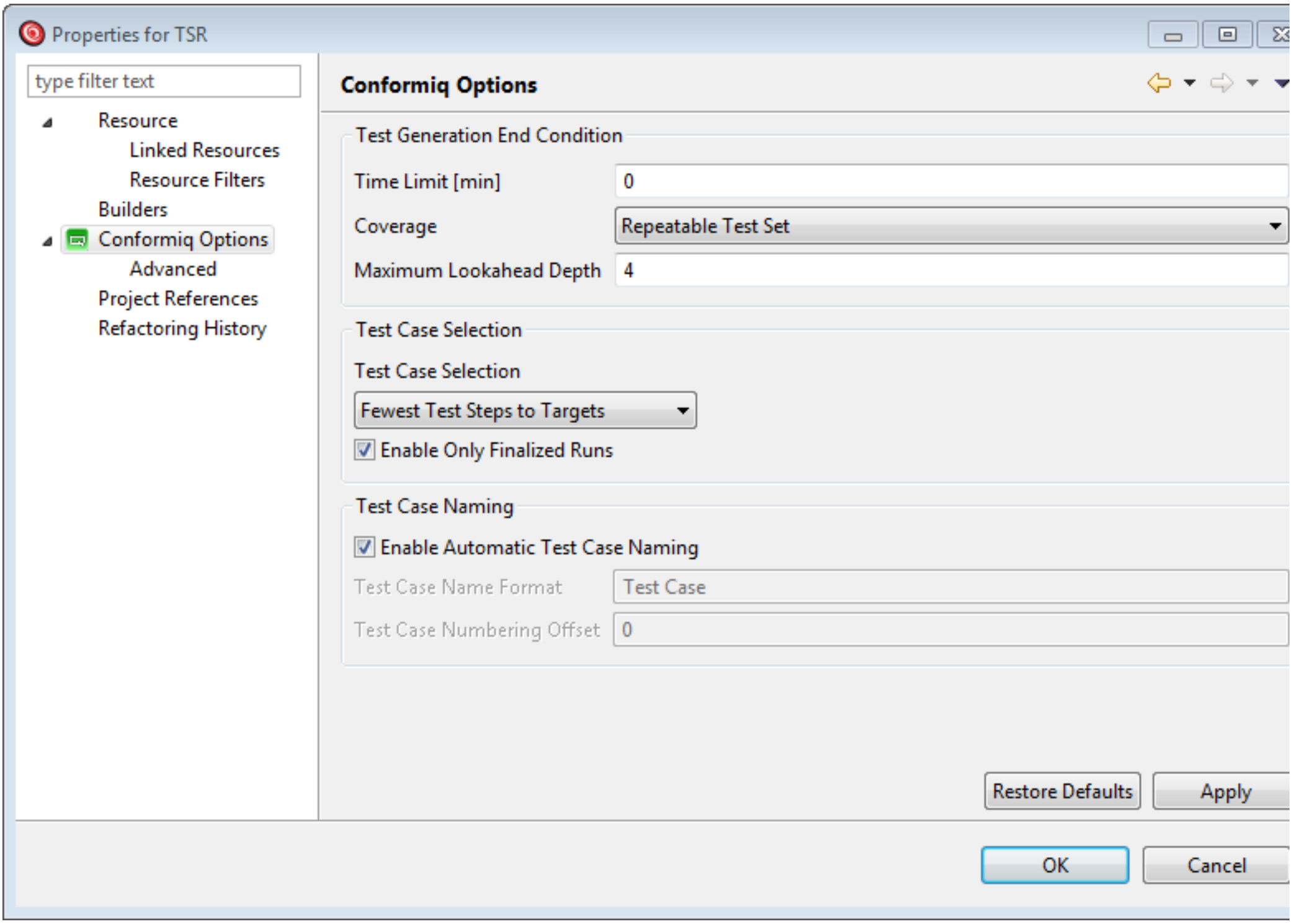
Setting project-wide generation parameters

Some parameters for the generator can be configured on the project level. To access them, use *Properties* in the context menu of a project. The *Generation Options* page allows to configure the most common settings:



- *Enable Only Finalized Runs*
  If selected, the generator will only produce tests that trace the model from start till end (final state at root level). If the end is unreachable, no tests will be generated.
- *Maximum Lookahead Depth*
  This setting controls, how deeply, the generator will keep searching the model, when none of the possible steps (~triggers) directly increase the coverage.

⚠ Using *enable/Disable advanced options* it is possible, to access more complex expert-features. They are described below, but should be used with care.



The settings in this dialog are explained below.

Test Generation End Condition

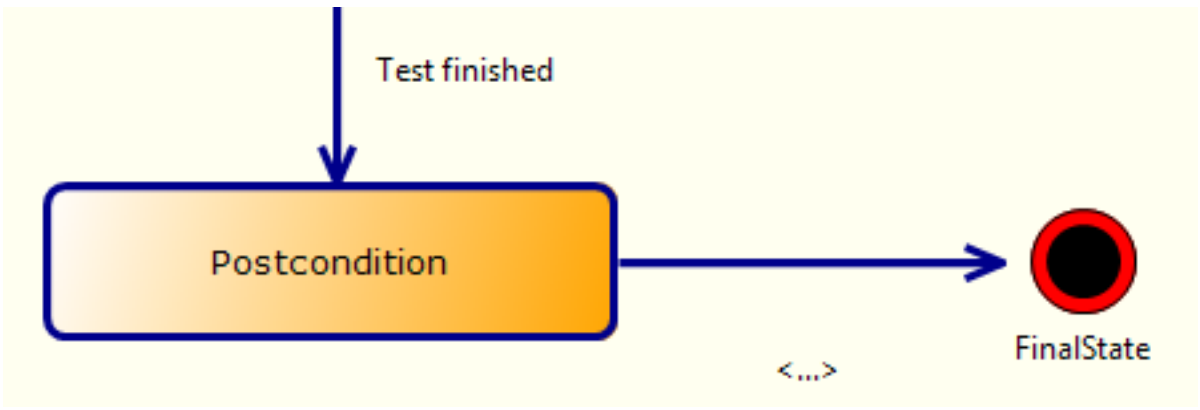
- **Time Limit:** If a value other than `0` is set here, the generator stops its caluculations after the specified number of minutes. `0` means that no time limit has been set.
- **Coverage:**
  - **Stop at Full Requirement Coverage**: When *Stop at Full Requirement Coverage* is activated, the test case generation is immediately stopped when 100% of the requirement coverage is reached. All requirements used in the usage model are covered by at least one generated test case. *Stop at Full Requirement Coverage* has priority over *Stop at Full Coverage*, which causes the generation to be stopped as soon as all requirements have been covered - even if further test cases are possible to cover additional test targets. It is recommended to disable the *Stop at Full Requirement Coverage* setting.
  - **Stop at Full Coverage**: When *Stop at Full Coverage* is activated, the test case generation is stopped as soon as 100% of the desired coverage criteria have been reached, i.e. all test targets are covered by the set of

generated test cases . If you deactivate the option, depending on the model, the test cases are no longer deterministically selected, which can lead to unnecessary changes in the amount of generated test cases. It is recommended to disable the *Stop at Full Coverage* setting**.**

- **Maximum Lookahead Depth**: The *Lookahead Depth* parameter specifies the extent to which the test case generator should include events in the course of a user model in the decision-making process. If a very low value is set here, dependencies in the sequence of successive test steps are not visible for MODICA and are not taken into account, which means that possible test sequences are not recognized. If, however, a very high value is selected, this leads to very long generation times, because a large part of the possible future events are precalculated at every step through the model, and any dependencies are included in the path determination. Useful values for the lookahead depth are values in the range 3-5, and if these values do not lead to a satisfactory result, it is assumed that there are problems in the usage model. It is recommended to first carry out the test case generation with a low value for the *lookahead depth* parameter (e.g., 4) and to increase this value if test targets are not reached which could theoretically be achieved**.**

Test Case Selection

- **Test Case Selection:** The generator's strategy can be influenced here. It is recommended to select *Fewest Test Steps to Target* - there are other possibilities for experts available.
- **Only Finalized Runs**: If *Only Finalized Runs* is activated, only test cases are generated that reach a defined final state (*FinalState*). This means that only test cases are generated that reach the *FinalState* in the main diagram of the usage model.



It is recommended to enable the *Only Finalized Runs* parameter.

Test Case Naming

- **Enable Automatic Test Case Naming**: This setting is used to enable the "NamingFragment" feature. If "naming fragments" are not used in a test case and no unique test case name can be generated, MODICA adds a unique characteristic to the name, for example, *transitions* or *states* that are only passed by this test case, covered *requirements*, a calculated value for a value assignment, etc. Once the *Naming Fragments* are used in a model, the *smart test case naming* is disabled and the modeler must ensure that the name of the test case is unique. In the fields *TestCaseNameFormat* and *TestCaseNumbering Offset* default names for created test cases can be set. It is recommended to activate the *Automatic TestCaseNaming* setting.
- **TestCaseNameFormat** and **TestCaseNumbering Offset**: This selection is only available if *Automatic test case naming* is disabled. In the *TestCaseNameFormat* field, a default name can be specified and in *TestCaseNumbering* Offset the start value with which the numbering of the test cases should start. For example, as *name format* `Test_Case_` and as *Numbering Offset* `0`, the created test cases are named as follows: Test_Case_0, Test_Case_1, Test_Case_2, etc

Set specific test targets

These specific settings apply only to one generation or one test target and are ignored by all other test case generations.

The coverage criteria (including testing goals or test targets) of a generation strategy are necessary to guide the generator through a usage model and enable it to make certain decisions and find behavior patterns. By exploring the structure of a state chart(states, transitions) or the constructs of the description language (conditional branches, Instructions, calculations)

the generator can derive test cases from them.

Cover criteria are hierarchically organized and packed into groups. The deeper you navigate into a group, the more fine-grained settings can be made.

The following groups are currently available:

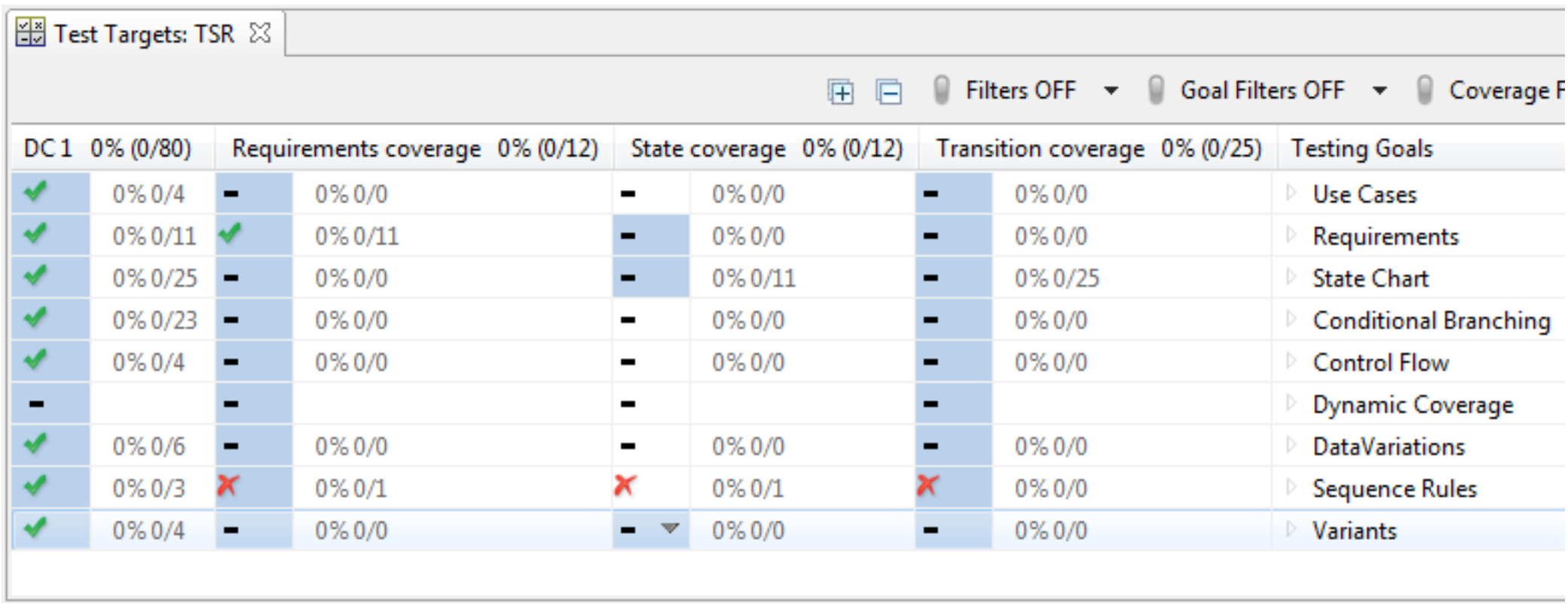
- Use Cases (Not relevant for MODICA and not to be used)
- Requirements
- State Chart
- Conditional Branching
- Control Flow
- Dynamic Coverage
- Data Variations (Only when used)
- Sequence Rules (Only when used)
- Variants (Only when used)

The test criteria are displayed in the *Test Targets View* in the *Test Generation Perspective*. This view always shows the testing goals of the generation strategy, which is currently selected in *Project Explorer*.

Generation strategies - examples

In the example project TSR, four test generation strategies are given as an example:

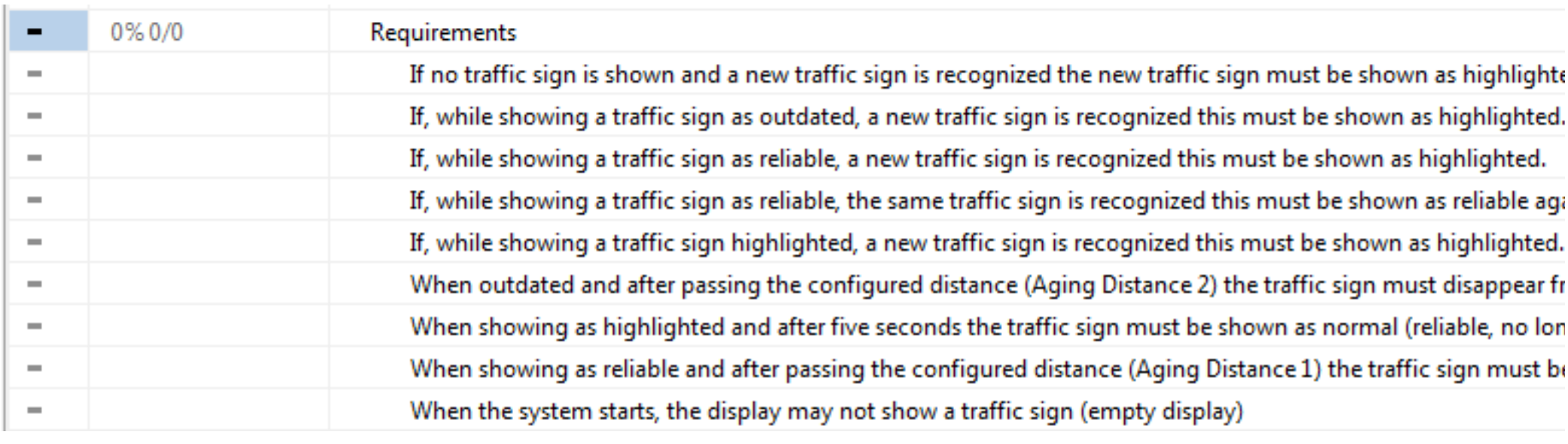
- DC1
- Requirements coverage
- State coverage
- Transition coverage



Using the three buttons *Filters ...*, *GoalFilters ...* and *CoverageFilters ...* in the header of the *Test Targets View* you can control the displayed elements in the table. The buttons each have a submenu that contains the elements to be filtered. If a filter is activated, the button shows ... *ON*, otherwise ... *OFF*. These buttons can be used to increase clarity and to quickly locate e.g. set test targets.

The following describes how the different test target groups can be used to set and achieve test targets.

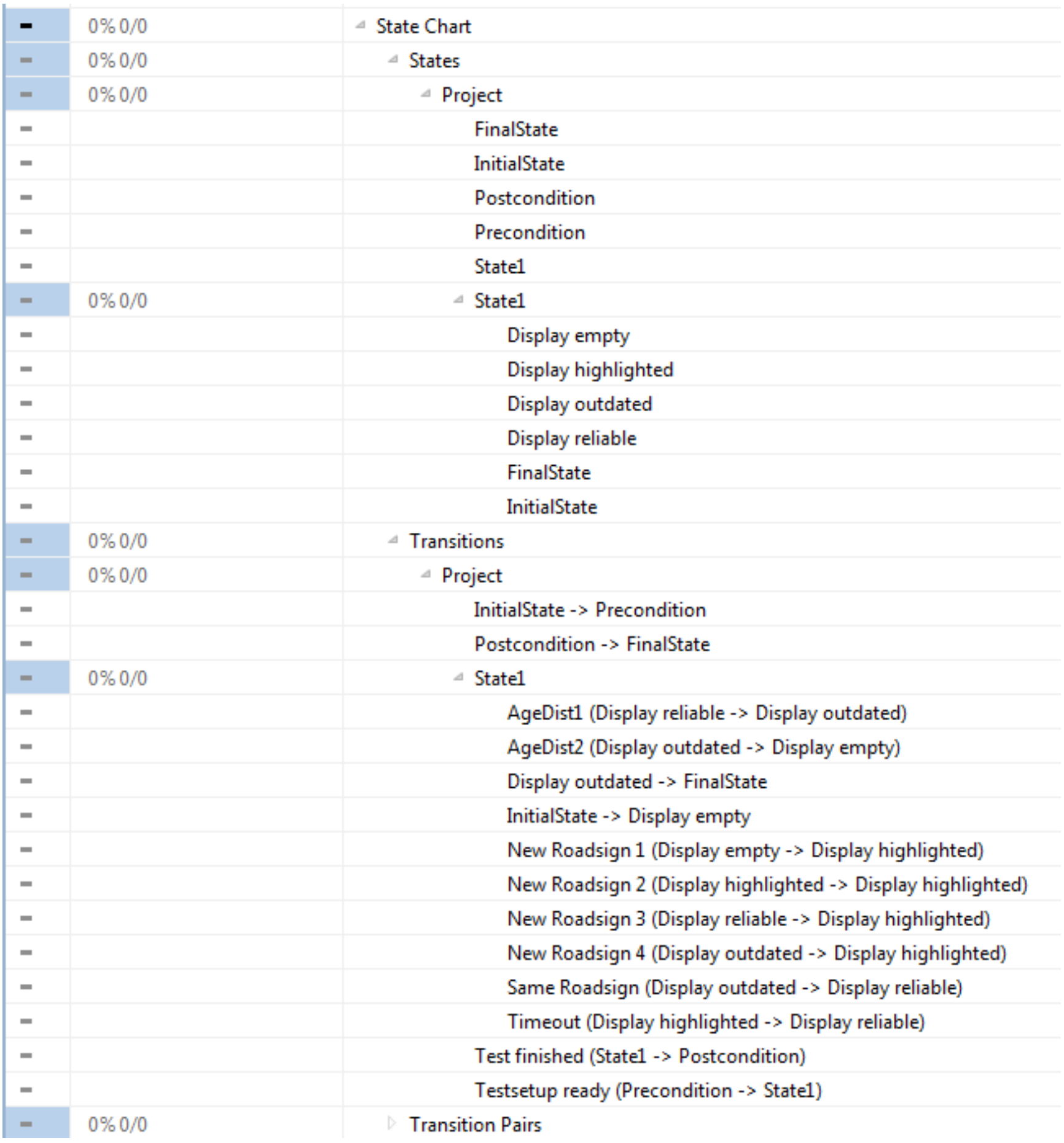
Requirements



In the *Requirements* group, individual *requirements* can be selected to make the test case generator create test cases that cover all selected *requirements*.

Attempts are made to generate the minimum number of test cases in order to cover all selected *requirements*.

State Chart



In the *State Chart* group you can select elements of the model, which should be fulfilled by the generation.

These model elements are:

- **States**: All *states* which can be found in the model can be explicitly selected or deselected as a generation target. If all *states* (State Coverage) are selected, this causes the test case generator to attempt to go through each state of the model at least once in the set of all generated test cases.
- **Transitions**: All *transitions* which can be found in the model can be explicitly selected or deselected as a generation target. If all *transitions* (Transition Coverage) are selected, this causes the test case generator to attempt to go through each transition at least once in the set of all generated test cases.
- **Transition Pairs**: All pairs of possible successive *transitions* of a model can be explicitly selected or deselected as a generation target. If all *transition* pairs are marked, the test case generator tries to run through every *transition* pair at least once in the set of all generated test cases. This function has the potential to generate a lot of test cases. In larger models, this function should therefore be used iteratively. That is, we should try to select only a small set of specifically desired transition pairs in the subtree. If this is complex, however, it is also possible to set all pairs as test targets. However, it will often be the case that many of these pairs can not follow each other directly because the present *guards* in the model prevent the direct sequence.

- **Implicit Consumption** (has no function in MODICA, can be ignored): This point is automatically deactivated and should not be used in MODICA.

Conditional Branching

This section represents internal procedures in MODICA and must set on *Do not care*. A different marking should not be made in MODICA.

Control Flow

This section represents internal procedures in MODICA and must set on *Do not care*. A different marking should not be made in MODICA.

Dynamic Coverage

In this group, there are test targets that also allow to create test cases which do specific things multiple times or in different sequences and combinations. However, many of these test targets can lead to a very large number of test cases depending on the type of model. Here sequence rules could be the better solution.

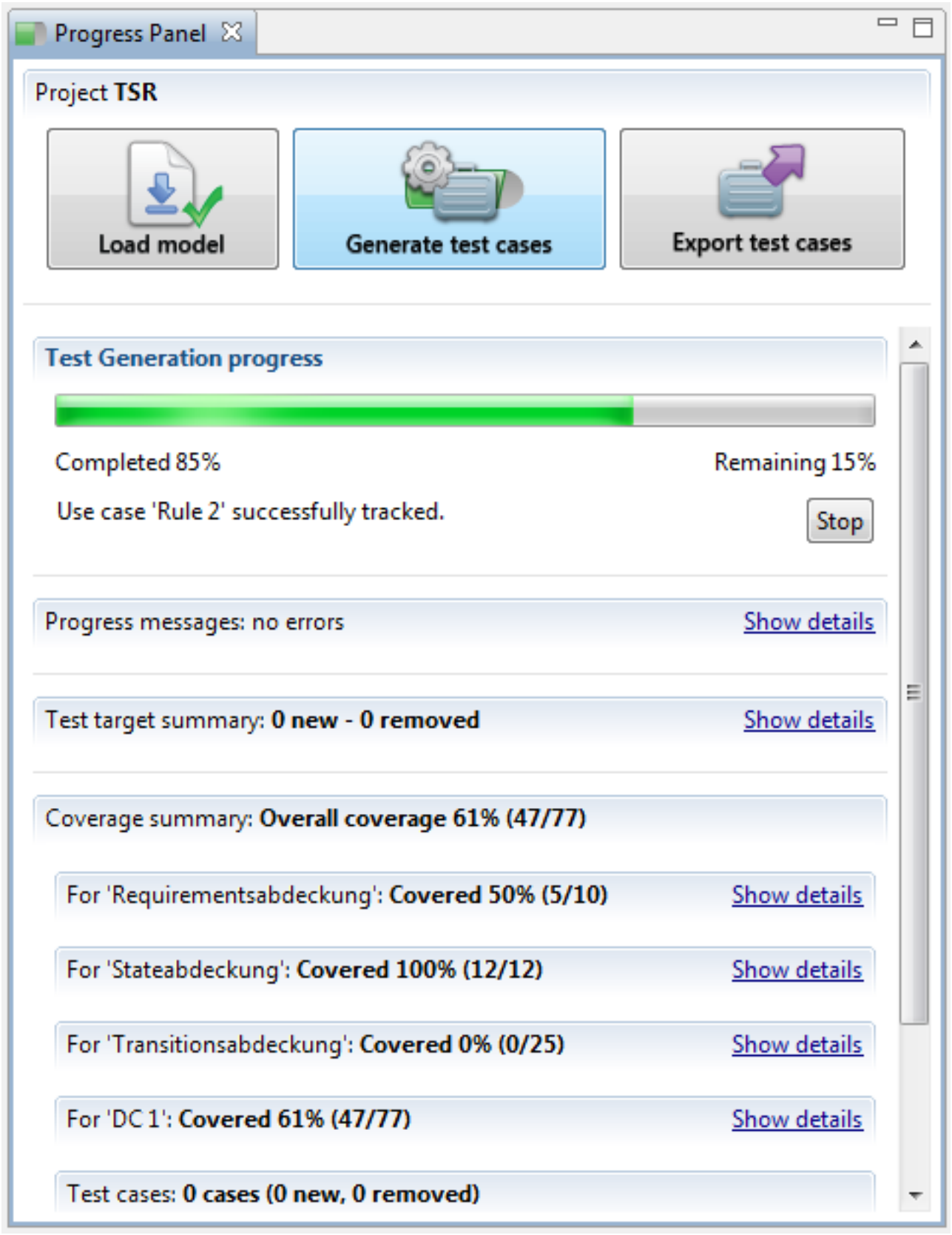
-		Dynamic Coverage		
-		Parallel Transitions		
-		All paths: States		
-		All paths: Transitions		
-		All paths: Conditions		
-		All non-looping paths: States		
-		All non-looping paths: Transitions		
-		All non-looping paths: Conditions		

Generate test cases

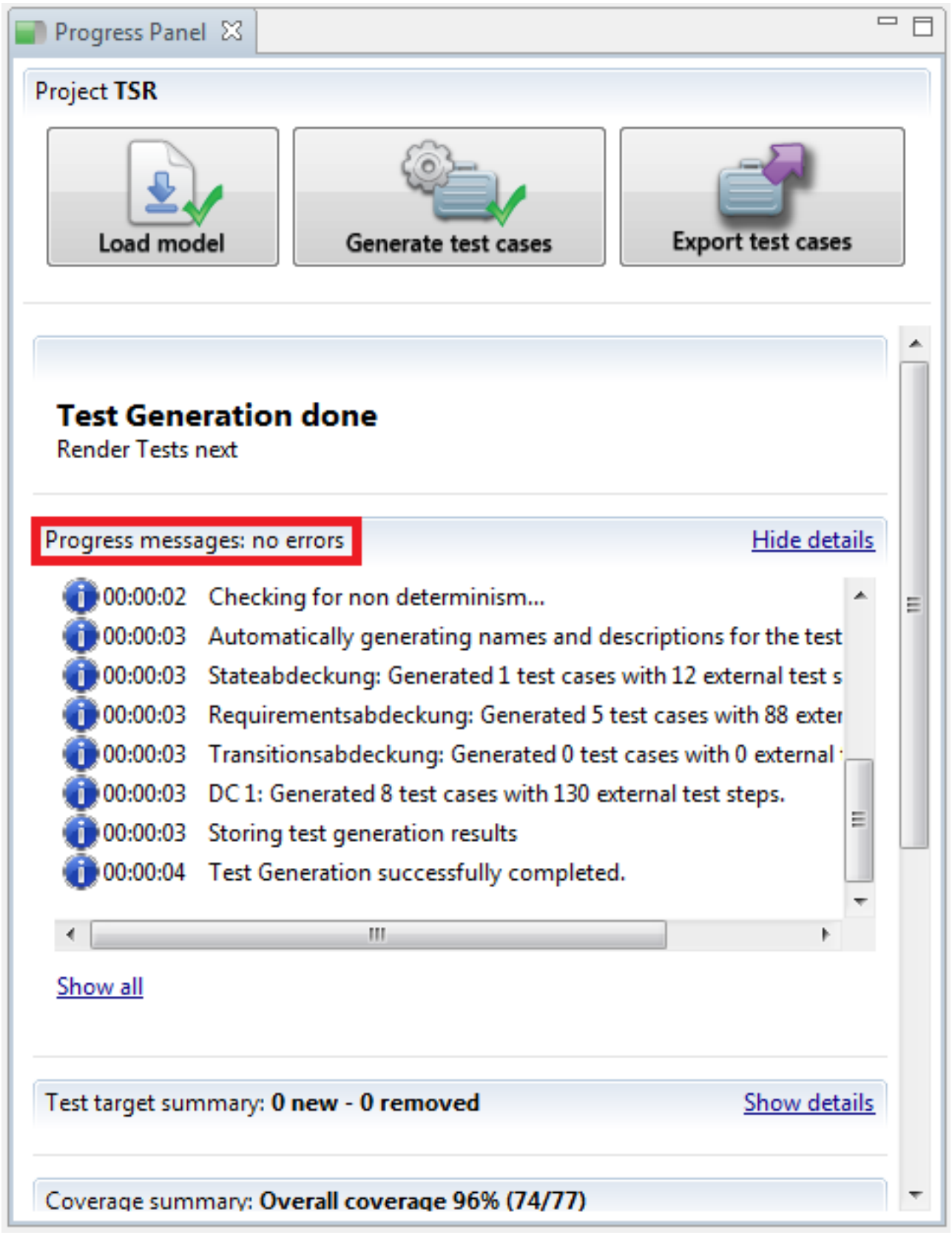
If the desired test case generation strategies have been set, the generator can be started via the button *Generate test cases*. All test cases are generated for all defined generation strategies. So in the example, `DC1, requirements coverage, state coverage,` and `transition coverage`.

The entire set of all test cases is generated, which is possible with the currently defined generation strategies. However, only the tests that meet the required conditions (test targets) are displayed for each generation strategy.

After the test case has been started, the progress can be tracked in the *Progress Panel View*.



If you click on *Show details* in the *Process messages* section, the exact course of the test case generation can be traced. Here, for example, information about the number of generated test cases per *design configuration* is displayed or possible errors are listed.



Tipp

If you are in the *Modeling Perspective* and would like to start the test case generation before switching to the *Test Generation Perspective*, you can click on the gear in the upper toolbar. This automatically executes the *Load model* and *Generate testcases steps* one after the other.

Visualization and Analysis of Testcases

The generated test cases can be analyzed in the following views.

- Test Cases
- Model Browser
- Test Case Steps

Test Cases

In this view, all created test cases are displayed in enumerated form. Each test case can be expanded so that you can see its test case description.

By selecting a generation strategy in the *Project Explorer*, this view shows exactly the set of test cases that meets these settings.

Test Cases: TSR | Test Case Steps

Filters ON | Sort Tests Shortest, Oldest Test Firs

ID	Name	Targets covered	Created	Priority
1	De_NR1:70_Dh_TO_Dr_AD1_Do_	105 (4 req.-s, 101 ot...	2016-19-12 07:...	1
2	De_NR1:80_Dh_TO_Dr_AD1_Do_	106 (4 req.-s, 102 ot...	2016-19-12 07:...	1
3	De_NR1:100_Dh_TO_Dr_AD1_Do_	107 (4 req.-s, 103 ot...	2016-19-12 07:...	1
4	De_NR1:70_Dh_NR2_Dh_TO_Dr_AD1_Do_	113 (5 req.-s, 108 ot...	2016-19-12 07:...	1
5	De_NR1:70_Dh_TO_Dr_AD1_Do_SR_Dr_AD1_Do_	114 (5 req.-s, 109 ot...	2016-19-12 07:...	1
6	De_NR1:70_Dh_TO_Dr_NR3_Dh_TO_Dr_AD1_Do_	114 (5 req.-s, 109 ot...	2016-19-12 07:...	1
7	De_NR1:70_Dh_TO_Dr_AD1_Do_NR4_Dh_TO_Dr_AI	114 (5 req.-s, 109 ot...	2016-19-12 07:...	1
8	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	114 (5 req.-s, 109 ot...	2016-19-12 07:...	1
9	De_NR1:70_Dh_TO_Dr_AD1_Do_AD2_De_NR1:70_[	115 (5 req.-s, 110 ot...	2016-19-12 07:...	1
10	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	123 (6 req.-s, 117 ot...	2016-19-12 07:...	1
11	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	123 (6 req.-s, 117 ot...	2016-19-12 07:...	1
12	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	123 (6 req.-s, 117 ot...	2016-19-12 07:...	1
13	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	123 (6 req.-s, 117 ot...	2016-19-12 07:...	1
14	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	131 (7 req.-s, 124 ot...	2016-19-12 07:...	1
15	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	123 (6 req.-s, 117 ot...	2016-19-12 07:...	1
16	De_NR1:70_Dh_NR2_Dh_NR2_Dh_NR2_Dh_NR2_DI	125 (6 req.-s, 119 ot...	2016-19-12 07:...	1

Test Description

Scenario classification: Start Test, Check: Display is empty, Action: Show new traffic sign, Check: Traffic sign is displayed highlighted, Action: Wait 5 seconds, Check: Traffic sign is displayed reliable, Action: Drive 5000 meters, Check: Traffic sign is displayed outdated, Action: Drive 2000 meters, Check: Display is empty, Action: Show new traffic sign, Check: Traffic sign is displayed highlighted, Action: Wait 5 seconds, Check: Traffic sign is displayed reliable, Action: Drive 5000 meters, Check: Traffic sign is displayed outdated, End Test.

Main Testing Targets

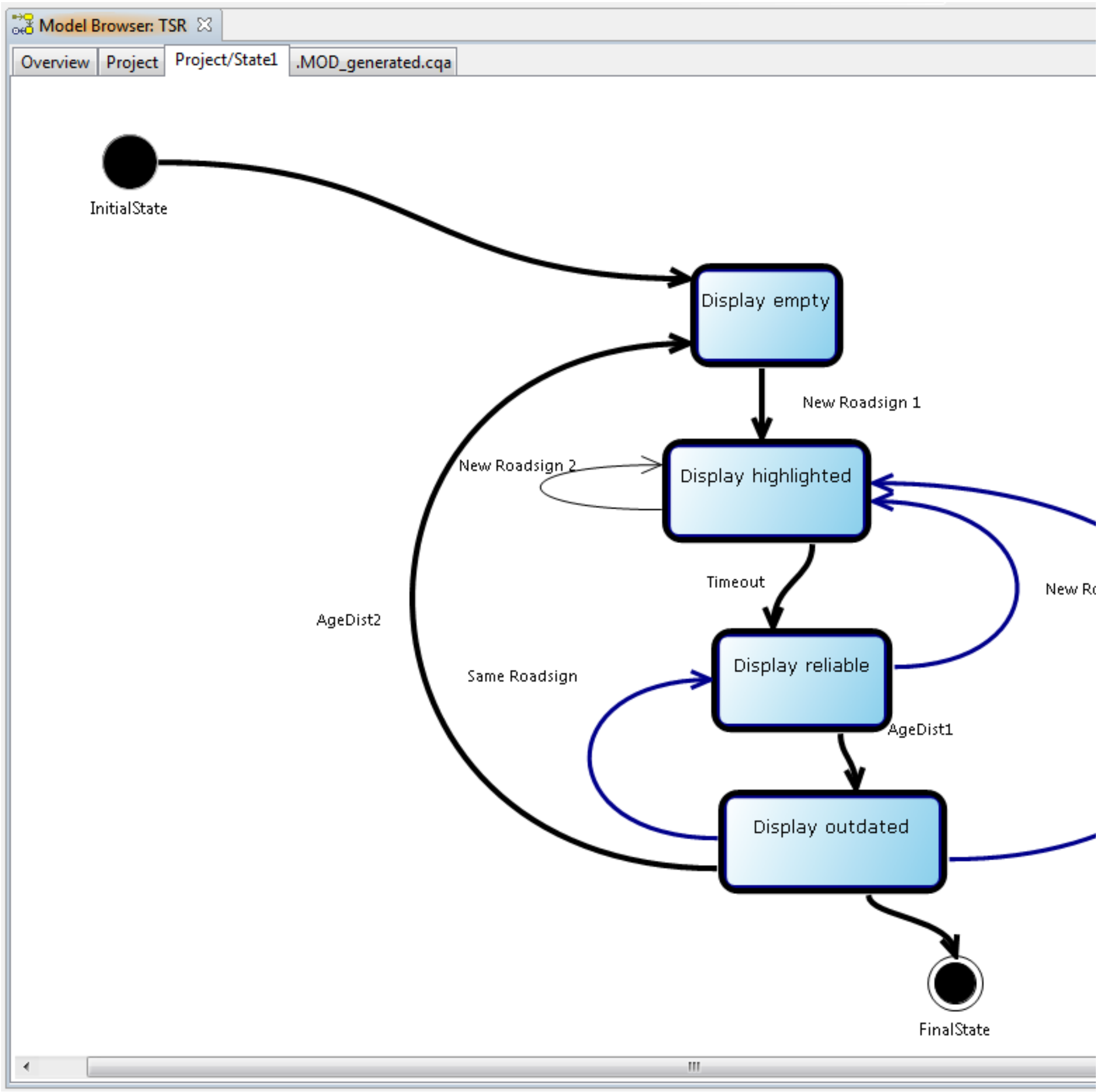
Overall coverage

Requirements	5/9
State Chart	31/87
State Chart - States	11/11
State Chart - Transition Pairs	10/22
State Chart - Transitions	10/14

Model Browser

By selecting a test case in the *Test Case List View*, it is automatically selected in all other analysis views.

In the *Model Browser View*, the path of the selected test case is marked in the diagram:



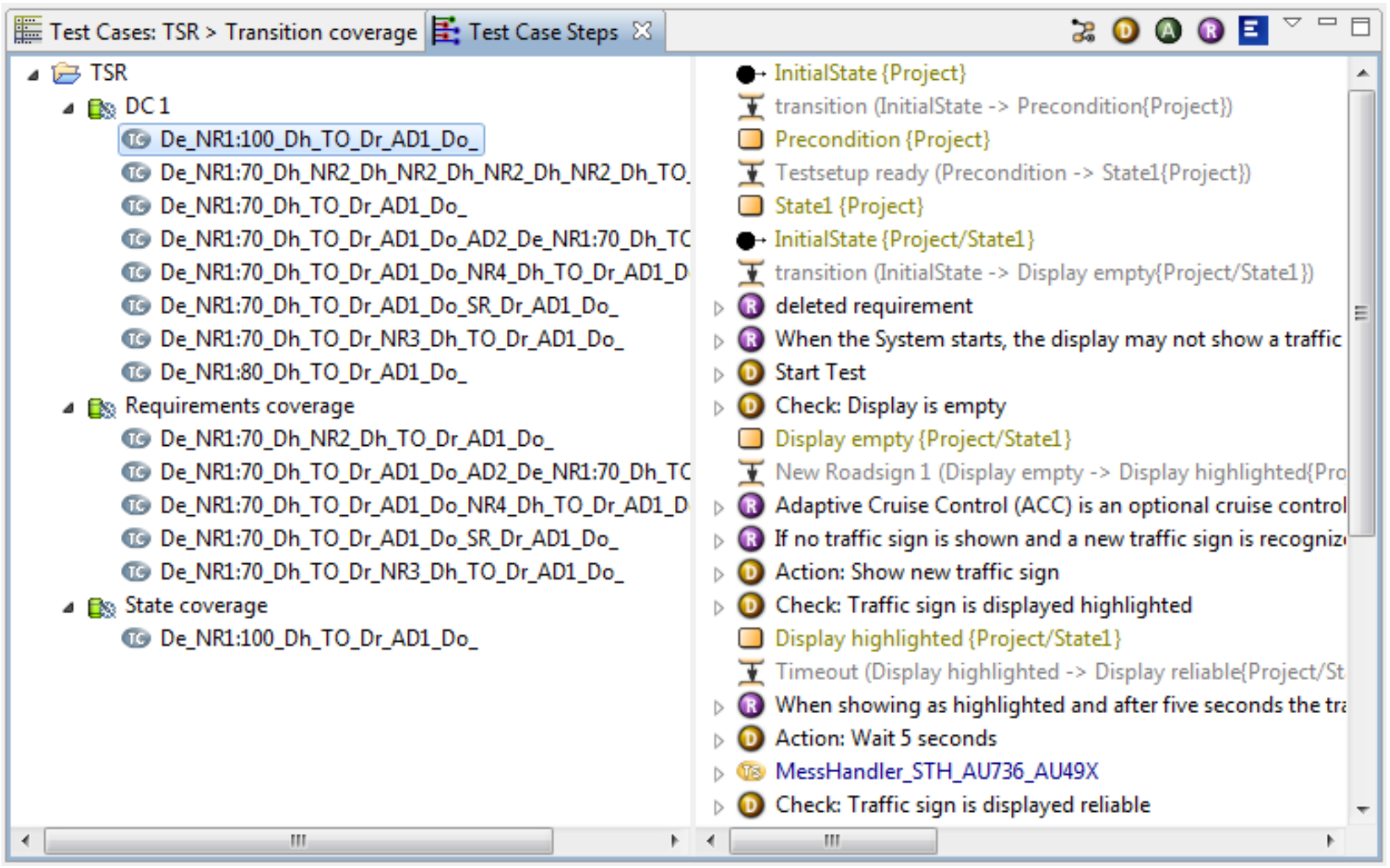
Test Case Steps

You can use this view to analyze the generated test cases before they are exported to a test automatisation environment.

Here you can see all created test cases, grouped in the *design configurations* in which they were created.

On the right side of the view, all objects of a test case are displayed in a list.

Further settings and a more detailed description can be found in the chapter Test Case Steps.

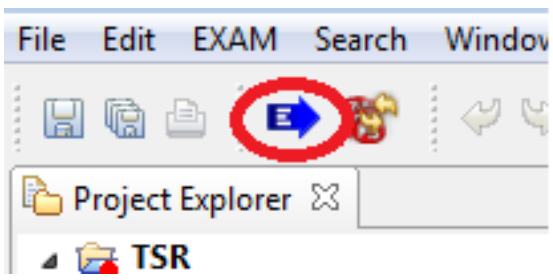


Export test cases

Currently there are the possibilities to export the generated test cases to EXAM or to convert them into several HTML documents.

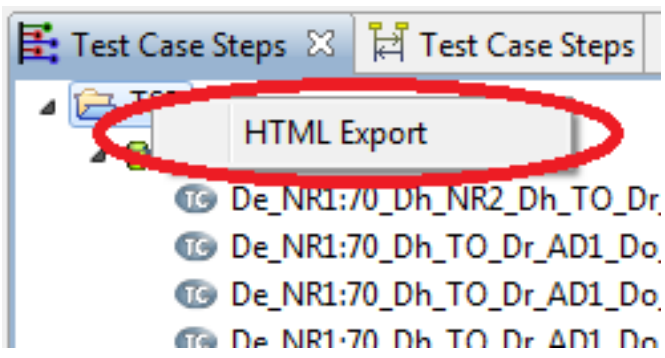
Export to EXAM

By using the blue arrow in the upper toolbar the test cases are exported to EXAM.



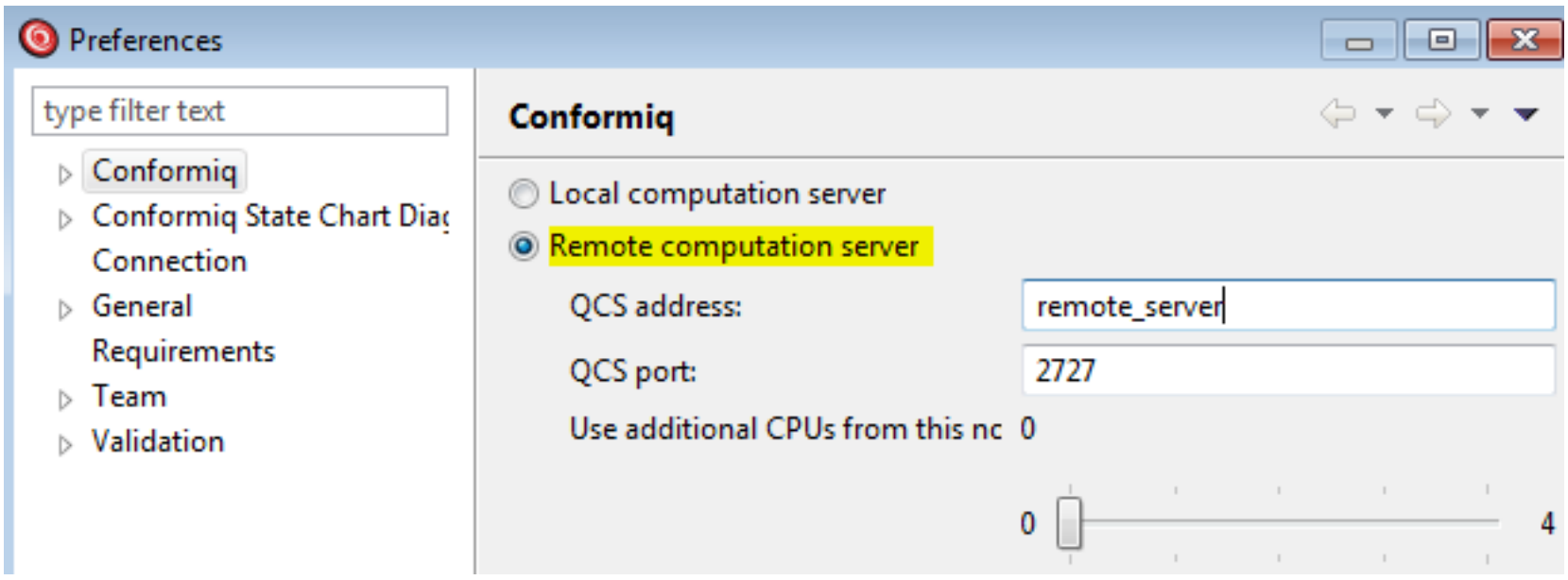
HTML-Export

The HTML export is done by *right-clicking* on a project or a *design configuration* in the *Test Case Steps View*. A detailed documentation on this export can be found in the Test Case Steps View chapter.



Remote Computing Server

Since the test case generation requires a high computing load, it can be transferred to a different computer (remote computation server). Settings are located under *Window → Preferences → Conformiq*.

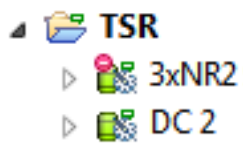


The settings for swapping to another computer must be executed **immediately after the start** of MODICA, since only in this way can it be ensured that there is no connection to the Computation Server. Existing connections can no longer be changed, a restart of MODICA would be necessary.

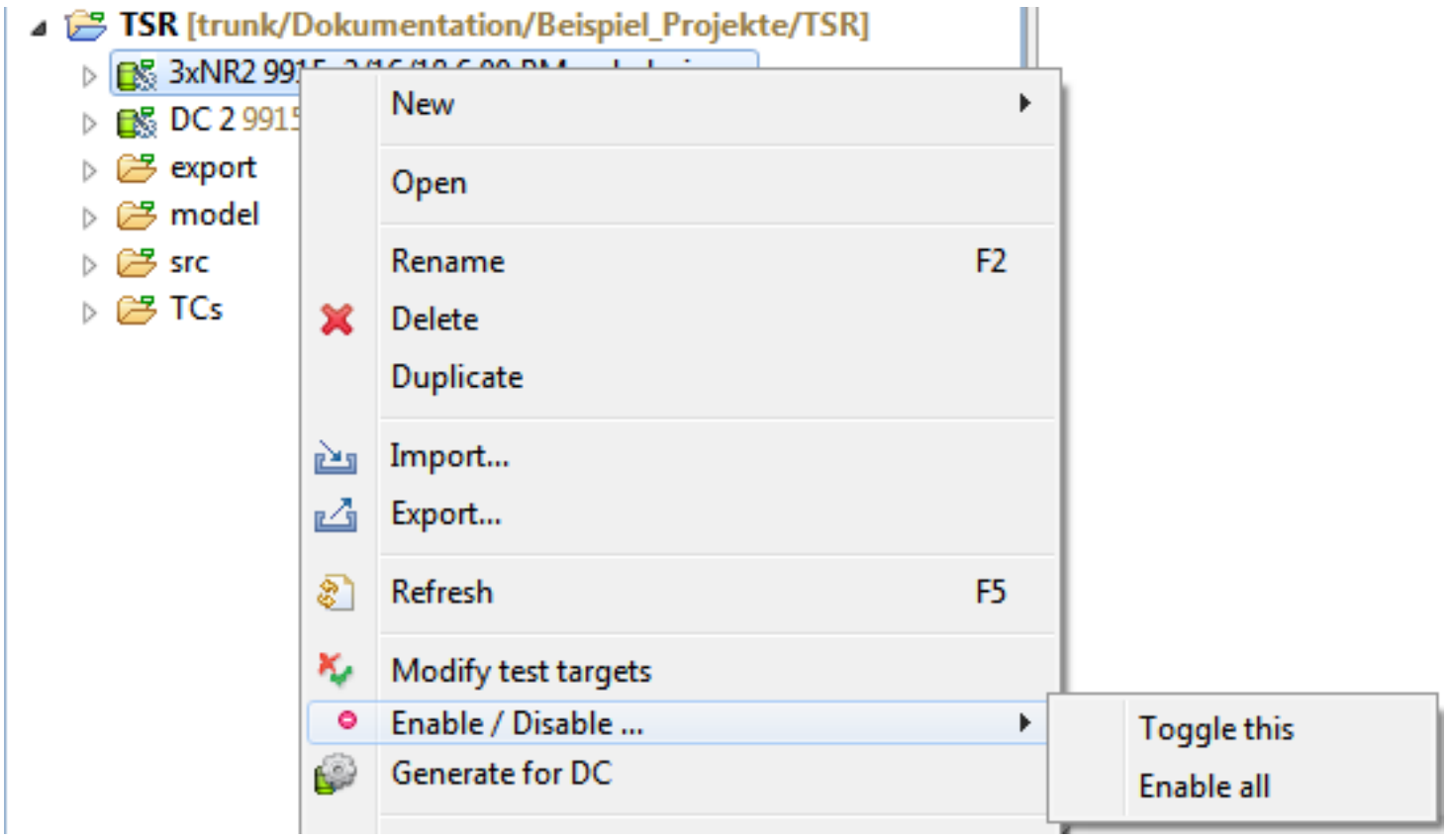
Generating tests for individual DCs

DCs can be deactivated for generation using the *Project Explorer*. In a DC's context menu *Enable / Disable* can be used for this purpose.

Deactivated DCs show a red dot in the upper left corner of their icon:



The context menu also offers to generate tests only for the selected DC. This deactivates all other DCs and starts the generation process.


TSR [trunk/Dokumentation/Beispiel_Projekte/TSR]
3xNR2 991
DC 2 991
export
model
src
TCs
New
Open
Rename F2
Delete
Duplicate
Import...
Export...
Refresh F5
Modify test targets
Enable / Disable ...
Generate for DC
Toggle this
Enable all

Sequence Rules

The test set to be generated should often reach a desired coverage but should remain compact in the number or length of test cases. This is an initially very abstract requirement which has a wide range of impacts and presents a challenge for the manual and automatic (or model-based) test case finding. In many cases, it may be useful to involve the domain knowledge of the tester and, for example, to provoke particularly interesting partial sequences in a more generic model. Such guidance of the generator is possible in MODICA via the sequence rules. This can be a useful tool for increasing the test depth, especially in a combination of model-based approaches and black-box testing.

The use of sequence rules is divided into three areas:

1. Define (= marking) model locations with *visit points*. In the sequence rules you can refer to these points. (Optional step)
2. Define *sequence rule*.
3. Configuration of the Generation (Assigning *sequence rules* to *design configurations*)

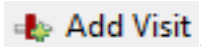
Basic behavior

To construct a *sequence rule*, you have to prescribe particular actions or model locations. Specifically, you can prescribe the next *trigger* or the next point marked with a *visit* in the diagram. In simple models it may already be sufficient to create the *sequence rules* based on the *triggers* that are contained in the model anyway. The optional setting of *visits*, however, means additional flexibility, especially if only partial sequences have to be specified.

In principle, *sequence rules* should not be created too early, because significant changes of the model (new / different *triggers* or *visits*, changing the paths, ...) may require changes to the *sequence rules*.

Optional step: Definition of visit points

Visit points can be placed at any point in the model (*states, transitions*). They are added to the *actions* from *states* (usually in the *EntryAction*) and *transitions*.

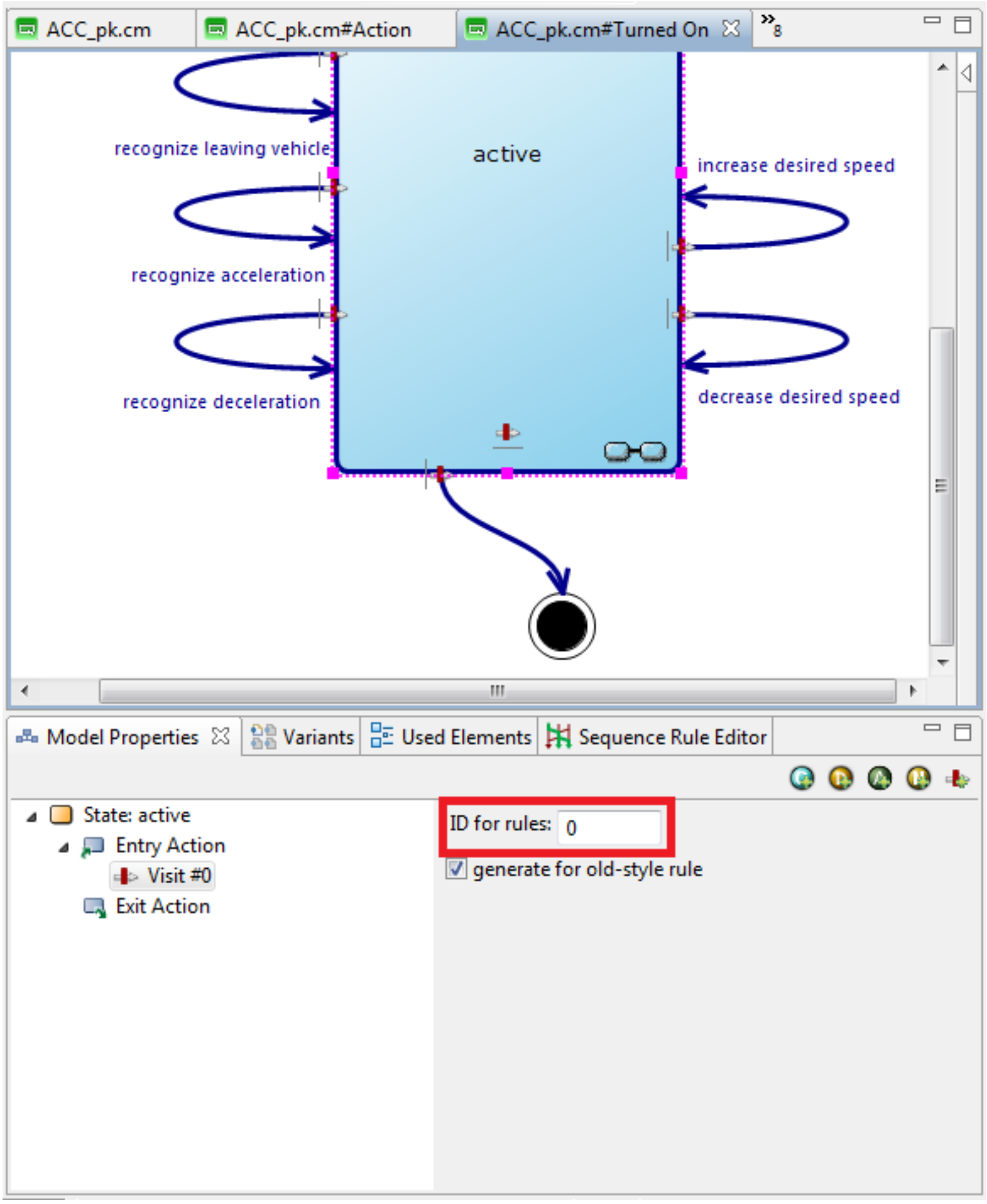
Visit points are created by selecting a *transition* or a *state* in the model, right-clicking on the appropriate action in the *Model Properties View* and clicking on 

. Otherwise, you can also use the button in the toolbar of the *Model Properties View*.

A new entry 

appears in the action area. The number (ID) is automatically incremented here, but can be changed manually.

If there are many *visits* in the model, it is advisable to choose the IDs of the *visit points* according to their own intuitive rule. In most models, however, very few *visit points* should be sufficient.

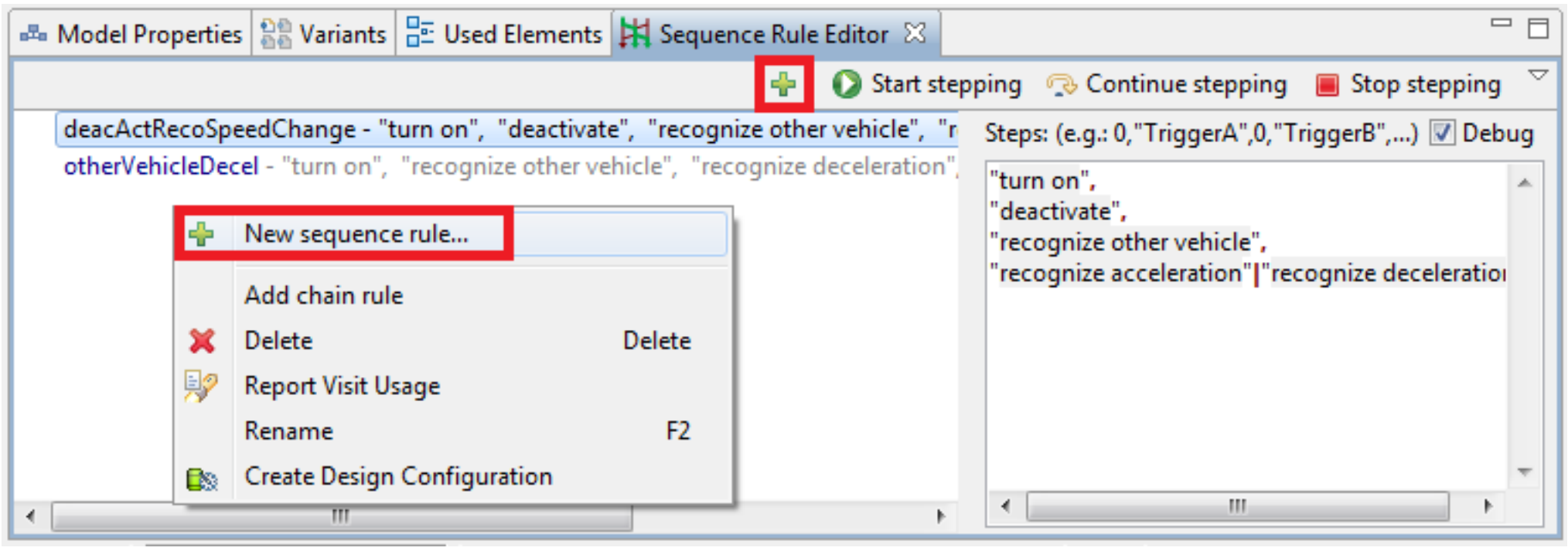


Defining sequence rules

Create a new sequence rule

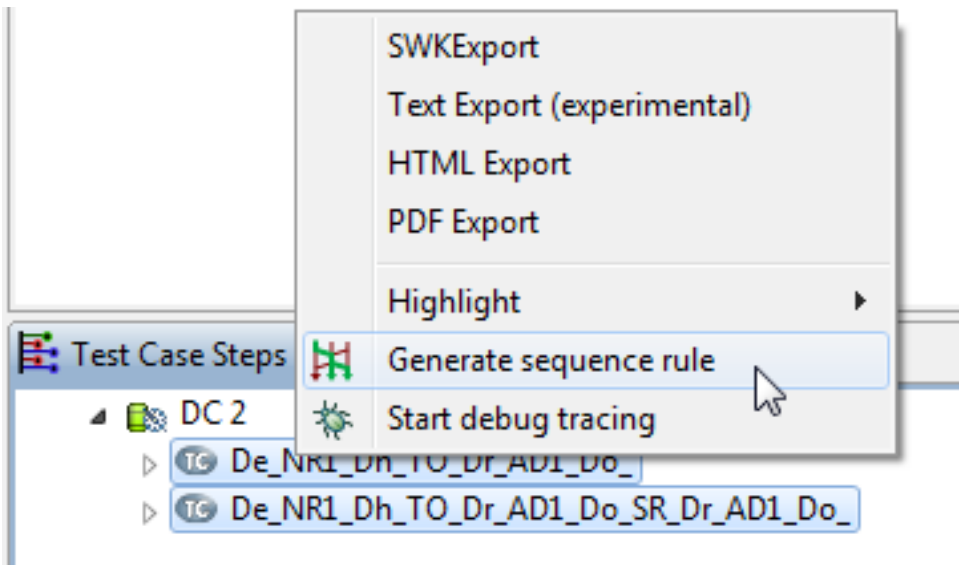
In MODICA, *sequence rules* are defined in the *Sequence Rule Editor*:

A new rule set can be created by *right-clicking* → *New Sequence Rule* or by using the button  in the toolbar.

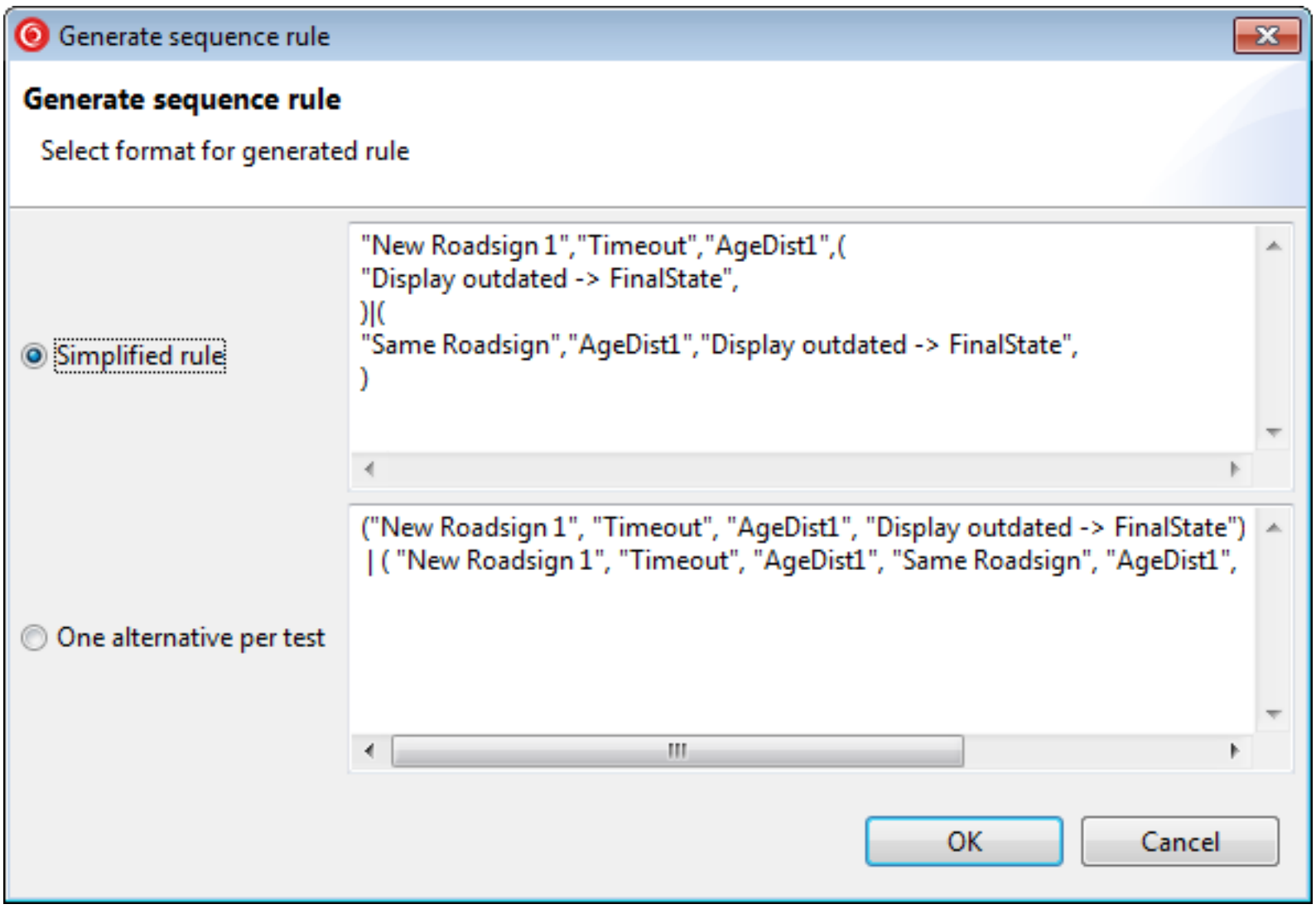


Generate sequence rules from existing tests

The easiest way to get familiar with sequence rules is to generate them from existing tests. This can be done using the context menu in the Test Case Steps View:



As a next step, a dialog appears with two different rule-generation styles which the user can pick: (The options are identical if just one test has been selected)



- The simplified style emphasizes the steps that the selected tests have in common, in the above example both begin with three identical steps and only after the brace, the different paths of the tests are expressed. This style is often more compact but may also be harder to read.
- One alternative per test is a more direct rule that lists all selected tests as alternative and for each of them the full sequence of triggers is expressed. This style is easy to understand but rules built from this can be hard to maintain due to their verbosity (The same elements appear many times).

When confirming the dialog, a sequence rule is generated. It can be changed manually or used as-is to limit the generator to the selected tests.

Manually building rules (and understanding them in-depth)

Basic constructs

The actual rule can now be formulated under *Steps*. The two basic constructs are

- Specify the next **trigger** by means of the *trigger* text: `"Trigger"`
  Meaning: The next *trigger* must be this trigger. (=> any number of *visits* are allowed)
- Specify the next **visit** using the visit ID: `917`
  Meaning: The next *visit* must have exactly this ID (=> any number of *triggers* are allowed)

Connections

A rule can be linked with additional syntax:

- A **concatenation** (sequence) of *triggers* and / or *visits* by comma,
  Meaning: All elements have to occur in the given order. Other elements **between** them may appear depending on some rules:
  (Syntax: `element_before_the_comma , element_after_the_comma`)
  - if the element_after_the_comma is a *trigger*, any *visit* is allowed after the element_before_the_comma.
  - if the element_after_the_comma is a *visit*, any *trigger* is allowed after the element_before_the_comma.

  Example: `"A", "B", "C"` writes the sequence of three *triggers* A, B, C - it is not allowed that any other triggers are

generated between them. Visits are allowed.
- An **alternative** of *triggers* and/or *visits* using Pipe `'|'`
  Meaning: all elements separated by pipes may occur
  For example, `"A" | "B" | "C"` next writes a trigger that must be either A, B, or C. Other triggers are not allowed, but visits are allowed.

In concatenation and alternatives, *visits* and *triggers* can be combined.
For more complex rules, both linking elements are often needed. The rule `"A", 0, "A" | "B", "C"` is interpreted as follows:

- First a trigger `A` is prescribed
- Then visit `0` is required
- Then one of the two triggers `A` or `B` is prescribed (alternatives have priority before concatenation - analogous to the rule "point before line")
- Then the trigger C is prescribed

⚠ For experts

Not any combination of alternatives and concatenation is supported. In some cases, the step *Load model* will detect a corresponding error (message: "Failed to read use cases: [...]"):

- Alternatives must not be directly following one another. → `0|1,0|1` is not permitted
- Several alternatives must not end at the same time. → `0|(2|3)` is not permitted

However, these problematic constructs can be easily transformed into valid rules. `0|(2|3)` can also be represented as `0|2|3`, for example. In other cases, the appointment of an additional *trigger / visit* can help quickly. `0|1,0|1` would thus be depending on the model something like `0|1,"Trig",0|1`. Moreover, `(0,1)|(0,0)|(1,0)|(1,1)` would be possible, too.

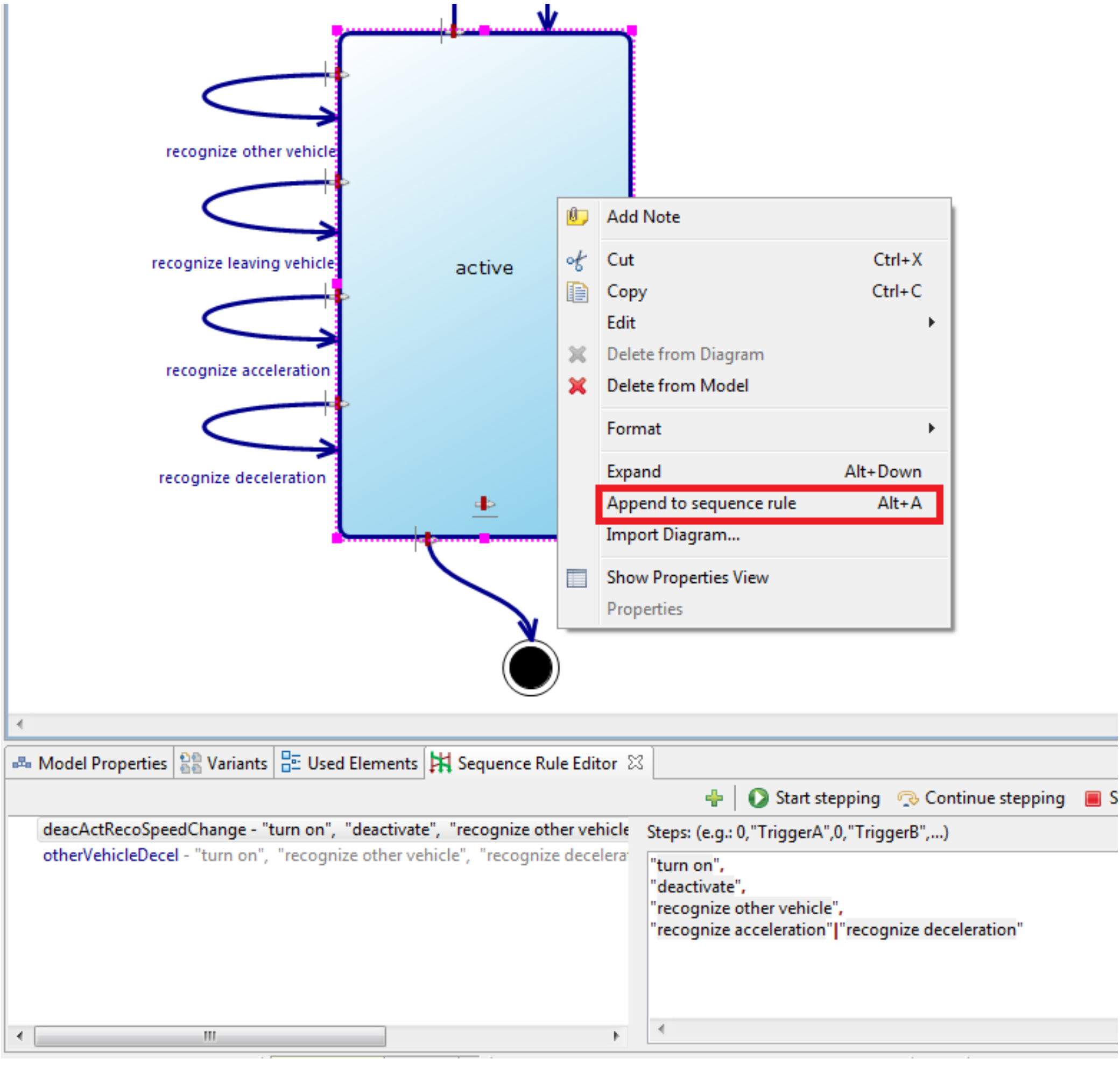
Sequence rule creation using the diagram

Elements can also be directly added to a *sequence rule* via the diagram. To do this, *right-click* on the corresponding element in the diagram and choose *Append to sequence rule*.

If no sequence rule is selected in the *Sequence Rule Editor*, a dialog box appears asking the user to which Sequence Rule the element should be attached. If a sequence rule is already active in the Sequence Rule Editor, the element is immediately appended at the end of this *sequence rule*.

Note:

- There must be at least one *sequence rule*
- A *tranisiton* can only be added if it is triggered
- A *state* can only be added if it contains a *visit*



Readability and comments

In particularly complex cases or for better readability, it may be necessary to nest alternatives and concatenation with round brackets. For example, the rule 0,1 | (2,3 | 4), 0 allows exactly three sequences of *visits*:

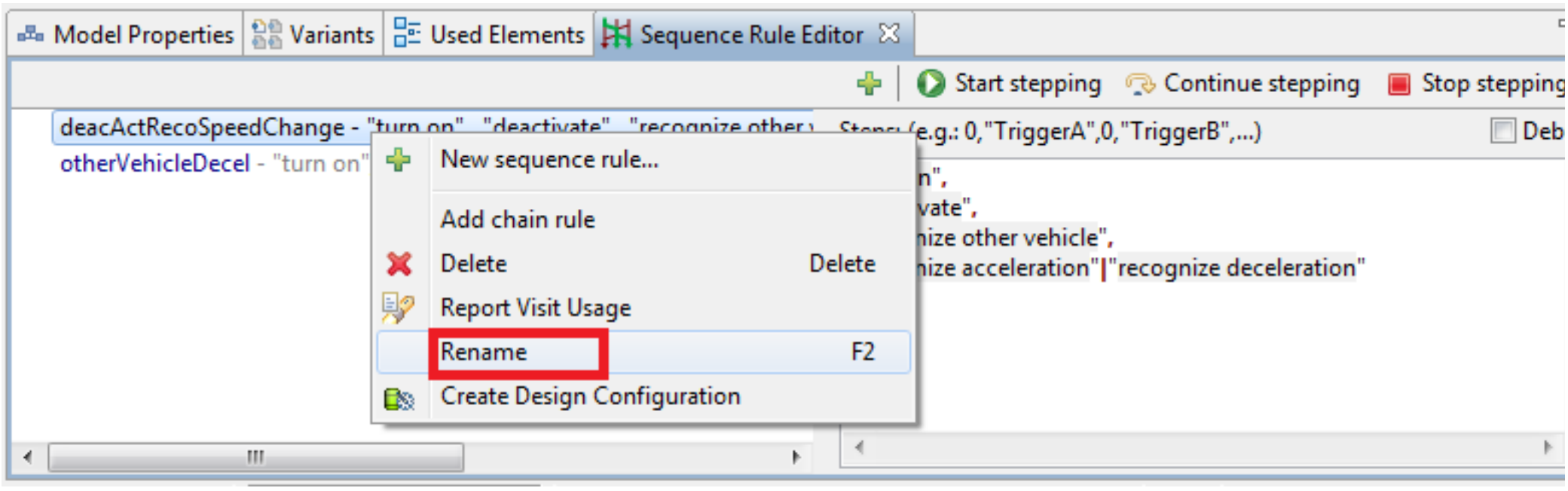
- 0,1,0
- 0,2,3,0
- 0,2,4,0

For better comprehension, rules can also be distributed to multiple lines. Line breaks, spaces, and tabulators are allowed between all elements and are internally ignored. It is also allowed to write comments. The syntax is identical to Java: `/ * * /` and `//`. If the first line contains a /* */-style comment, it will be displayed next to the name of the sequence rule.

Rename sequence rules

A *sequence rule* can be renamed by *right-clicking* → *Rename*. The following rules must be observed:

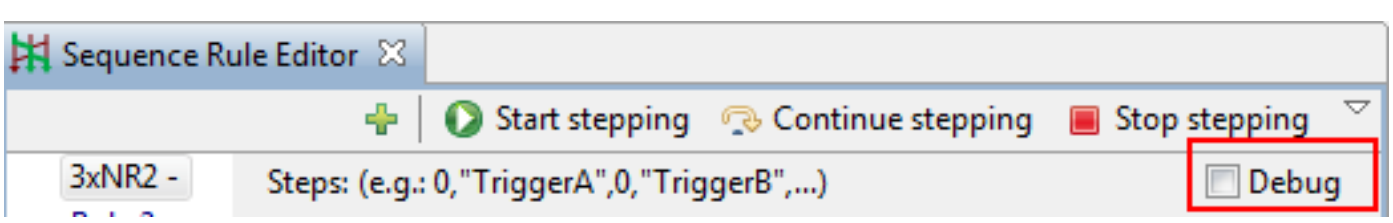
- a name must begin with an alphanumeric character
- Only alphanumeric characters, spaces, bindings and underscores as well as pipes (, | ') are allowed
- a name can not end with a space
- a name must be unique
- the new name must be different from the original and can not be empty



Debugging sequence rules

⚠ For experts

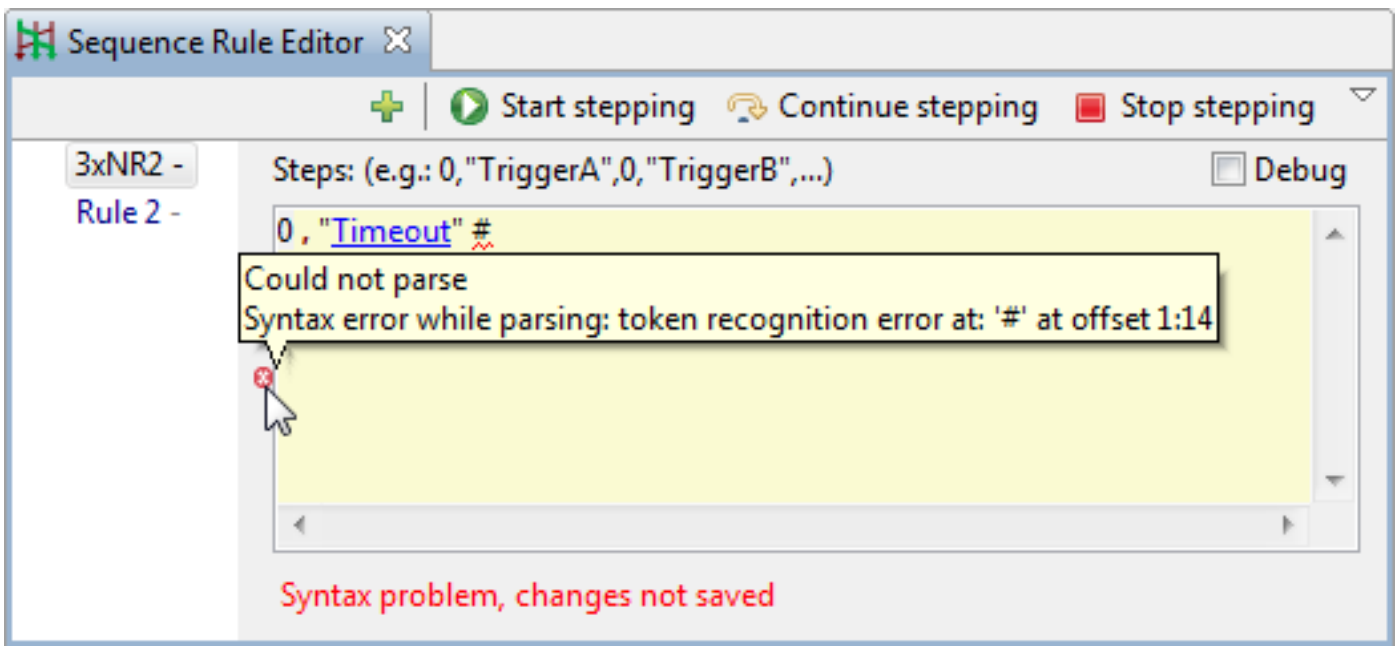
If test sequences without the *Only Finalized Runs* flag are generated for debugging purposes, the use of *sequence rules* may result in fewer partial test cases than expected (often 0), since the semantics of the *sequence rules* basically prescribes that each test case must reach the end of the rule. In order to be able to generate test cases which only cover parts of a *sequence rule*, the checkbox *Debug* can be enabled in the *Sequence Rule Editor*. This function should only be used in conjunction with unchecking *Only Finalized Runs under Project → Properties → Conformiq Options → Only Finalized Runs*.



Validation

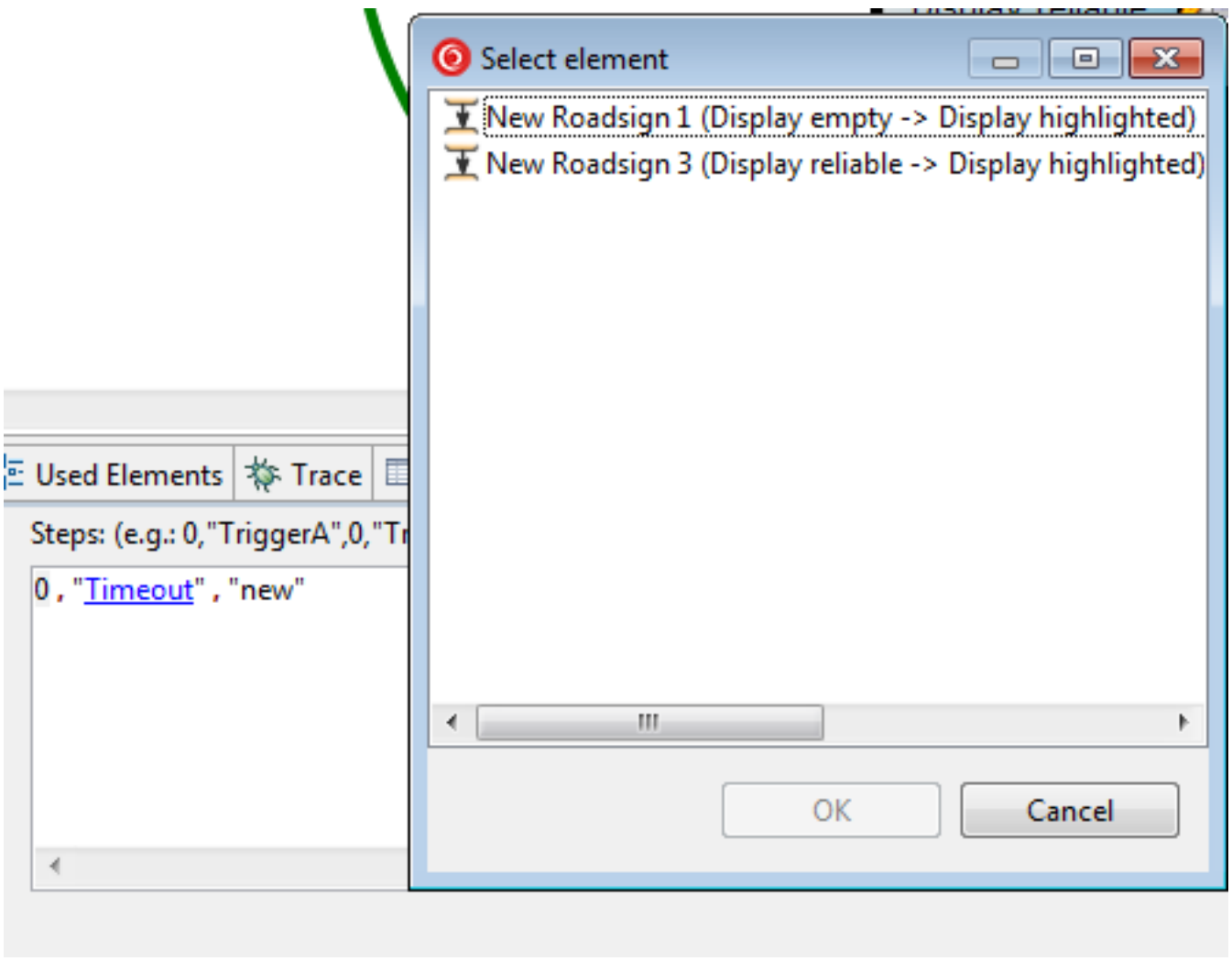
Syntax check

The syntax is checked directly while typing in the editor field. As soon as an invalid expression is detected, a red cross appears at the left edge of the editor field. The tooltip on the cross gives hints about the type of error.



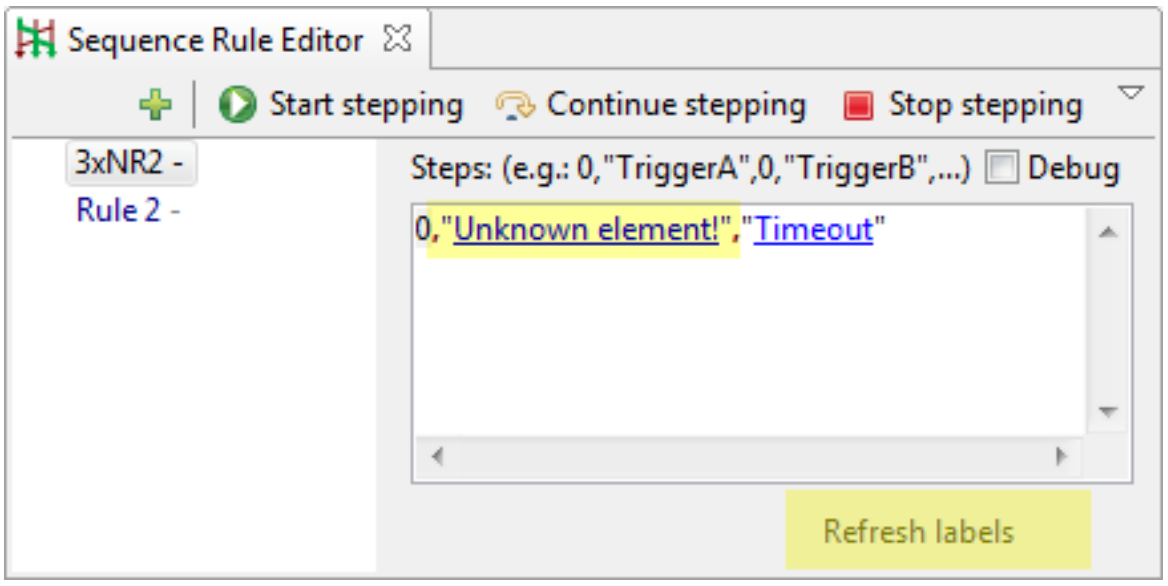
Element check

If the syntax is valid, the sequence rule is automatically saved. If it contains any unlinked "Trigger" references, the project is searched for matching transitions that have a trigger. The transition labels are the main criteria for this search. If no unique matching is found, a dialog is shown to ask the user to perform the matching manually:



After matching has been done, only the link target will be stored in the project. Upon re-loading the sequence rule (or when Refresh labels is used in the context menu), the link-text is updated according to the current transition label.

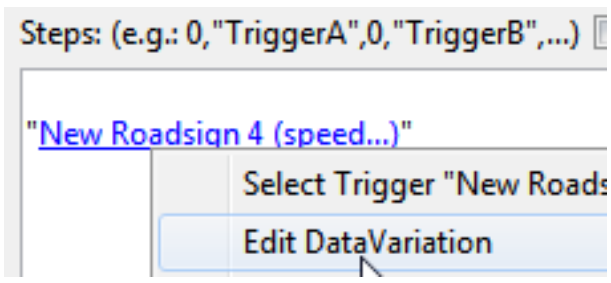
When the target of such a link is lost (e.g. because the transition has been deleted), the link will be displayed as *Unknown element!* and must be fixed. Missing link targets also result in problem markers from the project validation.



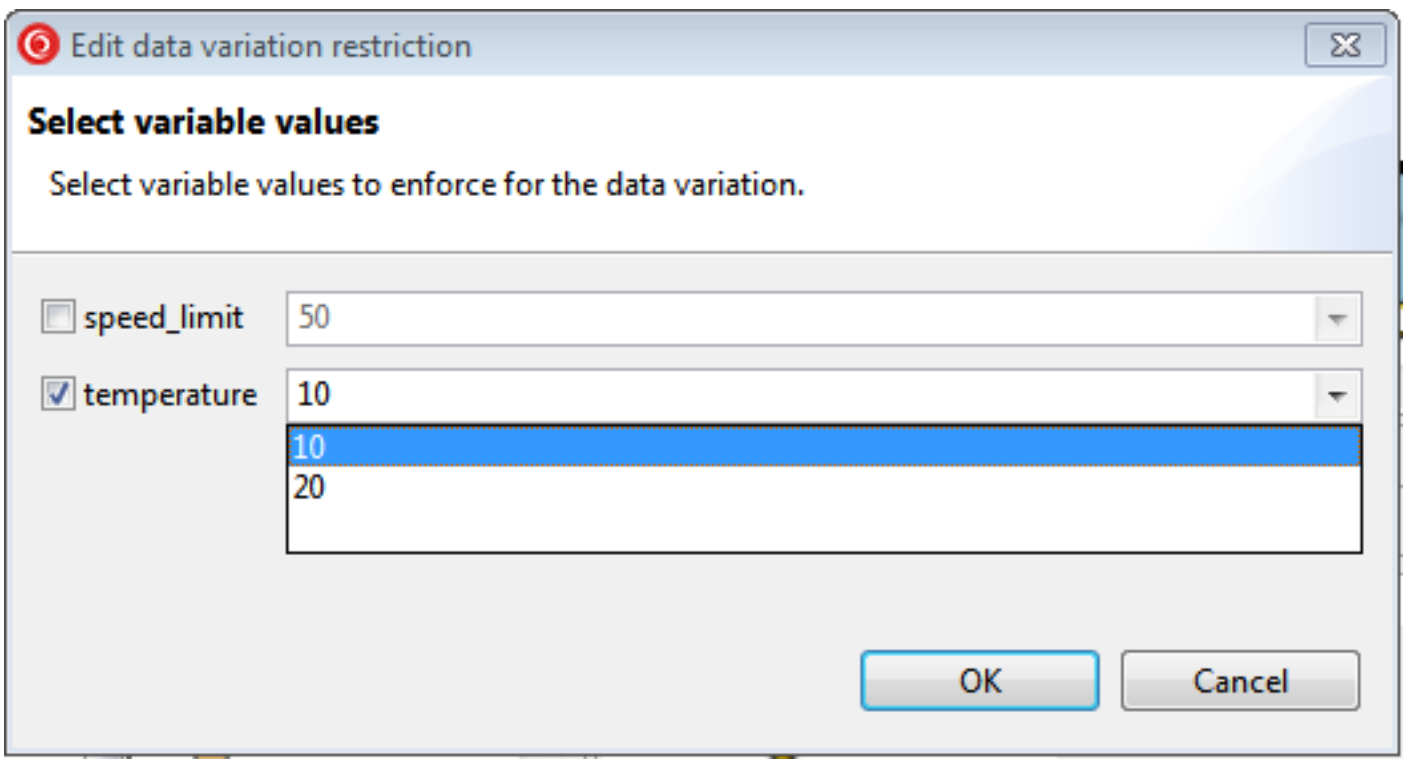
If the project contents have changed significantly since a sequence rule has been opened, the text *Refresh labels* is shown. It can be clicked to refresh outdated link-texts in the rule.

Restricting data variations in sequence rules

Triggers with associated data variations can be restricted in sequence rules.This is done in the context menu of the sequence rule editor:



A dialog is then shown that allows to configure the restrictions:



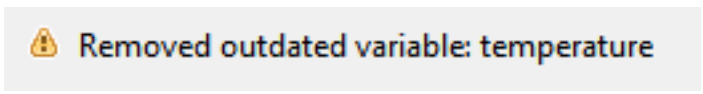
Alternatively, a sequence rule with data variation restrictions can also be created using the create sequence rule features (from *Test Case Steps View* and *Trace View*).

Illegal configurations

In rare usescases, it is possible to manually configure non-deterministic sequence rules with this feature. When choices (using |) are included, care must be taken that the different parts of the rule do not overlap. E.g. for a Data variation with two variables, the following would be illegal:

```
"trigger (a=1)" | "trigger (b=10)"
```

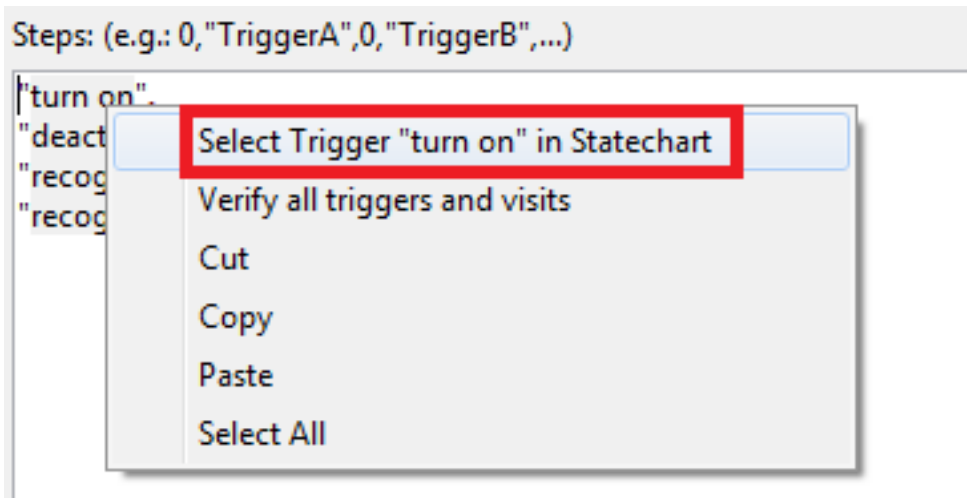
Similarly, changing variables or data variations may make sequence rules become invalid. In this case, the above dialog will show a warning on the next edit:



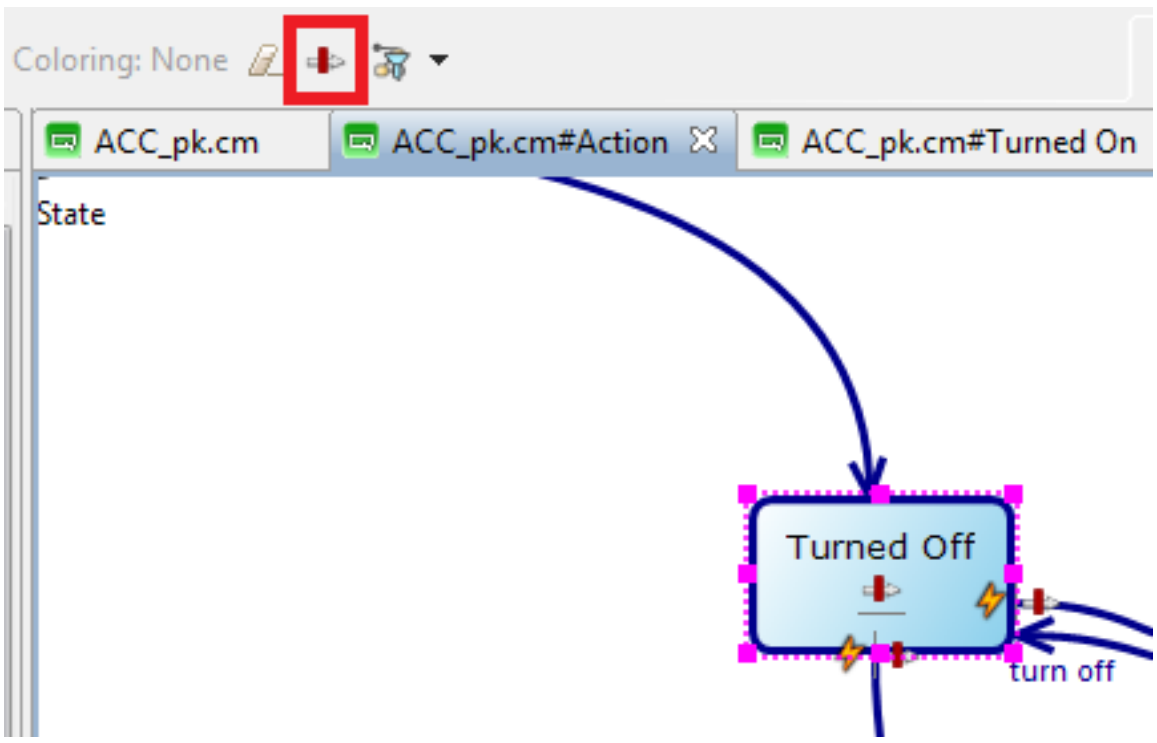
Additional UI support for sequence rules

There are some additional features for working with *sequence rules*.
By right-clicking on a *trigger / visit* → *Select Trigger/Visit in Statechart*, you can quickly display and select the appropriate diagram element. Three subsequent actions are possible:

- If the item was found, it will be highlighted in the *Editor*
- If more than one item was found, a warning is issued
- If an element is not found, an error dialog is displayed



In addition, the MODICA Toolbar contains a button that shows all visits in all diagrams:



Highlighting

The highlighting allows a clearer overview of the individual steps of a *sequence rule*.

The selected dark green State I contains the Visit 0 (auto selection of the *Sequence Rule Editor*).

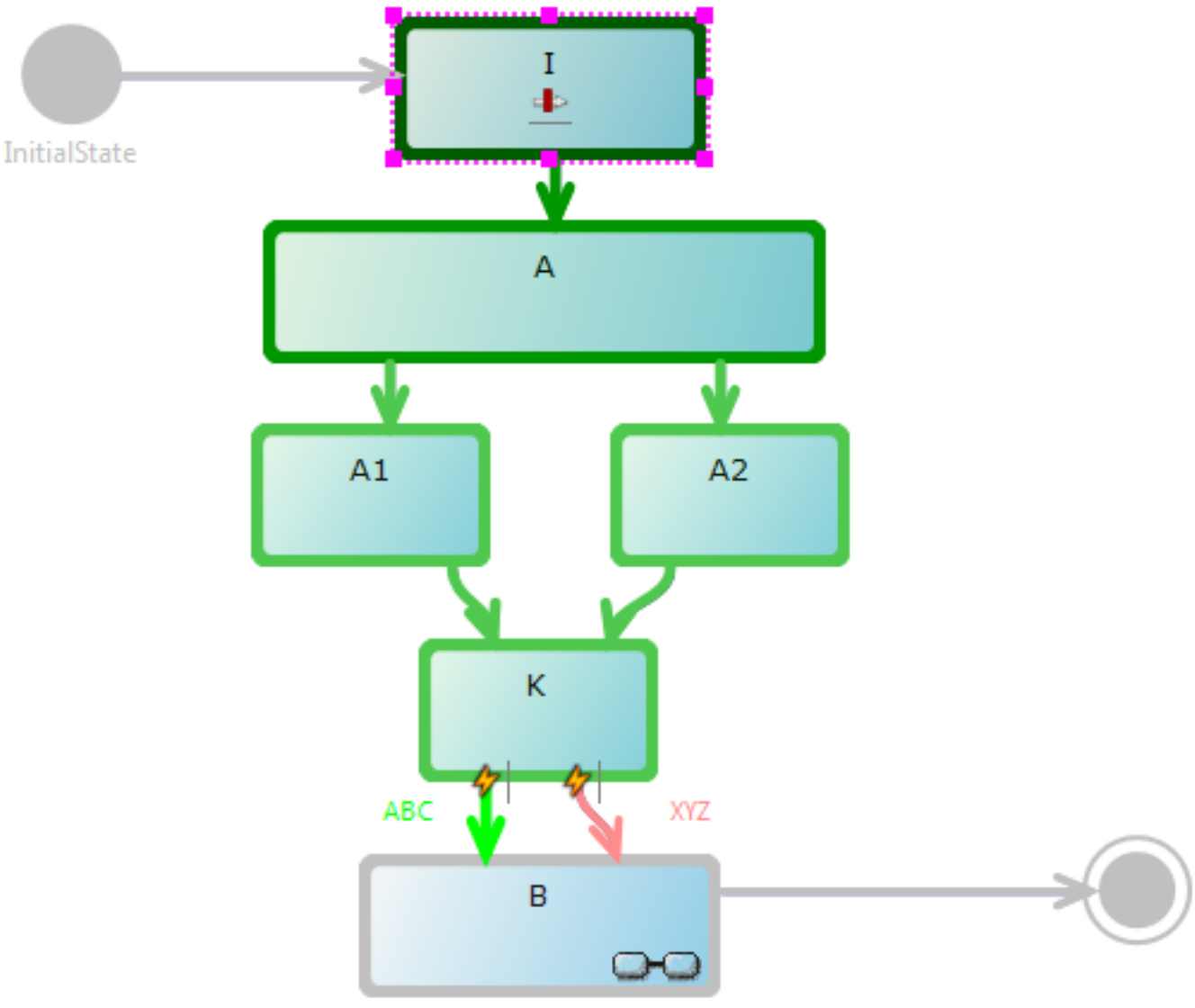
The medium green *state* A (as well as the incoming transition) follows **immediately** as the next step.

The light green *states* A1, A2 and K (as well as the different *transitions*) are **possible subsequent** options.

The neon green *transition* ABC shows that this element is the **required next step** of the sequence rule.

The red *transition* XYZ shows that this trigger or visit is **implicitly forbidden** by the sequence rule.

All other *states* and *transitions* are gray.

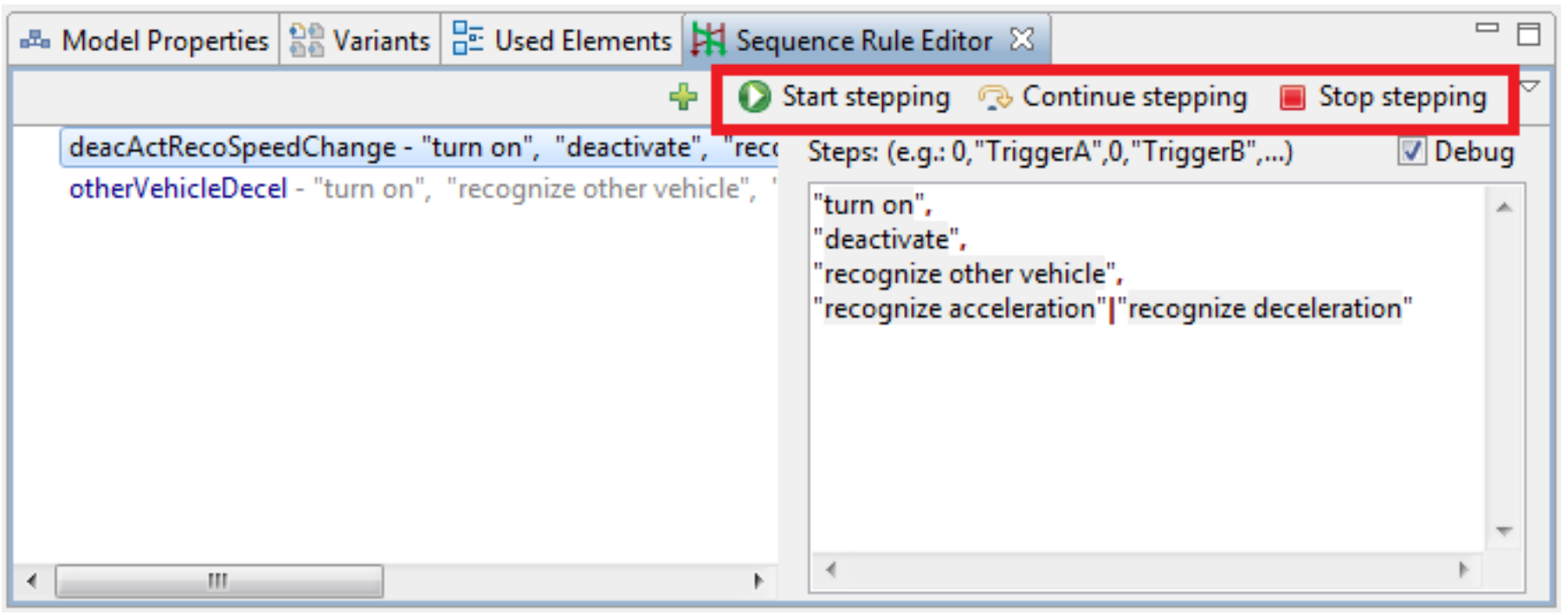


The legend of the individual colors appear as a tooltip of *Coloring [...]* in the main toolbar. If you click the eraser-icon on the right, you leave the stepping mode and the highlighting is removed from the statechart.

Stepping

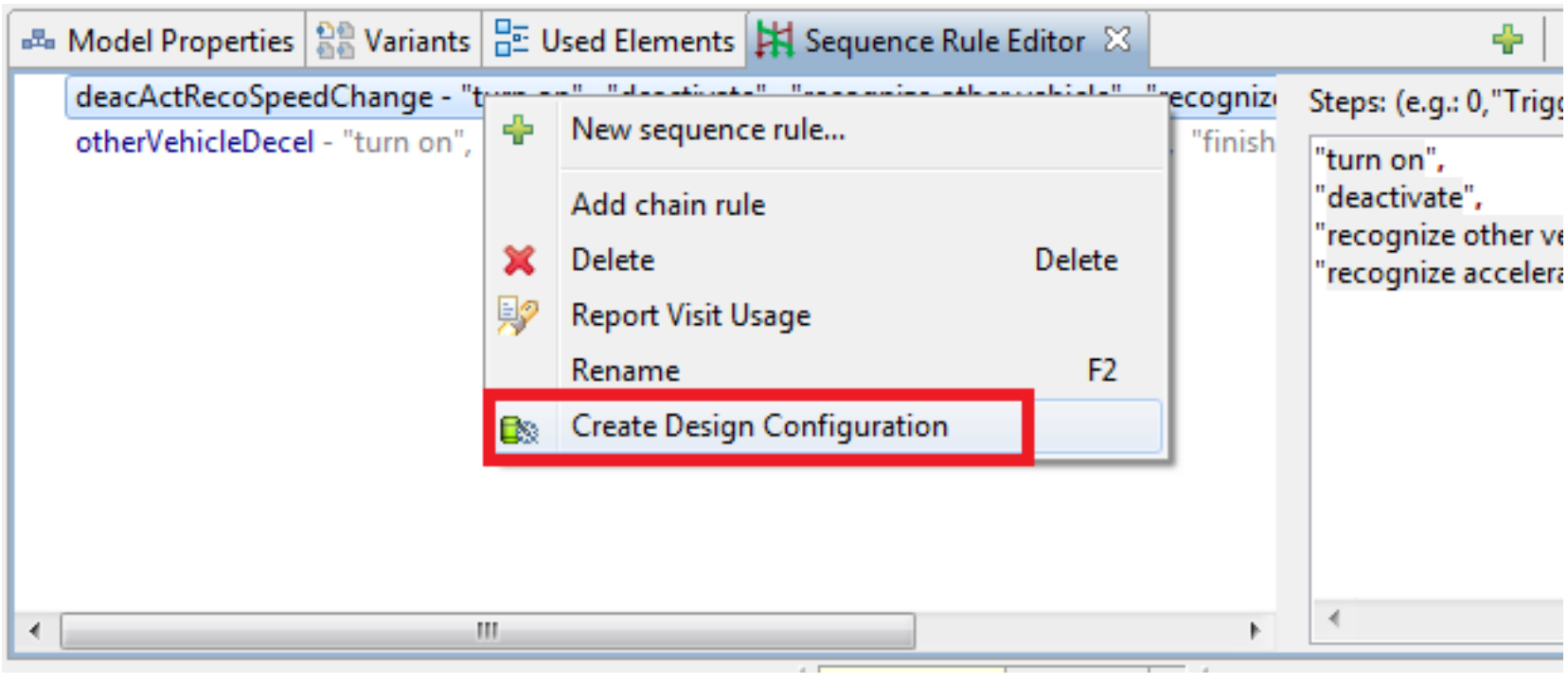
The various steps can be displayed with the 3 buttons of the *Sequence Rule Editor*. The individual steps are marked in color as described above.

- *Start stepping*: The selected sequence rule is started.
- *Continue stepping*: Jumps to the next step of the sequence rule. If more than one step is possible, a dialog window opens in which you can select the desired next step.
- *Stop stepping*: Terminates the stepping.

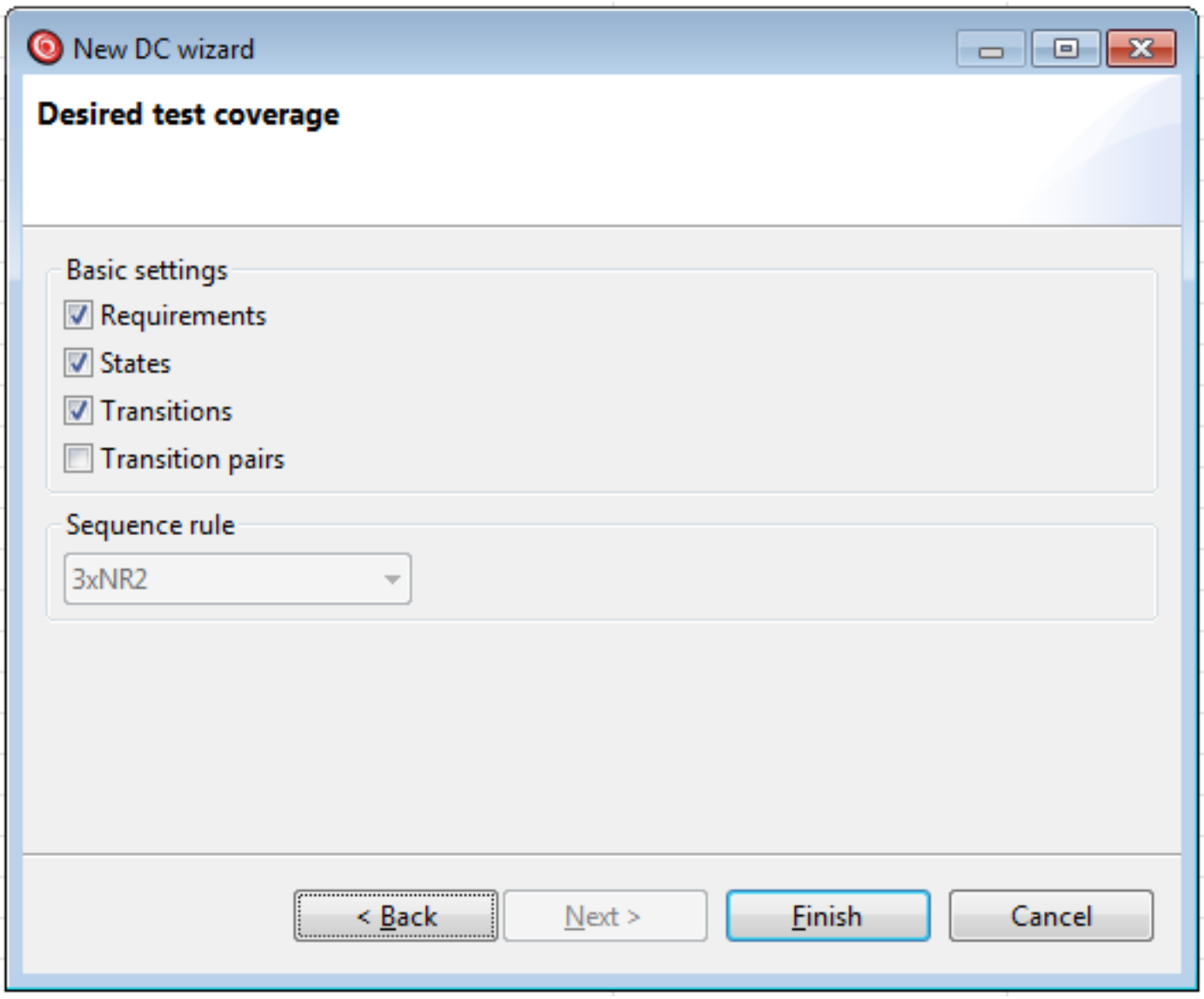


Configure Design Configurations with sequence rules

A new *design configuration* is created via *right-click* on the *sequence rule*.

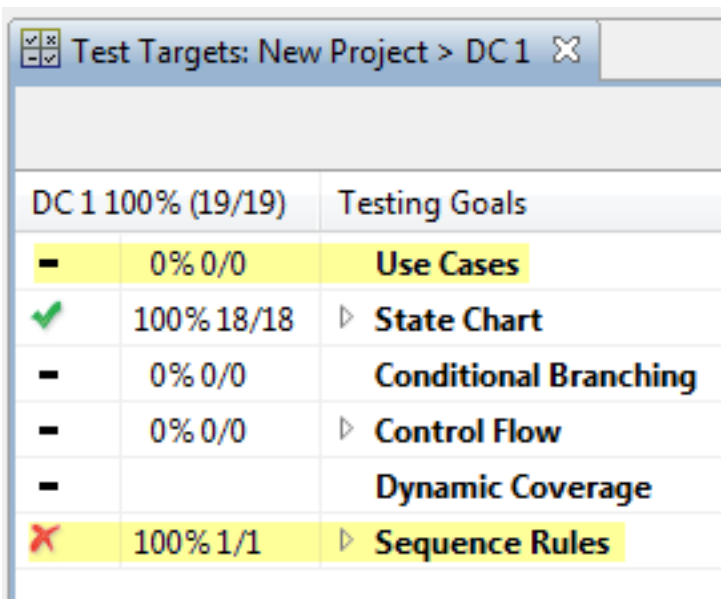


In the appearing *New DC wizard* dialog, the name of the *sequence rule* is already entered as the name of the *DC configuration*, and the test destinations are already preselected accordingly.



Sequence rules, Create new design configuration - settings

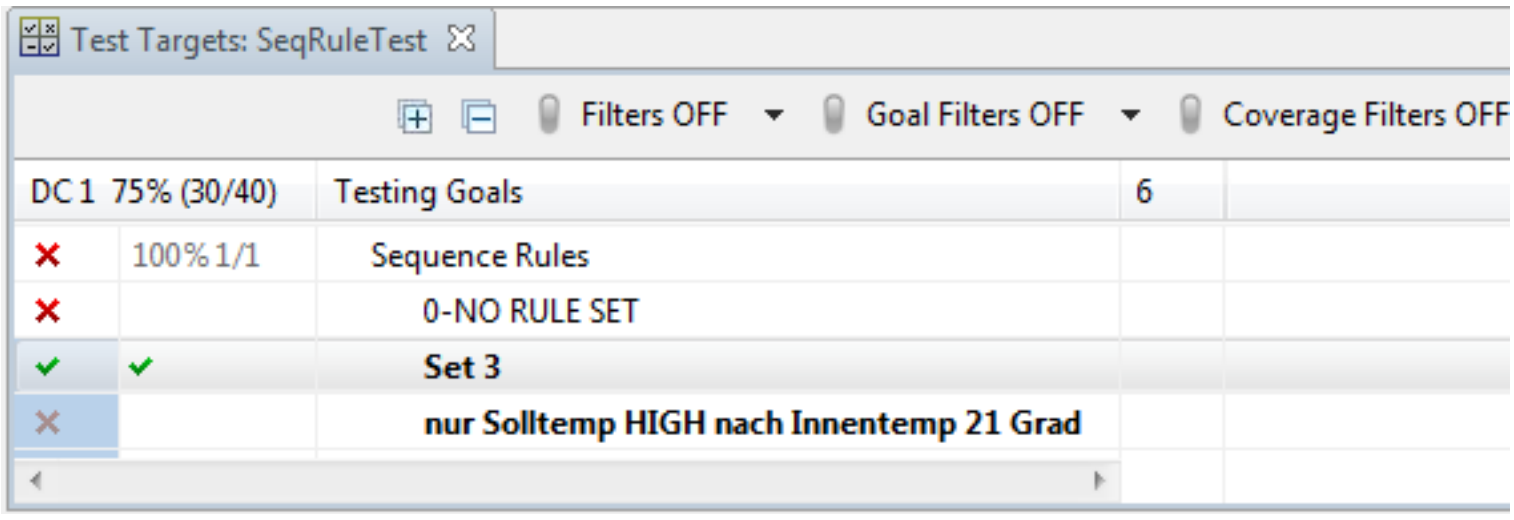
As soon as at least one sequence has been stored in a model and the model has been compiled (*Load model*), a new sequence rule section appears in the *Targets View*.
This area displays all the *sequence rules* defined in the project.
In principle, projects with *sequence rules* should be taken to ensure that the internal test goal *Use Cases* is always on *Do not Care.* In addition, the *sequence rules* main point should be set to *Block* for each *design configuration*.



Activation / deactivation of sequence rules

Only one sequence rule in a particular *design configuration* should be set to *Target* in the *Test Targets View*.

Example:



Here the main point *Sequence Rules* is marked with *Block* as described above. If no *sequence rules* should be used in a *design configuration*, although these are defined, *0-NO RULE SET* must be set to *Target*. If, on the other hand, a rule set for a *design configuration* has to be specified, only this has to be marked with *Target*. There should never be more than one sub-node of *Sequence Rules* marked with *Target*. In the example above, only test cases are generated for the *design configuration* DC 1, which follow the *sequence rule* with the name Set 3.

Validation of the Modeling

In MODICA, inconsistencies and modeling errors can be detected and displayed to the user before loading the test case generation model.

This validation is often done automatically, eg. when modifying the model and saving it.

Carry out the validation

Validation can be done in several ways:

Where	How	What is checked	Kind of error message
Main Toolbar	Click on the  Icon	whole model	message in the *Problems View*
Project Explorer	Right-click on *Project -> Validate*	whole model	message in the *Problems View*
Sequence Rule Editor	Right-click in the editor -> *Verify all triggers and visits*	Syntax and elements of a sequence rule	error dialog
Model Properties View	edit field of *Naming Fragments, Guards, Descriptions* und *Calculations*	Syntax in the edit field	Yellow coloring of the edit field

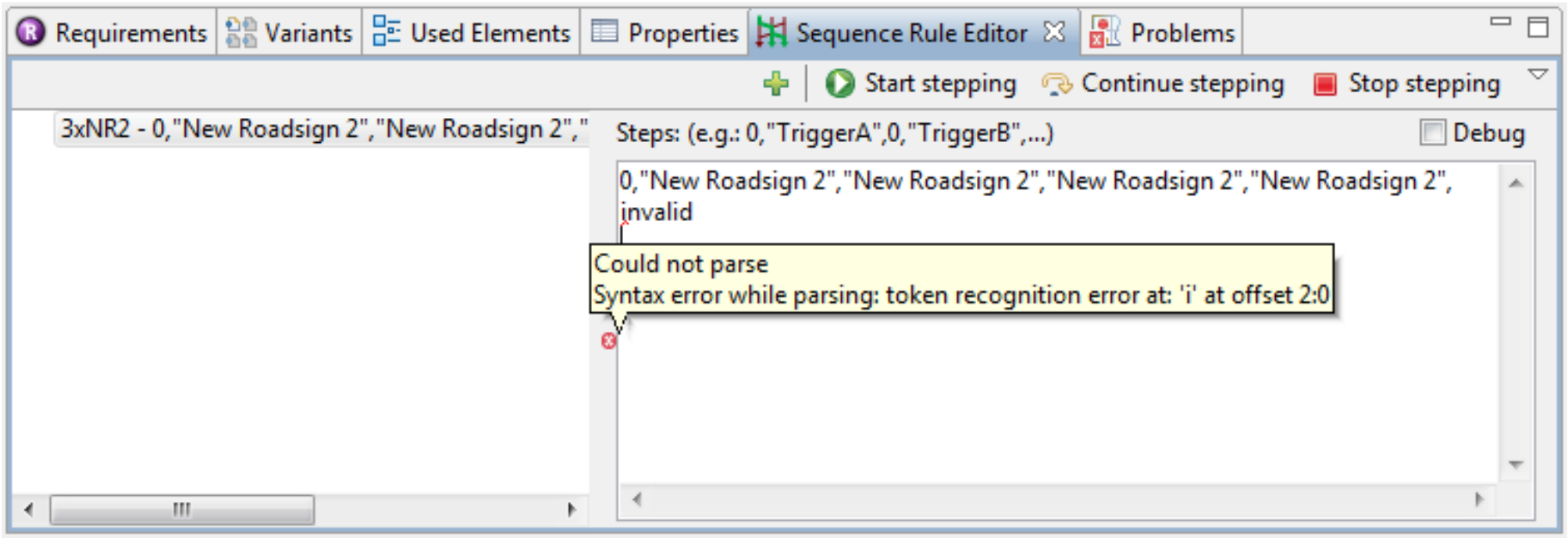
Sequence Rules

Syntax checking

The syntax is checked directly while typing in the editor field:

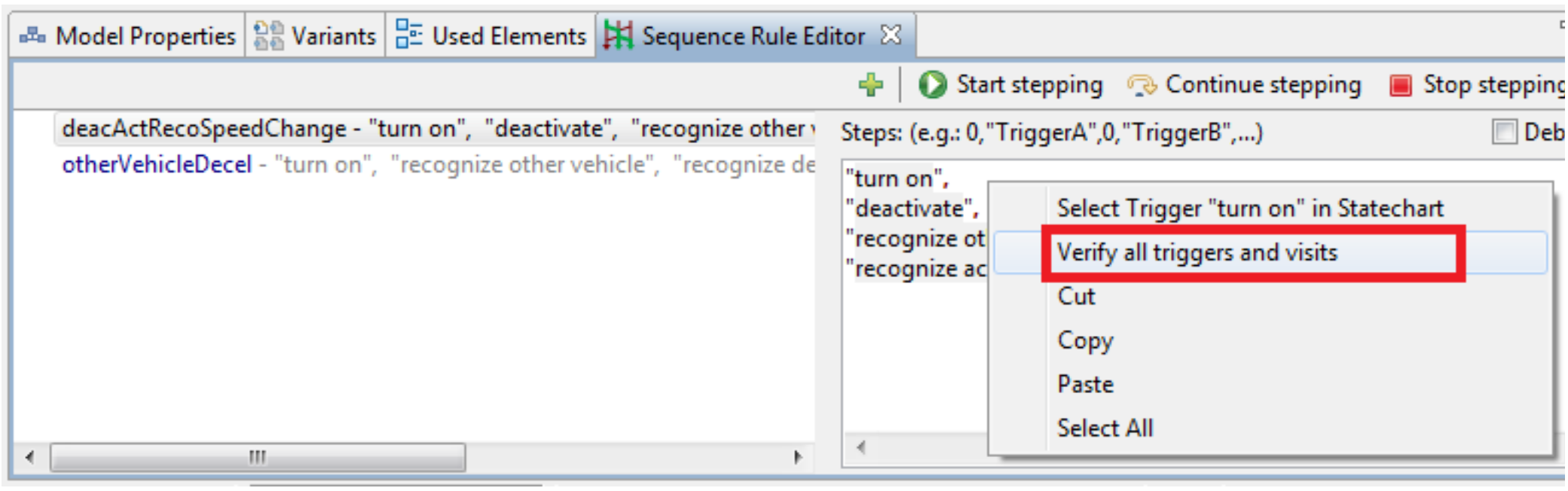
- Individual steps must be separated by commas (eg. `0, "Trigger1", 1`)
- Several alternatives can be separated from each other by `'|'` (eg. `"Trigger2" | "Trigger2"`)
- Steps are
  - Visits as Integers (eg. `0,1`)
  - names of triggers in quotation marks (eg. `"Trigger1", "Trigger2"`)

As soon as an invalid expression is detected, a red circle with a white cross on it appears at the left edge of the input field. The tooltip on the cross gives hints about the type of error.



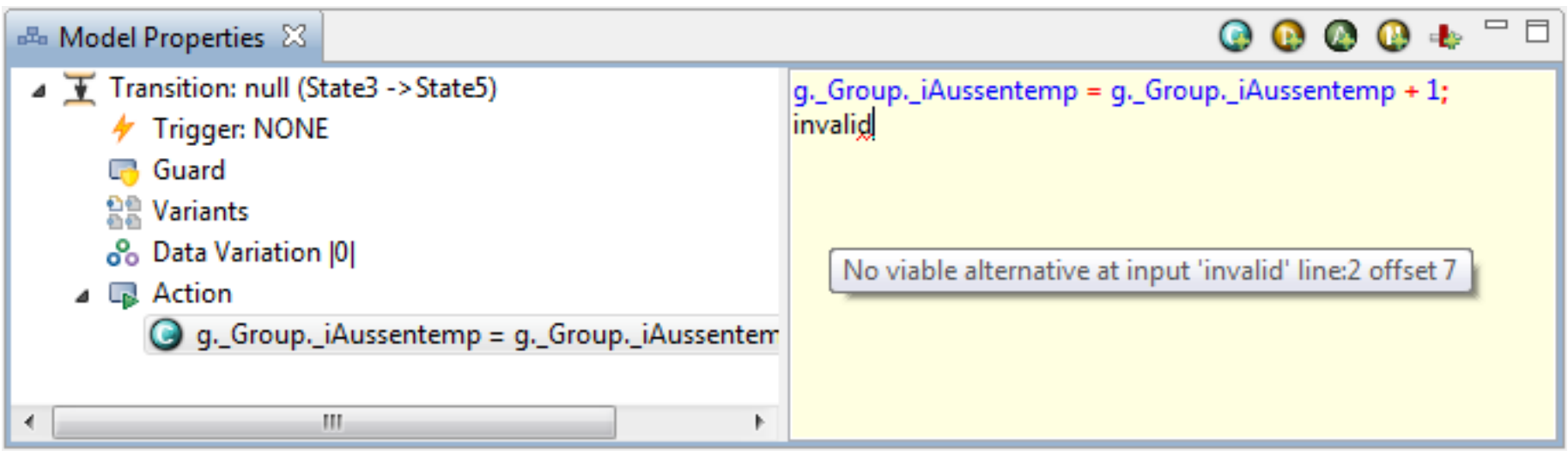
Element checking

If the syntax is correct, the existence of all *visits* and *triggers* can then be checked by *right-clicking* in the editor field → *Verify all trigger and visits*. The user is notified of the result of the validation via a dialog box.



Model Properties

In the *Model Properties View*, the syntax in the edit field for *Calculations, Guards, Descriptions,* and *Naming Fragments* is checked directly at the input. If an error occurs, the background of the edit field is colored yellow, the error in the code is red, and the tooltip displays a description of the error type.



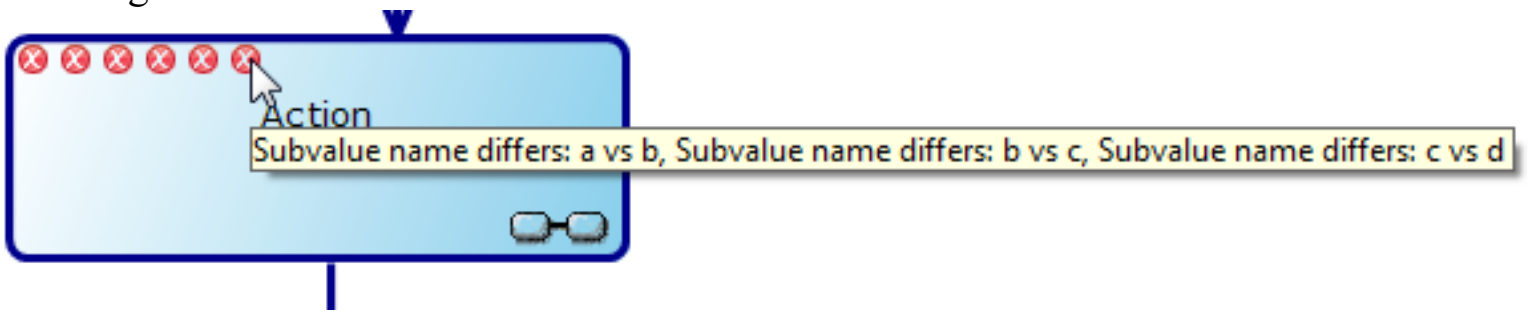
Visualization of the results

The results of the validation can be seen at the right bottom of the screen:

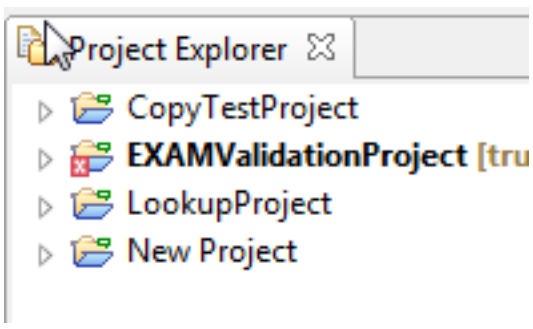
-  the project TSR contains at least one error
-  the project TSR contains at least one warning
-  the project TSR does not include any errors or warnings

Additionally, the results of the validation are displayed more detailled at various points in the *modeling* view:

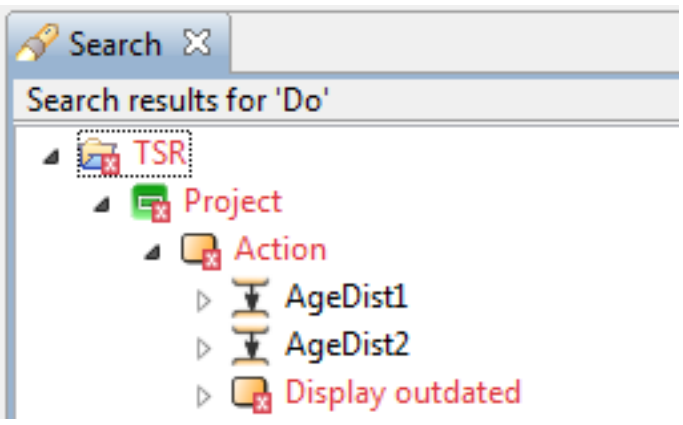
- In the diagram:



- In the *Project Explorer*:

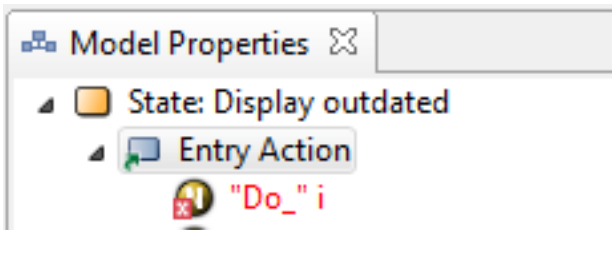


- In the *Used Element View*:



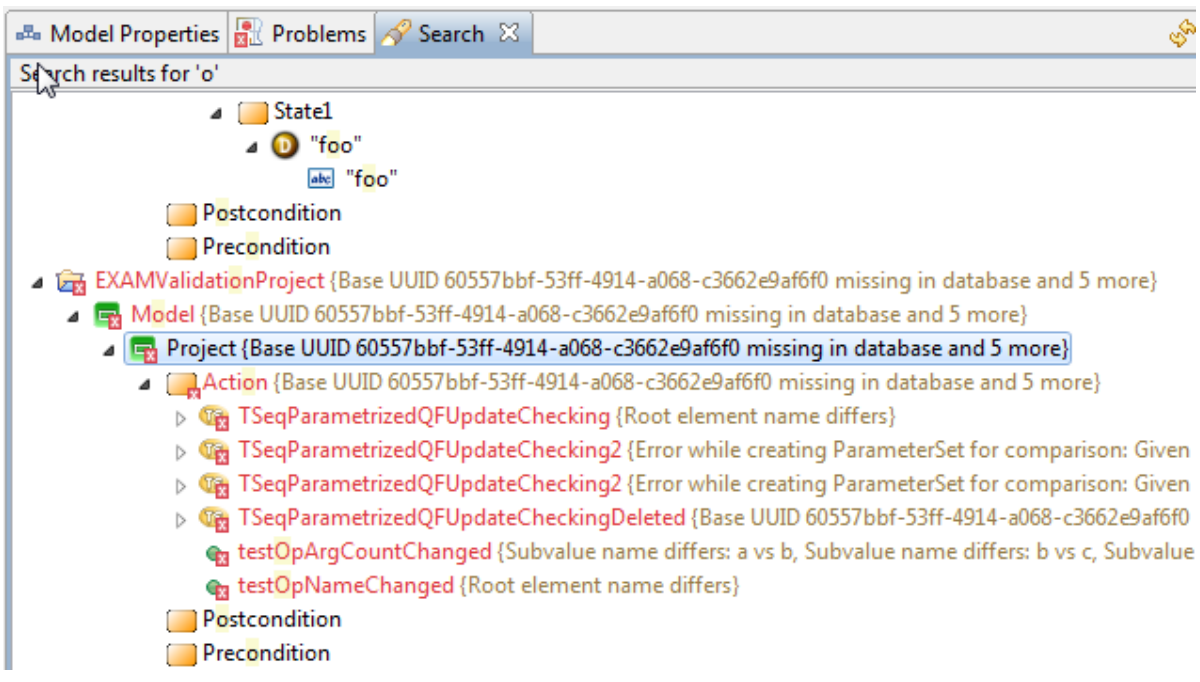
(Icon and color)

- In the *Model Properties View*:

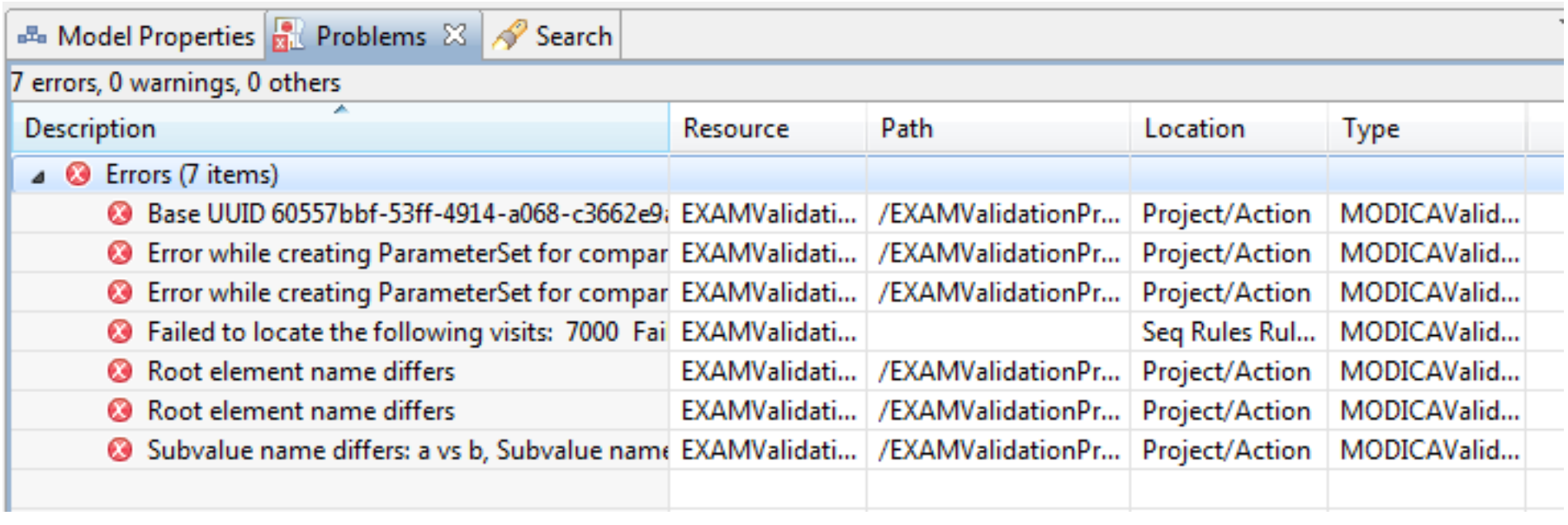


(Icon and color)

- Among the *Search-Result s*:



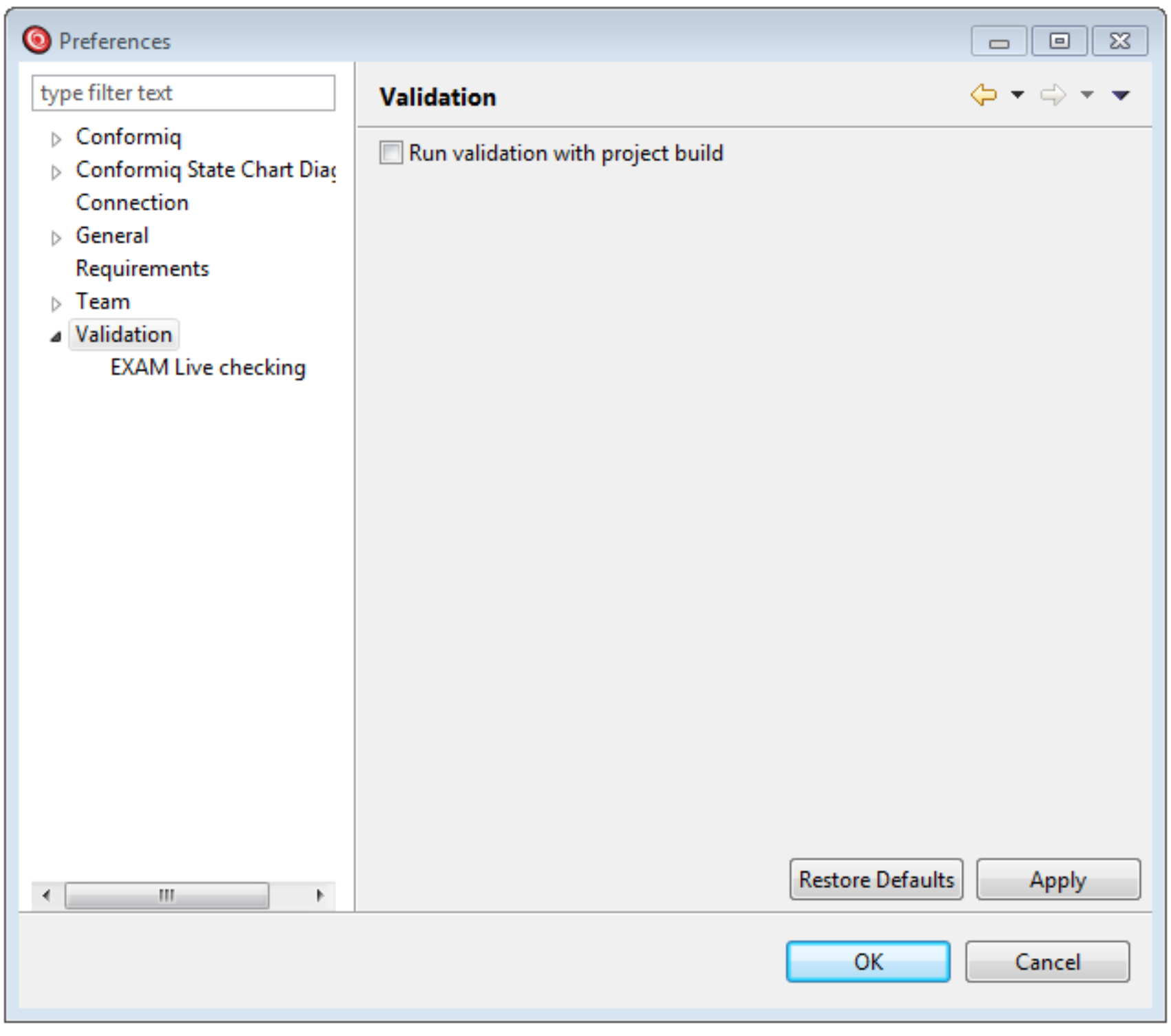
- In the *Problems View*:



Settings for validation

General

By default, the complete model is validated after each saved project change. If this is not desired, the function can be deactivated under *Window → Preferences → Validation* by unchecking the hook for *Run validation with project build*.



Messages in the Problems View

The most common messages in the problem view are explained in the below table with possible solutions.

Trigger used multiple times: *Trigger1*	Warning	In a project the same name of a trigger was used several times. The name of a trigger should be unique within a project. (They are always uniqe, when [auto] triggers are used)	Rename one of these triggers
State with mix of triggered and untriggered outgoing transitions	Warning	Outgoing transitions from a state are partially trigged. All outgoing transitions of a state should have a trigger if one outgoing transition has a trigger. (This may be ok for some models)	Consistent allocation of triggers
State name used multiple times in one diagram: *State1*	Error	In a subdiagram, two states have the same name.	Rename one of these states
Visit appears in exit action	Warning	In a state, a visit has been assigned to an exit action. Visits are usually assigned to the entry action. In special cases, however, the exit action can be explicitly desired as a visit goal. Whether this is really the case should be rechecked before.	Possibly move the visit to the entry action
Load Model failed: Reading model part (C:/.../*.*cm*) **failed (program internal error)**	Error	The project file * .cm is defective or damaged	Syntax check of the * .cm file

Unused variable: *name of the variable*	Warning	A variable defined in the Variables View is not used in the model.	The cause of the error could be that a variable was renamed and the refactoring in the Variables View was deactivated at the same time. As a result, the variable names in the model would have to be adapted manually. Otherwise, the unused variable can also be deleted.
DataVariations only contains single value for *variableName*	Warning	In a DataVariation, only one value was assigned to a variable.	Verify that the dataVariation makes sense here, if necessary, delete the variabe or insert additional values for the variable
Duplicate variable+value pair in **DataVariation**	Error	In a DataVariation, a variable was assigned the same value several times.	Check whether the dataVariation makes sense, if necessary, change the values of the variables
Syntax Error: ...	Error	There is a syntax error in an element with user defined expressions or code (e.g. Calculation)	Double click problem to find element and fix syntax error.
Undefined variable reference: ...	Error	A non-existing variable is used in the model (e.g. a Calculation)	Create Variable or modify/remove the reference.
History state has no incoming transitions	Warning	A state is declared as a History State, but there are no incoming transition that are using this state. The result is that the History State has no effect.	Change (at least one) incoming transitions and move their end to the circle with the "H" within the state instead of the border of the state.
Guard conditions for outgoing transitions may be incomplete	Warning	A state (or junction) with • more than 1 outgoing transition (within diagram) • where none of them has a trigger • at least 1 transition has an empty guard	Declare a guards (or triggers) at all outgoing transitions
Comparison with '==' has no effect here, expected '='	Warning	In a calculation the '==' operator is used without an effect.	Correct the calculation. Probably change '==' to '='.
Test target... should be ...	Warning	A setting in the test target view has not been configured properly in the test targets view	A setting for one or more test targets should be changed according to the description ⚠ Changing test targets does not lead to an automatic validation

		(or using a wizard)	of the project. A check can be initiated manually from the MODICA toolbar.
Test target... must be ...	Error		

Unrecognized errors

Not all problems can be detected in the offline-validation. Further error messages that can happend during *Load model* or *Test Generation* in the *Progress Panel are listed below.*

Project.Action.State1 (-> Project.Action.State2): **Undeclared identifier '...'**or **Found syntax error after ...** or similar	Error	The state or the transition *(heer: Project.Action.State1 (-> Project.Action.State2))* contains a syntax error.	Verification of the code for all Naming Fragments, Descriptions, Calculations and Guards of the State / Transition
Project.Action.State1 (-> Project.Action.State2): **Type 'class:Globals' has no member named '...'**	Error	In the case of the calculations / guards of the state / transition, a global variable is accessed, which is not declared	Create the corresponding global variable or change the calculation / guard
Project.Action.State1 (-> Project.Action.State2): **Incompatible types in expression '...'**	Error	In the case of the calculations or guards of the state / transition exists an expression with two variables, which are not compatible by type (eg 'string' + 'integer').	Correction of the calculation rule

Used Elements Analysis

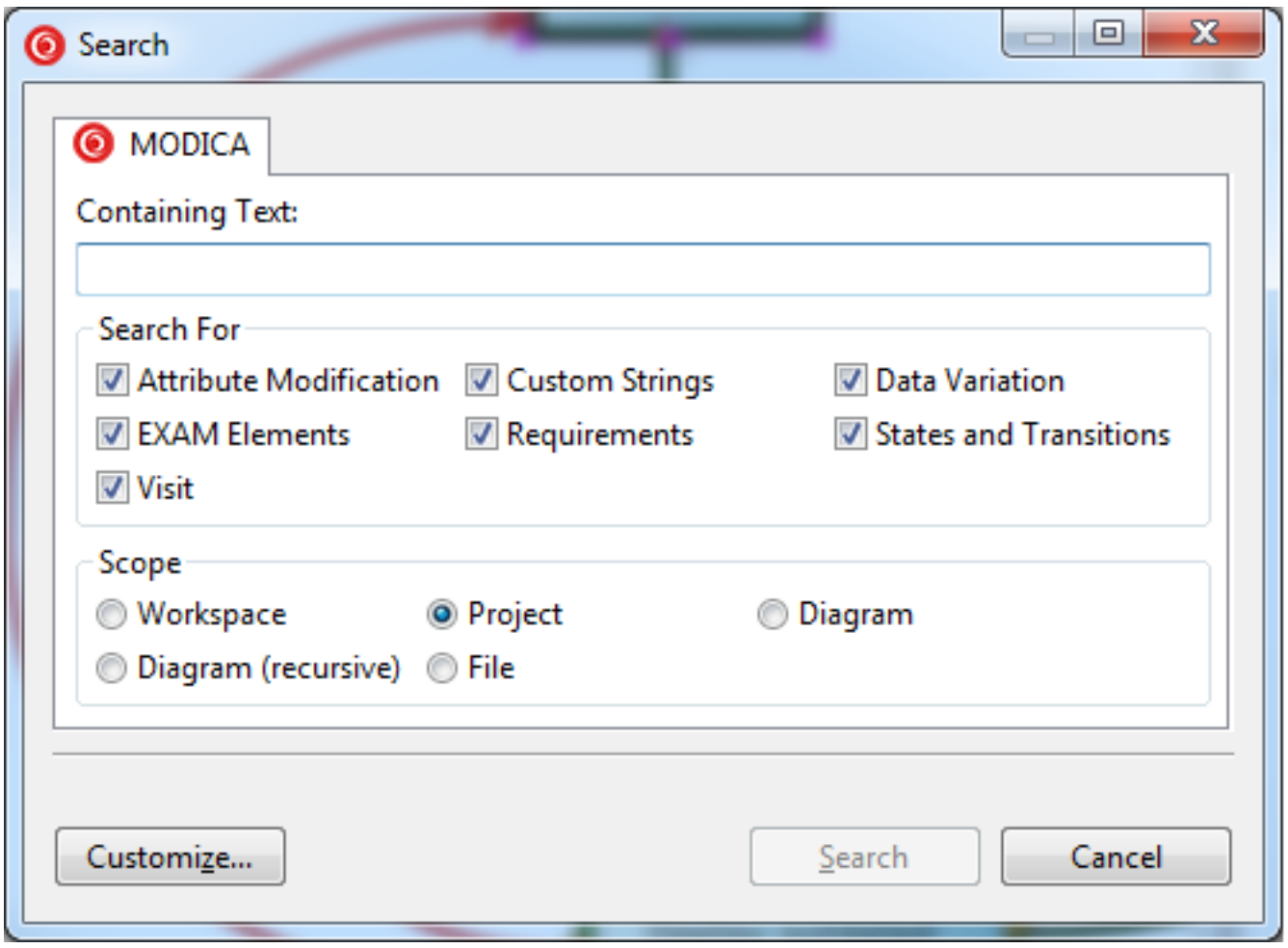
Search & Replace

Search and Replace is an important and basic tool in any development tool to find used objects in the project or replace them with a corresponding other object.

In the following, these two functions are described in the MODICA tooling.

Search

The search in MODICA can be started using the key combination `Ctrl + H` or in the menu bar in the *Search* menu. The following search dialog appears in which the search can be configured.

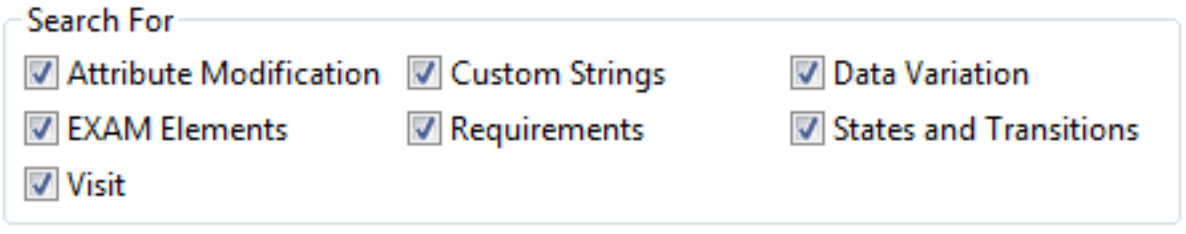


Used elements analysis, Search settings

Configuring the search

The following search options can be configured for the string to be searched for.

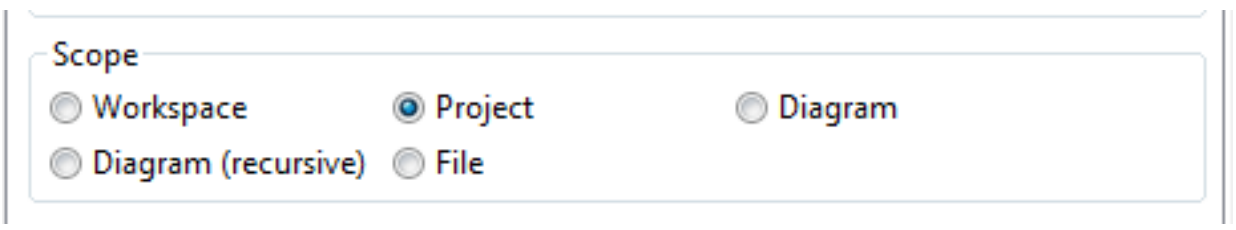
Search For: Select the MODICA objects in which you want to search for the specified text.



Used elements analysis, Search settings: Search For

Attribute Modification	Search for modifications on attributes in the model
Custom Strings	Search for text used within MODICA's own objects (*Description, Naming Fragment, Calculation*, etc.)
Data Variation	Search over all *Data Variations*
EXAM-Elements	Search for EXAM elements (also parameters)
Requirements	Search within *requirements*
States and Transitions	Search *for state* names and *transitions* labels
Visit	Search for *visit* elements

Scope: Select the project frame in which you want to searched.



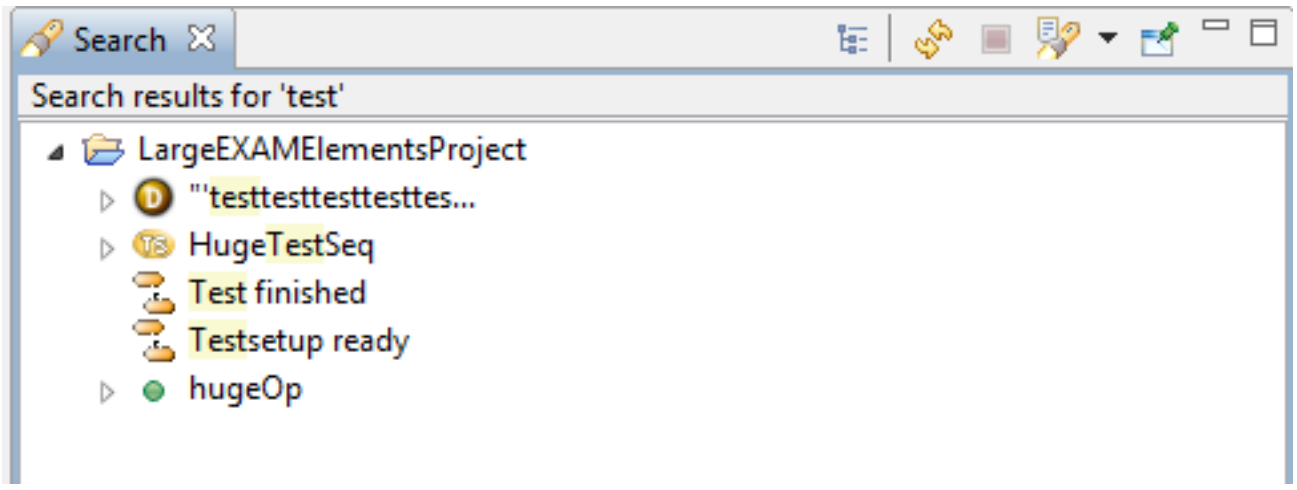
Used elements analysis, Search settings: Scope

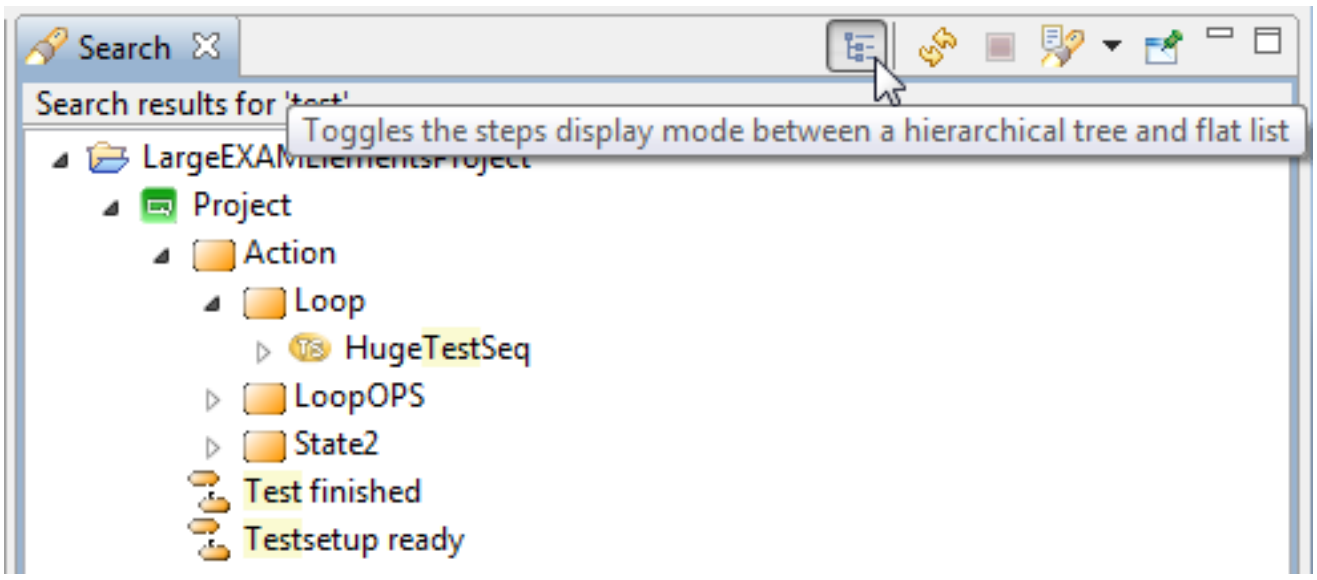
You can choose between the following areas:

Workspace	Search in all projects in the workspace
Project	Search in the currently selected project. Before, a project should be selected in Project Explorer
Diagram	Search in current diagram (active editor)
Diagram (recursive)	Search in current diagram and in sub diiagrams
File	Search in all diagrams of the current statechart (all *.cm file content)

Preparation of search results

The result of a search is displayed in the View Search. This will appear the first time a search has been performed. The results are displayed as a tree using the hierarchy from the searched diagrams or using a flattened list.





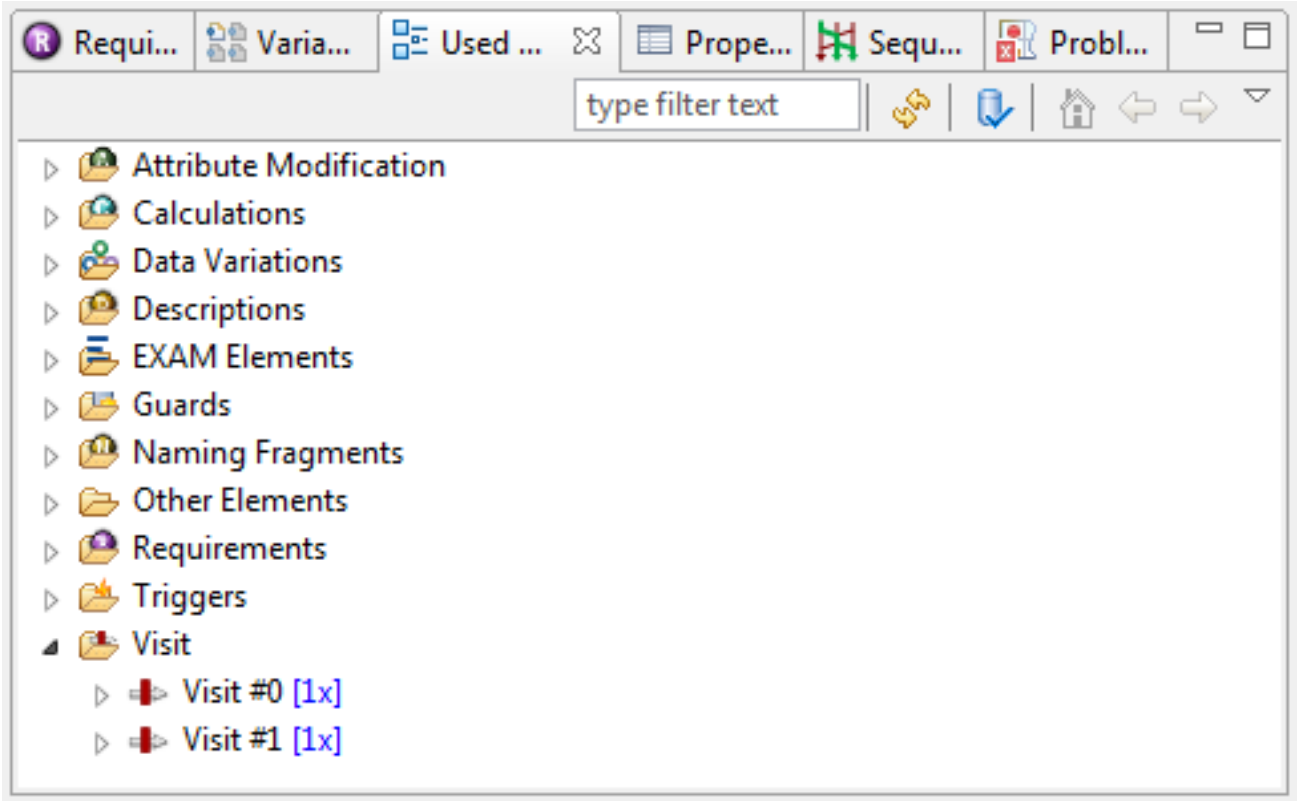
For the sake of clarity, single hierarchy steps or empty elements are partly invisible. If you look for example only in a project, this is not shown in the search results.

Refactoring

To enable automatic refactoring, the Refactoring Button must be activated in the *Variables View*. If this is activated, any renaming or moving of variables or folders causes an automatic refactoring, i. the entire model is adapted to match the new varibal name or folder. When deleting variables, refactoring causes a security check to determine whether the variable is still used in the model. If variables are still used, you can cancel the deletion.
In large models or with many open editors, refactoring can take a few seconds.

Used Elements View

In the *Used Elements View*, all objects used in a MODICA model are listed and displayed. Double-clicking on a specific element opens the corresponding editor and selects this element automatically. In addition, the number of sites of identical elements is displayed in square brackets.



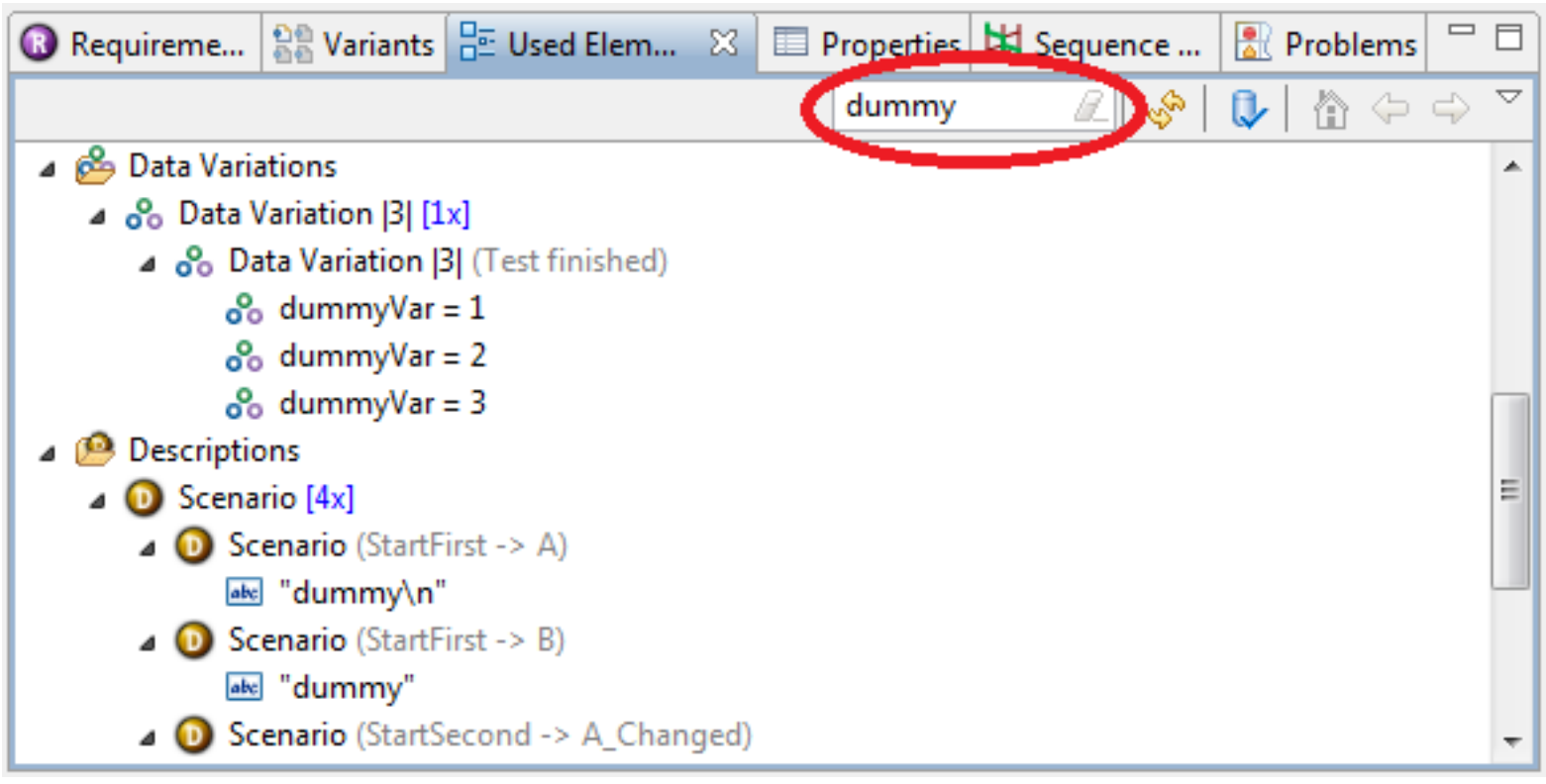
Used elements analysis, Used Elements View

The objects used are grouped in a tree structure as follows:

- Attribute Modification
- Calculations
- Data Variations
- Descriptions
- EXAM-Elements
- Guards
- Naming Fragments
- Requirements
- Triggers
- Visit

Search

With the help of the search function in the upper toolbar of the *Used Element Views*, all used elements can be searched for a certain string, in the screenshot `'dummy'`. The results are listed directly below - sorted according to the groups mentioned above.

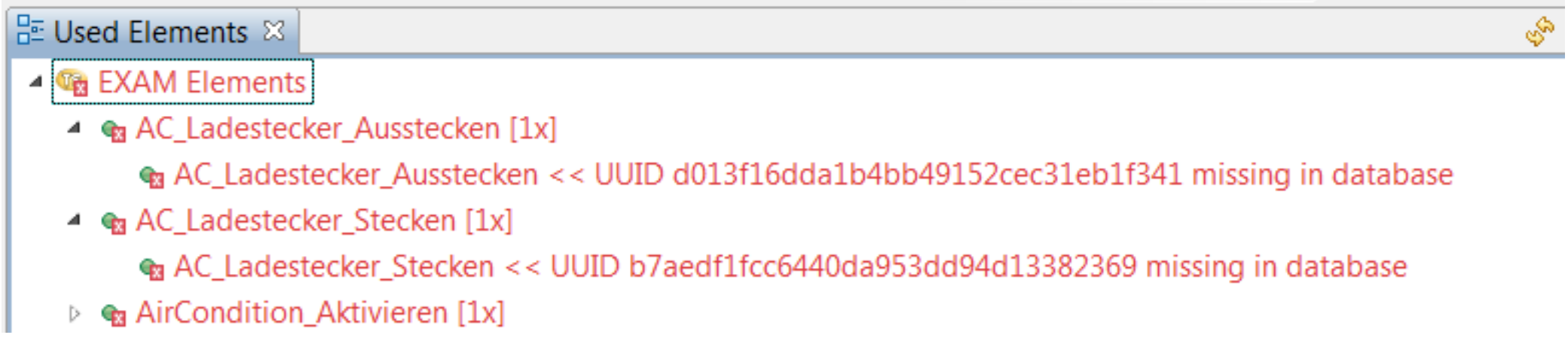


Used elements analysis, Used Elements View - Search

Automatic element check (aka Check Validity)

In the *Used Elements View*, there is the option to match the existing model against the underlying databases.

To do this, select the *Check Validity* button  in the *Used Element View*. This process can take some time depending on the project size and network connection. After the model has been checked against the database, all problematic model elements are marked in red and provided with an error message.



Used elements analysis, Used Elements View - Validation

The following changes are automatically found and marked:

- Missing connection to the associated EXAM server.
- Element does not exist in the EXAM database.
- Element name is changed to EXAM database.
- Sub-elements (e.g., operation arguments) are changed in the EXAM database.
- ParameterSet(s) are changed in the EXAM database.

Project Exchange and Versioning

For various purposes it may be necessary to send MODICA projects, to version them or to create backup copies. This chapter provides an overview of the necessary steps.

Overview of file storage and exchange of MODICA projects

MODICA projects are usually found as folders or .zip archives that contain some files. In principle, these folders and archives can be exchanged, filed and versioned as a whole. However, there are some files that can be regenerated again and therefore can be omitted for saving space for these purposes.
These are the following files or folders:

.database
.bphits-database
.metadata (also with number abbreviations at the end)
.stable-model (also with number abbreviations at the end)
model/.MOD_generated.cqa
model/usecases.ucs

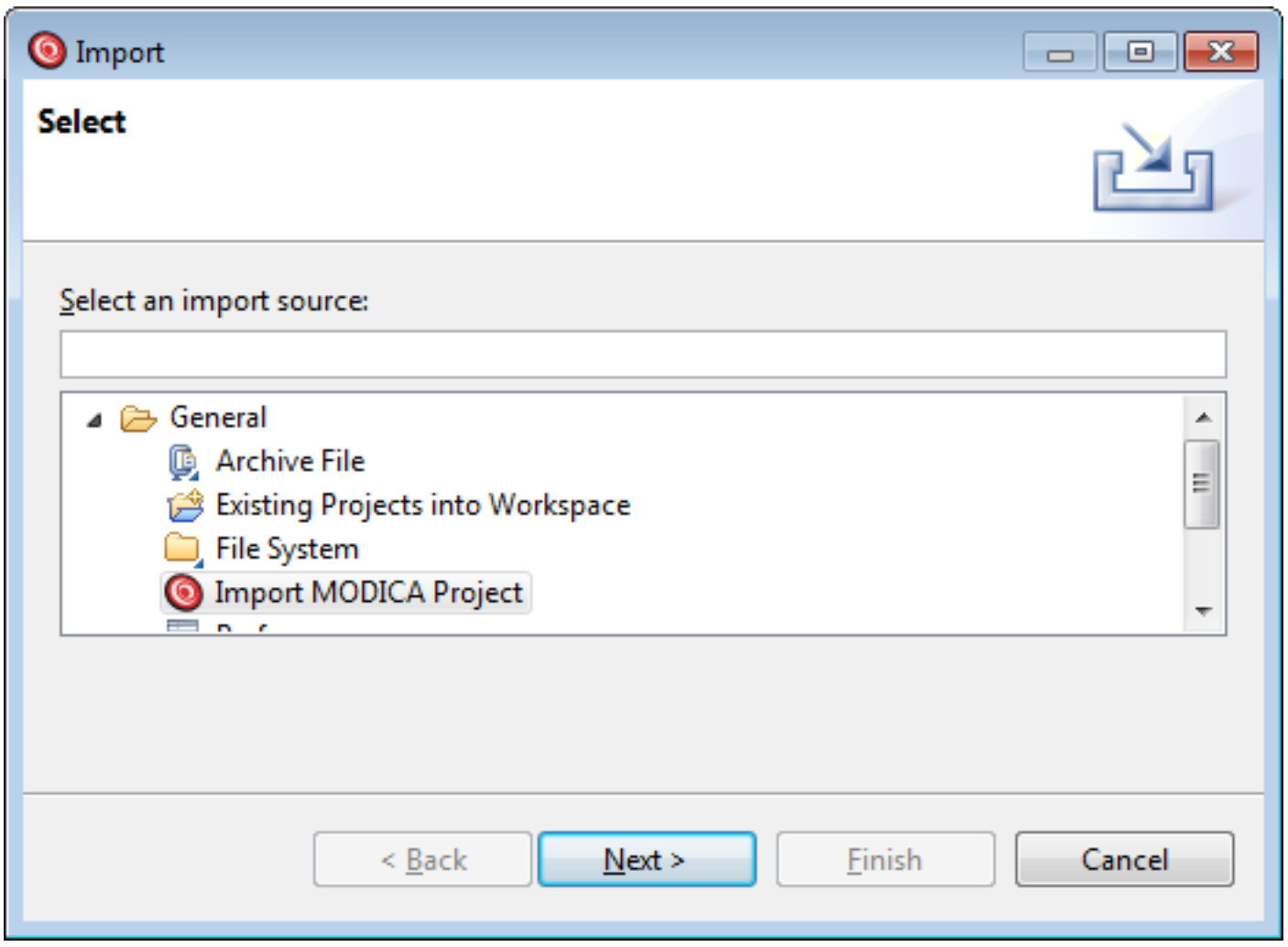
All other files and folders should be considered important in case of doubt.
⚠ **If these files are omitted during a project exchange, the following must be observed:**

- These files contain, among other things, the generated test cases. If necessary, they must be regenerated afterwards.
- After opening or importing a model without the named files, the selectable generation targets (visible in the *Test Targets View*) will be regenerated at the next "*Load model*".
  In the course of this regeneration, any changed settings for these targets are rejected.
  However, the settings can be restored using the following steps:

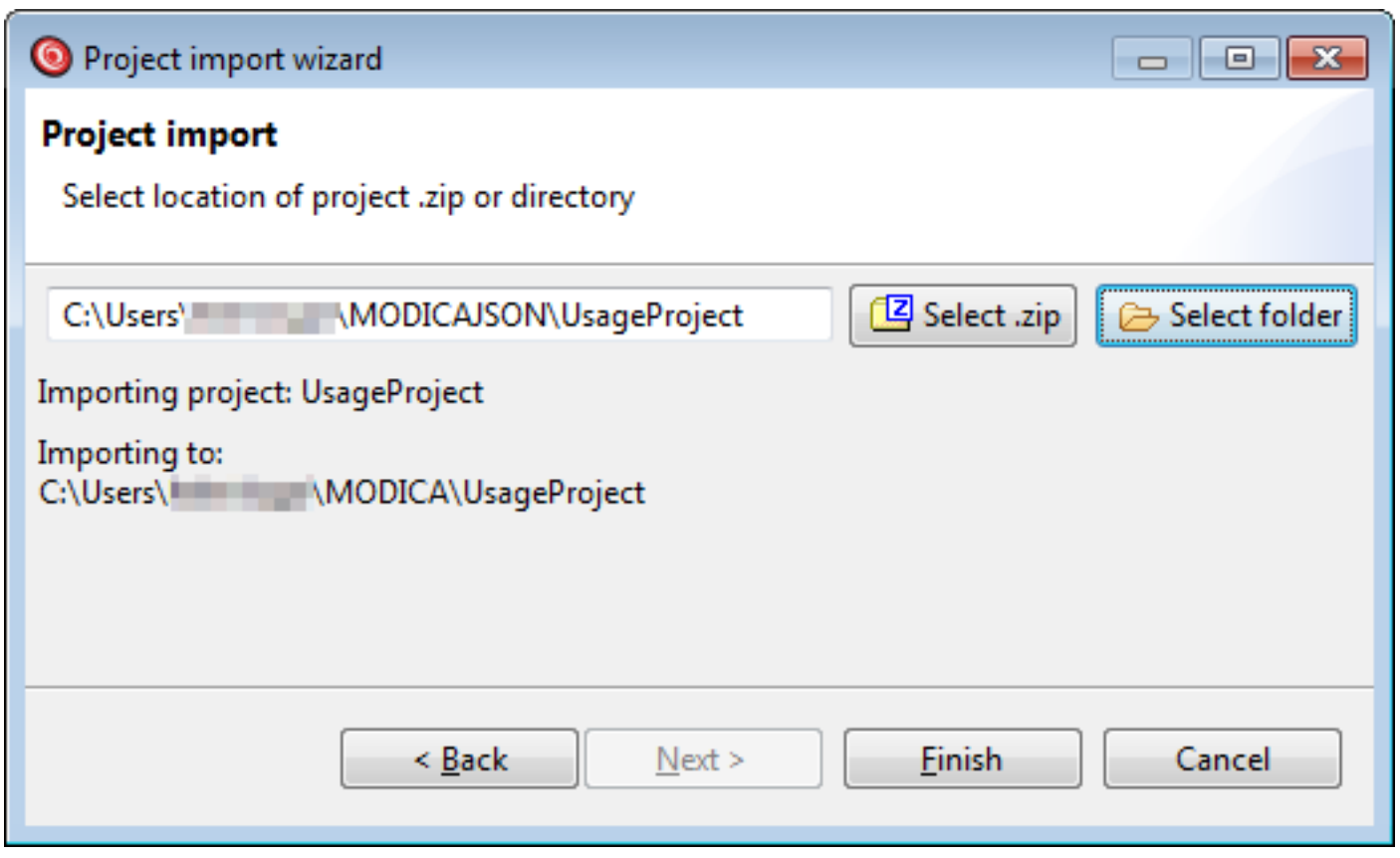
  1. Close the project in MODICA (*Project Explorer → right click on Project → Close Project*)
  2. Copy the *.qtronic* file again into the project folder, overwriting the existing one (or reset the local changes when using a tool such as SVN)
  3. Open the project in MODICA (*Project Explorer → right click on Project → Open Project*)

Export / import a project as .zip file

In addition to the possibilities for data exchange known from other Eclipse-based products (*Project Explorer → Import / Export projects →* ...), there is also a simplified import and export wizard in MODICA. These are also called via the *Project Explorer*:



Afterwards, a folder or zip archive can be selected which automatically runs through a short consistency check and can be imported on success:



The MODICA export wizard works in the same way, but allows only one export as a zip archive.

If the same MODICA versions are used, the project zip archives can be freely exchanged. If an upgrade is carried out due to different versions, the exchange with old MODICA installations is no longer supported. These should then be updated.

Versioning

Especially for important projects, it is often not sufficent to save individual project stands as zip archives and to manually insert them into a backup process. In addition, a collaboration of several people can quickly lose the overview if no central project storage is available. As a solution to this problem, a file management system, such as Subversion (SVN), is recommended in MODICA.

Many clients are available for SVN. In principle, their use for MODICA projects is not significantly different from other fields of application, such as source code management. It should be noted, however, that the generated files should not be included in the versioning. This is especially true if the storage space for the version management is expensive since the generated data can exceed 100 MB per version depending on the project. In SVN, features such as "svn-ignore" and "global ignores" can be used here to prevent accidental check-in. When omitting the generated files, however, the warning for importing partial projects from the introductory section of this chapter must be observed.

Regardless of the selected versioning tool, it should be noted that MODICA projects should not be modified by several people at the same time. Particularly in the case of more complicated modifications, their correct assembly can be difficult.

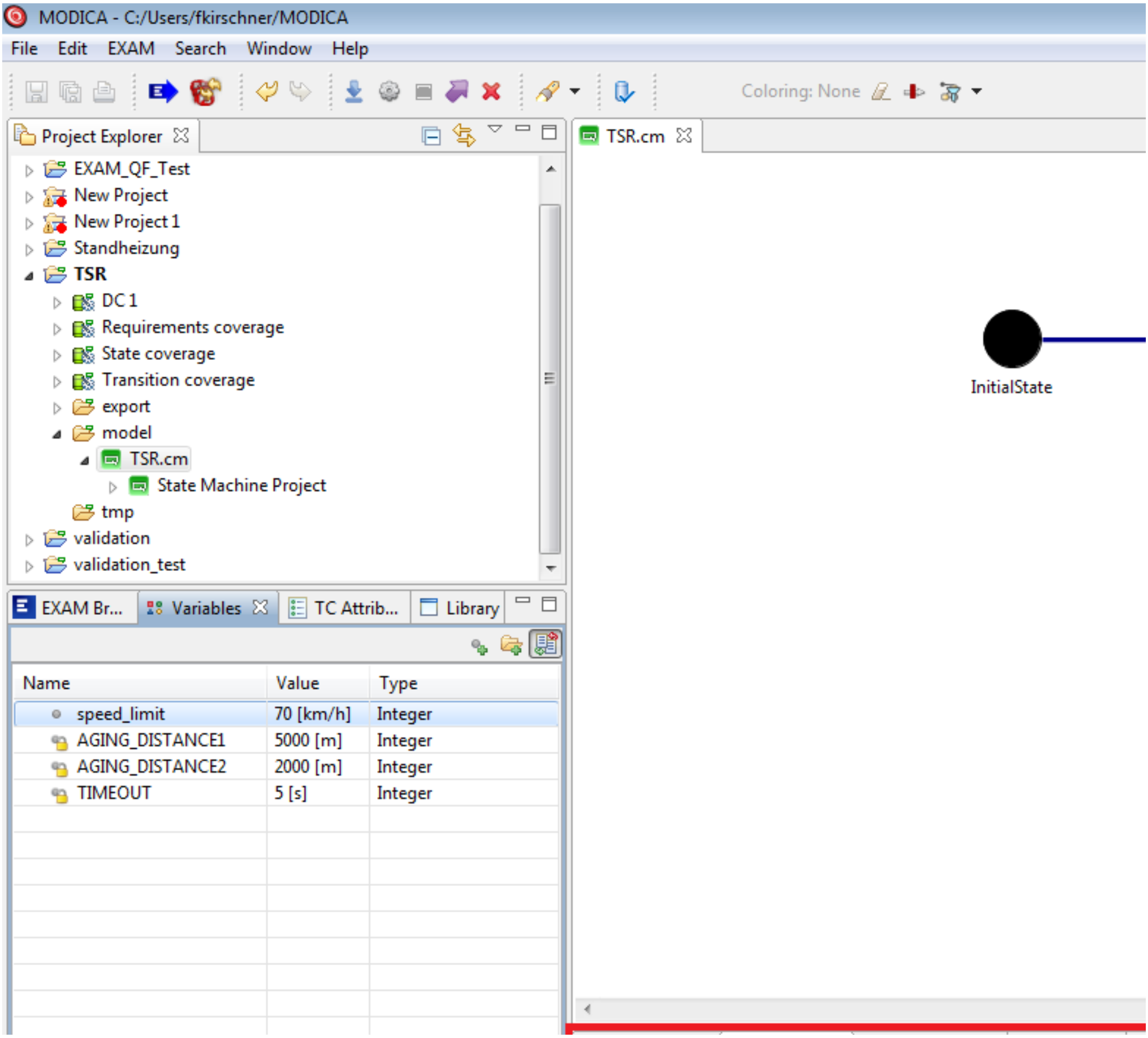
Variant Management

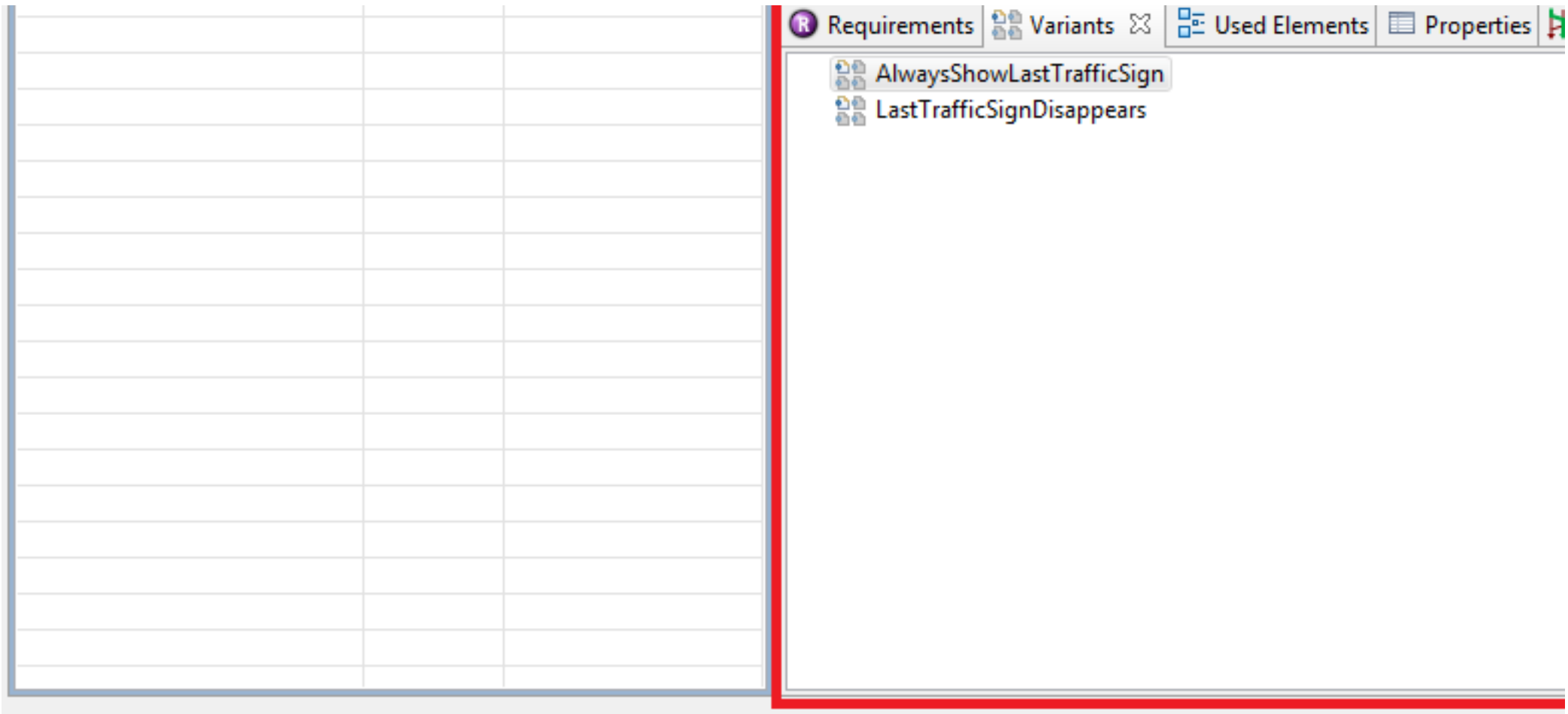
An ever-increasing number of products, which are very similar and differ in parts only in detail, presents challenges for manufacturers with regard to profitability, quality and structured variant management.
For this reason, the so-called variant management is supported in MODICA and the modeling of variants is made possible via a special graphical interface in MODICA usage models.
Variant management in its entirety consists of three parts:

- Creating and editing *variants* (classifications).
- Configuration of the MODICA model areas, which are only relevant for special *variants*.
- Definition/Configuration of design configurations, which take *variants* into account.

Creating and editing variants (classifications)

Variants are created in the *Variants View*.



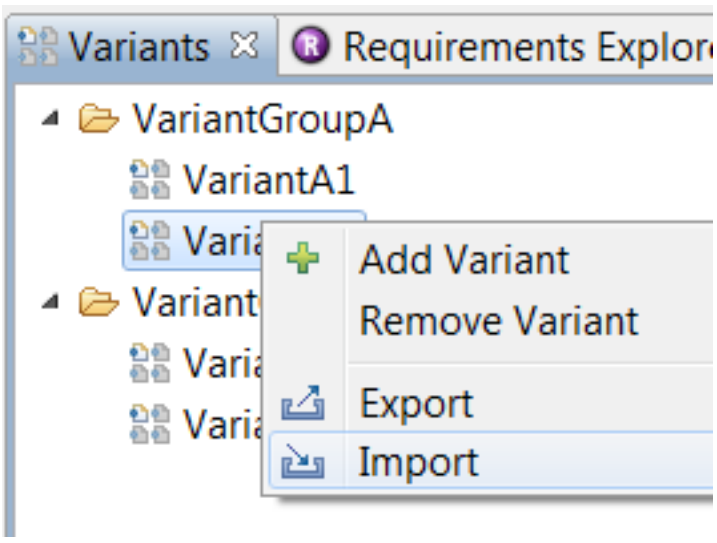


Variants View

In this view, *variants* can either be re-created, modified, or imported from an already created variant file.

Create / change / delete variants

You can use the *Variants Views* to edit the available *variants*. To do this, use the context menu (Open by right-clicking) of the view or the local toolbar  :



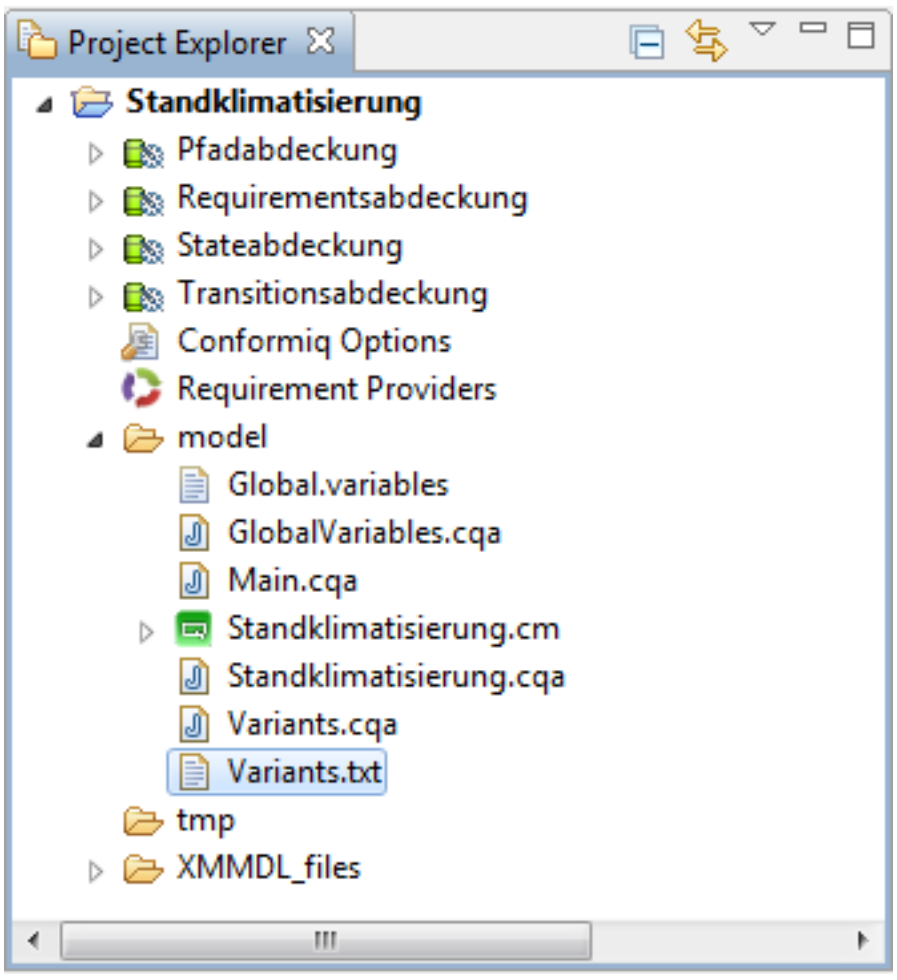
Variants View, commands

The editing of *variants* for existing models influence the variants in the model according to the following rules:

- New *variants*: All *transitions* belong to this *variant* as standard.
- New sub-*variants* (also by moving): The father-*variant* can no longer be deselected independently. If it has just been deselected on a *transition*, this information is deleted.
- Rename / move *variants*: The *variants* are adapted analogously in the model.
- Delete *variants*: No *transition* belongs to this *variant*. Other *variants* remain unaffected.
- Changes to *variants* that delete the deselect information of *transitions* are irreversible.

Import / Export of variants sets

Furthermore, created variant sets can be saved in a variant file via the buttons  or imported from a variant file.
By default, the variants created in a MODICA project are stored in the hidden file `Variants.txt`, which is located in the project folder in the subfolder *model*.



Variants View, Import/Export of variants (view of hidden project components)

Importing a variant file replaces the content of the file `Variants.txt` and the variants contained in the project.

To be able to use the *variant sets* created in a project in another project, the contents of the `Variants.txt` file can be saved in an exchange file of any name and then be reused in other projects using the variant import mechanism.

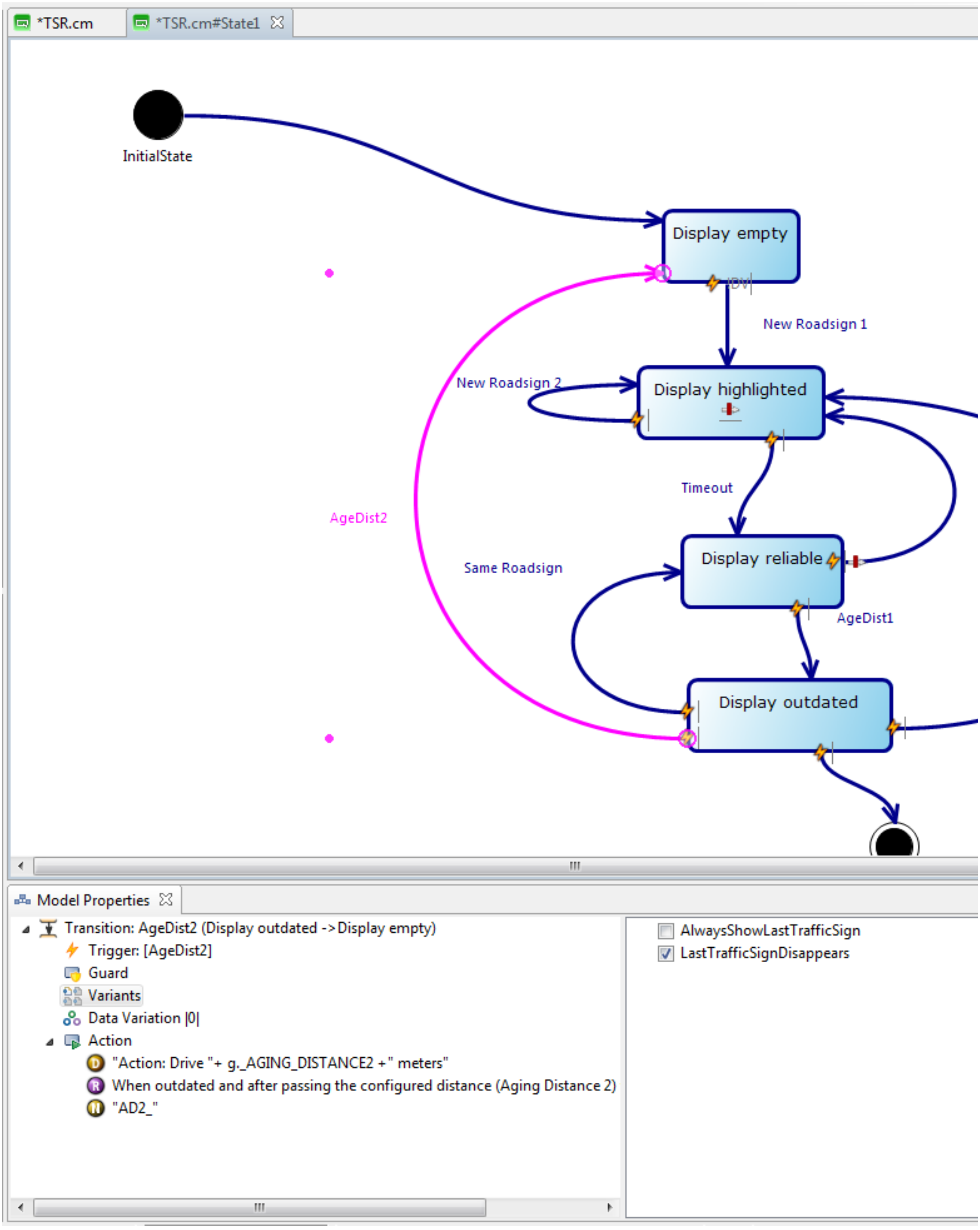
Variant management in the MODICA model

Once *variants* are created, edited or imported into the project, they can be used in the MODICA model.

The configuration of a MODICA model with regard to *variants* is made by marking transitions (and their functions / properties in this model) as part of a system *variant* or explicitly not as part of a system variant. This is done in the *Variants* section of the *transitions*, in which the imported *variants* are activated or deactivated.

Example:

If it is not possible in the example TSR for the system *variant* AlwaysShowLastTrafficSign (and all hierarchically subordinate variants) to hide a traffic sign after a certain distance, this *variant* is deactivated in the corresponding *transition*. To do this, click on the *transition*, which should not be present in the *variant* - in this case AgeDist2. In the *Model Properties View*, select *Variants* and uncheck the checkbox *AlwayShowLastTrafficSign*.



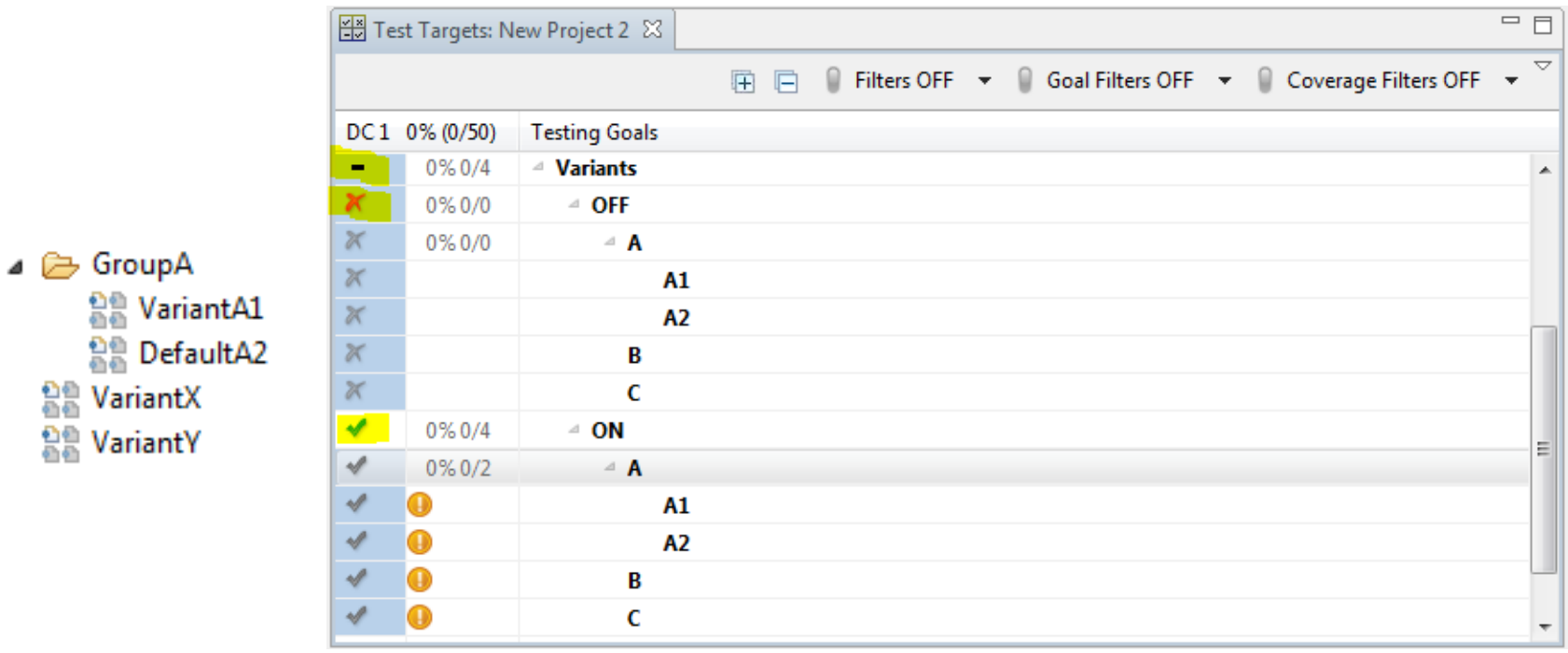
Variant management in MODICA models

Definition / Configuration of Design Configurations for variant management

In order to integrate the *variant management* into the test case generation, corresponding *design configurations* must be created.

As soon as a model is compiled with configured *variants* (*LoadModel*), all *variants* that are available in the project are now available for selection in the *Test Targets View* and can be configured.
For each *variant*, a *[...] ON* and a *[...] OFF* entry are created in the *Test Targets View*. To select a *variant*, the *[...] ON* entry must be set to target ✔ and the *[...] OFF* entry on block ✘ . To deselect it, exactly the opposite setting is necessary.

Example:



Variant management with configured design configuration

In the screenshot above, such a default configuration is set. Yellow highlights the manually set changes. The orange icons in the second column have no special meaning for the *variants*.

With the above default setting, all variants are initially marked as active, that is, The entire diagram can be traversed. If you want to deactivate a *variant*, the corresponding entry in the *ON* tree is set to block ✘ . In return, the corresponding entry in the *OFF* tree must be set to target ✔ . The opposite configuration is essential here. Otherwise, lead to the generation may lead to double or no test cases.

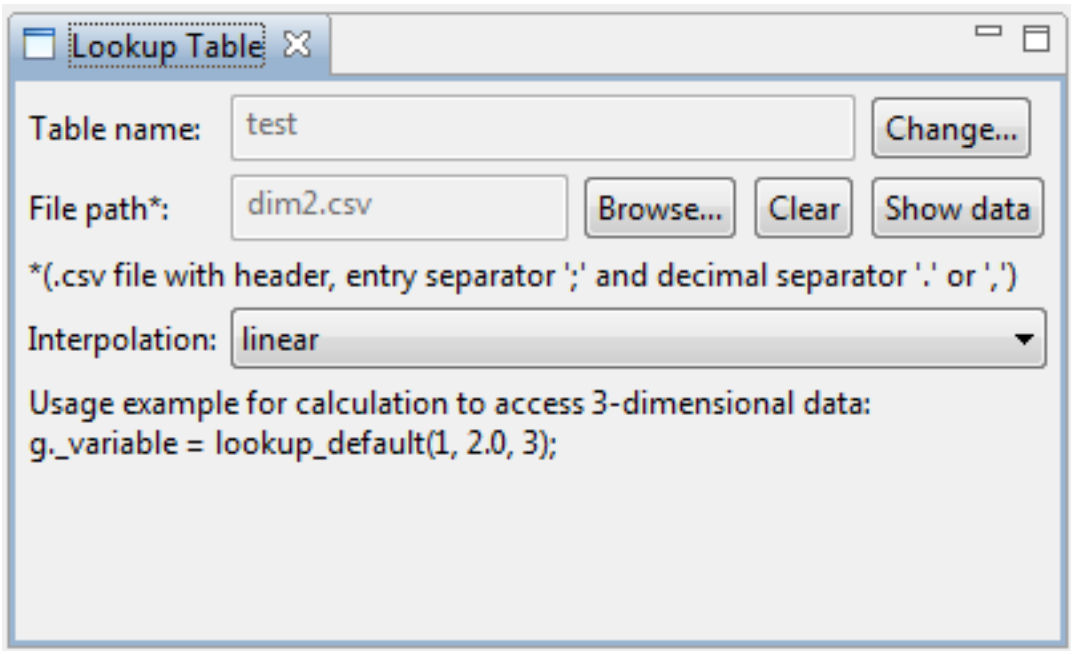
It is also important to make these settings in **all** design configurations.

For more information on how to configure *design configuration* and how to generate a test case, refer to chapter Test case generation.

Lookup Tables (advanced)

MODICA allows the import of *look-up* tables for data that can not easily be generated by calculations in the model. One possible example would be solutions for differential equations, or complex, non-linear functions.

Such data can be imported into a project in MODICA using the *Lookup Table View*. This view can be displayed in the main menu using *Window → ShowView → Lookup Table:*



This view allows you to import a .csv file into a project that has the following format:

- 1st line: column headings (can be empty)
- Semicolon as a separator between the columns
- At least 2 columns
- Last column contains value to be returned
- All other columns contain the arguments for the value query
- As values, floating-point numbers and integers are allowed
- A point can be used as a decimal separator (no comma!)

Even if the following example calculates only one sum of two arguments for easy comprehension, simple calculations should not be done in practice using the *lookup table* - they should be modeled as usual in a *calculation*.

.csv (2-dim a+b)

```
a;b;Sum of a and b
0;0;0
0;1;1
0;2;2
1;0;1
1;1;2
1;2;3
2;0;2
2;1;3
2.0;2.0;4.0
```

In the model, the values of a *look up table* named sum can be used in ***calculations*** as follows: (for the above csv file)

Calculation code

```
g._result= lookup_sum(g._a, g._b);
```

The result stored afterwards in `g._result` can then be used in ***guards***, ***naming fragments*** etc. However, a direct call to `lookup_sum (...)` in these elements is not allowed and generates an error message in *Load model*.

Interpolation

The Interpolation field is set to *none* by default. In which case the queried arguments are used directly for access. If a table contains, for example, entries for a = 1 and a = 2, a query for a = 0.5 can not be answered first.

If you change the setting to *nearest*, the query is rounded to the nearest whole integer (from 0.5 is rounded up). For example, the following values would result with the above .csv file:

a	**b**	**lookup_sum(a,b)**			
		none	**nearest**	**linear**	
1	2	3	3	3	Immediate hit, therefore all the same
0	0	0	0	0	Immediate hit, therefore all the same
0.4	0	ERROR	0	0.4	a = 0.4 not in data source, so none exits
0.9	1	ERROR	1.9	2	a = 0.9 is not in data source, so none exits
2.4	0	ERROR	2	ERROR	Either 2.4 or 3 are stored for a, so only *nearest* works which returns a = 2
3	0	ERROR	ERROR	ERROR	Desired values are outside the table

Highlight Coverage (en)

After the test cases have been generated, the coverage can be displayed in the diagram with regard to various criteria. Thus, conclusions can be drawn about the focus of the respective *testsuites* or *testcases*.

To do this, go to the *Test Steps View*, *right-click* on the project/testuite/testcase and select *Hightlight Coverage*.

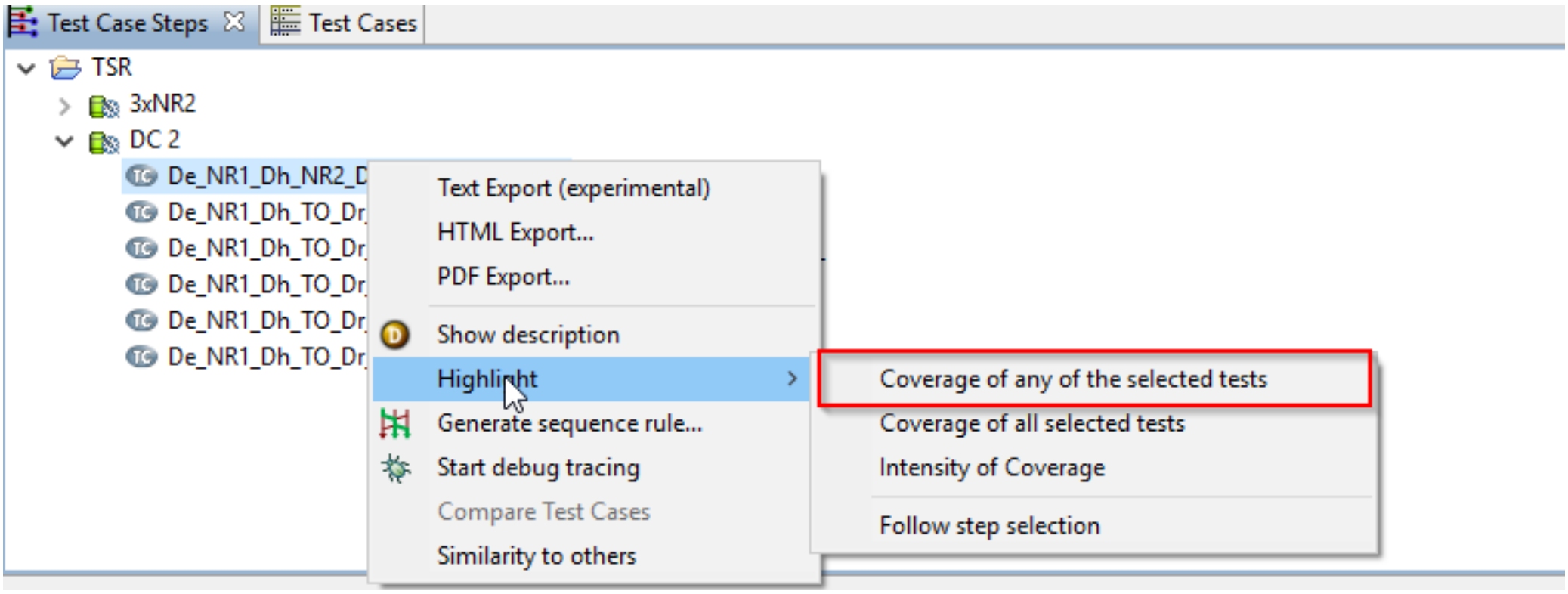
The following options are possible, which are described in more detail below.

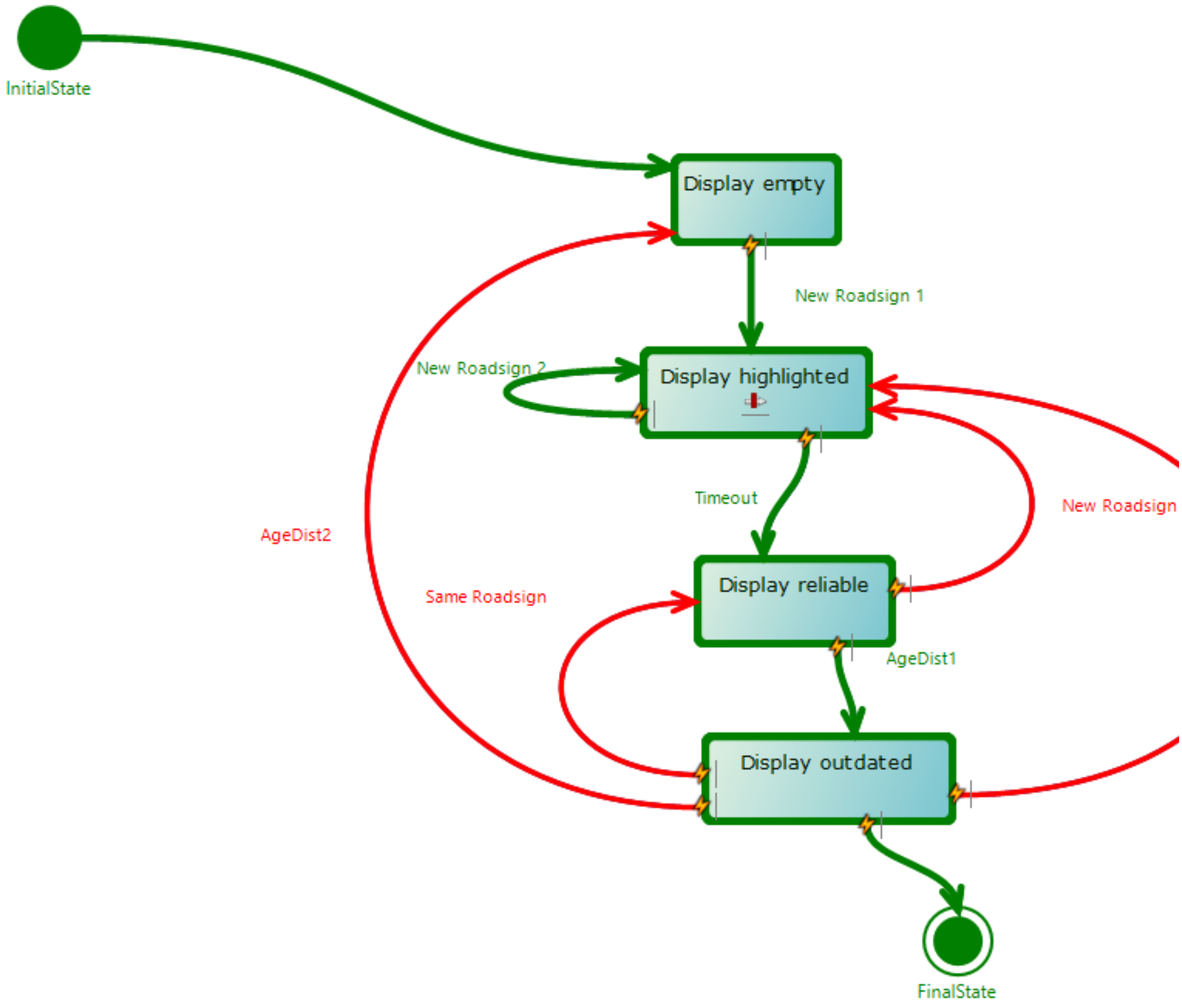
- Coverage of any of the selected tests
- Coverage of all selected tests
- Intensity of Coverage

Coverage of any of the selected tests

This function is used to illustrate whether or not an element has been **covered at all**. That is, as soon as an element has been traversed at least once, it receives a green color, otherwise red.

In the example, the test case which covers all states was selected. In this case it is sufficient to go through the model only once from top to bottom without any loops. As a result, all *transitions* that 'lead up' are not covered and are red.

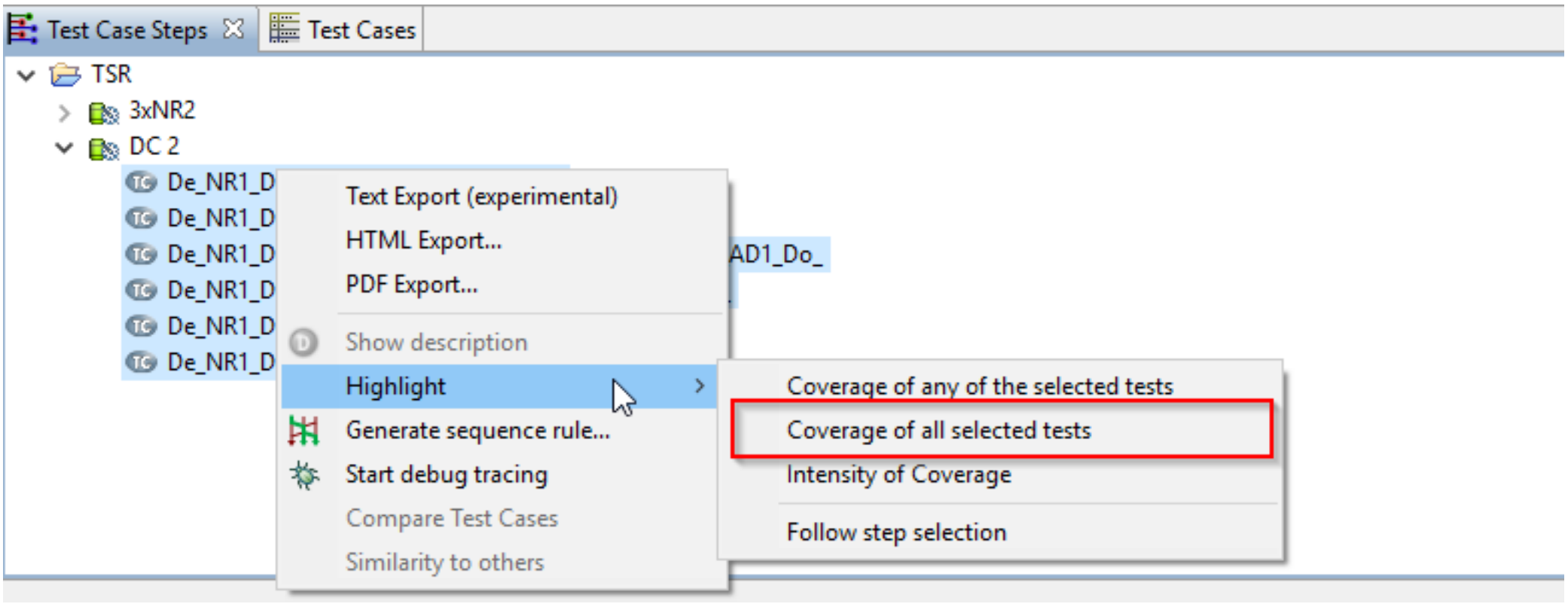


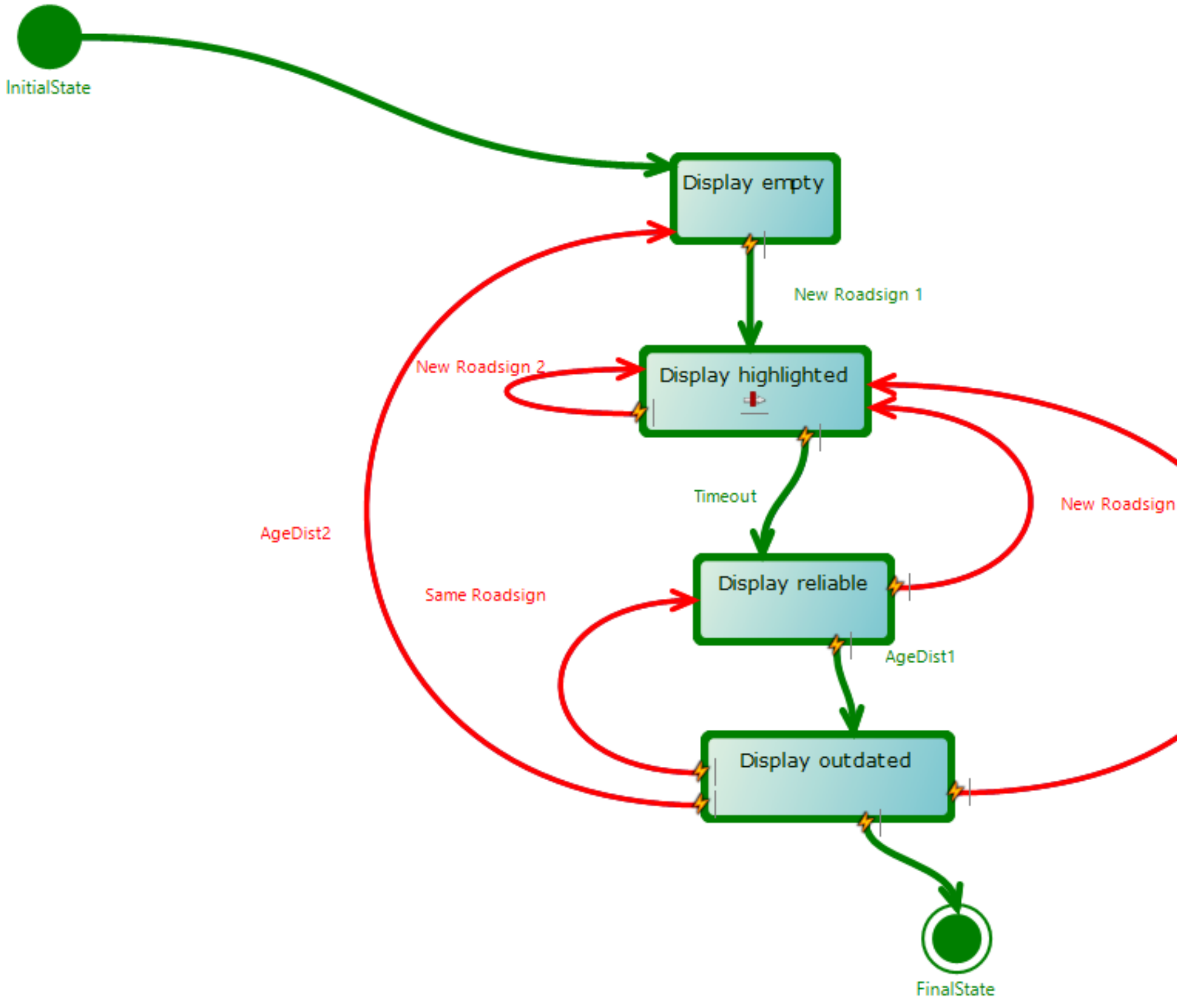


Coverage of all selected tests

"Coverage of all selected tests" function marks equal elemetns of selected test cases that have been covered. These elements are green.

In the example, several test cases were selected and the model shows us the elements that are the same in all selected test cases. That means all green marked elements are covered in every test case.

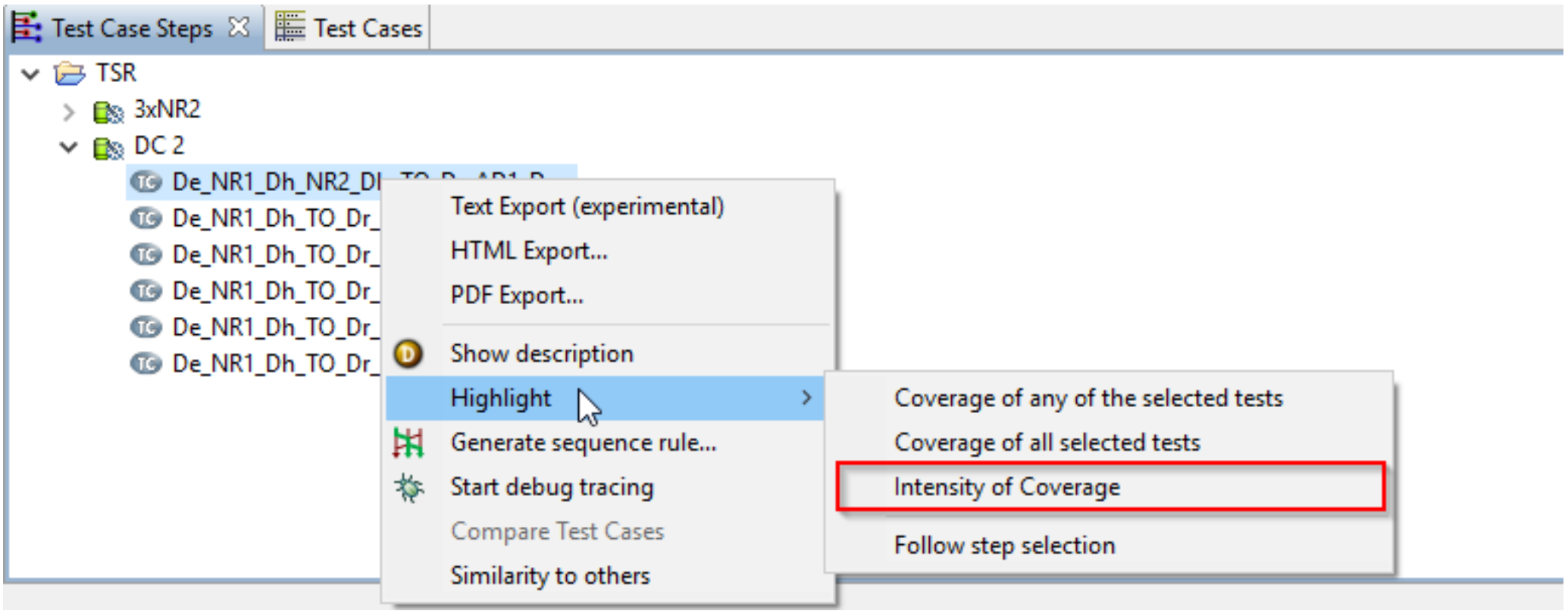

Test Case Steps
Test Cases
TSR
3xNR2
DC 2
De_NR1_D
De_NR1_D
De_NR1_D
De_NR1_D
De_NR1_D
De_NR1_D
AD1_Do_
Text Export (experimental)
HTML Export...
PDF Export...
Show description
Highlight
Generate sequence rule...
Start debug tracing
Compare Test Cases
Similarity to others
Coverage of any of the selected tests
Coverage of all selected tests
Intensity of Coverage
Follow step selection

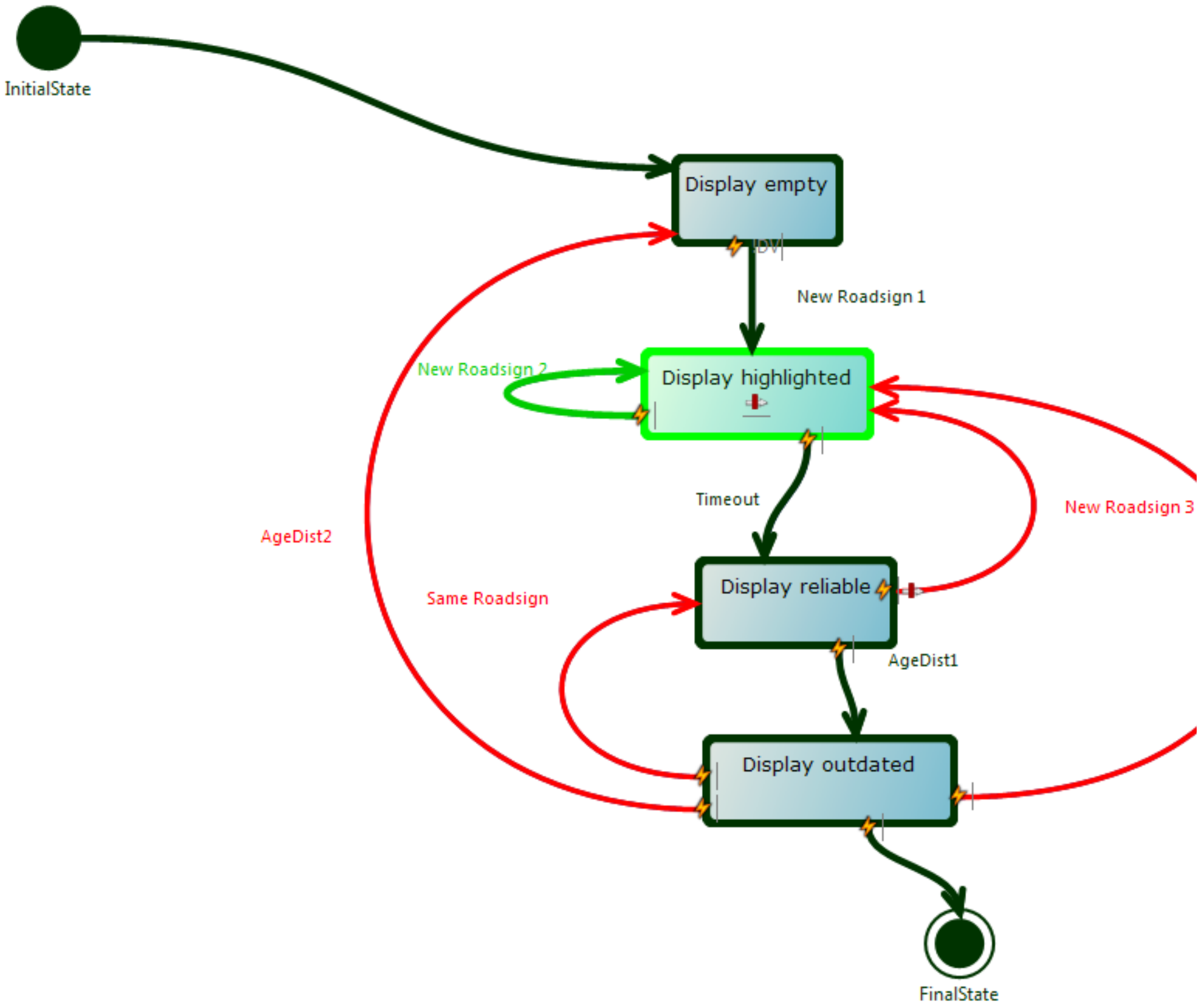


Intensity of Coverage

With this function, all **test steps** of all selected *test cases* are used as a basis. The more often a diagram element has passed through, the more intense (brighter) its green color becomes. Thus it is possible to graphically illustrate which diagram elements have been passed through very often.

In the example, the *test case* was selected, which passes through the *transition* New Roadsign 2 three times in succession. Accordingly, this transition and the *state* Display highlighted are dyed intensely green. In comparison, all other *states* or *transitions* were not (red) or very rarly traversed (in this caseonly once → dark green color).


Test Case Steps
Test Cases
TSR
3xNR2
DC 2
De_NR1_Dh_TO_Dr_
De_NR1_Dh_TO_Dr_
De_NR1_Dh_TO_Dr_
De_NR1_Dh_TO_Dr_
Text Export (experimental)
HTML Export...
PDF Export...
Show description
Highlight
Generate sequence rule...
Start debug tracing
Compare Test Cases
Similarity to others
Coverage of any of the selected tests
Coverage of all selected tests
Intensity of Coverage
Follow step selection


InitialState
Display empty
New Roadsign 1
Display highlighted
New Roadsign 2
Timeout
New Roadsign 3
AgeDist2
Same Roadsign
Display reliable
AgeDist1
Display outdated
FinalState

TC-Stepping

This feature makes it possible to trace each step of a test case in the diagram.

For this, you should switch to the *analysis perspective* using *Window → Open Perspective → Analysis*. In the *Test Case Steps View*, the stepping can be started by *right-clicking* on the desired test case → *Follow Step Selection*. First you have to select the top element in the right part of the *Test Case Steps View* (usually the *InitialState*) and you can now navigate up or down through the test case with the arrow keys.

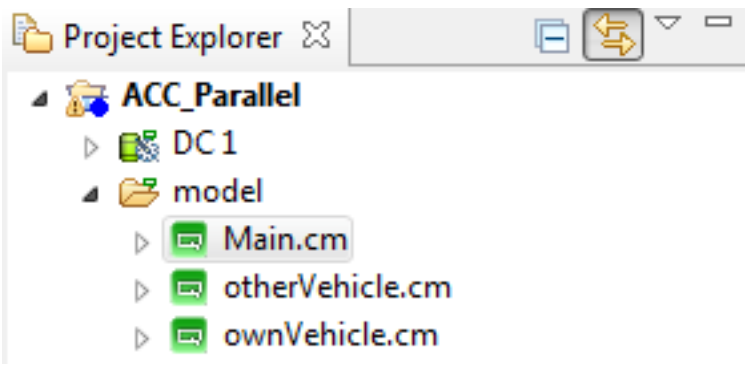
In the diagram, the element to which the corresponding step in the test steps list belongs is shown in light green. All other elements which have already been passed through or are still being passed through are dyed in medium green, all other elements which are not passed through in this test case are shown in red.

Parallel Statecarts

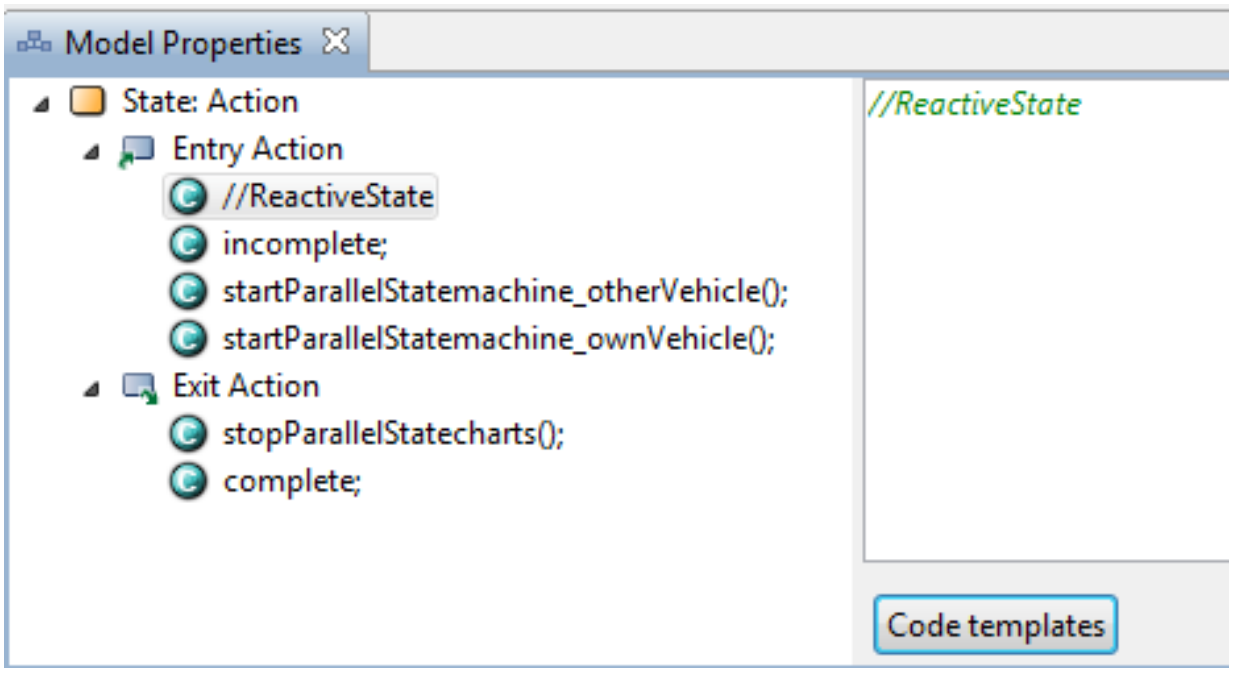
If a system-under-test has several interfaces which can be operated independently of one another, it may be useful to model these processes as parallel statecharts. This can be particularly useful if the testoracle is complex and has to react to many different interactions.

Many types of parallel processes can also be modeled using means such as history states, variables, and so on. However, in addition to difficult to understand diagrams, this also creates some problems for the test case generator and makes it difficult to control the desired parallelism.

In MODICA, several statecharts can be created and started parallel for these purposes. However, the main statechart (precondition-action-postcondition) retains a special role. In most cases, it should contain only the testoracle within "Action". For example, a parallel project could be divided into the three Statecharts Main (Testoracle) as well as ownVehicle and otherVehicle:

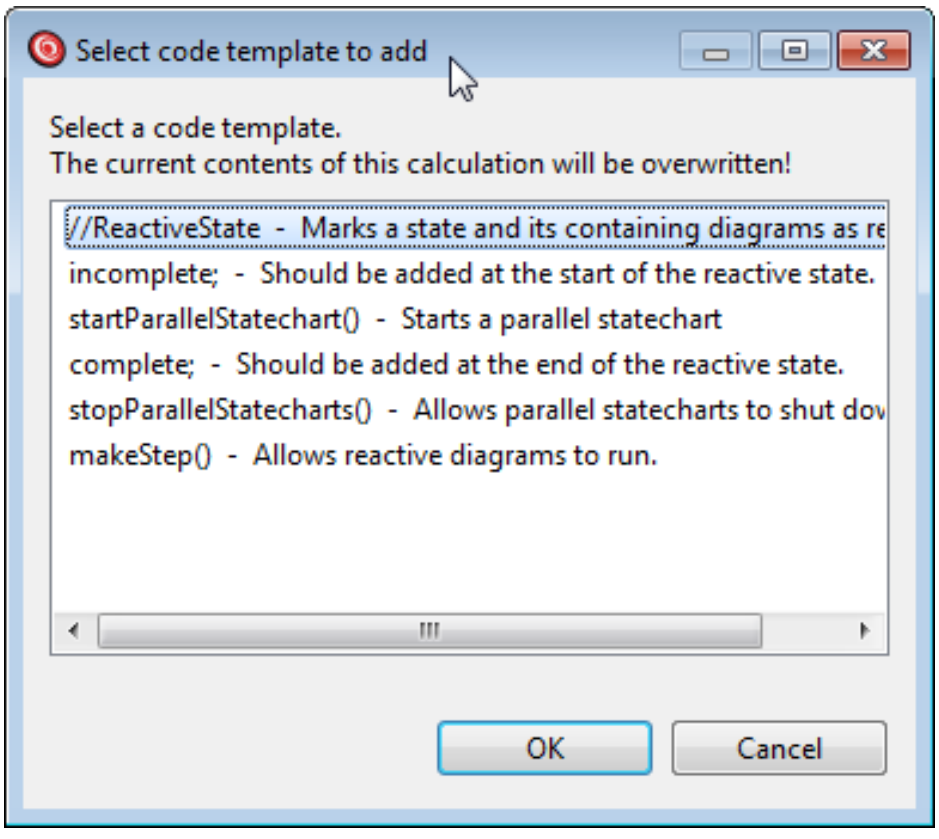


The state "Action" on the first level of the main statistics **should always be filled according to the pattern**:



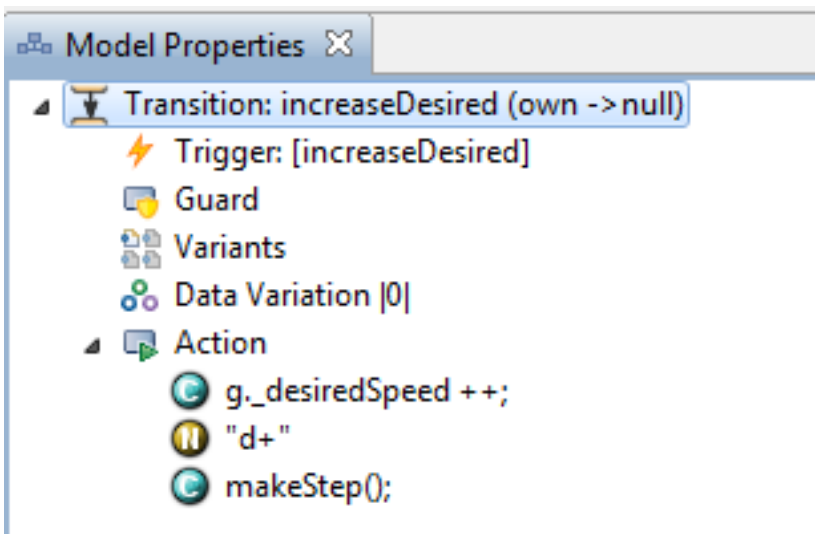
- **//ReactiveState** - Marks the state as a reactive state whose content is to wait for actions in other diagrams.
- **Incomplete;** - Prohibits the interruption of a test case as long as "Action" has not been left. ⚠ -> Unlike normal projects, "Only Finalized Runs" is disabled for parallel projects.
- **startParallelStatemachine_...();** - Next to the main statechart two additional statecharts "otherVehicle" and "ownVehicle" started. They serve in the example of modeling the behavior of the driver, as well as another vehicle. They are modeled as their own .cm files. A final state is not required within these statecharts.
- **stopParallelStatecharts();** - Runs the parallel statecharts down.
- **complete;** – Allows to terminate the test cases after "Action".

The button "Code templates" allows the quick generation of the above elements by means of a selection dialog:



The following notes should also be observed:

- The transition from Action towards the Postcondition must have a trigger.
- Whenever a variable is modified in parallel statecharts, which the reactive testoracle could wait for, a calculation "makeStep ();" is added:



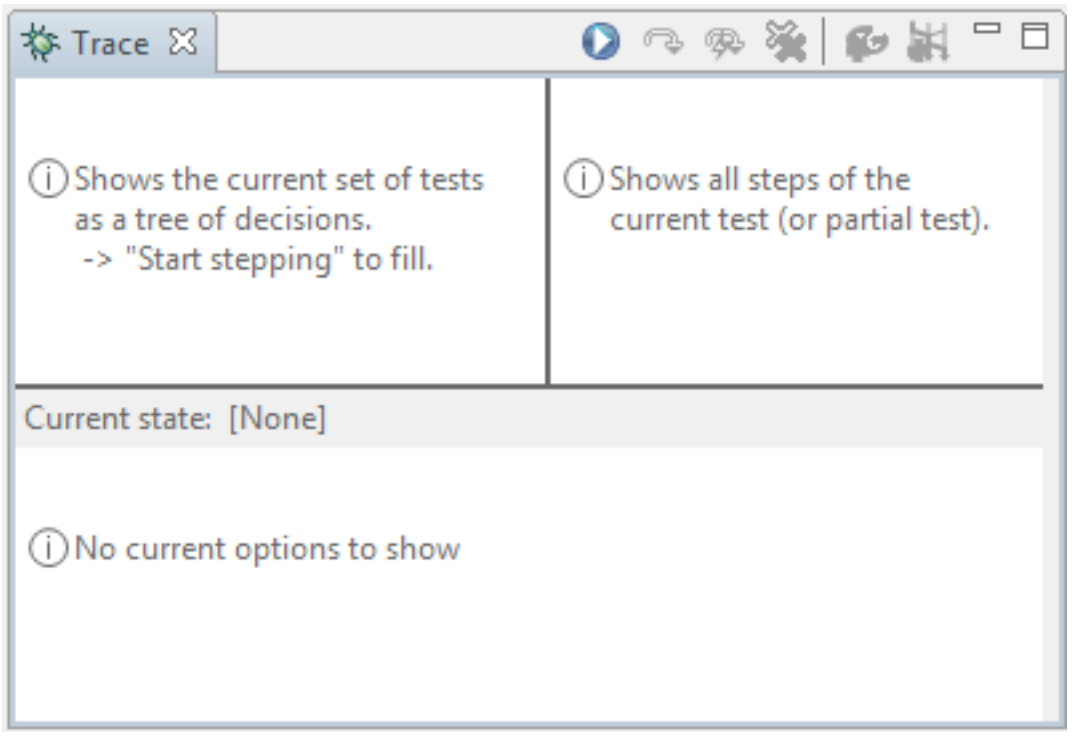
- To get a linear test case at the end, the generator shortens the model.
- Switching between the individual statecharts can only be done by triggers, guards in the reactive model part or "makeStep ();" should be made.

The functionality of the "stepping" feature for sequence rules is still limited for the use of parallel statecharts.

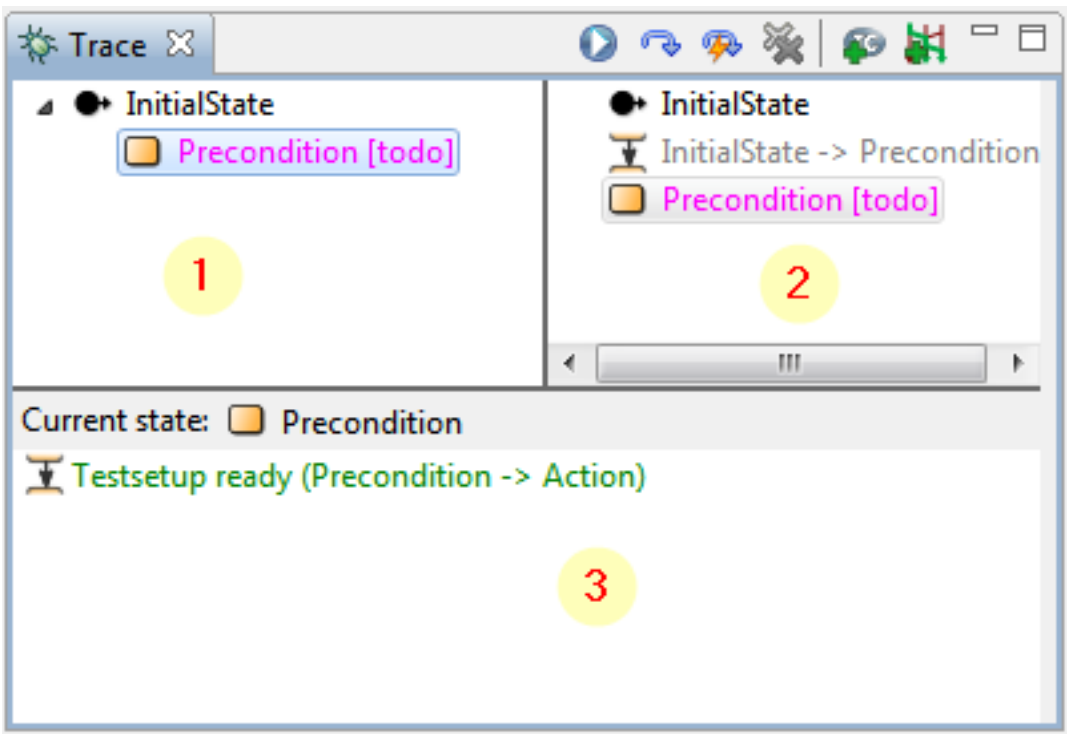
Model-execution for tracing/debug

Besides the automatic generation of Testcases, MODICA also allows semi-automatic execution of the models. This can be done using the *Trace* view. It supports many use cases like gaining better understanding of the testcase generator, finding and reproducing bugs in the model or simply seing the model "in action".

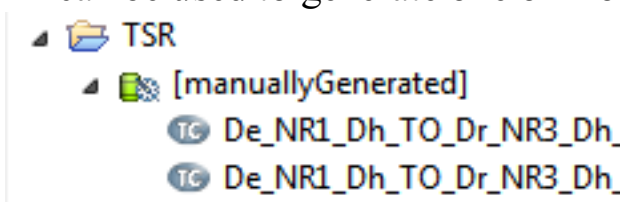
Before a stepping is started, the view only shows explanations of the three sections.



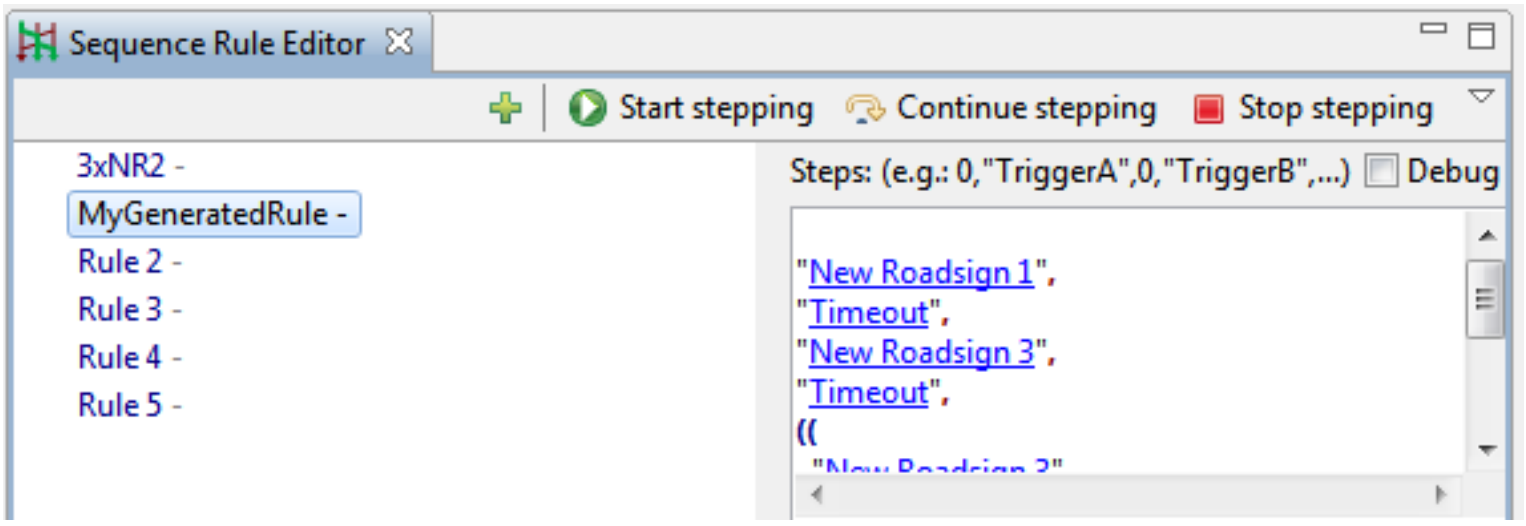
After starting the stepping, the view looks as follows:



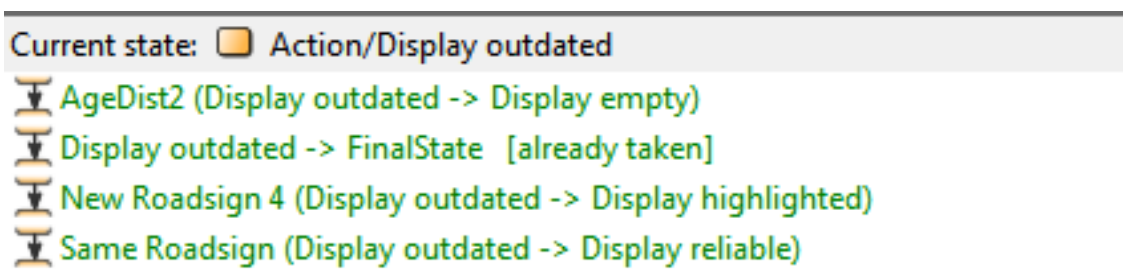
- Using *Start stepping*, the step-wise execution of the model can be started at the initial state of the main statechart. When sequence rules are available, one can be chosen to restrict the manual stepping.
- Step-wise execution can now be done using (single steps) or with (bigger steps).
- can be used to discard the current stepping.
- can be used to generate one or more tests from the current stepping, they appear in the *[manuallyGenerated]* DC:

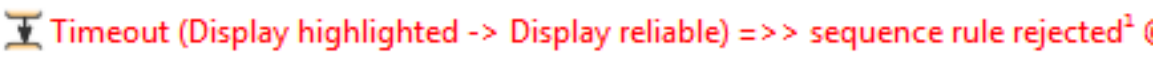
- can be used to store the current stepping in a sequence rule. This can be used to build sequence rules simply by stepping through the diagram:



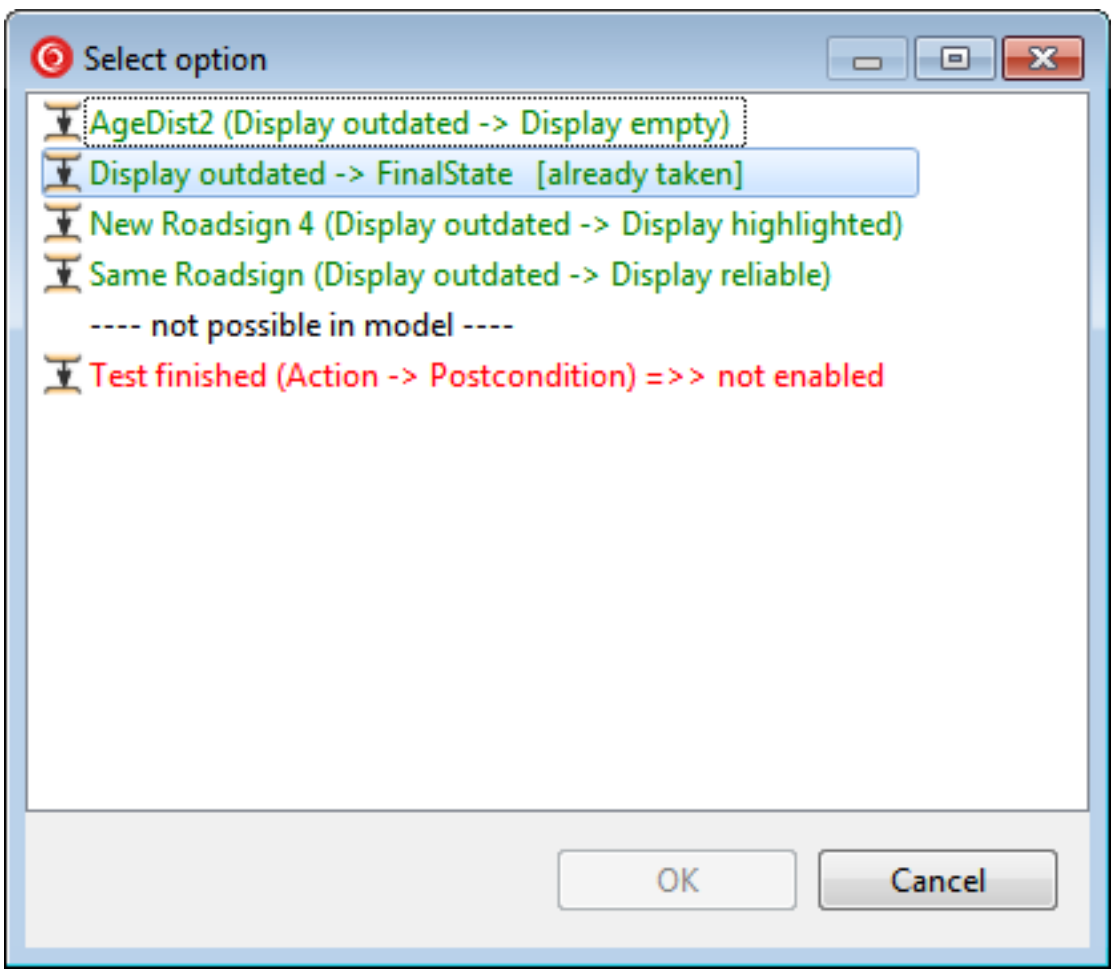
The execution considers guards, calculations and priorities in hierarchical diagrams to determine the next possible transitions. The next transitions at the selected state are displayed in the lower section of the view. If a transition has already been taken, it is annotated with *"[already taken]"*.



When executing with a sequence rule, some options may be displayed with a message that the option is not allowed in the current rule:



Additionally, it is also possible to inspect the disabled transitions when continuing from a state:



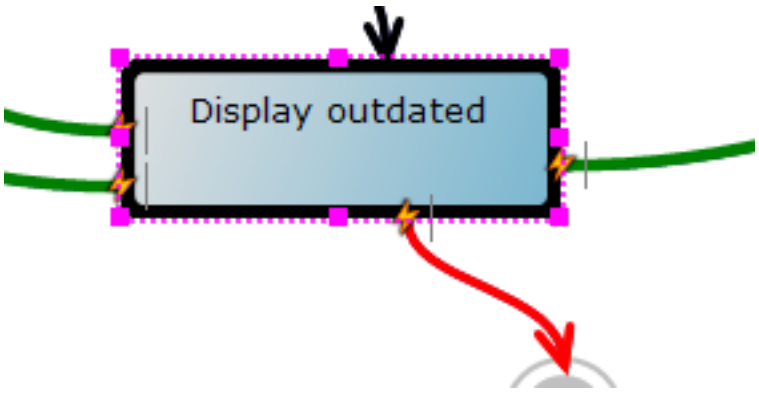
In the above example ...

- four different transitions are currently possible (green) and the user can select one of them using a double click (one was taken previously).

- one transitions are currently not allowed (red/brown)

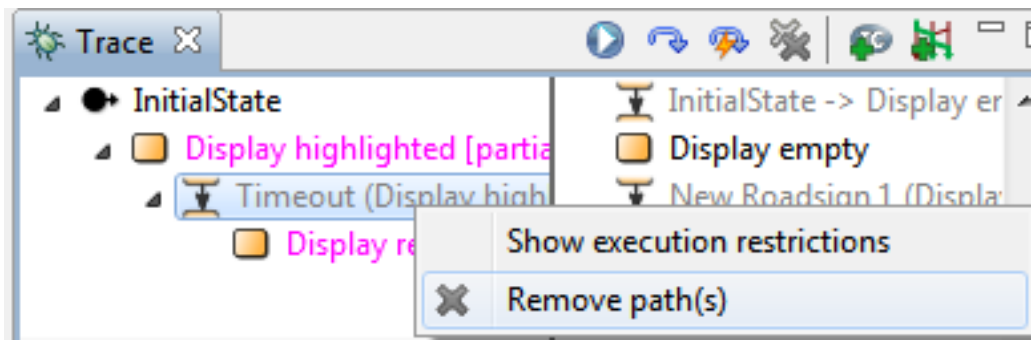
Depending on the execution state, model-bugs or the completed execution of the model will also be displayed using sub-elements.

Besides the tree-rendering in the *Trace* view, the appropriate diagram elements are also shown in the editor and highlighted:

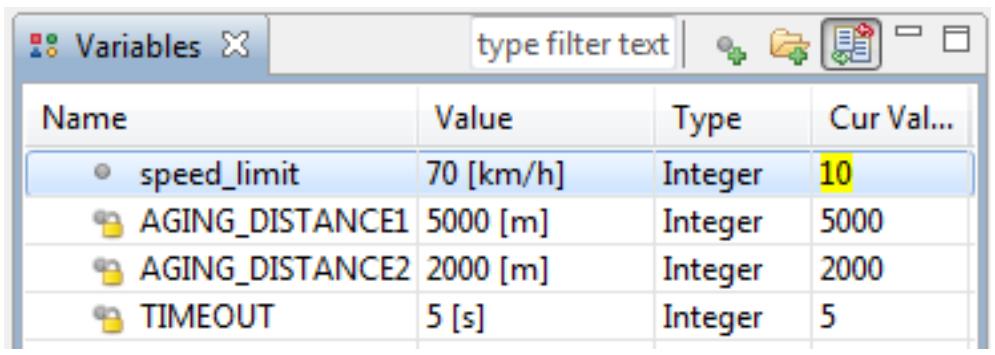


Modellausführung: Darstellung im Diagram

When selecting a [partial] state, you can continue on another path as before. Additionally, it is possible to completely discard transitions using the context menu. Doing this will remove all subitems of the selected transition from the tree (and the stepping). *Show execution restrictions* allows to display debug information about the current sequence rule if one has been chosen at the start.

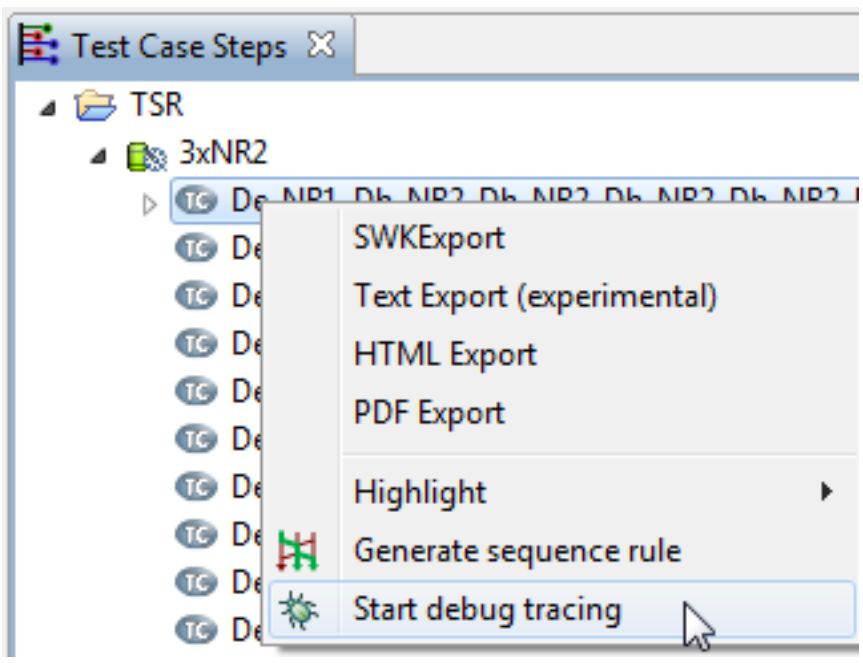


During the execution, the current values of all variables can be checked in the *Variables* view. When a change occurred in the preceding step, the value will be highlighted with a yellow background. It is also possible to look at previous values of variables by selecting earlier steps in the *Trace* view.



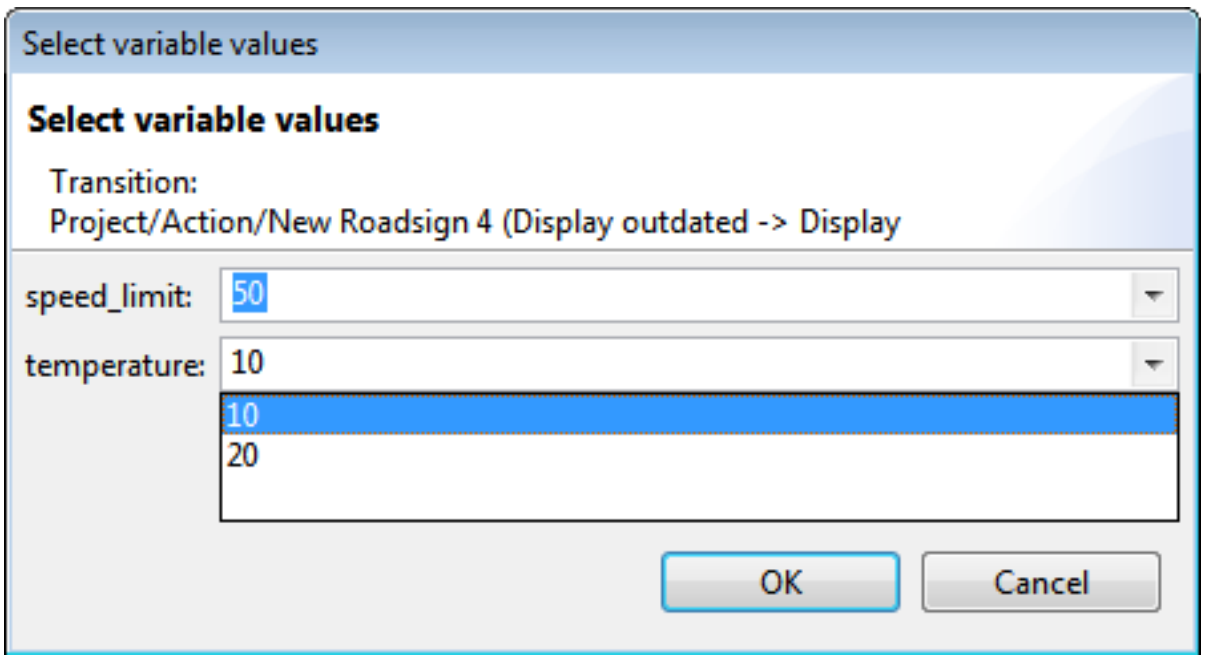
Modellausführung: Variablenwerte

Tests that have already been generated, can also be loaded into the *Trace* view. This allows to inspect it in detail or change the path in the model, to explore other paths.



Modellausführung: Start debug tracing

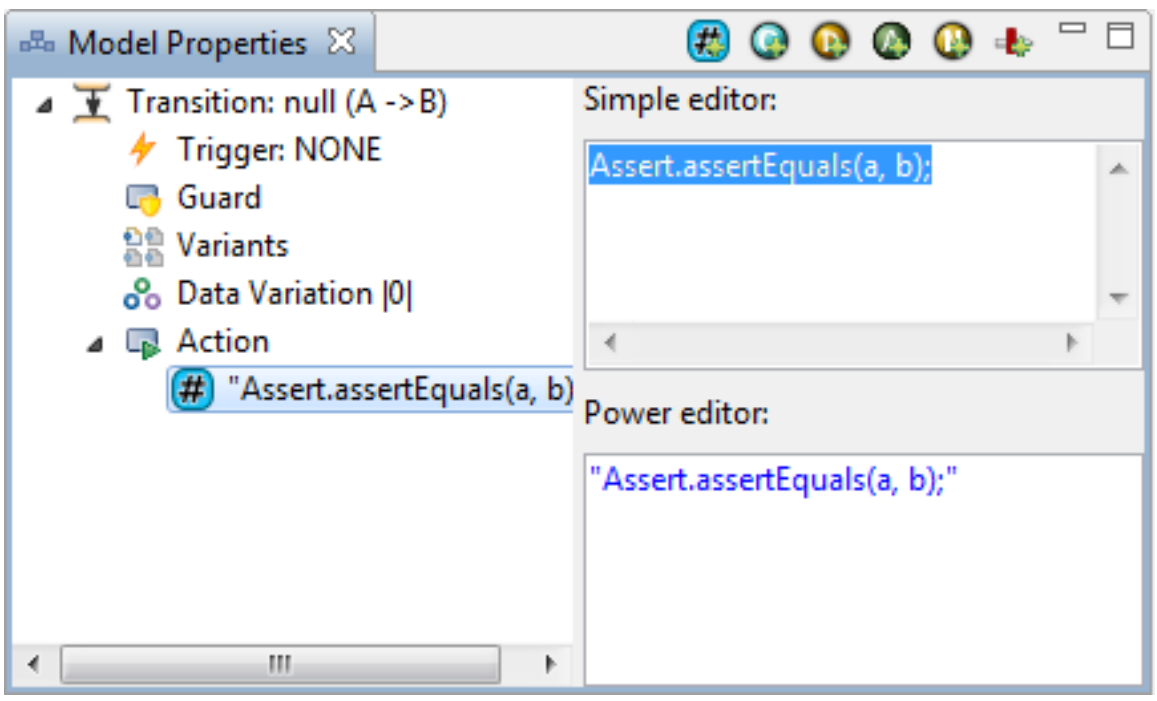
If the model contains Data Variations, the concrete values must be selected in a dialog when stepping:



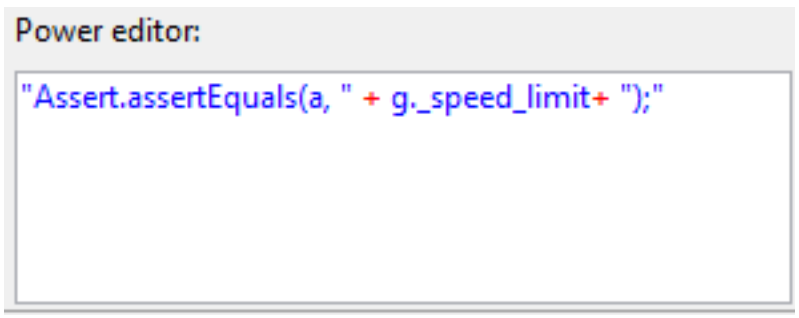
Test generation for other frameworks and languages

There are many text based test specification languages such as Gherkin or JUnit. Although having specific UI for a language or framework is helpful, MODICA can be used to generate tests any text based test specification languages.

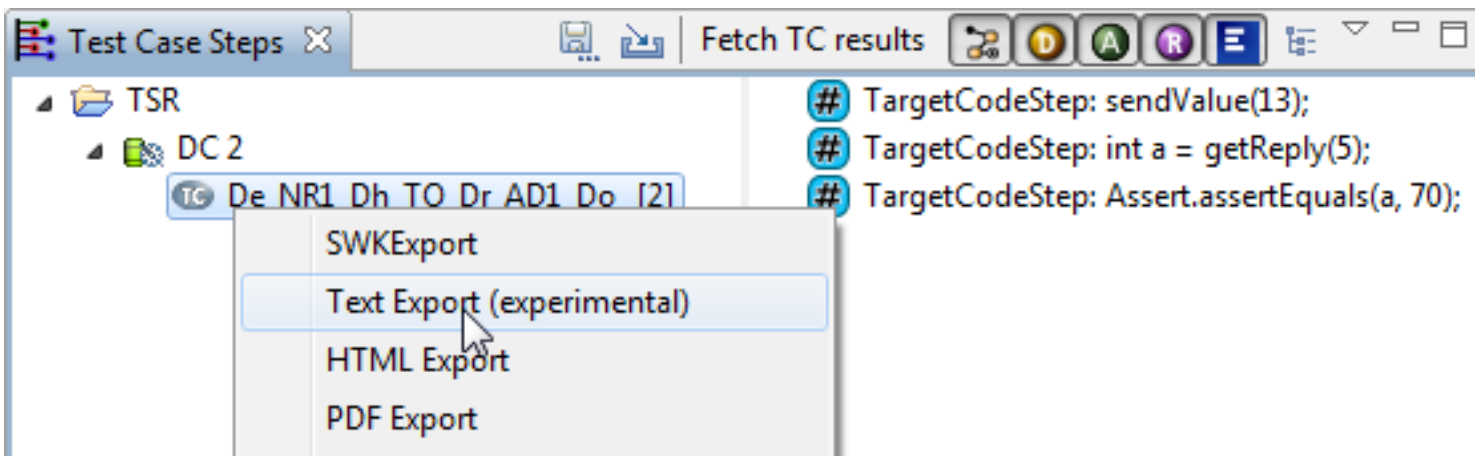
The main step is to add fragments for these execution platforms to diagram elements using *Target Code Elements* in the *Model Properties View* as shown in the below example for JUnit:



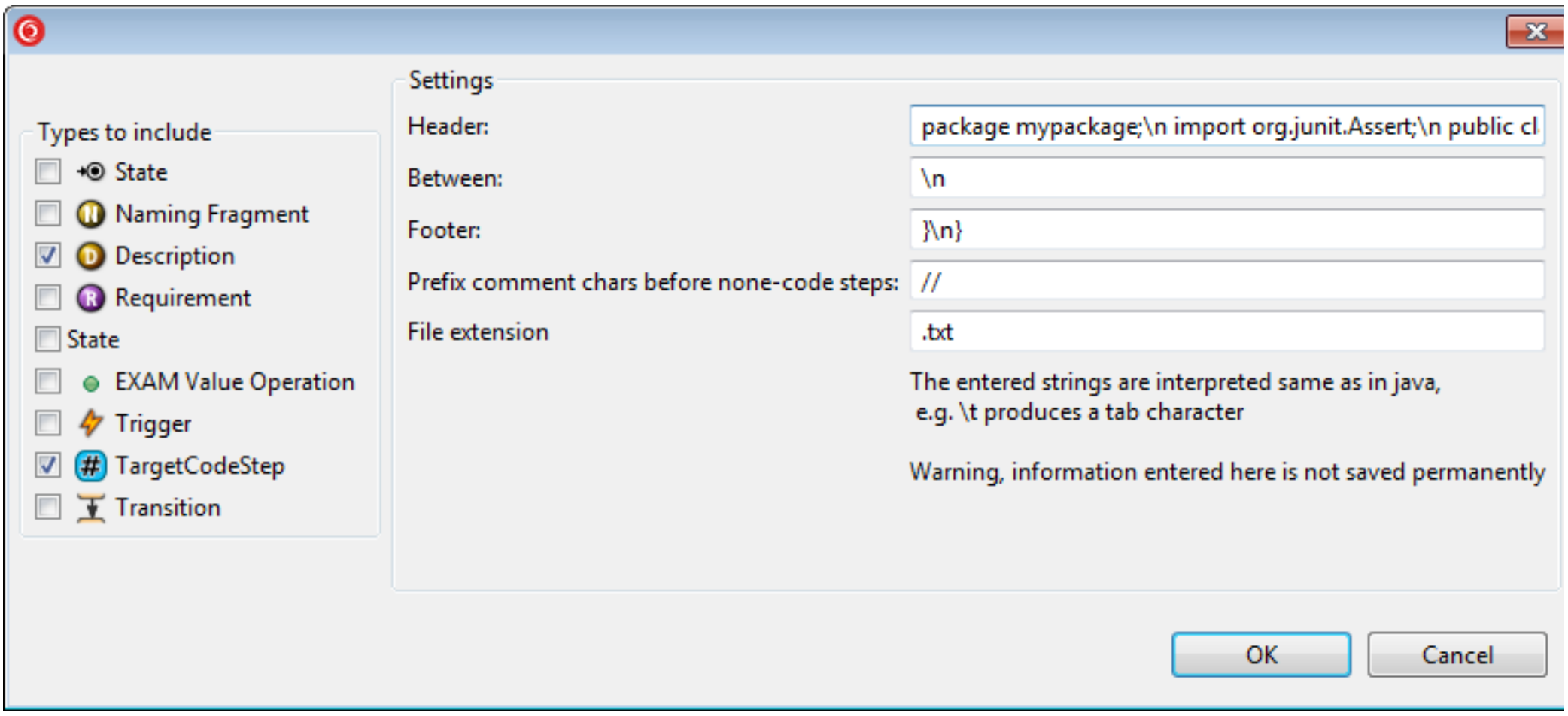
Besides static strings and variables from the target language (a and b in the above example), it is also possible to add dynamic variables form the MODICA Variables library using the *Power editor*:



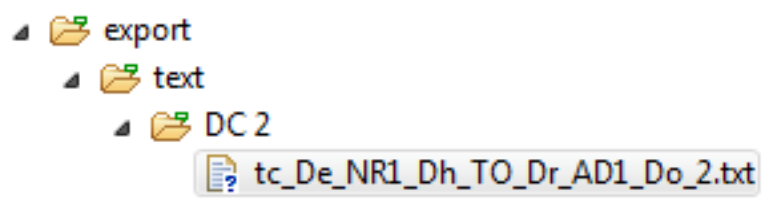
After generating tests they can be exported using the *Text Export* in the *Test Case Steps View*:



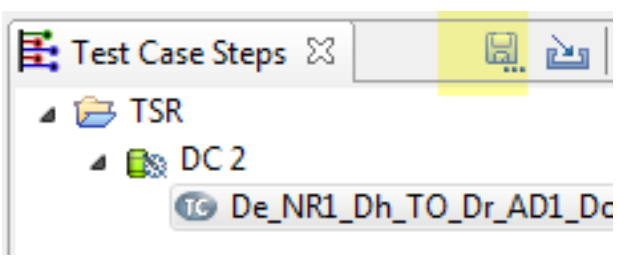
In the following dialog settings like file header can be configured and the exported element types can be selected. In the example, settings have been configured for JUnit. (The contents of TargetCodeElements and the dialog fields should fit for the selected file extension!)



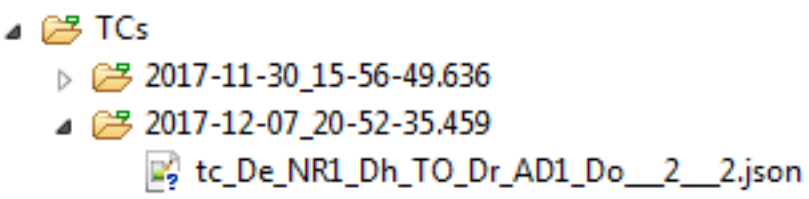
For each selected testcase, this will produce a .txt file in the export folder which can often be enough for a quick prototype.



Besides the simplistic text export, it is also possible to interact with tests saved in the internal format of MODICA in more complex scenarios. Using the save button in the *Test Case Steps View*, you can store MODICA tests on your disk.



They are then stored in a simply JSON format that can easily be parsed using languages such as python, javascript, ... in order to generate tests in arbitrary custom formats.



Command line usage

Some of MODICA's functions are available from the command line.

To access them, run MODICA.exe with additional arguments that will be processed in the given order. Alternatively, you can select a file with one command per line using *Edit → Run Macro File*.

Supported commands

Command	use
-selectProject=...	selects a project (for generation)
-openProject=...	opens a project
-closeProject=...	closes a project
-exit	shuts MODICA down
-generate	generates tests
-selectElement=...	tries to find and select a diagram element in the editor - the XMI-ID must be provided ℹ fails silently if the ID is not found ℹ only supported for transitions, states and pseudo states
-importDiagram=...	imports the diagram from a given xml/uml/xmi file into the currently selected BasicState (→ -selectElement) ⚠ For some imports, dialogs may be shown that can not be automated yet (e.g. select statechart if more than one is found)
-exportTests=JSON|HTML|TEXT|XML	exports generated tests (entire project using default settings) ℹ JSON export uses the folder TCs instead of export ℹ TEXT will store all elements if no selection has been configured
-runFile=...	runs commands from a given file instead ⚠ all other commands are ignored ⚠ can not be used in a command file itself

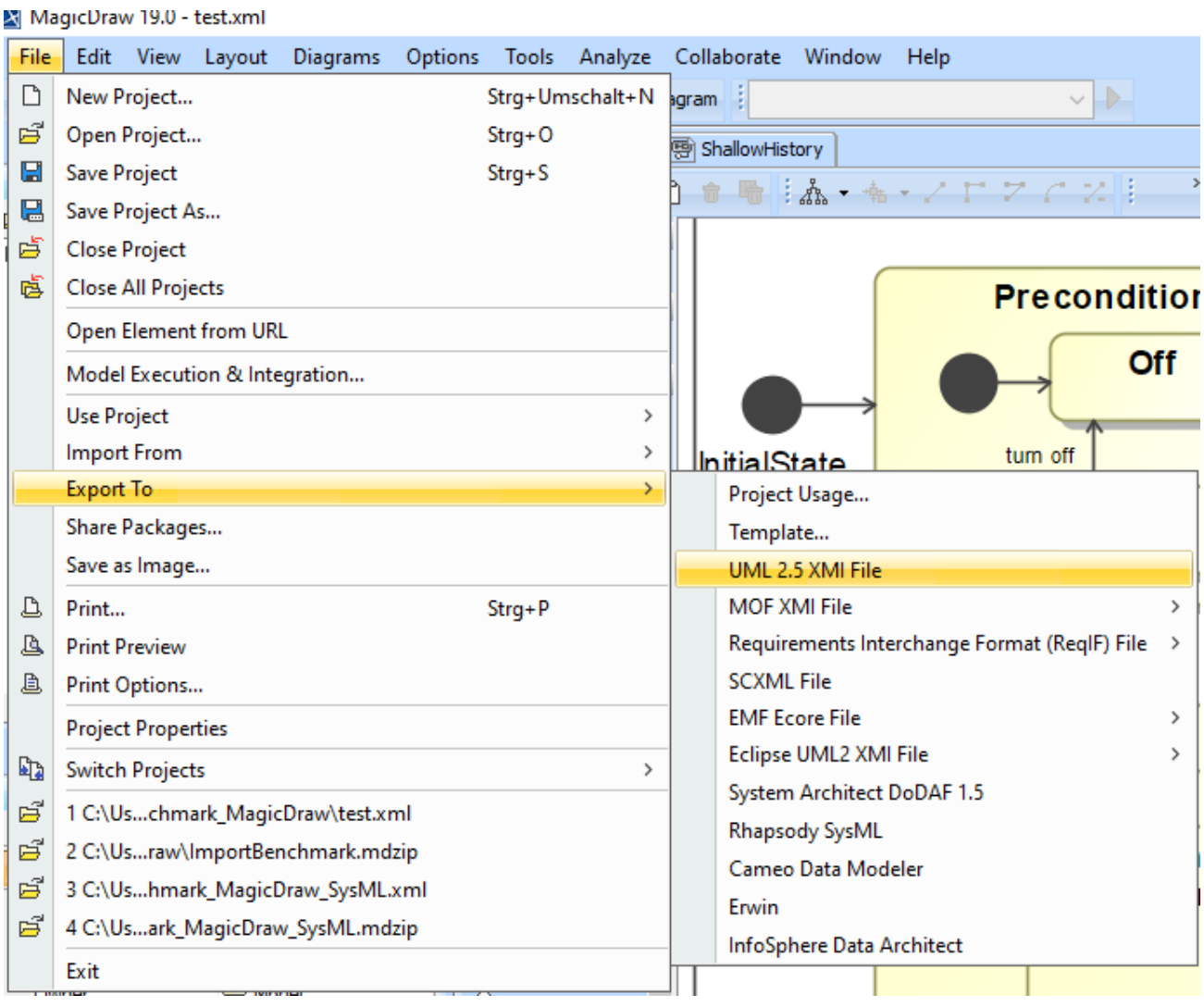
⚠ any arguments after = must not contain spaces when passed from command line - when read from a file, spaces are ok

Import of models from MagicDraw

Since version 4.2.0 MODICA supports the import of state chart diagrams from MagicDraw. When modeling in MagicDraw, the user should follow the following points in order to continue working in MODICA as effective as possible. This documentation relates to MagicDraw 19.0.

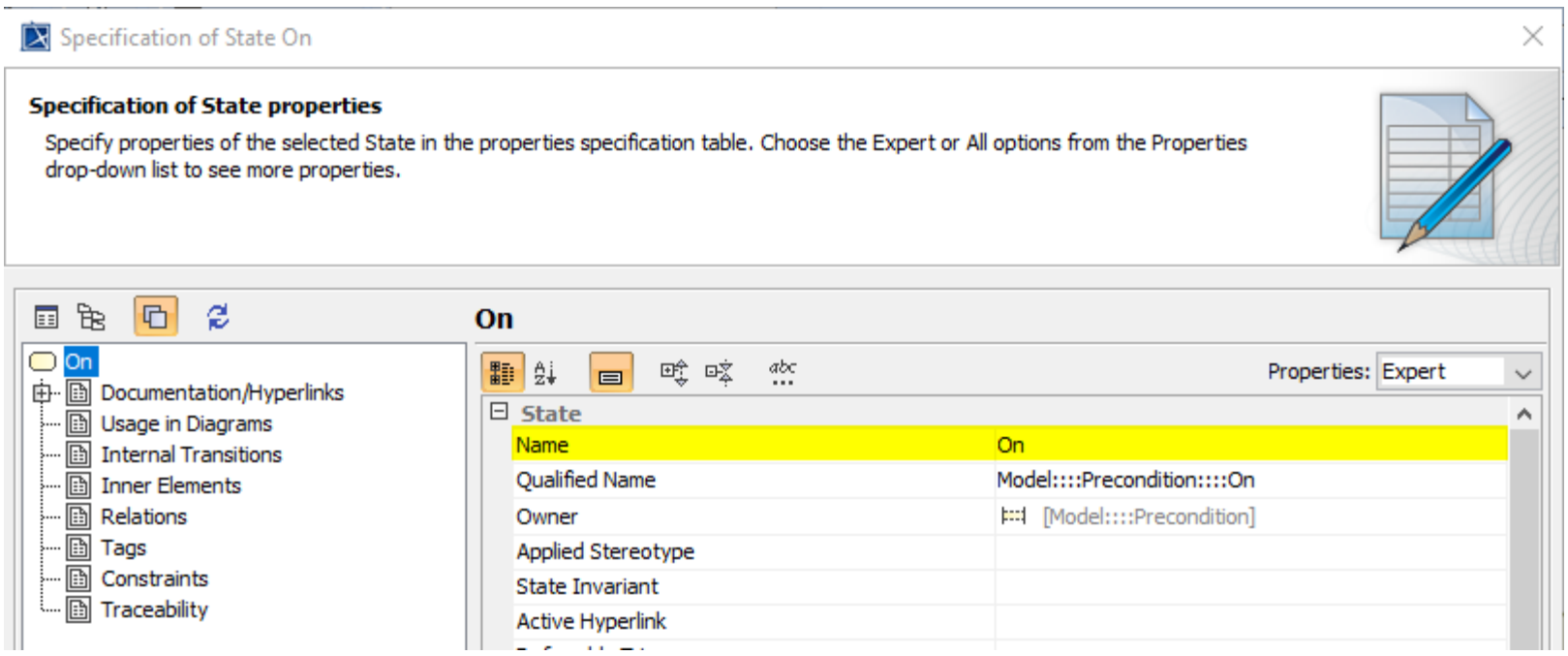
Export

The recommended export format for state machines is 'UML 2.5 XMI' (see following image).

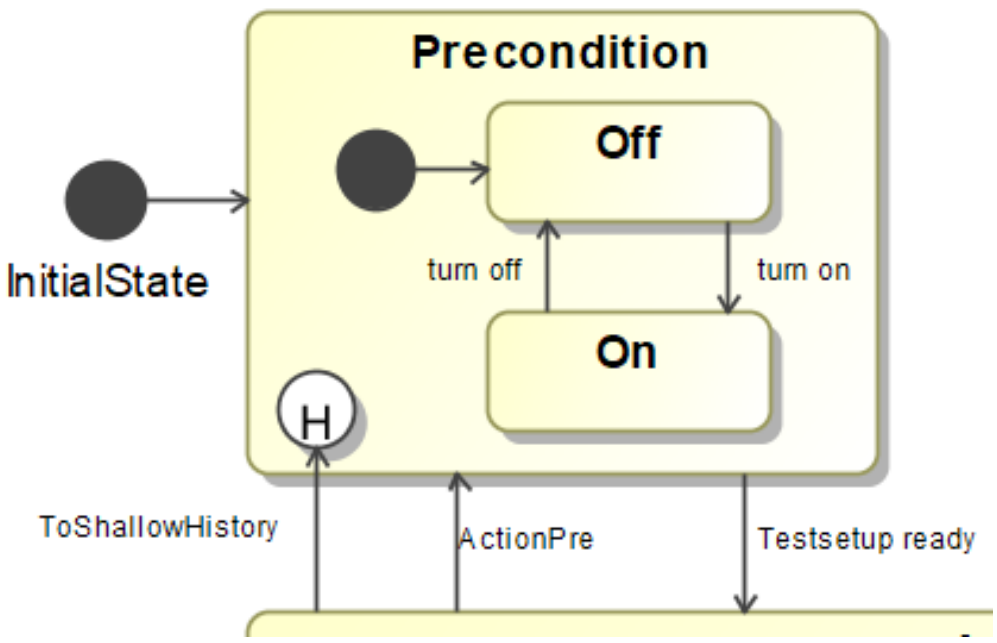


States and Transitions

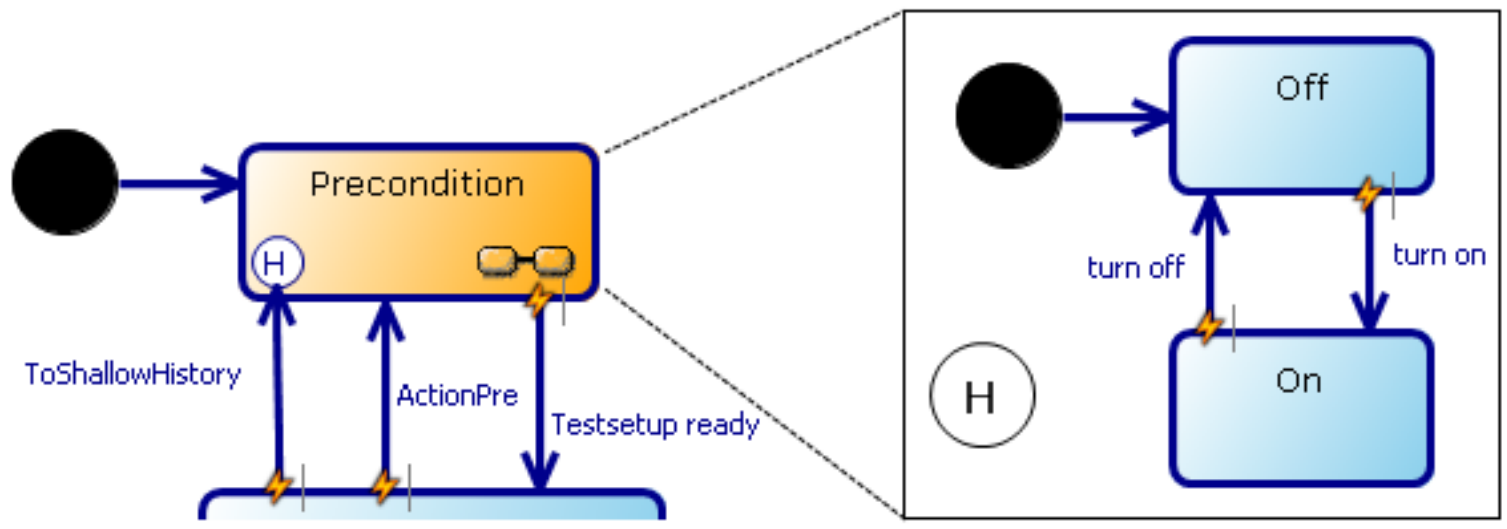
In MagicDraw, all kinds of states can be used, that are supported by MODICA. The names of initial states and shallow/deep history states are not imported. All other states get their names from the 'Name'-field of their Properties (see following image). Should this field be empty, the state gets his XMI-ID for a name. Transition names are taken from the same field. If it's empty, the transition remains nameless.



The following image shows a 'Composite State' in MagicDraw ...



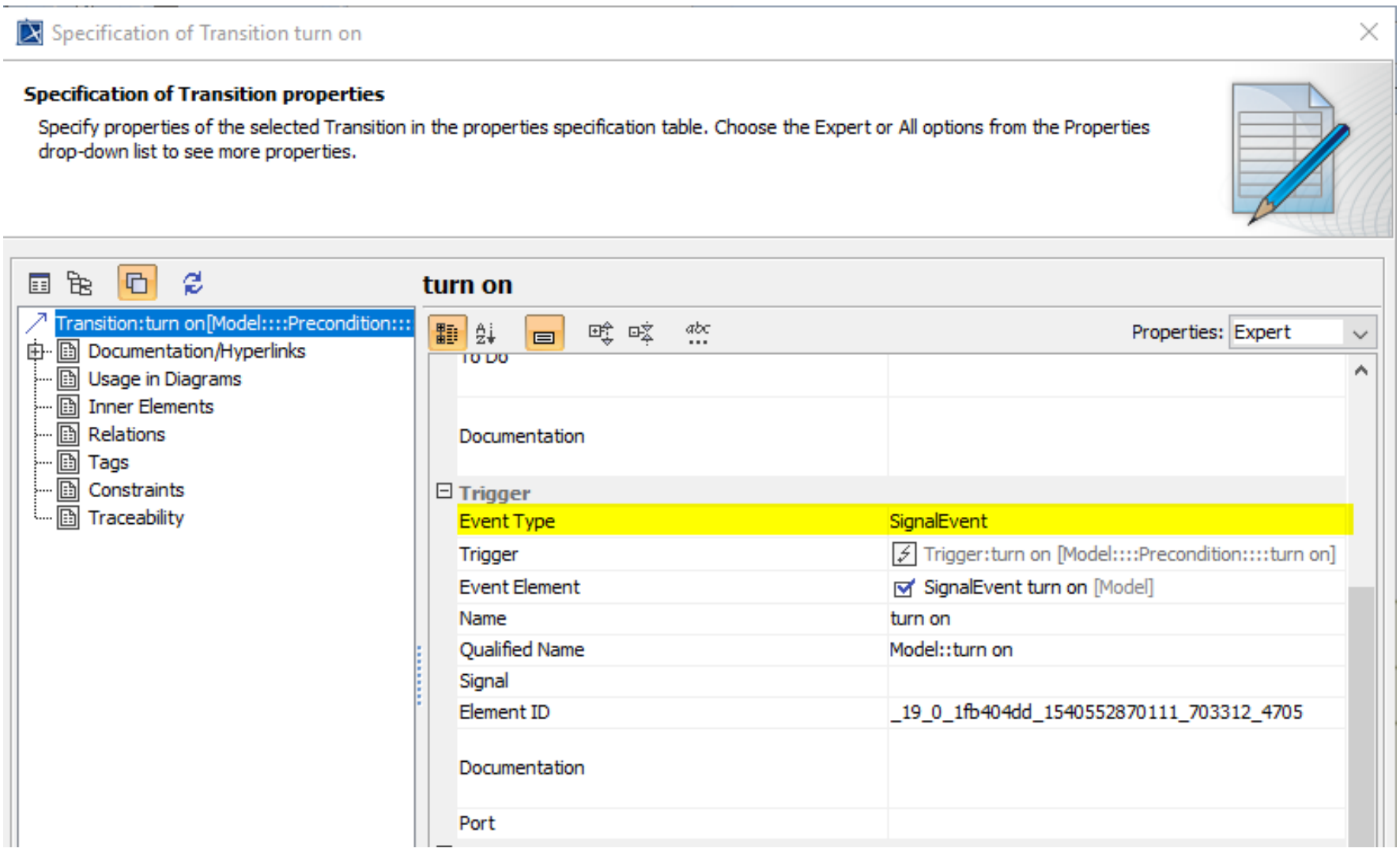
and its equivalent in MODICA:



Trigger

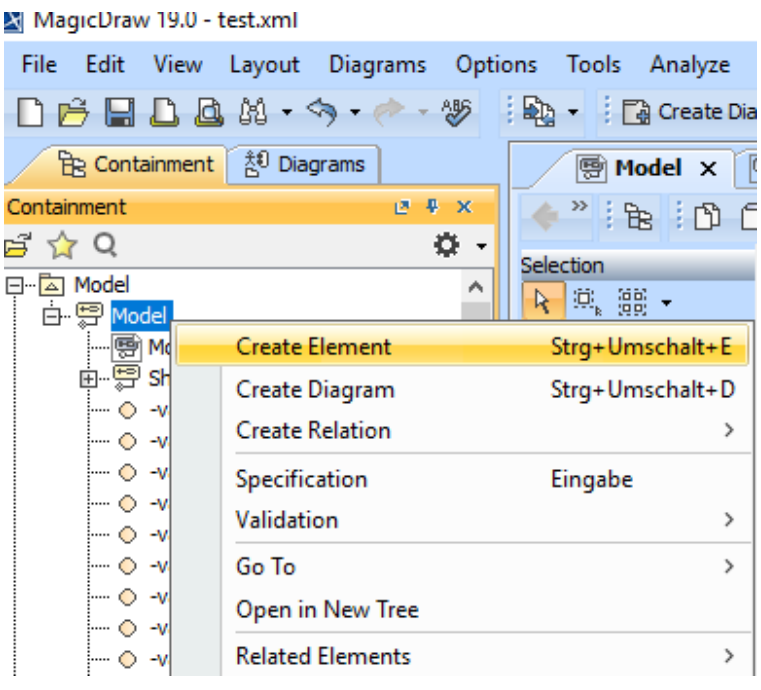
The MagicDraw-equivalent to a trigger in MODICA is the SignalEvent. A SignalEvent can be added to a transition, by opening its properties and selecting 'SignalEvent' in the 'Event Type'-field in the section 'Trigger' (see following image). The trigger can be given a name, but it won't be imported to MODICA. The 'Event Element'-field is automatically set with a

SignalEvent, that has the name of the transition. It can also be set with an already created SignalEvent. Thus, after the import to MODICA, multiple transitions can have the same trigger. Here, it's still recommended, that triggers and transitions have a 1:1-relation.

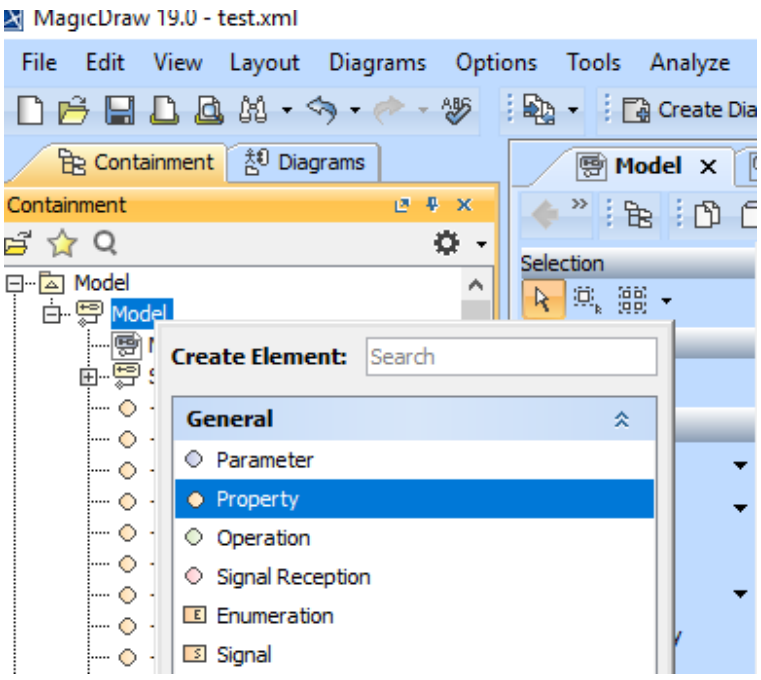


Variables

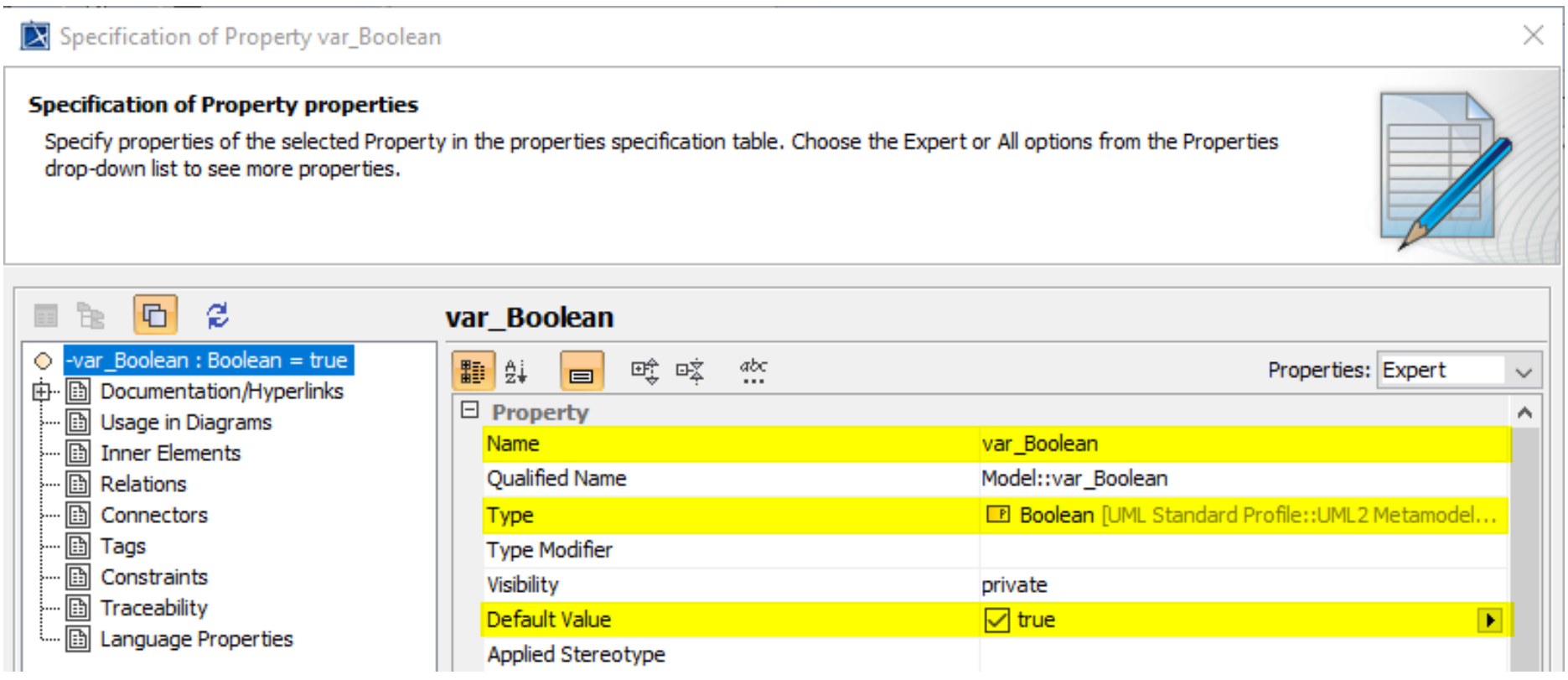
A variable in MODICA corresponds to a Property in MagicDraw. A property is created in the 'Containment'-tree. With a right-click on the 'State Machine'-node and another click on 'Create Element' ...



a menu is opened in which 'Property' is chosen.



In the property's properties its name, type and default value can be set.



At the import MODICA tries to give the Variable a fitting type. The following table shows the assignments. Should there be an unknown or no type on hand the variable gets the type 'String'.

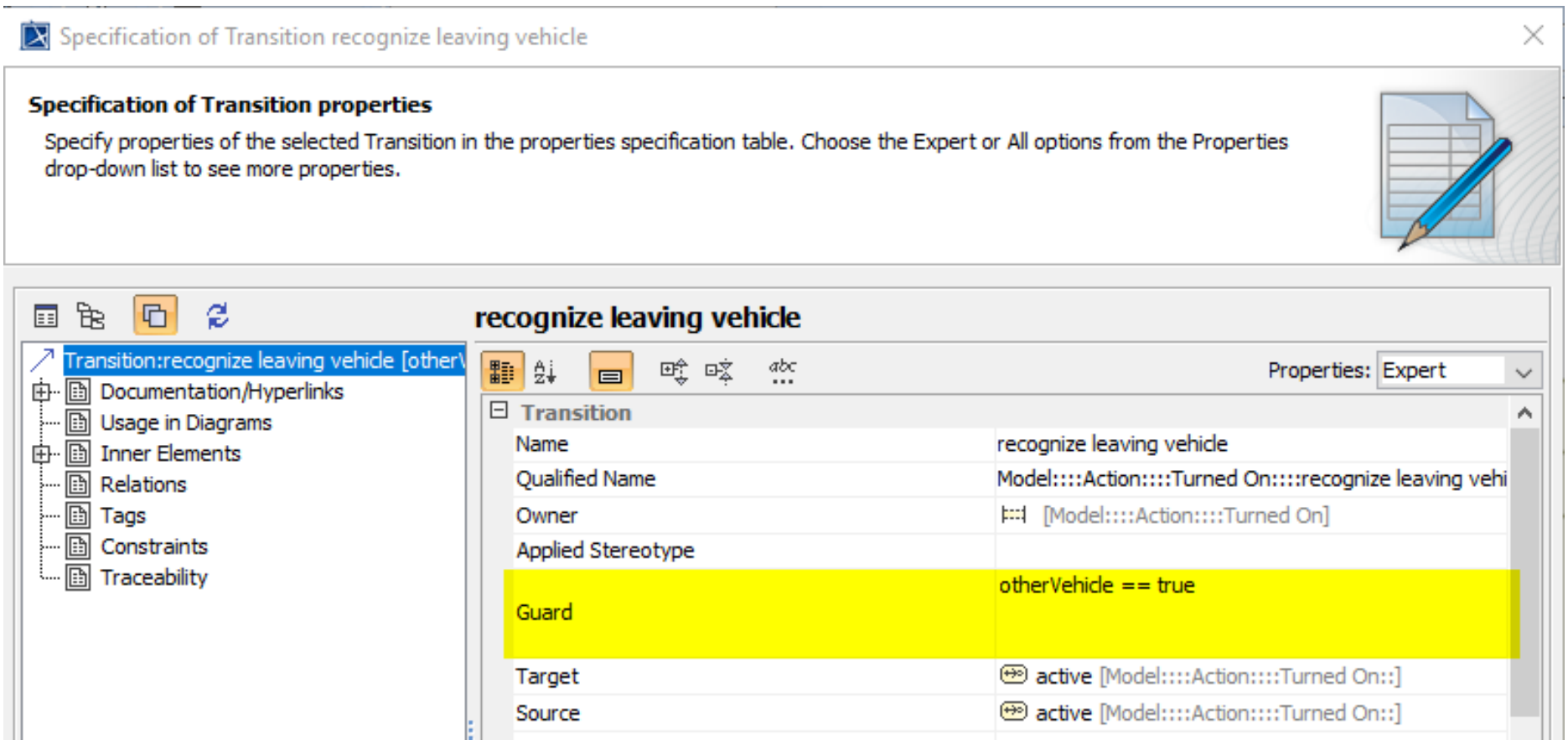
MagicDraw	Modica
boolean, Boolean	Boolean
int, Integer, UnlimitedNatural, short, long	Integer
float, Real, double	Float
String	String

If there's no default value available or if it doesn't fit the assigned data type in MODICA, a initial value is given.

Guards

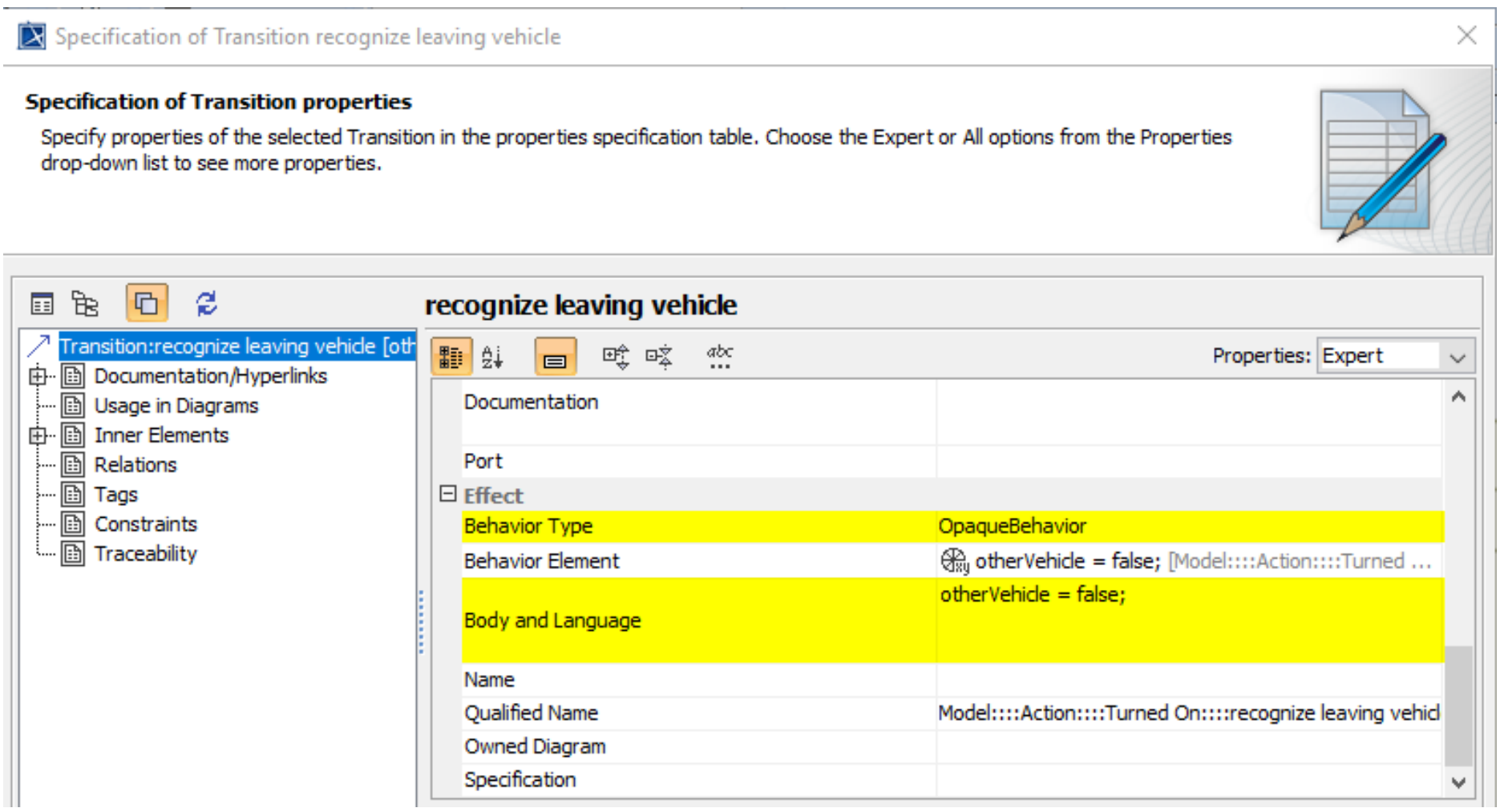
Guards can be defined in the properties of a transition. The guard in the following image contains the identifier 'otherVehicle'. If there's a property (see section 'Variables') with this name in the state machine, the guard is modified at the import. Because a variable 'otherVehicle' is created at the import, the content of the guard changes to 'g._otherVehicle ==

true'. The 'g' represents the scope of the variable within the MODICA-project.



Calculations

In MagicDraw, MODICA's calculations are defined in the properties of a state (sections 'Entry' and 'Exit') or a transition (section 'Effect') (see following image). For this purpose, 'OpaqueBehaviour' is selected as 'Behaviour Type' and the calculation's content is inserted in 'Body and Language'. As with the guards, the identifier of an available variable gains a leading 'g._'.

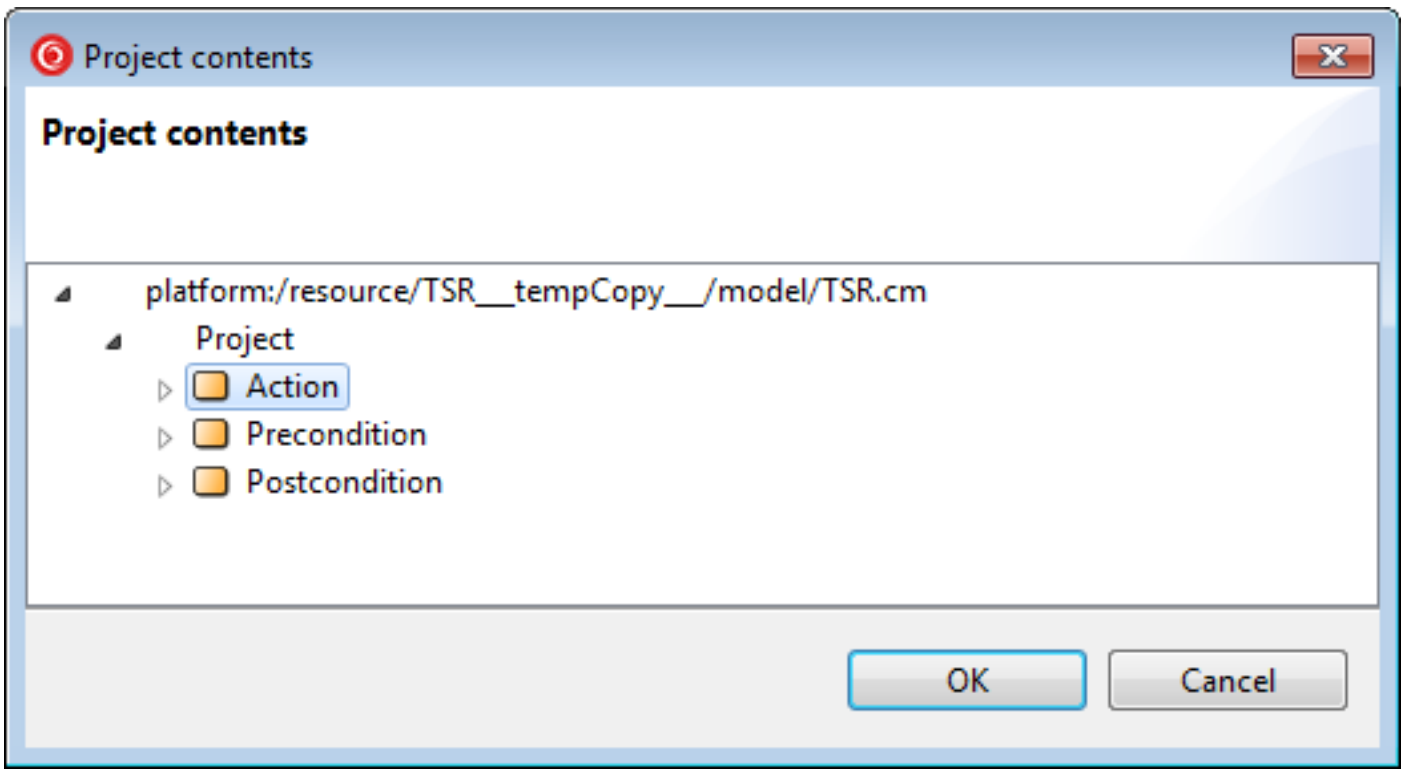


Modellimport with Merge

If a model from an XMI file is changed externally (e.g. in a modelling tool like Magic Draw), it can be re-imported in MODICA. As the model may have been changed in MODICA since the last import, a merge functionality has been added to integrate these two changesets.

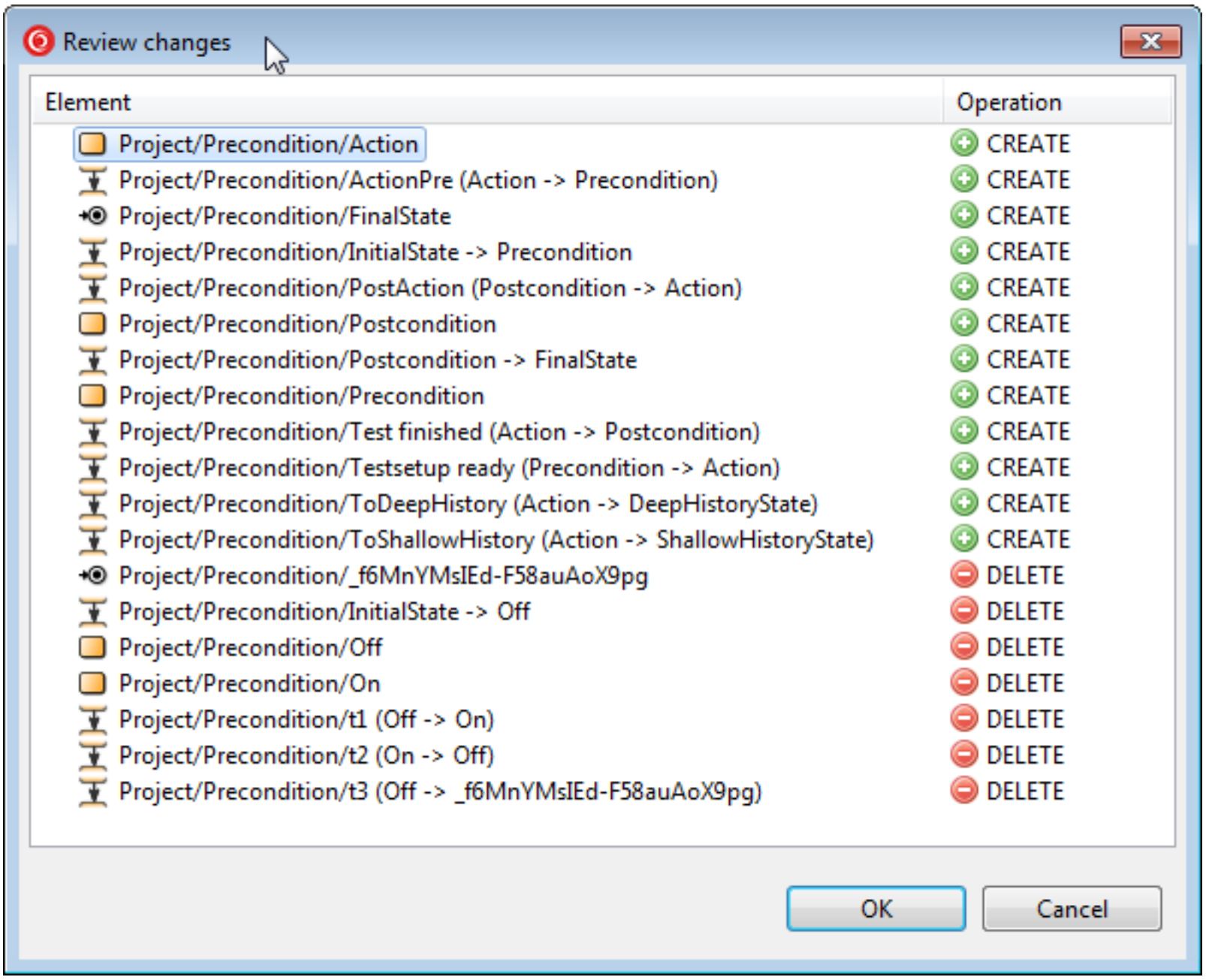
This merging importer can be started through the right-click-menu of a project (*Project-Explorer* → right click on Project → *Import XMI and merge with changes*)

In the first step, the user is asked where in the model the import should be performed:



The second step requires the user to select the xmi file to be merged. Depending on the file, it may be necessary to select one of multiple statecharts contained in it.

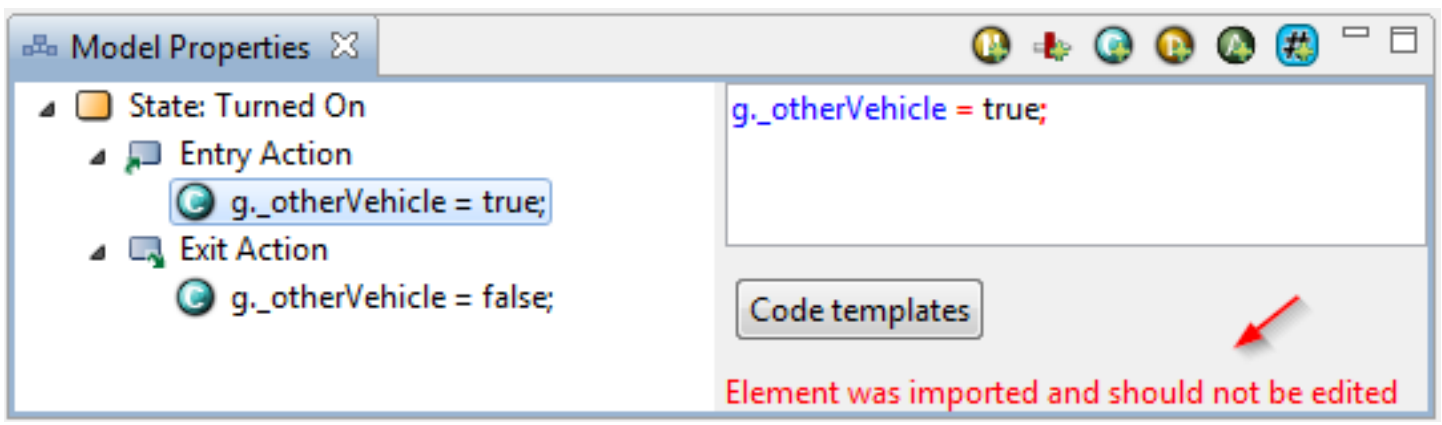
Afterwards, the import is started and a comparison dialog is shown that describes the changes to be performed.



Depending on the scenario, different entries may appear here and some of them can also have sub-entries (when changes are merged that affect the *Model Properties View*):

- **CREATE**
  An element will be created newly (did not exist in MODICA before the import)
- **UPDATE**
  An existing Element is to be changed as part of the merge. (The check for this is very conservative - all elements touched by the import have at least this status)
- **DELETE**
  A previously imported element is no longer part of the Import file and will be deleted.

The *Model Properties View* shows elements that were created by an import with a warning:



When these elements are changed in MODICA, a future import may overwrite the changes during DELETE or UPDATE steps.

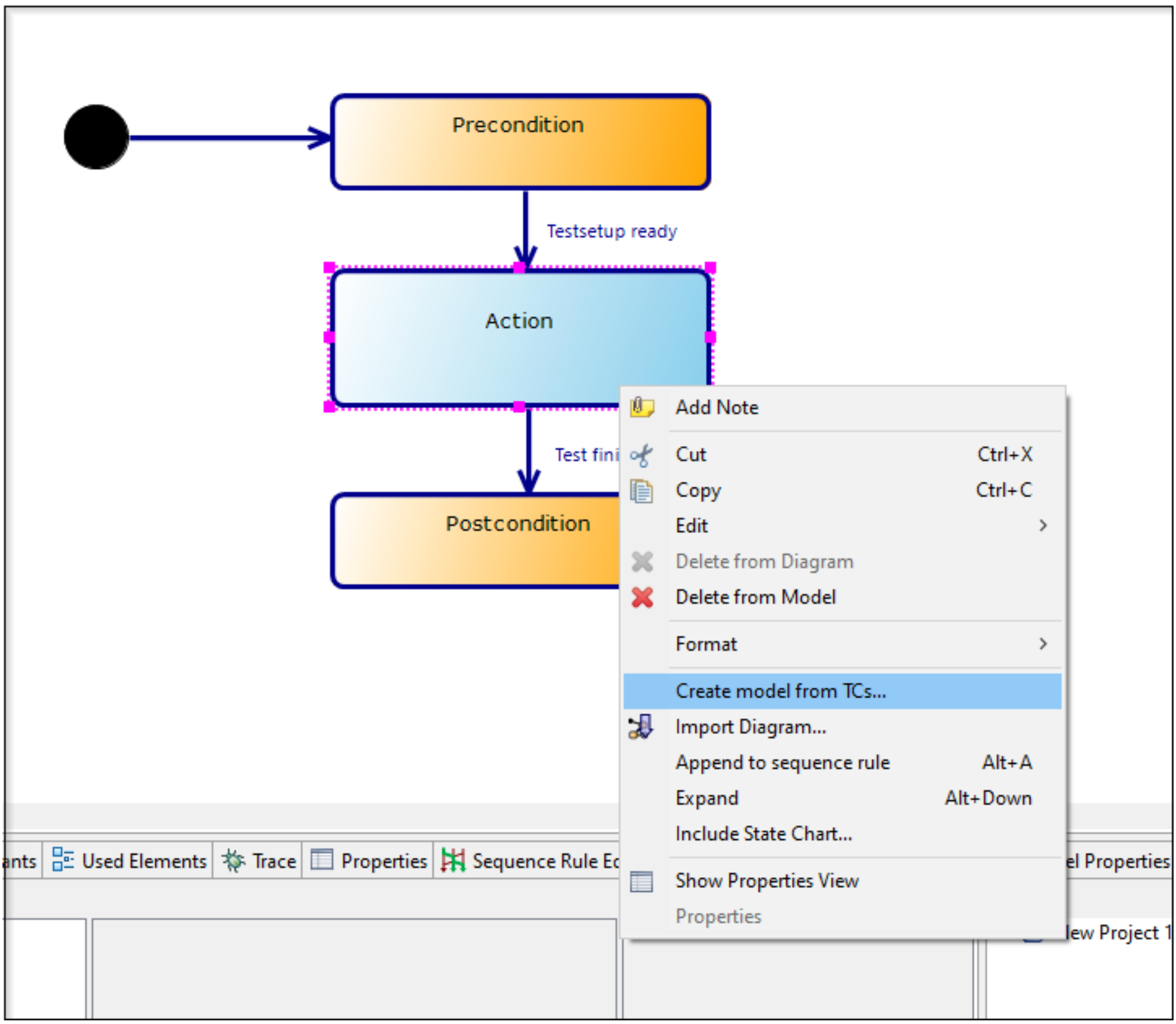
Model generation from existing test cases

MODICA supports creation of state machines by reading saved test cases. Chains of states will be created from the read test cases which consist of the test steps of the read test cases. The transitions between the states contain test steps in the actions blocks. A chain will be created from every test case which can be established by a transition from the start state. The test case generator is able to create test cases which in turn correspond to the sequence of the read test cases. The following information will be processed during the creation of the test cases and transferred to the model:

- Descriptions
- Requirements
- Calls of the test execution enviroment

The generated model can be extended with further information (such as: links to requirements). It is also possibility to change the model to populate new functions or improve the model.

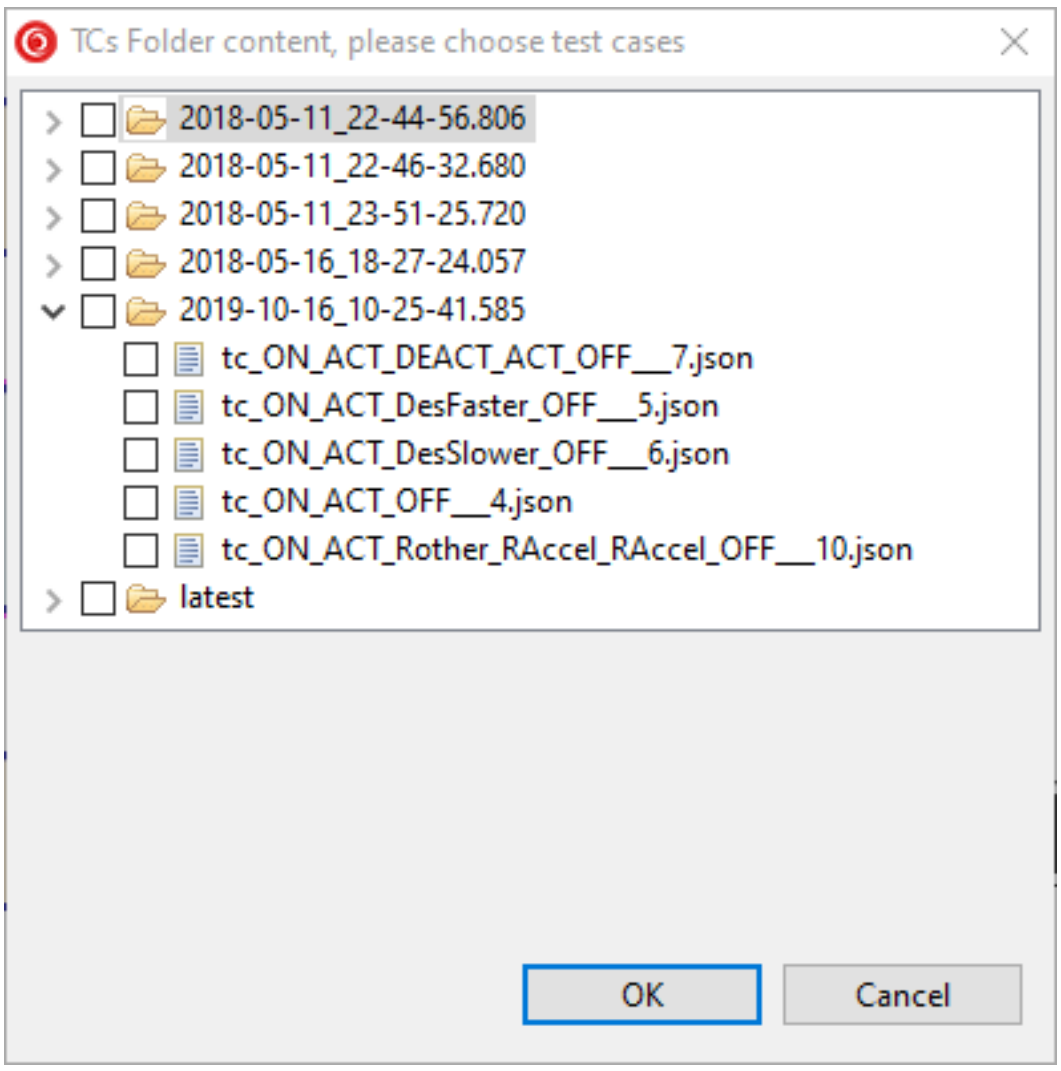
To create the model from test cases, an empty state will be selected first and the menu item "Create model from TCs..." will be executed.



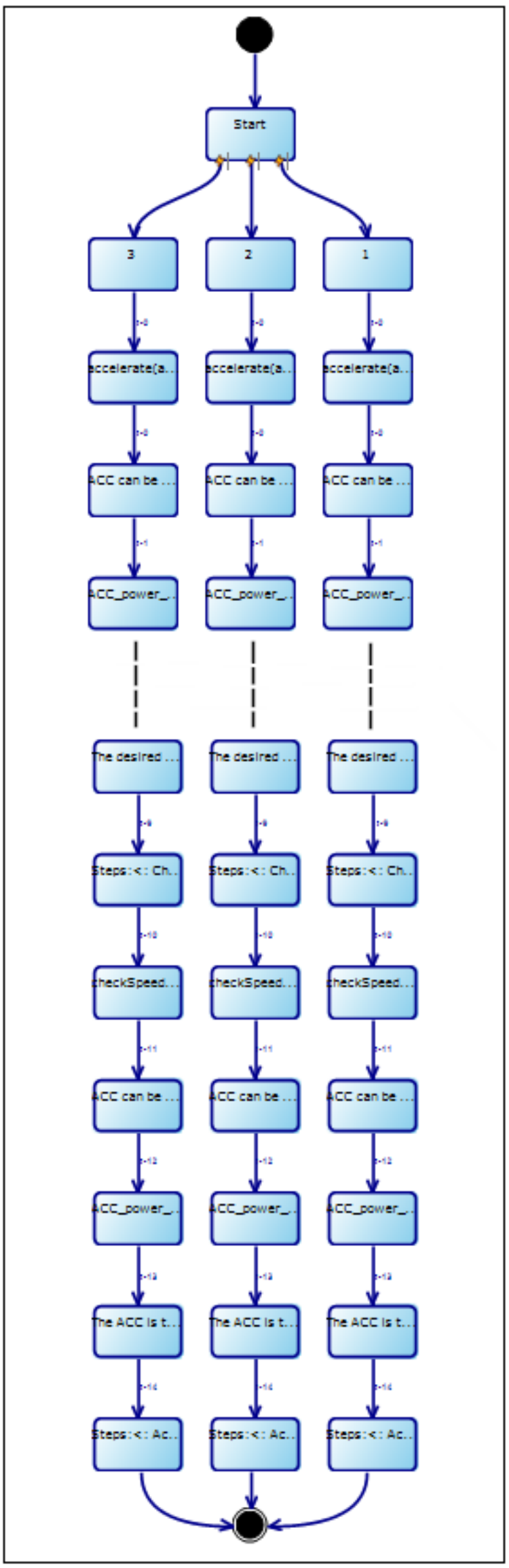
As a result a dialog will be opened that displays the test cases in the "TCs" folder in the project tree. The folder "TCs" will be created in case that test cases will be stored or generated.

The test cases can be selected separatly. Alternatively, the parent folder can be selected to select all test cases of the folder.

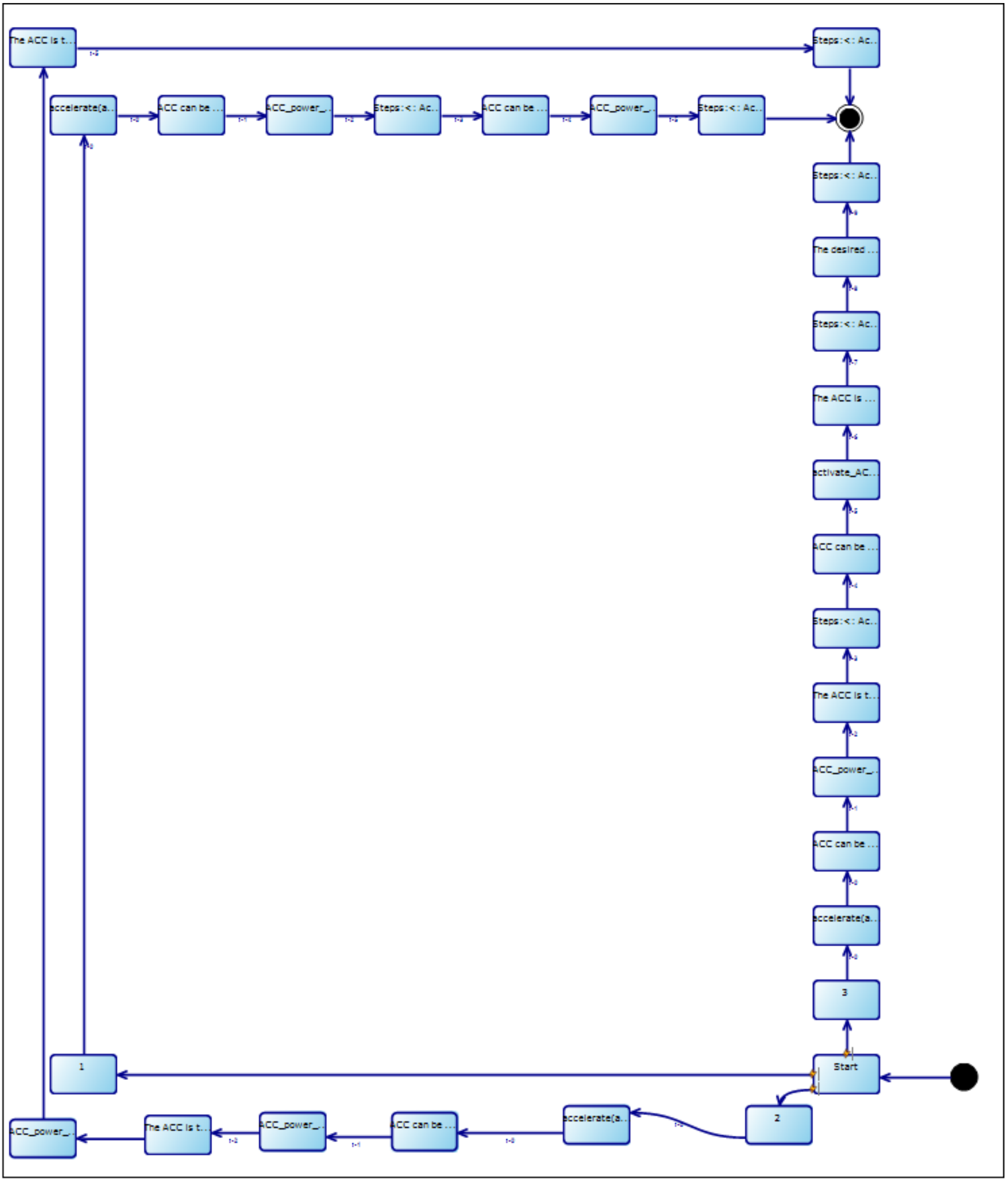
If the test case with the same name already exists in the list or has already been selected, this test case will not be selected. The name of the test case has to be unique.

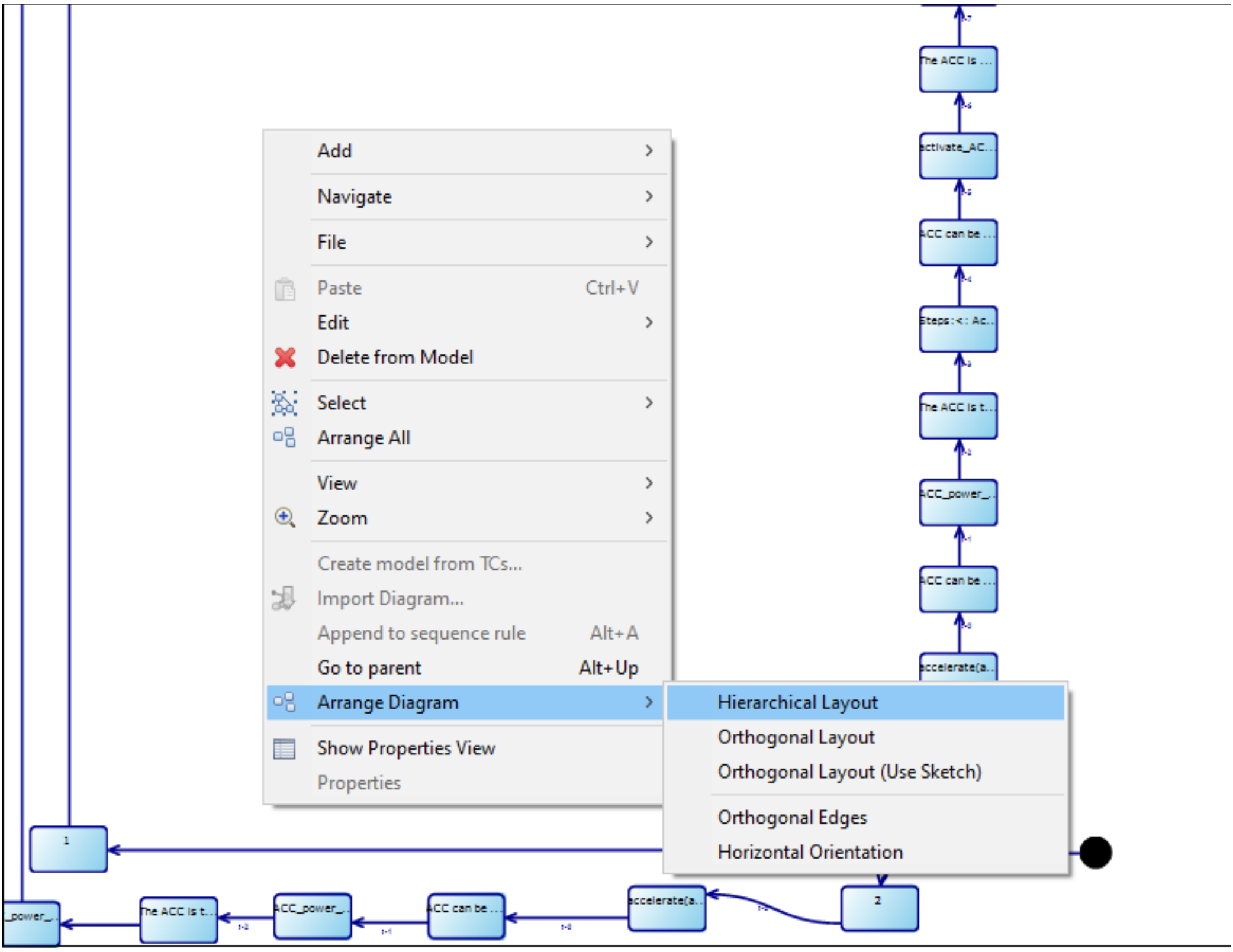


After the test cases were selected and confirmed with OK button, the model will be generated in the selected state.



If the states are not arranged correctly, the layout can be adapted over the context menu of the diagram "Arrange Diagram → Hierarchical Layout".


The ACC is t...
Steps:<: Ac...
accelerate(a...
ACC can be ...
ACC_power_...
Steps:<: Ac...
ACC can be ...
ACC_power_...
Steps:<: Ac...
Steps:<: Ac...
The desired ...
Steps:<: Ac...
The ACC is ...
activate_AC...
ACC can be ...
Steps:<: Ac...
The ACC is t...
ACC_power_...
ACC can be ...
accelerate(a...
3
1
Start
2
ACC_power_...
The ACC is t...
ACC_power_...
ACC can be ...
accelerate(a...


Add
Navigate
File
Paste Ctrl+V
Edit
Delete from Model
Select
Arrange All
View
Zoom
Create model from TCs...
Import Diagram...
Append to sequence rule Alt+A
Go to parent Alt+Up
Arrange Diagram
Show Properties View
Properties
Hierarchical Layout
Orthogonal Layout
Orthogonal Layout (Use Sketch)
Orthogonal Edges
Horizontal Orientation
The ACC is ...
ctivate_AC...
ACC can be ...
Steps:<: Ac...
The ACC is t...
ACC_power_...
ACC can be ...
ccelerate(a...
1
2
power...
The ACC is t...
ACC_power_...
ACC can be ...
ccelerate(a...

Attachment

Shortcuts

Navigation in the Editor

Action	Shortcut
Create/select subdiagram	Alt + DOWN
Select the parent diagram	Alt + UP
Zoom +/-	Ctrl + Scrolling the mouse wheel

Model Properties View

Action	Shortcut
Change focus from the *Editor* to the *Model Properties View*	Alt + Shift + Return
Add *Attribute Modification*	Ctrl + Shift + A
Add *Calculation*	Ctrl + Shift + C
Add *Description*	Ctrl + Shift + D
Add *Naming Fragment*	Ctrl + Shift + N
Add *Visit*	Ctrl + Shift + V

Stepping - Sequence Rules

Action	Shortcut
Start Stepping	F11
Next Step	F6

Execution platform integrations