



# ERTMSFormalSpec Workbench User Guide

Version 0.9.0

# 1 Table of contents

1	Table of contents .....	2
2	Revision history .....	4
3	Introduction .....	5
3.1	References .....	5
3.2	Terms and definitions .....	5
4	EFSW Introduction .....	6
4.1	The language .....	6
4.2	The workbench .....	6
4.3	Workbench installation .....	7
4.4	Initial workbench state .....	7
4.4.1	Opening a EFS file .....	7
5	Subset-026 Specification browser .....	8
5.1	Opening the specification browser .....	8
5.2	Specification browser description .....	8
5.2.1	Overview .....	8
5.2.2	Requirements identification .....	10
5.2.3	Tools related to the specification view .....	11
5.3	Process related to requirements .....	12
5.3.1	Review the requirements .....	12
5.3.2	Create the model for the requirement .....	13
5.3.3	Update the model of an Implemented requirement .....	13
6	Data dictionary browser .....	14
6.1	Opening the data dictionary browser .....	14
6.2	Data dictionary browser description .....	14
6.2.1	Overview .....	14
6.2.2	Tools related to the data dictionary view .....	15
6.2.3	Shortcuts view .....	16
6.3	Name spaces .....	16
6.4	Data types .....	17
6.5	Procedures .....	18
6.6	Variables .....	19
6.6.1	Sub variables .....	20
6.7	Functions .....	20
6.8	Rules .....	21
6.8.1	Specific case: a rule declared in a state .....	23
6.8.2	Pre-conditions .....	24
6.8.3	Actions .....	24
6.9	Model validations .....	24
6.10	Expressions .....	27
7	State diagrams .....	28
7.1	Display a state diagram .....	28
7.2	Manipulations of a state diagram .....	29
7.2.1	Add a new state .....	29
7.2.2	Add a new transition .....	29
7.2.3	Selecting element in a state diagram .....	29
8	Test browser and execution environment .....	30
8.1	Opening the test browser .....	30
8.2	Test browser description .....	30

8.2.1	Overview .....	30
8.2.2	Tools related to the test view .....	31
8.2.3	Test actions .....	31
8.3	Test step .....	31
8.3.1	Actions .....	32
8.3.2	Expectation .....	32
8.4	Test step execution.....	32
8.5	Test Case, Sub sequence and Frame execution .....	34
8.6	Time Line.....	35
9	Translations.....	37
9.1	Open the translation view .....	37
9.2	Translation view description.....	37
9.2.1	Overview .....	37
9.2.2	Tools related to the translation view .....	38
9.3	Apply translation rules .....	39
10	Reports .....	40
10.1	Specification coverage report .....	40
10.1.1	Purpose.....	40
10.1.2	Structure.....	40
10.1.3	Launch the specification reporting.....	41
10.1.4	Spec issues report.....	43
10.2	Dynamic tests coverage report.....	43
10.2.1	Purpose.....	43
10.2.2	Structure.....	43
10.2.3	Launch the test coverage reporting .....	44
10.3	Data dictionary report .....	45
10.3.1	Purpose.....	45
10.3.2	Structure.....	45
10.3.3	Launch the model report .....	46

## 2 Revision history

Version	Date	Name	Description	Paragraphs
0.2.1	15/12/2010	Stanislas Pinte	First version	All
0.2.2	15/12/2010	Laurent Ferier	Document revision	All
0.2.3	06/01/2011	Svitlana Lukicheva	Added in/out variables	5.3, 6.4
0.3.1	28/01/2011	Laurent Ferier	Document revision according to release 0.3	All
0.3.2	21/02/2011	Svitlana Lukicheva	Added chapter "Reports"	8
0.4.0	04/03/2011	Laurent Ferier	Document revision according to release 0.4	All
0.5.0	05/04/2011	Laurent Ferier	Document revision according to release 0.5	All
0.6.0	13/05/2011	Svitlana Lukicheva, Laurent Ferier	Document revision according to release 0.6	All
0.7.0	17/10/2012	Svitlana Lukicheva, Laurent Ferier	Document revision according to release 0.7	All

### 3 Introduction

This document is a User's Guide to the ERTMSFormalSpecs Workbench (EFSW). It details the features and functions of the EFSW.

#### 3.1 References

Index	Title	Date
[1]	EFSW Technical design	15/12/2010

#### 3.2 Terms and definitions

Term	Definition
BL	Baseline
ESFW	ERTMSFormalSpecs Workbench
EFS	ERTMSFormalSpecs

## 4 EFSW Introduction

### 4.1 The language

The ERTMSFormalSpecs language is an ad-hoc language, developed by ERTMS Solutions for the sole purpose of modeling the ERTMS specifications.

The EFS model is made up of a data dictionary, containing all the variables of the model, a set of rules and other modeling elements, organized hierarchically. Even though the model is not mathematically very sophisticated, its complexity lies in the sheer amount of rules and variables.

The EFS model is a non-trivial artifact (estimated at 2000 rules, 500 variables, 1500 tests, 500 pages of specifications), far too large and complex to be managed without ad hoc tools.

### 4.2 The workbench

The EFSW is a graphical tool designed to develop, maintain, and document model-based development for the Subset-026. It is a desktop application, running on the Microsoft Windows platform.

The Workbench provides high-level features:

- Data dictionary browser
- Test browser and execution environment
- Subset-026 Specification browser
- Managing full specification traceability
  - between declarations (type, functions and variables) and specifications
  - between rules and specifications
- Managing full traceability between test cases and the requirements they verify. The workbench can also be used to determine the rules exercised by a specific test case.

The EFSW provides all features and functions necessary to support these high-level features, explained in details in the following sections.

The EFSW is composed of three main windows, which provide a different view on the modeled system

- The specification window provides a view of all the requirements related to the system. See Section 5 for further information about the specification window.
- The data dictionary window is used to model the system. It allows to create types, variables and procedures, functions. Section 6 further details this window.
- The test window allows to specify tests on the modeled system, allows executing them and provides reports on the executed tests. See Section 7 for further information about the test window.

### 4.3 Workbench installation

The complete EFS environment is distributed in a setup file which automatically installs the tool in the installation directory

`C:\Program Files\ERTMSSolutions\ERTMSFormalSpecs`

This can be changed during the setup phase. The installation process creates icons in the start menu to launch the EFS workbench and access the documentation.

The installer also checks that the required .NET framework 4.0 is installed on the target machine before performing the installation.

### 4.4 Initial workbench state

When the application is run, the display is empty, as depicted in Figure 1.

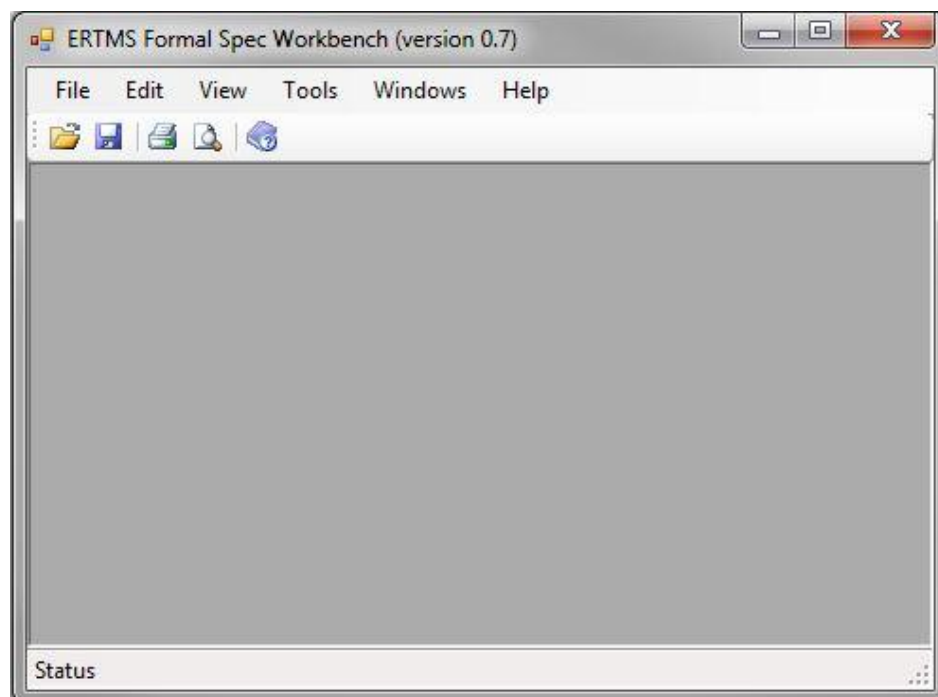


Figure 1 – Initial EFSW screen display

#### 4.4.1 Opening a EFS file

An EFS file must be opened to be able to model the system. Such files are located in the following directory

`${installationDirectory}\doc\specs\`

When the EFS file is loaded, specification browsers (see Section 5), data dictionary browsers (see Section 6) and the test browser (see Section 8) can be opened using the Tools menu.

Several files can be opened at the same time. The resulting modeled EFS system is the combination of those files. This allows, for instance, to add project specific behavior to a given system.

## 5 Subset-026 Specification browser

The specification view is used to display the requirements to be modeled. It can contain requirements from UNISIG Subset-026, or project specific requirements.

### 5.1 Opening the specification browser

The specification browser can be displayed using the [Tools\Specifications\Show specification view](#) as depicted in Figure 2.

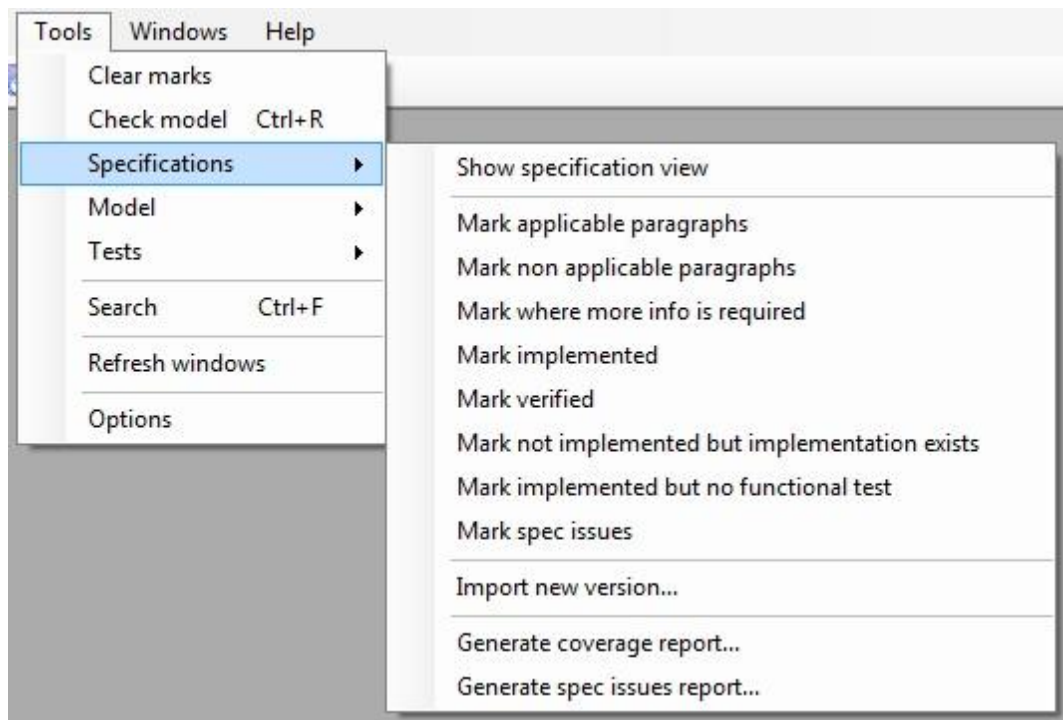


Figure 2 – Tools related to the specification view

If several EFS files have been opened, a dialog box is provided to select the EFS file on which the specification view should be opened.

### 5.2 Specification browser description

#### 5.2.1 Overview

The specification browser is decomposed into several parts, as depicted in Figure 3.

- The left tree view displays the input specifications in hierarchical order. It faithfully mirrors the content of the UNISIG Subset-026.
- The upper middle part, a property view, allows to visualize details of the element selected in the tree view. These details are:
  - The **name**, as displayed in the tree view
  - The **type** of element. These types can be either
    - **Title**: corresponds to a title in the text specification
    - **Definition**: corresponds to a definition in the textual representation
    - **Note**: corresponds to an explicative note
    - **Deleted**: indicates that the paragraph has been deleted in the requirement document, and does no more require any model



- **Requirement:** corresponds to a requirement that must be satisfied by the modeled system
  - **Table header:** corresponds to a header of a table
  - **Problem:** obsolete
- The **scope** of the requirement
  - **Track:** the paragraph only provides requirements for the trackside
  - **On Board Unit:** the paragraph only provides requirements for the on board unit
  - **On Board Unit and Track:** the paragraph provides requirements for both trackside and on board unit
- A **reviewed** indication: requirement as provided in Subset-026 may not be directly fit to the modeling. For instance, they need be split into several sub requirements. The flag indicates that the requirement has been reviewed by the requirement analyst and is ready for modeling.
- A **more info required** indication: the implementation of certain requirements may require some additional information to be completed. In this case they are marked with the flag “More info required”.
- A **spec issue** indication: the requirement is incomplete or contains an error
- The **implementation status** indicates in which state this requirement is
- **N/A:** no information
- **Implemented:** the requirement has been implemented
- **Not implementable:** the requirement cannot be implemented, because it is a title, a definition or it has been deleted.
- **New revision available:** the requirement changed after the import of a new version of the specification; its implementation needs to be reviewed
- The **unique identifier** which univocally identifies the model element in the model
- The upper right part, a messages view, can be used to visualize messages provided by the Workbench on the selected element (see Section 6.9 and 8.5).
- The lower right part of the window is decomposed into several tabs, which are active depending on the selected element
  - **Specification:** provides the text associated to the currently selected paragraph.
  - **Comment:** provides meta-information related to the selected item.
  - **Implementation:** lists the set of model elements implementing the current requirement.

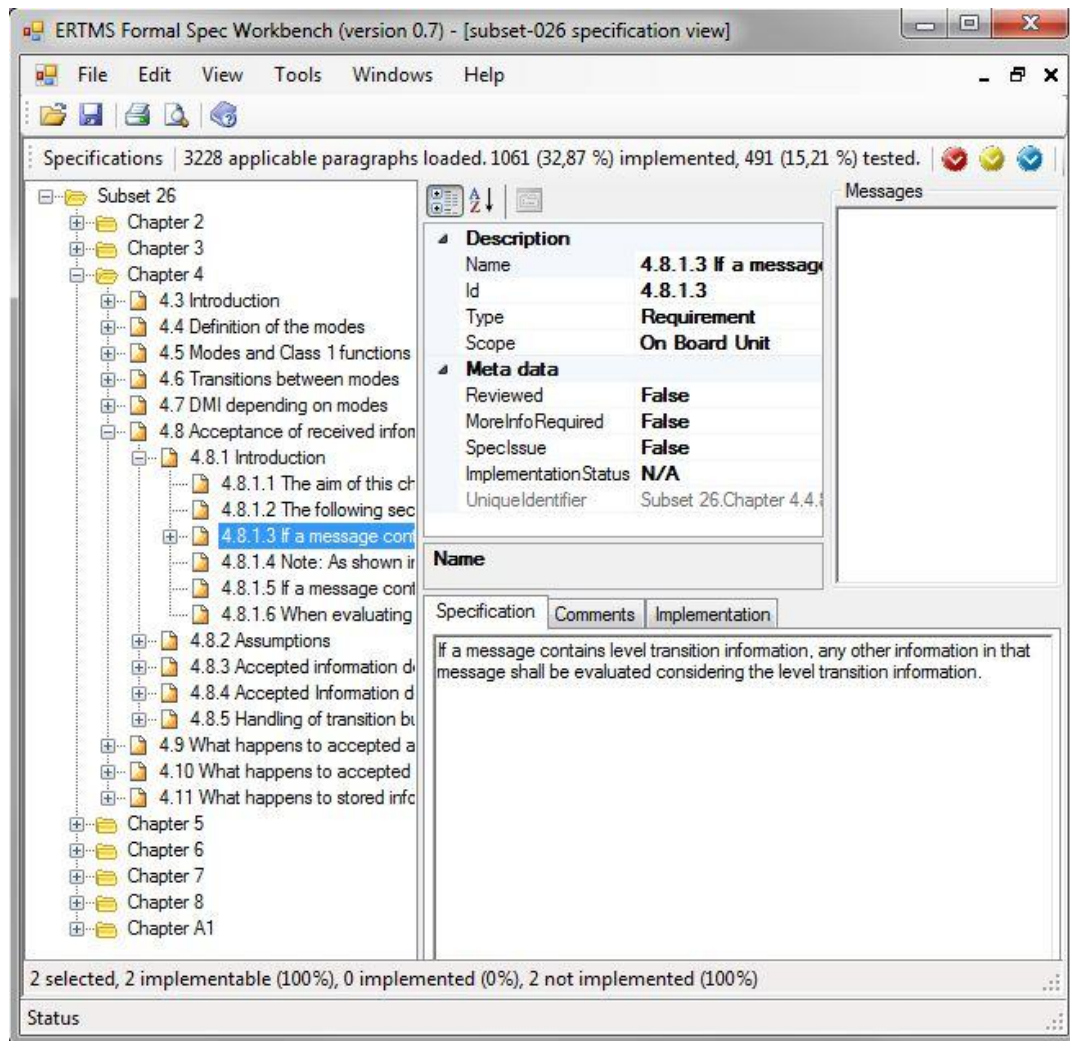


Figure 3 – Specifications browser

### 5.2.2 Requirements identification

The identifiers of the requirements, as depicted in Figure 4, correspond to the identifier used inside the Subset-026, with the following extensions:

- Requirements containing one table are further divided in sub-requirements, by adding an extra number to the identifier for each row of the table. (For instance, Chapter 4, paragraph 4.7.2 “DMI versus Mode table”) and adding before the requirement number the keyword
  - Table for the table header and
  - Entry for each table line.
- When a requirement is further split into several requirements, a sub number is added to the requirement number

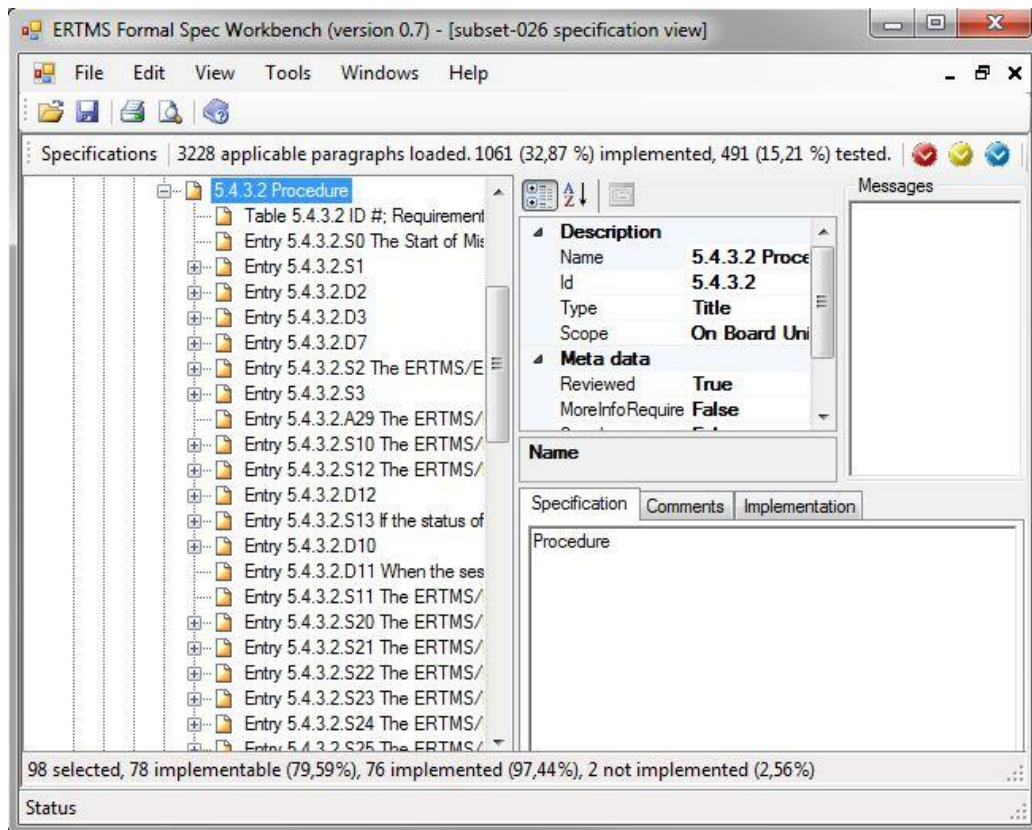


Figure 4 – Handling of tabular requirements in specifications browser

### 5.2.3 Tools related to the specification view

The Figure 2 shows the actions which can be executed on the specification.

#### 5.2.3.1 Search for (non) applicable paragraphs

Not all requirements in Subset-026 require an implementation. The EFS Workbench can be used to find the requirements that require an implementation, using the [Mark applicable](#) tool. This marks all requirements which require to be modeled in the system. A requirement requires a model when its type is Requirement.

In the same way, the [Mark non applicable](#) tool allows to mark all paragraphs which do not require an implementation.

#### 5.2.3.2 Search for paragraphs requiring additional information

The paragraphs requiring more information to be implemented can be found using the [Mark where more info is needed](#) tool. This marks all requirements which require some additional information to be modeled in the system.

#### 5.2.3.3 Search for unimplemented requirements

Un-implemented requirements can be found using the [Mark implemented](#) tool. This marks all requirements whose implementation is not complete. The implementation of a requirement is complete when

- The requirement status is Implemented
- All model elements associated to this requirement are also marked as Implemented.

#### 5.2.3.4 Search for requirements that have not been verified

Requirements that have not successfully passed the review process can be found using the [Mark verified](#) tool. This marks all requirements which have not been verified, that is, an implemented requirement for which at least one model is not marked as verified.

#### 5.2.3.5 Search for requirements that have not been verified

The [Mark not implemented but implementation exists](#) tool allows to show all requirements for which an implementation is available, but their status is still not implemented.

#### 5.2.3.6 Search for implemented requirements without functional tests

The requirements that are implemented but not yet covered by a functional test can be found using the [Mark implemented but no functional test](#) tool. This marks all requirements that are marked as implemented but are not yet covered by functional tests.

#### 5.2.3.7 Search for implemented requirements without functional tests

If a requirement contains an error, or is incomplete, it is marked as spec issue. The [Mark spec issues](#) tool allows to mark all the spec issues that have already been detected.

### 5.3 Process related to requirements

Requirements exported from Subset-026 are manually exported in XML format and imported in the Workbench using the import requirement tool. This procedure can be launched using the [Import new version](#) tool as depicted in Figure 2. If another version of the Subset-026 was already imported, it will be overridden by the new one.

#### 5.3.1 Review the requirements

Requirements must be reviewed for several reasons

- The conversion process between Subset-026 and XML file is manual, hence error prone.
- Review the type and the scope associated to the requirement.
- Requirements expressed in Subset-026 are not directly ready for modeling: some requirements are too small and must be merged with their neighbors; some others are too big and should be split.

As soon as requirements have been reviewed, the reviewer changes the Reviewed status of the requirement as **True**, as depicted in Figure 5.

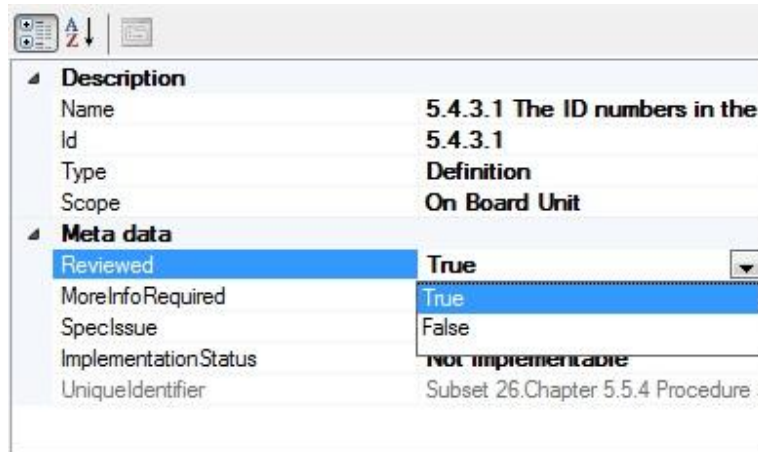


Figure 5 – Requirement has been reviewed

The tool presented in Section 5.2.3.4 allows to easily determine the requirements that still require to be reviewed.

### 5.3.2 Create the model for the requirement

Once the requirement has been modeled, the engineer changes the Implementation status of the requirement as **Implemented**. The tool presented in Section 5.2.3.2 allows to determine the requirements that still requires some modeling.

### 5.3.3 Update the model of an Implemented requirement

As soon as a model related to a requirement changes, the implementation status of the requirement automatically switches back from **Implemented** to **N/A**.

## 6 Data dictionary browser

The data dictionary browser is used to model the requirement as proposed in the specification window. This window can be used to model several items

- **Types** to be used by other model elements
- **Variables & procedures** which provide the system's state
- **Functions** which factorize common computations on the system
- **Rules** which provide the system dynamics

### 6.1 Opening the data dictionary browser

The data dictionary browser can be displayed using the [Tools\Model\Show model view](#) as depicted in Figure 6.

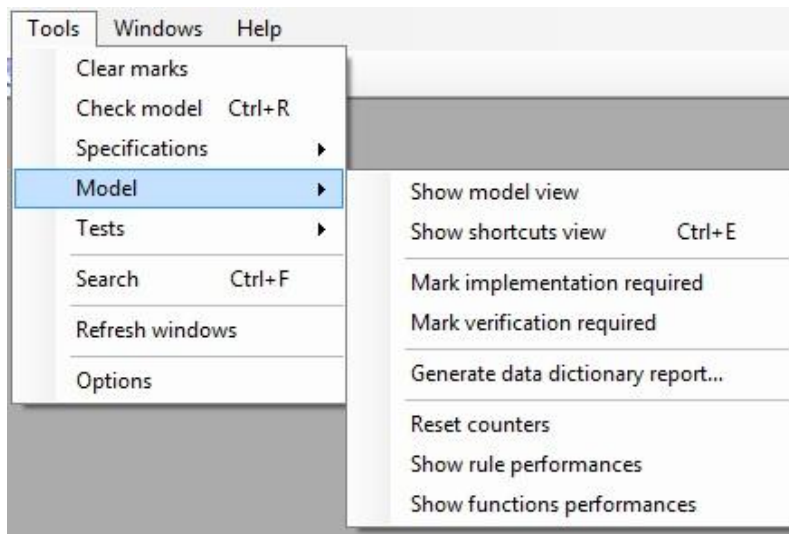


Figure 6 – Tools related to the data dictionary view

If several EFS files have been opened, a dialog box is provided to select the EFS file on which the data dictionary browser should be opened.

### 6.2 Data dictionary browser description

#### 6.2.1 Overview

The data dictionary browser is decomposed into several parts, as shown in Figure 7.

- The left tree view allows to select an element in the model.
- The upper middle part, a property view, allows to visualize details of the element selected in the tree view.
- The upper right part, a messages view, can be used to visualize messages provided by the Workbench on the selected element (see Section 6.9 and 8.5).
- The lower right part of the window is decomposed into several tabs, which are active depending on the selected element
  - **Description:** active only for elements related to a requirement. This view provides a textual description of the requirements related to the item (left part) and a pseudo-code describing the how the corresponding requirement has been implemented by the selected item (right part).



- **Expression:** displays the information associated to the current element, depending on its type:
  - for variables and types, their value
  - for function cases, pre-conditions, actions or expectations, their expression
- **Comment:** provides meta-information related to the selected item.
- **Usage:** the set of elements that are using the selected element. This usage view is available for types (in that case, it provides the variables that are instances of the corresponding type) and variables (in that case, it provides the rules that either read or write the content of the variable).

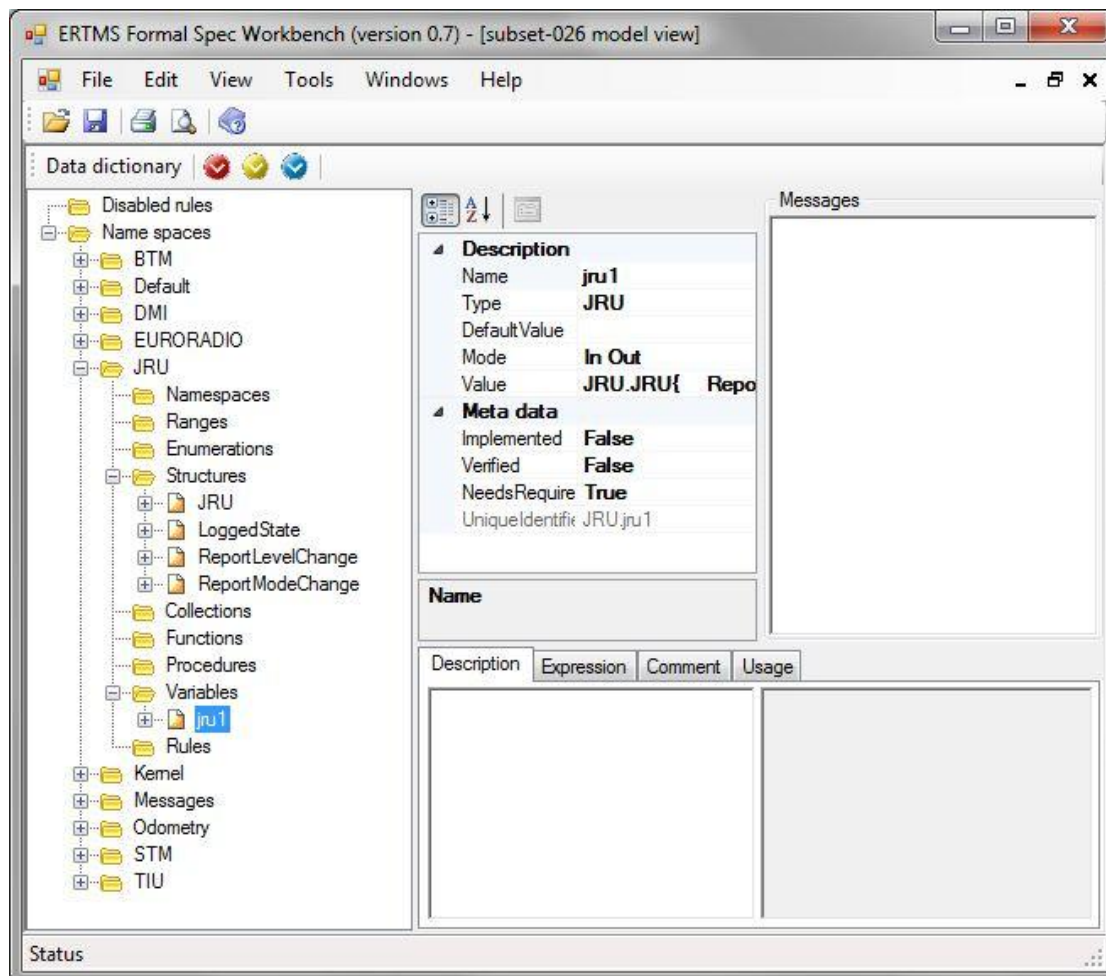


Figure 7 – Data Dictionary and name spaces

## 6.2.2 Tools related to the data dictionary view

The Figure 6 shows the actions which can be executed on the model.

### 6.2.2.1 Search for elements requiring an implementation

The [Mark implementation required](#) tool allows to display all the model elements that have not yet been marked as implemented.

#### 6.2.2.2 Search for elements requiring a verification

The [Mark verification required](#) tool allows to display all the model elements which implementation has not yet been verified.

#### 6.2.3 Shortcuts view

EFSW provides a way to quickly access some model elements, without searching it in the Data Dictionary view. These elements, named shortcuts, are situated in the shortcuts view, depicted in the Figure 8. The shortcuts can be created by drag and dropping model elements; they can then be assembled in different folders of the shortcut view. The folders can be created, deleted and renamed using the contextual menu actions.

To select in the model view the model element corresponding to a shortcut, just double-click on the latter.

The shortcut view is available using the [Show shortcuts view](#) tool, as depicted in Figure 6.

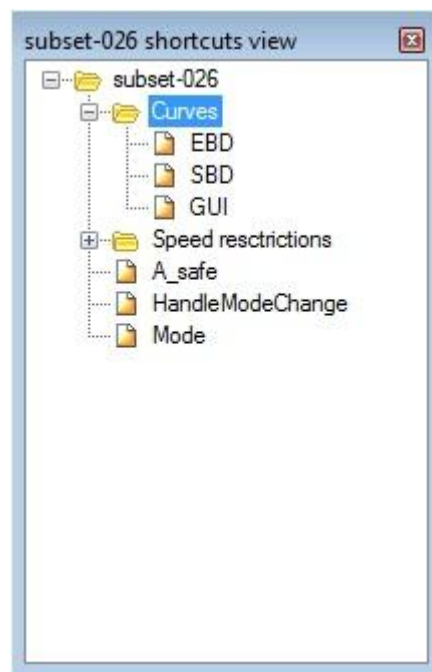


Figure 8 – Shortcuts view

### 6.3 Name spaces

Elements of the data dictionary are structured using Name spaces to group them together. Each namespace allows to define

- sub namespaces
- various data types necessary to support the model (ranges, enumerations, structures, collections and procedure sets). See Section 6.4.
- procedures and variables which model the system state. See Sections 6.5 and 6.6.
- functions which factorize common computations. See Section 6.7.
- rules which specify the system dynamics. See Section 6.8.



For instance Figure 7 displays the following name spaces: *BTM*, *Default*, *DMI*, *EURORADIO*, *JRU*, *Kernel*, *Messages*, *Odometry*, *STM* and *TIU*. It also displays the types and variables available for “JRU”. The variable *jrul* is selected and its details are available on the right part of the window.

## 6.4 Data types

The data types that can be modeled by the EFS Workbench are the followings

- **Ranges:** specify a numeric value with a low bound and a high bound. It is characterized by
  - A **name**, which identifies this type in the system.
  - A **precision** (Integer or Floating point).
  - A **default value**.
  - A **min value**.
  - A **max value**.
  - **Special values:** a set of values that have a particular meaning (for example, infinity, NA, ...).
- **Enumerations:** specify a set of discrete values identified by a name. For instance the Data State enumeration can be of the following values:
  - **DataState.Invalid**
  - **DataState.Unknown**
  - **DataState.Valid**

An enumeration is characterized by

  - A **name**, which identifies this type in the system.
  - A **default value**.

An enumeration can contain sub-enumerations, representing subsets of the enumeration’s values. For instance the Mode enumeration (representing the modes of a train) has a sub-enumeration “Mode.MA”, which contains all the possible modes that can be received in a Movement Authority.
- **Structures:** match the C-like struct. They are characterized by
  - A **name**, which identifies this type in the system.
  - A set of **sub elements** related to the structure.
  - A set of **rules** which provide the dynamic behavior of the structure’s variables.
  - A set of **procedures** which are available on all instances of that structure.
- **Collections:** Allow listing several values. Collections are typed: all elements of the same collection must be of the same type. They are characterized by
  - A **name**, which identifies this type in the system.
  - A **max size**, which indicates the maximum number of elements can be contained in the collection.
  - A **type**, which identifies the type of the collection.
  - A **default value**.

Additional information about data types is also available

- The **related requirements**, presented as sub-nodes of the data type on the left part of the window. They indicate the requirements that are modeled by the corresponding data type. One can add a new requirement to this list by drag & drop between the requirement expressed in the specification view and the data type requirement node.
- **Implemented**: indicates that the implementation of the data type is complete and can be reviewed
- **Verified**: indicates that the review has been performed on the data type and it corresponds to the related requirements. Any modification of the corresponding type removes the verified status, this indicates that a new review must be performed
- **NeedsRequirement**: indicates that a requirement is expected to be linked to this element.

Data type manipulations (add/remove) are performed in the workbench thanks to the contextual menu actions. Properties about types are altered using the property view on the right part of the Data Dictionary Browser.

## 6.5 Procedures

Procedures are used to keep track of the current system state. They define a process that should be followed by the system and are characterized by a state machine which indicates in which state of the process the procedure is, and provides the transitions between each states using Rules (see Section 6.8). Each one of these procedures is characterized by

- A **name**, which identifies this procedure in the system.
- **Parameters**: the formal parameters for this procedure.
- **Rules**: the rules to be activated when the procedure is called.
- A **state machine**, which defines a set of **states** in which the procedure can be. It also specifies an **initial state**: the state in which the state machine begins its execution.

A **state machine** is characterized by

- The set of **states** of this state machine.
- The **initial state** in which this state machine starts.
- The set of **rules**.

The state diagram representation of the state machine is available through the state diagram view, as described in Section 7.

A **state** is characterized by a name which identifies this state, along with the enclosing states and the enclosing procedure, in the system. Each state can be decomposed by a new state machine.

The **related requirements**, **implemented**, **verified** and **needs requirement** information about procedures are also available and follow the description provided in Section 6.4.

## 6.6 Variables

Variables are used to keep track of the current system state. They are characterized by

- A **name**, which identifies, along with the enclosing sub system/variable, the variable in the system.
- A **type**, as defined above.
- A **default value**, which overrides the default value specified for the type (if any).
- A **mode**:
  - **Internal**: the variable is used by the EFS model only.
  - **Outgoing**: the variable is written by the EFS model and read by the outside world.
  - **Incoming**: the variable is written by the outside world and read by the EFS model.
  - **In/Out**: the variable is used by both the outside world and the EFS model.
  - **Constant**: the variable is initialized at EFS start up. Its values then never changes. It pertains to the “Data Prep”.
- A **value**, which is the current value of the variable.

The **related requirements**, **implemented**, **verified** and **needs requirement** information about variables are also available and follow the description provided in Section 6.4.

Variable manipulations (add/remove) are performed using the contextual menu. Properties about variables are altered using the property view on the right part of the Data Dictionary Browser.

### 6.6.1 Sub variables

When the variable's type is a structure, it is decomposed into several sub variables (according to the type's structure).

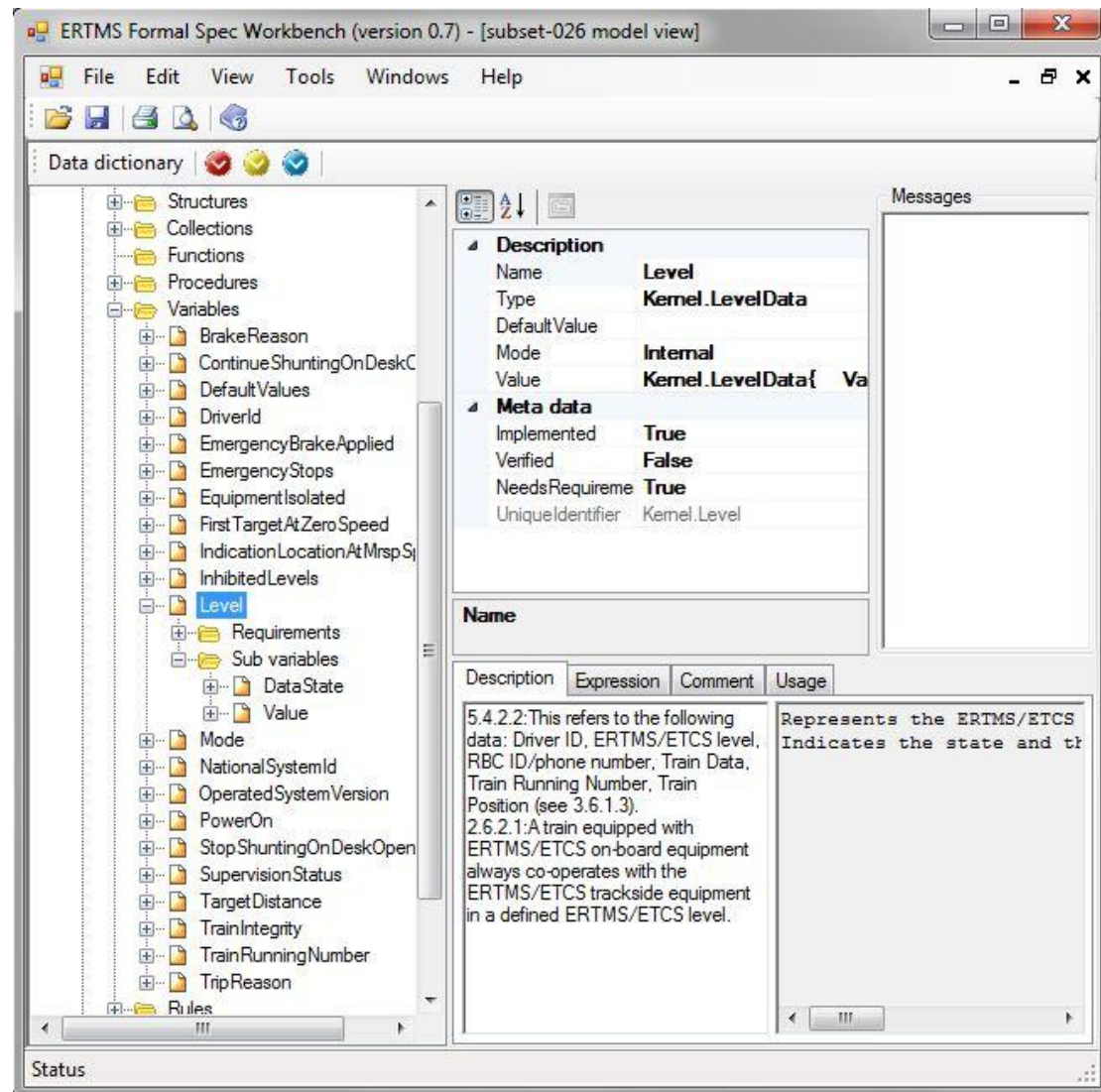


Figure 9 – Level variable & sub variables

## 6.7 Functions

Functions allow factorizing common computations in a single location. They are identified by a unique name and their return type and can have several parameters. Functions can be used to model complex functions<sup>1</sup> such as

$$f(x) = \begin{cases} \text{if } x > 0 \text{ and } x < 100 : x^2 + 1 \\ \text{if } x > 100 : 2x \end{cases}$$

<sup>1</sup> This kind of function is mainly used in braking curve computation

by defining several cases (for this example, 2 cases). Each case may have several pre-conditions. The value of the function is computed according to the expression associated to the match case. Note that pre-conditions of cases must be mutually exclusive and can only consider parameter values or constants.

An empty pre-condition is always evaluated to true.

An example of a function permitting to calculate the confidence interval of a train is depicted in Figure 10.

The **related requirements**, **implemented**, **verified** and **needs requirement** information about functions are also available and follow the description provided in Section 6.4.

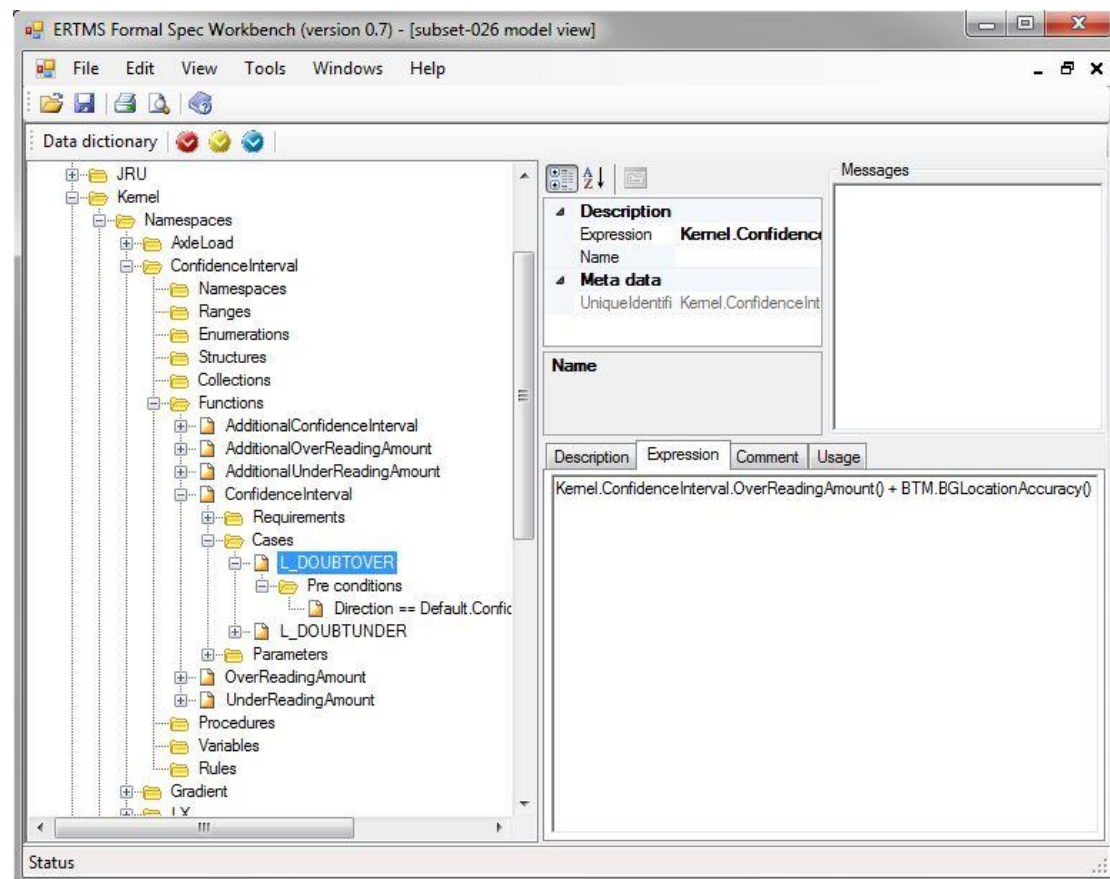


Figure 10 – Example of a function

Functions manipulations (add/remove) are performed using the contextual menu. Properties about functions are altered using the property view on the right part of the Data Dictionary Browser.

## 6.8 Rules

Rules model the behavior of the system. A rule is characterized by the following items:

- The rule **name**, which allows to identify it in the system.
- The rule **priority**. This indicates the part of the activation cycle in which the rule can be activated. The priorities are the following
  - **Input verification**: First part of the activation cycle. This phase handles verification of the IN variables.

- **Update internal** variables, this part of the cycle is performed after the input verification is complete. It updates the internal variables according to the data found in input data.
- **Processing.** This phase is performed after the update internal variable phase is complete. It is the main processing for the system.
- **Update output** variables. Based on the result of the processing phase, the output variables are updated during the update output phase.
- **Clean Up:** Cleans the contents of variables used during this cycle. This is the last step of the cycle.
- The mutually exclusive **rule conditions**. Only one of the rule's conditions can be activated when a rule is activated. When several conditions could be activated, the EFS interpreter activates the first one.

Each rule condition holds the following information

- The **pre-conditions**, specifying when the rule condition can be activated. A condition can be activated if all its pre-conditions are satisfied (see below).
- The **actions**, specifying what to do when the rule condition is activated. An action consists of modifying the state of the system.
- The **sub-rules**, which can only be activated if the enclosing rule condition is activated<sup>2</sup>.
- The **requirements** implemented by that rule.

The **related requirements**, **implemented**, **verified** and **needs requirement** information about rules are also available and follow the description provided in Section 6.4.

Rules manipulations (add/remove) are performed using the contextual menu. Properties about rules are altered using the property view on the right part of the Data Dictionary Browser.

---

<sup>2</sup> In other words, a sub-rule is active when its pre-conditions and all the pre-conditions of all its parent rule conditions are satisfied.

### 6.8.1 Specific case: a rule declared in a state

When a rule is declared in a state, it is only triggered when (as usual) its pre-conditions are satisfied but also when the current state of the state machine corresponds to the state in which the rule is declared. For instance, the Figure 11 displays the sub rule related to the state StartOfMission.A40.

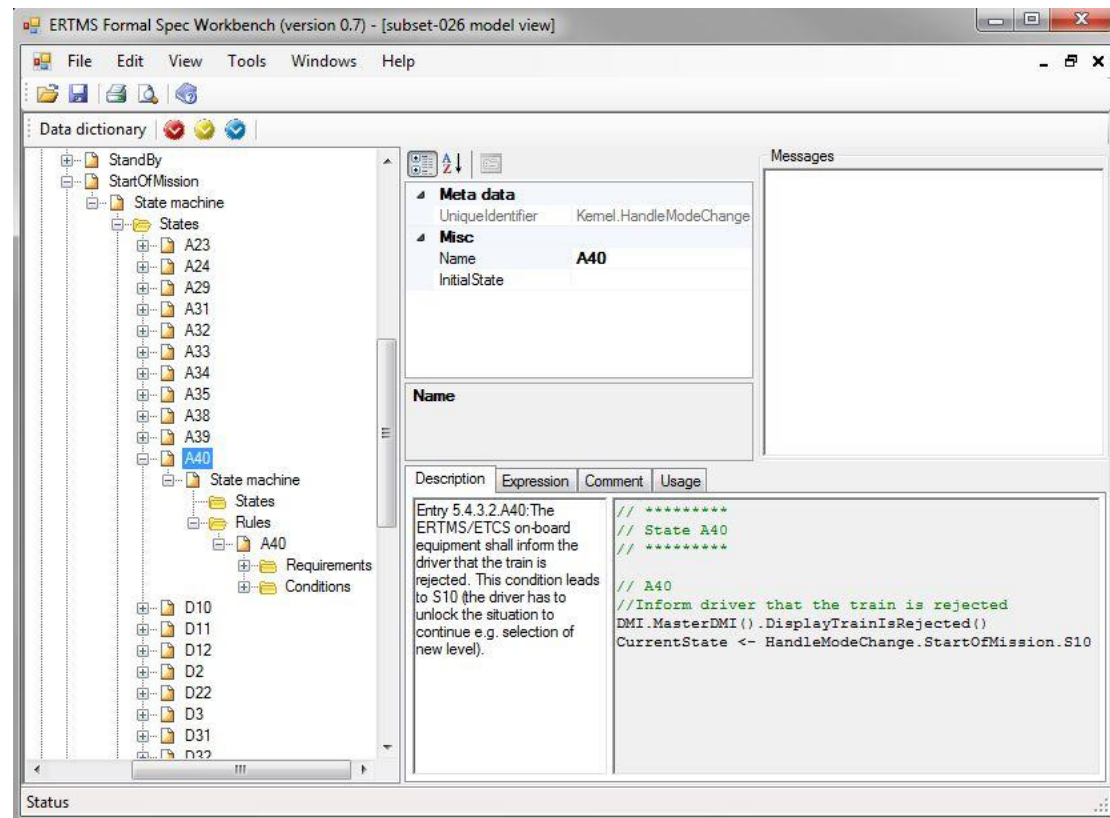


Figure 11 – Sub-rule example: Start of mission procedure, sub-rule A40

The rule A40 is triggered only when its pre-condition is satisfied (always true) and the current state of the state machine is A40 or a substate of A40). When that rule is activated, it executes two following actions

- The procedure **HandleModeChange** changes its state to the state S10.
- The procedure **DisplayTrainIsRejected** of the master DMI is called and informs the driver that the train is rejected.

Please note that this also corresponds to a transition in a state diagram, as it is represented as such in the corresponding state diagram (see Section 7).

### 6.8.2 Pre-conditions

Each rule has a list of one or more pre-conditions that must all be true for a rule to be triggered. Each pre-condition is defined by its name, containing the plain text of the expression associated to this pre-condition.

### 6.8.3 Actions

Each rule contains a set of actions which are executed when the rule is activated. An action is defined by its name, containing the plain text of the expression associated to this action. The action can be a variable update or a procedure call.

## 6.9 Model validations

The model validation engine can be activated by selecting the menu item [Tools/Check model](#) which performs the following checks:

Model	Description	Severity																																									
Expression	Expressions must be syntactically correct	Error																																									
Expression	Expressions must be type correct	Error																																									
Expression	Designators used in expressions must refer to a valid model element	Error																																									
Expression	Designators used in expressions must have a valid type	Error																																									
Typed Element	The declaration of a typed element (variable, procedure, structure element, function) defined in the model must have a valid type	Error																																									
Typed Element	Recursive types are not allowed	Error																																									
Field of a structure	<div>The mode of a field of a structure must be at least more restrictive than the mode of its enclosing field according to the following table</div> <table><tr><th rowspan="2">Enclosing field</th><th colspan="5">Field</th></tr><tr><th>Constant</th><th>Incoming</th><th>In/Out</th><th>Internal</th><th>Outgoing</th></tr><tr><td>Constant</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Incoming</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>In/Out</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Internal</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Outgoing</td><td></td><td></td><td></td><td></td><td></td></tr></table>	Enclosing field	Field					Constant	Incoming	In/Out	Internal	Outgoing	Constant						Incoming						In/Out						Internal						Outgoing						Warning
Enclosing field	Field																																										
	Constant	Incoming	In/Out	Internal	Outgoing																																						
Constant																																											
Incoming																																											
In/Out																																											
Internal																																											
Outgoing																																											
State machine	The initial state of a state machine is not empty and corresponds to a state of that state machine	Error																																									



State machine	The name of states in a state machine are valid (e.g. do not contain space)	Error
Type	Types are uniquely identified	Error
Type	A type should define a valid default value	Error
Variable	The variable semantics cannot be empty. Fill the field comment defined for the variable	Info
Requirement related	When the implementation of a requirement is completed, all model element which implement that requirement must also be marked as “implementation completed”.	Info
Requirement related	A model element related to a requirement (type, variable, procedure, function, rule) should refer to at least one requirement. This is only true if the flag NeedsRequirement is set to true.  Either set the flag NeedsRequirement to false or provide a link to a requirement.	Info
Requirement link	The link to a requirement must refer to a valid requirement	Error
Rule	If one of the pre-condition is in the form  <i>Variable == 'Request.Response'</i>  there must be an action which sets that variable to <i>'Request.Disabled'</i> . This ensures that all requests for which a response was received are disabled.	Error
Rule	An incoming variable cannot be modified by the EFS.	Error
Rule	An outgoing variable cannot be read by the EFS.	Error
PreCondition	The variable on which the pre-condition is computed must exist in the system	Error
PreCondition	The variable type must match the Operand type, according to the operator semantics. The resulting value must be a Boolean.	Error
PreCondition	Operator == should not be used for state comparison	Warning
PreCondition	Operator != should not be used for state comparison	Warning

Action	The variable which is modified by the action must exist in the system	Error
Action	The variable type must match the expression type	Error
Expectation	The variable which is checked by the expectation must exist in the system	Error
Expectation	The variable which is checked by an expectation must match the expression type	Error

After the validation is performed, each node of the data dictionary view is displayed according to the following rule in the following color:

- **Red**: an error has been found on the corresponding item.
- **Orange**: an error has been found in one of the sub elements.
- **Brown**: a warning (and no error) has been found on the item.
- **Yellow**: a warning (and no error) has been found in one of the sub elements.
- **Blue**: an informative message (and no error nor warning) has been found on the item.
- **Light blue**: an informative message (and no error nor warning) has been found in one of the sub elements.
- **Black**: nothing has been found either on this item or one of its sub elements.

Figure 12 shows an example of the result of the Check model action. In this case, there is an error on one of the actions of the rule Update LRBG Values, and info messages on several items of the tree view.

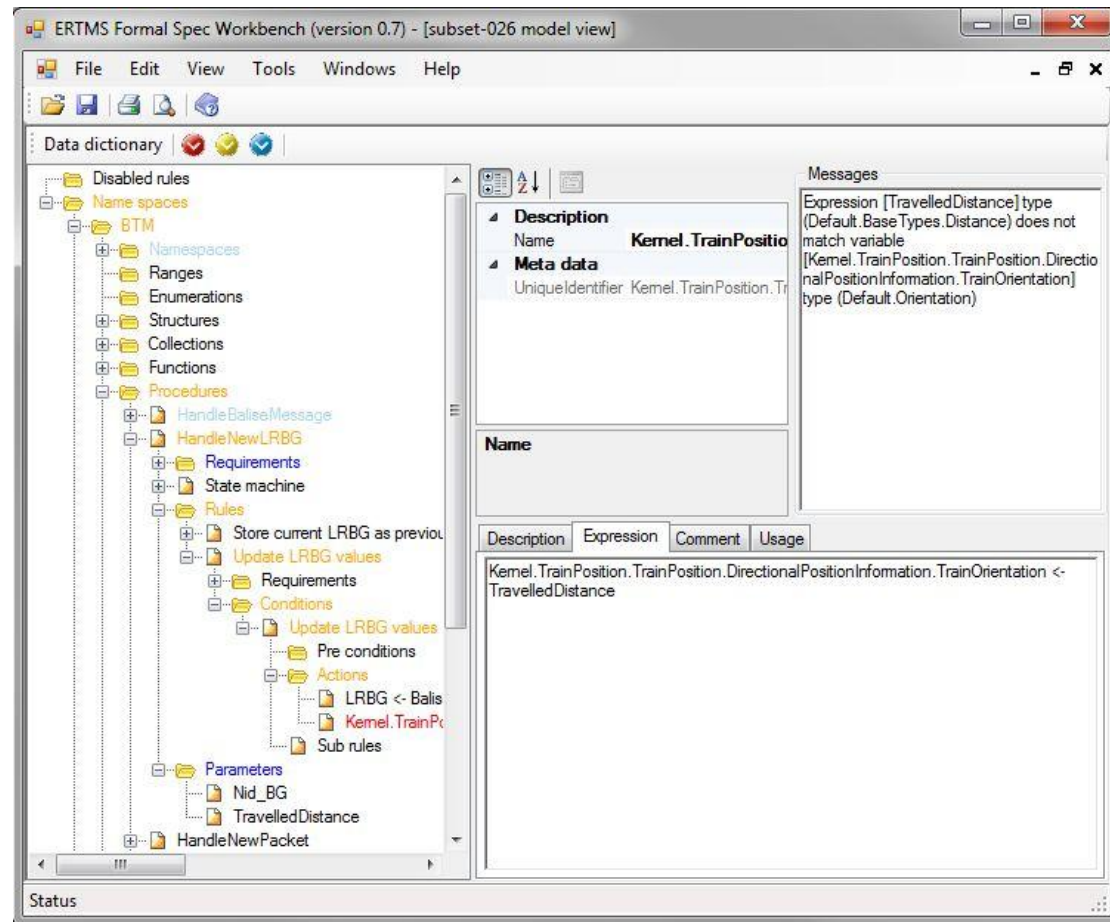





Figure 12 – Example of model warnings/errors

The buttons ,  and  can be used to navigate to respectively the next **error**, **warning** or **information** message, according to the selected node in the tree.

## 6.10 Expressions

Expressions are used by the EFSWorkbench to evaluate values for pre-conditions (see Section 6.8.2), actions (see Section 6.8.3) and expectations (see Section 8.3.2). They follow the BFN grammar described in the EFSW Technical Guide (see [1]).

## 7 State diagrams

State machines described in the system can be visualized using state diagram view.

### 7.1 Display a state diagram

To display a state machine, select the corresponding procedure or state machine in the data dictionary window, in the contextual menu, select [View state diagram](#).

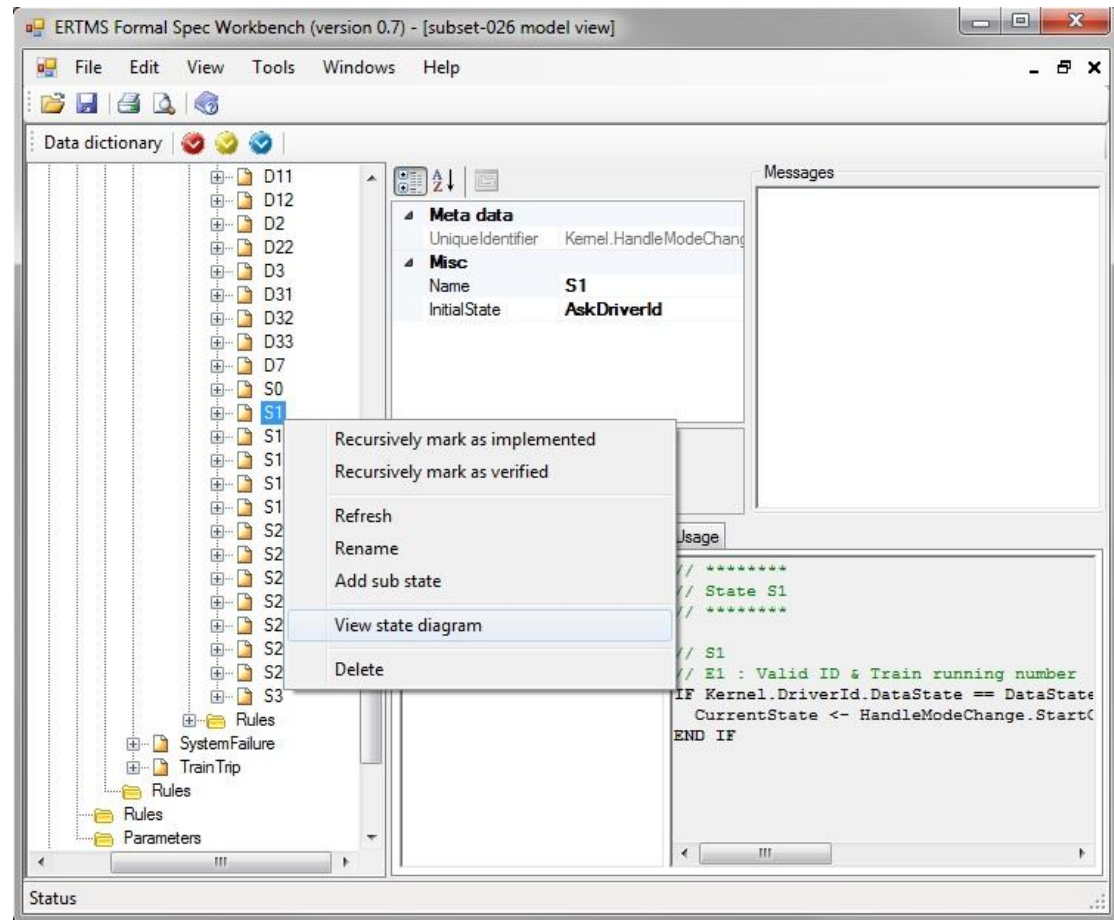


Figure 13 – View the state diagram associated to a procedure

Based on the state machine information and on the rules of the model, the Workbench builds the state diagram representation of the state machine and displays it in a new window, as depicted in Figure 14.

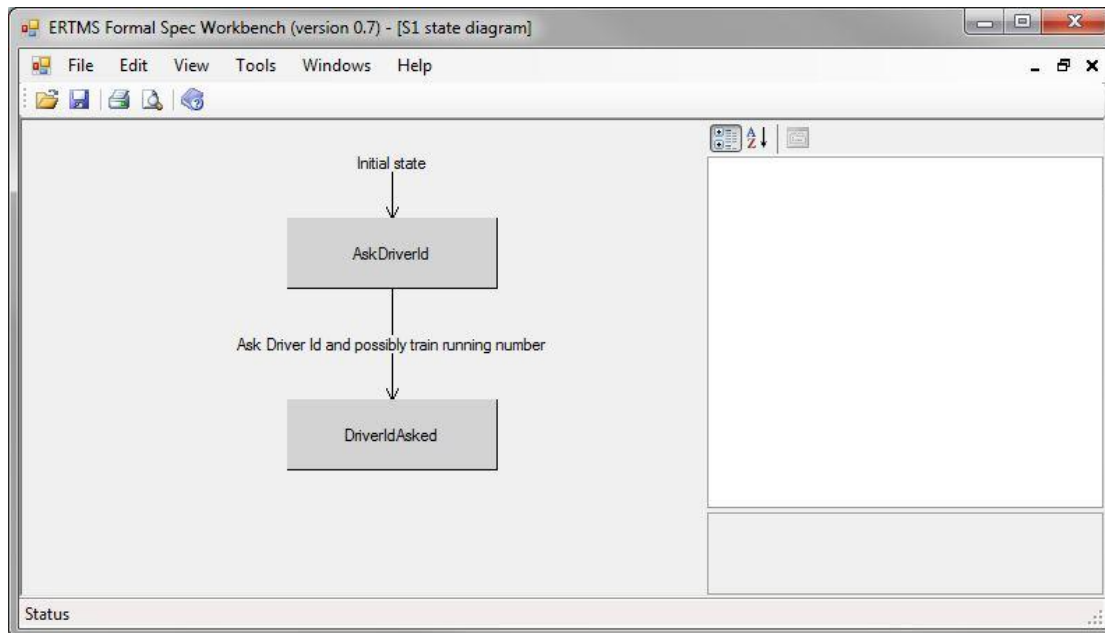


Figure 14 – State diagram view

## 7.2 Manipulations of a state diagram

New states and new transitions can be added in the state diagram using the graphical view, using the contextual view.

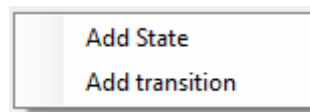


Figure 15 – Contextual menu for a state diagram

### 7.2.1 Add a new state

To add a new state, select [Add state](#) in the state diagram window contextual menu. This creates a new state in the corresponding state machine and updates the state diagram accordingly.

### 7.2.2 Add a new transition

A new transition is added by selecting [Add transition](#) in the same contextual window. This creates a new rule associated to this state machine whose pre-condition checks that the current state machine is in the source state of the transition, and adds an action which sets the current state machine state to the target state of the transition. Details about the transition (such as additional pre-conditions) can be accessed through the data dictionary view.

### 7.2.3 Selecting element in a state diagram

When a state or a transition is selected, its details are displayed in the right side of the window and the corresponding element is selected in the Data Dictionary window.

## 8 Test browser and execution environment

The EFSW provides an environment to define and run several kinds of tests:

- UNISIG Subset-076 test sequences
- Additional functional tests targeted at EFS model coverage
- User-defined tests

### 8.1 Opening the test browser

The test browser can be displayed using the [Tools\Tests\Show tests view](#) as depicted in Figure 16.

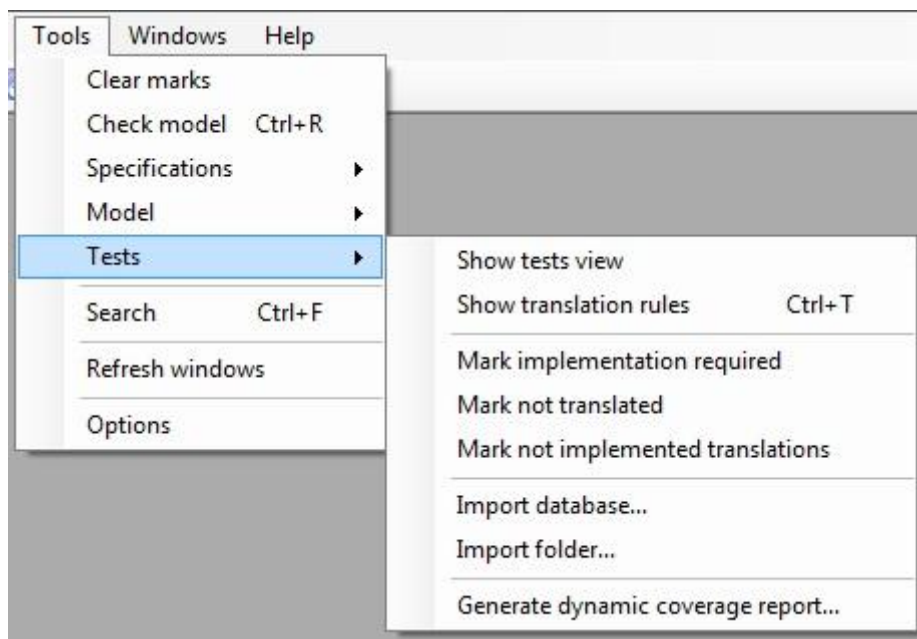


Figure 16 – Tools related to the test view

### 8.2 Test browser description

#### 8.2.1 Overview

The test browser is decomposed into several parts, as shown in Figure 17.

- The left tree view represents the tests structure and allows to select a test element. Tests are structured in the following way:
  - at the top level, tests are grouped into **test frames**
  - each frame is decomposed into **sub sequences**
  - each sub sequence is decomposed into **test cases**
  - each test case is decomposed into several **ordered steps**
  - each step is decomposed into several ordered **sub-steps**
- The upper middle part, a property view, allows to visualize details of the element selected in the tree view.
- The upper right part, a messages view, can be used to visualize messages provided by the Workbench on the selected element (see Section 6.9 and 8.5).
- The lower right part of the window is decomposed into several tabs, which are active depending on the selected element
  - **Description:** text editor allowing to enter free text expressions.

- **Time line:** displays **expectations**, **rule activations** and **variable updates** which occurred during the test (see 8.6);
- **Comment:** provides meta-information related to the selected item.

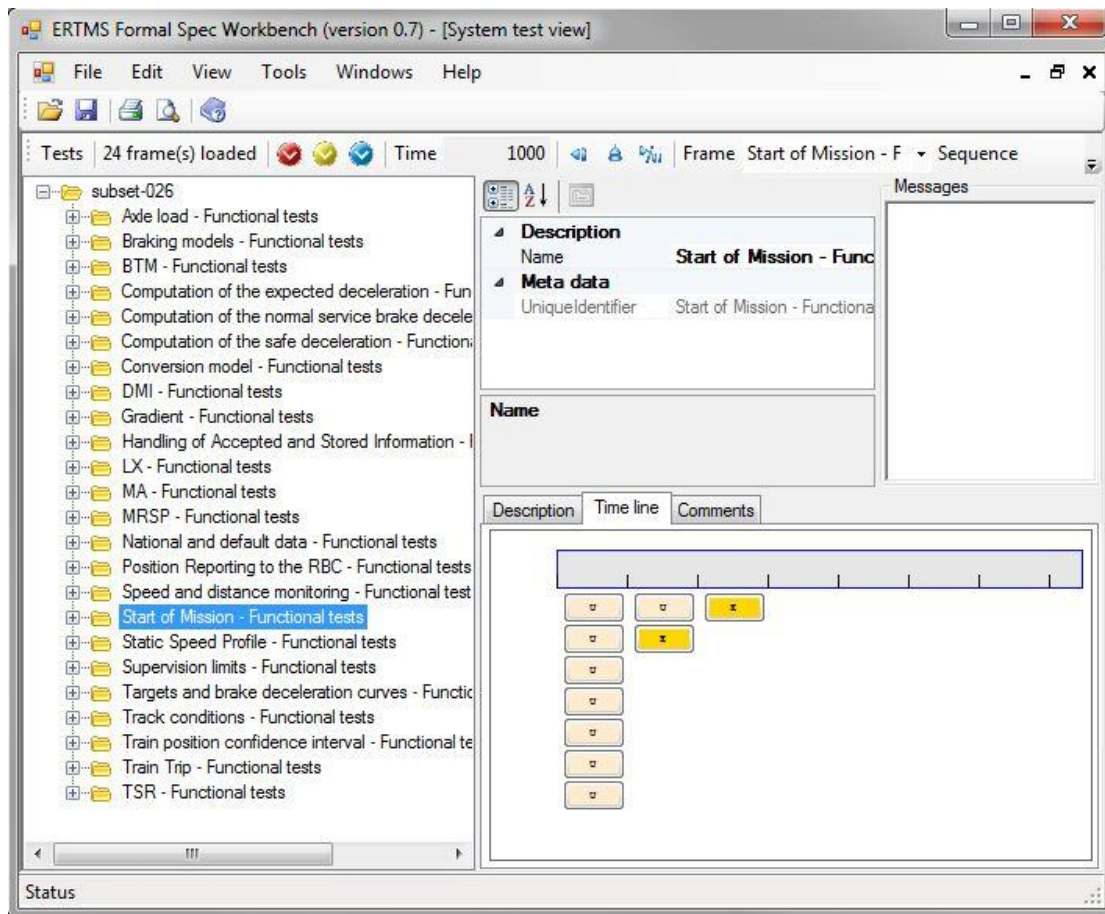


Figure 17 – Test browser details

### 8.2.2 Tools related to the test view

The Figure 16 shows the actions which can be executed on the tests structure.

#### 8.2.2.1 Search for test elements requiring an implementation

The **Mark implementation required** tool allows to display all the test elements that have not yet been marked as implemented.

### 8.2.3 Test actions

The stop button allows reinitializing the test infrastructure whereas the play button allows activating the interpretation machine once. This modifies the event displayed in the time line (see 8.6)

## 8.3 Test step

A step is composed of a list of ordered sub-steps, which contain a set of actions and a set of expectations.



### 8.3.1 Actions

Once the step is activated, it modifies the state of the system by applying all actions related to that step (see 6.8.3 for actions application). After that, the interpreter is activated until all expectations of this step are either reached or failed.

### 8.3.2 Expectation

Expectations describe constraints about the system state which must be satisfied once a step is activated. Expectations hold the following information

- **Name:** corresponds to a boolean expression which should be evaluated to **True** in order to be considered successful.
- **Blocking:** false indicates that the following sub-step can be activated, even if this expectation is not reached<sup>3</sup>.
- **Deadline:** provides the time after which the expectation should be satisfied before the expectation is considered as Failed.

## 8.4 Test step execution

Activating a test step is achieved by selecting the contextual menu **Run until expectation reached**. This activates all previous steps until their expectations are reached, including this step. Activating a step without checking that all its expectations are reached is performed using the contextual menu item **Run once**. The test execution environment inner workings are fully described in the EFSW Technical Guide, in section 5 “Interpretation Model” (See [1]). Activating a step updates the time line according to the raised events (see 8.6).

After a step has been executed, the corresponding test tree node is displayed according to the color coding convention explained in Section 6.9: failed expectations are displayed in **red** as depicted by Figure 18.

---

<sup>3</sup> For now, only blocking expectations are implemented



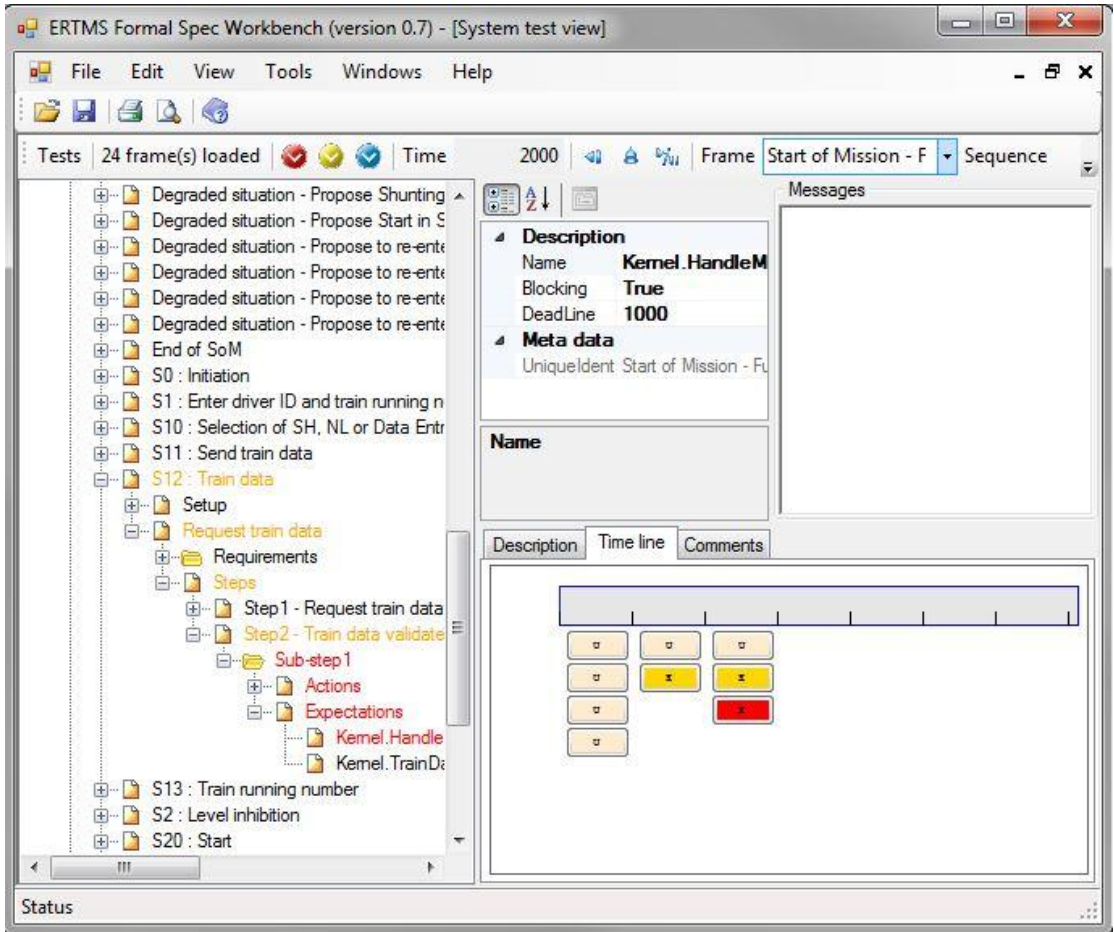


Figure 18 – Failed expectation

## 8.5 Test Case, Sub sequence and Frame execution

The contextual menu item **Execute** can be selected to

- Execute a test case enclosed in a subsequence, when applied on a test case.
- Execute all test cases of a sub sequence, when applied on a sub sequence.
- Execute all sub sequences in a test frame, when applied on a test frame.
- Execute all tests frames, when applied at the top of the tree view.

After the test has been executed, failed expectations are displayed in **red** as depicted by Figure 19, enclosing nodes are displayed in **orange**. In this case, the Time Line is not updated with the test events (see 8.6).

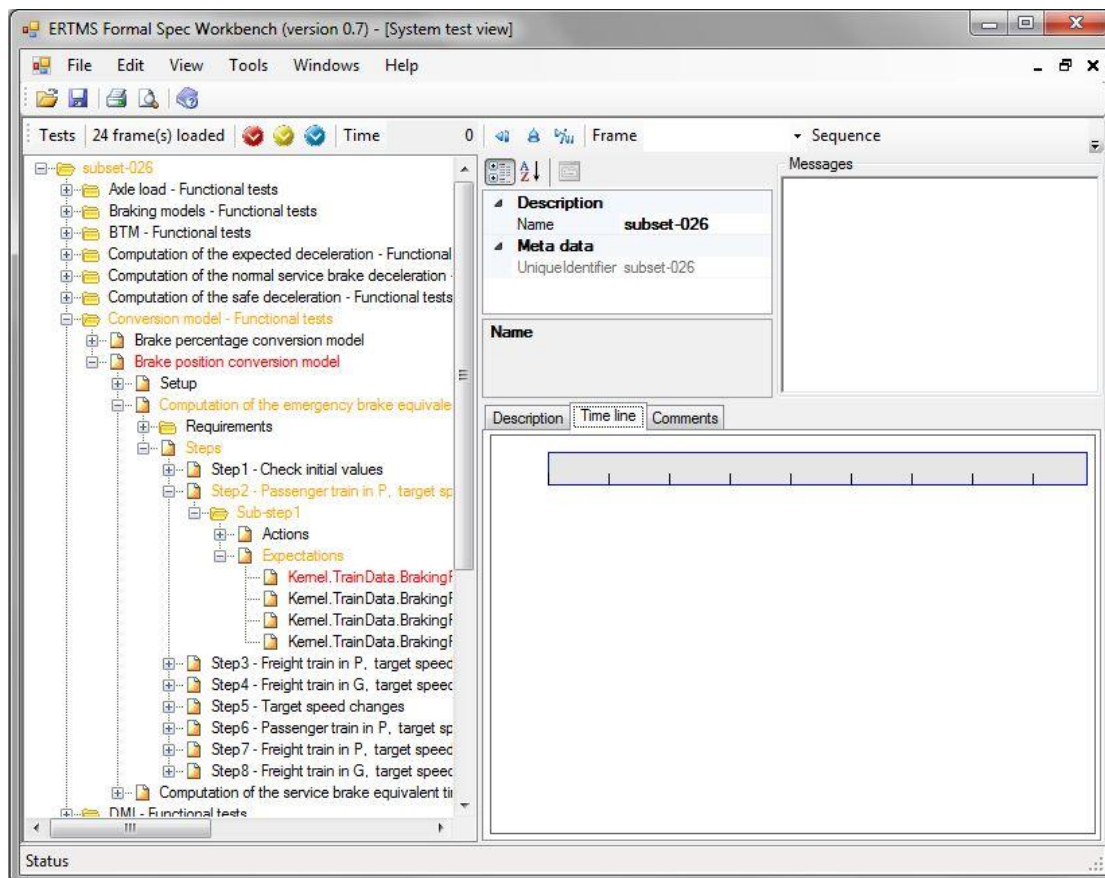


Figure 19 – Executing all the tests - failed tests are marked in orange

## 8.6 Time Line

The timeline component displays graphically events that occurred in a test:

- **Activated rules** are displayed in blue. Clicking on the blue button navigates to the rule inside the Rule Browser component.
- **Variable updates** are displayed in light rose. Clicking on the rose button navigates to the variable in the Data Dictionary component.
- **Unfulfilled expects** are displayed in grey. Clicking them navigates to the expect in the test browser.
- **Fulfilled expects** are displayed in yellow. Clicking them navigates to the expect in the test browser.
- **Failed expects** are displayed in red. Clicking them navigates to the expect in the test browser.

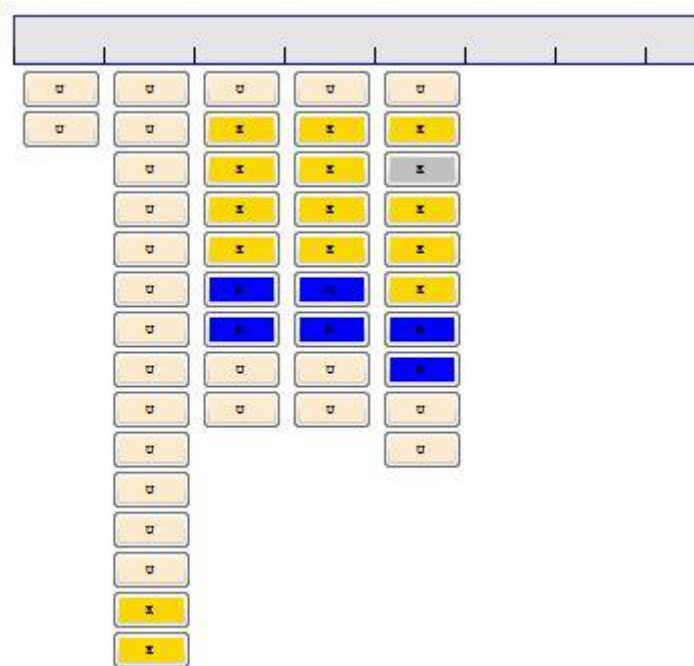


Figure 20 – Timeline component

The Workbench provides a means to filter the displayed elements of the time line, in order to limit their number. This can be done thanks to [Configure filter](#) field of the time line's contextual menu, as depicted below.

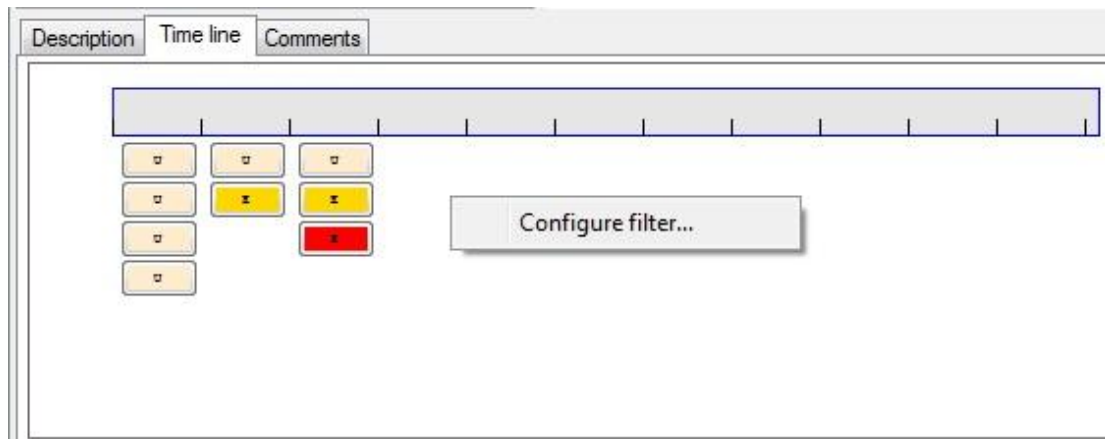


Figure 21 – Contextual menu to configure the time line filter

The Figure 22 shows the filter options. These options are:

- Show or hide **rule activations**.
- Show or hide **variable updates**.
- Show or hide **expectations**.
- **Namespace** filtering.
- **Regular expression** filtering.

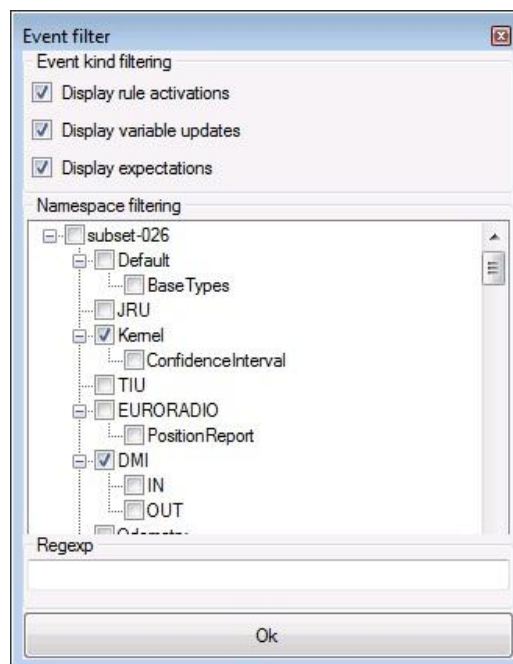


Figure 22 – Filter options

## 9 Translations

Translations can be defined to automatically translate tests steps imported from Subset-076 into actions, and expectations.

### 9.1 Open the translation view

The translation view can be opened using the tool menu entry [Tools/Tests/Show translation rules](#), as depicted in Figure 16.

If several EFS files have been opened, a dialog box is provided to select the EFS file on which the translation view should be opened.

### 9.2 Translation view description

#### 9.2.1 Overview

The translation view allows to specify pattern matching rules which translate into actions and expectations. The translation browser is decomposed into several parts, as shown in Figure 23.

- The left tree view represents the translations. A translation contains a set of source texts, which represent the text that has to be translated, and the set of actions and expectations corresponding to these texts.
- The upper middle part, a property view, allows to visualize details of the translation selected in the tree view (the implementation status and the name, corresponding to the text associated to the translation).
- The upper right part, a messages view, can be used to visualize messages provided by the Workbench on the selected element (see Section 6.9 and 8.5).
- The lower right part of the window shows the expression(s) corresponding to the selected translation.

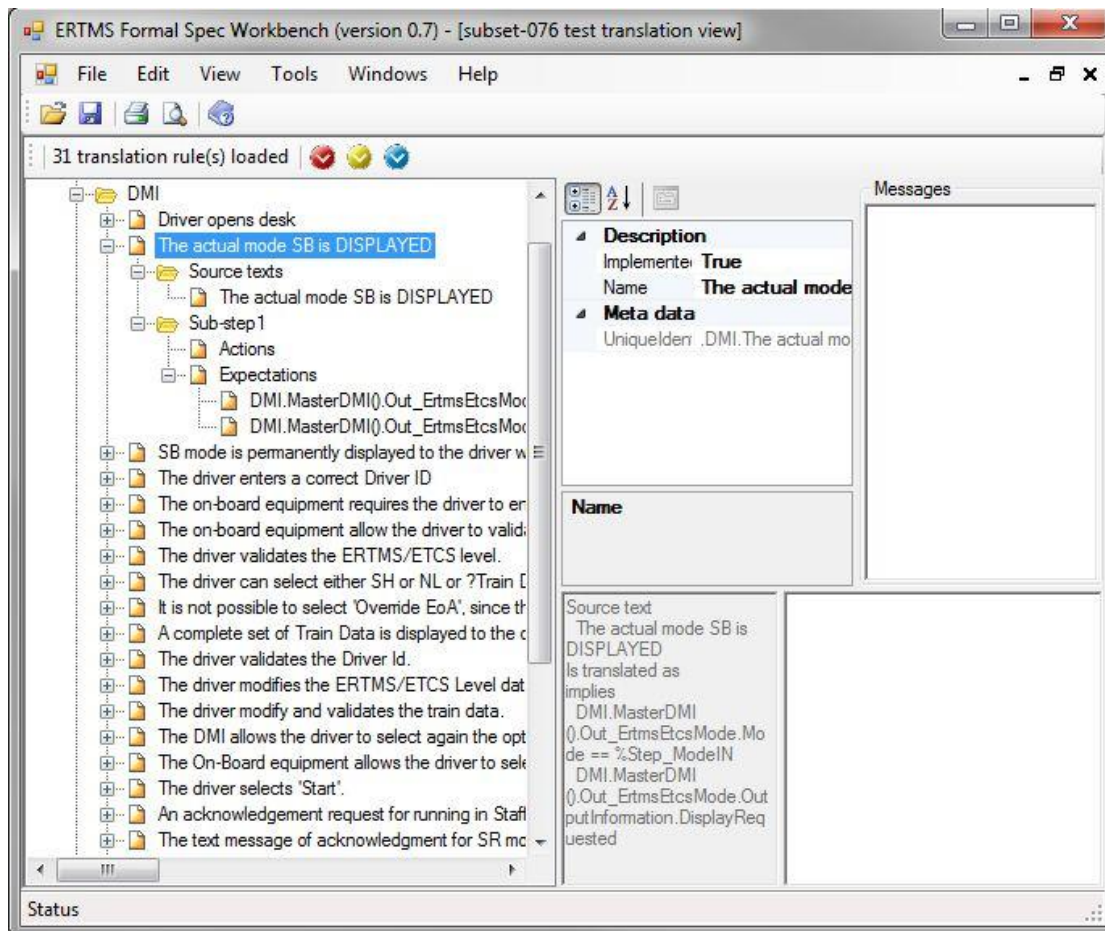


Figure 23 – Translation view

Each entry of the translation view provides the list of matching texts and the pre-conditions, actions, expectations or post actions to be placed in the step whose text matches one of the source text provided.

### 9.2.2 Tools related to the translation view

The Figure 16 shows the actions which can be executed on the tests structure.

#### 9.2.2.1 Search for not translated tests

The [Mark not translated](#) tool allows to display all the test steps which require a translation but are not yet translated.

#### 9.2.2.2 Search for not implemented translations

The [Mark not implemented translations](#) tool allows to display all the translations which have not yet been implemented.

#### 9.2.2.3 Import a database

The [Import database](#) tool allows to import a subset 76 test database as a new test sequence.

#### 9.2.2.4 Import a folder

The [Import folder](#) tool allows to import all subset 76 database test sequences located in a directory in EFS.



### 9.3 Apply translation rules

To apply translation rules, open the test browser, select a node in the tree view, and in the contextual menu, select **Apply translation rules**, as shown below.

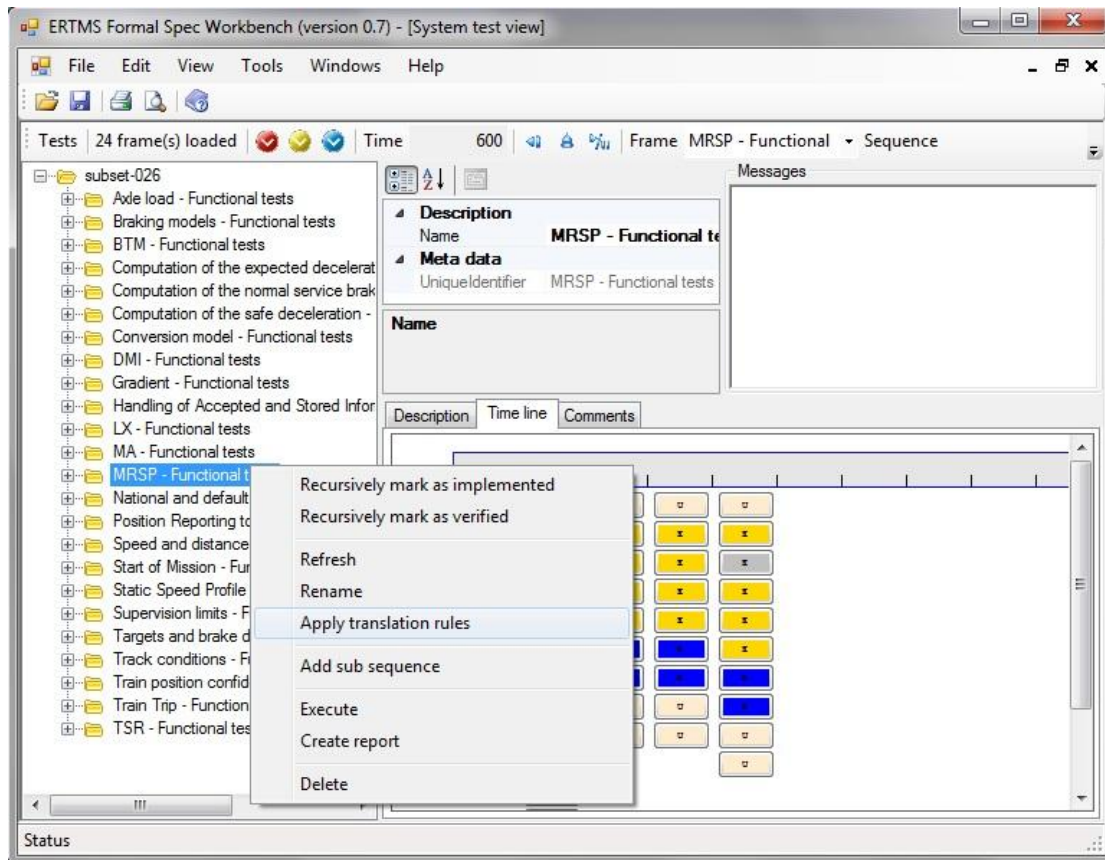


Figure 24 – Apply translation rules

This updates all the nodes enclosed by the selected node according to translation rules.

## 10 Reports

The EFSW provides the possibility to generate reports in *pdf* format which provides data about coverage of the specification by the model, as described in Section 10.1, and dynamic model coverage resulting of the test executions as described in Section 10.2.

### 10.1 Specification coverage report

#### 10.1.1 Purpose

The purpose of the specification coverage report is to provide requirement coverage information based on the modeled elements, along with implementation statistics.

The **specification coverage** part of the report provides the status of all the requirements specified in the requirement document. This status can be one of the followings:

- **Implemented:** indicating that the modeling of the requirement is complete. This also ensures that all model elements which model this requirement are also marked as implemented..
- **N/A.** indicates that the implementation status of this paragraph is not yet known. In this case, implementation is not performed.
- **Not implementable.** Indicates that the paragraph does not need to be implemented because it is not a requirement.

The **requirement coverage report** provides the list of all the requirements of the specification along with the model elements that implement them.

The **model coverage report** provides the list of requirements associated to each model element.

#### 10.1.2 Structure

The specification coverage report can be composed of four following chapters:

- **Specification coverage.** This chapter can be composed of two sections:
  - **Statistics.** This section provides a table with the following information:
    - Total number of specification paragraphs.
    - Number of applicable paragraphs. The paragraph is applicable if its scope is "OBU" (on board unit) or "OBU and Track" and if its type is "Requirement".
    - Number and percentage of covered paragraphs. This percentage is computed from the total number of applicable paragraphs. A paragraph is considered as covered if it is marked as "implemented" and all the EFS elements that are related to this paragraph are marked as "implemented".
  - **Specification.** This section provides a table with an entry for each specification paragraph. For each paragraph the table provides its scope ("OBU", "OBU and Track" or "Track"), its type and its implementation status.
- **Covered requirements.** This chapter provides the list of requirements covered by the model and can be composed of two following sections:



- **Statistics.** This section provides a table with the following information:
  - Number of applicable paragraphs.
  - Number and percentage of covered requirements.
- Covered requirements. This section provides a table with an entry for each covered requirement. For each covered requirement, the table provides the list of the model elements that implement it with the associated comment.
- **Non-covered requirements.** This chapter provides the list of requirements that are not yet covered by the model and can be composed of two following sections:
  - **Statistics.** This section provides a table with the following information:
    - **Number of applicable paragraphs.**
    - **Number and percentage of non-covered requirements.**
  - **Non-covered requirements.** This section provides the list with the non-covered requirements.
- **Model coverage.** This chapter provides the list of implemented model elements and can be composed of the following sections:
  - **Statistics.** This section provides a table with the following information:
    - **Number of implemented model elements.**
    - **Number and percentage of modeled paragraphs.**
  - **Implemented rules.** This section provides a table containing an entry for each implemented rule. For each implemented rule the table provides the (list of) the paragraph(s) it implements.
  - **Implemented types.** This section provides a table containing an entry for each implemented type. For each implemented type the table provides the (list of) the paragraph(s) it implements.
  - **Implemented variables.** This section provides a table containing an entry for each implemented variable. For each implemented variable the table provides the (list of) the paragraph(s) it implements.

### 10.1.3 Launch the specification reporting

The specification coverage report creation window is accessible via [Tools/Specifications/Generate coverage report...](#) (See Figure 25).

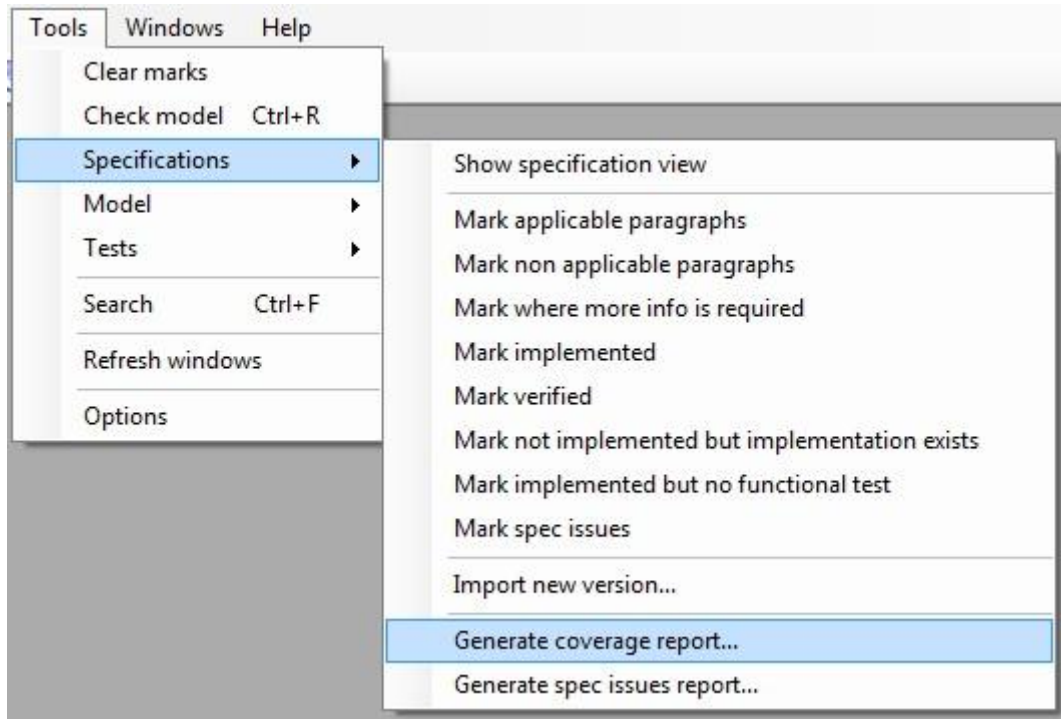


Figure 25 – Launch the specification coverage report

This opens the dialog which allows to select the report options, as depicted below

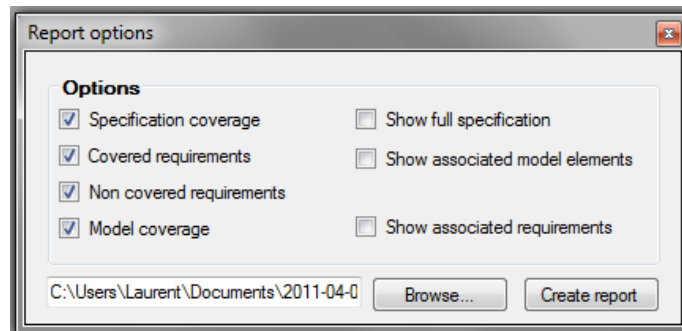


Figure 26 – Specification coverage report options

The figure above shows the window permitting to select the different report options. The check boxes in the left column allow to create the corresponding report chapters (described in Section 10.1.2) with their "Statistics" section. The check boxes of the right column allow to add additional information described below:

- **Show full specification** check box is accessible only when the "Specification coverage" check box is selected. This option allows to add the "Specification" section to "Specification coverage" chapter.
- **Show associated model elements** check box is accessible only when the "Covered requirements" check box is selected. This option allows to add the information about the elements implementing the covered requirements in "Covered requirements" section of "Covered requirements" chapter.
- **Show associated requirements** check box is accessible only when the "Model coverage" check box is selected. This option allows to add the list of requirements modelled by each model element in the sections "Implemented rules", "Implemented types" and "Implemented variables" of "Model coverage" chapter.

The "Browse" button allows to select the name and the folder of the generated report.

#### 10.1.4 Spec issues report

The [Tools/Specifications/Generate spec issues report...](#) tool allows to create a report containing:

- all the spec issues that have been detected during the implementation of the model.
- all the design choices that have been made during the implementation of the model

## 10.2 Dynamic tests coverage report

### 10.2.1 Purpose

The purposes of the dynamic tests coverage report are to provide model elements dynamic coverage by either all the implemented tests, a specific test frame, a specific sub sequence, a specific test case or a specific step of a test

The report can also be used to provide the log associated to a particular step of a test. This log information contains the sequence of operations executed by the step along with the time when these operations occurred.

### 10.2.2 Structure

The dynamic tests coverage report can provide the information at different **levels** (called filters) of the tests tree (frames, sub sequences, test cases or steps) depending on user's choice (see Section 10.2.3). The topmost level is that of frames. A level can be included in the report if it is the topmost level or if its enclosing level is included in the report.

The report can be created for the whole tests set of the model or for a certain element of the tests tree. In that case the level of a selected element is the topmost. For example, a report can concern

- The whole tests tree, containing information of all its levels
- All the available frames
- All the frames and all the sub sequences
- A particular sub sequence with all its sub cases
- A particular test case

For each item of each selected level the report provides the percentage of activated rules of the EFS model and (if selected) the list of these rules and/or the list of rules that weren't activated. This information is computed as follows:

- For a **step**, the list of activated rules is computed by executing it and collecting the activated rules of its time line (see section 8.6)
- The rules activated by a **test case** are the set of rules activated by all its steps
- The rules activated by a **sub sequence** are the set of rules activated by all its test cases
- The rules activated by a **frame** are the set of rules activated by all its sub sequences

### 10.2.3 Launch the test coverage reporting

The specification coverage report creation window is accessible via [Tools/Tests/Generate dynamic coverage report...](#) (see figure below **Erreur ! Source du renvoi introuvable.**).

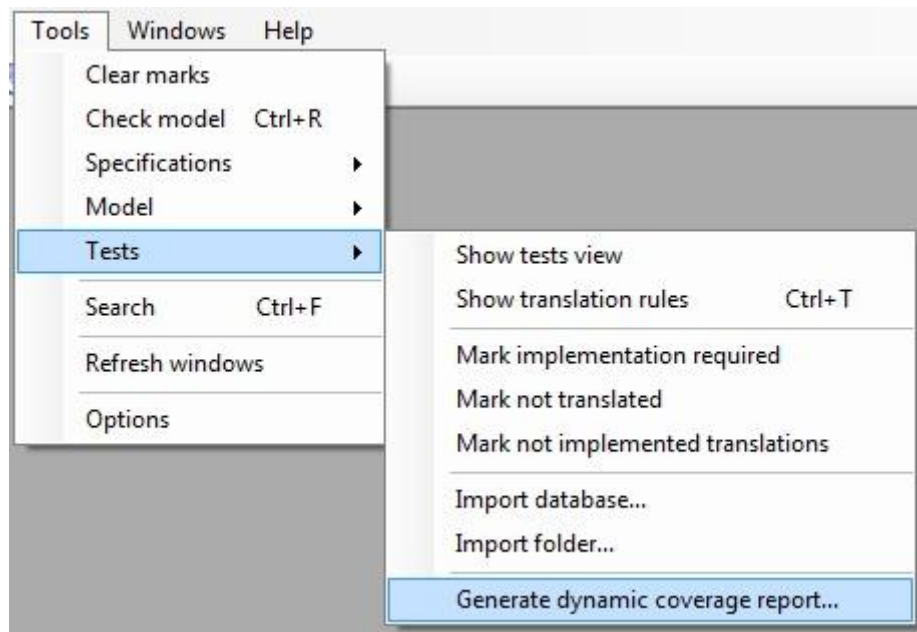


Figure 27 – Launch the dynamic test coverage report

The option of a **partial** report creation for a selected item of the tests tree is accessible via the "Create report" option from its contextual menu.

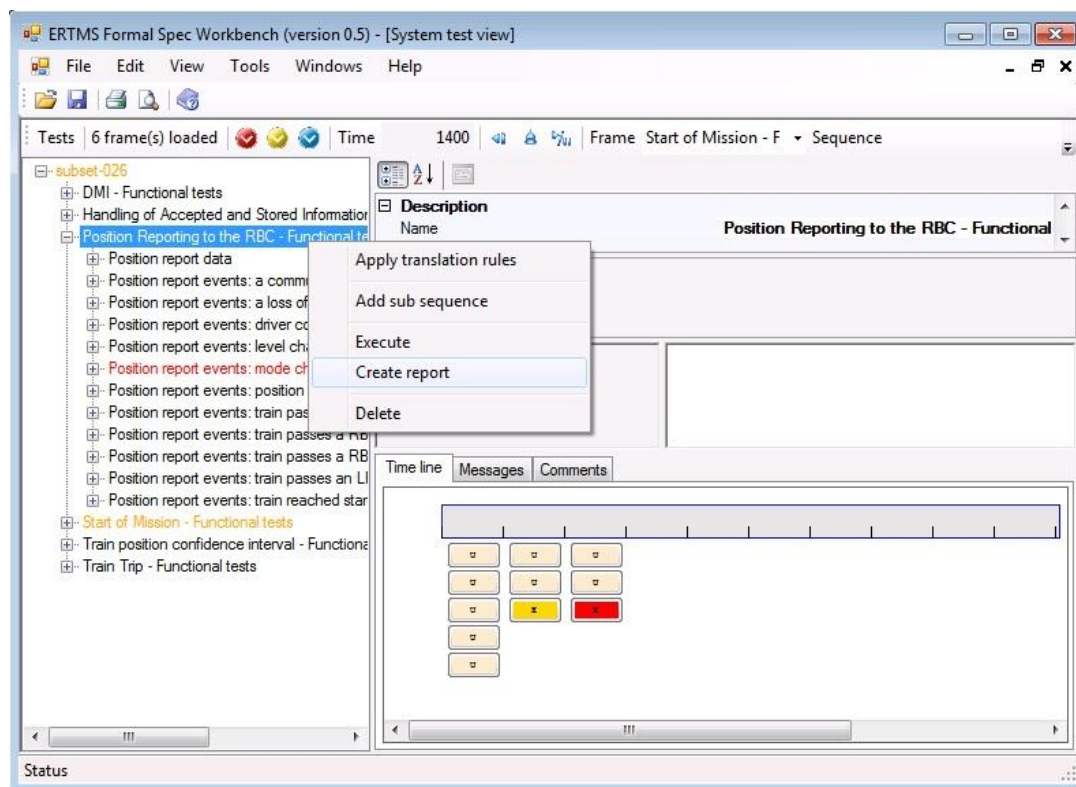


Figure 28 – Report creation for a specific sub sequence

This action opens the dialog box displayed below.

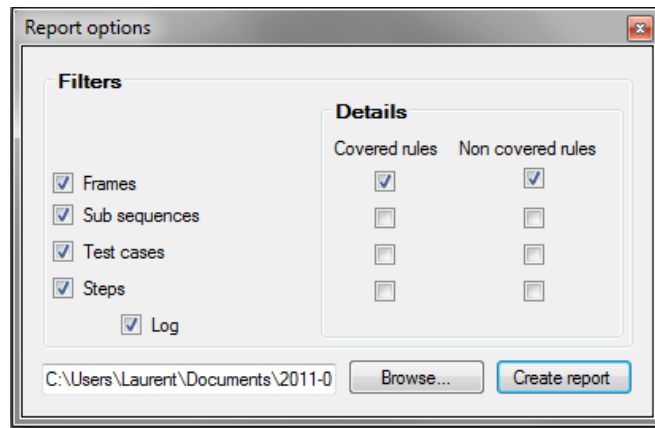


Figure 29 – Dynamic tests coverage report options

The figure Figure 29 shows the window permitting to select the different report options. The check boxes in "Filters" group box allow to select the different levels of the report (see section 10.2.2). The corresponding check boxes in the "Details" group box allow to add the list of covered and/or not covered rules to the corresponding level. "Log" check box allows to display the log information for the different steps.

The "Browse" button allows to select the name and the folder of the generated report.

## 10.3 Data dictionary report

### 10.3.1 Purpose

The purpose of the data dictionary report is to provide information about the model elements whose implementation is complete. The report can be created on two different levels of details:

- **Default level:** the report provides only the list of implemented model elements together with their associated requirements.
- **Detailed level:** the report provides all the available details for each implemented model element.

### 10.3.2 Structure

The report is divided in several chapters each one corresponding to one of the data dictionary namespace. Depending on the user's choice, each chapter can contain information about its

- **Ranges.** For each implemented range of the current name space, the report contains information about its min and max values, about its default value and precision. The list of its special values is also provided.
- **Enumerations.** For each implemented enumeration of the current name space, the report provides the list of its possible values together with its default value.
- **Structures.** For each implemented structure of the current name space, the report provides the description of its sub-elements, its procedures and its rules.
- **Collections.** For each implemented collection of the current name space, the report provides its type, its max size and its default value.

- **Functions.** For each implemented function of the current name space, the report provides its return type, the description of its parameters and its cases (their pre-conditions and expressions).
- **Procedures.** For each implemented procedure of the current name space, the report provides the description of its parameters, rules and the state machine.
- **Variables.** For each implemented variable of the current name space, the report provides its type, default value (if any) and its mode.
- **Rules.** For each implemented rule of the current name space, the report provides the description of all its rule conditions and its priority.

For each one of the elements described above, the data dictionary report provides a comment describing its utility. If a model element which implementation is not yet complete contains some sub elements that are already implemented, this element is also included in the report. Each model element of the report contains also a table describing its implementation and verification statuses.

### 10.3.3 Launch the model report

The data dictionary report creation window is accessible via [Tools/Model/Generate data dictionary report...](#) (see Figure 30).

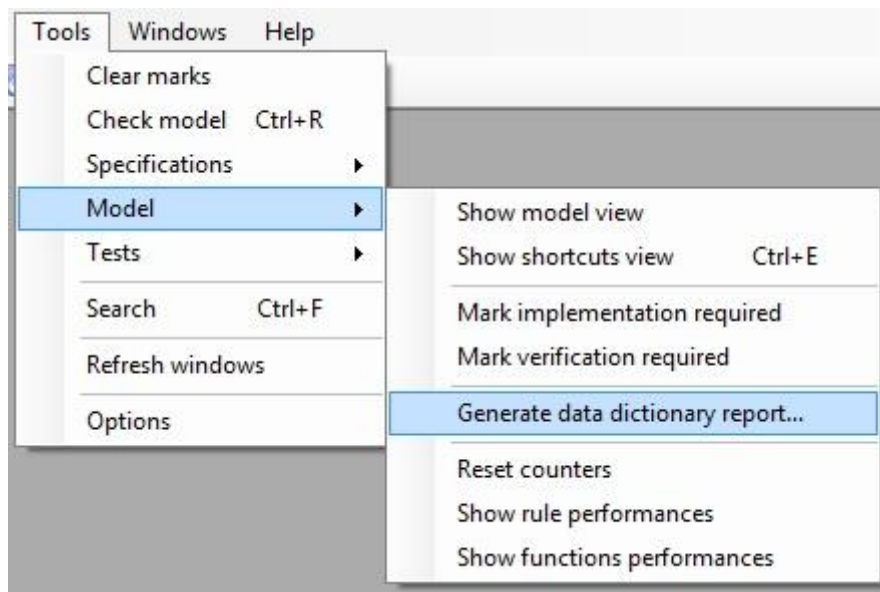


Figure 30 – Launch the data dictionary report

This opens the dialog which allows to select the report options, as depicted below.

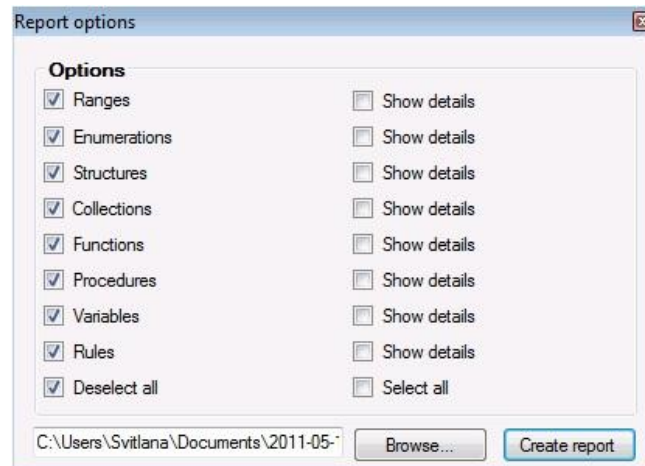


Figure 31 – Data dictionary report options

The different check boxes allow to select the type of elements to be included in the report, and precise if the latter has to describe the details of these elements.