ERTMS Formal Specs Workbench

Coding Guidelines, Version: 0.4

Moritz Dorka (MD)

16/01/2014

Revision History

Revision	Date	Author(s)	Description
0.1	31/10/2013	MD	initial draft
0.2	12/11/2013	MD	add ”How to use this file”, ”Comments”, ”Glossary”, ”Bibliography”
0.3	14/12/2013	MD	add ”Committer checklist”
0.4	16/01/2014	MD	add a section about ”types” for the DSL

Contents

How to read this file

This document lists rules to be followed when coding in the Domain Specific Language (DSL) of the ERTMS Formal Specs Workbench (EFS) tool. It is targeted at both model developers and test-case designers. ...
It will not cover the technical implementation of the underlying DSL or give general instructions on how to use the EFS-tool. For this refer to the Technical design [**?**] and the user guide [**?**], respectively.

In the appendix A a short checklist may be found which describes how to correctly prepare a commit for the ERTMSFormalSpecs repository on GitHub.

TODO$_{MD}$

Typographical conventions

Rule texts

Each rule has a unique identifier composed of the current section and a running number. Words to be used verbatim within the rule texts are printed in *emphasized* letters. Rule texts conclude with the initials of the rule author and the date of the last change of the rule.

Code examples

Code of the DSL used for examples within the rules is printed in `typewriter` font with

- language keywords in **`bold`**,
- pre-defined functions `underlined`,
- placeholders *`emphasized`* and
- string literals in a sans-serif font.

For a documentation of the keywords and the pre-defined functions see [**?**].

1. Domain specific language

1.1. General

1.1.1. TODO$_{MD}$ **EMPTY** vs. **NOT** Available() vs. [] vs. (**COUNT** *List*)== 0

MD, 31/10/2013

1.1.2. **SUM** *List* **USING X** shall be preferred over **REDUCE** *List* **USING X** + **RESULT**.

MD, 31/10/2013

1.1.3. **THERE_IS_IN** *List* | *Condition* shall be preferred over **MAP** *List* | *Condition* **USING X**.

MD, 31/10/2013

1.2. Types

1.2.1. Enumerations

1.2.1.1. Always use the symbolic names of the items of an enumeration and never their numerical value.

MD, 14/01/2014

1.2.2. Ranges

1.2.2.1. Do not use any kind of arithmetic operator on special values except for equality (that is ==).

MD, 16/01/2014

1.2.2.2. Do not create a range that only consists of special values. Use an enumeration (see section 1.2.1) instead.

MD, 16/01/2014

2. Model

2.1. General

2.1.1. All decisions made by the modeler which are not purely cosmetic (i.e. using a Function instead of a Procedure), but have an influence on the actual behavior of the model shall be documented in the section *Design Choices* of the Specification View and linked to the respective code.

MD, 12/11/2013

2.2. Functions

2.2.1. Cases

2.2.1.1. General

2.2.1.1.1. TBD$_{MD}$ Cases shall only contain trivial expressions as their return values (i.e. all logic shall be defined inside pre-conditions).

MD, 31/10/2013

2.2.1.1.2. TBD$_{MD}$ Complex return expressions and pre-conditions shall not be used together. this rule is mutually exclusive to 2.2.1.1.1.$_{MD}$

MD, 31/10/2013

2.2.1.2. Pre-conditions

2.2.1.2.1. Cases with no pre-condition assigned shall always come as the very last case.

MD, 31/10/2013

2.2.1.2.2. Cases with no pre-condition assigned shall always be named *Otherwise*.

MD, 31/10/2013

3. Tests

3.1. Structure

3.1.1. TBD$_{MD}$  A Sub-step shall either contain Actions or Expectations but not both.

MD, 12/11/2013

3.1.2. The first Sub-step of the first step of a test frame shall only contain the expression `Kernel.InitializeTestEnvironment()`.

MD, 31/10/2013

3.1.2.1. There shall be no Expectation associated with this Sub-step.

MD, 31/10/2013

3.2. Naming conventions

3.2.1. Individual test steps shall be named *Step n - explanation*.
Explanation is a user-defined text to summarize the actions and expectations of the current step.

MD, 31/10/2013

4. Comments

4.1. General

4.1.1. Comments shall be used wherever the inner workings of a code are not directly obvious from the documentation provided by the linked requirements or design choices.

MD, 12/11/2013

4.1.2. Comments shall always be assigned to the narrowest possible scope to which they apply (i.e. to an individual test-step rather than to the entire test).

MD, 12/11/2013

4.1.3. Comments shall not be used to justify Design Choices. See rule 2.1.1. instead.

MD, 12/11/2013

4.1.4. redundant$_{MD}$  Comments shall always end with #<*github_username*>, *yy/mm/dd*. Where <*github_username*> is to be replaced with the actual username of the comment-author and *yy/mm/dd* with the current date.

MD, 12/11/2013

5. Git

5.1. General

5.1.1. Own edits shall always be committed into a personal fork of the EFS repository and then transferred by issuing a pull-request. See 5.3.

MD, 31/10/2013

5.1.2. The EFS repository located at `github.com/openETCS/ERTMSFormalSpecs.git` shall always be named *upstream* inside SmartGit.

MD, 20/01/2014

5.1.3. If the user has write-access to the EFS repository it is advisable to force a different username for it inside SmartGit so that writing is limited to pull requests as described in 5.1.1. This can be achieved by entering the reposisitory URL like so: `https://dummy@github.com/openETCS/ERTMSFormalSpecs.git` (*dummy* being the "different username" here).

MD, 20/01/2014

5.2. Commits

5.2.1. General

5.2.1.1. numerous small commits dealing with a single issue are preferable over few huge commits possibly dealing with many different issues.

MD, 31/10/2013

5.2.2. commit description texts

5.2.2.1. The text shall not contain line-breaks (i.e. consist of a single line only).

MD, 31/10/2013

5.2.2.2. Commits dealing with the modeling part of the tool shall begin with *EA_MODEL* followed by a reference to the sections of the subset the commit is related to and a descriptive text.

MD, 31/10/2013

5.2.2.3. Commits dealing with the testing part of the tool shall begin with *EA_TEST* followed by a reference to the sections of the subset the commit is related to and a descriptive text.

MD, 31/10/2013

5.2.2.4. Commits dealing with the documentation part of the tool shall begin with *EA_DOC* followed by a reference to the documents the commit affects and a descriptive text.

MD, 20/01/2014

5.3. Pull-Requests

5.3.1. General

5.3.1.1. Use a separate branch for each new "feature" (i.e. a coherent set of tests) and then issue a pull-request from that branch.

MD, 20/01/2014

A. Committer checklist

This is a short checklist that everyone willing to push code onto the ERTMSFormalSpec repository should go through before every commit.

1. **make small commits**
   Each commit should only change one logical thing (i.e. one test-case at a time). See 5.2.1.1.

   MD, 14/12/2013

2. **use meaningful titles for new nodes**
   For tests see 3.2. For the model: be creative!

   MD, 14/12/2013

3. **link to requirements**
   Always link your tests and model-code with the respective requirements.

   MD, 14/12/2013

4. **set implementation-flag**
   If you implemented any requirements make sure to mark their respective nodes in the specification view with *Implementation Status: Implemented*.
   Reviewed: True should also has to be set if you verified the requirement text matches with that of the original word document of subset-026 [**?**].

   MD, 14/12/2013

5. **check model**
   Please hit *CTRL+R* in EFS and see if there are any errors introduced by your work. If so, fix them.

   MD, 14/12/2013

6. **commit to your own fork**
   Commit your changes to your own fork of the ERTMSFormalSpec repository and then create a pull request. See 5.1.1.

   MD, 14/12/2013

7. **use meaningful commit messages**
   It is merely impossible to change them afterwards so please obey 5.2.2.

   MD, 14/12/2013

Glossary

DSL . . . . . Domain Specific Language

EFS . . . . . ERTMS Formal Specs Workbench

List of Corrections

List of PDF comments