ERTMS Formal Specs: a domain specific language to formalize ERTMS specifications for onboard unit development

Laurent Ferier (laurent@ertmssolutions.com)
ERTMS SOLUTIONS – Brussels

Stanislas Pinte (stant@ertmssolutions.com)
ERTMS SOLUTIONS – Brussels

Darius Blasband (darius@ertmssolutions.com)
ERTMS SOLUTIONS – Brussels

The European Railway Traffic Management System (ERTMS) defines standards for interoperability between the onboard train protection systems (ETCS) and the railway infrastructure. The ERTMS specifications are of significant size, and as of today, specified in plain English only. ERTMSFormalSpecs (EFS) is a project based on a domain-specific language, designed to express the ERTMS specification in a concise and verifiable formal representation. It aims at being understandable by domain specialists while retaining the ability to be interpreted or translated to efficient executable representations by fully automated means.

1 The Context
The European Railway Traffic Management System (ERTMS) defines the standard for interoperability between the onboard train protection systems (ETCS) and the railway infrastructure. This standard is decomposed into several documents (named Subsets) focusing on specific parts of the system.

The ERTMS Formal Specs (EFS) focuses on modelling the requirements expressed in Subset 26 – System Requirements Specification related to the trainborne equipment, and applying the tests specified in Subset 76 – Test cases on the model.

This modelling effort is aimed at generating code according to the customer’s target’s language and providing the required artifacts to prove the match between the generated code and the ERA requirements, as presented in Figure 1. The customer can also use the model without code generation. In that case, EFS provides artifacts that can be used to verify the match between the model and the ERA requirements.
The modelling effort requires to tackle the following issues:

- **Natural language**: Requirements in Subset 26 are expressed in plain English. This may lead to misinterpretations inherent to natural languages.

- **Structure**: Subset 26 organization follows the structure associated to requirements. This structure is not suited to model these requirements in a formal system, such as a computer program.

- **Size**: The size of the Subset 26 (500 pages – 39 messages – 47 packets – 176 variables) makes the work of checking the match between implementation and requirements extremely difficult. This work must nevertheless be performed for certification purposes.

- **Completeness**: While extreme care has been taken while specifying the Subset 26, several requirements may still have been overlooked by the engineer. A formalization of these requirements, supported by automated tools, allows identification of the holes in the specification.

- **Consistency**: The amount and complexity of requirements forbids using only pair reviews to guarantee inconsistencies between the requirements. This consistency check must be supported by automated tools.

- **Releases**: The ERA specifications are in constant evolution to match new challenges. One must be able to update the model according to the new requirement releases.

The following sections shall demonstrate how the EFS address to handle the issues mentioned above.

## 2 EFS Description

The EFS product is made up of two components. The **model** includes the requirements expressed in Subset 26, the complete data description, sub-system decomposition and business logic of an EVC, specified in UNISIG Subset-026 specifications, expressed in the EFS language, and the tests from Subset 76 tests as well as additional tests needed to cover the Subset 76 holes.
The **workbench** is a graphical tool designed to develop, maintain, and document model-based development for the Subset-026. It is a desktop application, running on Microsoft Windows platform.

The **EFS language** is a domain specific language, developed by ERTMS Solutions for the sole purpose of modelling the ERTMS specifications. It is thus not clobbered by history of irrelevant application domains, nor by universal features that are commonly included in general purpose formalizations. This language design is driven by two opposite constraints:

- **Constraint 1**: To be as close as possible to the artifacts used in the UNISIG Subset-026 specifications (e.g. plain English, state diagrams, tables) to demonstrate the equivalence between specification and implementation
- **Constraint 2**: To be formal to allow interpretation, inferences and deduce system properties.

### 2.1 Characteristics

The following design principles used in EFS set it apart from other general-purpose formal methods: to be traceable, to be understandable, to allow support for automated reasoning and to be testable. The following sections present each design principle and show how they are applied in EFS.

#### 2.1.1 Traceability

The model supports traceability between requirements, the corresponding model, and the tests used to verify the model. This information is of utmost importance and allows the Workbench to enforce several structural constraints:

- Each model element must be related to at least one requirement
- Each test must be related to at least one requirement
- Each requirement must be related to at least one model element and one tests

It is the responsibility of the modelling engineer, supported by the modelling process (see Section 2.2), to ensure that both model and tests comply with the related requirements.

This trace information is used to perform impact analysis when changes occur in the model. When a requirement changes, impact analysis help determine the parts of the model that are affected by this requirement, and should be reviewed.

Impact analysis is also used to select the tests that should be rerun according to changes of part of the model. Note that this information is only interesting when performing integration tests, which require hardware mobilization, engineers... As we shall see in Section 2.1.4, tests performed on the model itself are quick and easy to execute.

#### 2.1.2 Understandable

The EFS language has been specifically designed to be understandable by domain experts. A flaw of general-purpose formal language is that such languages can only be understood by the formal language specialists\(^1\). This makes the formalization of the system unverifiable by the real users, the domain experts.

---

\(^1\) For instance, see [FmSurvey] underlining that the [SACEM] project team reported a difficulty in communication between the verifiers and the signalling engineers, who were not familiar with the B-method.
To overcome this, the EFS model is split in two layers.

- The complete behaviour of the EVC system is specified using a **rule based layer**, a restricted formal language described in Section 2.1.3. This is the only interpretation/translation artifact used in EFS.

- The **higher layers**, translations of the rule layer, display the model using artifacts commonly used in the UNISIG Subset-026 specifications (such as state diagrams, tables and flowcharts). The requirements that are expressed in plain English, cannot, in the general case, be translated into high level diagrams. In that case, the rules are translated in pseudo-code to be easily reviewed by the domain specialist.

These higher layers allow the domain expert to trace back the model to the specifications. Model animation, as we shall see in Section 2.1.4.2, is performed using the rule based layer, but is displayed to the user using higher layer views.

Figure 2 shows that, even if the complete state machine is expressed in terms of rules (low level layer), the Workbench displays this information using state diagrams which correspond to the state diagrams as expressed in the Subset26, easing the model verification process.

![State diagram example](http://www.ertmssolutions.com)

Another example, illustrated by Figure 3, displays the Subset 26 English requirement in the left part of the description tab whereas the corresponding pseudo-code, based on the underlying model, is provided in the right part of the description tab. Using this artifact, the system engineer can easily verify that the implementation matches the related requirement.
2.1.3 Reasoning about the model

The behaviour of the EFS model is expressed as simple rule applications. These rules are composed of two parts

- **Preconditions**: a set of conditions that should all be evaluated to true in order to apply the rule
- **Actions**: actions express the variable updates to be applied when the rule is activated. Each action consists of a single variable modification, where the variable's new value is the value of a single expression

Rules can be organised in a hierarchical way to structure them together. This also allows optimizing the model interpretation by factorising common preconditions: when the preconditions of a rule are not valid, none of the sub rule must be evaluated.

This formal model allows to perform automated **simple proofs** such as **syntax checks** which ensure that expressions are correct, that all variables used are defined... and to perform type checks. Furthermore, automated **advanced proofs** can also be performed on the model, which includes

- Enforce **structural constraints** to ensure model consistency, for instance, that all message variables are always set
- Enforce **non-contradiction** between rules, which is performed by ensuring that, if two rules modify the same variable, they cannot be activated at the same time
- Ensure that all states of a state machine can be **reached**

In order to enable independent verification of the formal proof, the Workbench produces a “proof log”, i.e. a journal of all computations made by the proving routines inside the tool, executed on the model.
2.1.4 Testability
The EFS includes all official Subset 76 test sequences which can be run against the model inside the Workbench. It is important to note that these Subset 76 test sequences are the same tests that are performed inside the ERTMS Reference Laboratories\(^2\), against the final EVCs subsystems. Having the same tests performed on the formal model of the EVC and on the actual EVC hardware provides a strong evidence of compliance of requirements throughout the whole V-cycle.

2.1.4.1 Test definition
The Workbench allows to model tests the same way tests are specified in Subset 76. Tests are specified by a sequence of test cases, each test case being a sequence of steps. One step consists of

- altering input values in the model, which corresponds to changes induced by **input observables**
- checking output values from the model, which correspond to the **output observable expected values**

This test formalization allows exercising 100% of the model inside the Workbench.

Added to the test formalization, the EFS stores the textual representation of the test, as expected in Subset 76. This allows the domain expert to review the tests and ensure they correspond to the expected system behaviour.

Traditionally, the link between the Software Requirement Test Specification (SRTS) and the Software Requirements Specification (SRS) is manually established, i.e. a human being lists all SRS elements tested by a given test of the SRTS, and manually assess if the tests are really testing the requirements. This corresponds to the concept of feature defined for tests in Subset76. The EFS workbench allows to automatically perform coverage analysis, based on the feature concept and to identify the requirement that have not been verified by a test.

Moreover, thanks to model interpretation as presented in Section 2.1.4.2, the Workbench can be used to crosscheck the traceability information provided by features: the Workbench identifies the rules activated during a specific test and traces back the requirements involved in a specific test.

2.1.4.2 Interpretation
According to the fact that structure constraints are respected, the model can be interpreted in a non ambiguous way to perform simulations and tests. The Workbench can execute tests against the model using the rule interpreter. During test execution, a timeline displays all events that occurred, providing a visual way to analyze and verify system behaviour. This diagram displays

- The rules that are activated at a specific time
- The variables changes
- The observable values expected by the test currently run

For instance, the screenshot presented in Figure 4 displays a model execution where the value of an observable does not correspond to the expected value (in red). This indicates that the test is not successful.

---

\(^2\) CEDEX, DLR and Multitel as of 1st of January 2011.
Figure 4 - Test execution

A test can be executed as a whole, or step by step to trace the system evolution. During execution of a test, the test engineer can modify variables on the fly, to perform ‘what if’ investigations on the model.

During a step by step test execution, the high level diagrams are updated according to the system current state, as presented for State diagrams in Figure 5, which allows the system engineer to easily review the model evolution.

Figure 5 – Model evolution displayed in a state diagram
2.1.4.3 Test results
Based on the results gathered during test execution, the Workbench issues a test result report, indicating the successful tests, and if a test is failed, the output observable values that were not satisfied. A detailed test log is available for each test, showing how the model has been simulated.

The Workbench also issues a rule coverage report, illustrating which rule is exercised for each test, with a granularity of a single rule and of a single test step. This test report can be used to univocally identify untested rules.

2.1.5 Openness
The complete model is stored by the Workbench in a fully documented XML file. This allows third party software to perform additional work based on the formalization effort performed by ERTMS Solutions. For instance, a third party team can use this information to perform some model testing on the model using ad hoc tools.

2.2 Modelling process
The modelling process is performed according to the following phases

1. **Review the requirements.** The requirements have been manually encoded in the tool and need to be reviewed against the textual requirement as presented in Subset 26 to ensure that no error occurred during this phase. This phase is also used to split or group requirements according to the needs related to the modelling phase and to uniquely identify tables specified in Subset 26.

2. **Update the model** according to the requirement to be modelled, and create the relation between the model and the requirement. At the end of the modelling phase, the engineer indicates that the requirement has been implemented.

3. **Review the model.** When modelling is complete, another engineer must review the newly created model against the requirements it implements. When the review is complete, the engineer indicates that the model has been verified.

2.2.1 Test creation
A parallel work consists of creating tests which shall ensure that the model corresponds to the expected behaviour. The modelling process and the test creation is an iterative process, until both model and tests agree with the expected behaviour.

Each test must identify the requirement it verifies. As soon as both model and tests are created, the test engineer can use the test environment of the Workbench to ensure that the model is behaving correctly and create a test report. Since test execution is based on an in-memory test interpreter, it is cheap to execute them. Therefore, it is easy to always re-run all tests and ensure non regression on the entire model.

2.2.2 Historical data
Each action applied on the system (review a requirement, modify a model element, verify a model...) is tracked with the following information: the person that performed the action, the action performed and a timestamp.

The Workbench provides all the required tools to support this process. It allows to identify the requirements that were not yet reviewed, or that are not completely implemented, to identify the model which implementation is not complete, or which needs to be reviewed. It also identifies the requirements which are not covered by a test, or the rules which were never activated during testing.
2.3 Interpretation or code generation

The formal model is interesting *per se* since it provides a way to disambiguate the requirements expressed in natural language. It can also be used to detect gaps in the requirements or to determine the inconsistencies between them if any.

Thanks to its clean structure, the formal model can also be used to generate code to be embedded on the target platform. Target languages may vary from C, Ada to Scade or Simulink environments. Code generation plugins are created on a project basis, according to customers needs. The model can also be used as such by the customer, this implying that the customer develops a SIL4 model interpreter.

Either in the context of code generation or in the context of interpretation, tests can be used to ensure that the behaviour of the generated code is the same as the behaviour of the interpreted model.

2.4 Modelling effort

The current ERTMS Solution plan for the modelling effort is displayed in Figure 6. The complete formalization of Subset 26 shall be completed during the second quarter of 2012, along with 100% unit test coverage, whereas integration tests, as defined by Subset 76 are planned to be completed on at the end of the third quarter of 2012.

![Modelling effort](image)

**Figure 6 - Modelling effort**

3 Conclusions

In this paper, we have described that formalizing Subset 26 addresses several issues such as the fact that the specification is written in natural language and is intrinsically ambiguous. Furthermore, pair review is not sufficient to ensure that the specification is complete and consistent. Last, keeping the pace with the successive releases of the document causes issues when the corresponding implementation has been performed using standard implementation techniques.

To handle these issues, we proposed to formalize the requirements expressed in Subset 26 and formalize the tests expressed in Subset 76. The resulting model has the advantage of having an unambiguous interpretation. Tools can be used based on this model: interpretation of the model to animate it, test execution to ensure that the model behaves as defined in Subset 26 and Subset 76, automatic model walkthrough to statically prove properties about the model.
A domain specific language to formalize ERTMS specifications for onboard unit development

The formalization work is also focused on traceability issues: all model elements are traced back to their requirements, tests are traced to the requirements they verify. This information is used to perform coverage analysis, such as determine the requirements that are not modelled, or those that are not tested. Cross checks between model traceability tables and test traceability tables are performed using information gathered during testing, and can be used to improve the tests.

The model can be used to generate code to be run on the target platform. Tests defined on the model are in that case used to ensure the correspondence between the model and the generated code.

4 Further works

The formalization effort is currently ongoing by ERTMS Solutions team and shall be 100% completed at the end of the second quarter of 2012. This work will allow conducting further research in the following directions:

- How can EFS be used to improve the quality of the Subset 26 English-based, informal specifications?
- How can be EFS be used to proof that the EVC using EFS shall never be put in an unsafe state, i.e. how can EFS be used to proof safety-related properties?

The complete model will also allow to perform complex system model checking.

5 Bibliography

The following documents are referenced inside that document:

- [PolimiCameraRFI_FormalMethods] [http://www.ita.disco.unimib.it/quack/docs/Polimi-cameraRFIPolimi.pdf](http://www.ita.disco.unimib.it/quack/docs/Polimi-cameraRFIPolimi.pdf)
- [FmSurvey] [http://www-users.cs.york.ac.uk/~jim/fmsurvey.pdf](http://www-users.cs.york.ac.uk/~jim/fmsurvey.pdf)