Technical Report

- **Project**: DigiLightRail
- **WP**: WWW
- **Deliverable**: T6.3
- **Producer**: HASLab / INESC TEC
- **Title**: Integration testing the components of the EVEREST tool
- **Summary**: *This report (.....)*

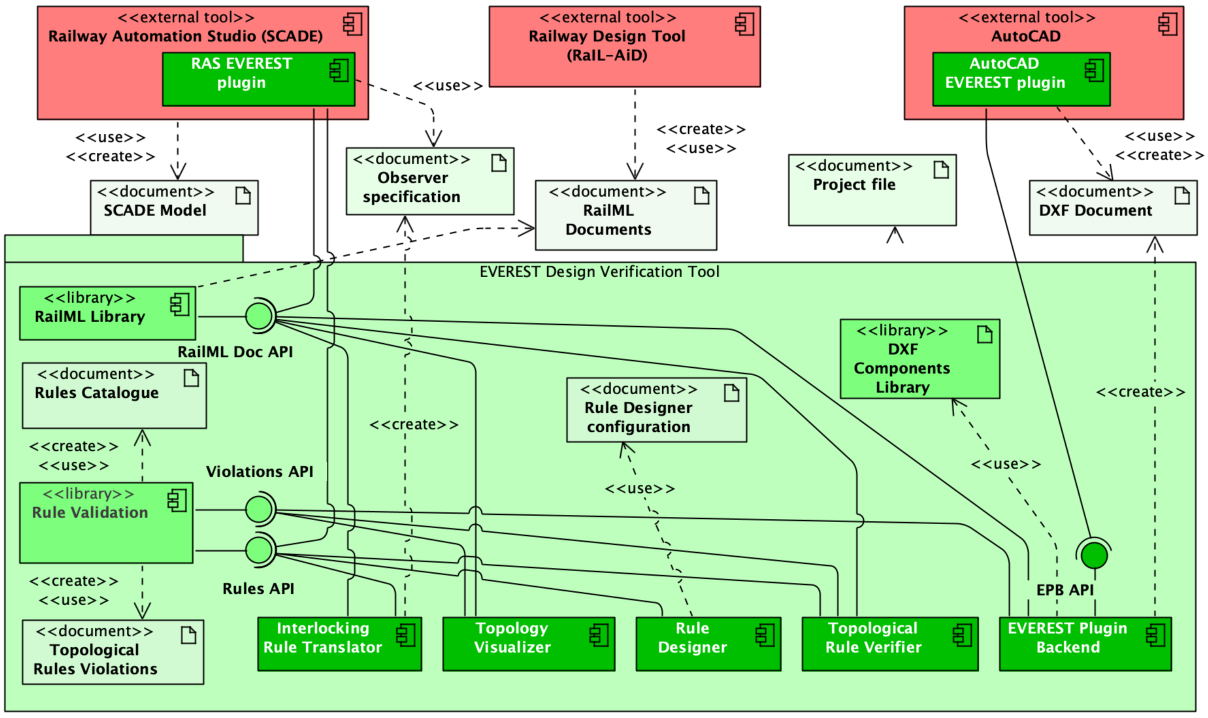
Introduction

Project DigiLightRail aims at designing and implementing a platform for the design of command and control systems for light surface trains (vulg "*metros*" in the French terminology), allowing for the configuration of *automatic train protection* (ATP) systems and the basic configuration of an entire *cyber-physical system of systems* (CPSoS) control and command system.

One of the main deliverables of DigiLightRail is a design automation tool, EVEREST (**E**facec **Ve**rification of **R**ailway n**E**twork**S** **T**ool). The goal of this tool is to automatically validate the design of railway infrastructures through user-defined infrastructure and interlocking rules. This tool's architecture, as shown in the picture below, is a set of loosely coupled components. These components will support the specification of rules, their verification against concrete railway topologies, and subsequent reporting of validation. The components are the following:

- **Rule Designer (RD):** offers a UI to define a set of rules that a railML model should follow.
- **Topological Rule Verifier (TRV):** grants the possibility to test a railML model against a set of user-defined infrastructure rules.
- **Topology Visualizer (TV):** provides a UI that can display railML topologies as graphs as well as infrastructure rule violations.
- **Interlocking Rule Translator (IRL):** translates interlocking rules to propositional logic formulas that specify observers that can later be incorporated into SCADE models using the RAS EVEREST Plugin.
- **RAS EVEREST Plugin (REP):** a Railway Automation Studio (RAS) plugin that creates SCADE observers from the propositional logic formulas resulting from interlocking rules.
- **AutoCAD EVEREST Plugin (AEP):** an AutoCAD's extension that allows users to import railML documents as DWG entities, enrich the railML model with topological information, and depict topological rule violations in a drawing.

On a technical level, these components are implemented as described in the component diagram below (see report T2.6 for details).



<<external tool>>
Railway Automation Studio (SCADE)
RAS EVEREST plugin
<<external tool>>
Railway Design Tool (RaIL-AiD)
<<external tool>>
AutoCAD
AutoCAD EVEREST plugin
<<use>>
<<create>>
<<document>> SCADE Model
<<document>> Observer specification
<<document>> RailML Documents
<<document>> Project file
<<document>> DXF Document
EVEREST Design Verification Tool
<<library>> RailML Library
RailML Doc API
<<document>> Rules Catalogue
<<library>> DXF Components Library
<<document>> Rule Designer configuration
Violations API
<<library>> Rule Validation
Rules API
EPB API
<<document>> Topological Rules Violations
Interlocking Rule Translator
Topology Visualizer
Rule Designer
Topological Rule Verifier
EVEREST Plugin Backend

In this document we report on the integration testing of main components that comprise the **EVEREST** design automation tool and their interaction. The unit testing of the Design Verification Tool libraries is addressed elsewhere (see report T5.4).

Scope

The goal of this process is to test the functionalities and the interaction of the various components of the EVEREST design automation tool, namely the sub-components that comprise the Design Verification Tool (DVT), and its interaction with the AutoCAD EVEREST Plugin (AEP) and the RAS EVEREST Plugin (REP).

Strategy and technologies

Our tests fit into the following main categories:

- testing the interaction of the sub-components of the DVT, focusing on usage scenarios for the various functionalities of the DVT
- testing the AEP, focusing on usage scenarios for the various functionalities of the AutoCAD side
- testing the interaction between the DVT and the AEP, in particular focusing in scenarios where the flow of information is
    - from the DVT to the AEP (the eventual introduction of violations found by the DVT Topological Rule Verifier)
    - from the AEP to the DVT (the integration of physical positions in the railML model to be used in the DVT Topological Rule Verifier)

- testing the interaction of the DVT and the REP, with scenarios where information is propagated from the DVT to the REP (feedback from the REP is not propagated back to the DVT)

In order to systematise the definition of the testing scenarios, we encoded them in a language for the specification of human-readable use cases, *Gherkin*, which is briefly presented next.

Test Specification

Gherkin is a human readable language which helps to describe business behaviour without implementation details[1]. It uses a set of special keywords to provide structure and meaning to executable specifications and plain language to describe use cases, allowing the removal of logic details from tests.

All the Gherkin test scenarios specified below are catalogued in the Azure DevOps Server of the EVEREST development team. The test's specifications were built with the following subset of Gherkin keywords:

- **`Given`**: used to describe the initial context of the system/test.
- **`When`**: used to describe an event or a user action with the system during the test.
- **`Then`**: used to describe the expected outcome.
- **`And`**: used to replace successive **`Given`**, **`When`** or **`Then`** steps.

Test Execution

There is some tool support to automate the execution of integration tests for .NET applications (the technology in which DVT is developed), in particular for test cases specified in Gherkin. Unfortunately we were not able to effectively deployed such tools, in particular:

- **SpecFlow**, a Behaviour Driven Development (BDD) framework for .NET that assists with the writing of feature files and code automation[2]. It allows users to write a set of step definitions using the Gherkin syntax and to bind the defined steps with methods of the code. This framework allowed the creation of test cases for both the Model and ViewModel layers, but it was not fully compatible with the Presentation layer developed with WPF. Therefore, we did not perform the integrated tests with this framework.
- **Appium**, an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms[3]. To automate Windows PC Desktop apps, Appium uses a driver that relies on the Microsoft WinAppDriver project, which is an Appium-compatible WebDriver server for Windows Desktop apps. With this tool, we were able to define tests on the UI layer of the EVEREST Design Verification Tool that rely exclusively on static UI components. However, this tool does not provide the means to access components such as

[1] https://cucumber.io/docs/gherkin/reference/
[2] https://specflow.org/
[3] https://appium.io/docs/en/about-appium/intro/

ListView cells that are generated on runtime. Due to this reason, we did not use this tool to perform the integrated tests.

We were also unable to find tool support for the automation of scenarios in AutoCAD to exercise the AEP test cases.

Due to these limitations, all test scenarios have been executed and checked manually. They were all executed manually with a successful result, as reported in the Azure DevOps Server of the EVEREST development team.

Test scenarios

In this section we present the tests made to check the correctness of the AutoCAD EVEREST Plugin and the EVEREST Design Verification Tool. We firstly present tests that check each one of these components and then we provide tests that aim to check the correctness of these components when working together. Each conducted test was specified in Gherkin.

AutoCAD EVEREST Plugin

In this section, we present the tests made to check the correctness of the AutoCAD EVEREST Plugin, taking into consideration the functionalities defined in document T2.6. This plugin also uses the railML library to perform its commands. The tests should be executed in the same order as presented in this document to produce the correct results, since the plugin's commands are sequential and dependent on each other.

Import railML area

The goal of this test is to use both AutoCAD EVEREST Plugin and the railML library to read a railML area file and import its model to a DWG drawing, testing the `IMPORT-RAILML` functionality. Its Gherkin specification is the following:

```
Given an AutoCAD DWG file
When the user types the command IMPORT_RAILML
And provides a railML file
And picks a coordinate in the DWG drawing
And enters the scale factor of the railML area
Then the railML area model should be visible in the DWG drawing
```

Break DWG entity

This test aims to check the process of breaking a single entity of a DWG drawing into net elements, testing the `MOVE-RAILML-BLOCK` functionality. Its Gherkin specification is the following:

```
Given an AutoCAD DWG file with a railML area
Given an LWPolyline entity in the DWG drawing
```

```
And an endpoint block of the railML model
When the user moves the endpoint block to its correct position in
the LWPolyline
And types the command BREAK_NET_ELEMENT
And selects the LWPolyline
And selects the endpoint block
Then the LWPolyline should break at the endpoint block’s position
```

Break all DWG entities

This test aims to check the process of breaking a set of entities of a DWG drawing into net elements, testing the `MOVE-ALL-RAILML-BLOCKS` functionality. Its Gherkin specification is the following:

```
Given an AutoCAD DWG file with a railML area
Given the LWPolylines in the DWG drawing that correspond to the
physical model
And an endpoint blocks of the railML model
When the user moves the endpoint blocks to their correct positions
And types the command BREAK_ALL_NET_ELEMENTS
And provides the layer of the LWPolylines to break
Then all target LWPolylines should break at the endpoints’ positions
```

Label net element

This test encompasses two commands of the AutoCAD EVEREST Plugin: `LABEL_NET_ELEMENT` and `VERIFY_LABELS`, that implement the `LABEL-NET-ELEMENT` and `VERIFY-LABELS` functionalities. In the first part of the test, the user should pick an entity that corresponds to a net element and execute the `LABEL_NET_ELEMENT` command to attach the net element’s id to that entity. While the second part of the test provides the means to check the correctness of the first part. In the second part, the user should execute the `VERIFY_LABELS` command to check if the selected entity was correctly labelled. Its Gherkin specification is the following:

```
Given an AutoCAD DWG file with a railML area with entities break
into net elements
Given a net element entity
When the user types the command LABEL_NET_ELEMENT
And enters the name of the net element
And types the command VERIFY_LABELS
Then the net element labelled should be painted in green in the
imported railML model
```

Label all net element

This test verifies two commands of the AutoCAD EVEREST Plugin: `LABEL_ALL_NET_ELEMENTS` and `VERIFY_LABELS`, that implement the

`AUTO-LABEL-NET-ELEMENTS` and `VERIFY-LABELS` functionalities. The user should execute the `LABEL_ALL_NET_ELEMENT` command and pick a layer to attach the net element's ids to all entities of that layer. Then, the user should execute the `VERIFY_LABELS` command to check if all entities were correctly labelled. Its Gherkin specification is the following:

```
Given an AutoCAD DWG file with a railML area with entities break
into net elements
Given all net element entities
When the user types the command LABEL_ALL_NET_ELEMENTS
And types the command VERIFY_LABELS
Then all net elements should be painted in green in the imported
railML model
```

Draw missing net elements

The goal of this test is to check the `DRAW-MISSING-NET-ELEMENTS`, implemented by the `DRAW_MISSING_NET_ELEMENTS` command. For this test, we chose the "hjallese.dwg" drawing that lacks entities to represent the net elements "ne3" and "ne6". The user should execute the test's command and verify if the missing entities are added to the drawing. Its Gherkin specification is the following:

```
Given the AutoCAD dwg file "hjallese.dwg"
And the railML area "hjallese.railml"
And all drawed entities are labelled
When the user types the command DRAW_MISSING_NET_ELEMENTS
Then two LWPolylines should be added to the drawing
When the user enters the command VERIFY_LABELS
Then the net elements "ne3" and "ne6" should be painted in green in
the imported railML model
```

Get physical coordinates of the drawing

In this test, that tests the `EXPORT-RAILML` functionality, the user should firstly move all non-endpoint blocks (signalIS, trainDetectionElements, …) to their correct positions. Then the user should execute the `EXPORT_RAILML` command that should update all railML blocks' `POS` attribute. The railML file that was imported into the drawing should also be updated. Thus, the net elements should contain a length attribute and the functional infrastructure elements should contain a pos attribute. Its Gherkin specification is the following:

```
Given an AutoCAD DWG file with a railML area
And all net element entities of the drawing are labelled
When the user moves all non-endpoint blocks to their correct
positions
And types the command EXPORT_RAILML
Then all railML blocks should have their POS attribute updated
And all net elements in the railML area file should have a length
attribute
```

```
And all infrastructure elements in the railML area file should have
a pos attribute
```

Import violations

This scenario tests the `IMPORT-VIOLATIONS` functionality, and requires that a violation file was previously generated for the railML area to test. Moreover, at least one of the violations reported should be committed by one or more functional infrastructure elements. The user should execute the `IMPORT_VIOLATIONS` command and pick the violations file. Then, all railML blocks representing the functional infrastructure elements that committed a violation should have their `ERROR` attribute updated in the drawing. Its Gherkin specification is the following:

```
Given an AutoCAD DWG file with a railML area
And all railML blocks contain a valid POS attribute
And a violations file with a violation containing a railML block
that do not follow a rule
When the user types the command IMPORT_VIOLATIONS
And selects the violations file
Then the violation message should be presented in the ERROR
attribute of the railML block
```

Design Verification Tool

In this section, we present the tests made to check the correctness of the EVEREST Design Verification Tool, taking into consideration the functionalities defined in document T2.6. This tool also uses the railML and the validation libraries to perform its features.

Create a new Project

This test aims to verify the correctness of a project creation. The user should enter a project name and choose a directory for the project's location and click the Create button. Then, the picked directory should contain a file with the entered project name and an `.eproj` extension. Moreover, the user should be redirected to the EVEREST tool's home page. Its Gherkin specification is the following:

```
When the user provides a project name
And picks the project directory
And clicks the Create button
Then the project file should be created in the provided directory
And the user should be redirected to the project home page
```

Add railML area

The user should use the project menu and select the "Add railML area" option to pick a railML file. The tool should use the railML library to read the selected file and display the railML area in the Topology Visualizer, as defined by the `DRAW-TOPOLOGY` functionality. Its Gherkin specification is the following:

```
Given an open project
And a railML file
When the user presses the Add railML button in the project menu
And picks the railML file
Then the railML area should be added to the project
And the railML topology should be visible in the Topology Visualizer page
```

Visualise railML area

This test requires a project with two or more railML areas. When the user is in the Topology Visualizer page, all railML areas should be listed in the topology tree. Then, when the user selects one of those areas in the tree, the selected area should be displayed in the Topology Visualizer canvas, as defined by the `DRAW-TOPOLOGY` functionality. Its Gherkin specification is the following:

```
Given an open project
And a set of railML areas belonging to the project
And the user is in the Topology Visualizer page
When the user selects an area from the topology tree
Then the selected area’s infrastructure elements, routes and tracks should be visible in the canvas
```

Highlight topology element

This test requires a project with one or more railML areas. The user should select one element of the topology tree. Then, the selected element should be highlighted in green in the Topology Visualizer canvas, as specified by the `HIGHLIGHT-SCOPE` functionality. The test’s Gherkin specification is the following:

```
Given an open project with a railML area
When the user selects an element in the topology tree
Then that element should be highlighted in green in the topology canvas
```

Create rule

The goal of this test is to verify the creation of a syntactically and type valid rule. In the Catalogue page, the user should click the “create rule” option to open the create rule form. Then, the user should enter a rule name, a description and enter a valid specification and location. Then, the tool will use the validation library to check the created rule and it should add the rule to the catalogue and set the rule as active in the current project. This exercises both the `CHECK-SYNTAX` and `CHECK-TYPE` functionalities. The test’s Gherkin specification is the following:

```
Given an open project
When the user clicks the + button
```

```
And selects the create rule option
Then create rule form should be prompted
When the user provides the rule name, description, specification,
and its location
And clicks the create button
Then the rule should be added to the catalogue
And the rule should be active in the current project
```

Create rule from pattern

The goal of this test is to verify the creation of a syntactically and type valid rule from a pattern. In the Catalogue page, the user should click the “create rule from pattern” option to open the create rule from a pattern form. Then, the user should pick one of the available patterns, provide a rule name, and fill all the pattern's arguments. Then, the tool will use the validation library to check the created rule and it should add the rule to the catalogue and set the rule as active in the current project. This exercises the `CHECK-SYNTAX`, `CHECK-TYPE` and `INSTANTIATE-PATTERN` functionalities. The test’s Gherkin specification is the following:

```
Given an open project
When the user clicks the + button
And selects the create rule from pattern option
Then create rule from pattern form should be prompted
When the user selects a pattern
And provides the rule name, pattern arguments, and its location
And clicks the create button
Then the rule should be added to the catalogue
And the rule should be active in the current project
```

Create group

The goal of this test is to verify the creation of a rule group. In the Catalogue page, the user should click the “create group” option to open the create group form. Then, the user should provide a group name, and select its location. Then, the tool should add the group to the catalogue. The test’s Gherkin specification is the following:

```
Given an open project
When the user clicks the + button
And selects the create group option
Then create group form should be prompted
When the user provides the group name and its location
And clicks the create button
Then the group should be added to the catalogue
```

Delete rule

This test checks the soft delete of a rule. In the rule page, the user should click the delete button. Then, this rule should not be listed in the catalogue tree. Its Gherkin specification is the following:

```
Given a catalogue rule
When the user clicks the delete rule button
Then the rule should be deleted
And the rule should not be listed in the catalogue tree
```

List deleted rules

This test requires a set of deleted rules. In the catalogue tree, the user should click the filters icon to reveal the filters menu. Then, the user should click the show deleted button to show the list of deleted rules in the catalogue tree. Its Gherkin specification is the following:

```
Given a set of deleted rules
When the user clicks the filters button in the catalogue tree
And selects the show deleted option
Then the deleted rules should be listed in the catalogue tree
```

Restore deleted rule

In this test, the user should select one of the deleted rules in the catalogue tree and click the restore button. Then, the restored rule should be visible in the catalogue tree when the show deleted filter is not active. The test's Gherkin specification is the following:

```
Given a deleted rule
When the user clicks the restore button
Then the rule should be restore
And the rule should be visible in the catalogue tree
```

Edit rule

In this test, the user should click the rule edit icon to open the edit rule form. In this form the user can change the rule name and the rule location to a valid group. After the user clicks the edit button in the form, the changes made to the rule should be visible. The test's Gherkin specification is the following:

```
Given a rule
When the user clicks the edit icon
Then the edit rule form should be prompted
When the user changes the rule's name and its location
Then the rule should be updated
```

Clone rule

The goal of this test is to verify the rule cloning feature. In this test, the user should click the clone button. Then, a new rule should be created with the same versions of the original rule and this new rule should be active in the project. The new rule's name should contain the original rule's name with a suffix "(Copy)". Its Gherkin specification is the following:

```gherkin
Given a rule
When the user clicks the clone button
Then an identical rule should be added to the catalogue with a name
suffix "(Copy)"
And the cloned rule should be active in the project
```

Create rule version

The goal of this test is to verify the creation of a syntactically and type valid rule version. In the selected rule page, the user should click the add version button to open the add rule version form. Next, the user should enter a description and a valid specification. Then, the tool will use the validation library to check the new version and it should add the version to the rule and set the new version as active in the current project. This exercises both the `CHECK-SYNTAX` and `CHECK-TYPE` functionalities. The test's Gherkin specification is the following:

```gherkin
Given a rule
When the user clicks the add version button
Then the add version form should be prompted
When the user provides the version specification and description
Then that version should be added to the rule
And that version should be selected in the project
```

List infrastructure rules

In this test, the user should navigate to the Evaluator tab. In this tab, only infrastructure rules that are active in the project should be listed. The test's Gherkin specification is the following:

```gherkin
Given a set of EVEREST rules
When the user is in the Evaluator tab
Then the project's infrastructure rules should be visible
```

Evaluate rule

This test checks the `EVALUATE-IS-RULE` functionality. It aims to check the eval of a single rule. In this test, the user should open the context menu of an infrastructure rule that is listed in the Evaluator tab and click the eval option. Then, the rule's state should change to valid or invalid. The test's Gherkin specification is the following:

```gherkin
Given an EVEREST infrastructure rule
When the user opens the options menu of that rule
And selects the eval option
Then the rule should be evaluated
And the rule state should change to valid or invalid
```

Evaluate all rules

This test checks the `EVALUATE-IS-RULE` functionality. It aims to check the eval of all active infrastructure rules. In this test, the user should click the eval all button in the Evaluator tab. Then, all rules' states should change to valid or invalid. The test's Gherkin specification is the following:

```
Given a set of EVEREST infrastructure rules
When the user clicks the eval all button
Then all project infrastructure rules should be evaluated
And their states should change to valid or invalid
```

Remove infrastructure rules

In this test, the user should open the context menu of an infrastructure rule that is listed in the Evaluator tab and click the remove option. Then, the rule should not be listed in the Evaluator. The test's Gherkin specification is the following:

```
Given an EVEREST infrastructure rule
When the user opens the options menu of that rule
And selects the remove option
Then the rule should be removed from the project
And the rule should not be visible
```

Update infrastructure rule

In this test, the user should open the context menu of a deprecated infrastructure rule that is listed in the Evaluator tab and click the update option. Then, the rule version state should change to UpToDate. The test's Gherkin specification is the following:

```
Given an EVEREST infrastructure rule with a version status that is
deprecated
When the user opens the options menu of that rule
And selects the update option
Then the rule should be updated to its last version
And the rule version status should change to UpToDate
```

Update infrastructure rules

In this test, the user should click the update all button in the Evaluator tab. Then, all rules' versions states should change to UpToDate. The test's Gherkin specification is the following:

```
Given an EVEREST infrastructure rule with a version status that is
deprecated
When the user clicks the update all button
Then the rules should be updated to their last versions
And the rules version status should change to UpToDate
```

List interlocking rules

In this test, the user should navigate to the Translator tab. In this tab, only interlocking rules that are active in the project should be listed. The test's Gherkin specification is the following:

```
Given a set of EVEREST rules
When the user is in the Translator tab
Then the project's interlocking rules should be visible
```

Translate rule

This test checks the `TRANSLATE-IL-RULE` functionality. It aims to check the translation of a single rule. In this test, the user should open the context menu of an interlocking rule that is listed in the Translator tab, click the export option and pick a directory. Then, the rule's state should change to exported or invalid and the translation file should be present in the picked directory. The test's Gherkin specification is the following:

```
Given an EVEREST interlocking rule
When the user opens the options menu of that rule
And selects the export option
And picks the directory to export the rule
Then the rule should be exported
And the rule state should change to exported
And the translation file should be present in the picked directory
```

Translate all rules

This test checks the `TRANSLATE-IL-RULE` functionality. It aims to check the translation of all active interlocking rules. In this test, the user should click the export all button of the Translator tab and pick a directory. Then, all interlocking rules states should change to exported or invalid and the translations files should be present in the picked directory. The test's Gherkin specification is the following:

```
Given a set of EVEREST interlocking rules
When the user clicks the export all button
And picks the directory to export the rules
Then all project interlocking rules should be exported
And the rules state should change to exported
And the translation files should be present in the picked directory
```

Remove interlocking rules

In this test, the user should open the context menu of an interlocking rule that is listed in the Translator tab and click the remove option. Then, the rule should not be listed in the Translator. The test's Gherkin specification is the following:

```
Given an EVEREST interlocking rule
```

```
When the user opens the options menu of that rule
And selects the remove option
Then the rule should be removed from the project
And the rule should not be visible
```

Update interlocking rule

In this test, the user should open the context menu of a deprecated interlocking rule that is listed in the Translator tab and click the update option. Then, the rule version state should change to UpToDate. The test's Gherkin specification is the following:

```
Given an EVEREST interlocking rule with a version status that is
deprecated
When the user opens the options menu of that rule
And selects the update option
Then the rule should be updated to its last version
And the rule version status should change to UpToDate
```

Update interlocking rules

In this test, the user should click the update all button in the Translator tab. Then, all rules' versions states should change to UpToDate. The test's Gherkin specification is the following:

```
Given an EVEREST interlocking rule with a version status that is
deprecated
When the user clicks the update all button
Then the rules should be updated to their last versions
And the rules version status should change to UpToDate
```

Highlight violation

This test aims to check `SHOW-VIOLATION` functionality. After evaluating a set of rules and at least one rule failing at a set of infrastructure elements the user should navigate to the Topology Visualizer tab. Next, the user should be able to select the generated violation in the topology tree. Then, the route/track and the infrastructure elements that committed the violation should be highlighted in orange in the topology canvas. Its Gherkin specification is the following:

```
Given a project with one or more railML areas
And a list of violations with a set of railML infrastructure
elements
When the user clicks the violation element on the topology tree
Then the route or track related to the violation should be
highlighted in orange
And the infrastructure elements that committed the violation should
be highlighted in orange
```

Open a project with a rule reference no longer in catalogue

Given a project referencing a rule that does not exist in the shared catalogue file, when the user opens that project all other rules should be loaded and the project should be rendered normally. Moreover, a warning message should be displayed indicating the name of the rule that is no longer available in the catalogue. The test's Gherkin specification is the following:

```
Given a project with a rule that is no longer available in the rule
catalogue
When the user opens the project
Then the project is opened with warning message indicating the
unknown rule reference
```

Reload railML area

The goal of this test is to check the updating of railML areas in the EVEREST tool. When the user performs any changes directly on a railML file that is referenced by an open project in EVEREST tool and clicks the reload areas button in the menu, then the area's changes should be visible in the tool. Its Gherkin specification is the following:

```
Given an open a project with a railML area
When the user edits the railML area file
And clicks the reload areas button from the menu
Then the modifications performed are visible in the topology
visualizer
```

Eval a railML area without elements positions

This test checks the `EVALUATE-IS-RULE` functionality. This test requires a project with at least one railML area without positions generated by the EVEREST Plugin. When the user attempts to evaluate all project's rules or one single rule, then a warning message should be prompted indicating the names of the railML areas that do not contain any positions and the evaluation process should proceed normally. The test's Gherkin specification is the following:

```
Given an open project with a railML area without positions and a set
of rules
When the user evals a rule
Then the rule is evaluated
And a warning message is presented to the user indicating the railML
area does not contain any positions
```

Design Verification Tool ↔ AutoCAD EVEREST Plugin

In this section, we present two tests to verify the correctness of the Design Verification Tool and the AutoCAD EVEREST Plugin.

Import generated violations

This test is a combination of the Plugin’s import violations and the evaluation of infrastructure rules. In the EVEREST tool, the user should perform the evaluation of a set of rules. If any of the evaluated rules generated a violation file, then the user should perform the IMPORT_VIOLATIONS command in the CAD drawing that represents the same project. Then, all railML blocks referencing the infrastructure elements that committed the violations should have their ERROR attributes updated with the violations messages. Its Gherkin specification is the following:

```
Given a project with a railML area
And a rule that will fail at one or more infrastructure elements
And a DWG drawing with the same railML area with the complete
signalling process
When the user is in the Evaluator page in the DVT
And clicks the eval all button
Then the rule should be evaluated
And the rule state should change to invalid
When the user clicks the export violations button in the menu
And selects the directory
Then the violations file should be created in the directory
When the user is in the DWG drawing
And types the IMPORT_VIOLATIONS command
And selects the violations file
Then all railML blocks representing the infrastructure elements that
committed the violation should have their ERROR attribute updated
```

Fix violations

This test is a combination of the Plugin’s export command and the evaluation of infrastructure rules. In the CAD drawing, the user should move all railML blocks with errors to their correct positions and execute the EXPORT_RAILML command. Then, in the EVEREST tool the user should update the railML area exported with the reload areas option in the menu. Lastly, in the Evaluator tab the user should perform a new evaluation and the rules should change to valid. The test’s Gherkin specification is the following:

```
Given a project with a railML area
And a invalid rule that failed at one or more infrastructure
elements
And a DWG drawing with the same railML area with the complete
signalling process
When the user is in DWG drawing
And moves the railML blocks to their correct positions
And enters the EXPORT_RAILML command
Then the railML should be updated
When the user is in the DVT
And clicks the refresh railML areas option in the menu
And clicks the eval all button
Then the rule should be evaluated
And the rule state should change to valid
```

Design Verification Tool → RAS EVEREST Plugin