OETCS/WP7/Benchmark_ProR

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

openETCS@ITEA Work Package 7, Task 2: Secondary Tool Chain Benchmark “Management”

Using Eclipse ProR for openETCS

Requirements Management & Engineering and Traceability

Michael Jastram

September 2013

This page is intentionally left blank

openETCS@ITEA Work Package 7, Task 2: Secondary Tool Chain Benchmark "Management"

OETCS/WP7/Benchmark_ProR
September 2013

Using Eclipse ProR for openETCS

Requirements Management & Engineering and Traceability

Michael Jastram

Formal Mind GmbH
Universitätsstr. 1
40225 Düsseldorf, Germany

Email: michael.jastram@formalmind.com

Benchmark Report

Prepared for openETCS@ITEA2 Project

Abstract: This report documents the evaluation of Eclipse ProR, which is a candidate for the secondary tools for data, function and requirement management.

Some of the basic information regarding ProR has been taken from [7]. Further practical reading is available in English [6] and German [8].

This benchmark is based on the case study for the evaluation of the "Secondary tools for data, function and requirement management" [2]. This suggests to use Subset 026-3, § 6 (Management of older Version Systems - several subfunctions initiated inside other functions beside management of version (BL2/ BL3)).

In this evaluation report, only a fraction of this content has been modeled, due to time constraints. However, care has been taken to cover all activities central to the task at hand. Therefore, this report should allow the qualitative, but not quantitative evaluation of ProR. A discussion on performance has been provided in Section 3.8.

Modification History

Version	Section	Modification / Description	Author
0.1	All	Initial Version	Michael Jastram
0.2	1, 2	Bulk of Content	Michael Jastram
0.3	4, 5	Start on Content	Michael Jastram
0.4	All	Final Round	Michael Jastram

Table of Contents

Figures and Tables

Figures

Tables

1 Introduction

This chapter is concerned with evaluating ProR for the use as the requirements management tool in the itea2 openETCS project. ProR is an open source tool, which is part of the Eclipse Requirements Modeling Framework (RMF) [1]. As the underlying data format, ProR uses ReqIF, a standard for exchange of requirements with other tools. It therefore provides interoperability with industry-strenth tools like Rational DOORS or MKS Integrity.

1.1 Objectives

The objective of this report is to evaluate whether ProR is suited to be used for the openETCS management tasks, which include requirements engineering, requirements management and traceability to the various artefacts that will be created as part of the modeling process. The requirements have been captured by WP2 in D2.6-9 [4].

As the case study, Subset 026-3, §6 [2] has been used. Due to resource constraints, this benchmark covers only a fraction of that content, which should nevertheless be sufficient for an evaluation of the tool's feature set.

1.2 ReqIF[1]

The underlying data model of ProR is based on the Requirements Interchange Format (ReqIF) [12]. ReqIF is a file format for the exchange of requirements, standardized by the Object Management Group (OMG). Using it provides interoperability with industry-strength tools, and builds on top of a public standard.

ReqIF allows the structuring of natural language artefacts, supports an arbitrary number of attributes and the creation of attributed links between artefacts. It therefore provides the foundation of collecting and organizing artefacts in a way that users are comfortable with, but provides additional structure for supporting a solid traceability.

ReqIF was created in 2004[2] by the "Herstellerinitiative Software" (HIS[3]), a body of the German automotive industry that oversees vendor-independent collaboration. At the time, the car manufacturers were concerned about the efficient exchange of requirements with their suppliers. Back then, exchange took place either with lo-tech tools (Word, Excel, PDF) or with proprietary tools and their proprietary exchange mechanisms. ReqIF was meant to be an exchange format that would allow the exchange to follow an open standard, even if the tools themselves are proprietary.

1.2.1 The ReqIF Data Model

In general terms, a ReqIF model contains attributed requirements that are connected with attributed links. The requirements can be arbitrarily grouped into document-like constructs.

[1] This section consists in large parts of exerpts from [7].

[2] At the time of its creation, the format was called RIF and only later on renamed into ReqIF.

[3] `http://www.automotive-his.de/`

The most important construct is a *SpecObject*, which represents a requirement. A SpecObject has a number of *AttributeValues*, which hold the actual content of the SpecObject. SpecObjects are organized in *Specifications*, which are hierarchical structures holding *SpecHierarchy* elements. Each SpecHierarchy refers to exactly one SpecObject. This way, the same SpecObject can be referenced from various SpecHierarchies.

SpecObjects can be connected with *SpecRelations*, which are links between SpecObjects. Each SpecRelation contains a source and a target. In addition, a SpecRelation can have a SpecType and therefore AttributeValues.

ReqIF contains a sophisticated data model for *Datatypes*, support for permission management, facilities for grouping data and hooks for tool extensions.

ReqIF is persisted as XML, and therefore represents a tree structure. The top level element is called ReqIF. It is little more than a container for the *ReqIFHeader*, a placeholder for tool-spefic data (*ReqIFToolExtension*) and the actual content (*ReqIFContent*).

These are just a few constructs of of ReqIF, others exist for permission management, advanced structuring and more.

1.2.2 The Impact of ReqIF

Even though ReqIF was initially created as a file-based exchange format, we believe that it can be much more than that. By employing ReqIF directly as the underlying data model for an application, we can take full advantage of the model's versatility. Conveniently, the OMG made the data model available in the CMOF format, thereby facilitating the process of instantiating the data model in a concrete development environment. As we will see in the next section, RMF is based on EMF [13], which can use CMOF as an input.

On the significance on ReqIF and our reference implementation of the standard, we draw comparisons to model-driven software development: After the specification of UML, a lot of publications and work concentrated on this standard, paving the way for low-cost and open source tools. We hope that our open source reference implementation of the standard based on Eclipse can serve as the basis for both innovative conceptual work and new tools.

1.3 The Requirements Modeling Framework (RMF)

RMF is an Eclipse Foundation project that unifies a generic core engine to work with ReqIF content, and a GUI called ProR [1]. The vision or RMF is to have at least one reference implementation of the OMG ReqIF standard in form of an EMF model and some rudimentary tooling to edit these models. The idea is to implement the standard so that it is compatible with Eclipse technologies like GMF, Xpand, Acceleo, Sphinx, etc. and other key technologies like CDO.

1.3.1 High-Level Structure

Figure 1 depicts the high-level architecture of RMF. It consists of an EMF-based implementation of the ReqIF core that supports persistence using the ReqIF XML schema. The core also support the older versions RIF 1.1a and RIF 1.2.

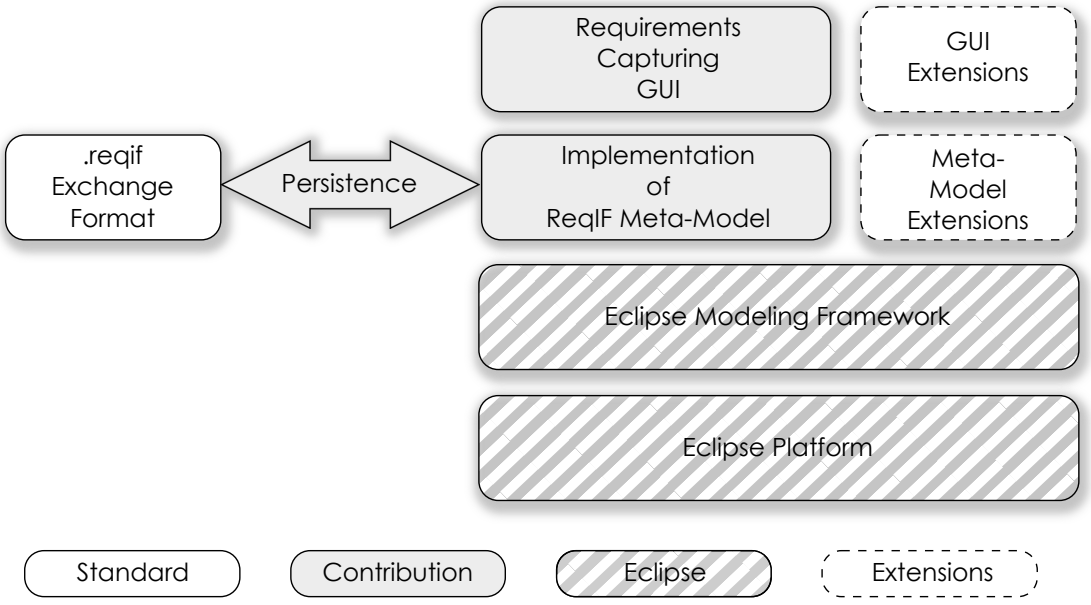

Figure 1. High-level architecture of RMF

The GUI for capturing requirements is called ProR. It operates directly on the ReqIF data model. This is an advantage compared to existing requirements tools, where a transformation between ReqIF and the tool's data model is necessary. Not all tools support all ReqIF features, therefore information may be lost in the process. ProR at this time only supports the current version of ReqIF 1.0.1, not the older versions.

These contributions have their origins in research projects, where they are actively used. In particular, these research projects already produced extensions, demonstrating the value of the platform. But ProR has reached a level of maturity that makes it fit for industrial use. In particular, RMF has been used in the ProSTEP ReqIF Implementor Forum, a forum for tool vendors, to demonstrate interoperability of various requirements tools, including ProR [3].

1.3.2 Extending RMF

RMF is designed as a generic framework for requirements modeling, and the ProR GUI is designed as an extensible application. It has been used and extended in various projects. Particularly relevant for openETCS, the Event-B evaluation used the Rodin extension of ProR to create a traceability of textual requirements to the Event-B benchmark [10].

This is an important aspect of the project. As we have seen in industry, heavy tailoring to the processes used and integration with other tools is what makes requirements tools successful. By using Eclipse as the platform for this tool, we can provide integration with modeling tools like Rodin [6] or Topcased [9]. By providing a versatile extension point, the behavior of the application can be adapted to the process employed. Some specific examples are given in Section 3.4.

1.4 ProR

ProR is the Graphical User Interface (GUI) or RMF. ProR is available as a stand-alone application, and it can be integrated into existing Eclipse installations. A screenshot of the ProR application is shown in Figure 2.

The rest of this document describes how to actively work with ProR, and how to use certain features for the problem at hand. A general overview of ProR will not be provided in this document, but many descriptions of the GUI are available [7, 8].

Further, the RMF website [1] features a 15-minute video, demonstrating the most important features of the tool. For readers not familiar with Eclipse and/or requirements tools, watching the video may be a good preparation for the next chapters.

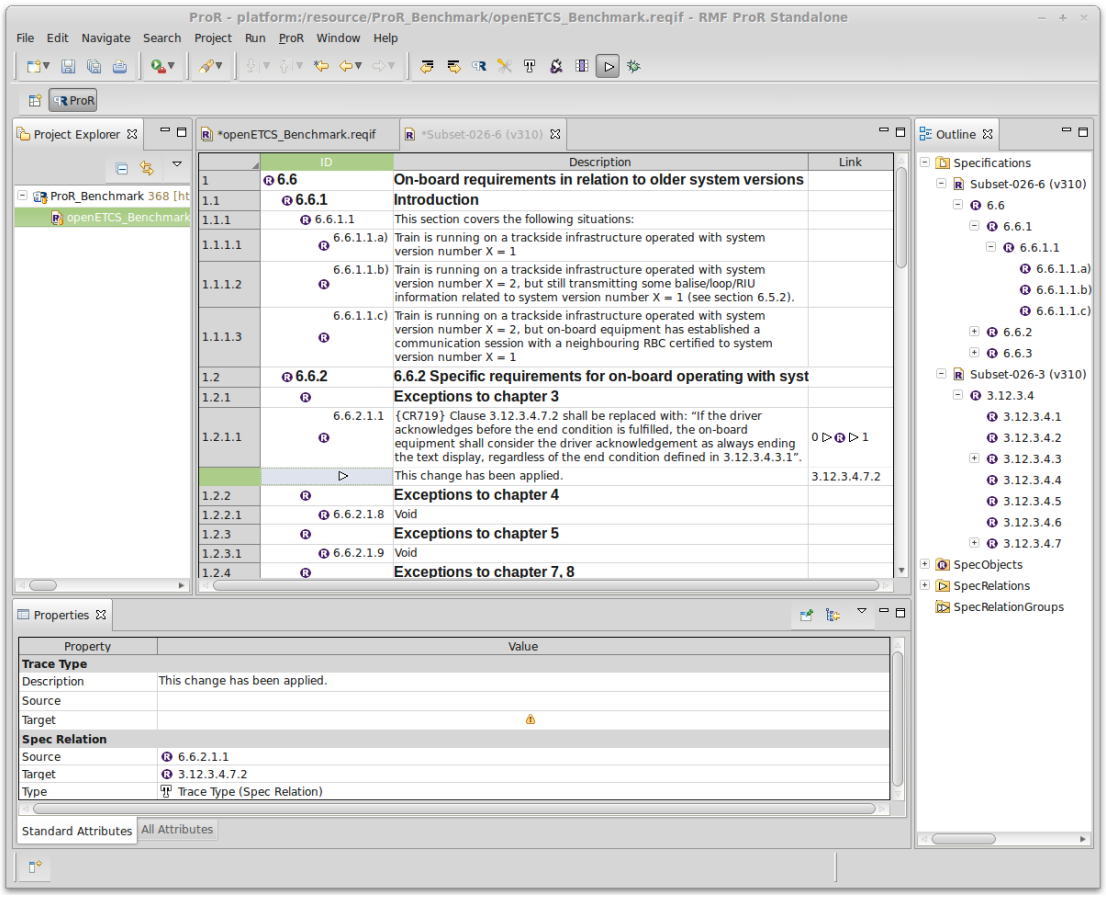

Figure 2. The ProR GUI.

2 ProR Installation and Configuration

This chapter is concerned with the installation and configuration of ProR. The installation is straight forward, but configuring team support requires some attention. This chapter is designed in a tutorial style, resulting in a running version of ProR, with the Benchmark project active and enabled for team work.

2.1 Installing ProR

ProR can be downloaded stand-alone, or installed into an existing application via its update site. The download is a convenient option for non-technical people who just want to get started with ProR. There are no special restriction for the update site version: ProR can be installed into any reasonably new Eclipse installation. However, some care has to be taken in particular regarding compatibility of the modeling tools, if the host installation uses them as well. Follow the following steps to install ProR:

Download ProR. The download URL is `http://eclipse.org/rmf/download.php`. The benchmark has been performed with version 0.8.0. Pick the correct file for your operating system.

Unpack ProR. The downloaded file is a zip file, which needs to be unpacked in a new folder that you need to create.

Launch ProR. In the folder you will find a file named "rmf-pror" (possibly with the .exe extension). You launch ProR by doubleclicking this file. **Tip:** It is convenient to create a shortcut to this file for launching.

Pick a workspace. Upon launching, you will be prompted for a workspace location. The default is as good as any other. **Tip:** If you check "Use this as default" you won't be bothered with this dialog any more.

Dismiss the Welcome Screen. When launched for the first time, a welcome screen is shown.

2.2 Team Support

For openETCS, it is important to be able to work in a team setting. Eclipse already provides a lot of the infrastructure for this. However, an extension to ProR is available that automates a lot of the activities that are necessary in a team setting. More importantly, this extension allow the comparing of requirements via the data model (instead of using XML files). This makes it mutch easier to resolve conflicts, should they arise.

Team support is part of a free suite of extensions called ProR Esssentials. While it is possible to install just individual components of this suite, we recommend to install all of them. Follow the following steps to install ProR:

Select Help | Install new Software... A dialog will open

Set "Work with:". From the dropdown, select the entry ending in "essentials".

Select "ProR Essentials". Once selected, complete the wizard. There will be a warning that the software is not signed.

Restart. The wizard will give you the option to restart, please do it.

Newsletter Subscription. We hope that you will subscribe to the ProR newsletter, but this can be declined.

2.3 Importing the Benchmark

Now that team support is installed, it still has to be configured. Team support uses Subversion. As gitHub allows access via subversion, we can access all projects from gitHub.

Tip: Before proceeding please make sure you have your gitHub password ready. You can proceed without it, but then you only have read acess.

File | Import... This will open a dialog.

Project from SVN. In the folder SVN, select the option "Project from SVN"

Select a Connector. This will trigger a new dialog. There are many ways for giving Eclipse access to Subversion, broadly separated into native connectors (that use your locally installed Subversion) or connectors programmed in pure Java (not relying on any OS resources). We recommend the second, specifically SVN Kit 1.7.10. This will trigger an installation wizard, which you have to follow through and eventually restart ProR.

Restart.

Redo the previous steps. Du to the installation, you have to repeat the previous steps: File | Import... | Project from SVN.

Provide the Project URL. For the Benchmark, we use the Subversion-URL from gitHub, extended by the path to the project, which is `https://github.com/openETCS/model-evaluation/trunk/management/ProR_FormalMind/ProR_Benchmark`. How the dialog is supposed to look is shown in Figure 3. Of course, you have to provide your own credentials.

Complete the Wizard. Complete the wizards with all the given defaults. **Tip:** Providing your gitHub credentials may trigger the secure storage. This is an Eclipse-specific or OS-specific storage which will hold your gitHub password — you can set any password, not (necessarily) the gitHub password.

With this, you have the Benchmark in your workspace, enabled for team work.

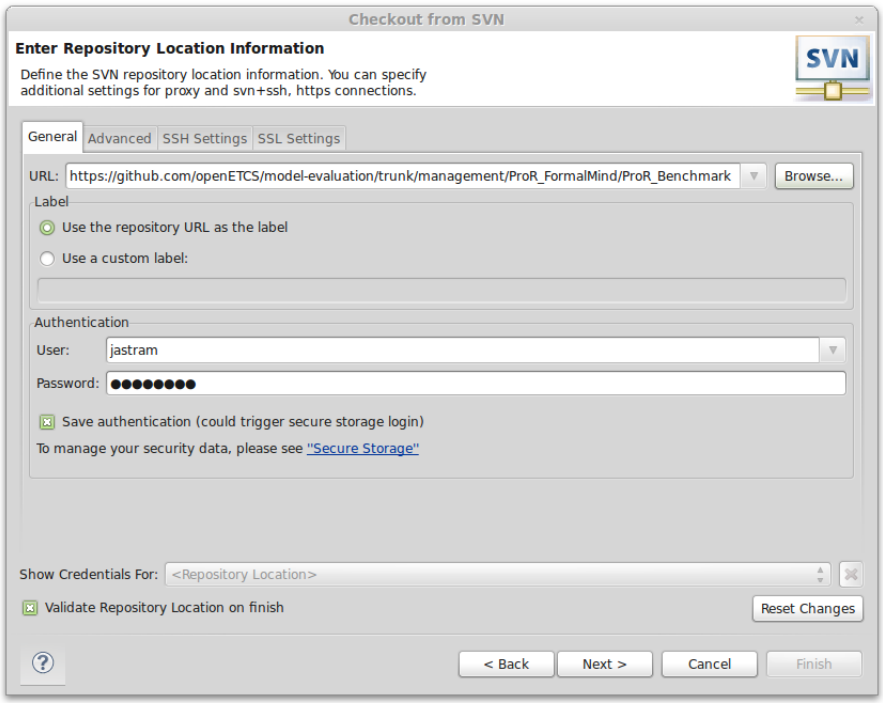

Figure 3. Importing the Benchmark into ProR.

3 Benchmark

This chapter describes the process of modeling the requirements with ProR and discusses various aspects of the modeling process.

3.1 How to Model

There are many ways to skin a cat. This section discusses a number of general questions on how to go about modeling.

3.1.1 Handling IDs

It is best practice to give each element a user-readable ID. Note that each ReqIF element has an internal unique ID that never changes, but that is not meant to be user readable. Considering that most elements already have an ID in the ERA document, we decide to keep those.

A plugin for ProR exists (a so-called Presentation) that automatically generates read-only user-readable IDs. This would result in labels like “REQ-1”. This mechanism should be employed when creating new elements. In this situation, however, they don’t seem to make sense.

3.1.2 Plain Text or Rich Text?

The ReqIF standard, and as a consequence ProR, support rich text (formatted text). While formatted text may seem more attractive (especially, as the ERA documents are available as formatted Word documents), we reject it for §6. The main reason is that formatted text is much harder to process by other tools (e.g. constraining the requirements text with a DSL).

However, to demonstrate that it is possible, we used rich text for modeling sections of §3.

It is possible to mix formatted and plain text. For instance, text that exists for information only could be formatted, while the actual requirements are kept in plain text.

Limited formatting could also be applied to plain text, by giving certain datatypes specific formatting. We chose this solution for §6 and use it to format headlines.

Note: To use rich text, ProR requires at least Java 7 from Oracle (not OpenJDK).

3.1.3 Inconsistencies in the Spec

There were a number of problems with the Word-based specification used. Specifically:

- The document was in change mode, resulting in strange emtpy blocks (e.g. 6.6.2.1.2 – 6.6.2.1.7, followed by an empty table).
- Some headlines did not have a section number, e.g. “Exceptions to chapter 4”. However, the correct section number could not be inferred.

- Some headlines did not have a section number, but allowed the correct number to be inferred, e.g. “General” between 6.6.3 and 6.6.3.1.1.
- However, as all the numbering is generated by Word, it should be considered fragile.

3.1.4 Tables

The document contains a number of tables, and it’s not clear what the best option for modeling them is. ReqIF supports tables, with each table cell being one SpecObject. However, ProR currently does not support the visualization of such tables (they can be created and read, but will not be shown in a table structure).

However, even if this feature were implemented in a user-friendly way, using it would bypass the rich type system that ReqIF provides. Instead, we created a dedicated SpecType for each table row. The description would be used as the label, while there would be table-specific attributes of the proper type (constraint integers, enumerations, etc.). This would also make further automated processing much easier. The downside is, that not all table columns are visible. To see all values in the Properties View, a row must be selected. Of course, the tabular view could be extended to show all columns with a few clicks. But this would result in showing columns that are used in only a few rows.

The worst option would be to use rich text, to put a (XHTML) table in one SpecObject.

Other options may be available. But it is clear that the constrains imposed by the word processor do not have to be transferred to ProR.

Figure 4 shows how this looks in practice. The package action table (after 6.6.3.1.5 in the document) has four columns. Two of these have been mapped to the visual columns *ID* and *Description*. The other two (*Operated system version number X = 1* and *2*) are only visible in the properties view. Not visible in the screenshot is the fact that the ID is restricted to the range 0–255, and that the entries for the last two columns are selected from a dropdown.

Some of the entries have footnotes. These have been modeled as links. The link targets can be seen at the bottom, and the right column shows that select row (Packet Number 51) has two outgoing links. The link targets don’t have to be visible: Links can be expanded, and by selecting an individual link, the properties of the target will be shown in the properties view.

3.2 Rich Text and References

As mentioned earlier, parts of §3 have been modeled using rich text. This is shown in Figure 5, demonstrating some of the rich text capabilities. For instance, bullet lists, bold and strike-through are supported.

The figure also shows that the original text has been modified, with new text inserted and removed text stroked out (this has been done manually, not automatically). This has been done mainly to remove references that have been inserted into the original document as plain text. Doing this is highly fragile: Inserting another paragraph in Word would change the subsequent numbering, leaving an incorrect reference[4]. The reference has been replaced with a SpecRelation. In the

[4]Incidentally, Word does in fact support cross-referencing, which would have solved at least this problem. Why this has not been used for a document like this is beyond me.

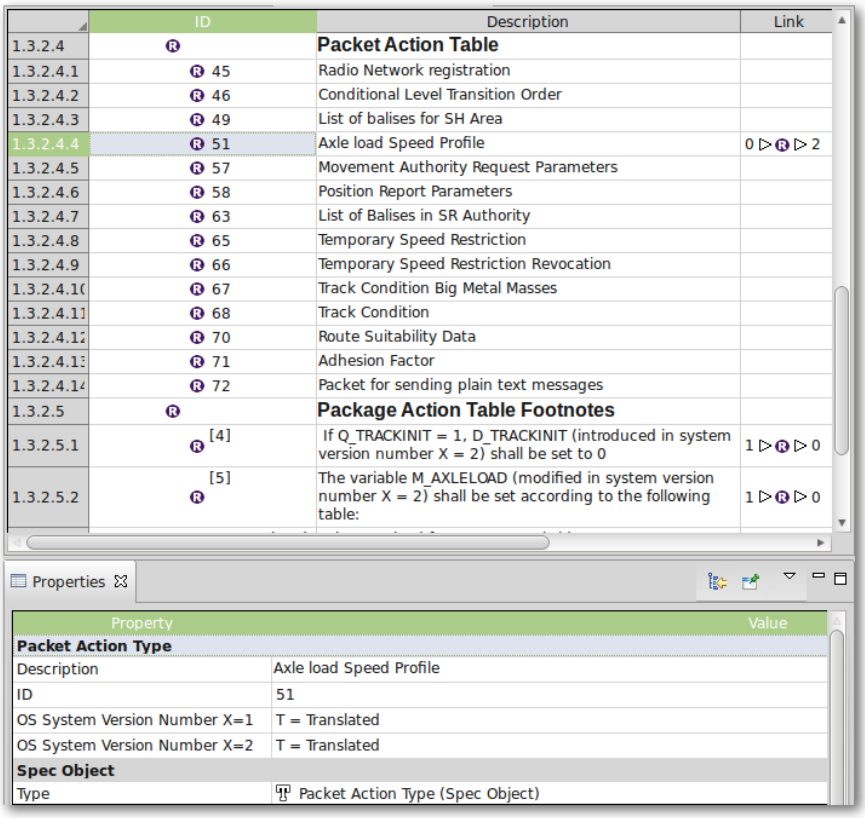

Figure 4. Representing the Package Action Table in ProR, using attributes instead of columns.

	ID	Description	Link
1	R 3.12.3.4	**Conditions for Start/End of Indication**	
1.1	R 3.12.3.4.1	It shall be possible to specify individual events for start/end condition of indication.	
1.2	3.12.3.4.2 R	The following events can be used to define the start condition: • Location • Mode (start display as soon as in mode) • Level (start display as soon as in level)	1 ▷ R ▷ 0
1.3	3.12.3.4.3 R	The following events can be used to define the end condition: • Location • Time • Mode (stop display when leaving mode) • Level (stop display when leaving level)	1 ▷ R ▷ 0
1.3.	3.12.3.4.3.1 R	It shall be possible to define whether one or all of the events used from the **referenced** lists ~~in 3.12.3.4.2/3.12.3.4.3~~ have to be fulfilled to define the start/end condition. This definition shall apply to both start and end conditions.	1 ▷ R ▷ 2
	▷		3.12.3.4.2
	▷		3.12.3.4.3
1.3.	3.12.3.4.3.2 R	In case a confirmation of the text message is requested, it shall be possible to define whether the driver acknowledgement is considered: a) As always ending the text display, regardless of the end condition defined in **the linked requirement** ~~3.12.3.4.3.1~~	0 ▷ R ▷ 1

Figure 5. Using rich text and SpecRelations to provide references.

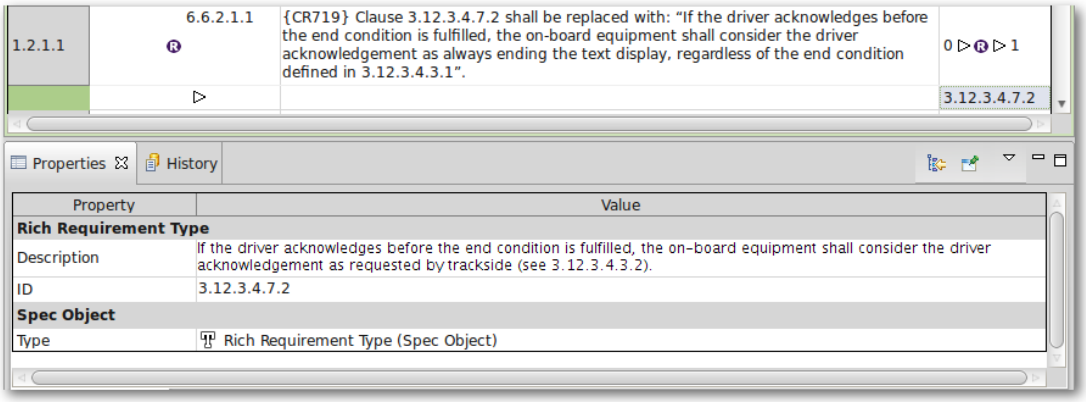

Figure 6. Showing the link target in the Properties View.

figure, the two references to 3.12.3.4.2 and .3 have been expanded. The description text of those links is empty, but by giving the SpecRelation attributes, some information could be entered here.

3.3 Traceability

Traceability is a crucial feature for openETCS. Traces in ProR are modeled using SpecRelations. These have already been put to use for linking footnotes, as described in Section 3.1.4. However, that use does not show the full potential of traces.

For this case study, we show how traceability between §6 and §3 can be realized. The following requirement will be used:

6.6.2.1.1	{CR719} Clause 3.12.3.4.7.2 shall be replaced with: "If the driver acknowledges before the end condition is fulfilled, the on-board equipment shall consider the driver acknowledgement as always ending the text display, regardless of the end condition defined in 3.12.3.4.3.1".

In contrast to the SpecRelations used earlier, a typed SpecRelation will be used this time to:

- Be able to annotate the trace with a description.
- To track whether the source or target of the SpecRelation has changed (for change management).

Change management can be realized with the "Linkmanagement Presentation", an extension from the ProR Essentials. To use it, a new SpecType for SpecRelatons has to be created with two boolean flags (for source and target, respectively). this is done via *ProR | Datatype Configuration...* Next, the presentation has to be added (via *ProR | Presentation Configuration...*).

Once this is set up, a link between the two elements (6.6.2.1.1 and 3.12.3.4.7.2) can be established. There are various ways to acomplish this. The easiest is to use the element's context menu to select *Initiate Linking*, and then to select the target element, selecting from the context menu, using the SpecType we previously created.

As a result, element 6.6.2.1.1 has now an outgoing SpecRelation, which can be expanded. Selecting the SpecRelation's row will show the attributes of that SpecRelation. However, selecting

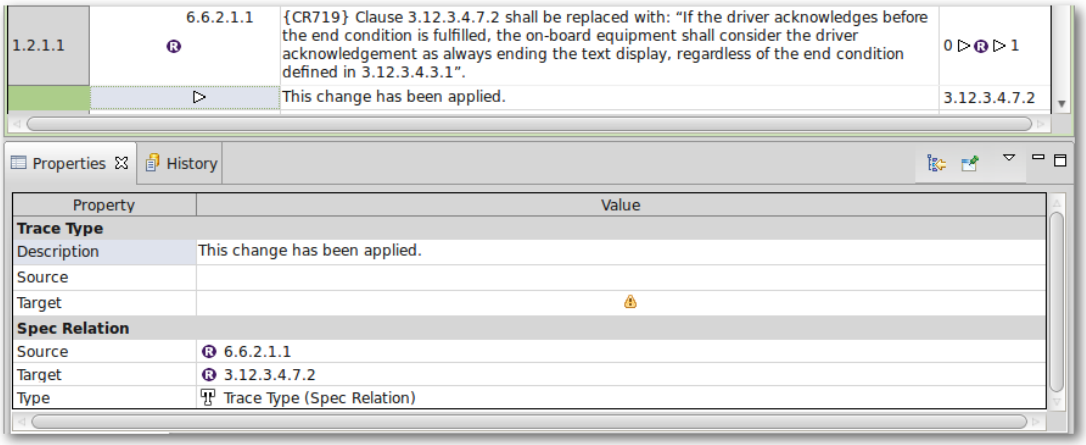

Figure 7. Showing the link target in the Properties View.

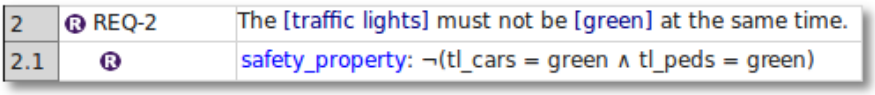

Figure 8. Showing how models and requirements can be integrated seamlessly. The first row shows color highlighting in the requirements text. The colored symbols are defined in the associated Event-B model. The second row shows an element from the Event-B model, which is referenced and therefore never outdated.

the last cell in that row (the *Link* column), the target element will be shown in the Properties View, as shown in Figure 6.

Now the change can be applied by updating element 3.12.3.4.7.2, which can be done directly in the Properties View. For good measure, some information on what has been done can be written in the description attribute of the SpecRelation. Figure 7 shows this, but this time with the SpecRelation selected. Note that the SpecRelation's description text is shown both in the main editor, as well as in the Properties View. Also note that the "Target" Attribute in the Properties View now shows a yellow warning triangle. This has been set by the Linkmanagement Presentation: It noticed that the target element had changed and set the flag correspondingly. Users can reset it by doublecklicking it, indicating that they verified the trace. The opposite is possible too: traces can be marked manually for inspection.

3.4 Traceability to EMF-based Models

Integrating ProR with other EMF tools is out of the scope of this document. As mentioned before, some of this has already been explored in the Event-B benchmark [10], e.g. in Figure 13 in that report.

Integration with existing EMF models can take place on two levels, both which are shown in Figure 8, which has been taken from [5]:

Linking to model elements. ProR supports a mechanism to create proxy elements that retrieve the content of other model elements and renders them in an appropriate fashion directly in the specification.

Color highlighting of model element names. It is also possible to use color highlighting in the requirements text to identify symbols that are defined in another model. Currently this has

only been realized using plain text attributes, as interfering with the formatting of rich text attributes is tricky.

3.5 Traceability to Other Artefacts

It is conceivable that traceability to other artefacts may become necessary, like test cases, test reports, source code, or other not-EMF-based models. ProR is capable to provide such traceability in a modular manner, but additional implementation effort is required for this.

If the artefact to be traced is available as a file in the same folder (i.e. the same Eclipse project), then adapter can be developed that monitors that file and reacts to changes. ProR provides an extension point that allows the seamless integration into the GUI, using the same mechanism described in Section 3.4. Such an integration would probably require more work, as the plug-in would have to provide a parser (while an EMF-based model is already parsed). It would probably be slower as well, as the whole file has to be monitored not just the model element in question. Nevertheless, all this can be neatly packaged in a separate plug-in, thereby guaranteeing maintainability.

It is also possible to monitor resources other than files (e.g. web services). If this is of interest, it may be useful to look into existing frameworks like OSLC.

To give a practical example, it would be possible to provide an adapter for JUnit unit tests (which stores the test results as an XML file). A requirement could reference a test (using a SpecRelation). The result could then be represented as a colored String attribute which would be green if all is ok, and showing the error condition in red, if there is a problem.

3.6 Reporting

Reporting capabilities are currently limited to converting a Specification to HTML. This is realized by selecting the editor containing the Specification to be exported and to initiate *File | Print...*. ProR will open a Web Browser with the exported Specification.

A Master student at the University of Düsseldorf is implementing advanced reporting as part of his thesis. This work is expected to be available in the open source by the end of the year.

3.7 Using Team Support

The team support from the ProR Essentials makes the most often used team actions available via the project's context menu (in the *ProR* submenu). The same, and more advanced actions are in the *Team* context menu of the project.

The team support offers more features, which are described as follows. However, these features seem to be broken in 0.8.0. We plan to fix them with the next version of ProR Essentials, expected in October 2013.

Team support can be used to browse the history of the project or individual files (*Team | Show History*). Differences between files can be inspected by selecting two files and picking *Compare with Each Other* from the context menu (currently broken). Rather than comparing the XML files, the two model trees will be compared, as shown in Figure 9. This feature is described in detail in [11]. A corresponding model viewer can be used to perform merges in the case of conflicts.

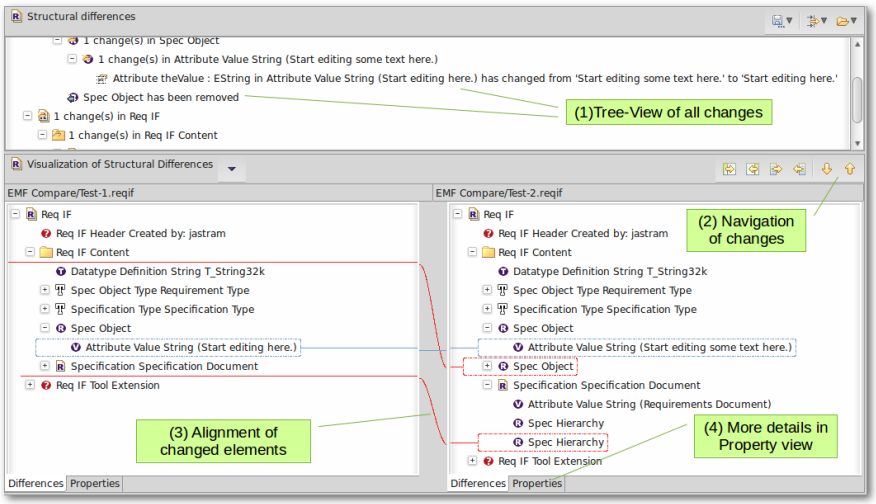

Figure 9. Comparing Models with ProR Essentials Diff. Rather than having a textual diff of the XML, the tool compares the model representation of the two versions. The structural differences are shown in a tree (1), while the actual changes can be compared graphically next to each other (3), with the ability to look at the actual properties (4). Step by step navigation through the changes is possible (2).

In addition to this, the team extension provides a number of convenience features (currently broken):

Auto-Add. New files in the project will automatically be added to version control. Currently, this has to be done by hand (can be done as part of the commit process).

Auto-Update. Users can elect to automatically run a subversion update upon opening a project. This ensures that they will always have the latest version of the requirements, minimizing the risk of conflicts.

Auto-Commit. Users can elect to automatically run a subversion commit upon closing the project. This will ensure that changes are pushed to the server as soon as possible, minimizing the risk of conflicts.

3.8 Performance

Due to time constraints, only a small number of requirements have been modeled. Therefore, this benchmark is not adequate to evaluate the performance of the tool.

However, ProR has been shown to handle thousands of requirements easily, which should be sufficient for fulfilling the needs of the openETCS project.

Performance is a crucial issue in the car industry, where ProR is already deployed. Here, users often have to deal with tens of thousands of requirements, and the goal of the project is to support that with an acceptable performance. Therefore, performance is expected to improve over time even more.

Last, a progress bar is shown upon opening a file, giving the user an indication on how long this will take. Once open, the performance is usually fast. Further, the progress bar slows the opening of a file down for small files, but not for big ones (this is a bug). Thus, the speed of opening a

small file should not be taken as an indication on how much longer it would take to open a big one.

3.9 Issues to be Resolved

We believe that this benchmark demonstrates that many of the features required for requirements management, engineering and traceability are found in ProR. Nevertheless, a few issues that need to be resolve have been identified:

Open Source. ProR is open source, and licenced under EPL, which is EUPL compatible. However, the components from the ProR Essentials are closed source, belonging to Formal Mind. If ProR is adopted for openETCS, we will open source those components that are used by openETCS.

Traceability to ERA documents. The ERA documents are currently available as Microsoft Word documents. We recommend the development of an importer that will allow to reimport those word documents without losing traceability.

Handling Tables. The ReqIF standard supports a scheme for managing tabular data, where each cell in the table is an atomic element. ProR supports this, but the representation is not user friendly. We recommend implementing a better representation for ReqIF tables.

Team Integration. The current implementation hints at what is possible, but some features that were implemented before are currently broken. These will be fixed in the next update, expected in October 2013.

4 Conclusion

We believe that we have demonstrated that professional requirements management and traceability is possible with ProR, and that traceability in EMF-based models can be realized with reasonable effort. As important ProR is a tool that fulfills a number of requirements that are essential for openETCS, including being open source and being based on Eclipse and EMF.

This report covers essential features for openETCS, including:

- Version control
- Atomic requirements with arbitrary typed attributes
- Plain text and rich text requirements
- Organization of multiple documents
- Traceability with attributed traces
- Change Management
- Integration with other EMF-based tools

Last, as ProR is based on the well-documented international ReqIF standard, integration with other tools is possible and relatively easy.

References

[1] Eclipse Requirements Modeling Framework. http://eclipse.org/rmf.

[2] Model Evalution Wiki: Secondary Tools for Data, Function and Requirements Management. https://github.com/openETCS/model-evaluation/wiki/Secondary-tools-for-data,-function-and-requirement-management.

[3] ProSTEP ReqIF Implementor Forum. http://www.prostep.org/en/projects/internationalization-of-the-requirements-interchange-format-intrif.html, 2013.

[4] Sylvain Baro and Jan Welte. D2.6-9: Requirements for openETCS. Technical report, openETCS. https://github.com/openETCS/requirements/blob/master/D2.6-9/D2_6-9.pdf?raw=true.

[5] Michael Jastram. Using RMF to integrate your models. http://www.formalmind.com/en/blog/using-rmf-integrate-your-models.

[6] Michael Jastram. A Systems Engineering Tool Chain Based on Eclipse and Rodin. In *Forms/Format*, 2012. http://www.stups.uni-duesseldorf.de/w/Special:Publication/jastram_forms_2012.

[7] Michael Jastram. *Managing Requirements Knowledge*, chapter 16: The Eclipse Requirements Modeling Framework, pages 353–372. Springer, March 2013. http://www.stups.uni-duesseldorf.de/w/Special:Publication/RMF_Mark_Book_Jastram_2013.

[8] Michael Jastram. ReqIF-OLUTION: Mit Eclipse und ReqIF zur Open-Source ALM-Werkzeugkette. *ObjektSpektrum*, 3, 2013. Please contact author for a PDF copy.

[9] Michael Jastram and Andreas Graf. Requirement traceability in Topcased with the requirements interchange format (RIF/ReqIF). *First Topcased Days Toulouse*, 2011. http://www.stups.uni-duesseldorf.de/w/Special:Publication/topcase-JaGr2011.

[10] Matthias Güdemann. openETCS Event-B Benchmark. https://github.com/openETCS/model-evaluation/blob/master/model/Event_B_Systerel/rodin-projects-github.pdf?raw=true, 2013.

[11] Michael Jastram. Comparing ReqIF Files with ProR Essentials Diff. http://www.formalmind.com/de/blog/comparing-reqif-files-pror-essentials-diff, 2013.

[12] OMG. Requirements Interchange Format (ReqIF) 1.0.1. http://www.omg.org/spec/ReqIF/, 2011.

[13] D. Steinberg, F. Budinsky, M. Peternostro, and E. Merks. *EMF Eclipse Modeling Framework*. Addison-Wesley, second edition, 2009.