OETCS/WP7/Benchmark_ProR

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

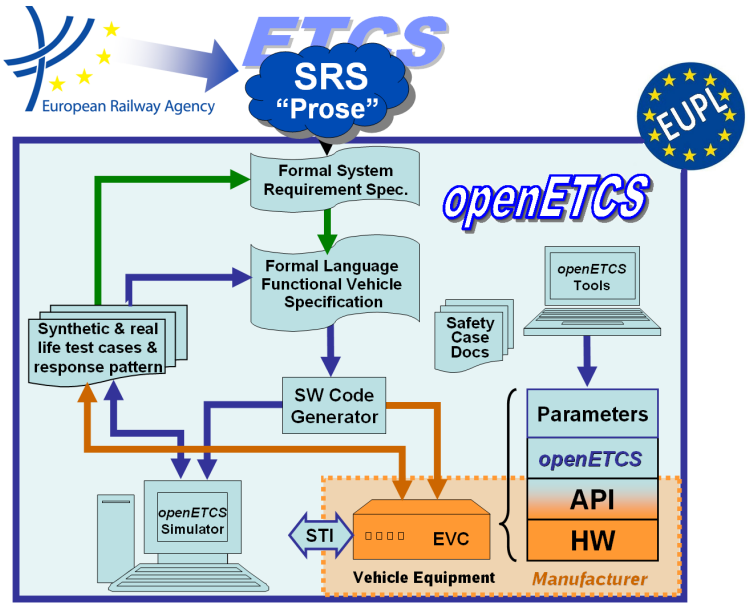
openETCS@ITEA Work Package 7, Task 2: Secondary Tool Chain Benchmark “Management”

Using Eclipse ProR for openETCS

Requirements Management, Engineering and Traceability

Michael Jastram

September 2013



Funded by:

Federal Ministry of Education and Research
Région de Bruxelles-Capitale
GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

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.

Much of the basic information regarding ProR has been taken from [4]. Further practical reading is available in English [3] and German [5].

Modification History

Version	Section	Modification / Description	Author
0.1	All	Initial Version	Michael Jastram
0.2	1, 2	Bulk of Content	Michael Jastram
0.2	4, 5	Start on Content	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 D2.3.

1.2 ReqIF[1]

The underlying data model of ProR is based on the Requirements Interchange Format (ReqIF) [8]. 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.

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.

[1] This section consists in large parts of exerpts from [4].
[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/`

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 [9], which can use CMOF as an input.

On the significance on ReqIF and our first-clean room 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. 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 **??** 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.

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

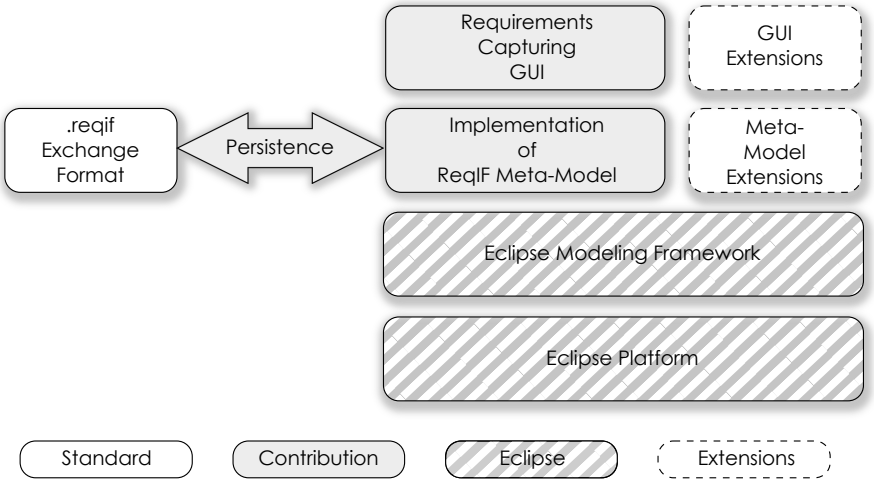


Figure 1. High-level architecture of RMF

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 [2].

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 [7].

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 [3] or Topcased [6]. By providing a versatile extension point, the behavior of the application can be adapted to the process employed.

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. Rather than describing it here, a quick tour of the tool will be provided in Chapter 3.

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.

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 **??**. 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.

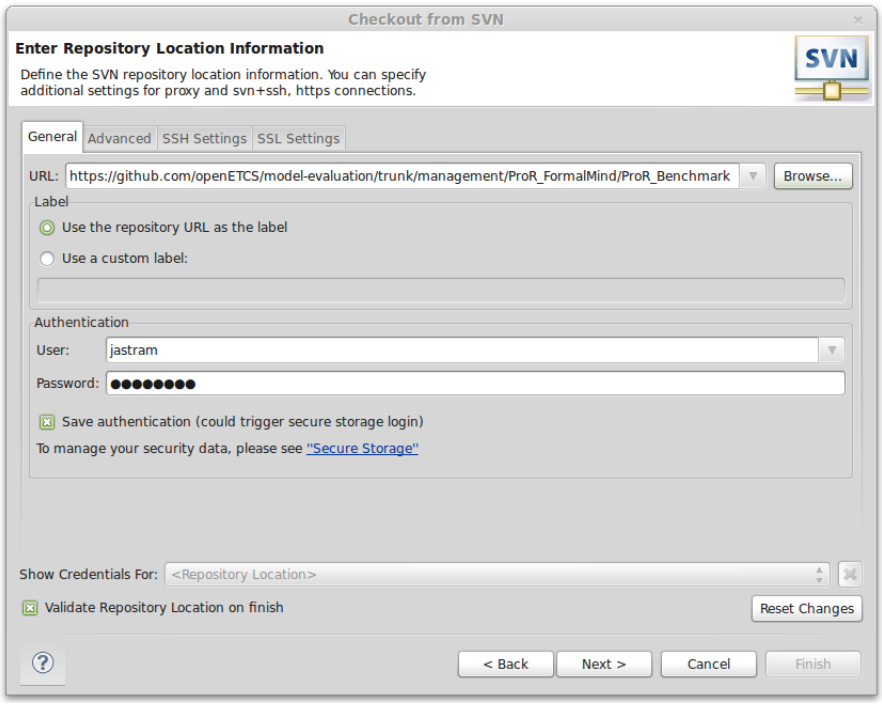


Figure 2. Importing the Benchmark into ProR.

3 A Quick Tour of ProR

3.1 Importing Project from gitHub

4 Benchmark

4.1 How to Model

4.1.1 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.

4.1.2 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 (esecially, as the ERA documents are available as formatted Word documents), we try to avoid it for now. The main reason is that formatted text is much harder to process by other tools (e.g. constraining the requirements text with a DSL).

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 and use it to format headlines.

4.1.3 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.

4.1.4 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 **??** 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.

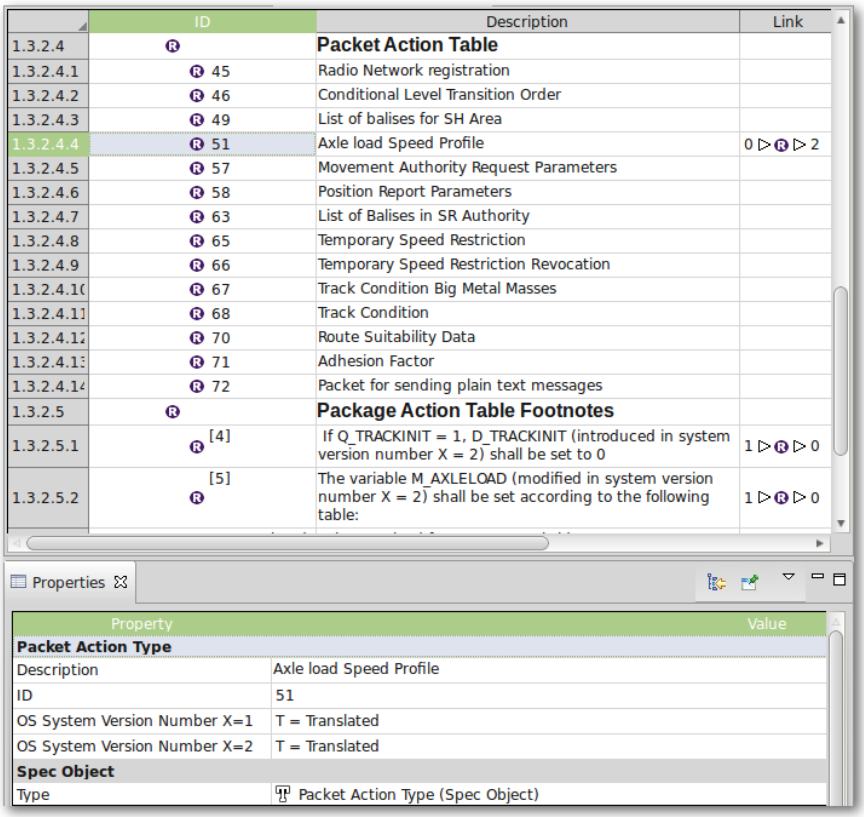


Figure 3. Representing the Package Action Table in ProR, using attributes instead of columns.

5 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.

Having said that, there are a number of issues that need to be addressed eventually:

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. We plan work on improving the diff and merge featues, as well as auto-update on open and auto-commit on close.

References

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

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

[3] 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.

[4] Michael Jastram. *Managing Requirements Knowledge*, chapter 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.

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

[6] 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.

[7] 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.

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

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