16 The Eclipse Requirements Modeling Framework

M. Jastram

Abstract: This chapter presents the the Requirements Modeling Framework (RMF), an Eclipse-based open source platform for requirements engineering. The core of RMF is based on the emerging Requirements Interchange Format (ReqIF), which is an OMG standard. The project uses ReqIF as the central data model. At the time of this writing, RMF was the only open source implementation of the ReqIF data model.
By being based on an open standard that is currently gaining industry support, RMF can act as an interface to existing requirements management tools. Further, by based on the Eclipse platform, integration with existing Eclipse-based offerings is possible.
In this chapter, we will describe the architecture of the RMF project, as well as the underlying ReqIF standard. Further, we give an overview of the GUI, which is called ProR. A key strength of RMF and ProR is the extensibility, and we present the integration ProR with Rodin, which allows traceability between natural language requirements and Event-B formal models.

16.1 Introduction

The Requirements Modeling Framework (RMF[1]) may be of relevance to the reader for a number of reasons. First and foremost, RMF extends the Eclipse ecosystem with a meta-model for modeling requirements, the Requirements Interchange Format (ReqIF). We hope that ReqIF will have a similar effect on requirements engineering to what UML did for modeling: providing a unified data model that tools could converge on. By being open source, RMF contributes to spreading the use of ReqIF.

Second, RMF contains the stand-alone ProR application, a platform for requirements engineering. This tool uses ReqIF as the underlying data model and therefore offers sophisticated, standardized data structures for organizing requirements and provides interoperability with industry tools. Especially in small companies or academic projects, users until now faced the dilemma: Tools like Word and Excel have wide acceptance, but limited features for requirements engineering. Professional tools like Rational DOORS[2] or IRQA[3] are not affordable. There are some free tools, like Trend/Analyst[4], Topcased [9] or Wikis[5]. But these either use their own data structures,

[1] http://eclipse.org/rmf
[2] http://www.ibm.com/software/awdtools/doors/
[3] http://www.visuresolutions.com/irqa-requirements-tool
[4] http://www.gebit.de/loesungen/technische-loesungen/trend-analyst-requirements.html
[5] http://www.mediawiki.org/

with their own limitations, or follow a standard with few features in respect to requirements, like SysML. ProR provides a lot of functionality out of the box, and offers interoperability according to an international standard. The interest that ProR created both in academia and industry confirms this.

Third, ProR can easily extended to provide additional functionality or for integration with other tools. ProR is implemented as Eclipse plug-ins. While it can run standalone, it can be installed in any existing Eclipse system. No special project type is required for ProR, therefore any Eclipse project can contain ProR requirements files. ProR provides an extension point, which allows developers to build tool integrations. For instance, we created an integration plug-in for Rodin, a tool for formal modeling [2]. After installing ProR into Rodin, it was possible to use drag and drop to integrate model elements into the requirements specification.

Forth, RMF can be used as a generic platform for working with ReqIF-based requirements, independently of an Eclipse-based GUI. This can be useful for a wide range of activities, from analysis, report generation, generation of requirements artefacts, product line management, etc. RMF is built on the Eclipse Modeling Framework (EMF), which supports the integration with other EMF-based offerings.

Before discussing the technical details, we will provide an overview of the current state of requirements modeling, both in industry and academia.

This chapter is structured as follows: Section 16.2 provides an overview of the current state of requirements modeling. Section 16.3 describes the data model of the ReqIF format. Section 16.4 introduces the architecture of the Requirements Modeling Framework (RMF), followed in Section 16.5 by a description and tutorial of ProR, the user interface. Section 16.6 demonstrates how the platform can be extended and integrated with other Eclipse-based software. This chapter concludes with Section 16.7.

16.2 Requirements Modeling

While this chapter is concerned with a software platform, it is useful to put it into the bigger context of requirements management and engineering, which we will do in this section. Ultimately, a tool is only useful if used properly, and with a clear goal.

A tool must support the activities found in requirements engineering and requirements management. These include the structuring of requirements, establishing traceability, handling versions, integrating with other processes (e.g. testing and project management), to name just a few. The specific activities depend to a degree on the process that the tool has to support.

After discussing requirements and specifications in general below, we will revisit the topic of tool support in Section 16.2.5.

16.2.1 Specifying Systems

Everything is build twice: First an idea forms in the mind, then the idea is realized. This is true from the smallest to the biggest projects, from hanging up a picture to building a space rocket. In the case of the space rocket, there would be a number of intermediate steps to account for the complexity and scale of the task at hand. The number of

intermediate steps and types of documentation depends on the size of the project, how critical it is, how many people are involved and many other factors. Nevertheless, requirements and specification are artefacts that are so important that they play part in all but the smallest projects.

Every project should have a *goal*. A goal typically says nothing about the "how" ("How do I achieve this?"), but the "what" ("What is it that I want to achieve?"). A goal is typically very simple and high-level. This does not necessarily mean that it is not precise or quantifiable.

A *requirement* puts the goal into the context of the world. A requirement for hanging a picture on a wall is that it stays there, which in turn has to take the picture's properties into account. This does not mean that it should indicate *how* the picture is mounted — a nail or two screws — because a good requirement does not provide a solution, but precisely describes the problem.

For big projects it is not practical to go directly from goal to requirements. The goal is typically broken down in subgoals, an overall architecture is established that allows partitioning of the tasks at hand, etc. In addition, there is a lot of overhead that does not directly contribute to the development, but that is crucial nevertheless. This includes artefacts for sub-disciplines like project management, testing, and many other areas of interest.

It is the *specification's* job to provide a solution to the problem. This is the place that describes that a nail shall be used to put up the photo, and where to put it. It is dangerous to look for solutions sooner than at this point, because it is easy to miss important requirements or something crucial regarding the context.

16.2.2 Structuring Requirements

A good structure of requirements can make a huge difference in their management and traceability, and quite a bit of research went into understanding this relationship better. In industrial environments, this manifests itself in standards like IEEE 830-1997 [7] or the relevant aspects of process frameworks like RUP [14].

In academia, Gunter, Jackson and Zave [6] developed WRSPM as a *reference model* for requirements and specifications. A reference model is attractive for discussion, as it draws on what is already understood about requirements and specifications, while being general enough to be flexible. There are a number of concrete approaches that fit nicely into the WRSPM reference model, including Problem Frames [8] , KAOS [3] or the functional-documentation model [16].

WRSPM distinguishes five artefacts:

Domain Knowledge (*W*) describes how the world is expected to behave.

Requirements (*R*) describe how the world should behave.

Specifications (*S*) bridge the world and the system.

Program (P) provides an implementation of S.

Programming Platform (M) provides an execution environment for P.

Inexperienced users sometimes confuse requirements and domain knowledge, but the distinction is quite important:

- Requirements describe how the world *should behave*, and the system is responsible for this.
- Domain knowledge describes how the world is *expected to behave*, and the functioning of the system depends on the domain knowledge holding.

The relationship between requirements, domain knowledge and the specification can be expressed formally:

$$S \wedge W \Rightarrow R$$

Or in words: Assuming a system that conforms to the specification S, and assuming that the domain properties W hold, the requirements R are realized. There are some subtleties (e.g. we are probably not interested in the trivial solution), but this is a central idea of WRSPM.

Note that WRSPM does not know the concept of a goal. But according to WRSPM, a goal is merely a high-level requirement. Also note that there is a whole category of approaches called goal-oriented requirements engineering (GORE) [19].

The reference model defines *phenomena*, which act as the vocabulary to formulate the artefacts. There are different types of phenomena based on their visibility. For instance, there may be phenomena that the machine is not aware of. Consider a thermostat: the controller is not aware of the temperature[6], but only of the voltage at one of its inputs.

The reference model can be applied to any requirements or specifications, no matter whether they use natural language or a formalism. Once applied, more formal reasoning about the specification is possible.

Informal requirements rarely explicitly distinguish between requirements, domain knowledge, or even specification elements and implementation details. In the following, *artefacts* will refer to all of them.

16.2.3 Informal and Formal Specifications

Artefacts can be formalized by modeling them using a formalism. Many formalisms exist, all with their respective advantages and disadvantages. Modeling can also be applied on various levels of the development process — for goals, requirements, the specification and even for the implementation.

---

[6] To be precise, whether the controller is aware of the temperature or not depends on where the line is drawn between system and environment. In this simple example, the sensor is not part of the system (the controller).

Some formalisms are more, others less “formal”. Often, a formalism only models a certain aspect of the specification and has to be complemented with additional information. Here are a few examples:

Context Diagrams only formalize a small aspect of a system, it’s boundary to the world. They help in the requirements elicitation process, by forcing us to define the boundary of the system and to identify the actors that can interact with it. Using a context diagram in the elicitation process will leave its traces in the structure of the requirements (i.e. by systematically enumerating all actors and how they interact with the system). They are formal only in the sense that they allow reasoning about a tiny aspect of the system and need to be complemented with much more information.

UML and SysML Diagrams provide modeling elements for many elements of the system and their relationship, ranging from class diagrams for object relationships to state diagrams for transitions. While they are useful, they are not formal enough to express complex functionalities and must be complemented somehow, for example by use cases.

Problem Frames [8] introduce *problem diagrams*, which extend the notation of context diagrams and make the problem explicit by showing the requirements in the diagram. The notation of context diagrams is also formalized by distinguishing between machine domain, designed domains and given domains. The notation further introduces *problem frame diagrams* for concisely recording problem frames.

Z, VDM, B and many others are a particular kind of mathematically-based techniques for the specification of *sequential* behavior. These and similar notations are used to specify and verify systems. While formal methods do not guarantee correctness, they can greatly increase the understanding of a system and help revealing inconsistencies, ambiguities and incompleteness that might otherwise go undetected [1].

CSP, CSS and others are formal methods that are used for specifying *concurrent* behavior.

16.2.4 Traceability

Traceability refers to the relationships between and within the artefacts and other elements [5, 13]. These are plentiful and exist implicitly. But the implicit traceability can be made explicit. By doing so, those traces become themselves artefacts that must be maintained. Therefore, the benefits and costs of making traces explicit must be weight carefully — as with some artefacts, the cost of stale traces may be higher than the cost of no explicit traces.

Making traces explicit can in itself provide useful information. Consider the "is realized by" relationship between requirements and specification. Such a relationship would immediately identify those requirements that are not specified yet, namely those requirements that have no outgoing traces. Such a requirement can then be inspected and the specification extended to realize it. After the specification has been extended, a new trace is created, marking the requirement as realized.

While this approach works in principle, there are at least two problems with it. First, which elements will be traced? It would be nice if there was a one-to-one relationship between requirements and specification elements, but this is true only for the simplest toy examples. In practice, this is an n-to-m relationship, and sometimes one end of the trace can be elusive. Just consider quality requirements that apply to the system as a whole.

Maintenance is the second issue. Creating a trace correctly is one thing, but keeping it updated is quite another. Consider again the 'is realized by" relationship. All incoming traces would have to be verified to make sure that the specification element still, in fact, realizes all requirements that it traces. But this works only if all traces have been created in the first place. And when more corrections have to be done during this verification (both on requirements and specification), it may trigger another wave of verifications. Tools support can help to mark traces for verification — but how much this helps depends on the completeness and correctness of the traces.

The ease of traceability depends, amongst other things, on the structure and quality of the artifacts. For instance, one quality criteria for good requirements is the lack of redundancy. Not having redundancy also eases traceability. Further, there are many ways to structure the artefacts. A good structure can make traceability significantly easier. The structure depends on notation and approach. The approach guides the artefacts towards a certain structure, while the notation constrains how easy or difficult it is to express something. Some notations require a certain approach and may also push the artefacts in a certain structure. This is good if the notation is well-suited for the problem at hand, but it can be counter-productive if this is not the case. Just imagine drawing the blueprint of a house with UML, or to document an enterprise-system with a mechanical drawing. Other notations are open to everything, like natural language. But the downside in this case is that the notation provides no guidance, and can be ambiguous or contradicting.

16.2.5 The Importance of Tool Support

The previous sections described the concerns regarding the working with requirements and specifications. We created RMF specifically to provide a platform that could be used in academia and industry to realize their ideas. We believe that there is a real need for this: research projects often build their own tools in isolation with proprietary data structures, which vastly decreases their survival chances. In industry, we see a lot of customization of proprietary tools (for instance, there is a whole industry creating scripts for IBM Rational DOORS[7]). RMF in turn builds on the open ReqIF standard that is currently being adapted by commercial tools, and it is built on top of Eclipse

[7] http://www-01.ibm.com/software/awdtools/doors/

EMF [18]. Specifically, here are the areas where RMF could be put to use:

Structuring Requirements RMF provides all the features necessary for structuring artefacts, both according to WRSPM or other approaches. In itself, the ProR tool does not put any constraints on the structure, but this can be achieved via specific plug-ins which could also provide guidance to the use, for instance by providing wizards.

Model Integration as we will see in Section 16.6, an integration with models can be achieved via plug-ins as well, especially if the formal modeling tools are built using Eclipse EMF. In that case, referenced model elements can be seamlessly incorporated into ProR specifications.

Traceability ReqIF includes data structures for typed traces, and RMF can be extended for intelligent handling of the traceability. For instance, traces could be marked as suspect, as soon as the source or target element of the trace changes.

16.3 Requirements Interchange Format

We will provide a brief overview on the Requirements Interchange Format (ReqIF) [15] file format and data model. We are mainly concerned with the capabilities and limitations of the data model. The tool that we describe in Section 16.5 uses ReqIF as the underlying data model. Doing so 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[8] by the "Herstellerinitiative Software" (HIS[9]), 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.

The basic use case for ReqIF consists of the following steps (described in detail in the ReqIF specification [15]):

1. The manufacturer exports the subset of requirements that are relevant to the supplier, with the subset of attributes that are relevant.

2. Those attributes that the supplier is expected to modify are writable, other content is marked as readable only.

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

[9] http://www.automotive-his.de/

3. The supplier imports the data from the manufacturer into their system. If this not the first import, then the data may be merged into an existing requirements database.

4. The supplier can then edit the writable attributes, or even create a traceability to other elements in their database (e.g. a systems specification).

5. The supplier performs an export with the data relevant to the manufacturer.

6. The manufacturer merges the data back into their requirements database.

16.3.1 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. we'll first point out a few key model features and then provide more specifics from the ReqIF specification [15].

A *SpecObject* represents a requirement. A SpecObject has a number of *Attribute-Values*, 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.

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 place-holder for tool-spefic data (*ReqIFToolExtension*) and the actual content (*ReqIFContent*). The ReqIFContent has no attributes, but is simply a container for six elements. These are:

SpecObject A SpecObject represent an actual requirement. The values (Attribute-Value) of the SpecObject depend on its SpecType.

SpecType A SpecType is a data structure that serves as the template for anything that has Attributes (e.g. a SpecObject). It contains a list of Attributes, which are named entities of a certain datatype and an optional default value. For example, a SpecObject of a certain type has a value for each of the SpecType's attributes.

DatatypeDefinition A DatatypeDefinition is an instance of one of the atomic data types that is configured to use. For instance, String is an atomic data type. A DatatypeDefinition for a String would have a name and the maximum length of the string. Each attribute of a SpecType is associated with a DatatypeDefinition.

Specification SpecObjects can be grouped together in a tree structure called Specification. A Specification references SpecObjects. Therefore it is possible for the

same SpecObject to appear in multiple Specifications, or multiple times in the same Specification.

In addition, a Specification itself may have a SpecType and therefore AttributeValues.

SpecRelation A SpecRelation is a link between SpecObjects, it contains a source and a target. In addition, a SpecRelation can have a SpecType and therefore AttributeValues.

RelationGroup SpecRelations can be grouped together in a RelationGroup, but only if the SpecRelations have the same source and target Specifications. This construct got added to accommodate certain data structures of existing, proprietary requirements tools.

We just learned that there are four element types that can have attributes: SpecObjects, Specifications, SpecReleastions and RelationGroups. These four are all *SpecElementsWithAttributes*, or *SpecElements* for short. Each SpecElement has its own subclass of SpecType (SpecObjectType, SpecificationType, SpecRelationType and RelationGroupType). A SpecType has any number of AttributeDefinitions, which ultimately determines the values of a SpecElement. Correspondingly, a SpecElement can have any number of AttributeValues. The AttributeValues of a SpecElement depend on the AttributeDefinitions of the SpeElement's SpecType. This fact can not be deducted from the model.

The AttributeDefinition references a DatatypeDefinition that ultimately determines the value of the AttributeValue of the corresponding SpecElement. For each atomic data type of ReqIF, there is a corresponding DatatypeDefinition, AttributeDefinition and AttributeValue each.

The ReqIF specififcation [15] contains a number of class diagrams that nicely visualize these relationships.

ReqIF supports the following atomic data types:

String A unicode text string. The maximum length can be set on the Datattype.

Boolean A boolean value. No customization is possible.

Integer An integer value. The maximum and minimum can be set on the Datattype.

Real An real value. The maximum and minimum can be set on the Datattype, as well as the accuracy.

Date A date- and timestamp value. No customization is possible.

Enumeration An enumeration Datatype consist of a number of enumeration values. The AttributeDefinition determines whether the values are single value or multiple value.

XHTML XHTML is used as a container for a number of more specific content types. The AttributeValue has a flag to indicate whether the value is simplified, which can be used if the tool used to edit only supports a simplified version of the content. For instance, if rich text is not supported, and therefore the new content is stored as plain text.

ReqIF consist of 44 element types in total. The ones we just described are important for understanding ReqIF in general and this chapter in particular. Elements we omitted concern aspects like access control and identifier management.

16.3.2 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 [18], 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 clean-room implementation of the standard based on Eclipse can serve as the basis for both innovative conceptual work and new tools.

This is by no means guaranteed, and there are examples where this approach did not work. For instance, the XMI format (in model-based community) was not that successful, and XMI was also promoted by OMG.

16.4 The Requirements Modeling Framework (RMF)

RMF grew out of the Deploy[10] research project [17] and the VERDE[11] research project. It is an Eclipse Foundation project that unifies a generic core engine to work with RIF/ReqIF content, and a GUI called ProR.

The vision or RMF is to have at least one clean-room 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.

[10] http://www.deploy-project.eu/
[11] http://www.itea-verde.org/

16.4.1 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.

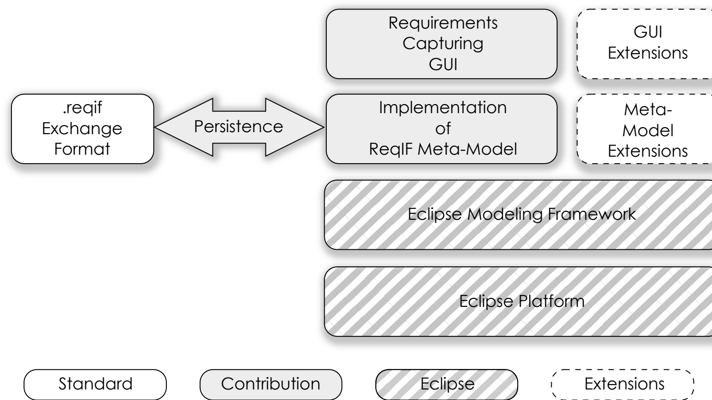

Fig. 16.1: High-level architecture of RMF

The GUI for capturing requirements is called ProR (see also Section 16.5). 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. These are depicted in Figure **??** as well and described in Section 16.6.

16.4.2 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, as we will describe in Section 16.6. It provides an extension point that allows the tailoring with plug-ins.

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 [2] or Topcased [9]. By providing a versatile extension point, the behavior of the application can be adapted to the process employed.

16.5 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.

This section will go through the more important features of ProR to provide an impression of the tool in action. We provided a more extensive introduction to the tool in [10]. We also created a screencast[12] that demonstrates the basic features of ProR.

16.5.1 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.

16.5.2 Creating a ReqIF Model

ReqIF models can be created in any Eclipse project, and manifest themselves as a .reqif file. A user creates a new ReqIF model via the FILE — NEW… menu, where there is a wizard for a new "Reqif10 Model". The wizard will create a new ReqIF model with a very rudimentary structure, with one Datatype, one SpecType with one Attribute, using the Datatype, one Specification with one SpecObject that uses the SpecType.

The user can then inspect the model structure in the outline and the properties views. ProR provides its own *Perspective*, which ensures that all relevant views are shown.

The editor in Figure 16.2 (the window in the middle) provides an overview of the model. The most important section is the one labeled "Specifications". Users can double clicking on specifications to open them in their own editor, as shown in Figure 16.3.

Each row represents a requirement (SpecObject), and each requirement can have an arbitrary number of attributes. Which specific attribute a requirement has depends on its type.

Users can configure the editor to show an arbitrary number of columns. Each column has a name. If an element has an attribute of that name, then the value of that attribute is shown in the corresponding column.

16.5.3 New Attributes

The actual information of requirements (SpecObjects) is stored in its attributes. Which attributes a SpecObject has depends on its type. Users can add more attributes to existing requirement types.

To do this, the user opens the dialog for the datatypes via PROR — DATATYPE CONFIGURATION… or the corresponding icon in the toolbar. The upper part of the dialog shows the data structures, while the lower part contains a property view that

[12] http://www.youtube.com/watch?v=sdfTNZduvZ4

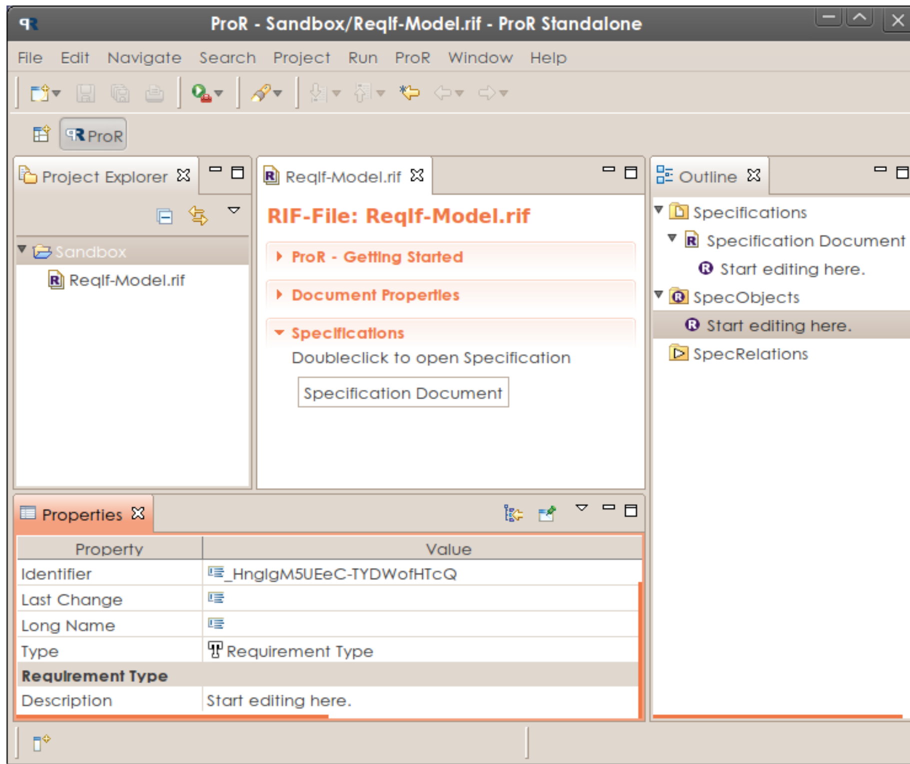

Fig. 16.2: ProR with a newly created ReqIF model, as produced by the wizard

allows editing the properties of the element that is selected in the upper part. New child or sibling elements can be added via context menus.

In this example, the user adds two attributes to the type "Requirements Type": an ID for a human readable identifier, and a status field, which is an enumeration. The result is shown in Figure 16.4.

The user just created a new datatype for the ID called "T_ID". For the status field, the user created a new enumeration of type "T_Status". In the figure, we can see the properties of the selected element in the lower pane, where they can be edited.

16.5.4 Configuration of the Editor

When closing the dialog and select a requirement, the three properties are visible in the properties view, where the user can edit them. But the main pane of the editor still only shows one column. The user can add new columns via PROR — COLUMN CONFIGURATION... (or the corresponding tool bar icon), which opens a dialog for this purpose. The dialog looks and works similar to the one for the datatypes. In this example, the user adds one more column called "ID". The dialog also allows the

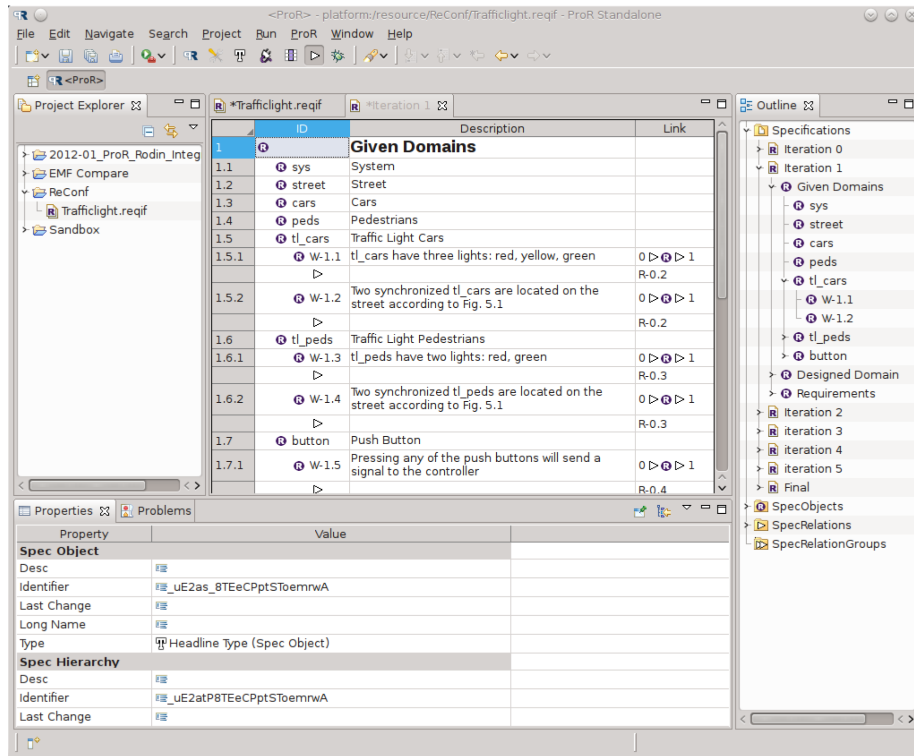

Fig. 16.3: ProR with the specification editor open. The screenshot shows some sample data. The screenshot shows the links expanded.

reordering of columns via drag and drop, and the user uses this mechanism to make the ID column the first one.

With this setup, the user can set the status of individual requirements by selecting them and updating the status field in the properties view. Upon clicking on the field, a drop-down allow the selection of the new status value. Had the user added the status field to the editor (as just described), they could adjust the values directly in the editor as well.

Using this approach, users can add an arbitrary number of attributes to a requirement, which the user can all see and edit in the properties view. A selected number can additionally be shown in the editor. For example, the user may decide that a requirement should have a comment field to record additional information.

16.5.5 Generating IDs

The ID column in now visible in the specification editor, but it is empty. While the user could add IDs simply by hand, this is error prone, and one would expect the tool to be able to handle this. ProR does not have the ability to generate IDs, but a “Presentation”

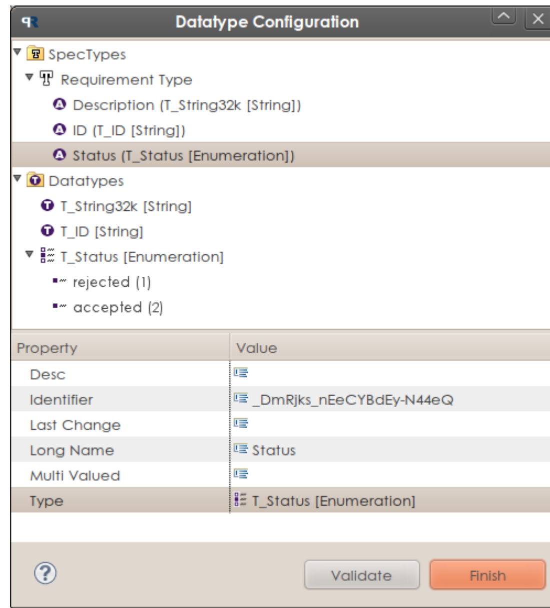

Fig. 16.4: The datatype dialog after adding some data

can. Presentations are ProR-specific plug-ins that modify the presentation of data and inspect and modify the data. Presentations are described from a technical point of view in Section 16.6.

To add a presentation, the user opens the presentation dialog via ProR — Presentation Configuration... (or the tool bar). The Select Action... dropdown lists all installed presentations, and by selecting "ID Presentation", the user creates a new configuration element. In the properties, the user adjusts the prefix and counter of the ID-presentation. But more important is the datatype that is associated with the presentation. In this example, the user selects "T_ID" — and this is the reason why the user created a new datatype for the IDs earlier.

After closing the dialog, all requirements that did not have an ID yet will have received one by the presentation.

16.5.6 Adding Requirements

Finally everything is ready for adding some data. The user does this via the context menus, but in several places, keyboard shortcuts are available as well. Upon opening the context menu for a requirement, the user adds new elements via the New Sibling and New Child submenus. A specification is a tree structure of arbitrary depth, and the left margin indicates via a corresponding numbering scheme the position in the hierarchy. In addition, the left margin of the first column is indented.

The context menu allows the creation of typed requirements — there is one entry

for each user-defined type — which can save a lot of clicking. But it is also possible to add untyped requirements or even empty placeholders (*SpecHierarchies*). Adding a placeholder can be useful for referencing an existing requirement. Requirements may appear multiple times, both in the same specification and in other specifications of the same ReqIF model.

To allow the rapid addition of requirements, ProR provides the CTRL-ENTER keyboard shortcut. Upon activating the shortcut, the new requirement is inserted below the one that is currently selected and has the same type.

Last, a user can rearrange requirements via drag & drop or copy & paste.

16.5.7 Linking Requirements

The user can link requirements via drag & drop. As drag & drop is also used for rearranging requirements, it has to be combined with a keyboard modifier. The key that needs to be pressed is dependent of the operating system and is the same that is used for creating file links and the like.

Once the user creates a link, the last column of the specification editor shows the number of incoming and outgoing links. The user can toggle the showing of the actual link objects (*SpecRelations*) via PROR — SPECRELATIONS..., which are then shown below the originating requirement (depicted in Figure 16.3). The last column of link objects shows the destination object (selecting that column will show the target requirement's properties in the property view).

The user can assign types to link objects, resulting in them having attribute values. The values will be shown in the specification editor, if the columns are configured correspondingly.

This concludes the brief overview of the usage of ProR.

16.6 Extending ProR

The functionality of ProR is quite limited, but this is by design. ProR can be extended using the Eclipse plug-in mechanism. Many features that should be standard in a requirements engineering tool will not be implemented into the ProR core, but could be made available via plug-ins. An example of this has been presented in Section 16.5.5, where a plug-in was responsible for generating user-readable IDs.

Likewise, functionality like versioning or baselining will not be integrated into the core. Versioning is already supported (albeit in a crude manner) by installing a repository plug-in like Subclipse[13] or Subversive[14] (Subversion support) or eGit[15] (git support). However, these plug-ins perform versioning on the file-level. In practice, versioning on the requirement level would be more desirable, and from a technical point of view, it is straight forward to realize this in the form of a plug-in.

[13] http://subclipse.tigris.org/
[14] http://www.eclipse.org/subversive/
[15] http://www.eclipse.org/egit/

Extensions for ProR exist — those have been driven mainly by academic needs so far. In this section, we demonstrate how a developer can integrate RMF with other Eclipse-based tools.

16.6.1 Traceability between Requirements and Event-B Models

The research project Deploy [17] is concerned with the deployment of formal methods in industry. Traceability between natural language requirements and formal models was one issue that the deployment partners were struggling with. Deploy continues to develop the Eclipse-based Rodin tool [2], which was the main deciding factor for using Eclipse for RMF. ReqIF was an attractive choice for providing interoperability with industry tools. By using EMF, we could build directly on the ReqIF data model, which allowed us to get a solid integration quickly. Figure 16.5 shows how we establish traceability between formal models and natural language requirements [12]. The formal modeling is done in Event-B (using Rodin). Integration is seamless via drag and drop, and a custom renderer supports color highlighting of model elements.

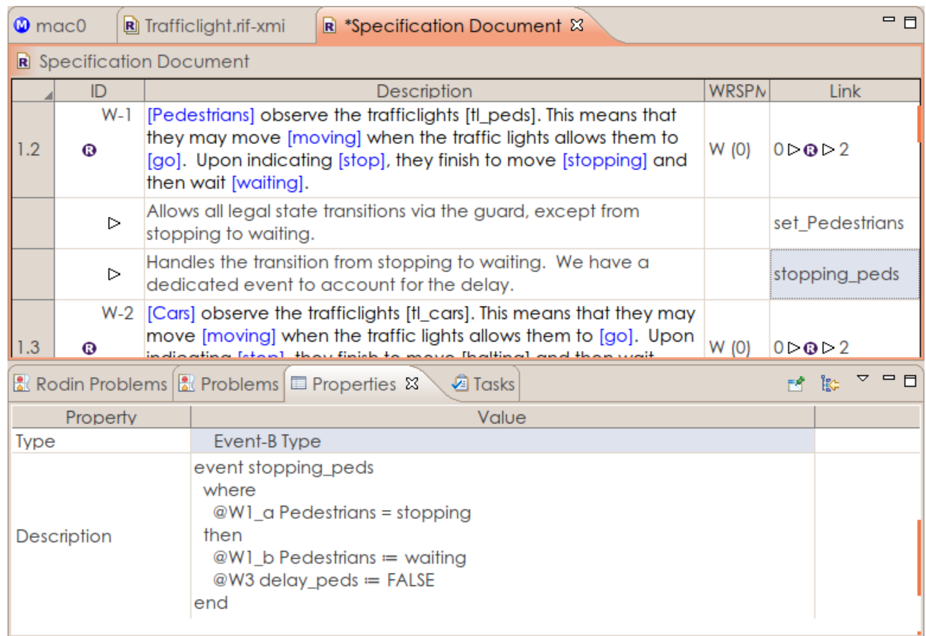

Fig. 16.5: Integration of ProR with an Event-B formal model. Model elements are highlighted in the requirement text, and annotated traces to the model show the model element in the property view.

16.6.2 Tracepoint Approach in ProR

The general concept of traceability in VERDE led to the decision to implement a traceability that is independent of the types of artefacts that are involved. Since Eclipse-based models are usually built on EMF, VERDE implements a generic solution for the traceability of EMF-based elements called tracepoints [11]. The core data structure is a mapping table with three elements: source element, target element and arbitrary additional information. The elements are identified by a data structure, the so called *Tracepoint*. The inner structure of a Tracepoint depends on the structure of the meta-model that is being traced, but is hidden from the traceability infrastructure.

We added an adapter for the tracepoint approach for ProR, which was easy to realize, as ProR is also built using EMF. This is true for all Eclipse-based offerings (and what the tracepoint plug-in is targeted at). The tracepoint application adds a new view to Eclipse. To set a tracepoint, the source and target are being selected in Eclipse and stored by clicking one button for each.

16.6.3 Integration of Domain-Specific Languages

The possibility to specify requirements with textual domain specific languages (DSLs) and to trace these to development artefacts is one of the foundations of the VERDE project, which drove the DSL extension for ProR [11]. A textual DSL is a machine-processable language that is designed to express concepts of a specific domain. The concepts and notations used correspond to the way of thinking of the stakeholder concerned with these aspects while still being a formal notation.

In the Verde requirements editor, the open-source tool Xtext [4] has been used. The introduction of Xtext allows any project to define their own grammar and modelling. Users can design and evaluate new formal notations for the specification of requirements.

The editor for the DSLs integrates itself directly into the requirements tool and is activated as a presentation (as described in Section 16.5.5). Upon editing, a popup-editor appears that gives immediate feedback to the user in the form of syntax highlighting and error markers, and supports the user by providing auto-complete and tool tips, similar to what users are used to in modern programming editors.

16.7 Conclusion

In this chapter, we gave a broad overview of the current state in requirements engineering both in academia and industry. We then introduced the ReqIF data model for requirements, as well as Eclipse RMF and ProR. Last, we provided a few examples on how this ReqIF-RMF-ProR-stack has been used to solve real problems.

As an Eclipse Foundation project, RMF relies on the feedback of users and on contributors to thrive. We hope that we spurred some interest both in academia and industry to see what RMF is capable of and to use it for their projects.

Bibliography

[1] E.M. Clarke and J.M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[2] J. Coleman, C. Jones, I. Oliver, A. Romanovsky, and E. Troubitsyna. RODIN (rigorous open development environment for complex systems). *EDCC-5, Budapest, Supplementary Volume*, page 2326, 2005.

[3] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: An Environment for Goal-driven Requirements Engineering. In *Proc. of the 19th int. conf. on Software engineering*, pages 612–613. ACM, 1997.

[4] S. Efftinge and M. Vlter. oAW xText: a framework for textual DSLs. In *Workshop on Modeling Symposium at Eclipse Summit*, volume 32, 2006.

[5] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, page 94101, 1994.

[6] Carl A. Gunter, Michael Jackson, Elsa L. Gunter, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17:37–43, 2000.

[7] IEEE. Recommended practice for software requirements specifications. Technical Report IEEE Std 830-1998, IEEE, 1997.

[8] M Jackson. *Problem frames: analysing and structuring software development problems*. Addison-Wesley/ACM Press, Harlow England ;;New York, 2001.

[9] Michael Jastram and Andreas Graf. Requirement traceability in Topcased with the requirements interchange format (RIF/ReqIF). *First Topcased Days Toulouse*, 2011.

[10] Michael Jastram and Andreas Graf. Requirements modeling framework. *Eclipse Magazin*, 6.11, 2011.

[11] Michael Jastram and Andreas Graf. Requirements, traceability and DSLs in eclipse with the requirements interchange format (RIF/ReqIF). In *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII*. fortiss GmbH, Mnchen, 2011.

[12] Michael Jastram, Stefan Hallerstede, and Lukas Ladenberger. Mixing formal and informal model elements for tracing requirements. In *Automated Verification of Critical Systems (AVoCS)*, 2011.

[13] Michael Jastram, Stefan Hallerstede, Michael Leuschel, and Aryldo G Russo Jr. An approach of requirements tracing in formal refinement. In *VSTTE*. Springer, 2010.

[14] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley, 2004.

[15] OMG. Requirements interchange format (ReqIF) 1.0.1. 2011.

[16] D. L. Parnas and J. Madey. Functional documents for computer systems. *Science of Computer programming*, 25(1):41–61, 1995.

[17] B. Part. Deploy project. 2008.

[18] D. Steinberg, F. Budinsky, M. Peternostro, and E. Merks. *EMF Eclipse Modeling Framework*. Addison-Wesley, 2 edition, 2009.

[19] A. Van Lamsweerde et al. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, volume 249, page 263, 2001.