Requirements Traceability between Textual Requirements and Formal Models Using ProR

Lukas Ladenberger and Michael Jastram

Institut für Informatik, Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf[**]
{ladenberger,jastram}@cs.uni-duesseldorf.de

Traceability within a system description is a challenging problem of requirements engineering [1]. In particular, formal models of the system are often based on informal requirements, but creating and maintaining the traceability between the two can be challenging. In [2], we presented an incremental approach for producing a system description from an initial set of requirements. The foundation of the approach is a classification of requirements into *artefacts* W (domain properties), R (requirements) and S (specification) [3]. In addition, the approach uses designated *phenomena* as the vocabulary employed by the artefacts. The central idea is that *adequacy* of the system description must be justified, meaning that $W \wedge S \Rightarrow R$. The approach establishes a traceability, and the resulting system description may consist of formal and informal artefacts.

We created tool support for this approach by integrating Rodin [4] and ProR [5]. Rodin is an Eclipse-based open tool platform for formal modelling in Event-B [6]. ProR is a platform for requirements engineering that is also based on Eclipse and part of the Eclipse Requirements Modeling Framework (RMF)[1].

A seamless integration between ProR and Rodin is possible, as both are based on Eclipse. The integration plug-in is installed into Rodin via an update site[2]. We designed it with the goal to support the approach described in [2] and to ease the integration of natural language requirements and Event-B. Supporting other formalisms is possible in principle, and we are currently working on supporting integration with classical B [7]. Figure 1 shows ProR installed inside Rodin.

The integration allows the identification of phenomena within natural language requirements (Rodin already allows the identification of phenomena in formal model artefacts); it supports the creation of traces between arbitrary artefacts; and it tracks whenever the source or target of a trace changes by marking it as "suspect" (allowing the re-validation of traces).

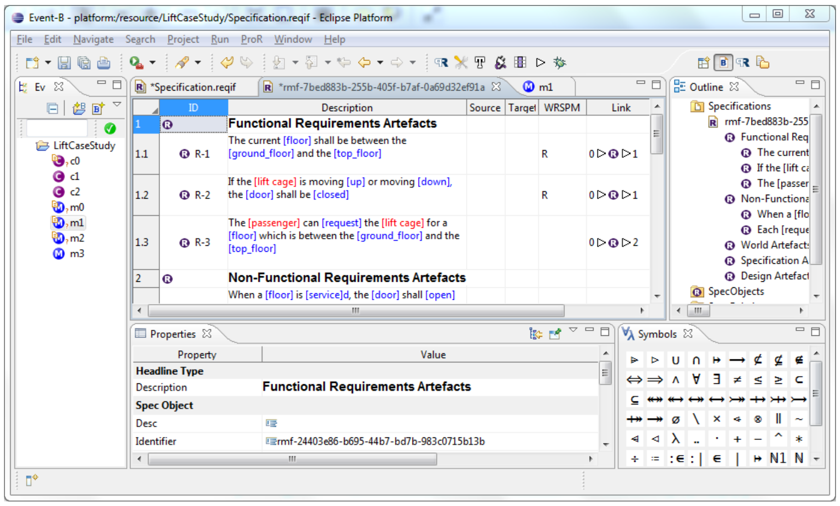
ProR already supports some features required for an integration. For instance, ProR supports classifying informal and formal artefacts as W, R and S. Other features had to be provided by an integration plug-in, as described below.

There are still some limitations, that we discuss in the conclusion. In the following, we describe the specific features of the tool in more detail.

[**] Part of this research has been sponsored by the EU funded FP7 project 214158: DEPLOY (Industrial deployment of advanced system engineering methods for high productivity and dependability).

[1] http://eclipse.org/rmf.

[2] The update site URL is http://www.stups.uni-duesseldorf.de/pror_updates.

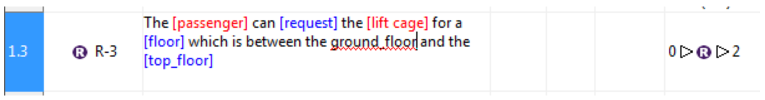


	ID	Description	Source	Target	WRSPM	Link
1		**Functional Requirements Artefacts**				
1.1	R-1	The current [floor] shall be between the [ground_floor] and the [top_floor]			R	0 ▷ Ⓡ ▷ 1
1.2	R-2	If the [lift cage] is moving [up] or moving [down], the [door] shall be [closed]			R	0 ▷ Ⓡ ▷ 1
1.3	R-3	The [passenger] can [request] the [lift cage] for a [floor] which is between the [ground_floor] and the [top_floor]				0 ▷ Ⓡ ▷ 2
2		**Non-Functional Requirements Artefacts**				
		When a [floor] is [service]d, the [door] shall [open]				

Property	Value
Headline Type	
Description	**Functional Requirements Artefacts**
Spec Object	
Desc	
Identifier	rmf-24403e86-b695-44b7-bd7b-983c0715b13b

Fig. 1. ProR running inside the Rodin Platform for Event-B modelling.

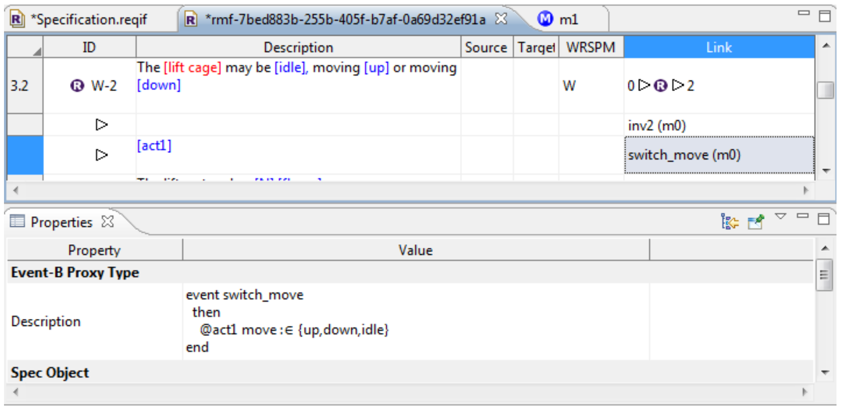
Tracing of phenomena used in artefacts Textual requirements are rendered by colour-highlighting those text passages that correspond to phenomena (Figure 1). The user has to mark them by square brackets. In doing so, the text passage is rendered in blue, otherwise in red, reminding the user that an undeclared phenomena is used. In addition, unmarked, recognised phenomena are highlighted as well to warn the user about a possible omission (Figure 2). The marked phenomena are automatically synchronized with the model.



1.3	R-3	The [passenger] can [request] the [lift cage] for a [floor] which is between the ground_floor and the [top_floor]			0 ▷ Ⓡ ▷ 2

Fig. 2. The tool warns the user about a possible omission.

Annotated traces to modelling elements Manual creation of traces between requirements and formal model elements is supported via drag and drop. Figure 1 shows how the right column "Link" of the specification editor summarizes the number of outgoing (target) and incoming (source) traces. Details of the outgoing trace can be switched on as shown in figure 3. Selecting an outgoing trace shows the targets properties in the Property View. For instance, we see in figure 3 that the trace target which is the event *switch_move* is selected. Selecting the target shows its attributes in the Property View, including the formal event itself. This is a reference to the model, not a copy of the event. As a consequence, whenever the formal model element changes, the reference also changes (and the trace will be marked as “suspect”, as described below). In addition, traces can be annotated if additional information is necessary.



*Specification.reqif | *rmf-7bed883b-255b-405f-b7af-0a69d32ef91a | m1

	ID	Description	Source	Target	WRSPM	Link
3.2	W-2	The [lift cage] may be [idle], moving [up] or moving [down]			W	0 ▷ R ▷ 2
	▷					inv2 (m0)
	▷	[act1]				switch_move (m0)

Properties

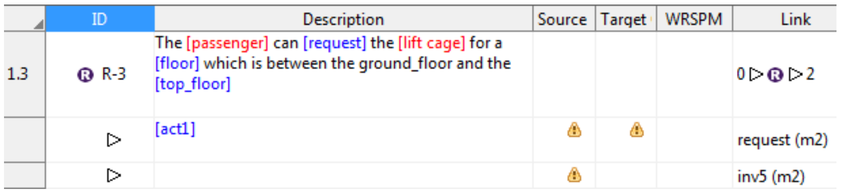
Property	Value
Event-B Proxy Type	

Description	event switch_move
	then
	@act1 move :∈ {up,down,idle}
	end

Spec Object	

Fig. 3. The unveiled traces of an element. As the link target is selected, the link target’s properties are shown in the Property View (the lower pane).

Change management to both the requirements and the formal model Last, when traced formal model elements change, the trace is marked as “suspect” by showing a small icon as demonstrated in figure 4. Two columns exist for the source and the target of the trace, respectively. The user sees at a glance which requirements or formal model elements need to be revalidated. This is particularly useful if the requirements document becomes large. By double-clicking on the “suspect” icon, the user can mark the trace as “revalidated” and the icon will be removed.



	ID	Description	Source	Target	WRSPM	Link
1.3	R-3	The [passenger] can [request] the [lift cage] for a [floor] which is between the ground_floor and the [top_floor]				0 ▷ R ▷ 2
	▷	[act1]	⚠	⚠		request (m2)
	▷		⚠			inv5 (m2)

Fig. 4. A small icon indicates whenever the source (the requirement) or the target (formal model element) needs to be revalidated.

Conclusion We believe that the integration between Rodin and ProR supports the user in managing requirements in natural language and the corresponding traces to formal model elements, as outlined by our approach described in [2].

There are still some limitations, however. While all required data structures exist, the tool would benefit from more sophisticated reporting. In particular, [2] lists a number of properties of a correct system description. While the presence of these properties does not guarantee correctness, their absence indicates a problem. Reporting on the state of these properties would be valuable.

Furthermore, we believe that the integration is useful even beyond supporting our approach. For instance, the capability of marking traces as "suspect" if the source or the target change could be useful in many situations, even without the use of formal methods.

Last, we believe that this integration brings two complimentary fields of research, requirements engineering and formal modelling, closer together.

References

[1] O. Gotel and A. Finkelstein: An Analysis of the Requirements Traceability Problem IEEE Computer Society (1994)

[2] M. Jastram and S. Hallerstede and L. Ladenberger: Mixing Formal and Informal Model Elements for Tracing Requirements AVOCS 2011 (2011)

[3] C. A. Gunter and M. Jackson and E. L. Gunter and P. Zave: A Reference Model for Requirements and Specifications IEEE Software Vol. 17, 37–43 (2000)

[4] J.-R. Abrial and M. J. Butler and S. Hallerstede and T. S. Hoang and F. Mehta and L. Voisin: Rodin: An Open Toolset for Modelling and Reasoning in Event-B STTT Vol. 12, 447–466 (2010)

[5] M. Jastram: ProR, an Open Source Platform for Requirements Engineering based on RIF SEISCONF (2010)

[6] J.-R. Abrial: Modeling in Event-B: System and Software Engineering Cambridge University Press (2010)

[7] J.-R. Abrial: The B-Book: Assigning programs to meanings Cambridge University Press (1996)