Requirement Traceability in Topcased with the Requirements Interchange Format (RIF/ReqIF)

Submission for First Topcased Days Toulouse 2011

Andreas Graf, Michael Jastram

1 Overview

One important step of the systems engineering process is requirements engineering. Parallel to the development of Topcased, which includes tooling for requirements engineering, a new standard for requirements exchange is emerging at the OMG under the name "ReqIF" (formally called RIF). In our talk we introduce the activities of two research projects and their tool developments, VERDE (Yakindu Requirements) and Deploy (ProR) and discuss possible synergies with Topcased.

1.1 The RIF/ReqIF Standard

RIF/ReqIF[1] is an emerging standard for requirements exchange, driven by the German automotive industry. It consists of a data model and an XML-based format for persistence.

RIF was created in 2004 by the "Herstellerinitiative Software", a body of the German automotive industry that oversees vendor-independent collaboration. Within a few years, it evolved from version 1.0 to the current version 1.2. The format gained traction in the industry, and a number of commercial tools support it.

In 2010, the Object Management Group took over the standardization process and released the ReqIF 1.0 RFC (Request For Comments). The name was changed to prevent confusion with the Rule Interchange Format, another OMG standard.

We were torn on whether to build ProR and Yakindu Requirements on RIF 1.2 or ReqIF 1.0. ReqIF is much cleaner, but is not yet finalized and there is no interoperability with industry tools. After we made the decision to cooperate, we agreed on RIF 1.2.

The ReqIF standardization group models the meta-model of the requirements in UML. Thereby the entrance of model-based software development keeps within a limit.

To give an impression on the structure and documentation of RIF/ReqIF, Figure 1 shows the UML diagram of the top level ReqIF element, taken from the ReqIF specification. You can see that the top level element contains exactly one header, one content element and any number of tool extensions. Of course, this is just a small detail of the standard. The specification is much more elaborate.

In RIF a SpecObject represents a requirement. A SpecObject has a number of AttributeValues, which hold the actual content of the SpecObject. SpecObjects are organized in SpecHierarchyRoots, which

1 http://www.omg.org/cgi-bin/doc7mantis/10-03-07.pdf

are hierarchical structures holding SpecHierarchy elements. Each SpecHierarchy refers to exactly one SpecObject. This way, the same SpecObject can be referenced from various SpecHierarchies.

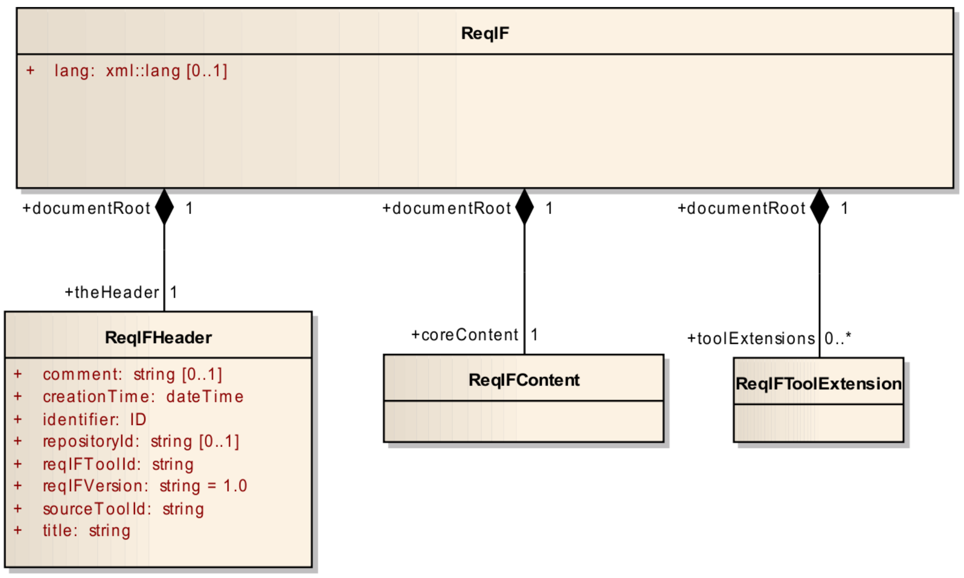


Figure 1: Part of the ReqIF Data Model

1.2 The Verde Project

The research project Verde[2] has the aim of providing a universal tool platform for the verification- and validation-orientated development of embedded systems. Verde is the abbreviation for “VERification-oriented & component-based model Driven Engineering for real-time embedded systems”. The focus is on the integration of the tools which are already in use at the industrial partners. Verde develops new tools and methods in the areas where there are gaps in existing tool-chains and procedures.

1.3 The Deploy Project

Deploy[3] is an European Commission Information and Communication Technologies FP7 project. Its goal is to make major advances in engineering methods for dependable systems through the deployment of formal engineering methods. Deploy will use the Event-B[4] formal method as a basis, for which tool support in the form of the Rodin platform exists. Event-B is a formal method for system-level modelling and analysis. Key features of Event-B are the use of set theory as a modelling notation, the use of refinement to represent systems at different abstraction levels and the use of mathematical proof to verify consistency between refinement levels.

2 Formal notation and models

Traditionally requirements are expressed in natural language. However the natural language suffers the disadvantage of not being precise enough and impedes (mechanical) analysis. Accordingly,

2 http://www.itea-verde.org/

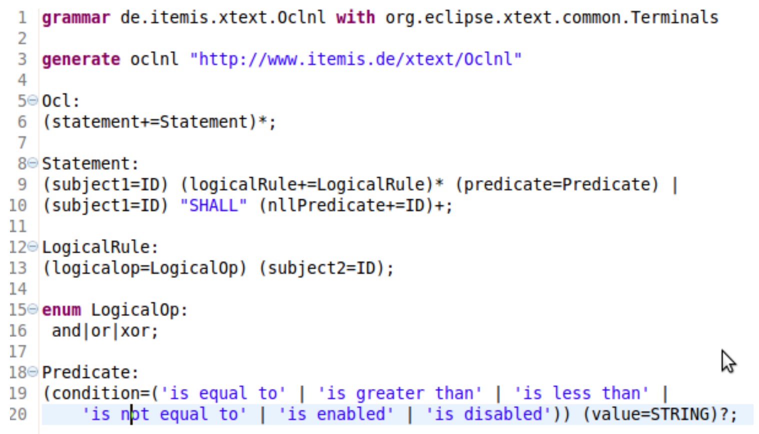
3 http://www.deploy-project.eu/

4 J. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

requirements engineering recommends to find clearer wording by means of formal notations. The engineer can directly write down the requirements in formal language or replenish them with models.

2.1 Domain-Specific Languages

With domain-specific languages (DSL), a method from software development, the suitability for the project can be improved. A DSL is a specific machine-processed language which is designed to characterise specific aspects of a system in a special domain. The concepts and notations used correspond to the way of thinking of the stakeholder concerned with these aspects. Domain-specific languages can thereby be graphical or textual. The advantage of DSLs has long be known. However, the best modelling language has only limited use if there is no tool support. The increasing spread of DSLs is also due to the availability of powerful tool support in recent years. Thus, their use on a broader-basis is possible. Modern age “Language Workbenches” build from a language specification complete development environments with integration, syntax-highlighting and navigation. In the Verde requirements-editor the open-source tool Xtext is adopted. The editor for the domain-specific languages integrates itself directly into the requirements tool. The grammar describes a language that allows both free text, that must be formulated in “shall”-form , and also a more formal notation for general conditions (see Figure 2).



```
grammar de.itemis.xtext.Oclnl with org.eclipse.xtext.common.Terminals

generate oclnl "http://www.itemis.de/xtext/Oclnl"

Ocl:
(statement+=Statement)*;

Statement:
(subject1=ID) (logicalRule+=LogicalRule)* (predicate=Predicate) |
(subject1=ID) "SHALL" (nllPredicate+=ID)+;

LogicalRule:
(logicalop=LogicalOp) (subject2=ID);

enum LogicalOp:
 and|or|xor;

Predicate:
(condition=('is equal to' | 'is greater than' | 'is less than' |
    'is not equal to' | 'is enabled' | 'is disabled')) (value=STRING)?;
```

Figure 2: A simple XText Grammar

2.2 The Event-B Method

In the Deploy project, we demonstrate our ideas using Event-B, a formalism and method for discrete systems modelling. Event-B is a state-based modelling method. The choice of Event-B over similar methods is mostly motivated by the built-in formal refinement support and the availability of a tool for experimentation with our approach.

Event-B models are characterized by proof obligations. Proof obligations serve to verify properties of the model. To a large degree, such properties originate in requirements that the model is intended to realize.

Eventually, we expect that by verifying the formal model we have also established that the requirements to which they correspond are satisfied.

3 Traceability

To assure that the requirements are fully implemented in the system, it is necessary to trace this during the whole development process. This enables validation and verification. On changes to the requirements the effect on the software can easily be tracked down. Vice versa in later phases on changes it is easy to analyse which requirements were affected by the changes. Therefore it is necessary not only to assign requirements to artefacts but also to relate artefacts of earlier phases with artefacts of later phases.

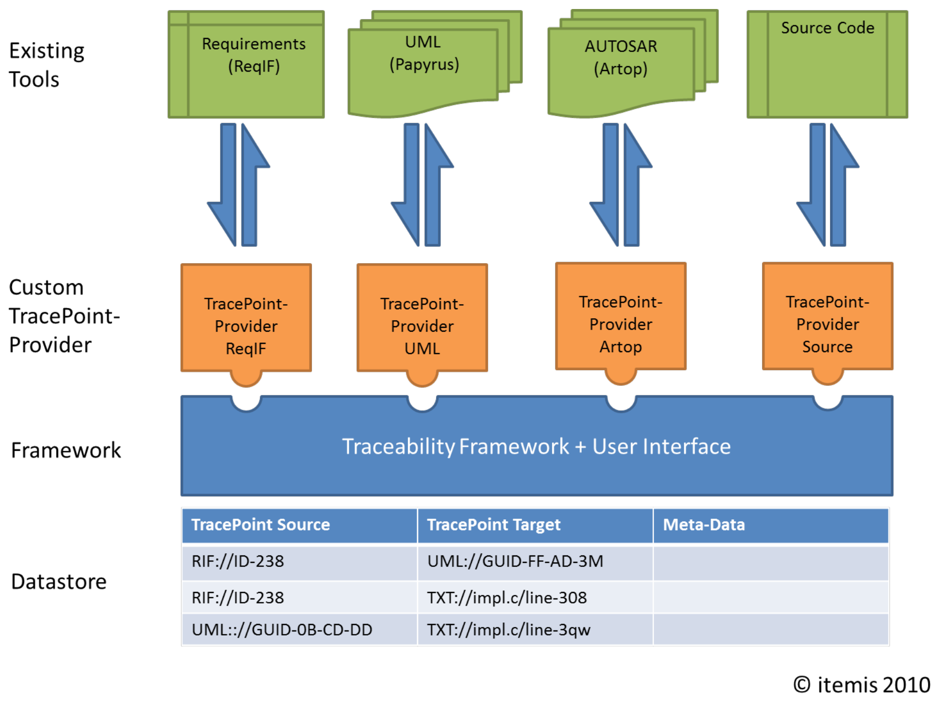
3.1 VERDE traceability

This generalization is reflected technically in an generic solution for the traceability of EMF-based elements (see Figure 3). Centre of the implemented solution is a mapping table with three elements: origin element, target element and additional information like description etcetera. The elements are defined over an identifier, the so called ‚"Tracepoint". The exact structure of a tracepoint at the same time depends on the corresponding model. A UML model element for example is identified by an unique ID, a C source file in contrast is identified by file path and name. The basic implementation does support the assignment of model elements with a simple user interface: the origin element gets selected in the corresponding tool and thereafter marked in the window for assignments. After this, the target element is marked in the corresponding tool and the connection is imprinted in the assignment window. Similarly the assigned elements can be evaluated from every single element and can be navigated to. A more user friendly adoption to model types is possible over extensions. These provide more information on tracepoints to the tracing kernel:

:: **A readable name.** This is necessary since the element defining identifiers are often cryptic to the human user and thus barely usable for an oversight.

:: **Information on the design of the model element.** This could be information e.g. on diagrams, editors etc. Thereby the required view in Eclipse is restored on navigation to a corresponding element.

Technical solutions of other tools are often based on the import of requirements into the target model. In this the requirements are imported with the stereotype "«Requirement»" similar to the UML and traceability is enabled by means of modelling. However, this has the constraint that not all models support suchlike extensions. AUTOSAR for example does not have such an extension concept and also the traceability to code is not considered.


Existing Tools
Requirements (ReqIF)
UML (Papyrus)
AUTOSAR (Artop)
Source Code
Custom TracePoint-Provider
TracePoint-Provider ReqIF
TracePoint-Provider UML
TracePoint-Provider Artop
TracePoint-Provider Source
Framework
Traceability Framework + User Interface
Datastore
© itemis 2010

TracePoint Source	TracePoint Target	Meta-Data
RIF://ID-238	UML://GUID-FF-AD-3M	
RIF://ID-238	TXT://impl.c/line-308	
UML::/GUID-0B-CD-DD	TXT://impl.c/line-3qw	

Figure 3: The Architecture of the Traceability Framework

3.2 Event-B Traceability

The Deploy approach to traceability is based on WRSPM[5]. WRSPM is a reference model for applying formal methods to the development of user requirements and their reduction to a behavioural system specification. WRSPM distinguishes between artefacts and phenomena. Phenomena describe the state space (and state transitions) of the domain and system, while artefacts represent constraints on the state space and the state transitions. The artefacts are broadly classified into groups that pertain mostly to the system versus those that pertain mostly to the environment.

In our approach[6], we use "proper requirements" for artefacts of the formal model that correspond to WRSPM-requirements, while we use just "requirements" for natural language requirements, as provided by the stakeholders. In practice, these may also contain information about the domain, implementation details, general notes, and all kinds of related information.

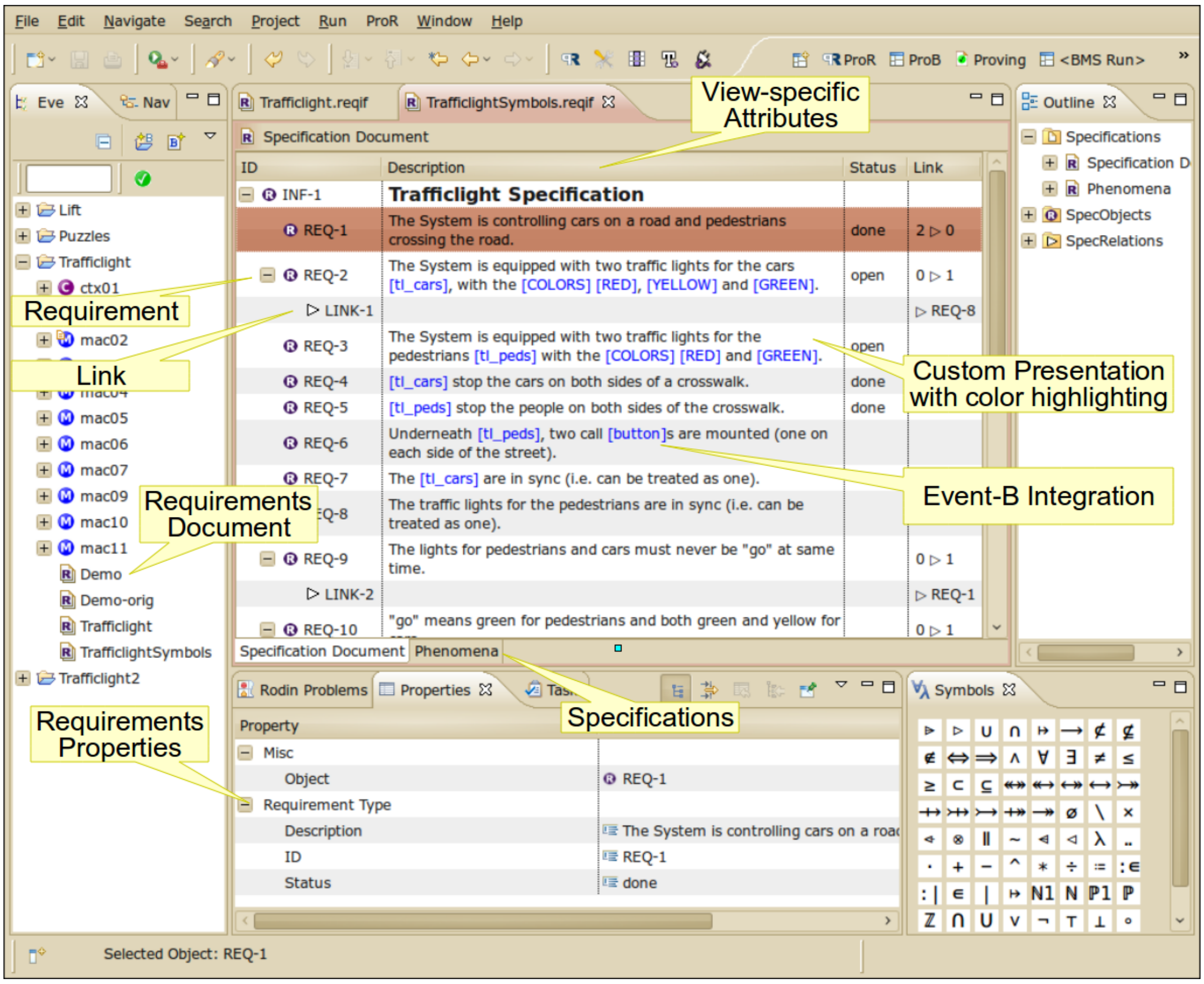
Our goal is to establish requirements traceability from natural language requirements to an Event-B formal model, using WRSPM to provide structure to both the requirements and the model. To achieve this, we introduce a "realize" relationship that we refine into different types of traces, as well as criteria to verify that the traces have been established correctly. This allows us to "justify" that the formal model realizes the requirements.

5 C.A. Gunter, M. Jackson, E.L. Gunter, and P. Zave, "A Reference Model for Requirements and Specifications," IEEE Software, vol. 17,2000, pp. 37-43.

6 M. Jastram, S. Hallerstede, M. Leuschel, and A. G. Russo Jr, "An Approach of Requirements Tracing in Formal Refinement," VSTTE, Springer, 2010.

4 Tool Support

The Verde-Project produced an EMF[7]-based implementation of the RIF data model, which represents the foundation for both the Verde and the Deploy projects. The two projects provide their own GUI that is adapted to the corresponding approach. Due to the reliance on EMF, both tool solutions can be installed into any Eclipse-based system for deployment, including Topcased.



Specification Document

ID	Description	Status	Link
INF-1	**Trafficlight Specification**		
REQ-1	The System is controlling cars on a road and pedestrians crossing the road.	done	2 ▷ 0
REQ-2	The System is equipped with two traffic lights for the cars [tl_cars], with the [COLORS] [RED], [YELLOW] and [GREEN].	open	0 ▷ 1
▷ LINK-1			▷ REQ-8
REQ-3	The System is equipped with two traffic lights for the pedestrians [tl_peds] with the [COLORS] [RED] and [GREEN].	open	
REQ-4	[tl_cars] stop the cars on both sides of a crosswalk.	done	
REQ-5	[tl_peds] stop the people on both sides of the crosswalk.	done	
REQ-6	Underneath [tl_peds], two call [button]s are mounted (one on each side of the street).		
REQ-7	The [tl_cars] are in sync (i.e. can be treated as one).		
REQ-8	The traffic lights for the pedestrians are in sync (i.e. can be treated as one).		
REQ-9	The lights for pedestrians and cars must never be "go" at same time.		0 ▷ 1
▷ LINK-2			▷ REQ-1
REQ-10	"go" means green for pedestrians and both green and yellow for		0 ▷ 1

Property	
Misc	
Object	REQ-1
Requirement Type	
Description	The System is controlling cars on a roa
ID	REQ-1
Status	done

Figure 4: The ProR GUI (here shown running inside the Rodin Platform)

4.1 ProR

The tool from the Deploy project is called ProR[8]. The user has a customizable, tabular view of the requirements. ProR uses RIF SpecRelations for traceability. Formal Event-B elements have a corresponding proxy object in the RIF model that is automatically synchronized with the Event-B model.

ProR itself is independent from Rodin (the tool platform for Event-B). Instead, it is customized with Plug-in to allow for the integration with Rodin. It provides a number of Extension Points for this purpose. Correspondingly, it could be integrated with any other Eclipse- or EMF-based application.

7 Eclipse Modeling Framework, http://www.eclipse.org/emf/

8 http://pror.org

The GUI of ProR is shown in Figure 4. For Eclipse users, it should look immediately familiar. The left pane shows projects that can be expanded to show their content. The view in the middle shows the RIF Specifications, which can be customized to show only selected Attributes in a table view. For the selected SpecObject (the selected row), all Attributes are shown in the Properties View on the bottom.

Figure 4 also shows how an integration with other tools may look like. Here we see the Rodin integration. Variables from the formal model are recognized and rendered in colour in the main view.

5 Integration with Topcased

We see a strong development of RIF/ReqIF in the industry and would suggest an integration with Topcased. We discuss three approaches and suggest a discussion with the Topcased community:

1. **RIF import for Topcased:** There is already a number of importers. The concept of importers could be used for RIF. This would be fairly easy to implement, but a lot of the features of RIF are not taken advantage of.
2. **Adding a RIF Requirements model to Topcased:** This would allow for a tight integration of RIF. But it would result of two independent requirements models in Topcased, leading to redundancy and duplication of effort.
3. **EMF-Based Synchronizer inside Topcased:** This should be fairly easy to implement and would allow for a tight integration between the RIF editor and the Topcased requirements model. At this point, this appears to be the most attractive approach.

6 Conclusion and Future Plans

RIF support would give Topcased an interface to commercial requirements management tools and would thereby make it more attractive for professional deployment. For the same reason that Verde and Deploy joined forces, everybody would benefit from closer collaboration.