A Systems Engineering Tool Chain Based on Eclipse and Rodin

Michael Jastram[1,2]

[1] Heinrich-Heine Universität Düsseldorf
[2] Formal Mind GmbH

Abstract. Formal methods are experiencing a renaissance, especially in the development of safety-critical systems. An indicator for this is the fact that more and more standards either recommend or prescribe the use of formal methods.
Using formal methods on an industrial scale requires their integration into the system engineering process. This paper is exploring how an integrated tool chain that supports formal methods may look like. It thereby focusses on our experience with tool chains that are based on the open source Eclipse platform in general, and the Rodin formal modeling environment in particular.
Open Source allows organisations to remedy the risk of being dependent on one single vendor. This includes the risk of the feature set provided: users can add missing features themselves or commission their inclusion to any competent party, rather than having to rely on the vendor to implement it. It further includes the risk of maintenance and long-term support.
We see industrial interest in open source for systems engineering in general, and Eclipse in particular. Eclipse is attractive, because its license is business-friendly. Further, its modular architecture makes it easy to seamlessly integrate the various Eclipse-based tools for systems engineering.
This paper focuses on an ecosystem that is accumulated around two Eclipse-based platforms, First, the Rodin platform is an open source modeling environment for the Event-B formalism. Second, the Requirements Modeling Framework (RMF) is a platform for working with natural language requirements, supporting the international ReqIF standard.

1 Introduction

This paper is concerned with two trends in systems engineering. First formal methods are finally deployed to complement informal development methods, especially for the development of safety-critical systems [15]. Second, open source is becoming a serious alternative to commercial tools, and Eclipse in particular is being used more and more in systems development.

But both trends are at an early stage. The use of formal methods is still the exception and not the rule. This is due to the fact that the established

development approaches still work, and that the risk and cost of using formal methods is considered high [3].

Open source is considered risky as well. For the longest time, uncertainty with respect to maintenance and support was an issue, and the quality of commercial software used to be higher. But most of these issues have been addressed by companies that specialize on this niche [13].

We believe that the majority of pieces for an Eclipse-based tool chain supporting formal methods already exists. The task at hand is to provide the missing pieces, to raise awareness of it and to encourage its use in academia and in industry.

This paper is structured as follows: In the reminder of the introduction (Section 1) provides an overview of the developments in the area of formal methods, as well as the disciplines in systems engineering that a successful tool chain must cover.

Section 2, we investigate the available Eclipse-based tools for systems engineering, and how they are used today.

Section 3 takes a closer look at Rodin and how it has been deployed in industry. We also investigate its ecosystem, what extensions exist, and which are missing for completing a systems engineering tool chain.

Section 4 concludes with an outlook and plans for making Eclipse-based systems engineering a reality.

1.1 Formal Methods

The idea of using mathematics to precisely specify systems has been around for a long time. Formal methods are a particular kind of mathematically-based techniques for the specification, development and verification of software and hardware systems.

Unfortunately, for a long time it was not clear how these methods could be applied on an industrial scale. Furthermore, there were a number of misconceptions about formal methods that discouraged potential users further [4]. Also, formal methods can be applied in various stages of system engineering, from formal requirements all the way to code generation. There are drastic variations in pay-off, depending on the area of application [3].

The attitude is slowly changing, as industrial success stories become available (see Section 3). Further, the value of formal methods has been recognized especially for the development of safety-critical systems [15]. This insight manifested itself in various standards, which recommend or even mandate the use of formal methods. Examples include ISO 26262 or certain Safety Integrity Levels (SIL).

In traditional development methods (e.g. [5], [11]), initially the project costs are low and reach their peak during the implementation phase. In contrast, the use of formal methods imposes high costs during the requirements elicitation and specification phases, with the promise of much lower costs during the implementation phase. Further, the real pay-off is achieved during the maintenance phase and reuse.

The formalism introduced in Section 3 is called Event-B. 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.

1.2 Systems Engineering Disciplines

The objective of this paper is to explore what is missing for an integrated systems engineering tool chain. To answer this question, we need to identify the disciplines for which tool support must exist, and how the tools must be integrated. The following broad disciplines were taken from OpenUP [2].

Requirements. How to elicit, analyse, specify, validate, and manage the requirements for the system to be developed.
Architecture. How to create software architecture from architecturally significant requirements. The architecture is built in the Development discipline.
Development. How to design and implement a technical solution that conforms to the architecture and supports the requirements.
Test. How to provide feedback about the maturing system by designing, implementing, running, and evaluating tests. This typically includes validation.
Project Management. How to coach, facilitate, and support the team, helping it to deal with risk and obstacles found when building software.
Deployment. How to plan for and deploy a solution.
Environment. How to customize process and tools for a project or organization.

2 Systems Engineering with Eclipse

A variety of open source software for systems engineering is available[3]. However, most of these tools are designed as stand-alone software. While integration with other tools is possible, the result is rarely seamless.

Eclipse is an open source framework for building platform-independent GUI applications. It consists of an extensible plug-in system that is designed for extensibility. To be more precise, the Eclipse core merely provides the capability of managing plug-ins, the actual functionality that is relevant to the user is always provided by plug-ins. Further, plug-ins that have the potential to be reused, typically provide "extension points", which are interfaces that can be used by other plug-ins. This allows the seamless integration of Eclipse-based tools that were developed independently. Section 3 provides an example of this, where the two independent tools Rodin and ProR (part of the RMF project) are integrated to realise traceability between requirements and formal specifications.

While Eclipse enables integration on the tool level, interoperability on the data level is important to allow individual components to be exchanged. Standards are crucial in this respect, and the effect of standardization could be seen

[3] Examples at http://wiki.developspace.net/Open_Source_Engineering_Tools

in the area of modeling: 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. In the area of requirements engineering, the recently established ReqIF standard seems to have a similar effect.

Eclipse is managed by the Eclipse Foundation, which ensures that official Eclipse projects are interoperable and follow certain intellectual property guidelines. As of this writing, the foundation manages roughly 250 projects. Countless other Eclipse project exists, which are not managed by the foundation, including the formerly mentioned Rodin platform.

While Eclipse was initially designed as an Integrated Development Environment for Java and later for Software in general, it recently gained attention in the embedded market. In particular, Airbus decided to cosponsor the development of Topcased, a software environment primarily dedicated to the realisation of critical embedded systems including hardware and/or software [6]. An integration of Topcased with RMF has been proposed and is under development [8].

The Eclipse ecosystem provides a number of projects that could be used in a systems engineering tool chain. This includes reporting tools; team support; modeling frameworks, both textual and graphical; modeling notations like UML and SysML; requirements management; domain specific language support; and real time software components.

In addition to these, there is a vast number of projects that are not managed by the Eclipse foundation. These are typically available through open source hosting services like Eclipse Labs, Git Hub or SourceForge, as well as public academic and commercial repositories.

Last, the Eclipse Public License makes it relatively easy to mix open source code with commercial closed-source offerings.

2.1 Systems Engineering Tool Chains

Typically, users start with a base installation of Eclipse that is already tailored, and customize it by installing additional plug-ins. A number of such base installations exist. A small number of popular base installations can be downloaded from the Eclipse web site, but many projects offer their own base installations on their respective project web sites.

There are a number of base Eclipse installations for systems engineering — examples include Topcased, Unicase or the Open System Engineering Environment (OSEE). Acknowledging the need for domain-specific tools, the Eclipse automotive working group started to develop an industry-specific tool.

However, there is also possible to start with one component of the systems development tool chain and to add all the missing pieces to it. We will present this approach in Section 3, where we step by step extend the Rodin platform to cover most aspects of systems engineering.

3 A Rodin-based Systems Engineering Platform

Our objective is to build a systems engineering platform that integrates formal modelling into the development process. There is support for several formal methods in Eclipse, like CZT (Z) or Overture (VDM) In this paper, we focus on the Rodin platform for Event-B modeling [1]. The Event-B formalism uses set theory and refinement, allowing the representation of different abstraction levels of the system. Consistency between the refinement levels is verified by mathematical proof. The Rodin tool consists of a modeling and proving environment.

We focus on Rodin, because it attracted a vibrant community, and a number of extensions for systems engineering already exist. The author's institution created two such extensions, the ProB model checker [12] and the ProR requirements tool [9]. But even more important, we provide a seamless integration of these tools with the Event-B modeling environment.

A simple "integration" of Eclipse tools can be achieved by installing them in the same base installation. But the added value of this is little more than running two application next to each other. The real value from integration comes by through the integration of the actual data.

In the following, we will explore the various systems engineering disciplines, starting with the core Rodin platform.

Modeling and Development. Development is the process of "designing and implementing a technical solution". The focus of the core Rodin platform is on the design process, where an abstract specification is refined step by step to add more and more implementation detail. The development must conform to the architecture and support the requirements. This is discussed below.

A central idea behind the development of Event-B was to keep the core of the language minimal. In an industrial context, this approach can be painful, as crucial features may be missing. But these needs have been addressed by the community in the form of additional plug-ins.

It is possible to further use the formal model and generate code from it. A few code generation facilities are available for Rodin. However, it is often more cost-effective to omit this last step and to use the formal model just as the specification [3].

Requirements. While Event-B models can be proven to be consistent, it is not always clear whether they are correct with respect to their requirements. This has been addressed by providing an integration with the ProR requirements tool [10]. ProR is part of the Requirements Modeling Framework (RMF), an Eclipse Foundation project. ProR is using the ReqIF standard for exchanging requirements as the underlying data model. This gives it interoperability with industrial requirements tools.

ProR and Rodin are both Eclipse-based, and therefore, a seamless integration was possible. We developed an integration plug-in [10], which allows linking of requirements and model elements with drag and drop, as well as colour highlighting of model elements in the requirements. The tool is accompanied with

a method, the ProR approach, which ensures consistency between the requirements and the formal model.

Architecture. One approach to architecture is the use of UML. While the Eclipse ecosystem has solid UML support, these tools have not yet been integrated. However, the UML-B plug-in for Rodin [14] provides a facility for generating Event-B models from UML class and state diagrams, thereby providing a tight integration.

UML-B state machines can also be animated and are therefore useful for validating the model's behaviour.

Testing and Validation. While Rodin generates proof obligations for the model, they only allow to check the model's consistency (i.e. with respect to invariants or refinements). But the model must also be validated against the requirements. The ProB validation platform provides a number of mechanisms for this purpose. First, models can be animated — this allows users to inspect the model's behaviour. Further, ProB can also generate tests via the MBT plug-in.

Deployment. At this point, the Rodin ecosystem has little to offer for deployment. Considering how weak the code generation facilities are, this is not that surprising. Eclipse provides technologies for deployment, but these are mainly targeted towards deploying code to application servers — but this differs vastly from what is needed when deploying embedded systems. There is certainly room for improvement here.

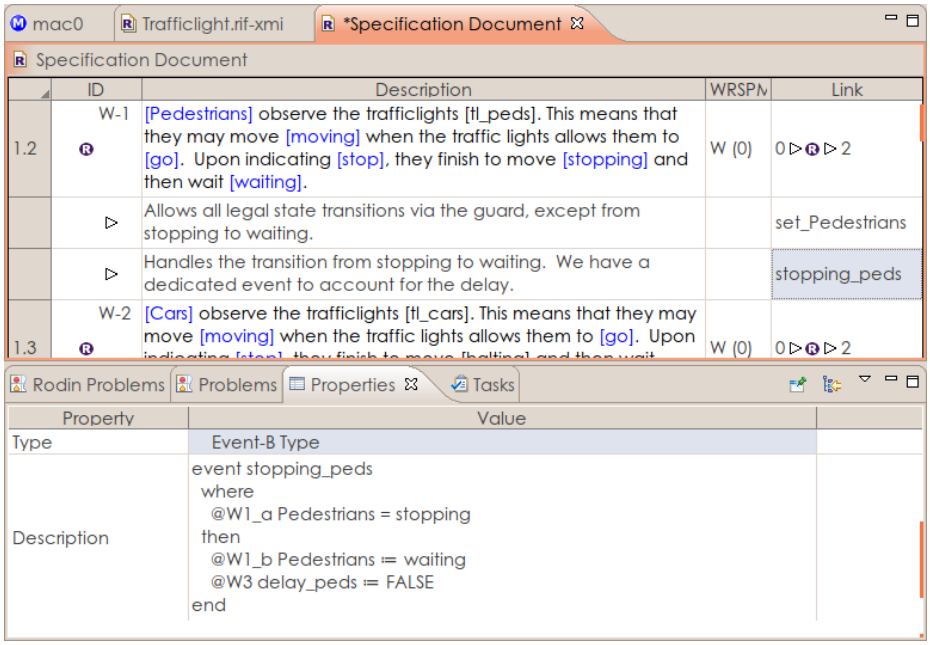
Environment and Project Management. Both, environment and project management are crucial to systems engineering, but not particularly specific to the use of formal methods. Promising tools can be found in the Eclipse ecosystem, but have not been explored specifically in the past. We believe that this is due to the fact that most projects where Rodin has been used were either academic or industrial research projects, but not commercial productions.

One powerful tool for both environment and project management is the Eclipse Process Framework, a customizable software process engineering framework. A useful tool for project management is Mylyn[4], a task and application lifecycle management framework. Mylyn provides tight integration with programming environments, and a corresponding integration with the Event-B modeling environment could be realised.

3.1 Integration of Rodin and ProR

The figure gives an impression on how the integration between Rodin and ProR has been realized, following an approach described in [10]. It shows some requirements of a traffic light system in the upper pane. Each row represents either a requirement or a trace (preceeded by a triangle). Each column represents an

[4] http://www.eclipse.org/mylyn/



Integration of ProR with an Event-B formal model.

attribute, the most prominent one being the requirements text. Some words are highlighted, representing elements that have been modelled formally. The details for the currently selected element ("stopping_peds") are shown in the Properties pane on the bottom. The corresponding model element is shown there, in this case the event "stopping_peds".

Links between the formal model and requirements can be established by using drag and drop. In addition, the system marks links where source or target have changed, thereby allowing the user to re-validate the relationship.

4 Conclusion and Outlook

What we presented in the previous section is not yet a complete tool chain. However, we learned that Eclipse is well-suited as an integration platform for such a tool chain, and that many building blocks already exist.

In our work, we realised that the formal specification takes on a central role in the development process: It is based on the architecture, it realises the requirements, it drives the implementation, it defines, to a degree, the tests, and it aids in measuring the progress for project management.

Work on Rodin and its ecosystem continues. Amongst others, the FP7 ADVANCE project specifically focusses on Rodin to "develop of a unified tool-based framework for automated formal verification and simulation-based validation of

cyber-physical systems". Irrespective of the success of Rodin, it is clear that Eclipse is here to stay in systems engineering.

Last, the recently started ITEA project openETCS [7] aims to use Eclipse-based open technologies and formal methods to develop systems that allow cross-European rail transportation. If this succeeds, it will be a big step forward for Eclipse based systems engineering.

Acknowledgements. The work in this paper is partly funded by DEPLOY and ADVANCE, both European Commission Information and Communication Technologies FP7 projects.

References

1. J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: An open toolset for modelling and reasoning in event-B. *STTT*, 12(6):447–466, 2010.
2. R Balduino. Introduction to openup (open unified process). *eclipse.org*, 2007.
3. D. M Berry. Formal methods: the very idea – some thoughts about why they work when they work. *Science of computer Programming*, 42(1):11–27, 1999.
4. J. P Bowen and M. G Hinchey. Seven more myths of formal methods. In *FME'94: industrial benefit of formal methods: Second International Symposium of Formal Methods Europe, Barcelona, Spain, October 24-28, 1994: proceedings*, page 105, 1994.
5. M. Broy and A. Rausch. Das neue v-modell® xt. *Informatik-Spektrum*, 28(3):220–229, 2005.
6. P. Farail, P. Gaufillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel. The topcased project: a toolkit in open source for critical aeronautic systems design. In *Embedded Real Time Software*, 2006.
7. Klaus-Rüdiger Hase and Jean Koulischer. openetcs: Open source prinzipien für das europäische zugsicherungssystem. In *ZEVrail Tagungsband*, 2012.
8. M. Jastram and A. Graf. Requirement traceability in Topcased with the requirements interchange format (RIF/ReqIF). *First Topcased Days Toulouse*, 2011.
9. M. Jastram and A. Graf. Reqif – the new requirements standard and its open source implementation eclipse rmf. Technical report, Commercial Vehicle Technology Symposium, 2012.
10. Michael Jastram. *The ProR Approach: Traceability of Requirements and System Descriptions*. Inaugural-Dissertation. CreateSpace, 2012.
11. P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2004.
12. M. Leuschel and M. Butler. ProB : an automated analysis toolset for the B method. *International Journal on Software Tools for Technology Transfer*, 10(2):185–203, 2008.
13. D. Riehle. The commercial open source business model. In *Value Creation in E-Business Management*, Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2009.
14. C. Snook and M. Butler. UML-B: formal modeling and design aided by UML. *ACM Trans. Softw. Eng. Methodol.*, 15(1):92–122, 2006.
15. Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods. *ACM Computing Surveys*, 41(4):1–36, October 2009.