

# Practical Systems Engineering

by the Systems Engineering Community

November 11, 2014



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Vision . . . . .	5
1.2	Scope . . . . .	5
1.2.1	Software vs. Systems Engineering . . . . .	6
1.2.2	Standards . . . . .	6
1.3	Tools . . . . .	6
1.3.1	Modularity and Extendability . . . . .	6
1.4	Background . . . . .	7
1.5	License . . . . .	7
<b>2</b>	<b>Tutorial</b>	<b>9</b>
2.1	Overview . . . . .	9
2.2	Tool Installation . . . . .	9
2.2.1	Eclipse . . . . .	9
2.2.2	RMF and Formal Mind Essentials . . . . .	10
2.2.3	Java FX . . . . .	10
2.2.4	RMF-EMF Traceability . . . . .	10
2.2.5	Additional Modeling Components . . . . .	10
2.2.6	Team Support . . . . .	10
2.2.7	Tool Configuration . . . . .	10
2.3	Import Requirements . . . . .	11
2.4	Glossary . . . . .	12
2.5	Data Dictionary with Ecore . . . . .	12
2.5.1	Creating the Ecore Model . . . . .	13
2.5.2	Working with Diagrams . . . . .	13
2.6	Modeling with Papyrus . . . . .	14
<b>3</b>	<b>Case Study</b>	<b>17</b>



# Chapter 1

## Introduction

Requirements Management and Engineering (RE&M) is taught, both in industry and academia. The availability of open source SE-tools, and Eclipse-based tools in particular, created some interest for using those tools for teaching.

As RE&M is often seen as a discipline of Systems Engineering, our scope is systems engineering, with an **initial** focus on requirements engineering.

### 1.1 Vision

The vision of this project is to create:

1. A set of teaching materials that is actively used;
2. Which is embedded in a larger SE context; and
3. Which explicitly focuses on applying RE.

### 1.2 Scope

The scope is the creation of teaching materials, centered around a case study, based on existing methods and tools. This is visualized in Figure 1.1.

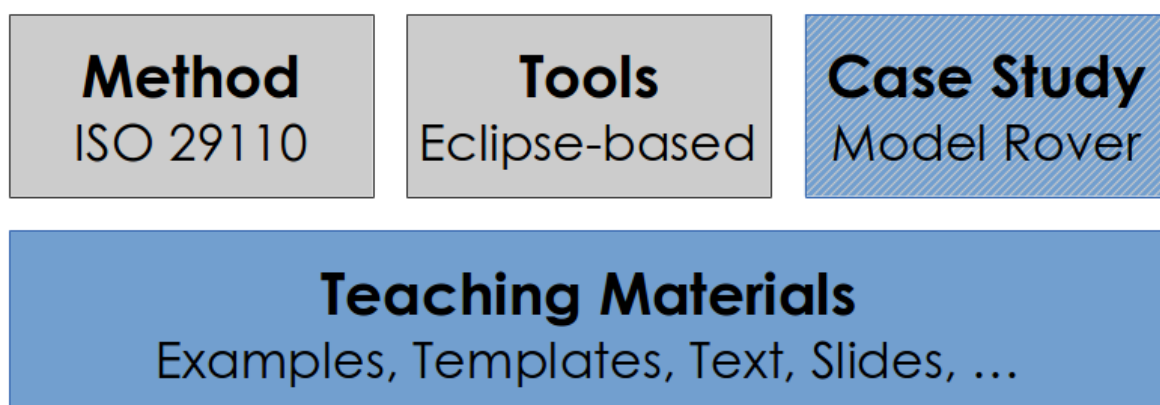


Figure 1.1: Scope of the SE teaching materials

### 1.2.1 Software vs. Systems Engineering

For the purpose of this tutorial, a system has interfaces with other software or hardware, while software is (more or less) stand-alone. Using this definition, we see systems engineering simply as an extension to software engineering, but with interfaces that can be unreliable.

### 1.2.2 Standards

ISO 29110 looks promising as the foundation for the method. Eclipse-based tools in general, and ProR for requirements engineering in particular, will be used. We are currently looking for a suitable case study, ideally using something that already exists. The focus will be on the creation of shared teaching materials.

## 1.3 Tools

A central idea of this project is the use of freely available tools, as we cannot expect students to invest in expensive tools. Tools will be based on Eclipse. Figure 1.2 shows on the left a simplified V-Model, depicting the tools we plan to use.

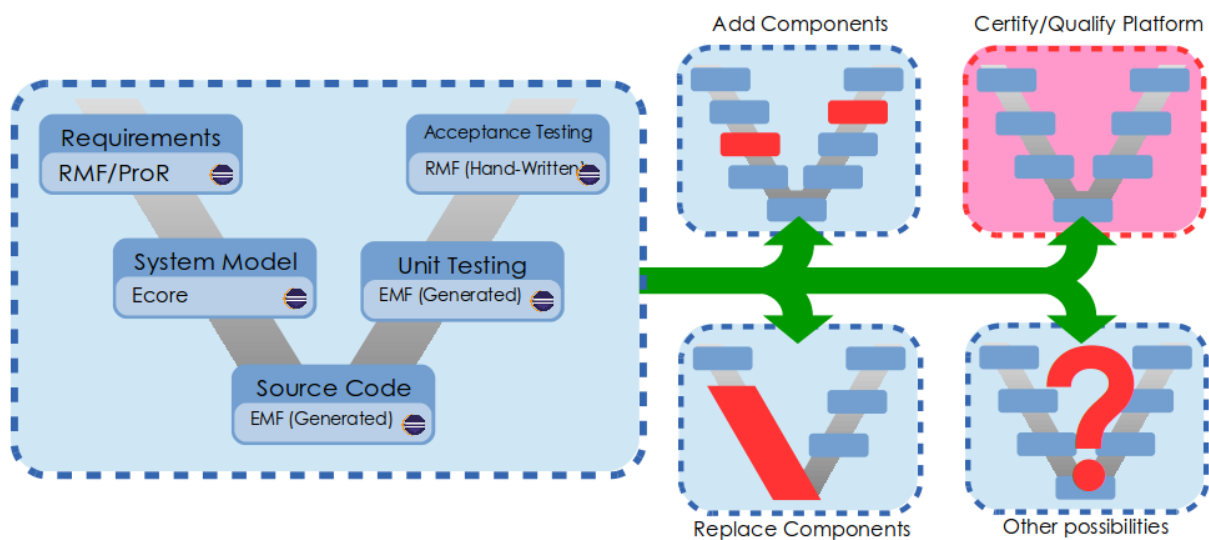


Figure 1.2: Tools used in this course

**Information.** In a “real” project, there would be many more tools and artifacts. We will keep tools and artifacts to a minimum, in order not to overwhelm the students.

As you can see, we plan on building a **minimal, complete, Eclipse-based** software engineering platform. It is loosely organized according to the V-Model.

### 1.3.1 Modularity and Extendability

Figure 1.2 depicts on the right how this toolchain can be adapted to your needs.

Openness helps drastically to make this possible:

**Open Standards** make it possible to replace individual tool components, without disturbing the toolchain as a whole. Open Standards that we use include ReqIF, Java and JUnit.

**Open Software** allows the toolchain to be tailored and seamlessly integrated in a way that is very difficult to do otherwise.

## **1.4 Background**

This project started in July 2014 as a discussion on LinkedIn. Thank you to all contributors!

## **1.5 License**

This content is licensed as Apache 2.0. If you contribute to the corresponding gitHub repository, you implicitly license the content that way.





## Chapter 2

# Tutorial

This chapter contains a mini-tutorial that has been used by Michael Jastram for the TdSE 2014 talk Modellgetriebene Systementwicklung mit Eclipse.

### 2.1 Overview

This tutorial covers the development of a small traffic light system, as shown in Figure 2.1.

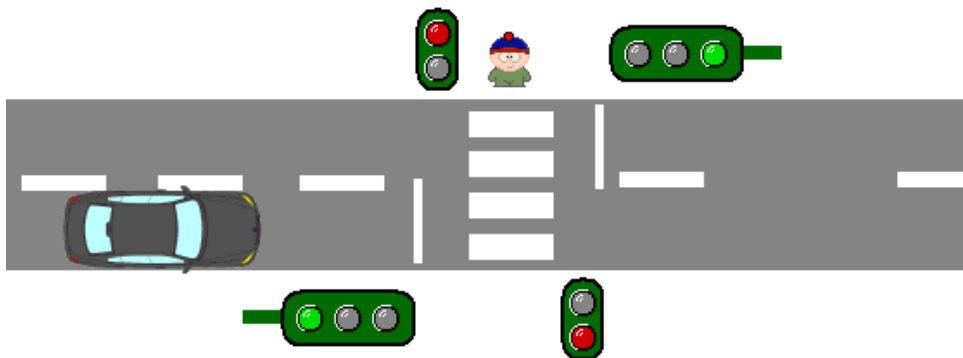


Figure 2.1: We will model a simple traffic light system.

### 2.2 Tool Installation

As of this writing, a complete toolchain is not yet available. The following describes the installation from various components.

#### 2.2.1 Eclipse

The basis for the toolchain are the Eclipse Modeling Tools. Please download for your platform and extract to a convenient location and start it.

**Information.** It may be not a bad idea to start with Polarsys instead, as it already includes Papyrus.

**Warning.** On Linux, please edit eclipse.ini and add the following parameter **at the top (two lines)**:

```
—launcher.GTK_version  
2
```

### 2.2.2 RMF and Formal Mind Essentials

Next install the RMF (requirements) tools, but the repackaged version from Formal Mind:

- Use this update site: <http://update.formalmind.com/studio>
- Unselect “Group items by category”
- Select **only** “Formal Mind Studio (Feature)”
- Complete the installation.

**Information.** The software is currently not signed, which will generate a warning. Please continue with the installation, in spite of this.

### 2.2.3 Java FX

If you want to use rich text in requirements, you need support for Java FX. Follow these steps:

- Use this update site: <http://download.eclipse.org/efxclipse/runtime-released/1.1.0/site>
- Optional: Unselect “Group items by category”
- Select **only** “Runtime Bundle Collector Feature”
- Complete the installation.

### 2.2.4 RMF-EMF Traceability

In the context of a public research project (itea openETCS), a traceability plug-in for connecting arbitrary EMF models has been developed.

- Use this update site: <http://openetcs.ci.cloudbees.com/job/openETCS-tycho/lastSuccessfulBuild/artifact/tool/bun>
- Unselect “Group items by category”
- Select **only** “ProR Tracing Feature”
- Complete the installation.

### 2.2.5 Additional Modeling Components

You can install additional components for modeling via **Papyrus via Help | Install Modeling Components**. For this tutorial, useful components include:

**Ecore Tools.** Support diagram notation for Ecore models.

**Papyrus.** Supports UML and SysML.

We had some problems with installing the Ecore Tools. If you cannot see a diagram in Section ??, then follow these steps:

Install software from this update site: <http://download.eclipse.org/ecoretools/updates/releases/2.0.1/luna>  
Select Ecore Diagram Editor

### 2.2.6 Team Support

Eclipse supports a number of team environments. We recommend the installation of the egit plugin, allowing to work with git repositories. The installation is described in the formalmind Studio Handbook.

### 2.2.7 Tool Configuration

We recommend to switch to the ProR perspective, to get started.

## 2.3 Import Requirements

Typically, you already have requirements available in some form. ProR includes a simple CSV-Importer that allows you to import existing requirements. Follow these steps:

- Create a new Project via **File | New | Project... | General | Project**
- Call it `tdse-1`
- Create a new Requirements Model via right-click on the project, selecting **New | Reqif10 Model**
- Call the Model `Trafficlight.reqif`
- Import the .csv file via **File | Import | formalmind Studio | CSV**
- Create a mapping for the two columns to String attributes, as shown in Figure 2.2

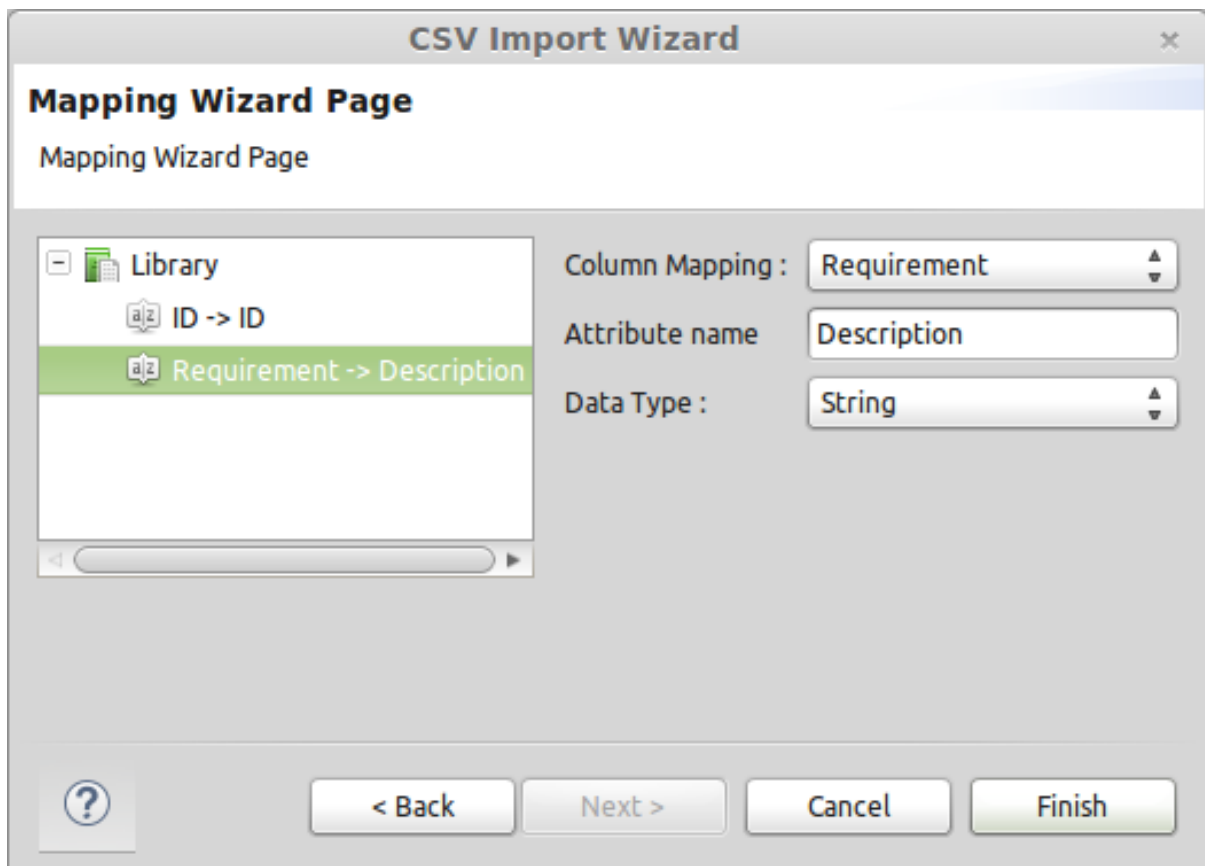


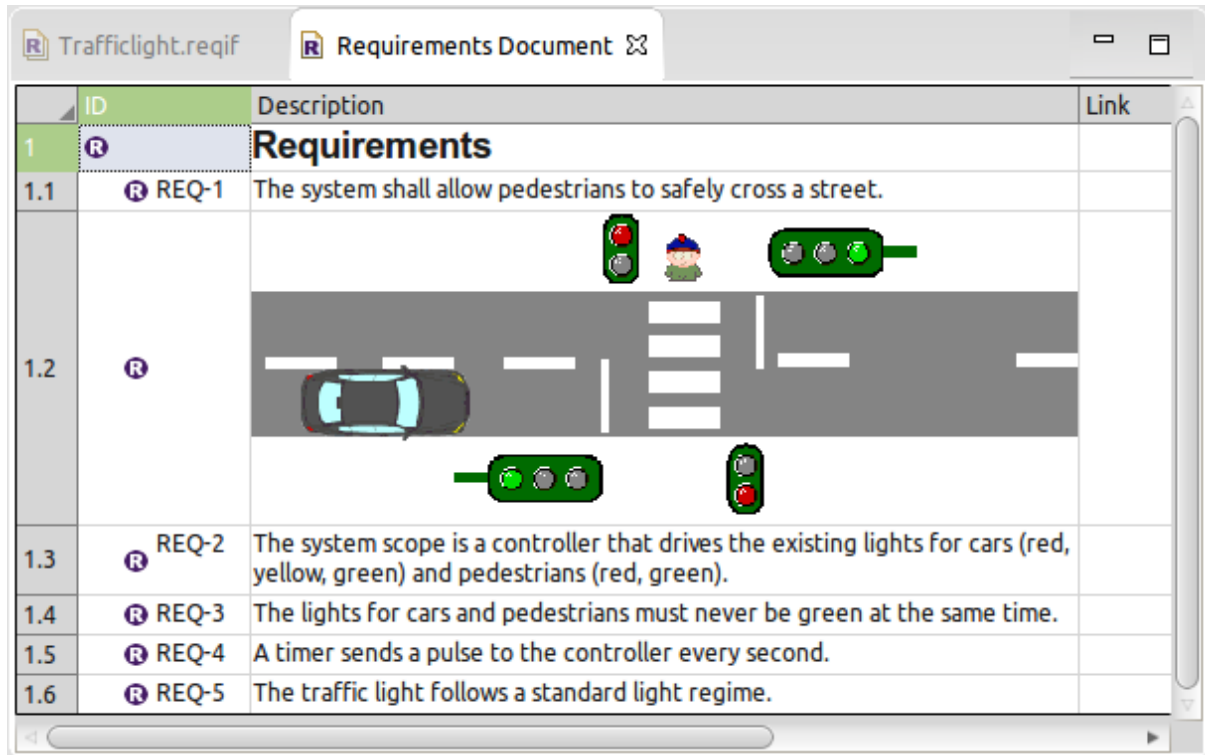
Figure 2.2: Result: The requirement is now a sibling of the chosen requirement.

After the import, the new requirements have been added to the existing requirements specification. There are a number of recommended improvements, for instance:

- Add a `SpecObjectType` for Headlines and configure the Headline Presentation, so that you can structure the text
- Once you create headlines, you can arrange requirements as child elements (instead of siblings) under them.
- You can create an information `SpecObjectType`, using XHTML and no IDs. Use one of these to insert Figure 2.1 into your specification (trafficlight.png).

- Configure the ID Presentation to automatically create IDs for requirements, and center-align the ID.
- Use the ID as a label (if available) by adjusting the Label Configuration.

The resulting specification is shown in Figure 2.3.



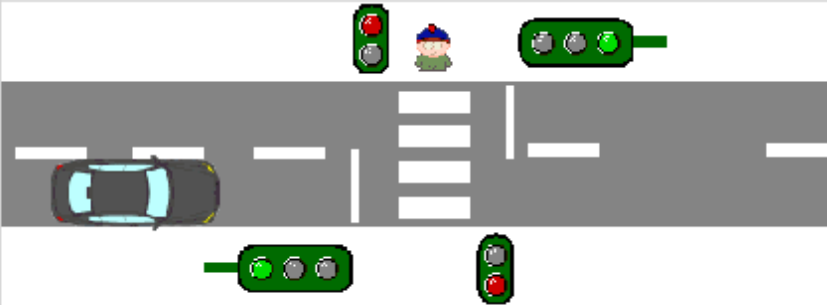
ID	Description	Link
1	<b>Requirements</b>	
1.1	REQ-1 The system shall allow pedestrians to safely cross a street.	
1.2		
1.3	REQ-2 The system scope is a controller that drives the existing lights for cars (red, yellow, green) and pedestrians (red, green).	
1.4	REQ-3 The lights for cars and pedestrians must never be green at the same time.	
1.5	REQ-4 A timer sends a pulse to the controller every second.	
1.6	REQ-5 The traffic light follows a standard light regime.	

Figure 2.3: The spec after completion of all steps so far.

## 2.4 Glossary

A glossary helps keeping track of terminology. In this section, the glossary management from formalmind Studio is introduced, which supports color highlighting in the requirements text.

Note that this kind of glossary is a “dead end”, in the sense that it cannot be used beyond its purpose. Contrast that with a model-based data dictionary, as described in Section 2.5.

The glossary is kind of cumbersome to configure. Therefore, we included a correctly configured Sample Project. Note that you need both the Highlighting and Keyword Highlighting presentations, in that order.

Figure 2.4 shows the glossary, and its application to the requirements, which have been rewritten to use the terminology of the specification.

In the screenshot, REQ-3 is being edited. This results in the word *green* being underlined in red, indicating that it is a recognized glossary entry. The syntax highlighting disappears when not in edit mode. Square brackets make a glossary term explicit. If a term is marked that way that is not in the glossary, then it is shown in red.

## 2.5 Data Dictionary with Ecore

Ecore is the modeling language of the Eclipse Modeling Framework. It has some similarities to UML Class diagrams, and is therefore well-suited for creating a precise data model. It has the following advantages:


			Term	Description
			1	<b>Glossary</b>
1	R	Requirement	1.1	R PedLight
1.1	R REQ-1	The system	1.2	R CarLight
1.2	R		1.3	R red
			1.4	R yellow
			1.5	R green
			1.6	R Tick
				An external regular trigger with a frequency of 1 Hz.
1.3	R REQ-2	The system scope is a controller that drives the existing [CarLight] ([red], [yellow], [green]) and [PedLight] ([red], [green]).		
1.4	R REQ-3	[CarLight] and [PedestrianLight] must never be green at the same time.		
1.5	R REQ-4	A timer sends a [Tick] to the controller every second.		
1.6	R REQ-5	The traffic light follows a standard light regime.		

Figure 2.4: Glossary Management in action.

**Easy to learn.** Especially if you already know class diagrams, you should be able to quickly learn Ecore.

**Code generation.** EMF allow the generation of Java code from Ecore models. You can even generate a GUI based on a tree view.

**Test stub generation.** EMF allows the generation of test code stubs, making it easy to cover the unit test level.

On the other hand, it has its limitations. In particular, it is not really possible to model dynamic aspects of the system.

**Warning.** While it is possible to mix this approach with the glossary management described in Section 2.4, we do not recommend it, as it would lead to redundancy. Redundancies should be avoided (DRY-principle: Don't Repeat Yourself).

### 2.5.1 Creating the Ecore Model

We recommend to create a new Ecore Modeling Project via **File | New | Project... | Eclipse Modeling Framework | Ecore Modeling Project**. This way, everything will be properly configured for code generation and other cool stuff. The model we use is shown in the right pane of Figure 2.5.

As you can see, the editors are arranged so that the requirements editor and the Ecore editor are visible at the same time. This is necessary, as links are created by dragging model elements from the Ecore model onto the requirements.

**Information.** Linking via Drag and Drop is a feature taken from the openETCS project, where it is documented.

We provided a preconfigured Sample Project, that allows annotating traces, as also shown in Figure 2.5 (the second link of REQ-2).

### 2.5.2 Working with Diagrams

Some people prefer diagrams to the tree-view shown in Figure 2.5, and diagrams can make communication easier. If you installed the tool as described in Section 2.2, then you can create a diagram from the Ecore

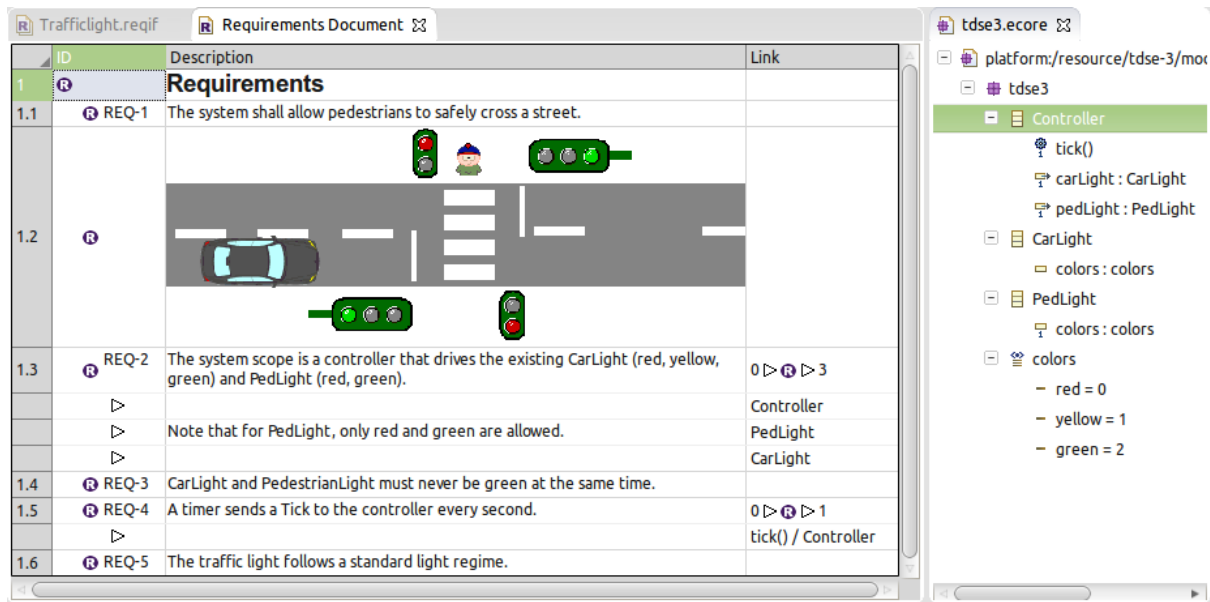


Figure 2.5: On the left the requirements with links into the Ecore-based data model, shown on the right.

model as described here. Diagram and model will be synchronized, but it's possible to only show a subset of the model elements in the diagram.

A diagram should already have been created upon project creation. You can open it by going to the Model Explorer or Project Explorer, and opening the .ecore model by clicking on the

+

to the left of the file name. it should show the package name (tdse3), and upon opening it again, it should unveil "tdse3 class diagram". Doubleclicking should open the diagram editor, which will be empty, except instructions on how to add elements.

By dragging elements from the project explorer into the diagram area, we end up with the editor, as shown in Figure 2.6. Elements and labels can be customized as one sees fit. Changes to the model, including the creation of new elements, will be reflected in the tree view of the Ecore model as well. Changes to the Ecore model will be seen here, but newly created elements will not appear on the diagram by default. They have to be added manually.

- Right-click on the .ecore file and select **Initialize Ecore Diagram...**
- The correct .aird file should already be selected as the

## 2.6 Modeling with Papyrus

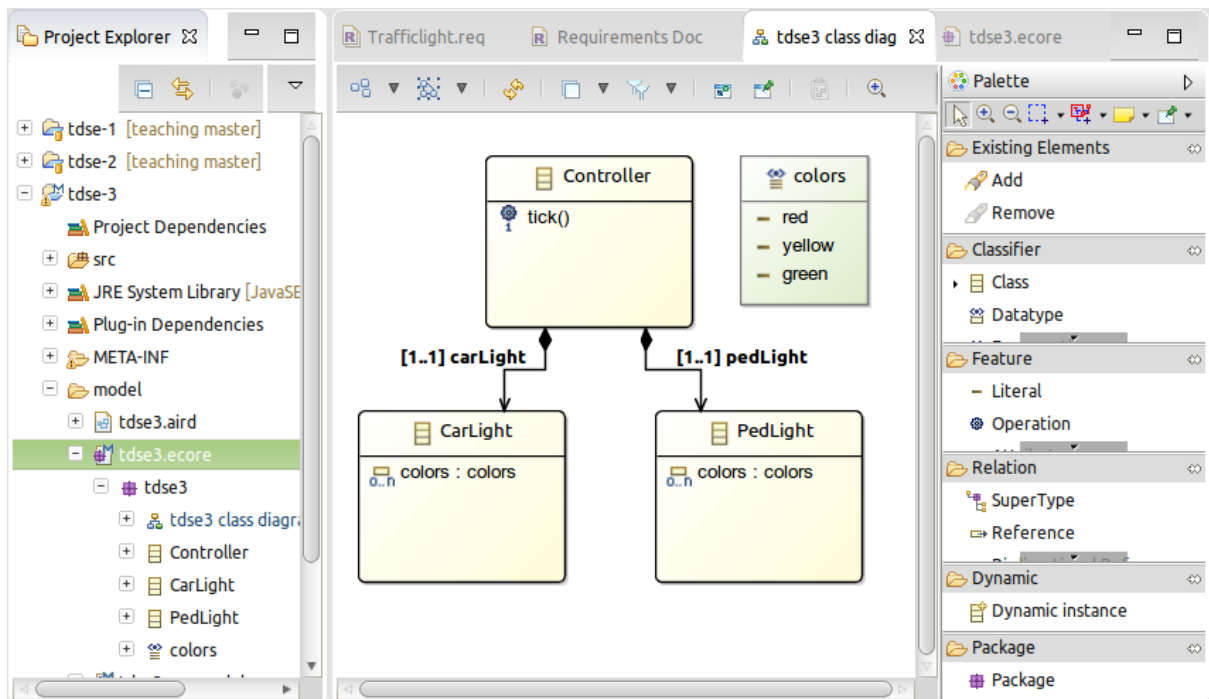


Figure 2.6: The diagram has been created by dragging elements from the Project Explorer (left) to the drawing area (middle). New elements can also be created by using the pallet on the right.





## Chapter 3

# Case Study

We have not decided on a case study yet. Candidates so far are:

**Coffee Maker.** A long-time favorite, and there are at least three available

**FAA Isolette.** This is a complete example from a safety-critical domain.

**Rover.** This one is driven by Gaël Blondelle from the Eclipse Foundation. On the plus side, it's great for the classroom, as the hardware is cheap. But in contrast to the others, there is nothing there yet.

Teaching Materials can be made available via relative links.