Specification of Embedded Systems
Summer Semester 2020

Session 1
Motivation – MBSE – SysML – Requirements Modelling

Jan Peleska
peleska@uni-bremen.de
Issue 1.1
2020-04-27

Chapter 1

Preface

In this document, an introduction into the course **Specification of Embedded Systems** is given.

Recall that a system specification describes the desired properties of a system to be developed. Spacifications can be **implicit** by stating desired properties, for example, using logical formulas. Alternatively, specifications can be **explicit** by providing a **model** of the system to be developed.

SysML

In this course, we will follow the model-based approach and introduce the **Systems Modelling Language SysML** [3] which is based on the well-known **Unified Modelling Language UML** [2]. While UML has a strong focus on software development, SysML extends this focus to integrated HW/SW systems that may in turn be integrated in larger mechanical objects that may be connected with each other over communication networks and/or mechanical devices. Such systems are usually denoted by **cyber-physical systems**.

Overview

This document gives an overview over this course and introduces requirements as a first modelling artefact. Moreover, the Papyrus SysML modelling tool is introduced. The document is structured as follows.

- In Chapter 2, the concept of model-based systems engineering is introduced, and related terms are explained.
- In Chapter 3, the Papyrus tools is introduced which will be used in this course for creating SysML models.
- As a first step towards practical modelling, the representation of requirements in a model is explained in Chapter 4

- In Chapter 5, an overview over the whole course is given.
- Chapter 6 refers to relevant literature.
- In Chapter 7, questions and exercises related to the material presented in Session 1 are listed, for you to work on in the first week of the semester.

Contents

List of Figures

Chapter 2

Motivation: Systems Engineering

2.1 Some Basic Terms and Definitions

Model-based systems engineering

This course is about **systems engineering (SE)**, which is the engineering discipline focussing on systematic system development. A sub-discipline of systems engineering is **model-based systems engineering (MBSE)**, where system development is guided by a series of formal **models**: just like a house is built from construction plans, complex systems consisting of mechanical parts, electronics, and software should be built from a well-defined reference specification.

SysML

We will introduce the well-known modelling formalism **Systems Modelling Language (SysML)** [3] for learning and applying the MBSE approach. SysML is a so-called **profile** of the UML [2], that is a specialisation of the latter for system development (as opposed to pure software development which can also be done using UML).

The term **formal** states that the **semantics** of models should be unambiguous, at least to experts of the modelling formalism. To recap the term 'semantics', recall that a model, just like a programming language, has a formal **syntax** specified by the modelling formalism. Every syntactically well-formed model should have a well-defined semantics, just like a computer program that compiles correctly, so that we can expect well-defined program execution behaviour. Semantics has two aspects.

Semantics

- **Static semantics** explains the meaning of the model structure, e.g.,

"this model consists of two sub-components communicating over a single shared variable interface". Moreover, it specifies all rules that can be checked without "executing" the model, such as usage of admissible types in expressions (e.g. you cannot add an integer to a string).

- **Behavioural semantics**[1] explains how input data to or stimulations of the model are transformed step by step into output data or reactions of the model.

Which aspects of a system to be developed should be covered by a model? To answer this questions, we look at the typical sub-division of **system properties**.

Modelling aspects

Structural properties specify the static decomposition of a system to be developed: components, interfaces, hardware structure.[2]

Behavioural properties (also called **functional properties**) specify the transformation of data and other dynamic reactions, as explained for the term 'semantics' above.

Non-functional properties describe all aspects of a system that are neither behavioural nor structural. Important examples are

- safety,
- reliability,
- availability,
- security,
- maintainability,
- usability,
- fault-tolerance,
- quality (e.g. requirements conformance),

but many more examples exist.[3]

[1] also called **dynamic semantics**

[2] Observe that since we are talking about *system* development, it's not only about software, but about software integrated into controllers integrated into surrounding HW like cars, aircrafts, trains, space ships, robots ...

[3] see `https://en.wikipedia.org/wiki/Non-functional_requirement` for a (still non-comprehensive) list of further non-functional requirements

Relevant modelling formalisms always support the specification of structural and behavioural properties. When it comes to non-functional properties, they differ quite a lot. During this course we will see that SysML is a modelling formalism which covers a rather wide variety of non-functional properties, which is one of the unique selling points of this formalism.

Model-driven

To emphasise that models should be the main drivers of a system development undertaking, the terms **model-driven architecture (MDA)**, **model-driven design (MDD)** are frequently used. In the same sense, the term **model-based** is connected with testing, code generation and other activities of a development process.

PIM and PSM

Typically, complex systems will be built from more than one model: a **platform-independent model (PIM)** specifies the expected properties of the system in a way which is independent on specific choices of hardware, operating systems etc. A **platform-specific model (PSM)** specifies how the general properties are mapped to concrete hardware, operating system, interfaces etc.

Example 1. Airbus is using the MBSE approach to specify the electronic service functions of the aircraft cabin (e.g. cabin pressure control, air conditioning, crew telephone system, passenger address system and hundreds more) as a library of PIMs (one for each service function), so that a uniform behaviour of these services over all members of the aircraft family (A318, A320, A340, A350, ...) is enforced.

The concrete **instantiation** of a service in a specific aircraft type is then represented in a PSM which has the same behaviour as specified in the associated PIM, but refers to concrete interfaces and reduced or enhanced service options. For example, control computers in the A350 aircraft communicate over AFDX which is a variant of the UDP/IP protocol over Ethernet[4], while computers communicate using the ARINC 429 protocol[5] in the A340 aircraft.

To provide another example, cabin lights in the A320 aircraft can be controlled in only two seat row columns (left and right of the single aisle), while there are three columns of light to control in A340 or A350 aircrafts that have two aisles.

These aircraft-specific **refinements** or instantiations of a PIM are represented in PSMs. □

[4] https://en.wikipedia.org/wiki/Avionics_Full-Duplex_Switched_Ethernet
[5] https://en.wikipedia.org/wiki/ARINC_429

Models come with different levels of abstraction. For example, a processor specification model could be represented in VHDL[6] or SystemC[7], which would look just like a computer program to a system modeller, but is really an abstraction from the perspective of chip designers. In the same sense, a C program is a model of the assembler code to be executed on a computer. Therefore, we consider computer programs also as models, but on a lower level of abstraction.

Programs are models

Please read [1, Chapter 1] about a more elaborate introduction to systems engineering.

Further reading

2.2 The Benefits of MBSE

Models of a system to be developed are elaborated for many reasons, we list some of the important ones here.

- A model library is the ideal means of storing knowledge about systems, products, or business processes.
- Finding bugs early during the development life cycle is less expensive than fixing bugs in a system that has already been deployed. Modelling the desired behaviour of a systems offers the opportunity to detect logical misconceptions already in the model, without having created a single piece of "real" hardware and without having programmed a single line of code.
- Models can be used as the source for automated code generation, if they are sufficiently detailed. This makes conventional coding of embedded systems superfluous.
- Models can be used to derive relevant test cases and test procedures for the system under development, so that tests have no longer to be programmed manually.
- Models help to trace requirements to implementation and verification artefacts in an automated way.

We do not discuss this any further here, because there is an exercise question about this for you to work on.

[6] `https://en.wikipedia.org/wiki/VHDL`
[7] `https://en.wikipedia.org/wiki/SystemC`

Chapter 3

Developing SysML Models with Papyrus

In this course, we will use the **Papyrus** tool for designing SysML models. Papyrus is available free of charge, it is open source, and it is well-known to support UML/SysML in a very comprehensive way, conforming to the standards [2, 3]. Some commercial tools have better usability, you will find that the workflow with Papyrus is sometimes a bit cumbersome. However, it does the job, and for the purpose of this course the conformance to the standards is most important.

You need to install this tool on your computer like this:

1. Download and install the **Eclipse IDE 2020-03 for Java Developers** `https://www.eclipse.org/downloads/packages/`
2. Open the Eclipse IDE, and use pull-down menu Help → Eclipse Marketplace ...
   - In the Search rider, insert *Papyrus*; this will give you 3 hits. Select Papyrus SysML 1.6 1.0.0 and install this plugin.

I have prepared three videos showing how to install and perform the first steps with Papyrus SysML in the Eclipse IDE, you can see this under 3 videos

- `https://youtu.be/NNRMMjb_BiQ` : This video tells you how to install Papyrus as an Eclipse plugin and create your first SysML modelling project, structured into packages.

- `https://youtu.be/VTmxlG7E-_4` : This video shows you how to create requirements in a SysML model and how to create requirements diagrams in order to structure the (potentially large) collection of atomic requirements. This is the first part of two videos.
- `https://youtu.be/ekiHlkaX0WU` : This is the second part of the video about requirements modelling.

Chapter 4

Requirements

The professional way of approaching a complex systems development undertaking is to first capture its **requirements**. A requirement is a specification of a desired system property. In today's industrial practice, requirements are written in natural language associated with informal diagrams, to facilitate the communication between customers and system development experts. Research communities advocate a formalisation of requirements, for example, using temporal logic[1], but this is not yet considered as state of practice.

Requirements should be part of the model

In any case, requirements should be well-structured and describe structural, behavioural, and non-functional properties of a system to be developed. Most modelling formalisms consider requirements as artefacts to be compiled separately, and serve as *inputs* to the modelling phase. The SysML modelling language has the advantage, that requirements are considered as part of the modelling language, so that they are represented as a (slightly informal) part of the model. The formal modelling elements (block diagrams, state machines, activity charts, we will learn about this in the coming weeks) can then refer to requirements, so that it becomes clear which formal parts of a model are responsible to represent a specific requirement.

In SysML, a requirement consists of a pair (id, text), where id is a unique identifier and text is a textual description of the requirement. The description may contain links to informal graphics.

Containment link

Requirements should be decomposed until they are **atomic**, that is, until they represent an indivisible property that does not contain any case dis-

[1]This is, for example, explained in the course **Theory of Reactive Systems** held by Wen-ling Huang in this semester, see `http://www.informatik.uni-bremen.de/agbs/jp/papers/trs_script.pdf`

tinctions. Non-atomic requirements can be used to structure the collection of atomic requirements. To depict the decomposition of requirements in a diagram, the so-called **containment link** is used. In Fig. 4.1, a requirements diagram related to Exercise 7.2.1 is shown. The "cross hair" end of a containment link is the higher-level composite requirement, and the undecorated end of the link attaches a lower-level requirement which is part of the higher-level one.

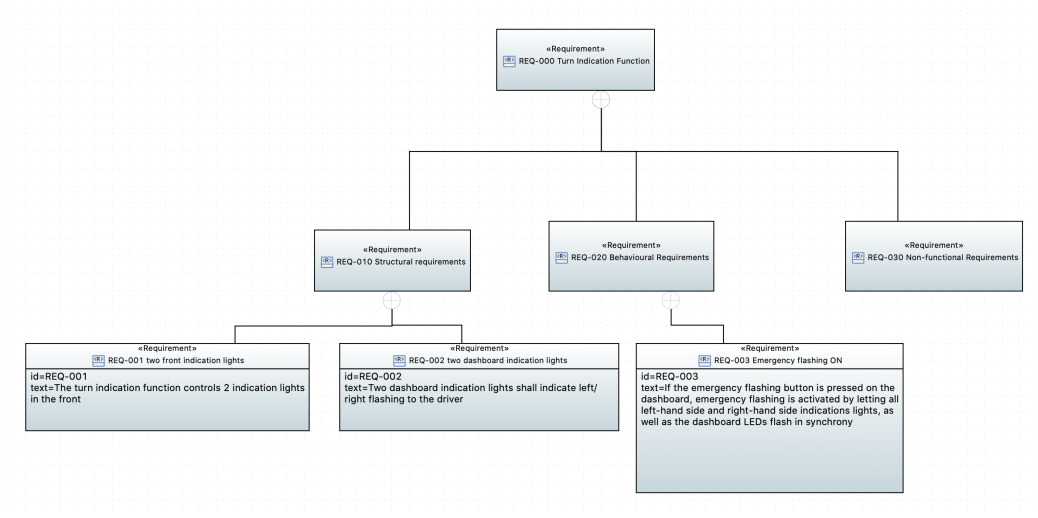

Figure 4.1: Requirements diagram with 3 atomic requirements for the turn indication controller.

Model creation

The videos referenced above describe how a SysML model is created with the Papyrus tool. A model should always be structured into **packages** imposing a sub-structure. For the start, the only package we need is requirements, we will add more packages in the next session.

Further reading

Please study Sections 13.1 — 13.4 in [1], where further information about the specification of requirements in SysML is given.

Chapter 5

What happens next? The Session Structure

This course is structured into the following sessions. Note that for obtaining only 3 ECTS credit points, it is only required to learn Session 1 — Session 4. From Session 5 on, C programming skills are required, but template programs will be handed out to start with.

Session 1. Motivation – MBSE – SysML – Requirements modelling

From-to: 2020-04-20 — 2020-04-30

Session 2. Modelling interfaces and structure – SysML ports – Blocks – Block definition diagrams – Internal block diagrams – Context specification – Specification of physical units

From-to: 2020-05-04 — 2020-05-15

Session 3. Modelling behaviour, Part I: operations – state machines – events – timing conditions

From-to: 2020-05-18 — 2020-05-29

Session 4. Modelling behaviour, Part II: activities – detailed requirements tracing

From-to: 2020-06-02 — 2020-06-12

Session 5. Model analysis: textual SysML model representation in XMI-format – LIBXML2 – parsing the XMI model – static model analysis

From-to: 2020-06-15 — 2020-06-26

Session 6. Code generation I: domain framework – hardware abstraction layer – signal interfaces – shared variable interfaces – state machine to code transformation

From-to: 2020-06-29 — 2020-07-10

Session 7. Code generation II: Activity to code transformation

From-to: 2020-07-13 — 2020-07-24

Chapter 6

Literature

As mentioned above, SysML is a profile of the UML, so for fully understanding SysML, an understanding of the UML is important as well.

- The reference specification for the UML is [2] and can be downloaded from `https://www.omg.org/spec/UML/2.5.1/PDF`
- The reference specification for the SysML is [3] and can be downloaded from `https://www.omg.org/spec/SysML/1.6/PDF`

Though being (of course) comprehensive, these standards are quite hard to read. A very good book introducing and motivating the SysML is [1]. It also explains technical details differently from the standards – this may facilitate understanding complex modelling concepts. The book is available online for you under `https://suche.suub.uni-bremen.de/peid=B79963566`.

Chapter 7

Questions and Exercises

7.1 Questions

7.1.1 Benefits of Model-based Testing

Study the paper [4] which has also been uploaded to Stud.IP and make a list of benefits claimed by the author to come from model-based testing. Explain the term **traceability** and describe why traceability is so important for safety-critical systems.

7.1.2 The Importance of Standards

Which standards are mentioned in [4] and [1, Chapter 1]? How do these texts explain the importance of standards?

7.2 Exercises

7.2.1 Requirements of the Turn Indication Model

As part of the material for Session 1, you find file `turn-indication-controller.pdf` which describes (a part of) the functionality involving the left-right indication lights in cars in natural language. Please study this text and perform the following activities.

1. Create a new Papyrus-SysML model called

`TurnIndication<Your Name>`.

The name-suffix helps me to avoid model name clashes when loading your models into my Eclipse workspace.

2. Create a package requirements[1]

3. Analyse the natural-language specification in file `turn-indication-controller.pdf` and identify atomic requirements from this text.

4. Create a SysML requirement in this package for each of the requirements you have identified in the previous step. Make sure that the requirements are properly specified with id and descriptive text.

5. For structuring the collection of requirements, create suitable names for non-atomic requirements collections, and draw several requirements diagrams showing the structured hierarchy of requirements. The structure should at least contain

   - structural
   - behavioural
   - non-functional
   - safety-related

   requirements

In the diagrams, only requirements and their containment links need to be shown. The additional relationships described in [1, Table 13.2] are not needed for the moment, they will be discussed at a later stage.

[1] Traditionally, package names are in small letters, no blanks, underscores, or special characters.

Bibliography

[1] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML, Third Edition: The Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2014.

[2] Object Management Group. OMG Unified Modeling Language (OMG UML), version 2.5.1. Technical report, OMG, 2017.

[3] Object Management Group. OMG Systems Modeling Language (OMG SysML), Version 1.6. Technical report, Object Management Group, 2019. http://www.omg.org/spec/SysML/1.4.

[4] Jan Peleska. Model-based avionic systems testing for the airbus family. In *23rd IEEE European Test Symposium, ETS 2018, Bremen, Germany, May 28 - June 1, 2018*, pages 1–10. IEEE, 2018.