Towards a Curriculum for Model-Based Engineering of Embedded Systems

Bernd-Holger Schlingloff

Humboldt Universität zu Berlin, Institut für Informatik
Fraunhofer Institut für offene Kommunikationssysteme FOKUS, Berlin
hs@informatik.hu-berlin.de

Abstract: Even though the theory and industrial practice of model-based techniques has reached a certain maturity, there seems to be no consensus of what should constitute a curriculum in this field. This position paper reports on several classes we gave on the bachelor and master level, as well as for industrial participants. We argue that there is a strong need for lightweight tools and platforms which allow to teach all relevant concepts, yet are easy to install and to use for beginners.

1 Introduction

An embedded system is a computational system which is a fixed part of a technical system. Model-based engineering is the craft of constructing technical systems from abstract models of the system's structure and behaviour. Within the last ten years, model-based engineering has become the preferred design methodology for embedded systems in automotive, aerospace, and other domains. There is a high industrial demand for skilled experts in this field. However, although a substantial body of knowledge in the field is available, there seems to be no consensus amongst academic teachers as to what constitutes the common core which should be taught at universities.

This paper is a first attempt to remedy this situation; it should be seen as a basis for discussion rather than a final proposal. We review the current status of the subject at different universities, and propose a curriculum which has been taught several times already. We discuss our experiences and the student's feedback, and give an outlook on further activities.

2 Model-Based Engineering of Embedded Systems

The subject arises from the intersection of two fields: Modelling in computer science, and embedded systems design. Both of these fields have been investigated for some time, and there is an extensive scientific literature on these subjects available [GHPS13, BJK+05, ZSM11]. For embedded systems, several universities have begun to start bachelor and

masters programs [UFra, UFrb, TUE, UPe]. Most of these programs are composed from classes taken from the computer science and electrical engineering curriculum. Model-based engineering is taught differently in different engineering disciplines: Mechanical engineers have, e.g., classes on simulation with Simulink, electrical engineers learn, e.g., control-theoretic modelling of dynamic systems, and computer science students attend classes, e.g., on software modelling with UML.

Combining these threads poses the challenge of selecting a coherent and consistent subset which nevertheless fits into the alloted time frame. Clearly, the union of all relevant topics could easily fill a complete bachelor and masters degree program. However, it is not agreed whether such a specialized program would be accepted by university administrations and students. Therefore, we constrain ourselves to the discussion of one module of two or four hours per week, with additional lab classes and maybe a specialized continuation in a subsequent semester.

Such a module could be part of any appropriate bachelor or master program in the engineering sciences. We do not want to restrain ourselves to a particular discipline such as electrical, mechanical, or software engineering. That is, we assume that the module shall be adapted to the specific prerequisite knowledge of the audience. For example, most computer science students have heard about UML in their second year, but have only vague knowledge of differential calculus. Mechanical engineers know about finite element methods, but have not heard about code generation techniques. Electrical engineers have learned about electronic circuit analysis, but may not be proficient in systematic test design. Therefore, the curriculum of the module needs to be adjusted to the department and prior skills of the participants.

Via the current Bologna process for the comparability of higher education qualifications, European degree programs are harmonized to uniform bachelor and master studies. Here, the proposed module could be allocated, e.g., as an interdisciplinary course on the bachelor level, which can be credited in several departments. Specialized extensions can be offered for different areas on the masters level.

3 A Proposed Curriculum

A module covering the subject under discussion must convey the most important principles of both embedded systems engineering and model-based design. Qualification goals are that participants are able to design and implement a reasonably complex embedded system. Thus, the module should have a strong emphasis on concrete examples. Furthermore, since the subject is still evolving, students should get an impression on current and future trends in the field. Therefore, they should not only acquire profound skills in a particular modelling language, but also learn different modelling paradigms, and meta-modelling concepts with which to link these paradigms. Similarly, they should not only experiment with the current computational and physical hardware on which present-day embedded systems are based, but also learn about future developments, e.g., in the cyber-physical domain. Students should be able to estimate economic and social impacts of the technol-

ogy. Therefore, productivity of the design method and quality of the resulting products are to be considered frequently.

Subsequently, we sketch a curriculum which has been taught twice at the Humboldt Universität zu Berlin, in various Erasmus-lectures at the University of Swansea, and in part at international summer schools in Hanoi and Thessaloniki.

1. **Basic definitions**
   This part contains the necessary foundations; it answers questions like "what is an embedded system", "which software engineering processes exist", "what are the basic constituents of embedded hardware", "what is the present and future market importance of embedded systems", "which main design challenges exist", etc. We also show some lab prototypes of embedded systems together with their design models in order to give a glimpse ahead.

2. **Requirements engineering**
   Here we discuss issues like the difference between user specification and technical specification, present some examples of industrial requirements documents, and discuss methods for requirements elicitation and -management. Students are also introduced to stakeholder analysis in order to understand how to capture the socio-economic aspects of a system under design.

3. **Systems modelling**
   A major instrument in dealing with the design complexity is systems engineering. Students are being introduced to concepts of SysML in order to model items like the system's lifecycle, deployment, variability and product line design. Use case diagrams are used to describe the human-machine interface and interactions with other systems. A goal is to teach the students how to develop a holistic view of a system in its intended environment.

4. **Continuous modelling**
   This part is a crash course in control theory. It covers the basic mathematics to describe the behavior of dynamical systems with inputs, including some linear differential equations, as well as block diagrams and continuous modelling tools such as Simulink or Scilab. Prototypical examples are an inverted pendulum as well as a two-dimensional cat-mouse race. Students learn how to clearly describe the system's boundaries, in order to distinguish between environment model and system model.

5. **Discrete modelling**
   Here, we introduce the classical models of software engineering: structural and behavioral diagrams for the static and dynamic description of systems. We emphasize that there is a large variety of well-established modelling languages, each in its own right. For practical demonstration purposes, we focus on UML class and component diagrams, communication and sequence diagrams, and state machine diagrams. Students learn how to come up with a first conceptual model, and how to refine this abstract model to a concrete one in various steps.

6. **Code generation**
   This part describes techniques to generate Java or C code from various models. We describe "human readable" and "machine optimized" code generation transformations for both block-diagram and state-transition models. Furthermore, we discuss how to build an actual running prototype by linking abstract events to concrete IO ports. An example is a blinking LED on an experimental board which can be interrupted by a push button. The main message for students is that "the theory actually works", i.e., that model based design is not a theoretical possibility but a practical engineering method.

7. **Meta-modelling**
   We introduce meta-modelling concepts as a generalization of code generation techniques. Whereas a code generator is just a model-to-text transformer, general model transformations are able to link different types of models. We use the MOF and QVT formalisms to demonstrate the concepts. We also include a discussion of domain-specific modelling languages such as ladder logic or function block diagrams for PLC programming. The idea is to enable students to extend their understanding also to modelling paradigms which are not mentioned in the course.

8. **Embedded platforms**
   This part gives a short survey of embedded hardware, as well as real-time operating systems. Topics include microprocessor architectures, FPGA and ASIC programming, system-on-chip and embedded CPUs, communication methods, power consumption, etc. On the software side, scheduling, resource allocation and process communication are discussed with example operating systems like RTLinux, FreeRTOS, or others. The choice of topics is mainly determined by the hardware available for the module. Luckily, with platforms like Arduino, Raspberry Pi, or Lego EV3, inexpensive working material for students can be procured. Depending on the available hardware, also current trends like Bluetooth Low Energy or Zigbee can be discussed here.

9. **Functional safety**
   After the very practical implementation work in the previous part, the second part of the module is concerned with quality assurance. First, we introduce the students to the basic concepts of functional safety as defined in IEC 61508. We present the techniques of hazard and risk analysis, as well as FMEA and FTA. Students are challenged to determine the safety integrity level of a certain system, and to conclude which measures are to be taken during the design. The goal is to raise the awareness that functional safety considerations can influence all stages of the system's design.

10. **Fault tolerant design**
    In close connection to the previous part, methods for fault tolerance are discussed. We pose the challenge to design a system of two communicating processors which emit a common synchronized pulse signal, such that each processor can be shut off and rebooted without interrupting the pulse. We discuss dual channeling and

identify possible single points of failure on several examples. Students learn how to analyze system models with respect to safety requirements.

11. **Software and tool qualification**
    This optional part discusses procedures for the assessment and qualification of software and software tools. We discuss the relevant standards, e.g., EN 50128, ISO 26262 and DO-330. Additionally, we discuss some coding standards like MISRA-C. Again, the goal is to raise awareness for safety-oriented design. To do so, we also exhibit some infamous compiler bugs and their potential effects.

12. **Model-based testing**
    In this part we present methods for test derivation from behavioural models. We compare manually designed with automatically generated test suites and explain different model-based coverage criteria. We also show how to re-use test cases from model-in-the-loop via software-in-the-loop to hardware-in-the-loop tests. This brings us to debugging and simulation methods, which are only briefly mentioned.

13. **Static analysis and software verification**
    This part deals with abstract interpretation and model checking. We show how to formulate and formally verify invariants for a behavioural model. We discuss the state-space explosion problem and demonstrate the limitations of the available technologies. Students should get a feeling for the capabilities of modern automatic and semi-automatic verification tools.

14. **Domain-specific methods**
    The last part highlights some methods which are specific to certain application domains. Foremost is the area of automotive software engineering, which has evolved to a curriculum of its own [SZ13]. Current topics are Automotive SPICE, ISO 26262, and the AUTOSAR standardized architecture. Other domains include embedded railway techniques and medical systems, where we discuss implantable devices and body-area networks. Another specialized domain is that of robotics and automation. Here we discuss items like virtual factories, mobile robotics and autonomous systems. We conclude with a summary and discussion of ethical responsibilities for certain industrial applications.

Lab classes

For the above lecture series it is essential that it is accompanied by suitable lab classes. Whereas the lectures present small, "ad-hoc" examples for each topic, the purpose of the lab classes is to deal with larger, coherent examples which cover several topics.

In our previous classes, we used several case studies as exercises.

- **Pedelec**
  This example describes a modern bicycle with electric auxiliary motor. The task is

to come up with requirements for the system and the control unit, to elaborate non-functional and safety requirements, to develop a specification for the pedal power amplification and battery management, build a system model, and to derive code for the display unit. We use IAR visual state and some SI-Labs evaluation boards for the actual implementation.

- **Pace maker**
  This example is a simplified version of a published industrial case study [Bos07]. Modelling of this case study is treated in an accompanying text book [KHCD13]. The students are to design the basic functionality of a fault-tolerant pacemaker in certain operating modes. The example includes timing, fault tolerance and safety considerations, and shows an actual industrial requirements document.

- **Türsteuergerät**
  This example is an automotive door control unit described in the literature [HP02]. The specification describes the operation of power windows, seat adjustment, and interior lighting. Students are to build structural and behavioural models for part of the functionality. Unfortunately, the requirements document is available in German only. Therefore, we plan to replace it by a more up-to-date case study which is currently being developed in the German SPES_XT project [SPE].

Potential continuation topics

The above curriculum is by no means a complete list of material which is relevant for the subject area. There are several additional topics which could be handled in specialized modules.

- **Physical design**
  Currently, this area is not included in our curriculum. With the advent of 3D printers, however, it is becoming more and more interesting to also include the physical design into the model-based design cycle. For physical modelling, tools like Catia and others are being used; a trend in the tools industry is to integrate these with other model-based design tools. However, currently this is still an open issue; it remains to be seen which tools will be available and usable for a university teaching environment.

- **Sensor technology**
  This is an advanced topic dealing with a large range of possible sensing techniques: from specialized flow sensors in pipes via highly accurate laser-scanners to CCD cameras and 3D image processing. In the above curriculum, we use only simple switches and pushbuttons. In particular, we do not concentrate on the design of "smart" sensors with nontrivial signal preprocessing. The area, however, is highly relevant and of growing importance. An interesting future perspective is given by energy harvesting techniques, which allow sensors to compute independently from an external power supply or battery.

- **Communication**
  As embedded systems are advancing to connected, "cyber-physical" systems, also the communication technology is becoming more and more important. Topics like the adaptation of the traditional ISO/OSI protocol stack to the needs of embedded systems could be discussed here, as well as high-speed optical links and wireless sensor networks. Special topics include technologies particular to embedded systems such as NFC and RFID, as well as technologies particular to specific application domain like car-to-car and car-to-roadside communication.

- **Profiles and extensions**
  There are several specialized modelling languages for different purposes. Most notably, UML offers a profiling mechanism for the definition on new languages. SysML as a profile for systems modelling has been treated above. Further profiles to be considered are MARTE for real-time modelling, and UTP, the UML testing profile. In the modelling world outside of UML, other formalisms have been developed which are significant for the topic. In particular, EAST-ADL and AADL are used for architectural modelling in the automotive and avionic domain, and should be included in an advanced course.

- **Multi-core processing and deployment**
  Even though present-day embedded systems are mostly built with traditional, 8-bit CPUs, the trend clearly will be to use up-to-date processors also for embedded tasks. Thus, a specialized topic is how to deal with multi-core (up to 16 CPUs) and many-core (with hundreds of CPUs) processors. Items to be discussed are on-chip synchronization and scheduling, memory access and package routing, and the deployment of models and tasks onto computing units.

- **Softwaretools for systems design**
  Any software development method should be accompanied with relevant tools. Especially in the area we are dealing with there is an abundance of tools, both commercial and experimental. It is interesting to discuss these tools in a systematic way. In particular, tools for hardware/software co-design and design automation were not discussed in the above curriculum. Also, the large area of development management tools has not been treated and could be an advanced topic.

- **Security**
  Of growing importance is the field of systems security, e.g., protection against malevolent attacks. The Stuxnet worm is a prominent example which raised public awareness to this issue. An advanced course could cover technologies for authentification, cryptography, firewalls, etc. In embedded systems, security also means resistance against tampering and counterfeiting, as well as intellectual property protection. Specialized techniques such as design obfuscation have been developed which can be discussed here.

- **Commercial aspects**
  Embedded systems are one of the areas with a huge potential for young, innovative start-up businesses. However, many of these new companies fail since the founders

have good technological, but little commercial knowledge. Here, the university could help by offering classes on market analysis and cost estimation, in particular with respect to the production and marketing of embedded systems. Courses on entrepreneurship and leadership could motivate students to found their own business in this area.

- **Cyber-physical systems**
  A "perspectives" course close to the area of science fiction could be dealing with the elaboration of scenarios for future societies: How will the internet of things evolve? Will there be ambient assisted living aids in every home? Are intelligent humanoid robots an everyday perspective? What other smart environments could be imagined? It would be important to treat these questions from a scientific perspective, e.g., consider not only technological feasibility but also commercial viability of the scenarios.

4 Experiences and conclusion

We have taught courses roughly following the above curriculum twice at the Humboldt Universität zu Berlin, and are preparing a third round. We also gave tutorials for industrial customers based on the material. Moreover, the curriculum has been tried in various Erasmus-lectures held at the University of Swansea, where it supported the preparation of a special embedded systems program. Additionally, we have been teaching at international summer schools in Hanoi and Thessaloniki, where we gave classes covering part of the above curriculum. Here, we report on the experiences with these lectures.

A major issue is the availability and usability of tools. As argued above, the course is almost useless if not accompanied by appropriate exercises and lab classes. Therefore, the selection of tools is a major concern in the preparation. Here, we are facing a dilemma: Commercial tools usually have a complex licensing strategy; often, they are available for students only for a short evaluation period, with limited functionality, or after a complicated registration procedure. In contrast, public-domain tools are readily available; however, they often require nontrivial installation procedures, offer weak documentation and online support, and sometimes even contain bugs. In our lab classes, students complained that the process of "getting started" with a particular tool often was much more time-consuming than the actual work with the tool itself. An ideal workaround for this problem would be to prepare a "live CD", where all necessary tools are pre-installed and there are small hands-on demos with detailed instructions how to use the tool. However, such a preparation is extremely time-consuming for the lecturer; it is not to be expected that this can be done on a wide scale. Moreover, since the subject area is under rapid evolution, such a live-CD would quickly become outdated; the immense effort would have to be done over and over again. Thus, we see a strong need for "lightweight" tools and platforms, which are easy to install and to get acquainted with. The Eclipse platform offers a plugin mechanism which allows to integrate different tools in the same development environment. This could be a perspective to mitigate the problems mentioned above. However,

since also Eclipse itself is evolving, there is the problem how to migrate tools from one version to the next. Currently, we are investigating different strategies for this.

Another issue in our classes was the problem of sticking to the predetermined time schedule. In fact, in none of the mentioned courses we could "cover" the whole plan, since we had to "uncover" items which we thought that they should have been known in advance. A potential reason for this could be that the curriculum attracts students from very different backgrounds. Therefore, things which may be obvious for some of the students may be completely new to others. A way to deal with this situation is to supply appropriate reading material for self-studies. Presently, we are referring the students mostly to original articles. Reading first-hand scientific literature is beyond the effort which average bachelor students are willing to invest for a module. Here, a textbook could be helpful, which describes the base technologies for each of the parts in more detail. Currently, we are discussing the possibility of an appropriate volume with a publisher.

The third issue which needs to be discussed is the rapid evolution of the field. The last decade has shown enormous achievements, and many of the topics of the curriculum are still "under construction". Here, our approach is to split the curriculum into those parts which are more or less settled, and those which are hot and evolving. The above suggestion is a step in this direction: by separating basic and avanced topics, we are able to smoothly incorporate new trends into our curriculum. We hope that this is helpful to others as well.

References

[BJK+05] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Bos07] Boston Medical: PACEMAKER System Specification, 2007. `http://sqrl.mcmaster.ca/_SQRLDocuments/PACEMAKER.pdf`, accessed 2014-01-19.

[GHPS13] Holger Giese, Michaela Huhn, Jan Phillips, and Bernhard Schätz, editors. *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IX, Schloss Dagstuhl, Germany, April 24-26, 2013, Tagungsband Modellbasierte Entwicklung eingebetteter Systeme*. fortiss GmbH, München, 2013.

[HP02] Frank Houdek and Barbara Paech. Das Türsteuergerät – eine Beispielspezifikation. IESE-Report 002.02/D, Fraunhofer IESE, 2002.

[KHCD13] F. Kordon, J. Hugues, A. Canals, and A. Dohet. *Embedded Systems: Analysis and Modeling with SysML, UML and AADL*. ISTE. Wiley, 2013.

[SPE] BMBF: SPES_XT Software Plattform Embedded Systems "XT". `http://spes2020.informatik.tu-muenchen.de/spes_xt-home.html`, accessed 2014-01-19.

[SZ13] J. Schäuffele and T. Zurawka. *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. ATZ-MTZ Fachbuch. Vieweg+Teubner Verlag, 2013.

[TUE] University of Eindhoven: Master's program Embedded Systems. http://www.tue.nl/en/education/tue-graduate-school/masters-programs/embedded-systems/, accessed 2014-01-19.

[UFra] Universität Freiburg: Bachelorstudiengang Embedded Systems Engineering. http://www.ese.uni-freiburg.de/startseite.html, accessed 2014-01-19.

[UFrb] Universität Freiburg: Master of Science Embedded Systems Engineering. https://www.tf.uni-freiburg.de/studies/degree_programmes/master/mscese_en, accessed 2014-01-19.

[UPe] University of Pennsylvania: Master of Science in Engineering in EMBEDDED SYSTEMS Curriculum. http://www.cis.upenn.edu/grad/embedded-curriculum.shtml, accessed 2014-01-19.

[ZSM11] Justyna Zander, Ina Schieferdecker, and Pieter J. Mosterman, editors. *Model-based testing for embedded systems*. Computational analysis, synthesis, and design of dynamic systems. CRC Press, Boca Raton, 2011.