Towards system development methodologies: From software to cyber-physical domain

Maria Spichkova Alarico Campetelli
Institut für Informatik
Technische Universität München
spichkov@in.tum.de campetel@in.tum.de

In many cases, it is more profitable to apply existing methodologies than to develop new ones. This holds, especially, for system development within the cyber-physical domain: up to a certain abstraction level we can (re)use the methodologies for the software system development to benefit from the advantages of these techniques.

1 Introduction

A lot of new ideas are just well-forgotten old ones, and a lot of newly developed methodologies are, in fact, the reinvention of the wheel. In many cases new methods or languages are introduced to deal with cyber-physical systems, where the already existing techniques could be more suitable to represent them, especially after an extension or adaptation to cover some special domain features. Moreover, the existing techniques could provide a connection to the representation of systems from other domains.

Cyber-physical systems are widespread in safety-critical domains, as vehicles, machines, aircrafts or medical instruments. A failure in these systems may lead to considerable loss of money or even endanger human lives. Therefore, it follows the importance of a correct behaviour of the systems, which can be guaranteed through analysis techniques, mostly in form of formal verification (cf. [4]). The support for a formal analysis approach is facilitated by a formal model representation, and a suitable modelling theory for these systems helps in their development, maintenance, simulation, and verification.

The main challenge here is to combine two worlds, the physical and the virtual one: software components operate in discrete program steps, meanwhile the physical components evolves over time intervals following physical constraints. Nevertheless, many physical properties can be represented in a similar manner as the software properties, e.g., in many cases it is possible to switch from the continuous time representation to the digital one without loosing the essential properties of the represented system [12].

From our experience within a number of industrial projects, speaking about the system architecture and properties on a certain abstraction level, ones do not need to distinguish physical signals and component from the software ones. In fact this difference may not have any advantages and, moreover, could make the system description too complicated and hardly readable. Thus, until we are speaking about logical level, we can benefit from using the software system development processes. For this reason, we present a system development methodology, which is a generalization of two methodologies successfully evaluated on three case studies from the automotive domain, with a suggestion to apply its general ideas for the development of cyber-physical systems.
2 System Development Methodology

One of the typical examples of cyber-physical system from the automotive field is a Cruise Control System (CCS). We modeled two different variants of the system using two development methodologies with similar strategies but different focal points. The first case study [7], developing an Adaptive Cruise Control (ACC) system with Pre-Crash Safety functionality, was motivated and supported by DENSO Corporation; while the second case study [19][18], developing a CCS with focus on system architecture and verification, was supported by Robert Bosch GmbH. Another case study [8] motivated and supported by DENSO Corporation was the development of the Keyless Entry-System, which was not only a comfort system with the distributed deployment, but also a system having a huge state space. We did not distinguish between physical and virtual parts of the system focusing on the logical representation here. Thus, a sample-property of a Cruise Control System can be represented as follows: If the driver pushes the ACC-button while the system is On and none of the switch-off constraints occurs, the system must accelerate the vehicle during the next time unit respectively to the predefined acceleration schema. This means that the system must analyze the information from sensors to check whether any switch-off constraint occurs, e.g., if the battery voltage is too low or if the gas pedal sensor fails.

Fig. 1 illustrates the structure of the generalized development methodology in a top-down manner. However, by developing a large system a number of iterations is needed to cope with gain of knowledge about the system, as well as modifications in the requirements. The boxes represent here the artifacts that have been developed and the arrows show from which other artifacts they were derived. The process starts by structuring of initial requirements in the way they follow specific syntactic patterns: this first step raises the level of precision by transforming the free text requirements into a structured form using specific pre-defined syntactic patterns, as presented in [9]. An informal specification consists of a set of words, which can be distinguished into two categories: content words and keywords. Content words are system-specific words or phrases, e.g., “Off-button is pressed”. The set of all content words forms the logical interface of the system, which can be understood as some kind of domain specific glossary that must be defined in addition. Keywords are domain-independent and form relationships between the content words (e.g., “if”, “then”). Thus, a semiformal specification consists of a number of requirements described via textual pattern, which is easily understood also by engineers unfamiliar with the formal methods. Using this description to structure the informal specification, we can find out missing information quite fast. Furthermore, we identify possible synonyms that must be unified before proceeding to a formal specification. This specification can be schematically rewritten to a Message Sequence Charts (MSCs) representation, as an optional step relevant for highly interacting systems.

The methodology proceeds by the translation of semiformal specification to FOCUS [3], a framework for formal specifications and development of distributed interactive systems, preferred here over other specification frameworks. Because it has an integrated notion of time and provides a number of specification techniques for distributed systems and concepts of refinement, as well as graphical notation, which is extremely important when we are dealing with systems of industrial size. We represent in FOCUS two kinds of specifications: system requirements and architecture. This prepares the basis to verify the system architecture against the requirements by translating both to the theorem prover Isabelle/HOL [16] via the framework “FOCUS on Isabelle” [17]. As a next step, we translate the architecture specification to a representation in the tool AutoFOCUS [1] to simulate the system. The requirements specification can be schematically translated to temporal logic or specification patterns, which gives basis to model-check the model (cf. [5]). The integration of model checking in AutoFOCUS approaches usability with

1The AutoFOCUS homepage: http://af3.in.tum.de
the following points: tight coupling of verification properties with model elements, visualization and simulation of counterexamples, and different specification languages for the formulation of properties. Dealing with these issues leads to one of the first model-based development environments incorporating property specification, model checking and debugging.

In the ACC case study \[7\] we apply model checking technique for temporal logic formulas, ranging between SAT solving and interactive verification with regard to its notational power and complexity. The verification was supplied by the SMV tool \[15\] and the properties are specified in LTL (Linear Temporal Logic, e.g. \[14\]), a widespread specification notation suitable for automatic model checking. In this formalism are supported boolean logical operators and temporal operators, further it allows for formalising properties of system states and their changes during system execution. We used SMV to check requirements in form of temporal logic formulas also in the Keyless Entry-System case study \[8\]. When necessary we abstracted variables/data types and/or restrict value ranges, especially for integer variables.

AutoFOCUS models can also be exported to Isabelle/HOL to prove its properties – it is in general a refinement of a FOCUS specification, thus its properties can be slightly different, i.e., more strict, from the ones specified on the FOCUS layer, but the proof schema, which has been developed for the FOCUS specifications, can be (partially) reused. Optionally, we can also represent an environment or a test model if this benefit the analysis of the concrete system. In the Cruise Control System case study \[19, 18\], we applied a semiautomatic verification with a theorem prover and fully automatic by a model checker. Formal requirements can be checked using a theorem prover, through a translator, which generates Isabelle/HOL theories from AutoFOCUS models. To generate the Isabelle/HOL code from the models, at first the user initiates code export for the data dictionary, which generates a theory containing data type declarations and function definitions used in the model. The internal behaviour
of components in the models may be defined by an automaton or a function specification. So, from each component in the models may be generated a theory containing the input and output interface definition and the transition function originating from the automaton or function specification. If the component is composite recursively the theories for all subcomponents will be generated and, ultimately, the theory for the considered component. Generating code for the root component of an AutoFOCUS model produces a set of theories encoding this model in Isabelle/HOL, as well as proof of theorems that support subsequent verification of properties in Isabelle/HOL. This twofold verification guaranteed, firstly that are verified actual properties of the design model and the implementation code; secondly the results of both techniques indicate an implementation fault in one of the generators.

Finally, we can switch from the logical to the real level, where we have to split our model into software and physical components. Transformation to a corresponding C code can be done using the corresponding code generator: we have shown that the C program produced by our code generator is an admissible simulation of the model. Altogether, the methodology guides us from an informal specification via stepwise refinement to a verified formal specification, a corresponding executable verified model, and also a corresponding verified C code implementation.

3 Related Work

There are many approaches on mechatronic/cyber-physical systems, however, most of them do not focus on the logical level of the system representation and loose the advantages of the abstract representation: a better overview, possibility to validate the system in the earlier phases, etc. There is a number of interesting attempts to define a modelling solution for the cyber-physical systems, for instance, the work presented in [20] defines an extensive support to the components communication and time requirements, while the model discussed in [10] proposes a complete model of the processes with communication. Nevertheless, in our opinion one limitation of such approaches is that the system is represented with a flat view, that is, there is only a single abstraction level to represent it. That could be a disadvantage in the project of a cyber-physical system, where experts of different domains should be able to cooperate and work in different views and abstraction levels of the system. Modeling theories for distributed hybrid systems such as SHIFT [6] and R-Charon [13], guarantee a complete simulation and compilation of the models, but they have no comprehensive verification support. The same limitation is for UPPAAL [2] and PHAVer [1], which provide the simulation, but a limited verification with restricted dynamics and only for small fragments.

4 Conclusion

In this paper we have suggested to reuse the generalization of two existing methodologies for the development of software systems to apply them within the cyber-physical domain, according to the results of three case studies motivated and supported by DENSO Corporation and Robert Bosch GmbH. Up to a certain abstraction level we can use the existing methodologies for the development of software systems also within the cyber-physical domain to benefit from the advantages these techniques have shown, as well as extend the modelling artefacts for the domain features. As for instance, introduce at some lower abstraction level continuous dynamic modelling capabilities and so its verification, in our existing tool and theory. The simulation of a full continuous dynamic has still a not optimal performance, so we are oriented for a discretization of the simulation, through the implementation of dynamical adapting sampling.
References


