

Requirements Management for Embedded Software: An Interdisciplinary Challenge

Matthias Jarke, RWTH Aachen University & Fraunhofer FIT
Ahornstr. 55, 52074 Aachen, Germany

Many engineering disciplines have long followed the quest for a “science of design”. The goal is to reach certain functional properties together with a specific desired behavior, while at the same time avoiding undesirable side effect behaviors and pursuing a broad spectrum of possibly conflicting “non-functional” goals. Sometimes, the models describing these aspects are quite refined and often reflect the basic problem understanding of a given discipline.

When the developers of such models need to communicate, e.g. in the design of embedded software or information systems solutions, the fundamental challenge of mutual understanding arises. Based on such an understanding, the next step is equally difficult: the detailed understanding of the relationships between the different modeling paradigms and models. This model mapping must be detailed enough to ensure adequate inconsistency management but often, it must also be left separate enough to preserve disciplinary company secrets. And last not least, cross-disciplinary traceability mechanisms are needed to document these relationships and their evolution in a manner that third persons who need to evolve the resulting system, can reuse the shared understanding and modeling that has been reached.

While we believe that this interdisciplinary challenge appears in basically any information systems or software engineering domain (including business IS), the automotive industry is an area where this interdisciplinary understanding has been facing a particularly strong challenge in the past few years. First, there was the step from mechanical engineering to the integration of electronics design, now there are the added chances and challenges given by the networking of component designs and the resulting software integration needs. And soon, these mobile car-level systems will be linked with the ERP systems of car vendors and service organizations to optimize planning and design feedback loops from cars on the road.

We are currently involved in what we believe is a rather prototypical example of such theory integration, namely the integration of model-driven software development and model-based control in the design of car engine controllers, one of the highly relevant issues for gas saving, climate change reduction, etc.

The two concepts sound very similar, but in fact stem from very different traditions in electrical engineering and software engineering. Model-based control refers to mathematical models of cybernetic feedback cycles which are often based on differential equations; from these models, the control software is then typically generated in an almost trivial manner after long experimentation and simulation at the mathematical level. Model-driven software development refers to typically discrete formal specifications of complex software systems (often component-based ones) from which code, or at least code frames and test cases, can be generated with formally proven properties.

In our experience, the many homonyms and the very different mathematics involved make interdisciplinary discussions extremely complicated. We therefore propose to abstract from the details of the formalisms to a generic framework of goals, dependencies, and information flows in order to facilitate interdisciplinary understanding and to assist in the documentation of cross-disciplinary design decisions, e.g. in cross-disciplinary version and configuration management. An extension of Eric Yu’s *i** framework shows some promise as an approach.