Introduction

Frits Vaandrager

Computing Science Institute, University of Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
fvaan@cs.kun.nl

With the decrease in the size and price of computing elements, more and more computers are used within real-world technical applications such as in avionics, process control, robotics, telecommunications and consumer products. The software within such applications is called "embedded" since it determines to a large extent the functionality of the product; the software and the rest of the product have no meaning without each other. Characteristics of these applications are:

- Complex interaction with environment. One can only design and reason about embedded systems if one takes the behavior of their environment into account. Frequently this environment is highly nondeterministic and intrinsically continuous. Moreover the rate at which input signals arrive is often enormous and signal processing and information filtering become key issues.
- High dependability requirements. Besides functional constraints many other aspects play a role in the design of embedded systems: timeliness, fault tolerance, availability, security, safety, etc.

In recent years there has been a dramatic growth of the number of embedded applications and of the size and complexity of the software used in these applications. For many products in the area of consumer electronics the amount of code is doubling every two years. As a result, for instance, a high-end television today contains about 1.4M of software and a GSM phone 2M of software. The number of microprocessors that we encounter daily in our lives has increased from 2 in 1985 to 50 in 1995. In 1994, a group of experts estimated that the number of software engineers who design and write embedded software will increase from 2 million in 1994 to 10 million in 2004. Indeed, embedded software is the fastest increasing area of information technology, with the exception of the "intermittent programming" by non-software professionals.

The construction of large real-time embedded computer systems is one of the most challenging tasks facing the computer science community, if not the engineering community as a whole, and raises important questions to both science and technology in areas such as extended product lifecycle, product families, interaction with the environment, productivity, and dependability. The objective of the School on Embedded Systems was to give young researchers a good feeling for the practical problems that arise during the construction of embedded systems, and an overview of some important methods and techniques that may help to address these problems. Amongst the lecturers at the School there was agreement that there are many architectural and algorithmic problems in the area of embedded systems that have well documented optimal or correct solutions, notably in the areas of real-time computing, distributed computing, and fault-tolerant computing. However, only a small subset of these solutions have so far translated into commercial, off-the-shelf products or operational computer-base systems. We already have plenty of theoretical results, what we need now is making these work in large-scale applications.

Formal Models and Analysis Within Computer Science enormous progress has been made during the last decade in the area of formal models that support analysis of embedded systems. A hierarchy of three (types of) models, each extending its predecessor, has been developed:

- *Reactive system* models that capture the *qualitative* temporal precedence aspect of time. These models can only identify that one event precedes another but not by how much.
- *Real-time system* models that capture the *metric* aspect of time in a reactive system. These models can measure the time elapsing between two events.
- *Hybrid system* models that allow the inclusion of *continuous* components in a reactive real-time system. Such continuous components may cause continuous change in the values of some state variables according to some physical or control law.

Building on their earlier work on reactive system models, *Kesten, Manna and Pnueli* give a detailed exposition of a state-of-the-art computational model for real-time and hybrid systems (the "clocked transition system (CTS) model"). They discuss requirement specification and system description languages based on this model, and proof rules by which valid properties of systems can be verified, showing that systems satisfy their specifications. *Branicky* studies real-world hybrid control systems from the perspective of Control Theory and introduces a mathematical model of them. Based on this model stability properties of hybrid systems can be analyzed, and optimal hybrid control systems can be synthesized. *Yovine* presents a survey of the different algorithms, data-structures and tools that have been proposed in the literature to solve reachability questions for real-time systems.

Real-time Systems One of the fundamental problems in real-time research is to devise accurate yet tractable models for real-time tasks. *Chen, Mok and Baruah* try to organize different models and their feasibility difficulties into a uniformed structure. Their paper covers probably the most general Liu&Layland-type real-time scheduling model to-date.

Traditional approaches to the design and implementation of real-time embedded applications transfer timing requirements into machine dependent operations using low level programming languages, such as Assembly or C. This approach reduces the modularity and reusability of software components. In order to address these difficulties, *Ren and Agha* develop a high-level language approach (the "RTsynchronizer") to support a stronger separation of design concerns.

Development and Engineering In his important contribution, *Le Lann* argues that system engineering rather than software engineering currently is the weakest (i.e., least rigorous) of all those engineering disciplines covered by what the IEEE Computer Society refers to as the Engineering of Computer-Based Systems. He introduces some basic principles that underlie proof-based system engineering, a discipline which serves to avoid system engineering faults. An analysis of the Ariane 5 Flight 501 failure serves to illustrate how proof-based system engineering also helps in diagnosing causes of failures.

In traditional design methodologies, the system designer typically develops the application in a sequential paradigm almost to completion before addressing issues of parallelism and mapping to a heterogeneous architecture. As the architectural complexity of these applications increase (which is often the case for embedded applications) this process becomes too costly since implementation must be started anew after the design. *Bhatt and Shackleton* present a new methodology and toolset aimed at addressing this problem and at improving the system development process for high-performance embedded applications.

Van den Hoogenhof examines some important techniques and constructions used in the mainstream object-oriented software engineering methodologies, and elaborates on their common properties and weaknesses in practical use. He concludes that the maturity of methodologies is only partially reflected by their properties. Perhaps the main problem is the attitude of software engineers who reflect their failures on the methodologies rather than taking the full responsibility themselves. *Kuusela* presents an overview of OCTOPUS, a software development method for embedded real-time systems that has been developed and is used by Nokia.

Case Studies This volume includes three case studies in which formal methods are applied in the development of realistic embedded applications. *Ravn, Eriksen, Holdgaard and Rischel* use Duration Calculus to develop a multi-threaded, multiprocessor control program for a robot. *Kars* uses the SPIN tool set and Z to formally specify and analyze a storm barrier control system. Finally, *Douglas, Harel and Trakhtenbrot* describe the use of statecharts for the specification of a cardiac pacemaker.

Industrial Perspective In his contribution, *Bourgonjon* discusses some impressive statistics that illustrate the increasing use of embedded software in electronic consumer products. *Van Rietschote* presents Telecommunication Applications as the largest, most complex embedded software applications in the world today, and discusses what Computer Science research could do for enabling programming large and complex embedded systems. *Veltmaat* describes the realistic side of embedded software development at ICT Automatisering, one of the largest embedded software development companies in the Netherlands, and discusses rules of conduct and possible ways to improve the cooperation between the software industry and the scientific community.