



Educational Objectives for Embedded Systems

Wolfgang A. Halang

Fernuniversität in Hagen
Chair of Computer Engineering
58084 Hagen, Germany
Wolfgang.Halang@FernUni-Hagen.de

Abstract. Adequate concepts, professionalism and ethics are identified as the goals in properly educating systems engineers for professional life. Criteria and concepts appropriate to design feasible solutions must be based on fully understanding the peculiarities of the environments where embedded systems are employed. Bad practices of the computing profession need to be overcome to foster professionalism, and the consequences of the shift from hardware to software must be realised. This gives rise to some rules of ethics and proper conduct.

Which issues should be addressed by curricula on real-time and embedded systems that are longer lived than technical artefacts, and that constitute elements of proper university education? My answer is that engineering professionals should be educated on the following fundamental essentials:

adequate concepts, professionalism, ethics.

Computer control totally differs from other areas of computing as processing takes place not only in real time, but also in real environments into which real-time systems are embedded and to whose requests they react. Essential is that they always deal with reality, and that there are always physical environments with their own laws, which cannot be forced under the rules of computers. Although performing control functions, embedded systems thus always take the position of slaves, whereas the environments constitute the masters. This very nature of real-time computing must particularly be taken into account when developing abstractions in order not to lose coherence with reality.

Owing to the embedding into physical environments, systems engineers must have a deep understanding of computing, of physics, and of the application-specific requirements. The methods to be employed are not general-purpose ones, but must always be selected in environment- and process-specific ways. Dealing with the real world also implies that physical constraints must be observed, holding first and foremost for the time dimension and time-related requirements. When designing systems, this calls for applying static design and analysis methods as well as the concept of resource adequacy, i.e., computers must physically provide all resources required in the worst cases. In other words, one may not resort to the popular method of dynamically creating “virtual resources”.

By their very nature, embedded real-time systems are always complex. To cope with them, engineers must employ a holistic view as design baseline, and methods apt to master complexity. Moreover, real-time systems are almost always safety-related, as they are embedded in environments which they control. For this safety issue the requirement of dependability results to be fundamental for real-time systems. It can only be met by a priori safety-licensing.

The performance criteria applicable to embedded systems are only very seldom quantitative, but mostly of a qualitative (binary, i.e., requirement fulfilled or not) nature. This is best exemplified by the most important criterion, viz., whether (hard) deadlines are met or not. Corresponding to the holistic view to be taken at design time, cost evaluations must be oriented at overall figures made up of costs for hardware, software, the embedding environment or processes, safety-licensing, potential damage in case of malfunction etc.

In contrast, the common focus on meaningless processor costs is typical for a number of mainstream misconceptions, which are rather unnatural from the common sense point of view. A further misconception is to conclude from randomly distributed data arrivals that real-time systems should not behave deterministically. On the contrary, as only deterministically behaving systems are safety-licensable, all computer reactions must be precisely planned and fully predictable, particularly for all cases of conflict and error. The need to observe worst-case requirements and physical constraints renders harmful all so-called dynamic and virtual features, which are so popular in mainstream computing.

Scheduling is still the topic of most active academic research in real-time systems. Its rationale is to optimise the utilisation of resources, which makes sense as long as resources are scarce and expensive. It turns into nonsense, however, when just the utilisation of processors is considered, which are now very cheap and available in abundance. For embedded and safety-critical real-time systems, maximum processor utilisation is irrelevant and, thus, cannot be a credible research topic anymore. On the system level, the costs of overdimensioned and, hence, underutilised processors are negligible, but may, on the other hand, be very well invested if system dependability can be enhanced by simplifying software. Since less or less complex software is also cheaper, higher spending in processors may even result in overall cost savings. In other words, the only meaningful optimisation criterion is to minimise the bottom line.

When it comes to the notion of time, the influence of mainstream computer science is most detrimental. For real-time control systems, however, time is the central concept, because processes proceed in time, and there is no functional correctness without correct timing. Time is an absolute measure and a practical tool allowing to plan (technical) processes and future events with all their mutual interactions requiring no further synchronisation. That is why the concept of time, which is an abstraction from natural phenomena and a model of thought, is so useful. The model employed in everyday life, viz., as the fourth dimension of our Euclidian space of experience, is also appropriate to design real-time systems. It is important to note that it is nonsense to model time, because time is already a model, that the time relevant to technical systems is defined by law,

that incorrect timing may have legal consequences such as liability suits, and that the model is closely approximated by technical means in form of atomic clocks, which provide the legal Universal Time Co-ordinated. All these notions are strange for computer scientists, but essential to real-time systems engineers.

As a scientific and technical discipline, computing is still highly immature, which is manifested by fashions, buzzwords, and hypes for supposedly new developments and artefacts. As a consequence, in the embedded systems field often adequate and proven tools, such as proper real-time programming languages, are abandoned in favour of worse tools, such as the non-real-time language C, for irrational reasons like gossip among managers, but not on the basis of sound evaluations of technical merits. In such an environment it comes as no surprise that good designs and engineering solutions are not recognised as such, are not well documented, and are not disseminated as design patterns. Instead, one resists to learn, and re-invents the wheel about every five years. Observing the field for over 30 years now, I have already seen many of these re-inventions, which very often are clearly inferior to the original solutions. This is highly unprofessional. A scholarly discipline ought to place one layer of knowledge on the other, and ought to prefer good solutions over just new ones.

Hence, there was negative progress in the field, which was more advanced around 1980 than it is now. This is not only evidenced by the situation in languages, but also, for instance, by the disappearance of good real-time computers or the fact that no contemporary real-time operating system can match the desired functionality as described and implemented in 1975. As a result, we witness a loss of quality, reliability, and safety. The reason for this decay is mainly the adoption of general-purpose hardware, software, methods and tools. Their adoption was not only caused by wrongly understood cost considerations, but to a large extent by the already mentioned fashions and other marketing gimmicks.

Systems engineers need to develop a professional “sense of proportion”, attributing correct weights to design requirements and issues. An outflow of this principle into the direction of real-time systems research is to address only real problems, but not “toy problems”. For designers it is very important to exercise self-discipline. There is a common habit among software developers to program any function anew, because they cherish their own programming skills. Professionals need to refrain from this. Although they may emotionally favour more elegant solutions, they observe design guidelines and programming rules in the interest of dependability. The main objective of self-discipline is to prevent the creation of artificial and unnecessary complexity, and to ease verification.

As result of a remarkable paradigm shift now over 60% of the revenue generated by the electrical and electronic industries can be attributed to software, whereas their tangible products and devices moved to a secondary rôle. Real-time systems engineers must be fully aware of this fundamentally changed situation. The sense of proportion demanded above will lead them to address in their designs the most pressing problems, i.e., the ones related to software and its intrinsic dependability deficit.

In contrast to instruction of knowledge and training of capabilities, the objective of education is to form personalities. From mature and valuable persons we expect a behaviour guided by ethic principles. Both the design of systems and the design of tools ought to be oriented at humans and their thinking. Systems are to serve people. Thus, system design should not be driven by interesting new technology or market forces, but by the real needs of people. There is a plentitude of so-called methodologies and formal methods around, which claim to support design processes and to allow for rigorous system verification. None of them found widespread acceptance. One of the reasons for this failure is that many tools do not appeal to the way of thinking of their envisaged users, in this case engineers.

Among the real needs of people is that technical systems should be dependable. This is of increasing importance as daily life in all its aspects more and more depends on technical systems of any kind. Embedded control systems are among the most complex and challenging ones as they are composed of elements from various engineering disciplines, and of hardware and software with their totally different natures. Actually as system integrators, on control engineers rests most of the responsibility for the proper function of technical systems. As many of them are safety-critical, real-time systems engineers must always be aware that life and health of persons depend on them. Therefore, their work must be guided by the fundamental principle to fight (artificial) complication, and to strive for the most simple solutions, because the latter allow for most easy verification.

The bad influence of the computing community can be overcome if systems engineers observe the following rules of conduct already well established in other scientific and engineering disciplines:

- take care that programs and systems remain intellectually tractable,
- prefer simple structures and solutions for their clarity,
- fight complexity and strive for simplicity,
- base your work on the achievements of earlier generations,
- do not abandon the good for the worse,
- resist market forces and fashions.

To summarise, the fundamental concepts of real-time computing and embedded systems differ considerably from mainstream thinking in computer science, causing the field to be much more closely related to electrical and control engineering. The mainstream computer science thinking needs to be overcome in order to meet the demands of real-time and real-world applications, and not to fall behind real-life practices. The pressing problems of embedded systems are their integrity and safety imposing responsibility for human life and well-being on real-time systems engineers. Given the present state-of-the-art, to cope with these problems first and foremost the dependability of software must be improved. To this end, complexity has to be fought in all aspects. In order to foster this, their education needs to convince future real-time systems engineers that simple solutions are the most difficult and challenging ones, because they demand high innovation and full intellectual penetration of subject matters.