Real-Time Systems Education: What is Really Essential?

Wolfgang A. Halang
Faculty of Electrical Engineering
FernUniversität
58084 Hagen, Germany
wolfgang.halang@fernuni-hagen.de

Abstract

Adequate concepts, professionalism and ethics are identified as the goals for the proper education of real-time systems engineers for a professional life. On a full understanding of the peculiarities of the area of real-time computing and control appropriate criteria and adequate concepts must be based to be employed in designing solutions to real-time problems. To foster professionalism, some bad practices associated with the computing profession need to be overcome, and the consequences brought about by the shift from hardware to software must be realised. This gives rise to some rules of ethics and proper conduct, whose observation can be expected from scientists and engineers.

1. Introduction

The preceding contributions to this volume as well as [4] highlighted a multitude of aspects both classical to or newly introduced into real-time systems curricula. These papers provide many new concepts for the contents and the structure of lectures, as well as for case studies dealt with in practical exercises. In other words, we learn from these articles which knowledge is to be taught to students of real-time computing and control, and which capabilities are to be acquired.

Owing to the rapid progress of science and technology, the knowledge learnt and the capabilities trained turn obsolete relatively quickly. The purpose of this paper is to answer the question, which issues should be addressed by real-time systems curricula that are longer lived than technical artefacts and that constitute elements of a proper education — education in contrast to instruction and training by which knowledge and capabilities are conveyed. My answer to this question is that future engineering professionals should be educated on the following essentials, which are of fundamental importance without being in danger of becoming out-dated:

- **Adequate Concepts,**
- **Professionalism,**
- **Ethics.**

In the sequel, I shall elaborate on these three issues. The first one is relatively technical. It can be summarised as the need to fully understand the peculiarities of the area of real-time computing and control, to base appropriate criteria and adequate concepts on this understanding and, finally, employing them in designing solutions to real-time problems. The coverage of professionalism will not address engineering professionalism in full, but will concentrate on some bad practices associated with the computing profession, and will point out the consequences as brought about by the shift from hardware to software. This will finally give rise to some rules of ethics and proper conduct, whose observation can be expected from scientists and engineers.

2. Adequate Concepts

Real-time computing totally differs from other areas of computing as it 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. This is why real-time systems are also called embedded systems, reactive systems, or responsive systems in dependence of different viewpoints taken. 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 (computer) systems thus always take the positions of slaves, whereas the environments constitute the masters. This very nature of real-time computing must particularly be taken into account when developing abstractions. Here utmost care has to be exercised in order not to lose coherence to reality.

The consequences of the embedding into physical environments are that real-time systems engineers must have a deep understanding of both computing, and of the physics and the requirements of each application area they are dealing with. There are no general-purpose methods. On the contrary, the methods to be employed must always be selected in an environment specific and process specific way. Dealing with the real world also implies that physical constraints must be observed, which holds first and foremost for the time dimension and time-related requirements. In the course of designing systems, this calls for the application of static design and analysis methods as well as for the concept of resource adequacy, i.e., computers must physically provide all the resources that are required in the worst case. In other words, one may not resort to the popular method of dynamically creating "virtual resources".

By their very nature outlined above, embedded real-time systems are always complex systems. To cope with them, engineers must employ a holistic view as design baseline, and methods apt to master complexity. System complexity has to be seen in the light of the fact that real-time systems are almost always more or less safety-related, as they are embedded in environments or processes which they control. This safety issue results in the requirement of dependability being fundamental for real-time systems. It can only be met by safety-licensing systems before they are put to use. Therefore, system builders are confronted with the difficult task to structure and design complex systems in such a way that their dependability can be a priori proven.

Real-time computing also differs from general computing with respect to assessing performance and costs. As shown in [3], the performance criteria applicable to this area are only very seldom quantitative, but most of the time of a qualitative (binary, i.e., requirement fulfilled or not) nature. This is best exemplified by recalling the most important performance criterion, viz., whether (hard) deadlines are met or not. When it comes to costs, the overall picture made up of costs for hardware, software, the embedding environment or processes, safety licensing, potential damage in case of malfunction etc. must be considered. Corresponding to the holistic view to be taken at design time, cost evaluations and comparisons must be oriented at overall costs ("bottom-line approach"), instead of concentrating on meaningless partial costs of, say, processors.

This rather common focusing on processor costs is typical for a number of mainstream misconceptions, which are quite unnatural, from the common sense point of view, and which obviously originate in computer science circles. Another one of these misconceptions is to conclude from randomly distributed data arrivals that real-time systems should not behave deterministically. On the contrary, all computer reactions must be precisely planned and fully predictable, particularly for all cases of conflict or error. The reason for this is that only deterministically behaving systems are safety licensable. We have already pointed out, that 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 most active academic research area in real-time systems. The original rationale behind scheduling is an economic one, viz., to optimise the utilisation of resources. This makes sense as long as resources are scarce and expensive. But it turns into nonsense when, as customary in scheduling research, the utilisation of just processors is considered, which are now very cheap and available in abundance. For embedded real-time systems, on which people may depend with their lifes, maximum processor utilisation is totally irrelevant and, thus, cannot be a credible research topic anymore. As exemplified by the cost relations of a typical machine tool, namely, 40% for mechanical parts, 20% for electrical parts including control computers, and 40% for software, the costs of overdimensioned and, hence, underutilised processors are negligible on the system level, 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 processor spendings may even result in overall cost savings. In other words, the only meaningful optimisation criterion is to minimise the bottom line.

People appear to be most brain-washed by mainstream computer science, when it comes to the notion of time. In contrast to our everyday experience, here time is abstracted away. 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 which allows to plan (technical) processes and future events with all their mutual interactions requiring no further synchronisation. That is why the concept of time, which intrinsically is an abstraction from natural phenomena and a model of thought, is so useful. The model employed in everyday life, viz., as the 4th dimension of our (Euclidian) space of experience, is also the one appropriate for the design of real-time computing systems. Finally, it needs to be pointed out that time should not be modeled, as it is popular in computer science circles, because it 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 time Universal Time Co-ordinated (UTC). All these notions are strange for computer scientists, but essential to real-time systems engineers.

3. Professionalism

In the preceding section we have discussed a number of misconceptions, which need to be overcome by real-time systems education. They have their origin in computer science or the computing profession, respectively, which also brought about the bad practices to be addressed in the sequel.

As a scientific and technical discipline, computing is still immature. Among others, this is manifestated by fashions, buzzwords, and a hype for supposedly new developments and artefacts. As a consequence, in the real-time systems field it can be observed that 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 and its derivatives, for irrational reasons like fashion and 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, the profession resists to learn, and re-invents the wheel about every five years. Observing the field for some 25 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, we can state that there was negative progress in the field of real-time systems, 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. Elzer analysed in [2], that the reason for this decay is mainly the adoption of general-purpose hardware, software, methods and tools. The adoption was not only caused by cost considerations, but to a large extent by the already mentioned fashions and other marketing gimmicks.

In the preceding section we pointed out that an holistic view and the "bottom-line approach" are among the concepts adequate for real-time systems design. This can be generalised into the professional principle that a "sense of proportion" is to be exercised, 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 or just academic questions of little or no value for the practising engineer.

For real-time systems designers it is very important to exercise self-discipline. There is a common habit among software developers to program any function anew, because they estimate their programming capabilities higher than those of others. Professionals need to refrain from this. They also need to observe design guidelines or programming rules set in the interest of enhancing dependability, although emotionally they may favour possible other solutions which may be more elegant. The main objective of self-discipline should be to avoid the creation of artificial and really unnecessary complexity, and to ease verification.

Very recently, a remarkable paradigm shift has taken place in engineering. 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. The importance of software on one hand, and its intrinsic problems on the other, are nicely summarised in the following quotations [5]:

> "We are now faced with a society in which the amount of software is doubling about every 18 months in consumer electronic devices, and in which software defect density is more or less unchanged in the last 20 years.
>
> In spite of this, we persist in the delusion that we can write software sufficiently well to justify its inclusion at the highest levels of safety-criticality."

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 dependability.

4. Ethics

In the introduction to this paper, education was placed in contrast to instruction of knowledge and training of capabilities, since its objective is to form personalities. From mature and valuable persons we expect that their behaviour is guided by ethic principles. Based on the considerations outlined above, in this section a number of ethic principles will be compiled which should be observed by engineers designing and implementing real-time computing and control systems.

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, the design of systems 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 the design process 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 depends more and more on technical systems of all kinds. The embedded control systems, which real-time systems engineers design, 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.

Finally, the bad influence of the computing community on the real-time systems field as outlined above can be overcome if real-time engineers observe the following rules of conduct, which are already well established in other scientific and engineering disciplines:

- base your work on the achievements of earlier generations,
- do not abandon the good for the worse,
- resist market forces, and
- resist fashions.

5. Conclusion

There is a number of fundamental concepts which constitute the distinguishing features of real-time computing. These concepts 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 real-time systems are their integrity and safety imposing responsibility for human lifes 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 of real-time systems. 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 [1].

References

[1] K. Biedenkopf: Komplexität und Kompliziertheit. *Informatik-Spektrum* 17, 82 – 86, 1994.

[2] P. Elzer: Missed Opportunities in Real Time Programming? in *Innovationen bei Rechen- und Kommunikationssystemen*, B. Wolfinger (Ed.), pp. 328 – 339, Heidelberg: Springer-Verlag 1994.

[3] W.A. Halang, R. Gumzej and M. Colnarič: Measuring the Performance of Real Time Systems, in *Real Time Programming 1997*, M. Maranzana (Ed.), Oxford: Pergamon 1998.

[4] W.A. Halang, C.E. Pereira, J. de la Puente, A. Shaw, J.J. Skubich, Th. Tempelmeier, J. van Katwik, H. Wedde and J. Zalewski: Real-Time Computing Education: Responding to a Challenge of the Next Century, in *Real Time Programming 1997*, M. Maranzana (Ed.), Oxford: Pergamon 1998

[5] L. Hatton: *Safer C: Developing for High-Integrity and Safety-Critical Systems*. New York: McGraw-Hill 1995.