Embedding Practical Real-Time Education in a Computer Science Curriculum

Theodor Tempelmeier
Fachhochschule Rosenheim, Germany

Abstract

This contribution reports on the approach to real-time education at the Fachhochschule Rosenheim. The importance of real-time education is acknowledged, but, instead of having a separate curriculum, the author favours an "embedded" solution. Real-time education is integrated into the normal curriculum of technical computer science by carefully adjusting the contents of all courses to each other.

1. Introduction

The importance and the market volume of real-time and embedded systems are well-known and need not be discussed here. Similarly, the need for a specific real-time education is beyond any doubt in the author's view, because of the well-known differences between desktop computing and real-time/embedded computing (timeliness, predictability, limited resources, etc. – see [1] for a general discussion). However, the author would dislike a separate curriculum and an extra degree for realising real-time education (and for many more specialised disciplines, too). Instead, real-time education can be embedded into the normal curriculum of technical computer science by carefully adjusting the contents of all courses to each other.

2. Computer science education at a German "Fachhochschule"

German "Fachhochschulen" are also called universities of applied sciences. They offer a variety of courses, often in the technical area. Studies end after 8 semesters (four years) with a degree in between a Bachelor's or Master's degree. A transition to these internationally understandable terms is under way. Education at German Fachhochschulen has a strong emphasis on practical aspects. However, this does not mean that no theory is taught, but it means that (useful) theories are always shown in the context of their practical applicability.

The Fachhochschule Rosenheim offers degree courses in computer science, with a specialisation towards economics or technical applications during the final year of studies. In the following, only the latter, i.e. the "technical computer science" branch, will be considered.

The curriculum of technical computer science at a Fachhochschule in Germany is fixed with respect to the time frames and the titles of the courses, but gives much freedom to the individual lecturer to fill in the contents of the courses ("freedom of teaching" as guaranteed by the German constitution). Usually, of course, the rough contents of the courses are determined by complete consent of all lecturers of a particular course of studies.

The author has been heading the "technical computer science" branch at the Fachhochschule Rosenheim for some years (comparable to a dean of studies), and has also lectured all key courses at least once. This has caused the lucky situation that the author could integrate his ideas about real-time education into the course of studies.

The first six semesters of studies of computer science contain standard courses, such as programming, software engineering, algorithms and data structures, compilers, data base systems, networking, operating systems, etc. Building on this basis, the curriculum of technical computer science at a German Fachhochschule could typically look like the following (the curriculum as in the state of Bavaria):

Semester	Course	Credit Points
...		
5	Procedural Programming Languages	4
...		
7	Computer Architecture and Organisation	6
7	Machine Level Programming	4
7	Technical Physics	4
7	Numerical Mathematics	4

0-7695-0134-6/99 $10.00 © 1999 IEEE

8	Real-time Systems	4
8	Technical Applications of Data Processing	6
8	Final year project	
	Optional courses on CAD, electronics, computer hardware (VHDL), busses and interfaces, image processing, and others	

Table 1: Excerpt from the curriculum of technical computer science at the Fachhochschule Rosenheim.

Credit points are roughly equivalent to teaching hours per week. Almost all courses use half of the time for practical exercises.

3. A real-time "curriculum"

3.1. General approach

From the above curriculum it is seen that only four hours per week during one semester are specifically devoted to real-time systems. This is considered to be insufficient by far. So some of the other courses have to serve a dual purpose: besides teaching the basics of each course, the interdependencies with real-time and embedded systems are also taught, mostly by examples, exercises, discussions, and remarks on the relation to real-time systems.

The author thinks, it is essential to know precisely what happens inside a computer when dealing with real-time systems. This refers to all levels of abstraction, from high-level language constructs down to interrupts, direct memory access and the like. The next three paragraphs will show, how this kind of knowledge is taught in sufficient detail as a preparation for the real-time course.

The author also thinks, it is essential to actually design and program a sufficiently large real-time system, in order to get a full understanding of the subject. Chapters 3.5 and 3.6 will show how this is achieved.

3.2. Procedural programming languages

Students learn C/C++ as their first programming language. In the course *Procedural Programming Languages* some basic principles of programming languages in general (and the shortcomings of C/C++) are shown. Ada is used as a vehicle for teaching. This language provides sound concepts, covers virtually all aspects of procedural and object-oriented languages, sharpens the mind for aspects of safety-criticality, and it is also a real-time language. Apart from Ada's real-time and safety features, the subtleties and the seemingly "trivial" aspects of programming languages are also imparted. For instance, the mega-minded students usually do not care to analyse the difference[1] in

```
c:     float x;    ...    x = x + 1.23;
Ada:   x : float;  ...    x := x + 1.23;
```

The students have to learn that even an understanding of such details is necessary for a real-time engineer. As a second example, treatment of fixed point numbers, which are very important in embedded systems, easily shows to the students that this issue is not so trivial as thought at first.

This course is held separately for the students of computer science/economics, with a different content, stressing other aspects than real-time.

3.3. Computer architecture and organisation

The course *Computer Architecture and Organisation* has the standard contents, as it is found in textbooks on this subject. It is interesting to see that textbooks on real-time systems also very often contain chapters about processor organisation, busses, interrupt systems, direct memory access, memory management, etc. It is agreed that this kind of knowledge is indispensable for real-time engineers (to know precisely what happens at the hardware level). However, this is not real-time knowledge per se, but a basis for real-time. So, the hardware course is taught with real-time in mind and with many hints to the forthcoming real-time course.

Again, a different course is held for the computer science/economics branch.

3.4. Machine level programming

Machine Level Programming was called *Assembler* formerly, and this characterises parts of this course very well. In the opinion of the author it is still reasonable to speak of assembler, though one should avoid to use it in projects. Assembler is not taught as just another (dull) programming language. Instead, it serves as an object of study of processor architectures and of compiler code generation. Thus, there are tight relations to the two courses *Computer Architecture* and *Procedural Programming Languages.*

[1] The constant 1.23 is missing an f to denote a float constant: 1.23f. By default, 1.23 is of type double, and this may cause unnecessary conversions between float and double, depending on the arithmetic expression [2]. This would be in particular undesirable, if an embedded system without floating point hardware were to be used. (However, in that case one should perhaps resort to fixed point types, anyway). In Ada, on the other hand, the constant 1.23 is of type universal-real, causing the "right" (single precision) type to be selected "automatically" [3].

A C compiler with an inline assembler is used (not a stand-alone assembler) for the following reasons. Firstly, this is an approach like in practice ("Do it in C, use (inline) assembler only if absolutely necessary"). Secondly, this makes it very easy to study the generated code. And again, a thorough understanding of the generated code, i.e. a detailed understanding of what goes on in the computer or what the compiler has done, is considered very important for real-time engineers.

Assembler programming may seem boring, questionable, or out-of-date to the students. So the subject *micro-controllers* has been integrated into this course, partly to make it more attractive and interesting, partly because it serves the purpose of real-time education. Small microcontroller boards with limited resources and without operating system are used as targets. This is a very common situation in many embedded systems. The course currently uses Motorola MC68332 micro-controllers [4]. The further development of architectures is then discussed along the Coldfire® RISC architecture [5].

Currently, it is planned to install equipment for a final top-level exercise. An actual brake system of a car would have to be controlled in a way to avoid skidding, i.e. a simple form of an A.B.S. (anti-blocking system) would have to be implemented. Students may probably compare their solution to an industrial ABS computer, because the Fachhochschule Rosenheim has good relations to a leading manufacturer of automotive electronics and plans further co-operation.

3.5. Real-time systems

The course on *Real-Time Systems* starts with a synopsis of the field. Next, the subjects

- understanding real-time implications of computer software
- understanding real-time implications of computer hardware
- simple solutions with small micro-controllers

need only briefly be repeated, because they have already been dealt with in other courses (see chapters 3.2 – 3.4).

The course on real-time systems can thus focus on the essential elements of real-time, e.g. time, scheduling, process interfacing, synchronisation, real-time operating systems, etc. Formal methods are not yet integrated in this course, though the author is well aware of their capabilities and their usage in practical projects [6].

In a series of practical exercises the students have to program elementary tasks on subjects such as digital and analog input/output, timing, synchronisation and tasking with a real-time operating system. VxWorks® and PC-targets are used. A model of a plant serves as an technical process to be controlled. The plant model is realised with the toy system Fischertechnik®, augmented with real-life industrial sensors and actors. For the exercises in this course, only the two robots of the plant model are used (fig. 1).

3.6. Technical applications of data processing

Design and project work of large real-time systems cannot be included in the real-time course, due to lack of time. Instead, parts of the course on *Technical Applications of Data Processing* are used for teaching the specific features of software engineering of large real-time systems [7-9]. In a "project game", the students then have to analyse, design, and implement a large system. The whole class constitutes the team, which has to organise itself according to the principles of general courses on software engineering, learned earlier during the course of studies.

In the past, a fictitious project "Automated Guided Vehicle System" (AGVS) was used several times for the project game, based on the author's experience with the control of such AGVS's [10]. Also, occasionally an elevator model (also made from Fischertechnik® elements) with micro-controllers networked via CAN is used. Currently, the project game is done with the plant model and thus builds on the basic skills from the exercises of the real-time course. The students have to analyse, design, and implement a control system for the whole plant model (fig. 2).

It must be conceded that the project game sometimes went seriously wrong and that numerous problems and mishaps showed up – just like in real projects. Nevertheless, the students even can learn from such failure and from their own mistakes.

3.7. Final year projects

Needless to say that many final year projects of the students of technical computer science are in the field of real-time systems. As usual at a Fachhochschule many of these final year projects are done in industry or in co-operation with industry. This is very attractive for the students because they get their hands on actual real-life problems and can measure the usefulness of their studies. On the other hand, these final year projects in industry are also a valuable feedback to the lecturers.

Some of the more ambitious final year projects include the redesign of parts of the spacecraft EURECA (European Retrievable Carrier) in co-operation with the European space agency [11], work on flight control systems in co-operation with DaimlerChrysler Aerospace and Eurocopter, data acquisition during test flights of the crew rescue vehicle of the international space station in co-operation with Kayser-Threde and many industrial automation projects from lemonade production to wood cutting.

4. Hardware-software codesign

Several experimental courses on VHDL had been held in the past, trying to teach hardware design principles to the students. The idea was to treat also hardware-software codesign issues, like in [12,13], because it is felt that hardware-software codesign will gain significantly in importance in the future. The students seem to have assimilated the issues very well. However, the VHDL synthesis tool in use showed annoying shortcomings, so that it was decided to suspend this course until better tools are available at the department. But it is firmly planned to resume this course in a reworked form next year or the following.

5. Conclusion

The way of doing real-time education by integrating all essential aspects into a standard curriculum has been shown. The approach can easily be implemented, because formally there is no specific curriculum, which would require approval by some higher authority. A prerequisite for this approach is a consent among all lecturers of the involved courses or otherwise some control of consistency of the involved courses. Similar approaches may be common among the Fachhochschulen in Germany. The method seems suitable to put emphasis on subjects other than real-time, too. The author is almost totally content with this approach.

Acknowledgement

The author would like to thank his former colleague at the Technical University of Munich, Gerhard Schrott, for many fruitful discussions about real-time systems and real-time education.

References

[1] Halang, W., Pereira, C., de la Puente, J., Shaw, A., Skubich, J., Tempelmeier, T., van Katwijk, J., Wedde, H., Zalewski, J.: *Real-Time Computing Education: Responding to a Challenge of the Next Century. In:* Real Time Programming 1997 (WRTP'97), Maranzana, M. (ed.). A Proceedings volume from the IFAC/IFIP Workshop. Lyon, France, 15-17 September, 1997. p 121-125. Pergamon, Elsevier Science, Oxford 1998.

[2] Trudell, B.: *Keys to Writing efficient code.* Embedded Systems Programming Europe, p. 7, November 1997.

[3] *Ada 95 Reference Manual.* International Standard. ANSI/ISO/IEC-8652:1995. January 1995. Intermetrics Inc., 1995.

[4] MC68000 8-/16-/32-Bit *Microprocessors User's Manual.* Motorola Inc., Prentice Hall, Englewood Cliffs, N. J., 1989. And: M68300 Family. CPU 32 *Central Processor Unit Reference Manual.* Motorola Inc. 1990.

[5] *ColdFire® Microprocessor Family Programmer's Reference Manual.* Revision 1 .O. Motorola Inc. 1997. http://www.mot.com

[6] Tempelmeier, T.: *Formal Methods-An Informal Assessment.* Internal technical report Dasa MT36 SR-1775-a. 41 pages. Daimler-Benz Aerospace, Ottobrunn Mai 1998.

[7] Nielsen, K., Shumate, K.: *Designing Large Real-Time Systems with Ada.* Intertext Publications & McGraw-Hill, New York 1988.

[8] Shumate, K., Keller, M.: *Software Specification and Design - A Disciplined Approach for Real-Time Systems.* Wiley 1992.

[9] Gomaa, H.: *Software Design Methods for Concurrent and Real-Time Systems.* Addison-Wesley, Reading 1993.

[10] Tempelmeier, T.: *Microprocessors in Factory Automation - A Case Study of an Automated Guided Vehicle System and its Integration into a Hierarchical Control Structure.* Proceedings EUROMICRO '86, Venice, September 15-18, 1986. Microprocessing and Microprogramming, 18 (1986), 647-656.

[11] Tempelmeier, T.: *Überarbeitung der Software-Architektur des EURECA-Satelliten. (Redesign of the Software of the EURECA Spacecraft. In German.)* ro Rosenheimer Hochschulhefte, May 1996, p. 63-67.

[12] Tempelmeier, T.: *A Note on Hardware-Software Codesign. In:* Real Time Programming Halang, W.A. (ed.). Proceedings of the IFAC/IFIP Workshop. Isle of Reichenau, Germany, June 22-24, 1994. P. 121-126. Pergamon Press, Elsevier Science, Oxford 1994.

[13] Schrott, G., Tempelmeier, T.: *Putting Hardware-Software Codesign into Practice.* Control Engineering Practice, 6 (1998) 397-402. Volume 6, Issue 3, March 1998.

Address of the author: Prof. Dr. Theodor Tempelmeier, Fachbereich Informatik, Fachhochschule, Marienberger Str. 26, D-83024 Rosenheim, Germany. Phone +49-(0)8031-805-510, fax -105, e-mail tt@fh-rosenheim.de, world wide web http://www.fh-rosenheim.de.

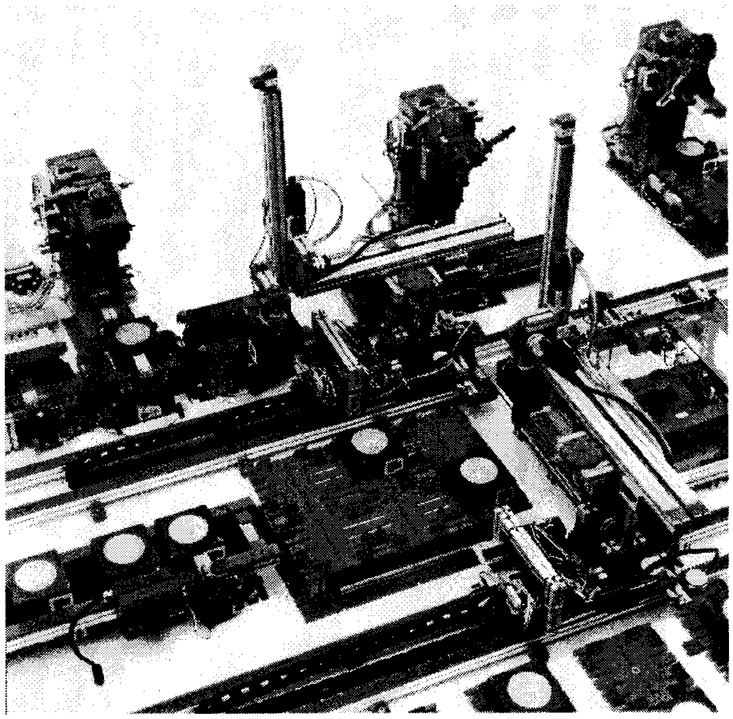

Figure 1: Detailed view (two robots) of the plant model used for real-time training

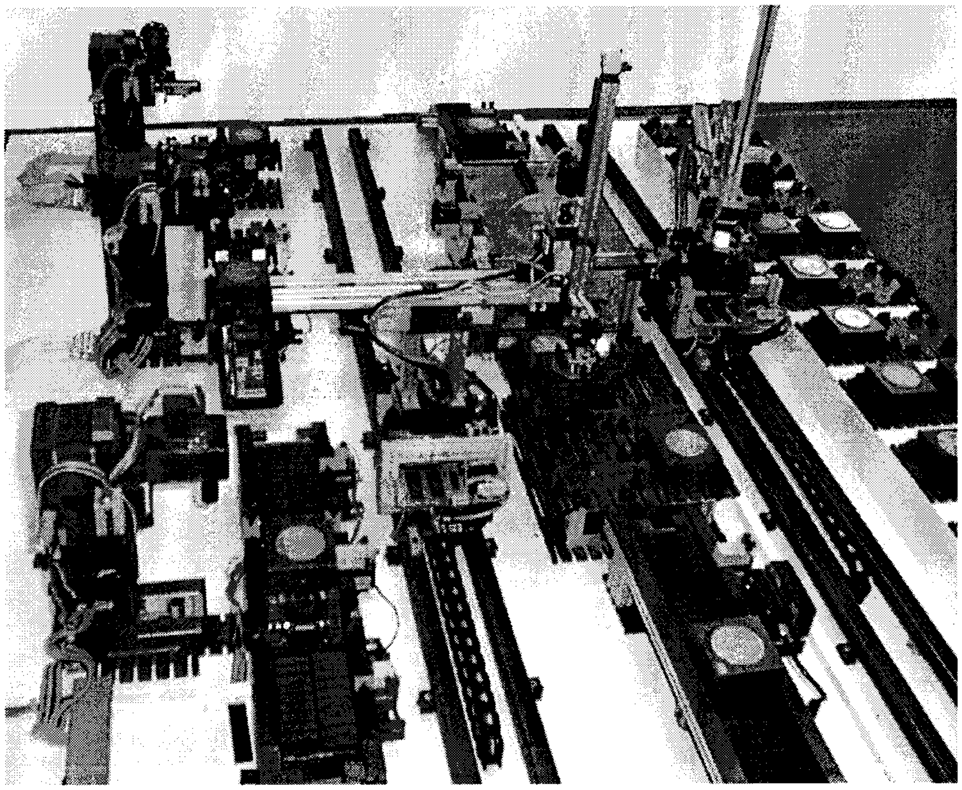

Figure 2: Overview of the plant model used for real-time training