UNIVERSITAT POLITÈCNICA
DE CATALUNYA

On real time teaching in non computer science curricula

Ramon Costa-Castelló

IOC-DT-P-2005-3
Gener 2005

Institut d'Organització i Control
de Sistemes Industrials

On Real Time Teaching in Non Computer Science Curricula

Ramon Costa-Castelló

***Abstract*— This work describes the topics taught on a real-time programming course. This course is part of the Degrees in Electronics, Mechanical Engineering and Automatic Control, and thus most students do not have a background in operating systems and related topics. This prevents us from using methods that are traditionally employed in teaching real-time topics to computer science students. The paper presents the specifics of the topics, methods and materials used on this course.**

***Index Terms*— Real Time System, Automation Software, Real Time Operating Systems.**

I. Introduction

Today, most automation-related devices are designed by means of digital technology. These technologies include PLCs (Programmable Logic Controllers), DSPs (Digital Signal Processors) and microcontrollers, which most electronic engineering students are familiar with. Although most of these devices have software libraries and development environments, their programming may be considered low-level. This kind of programming, however, is usually very time-consuming and can only be taken on in large productions or in cases in which a lot of computing effort is needed (e.g. DSPs). As the prices of powerful computers, such as PCs, fall and the market requirements that they must fulfil are extended, other ways of developing automation software are needed. The development of software for such systems requires skills and knowledge that exceed what is foreseen in generic engineering curricula.

Automation software has several characteristics that should be taken into account, the most important of these being concurrency (capability of simultaneously executing several programs) and time constraints. Although these requirements can be implemented using low-level programming schemes, such as cyclic executive or interruptive schemes, the use of higher level development techniques reduces development time and increases reliability. Real-Time Operating Systems (RTOSs) are a special family of operating systems (OSs) that are designed to allow tasks to fulfil their time constraints. This kind of tool is becoming very popular in the automation industry. Unfortunately, these tools are based on concepts and ideas that students outside of computer science (CS)/computer engineering (CE) curricula [1] are not familiar with. In order to minimize the problems students may have when dealing with these kinds of technologies, from the 1997-1998 academic year onwards, the School of Industrial Engineering of Barcelona (ETSEIB) has been offered a course that addresses topics related to Real-Time Systems (RTSs). This course is directed at last-year engineering students who are following generic courses that may include electrical, electronic and mechanical engineering. Although they may have studied elementary programming subjects, most of them do not have a background in OSs and other relevant topics in CS. This paper presents the specifics of the topics, methods and materials of this course,

Ramon Costa-Castelló is with the *Institut d'Organització i Control de Sistemes Industrials, Universitat Politècnica de Catalunya*, Barcelona, Spain. E-mail: ramon.costa-castello@ieee.org

the content of which is the result of several previous courses and analyses that the students see applied during the course.

II. Framework

Before following this course, engineering students at the ETSEIB follow at least two courses in basic computer science and programming, in which they learn basic programming methods such as *sequential search*, *up-bottom design*, *recursive design*, *divide and conquer* and *basic data structures*. This is the only experience in computer science that can be assumed in all the students. Their background knowledge does not include OO (object-oriented programming), UML (Unified Modelling Language) [2] or other tools employed in software engineering. On the other hand, most students are familiar with aspects of digital systems, such as timers.

Although in previous courses most algorithmic concepts are presented by means of an algorithmic language, in practical work the C programming language is used; therefore, knowledge of C can be taken for granted. Additionally, some of the students use this language in courses in microprocessor programming[1].

This framework limits the ways in which the concepts may be taught to students and conditions the goals of the course. For students to get the most out of it, the content and methods must be designed taking this framework into account.

The course's theory part is worth 3 credits, while the practical sessions are worth 1.5 credits, and they comprise ten lectures and seven laboratory sessions, all of these lasting two hours each. Due to the limited duration of the course, it is important to concentrate on relevant topics related to Real-Time Systems (RTS), without using complex tools that may be difficult for the students to grasp. Although the students who take the course do not develop OSs or RTOSs, they will have to use these tools; therefore, they need qualitative knowledge of their operation and associated problems.

[1]These courses may be taken at the same time as, or before, the course in question, depending on the curricular path followed by the student

III. Goals

Although several topics, such as real-time scheduling design methodologies [3], the design of drivers [4], object-oriented programming techniques[5], and security and reliability [6] are of great relevance to real-time systems, time restrictions and the students' academic background mean that many of these topics cannot be tackled during the course. The course is focused on those fundamental concepts and their implementation principles [7].

The most important goal of the course is to provide students with the fundamental concepts that allow them to implement simple real-time applications in the field of automation. These concepts are related to concurrency and time constraints. Although concurrency may be a very complex matter, the course's characteristics do not allow it to be studied in great detail. Concepts that are considered of great relevance are scheduling criteria and mutual exclusion access. In addition to these qualitative concepts, the quantitative tools that allow one to determine whether a given system will be able to execute all the necessary tasks whilst fulfilling time and data consistency constraints are deemed an important topic for engineering students (schedulability analysis [8])

In their professional lives, graduates will use tools to implement RTSs, which are presented in this course. RTOSs, as is the case with most technologies, are far from being ideal components, so it is important for students to learn what the main differences between ideal and real RTOSs are. As the effect of the most relevant of these technical phenomena can easily be included in schedulability analyses, it is important to show how this can be done in order to improve students' understanding of the phenomena.

Additionally, during their professional lives, students will

1) Introduction
2) Concurrency
   a) Task Concept
   b) Shared Resources
   c) Scheduling
      i) Time sharing
      ii) Real Time Schemes
3) OS Basics
   a) Functions
   b) Memory Management
   c) Device Drivers
   d) RTOS parameters
   e) API
4) High Level methods

Fig. 1. Syllabus outline

need to choose between different RTOSs, so it is important to teach the students the criteria that they might employ in choosing between them. Students are also shown the problems that they may encounter when using a traditional OS to implement applications with real-time constraints.

When students have completed the course, they should be able to implement simple automation applications by means of RTOSs and they should also understand how these components behave.

IV. Syllabus

Theory concepts are taught to the students by means of traditional lectures. In the following, the content of the syllabus (Fig. 1) and how this is organised as lectures is described. Although bibliographic aspects will be introduced in section VI, details are given on the basic reference work for each particular topic.

In order to spark the students' interest, the examples used will be related to automation, including filter, controller and state machine implementation[2]. The experience shows that this type of example especially motivates the students, and allows theory concepts from other courses to be linked to implementations in the real world.

1) Introduction {4 hours}

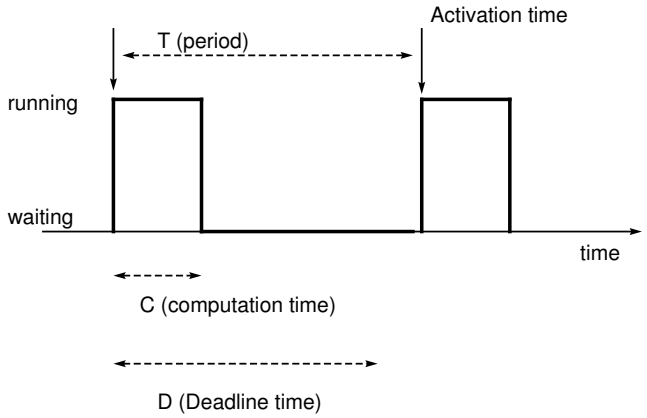

Fig. 2. Periodic task Ideal execution Chronogram.

The most important needs and characteristics of real-time systems are presented in this part [8, chapter 1]. Particular emphasis is placed on the parameters and time constraints used to define periodic and aperiodic tasks (Figure 2). An example that has been useful for the students concerns control task implementation. According to the theory [9], a certain algorithm must be executed periodically and the activation time should be kept stable. Most implementations will introduce some variations in the ideal activation time (e.g. jitter). In order to allow for these variations and for it to agree with the theory, a deadline which is around 10-15% of the period is permitted in the control community., It is then possible to obtain the periodic task parameters and formulation directly from this desired behaviour.

2) Concurrency {12 hours}.

[2]It is important to note that the specific topics of each implementation have been studied by students as part of previous courses and are therefore not dealt with on this course.

In this subject, we explain fundamental concepts related to the simultaneous execution of several programs and the benefits of describing the behaviour of a system by means of several programs, rather than building just one program to do the entire job. For example, when the control of a car is centralized, there are several tasks for each component, rather than a single program that controls everything.

This topic is divided into the following subtopics:

a) Task Concept [10, Chapter 4].

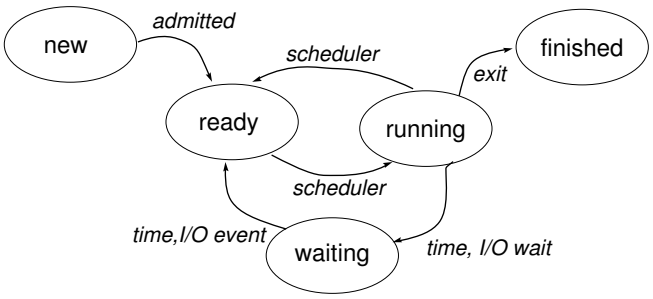

Fig. 3. Basic Task States and Transitions Between Them.

A task is defined as a program in execution. The different states in which the task may be are presented in this section (Figure 3). A qualitative interpretation of each of these states and transition analyses are also presented.

b) Shared Resources [10, Chapter 6]

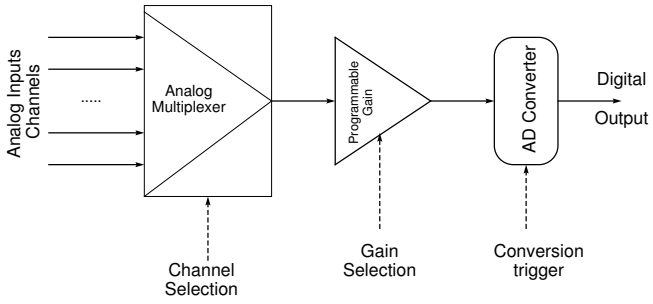

Fig. 4. AD/DA Card Logic Scheme.

The need for coordinating the different tasks running on a system is introduced. The shared resource concept is analyzed, and the need for *mutual exclusion* is presented. An example that has been used for several years is a system in which two tasks are implementing two independent control loops and both tasks are using the same AD/DA card. Like most low-cost AD/DA cards, it has several input channels that share the conversion elements (Figure 4). As a consequence, in order to ensure data consistency, it is necessary for the different tasks to use the card one after the other. In order to address this problem, the *semaphore* concept is introduced.

c) Scheduling

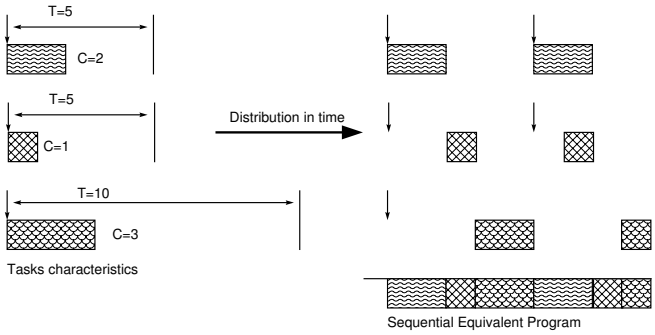

Fig. 5. Distribution in time to implement virtual concurrency.

Scheduling policies are a key concept in traditional real-time courses [11] [3]. Although most of the students enrolled on the course do not know much about OSs or schedulers, it is important to teach them about the role of the scheduler in real-time systems. In order to illustrate this concept properly, a system composed of three tasks is presented, and how these three tasks are divided into parts that are sequentially executed is explained. On the basis of this simple example, the concept of

cyclic executive[8] is introduced and the concept of virtual concurrency is visualized [Fig. 5].

The most important scheduling policies are introduced in a qualitative way. To make this presentation more effective and more attractive to the students, a simulator is used in the lectures. There are several simulators that can be used without cost; some of them are *RTsim* [11], *Cheddar* [12], and *RTA* [13].

In this topic, the following subtopics are addressed:

i) Time Sharing Schemes [10, Chapter 5]

The goals of a time-sharing scheme are presented; in particular, a *Round-Robin* scheme is analyzed. The difficulty of predicting the time response in this scheme is emphasized. Additionally, the role of priorities in this kind of scheme is discussed, and a range of reading material in which real OS scheduler behaviour is described is recommended to the students [14, Chapter 10] [15] [16].

ii) Real Time Schemes [8, Chapter 13]

The most relevant schedulers used in real-time systems are presented in two parts: how time is divided up between the tasks and how schedulability can be studied.

A) Static Priorities

Priority assignment methods are presented (RMPA, DMPA). The load concept and the main schedulability results are introduced. How task interaction may affect schedulability is analyzed; in particular, the *priority inversion* problem is described. In order to present this problem clearly, the well-known *pathfinder* [17] problem may be used. The *immediate ceiling priority* protocol is presented, and its definition and analysis tools are described.

B) Dynamic Priorities

Basic concepts related to dynamic priority scheduling schemes (EDF, LSF) are presented. Special emphasis is placed on the differences between dynamic and static priority schemes.

3) Basic concepts of OSs {12 hours}

In previous topics, an ideal framework was assumed. The goal of this part is to present technologic concepts that play an important role when an OS is used to implement a RTS.

In this topic, we discuss a set of technologies that are currently used in computer architectures that improve a system's performance but make estimating the computer execution time more difficult. These concepts include pipelining, memory hierarchies (cache, swapping) and DMA transfers among other things. All these concepts are described qualitatively by means of examples.

a) OS Funcionalities [10, Chapter 3].

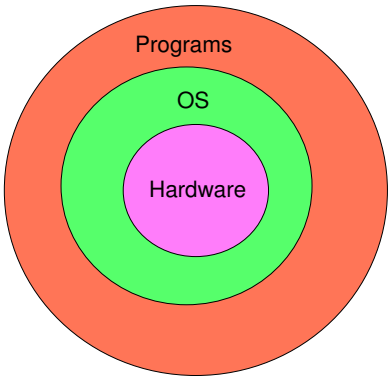

Fig. 6. OS role in a modern computer system.

The OS is presented as an abstraction layer over the hardware. The most important functionalities are described. The most important architecture approaches in OSs are described in a qualitative

way, and the relevance of these architectures when embedding the OS is analyzed.

In order to analyze the differences between a traditional OS and an RTOS, concepts like *reentrancy* and *preemptivity* are introduced to justify the need for special OSs in the implementation of RTSs.

b) Memory Management [10, Chapters 8,9].

Virtual memory and memory protection concepts are presented. The problems introduced by virtual memory in time determinism and the usefulness of memory protection when addressing a secure system are emphasized.

c) Device Drivers.

The role of the device driver is presented. As part of the real-time implementation chain, the device driver must be time-deterministic [18] [19][20].

d) RTOS Parameters [8, Chapter 16].

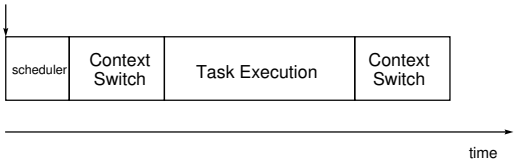

Fig. 7. Real Chronogram.

There are several differences between the ideal time response and the time response obtained in a real system, as the latter includes phenomena like latency and context change (Figure 7). All these parameters are defined and analyzed for several real RTOSs. Additionally, the issue of how these parameters might be included in a schedulability analysis is addressed.

The difference between a thread and a process is also defined[10].

e) Application Programmers Interface (API).

The system's call concept is presented, and the benefits of using a standard API for different OSs are discussed. POSIX real-time extensions are briefly presented [21].

4) High level Methodologies {2 hours}

The capabilities and functions offered by high-level development tools such as ADA[22] or JAVA-RT[23] are presented. The implicit handling of time and shared resources are presented as desirable tools.

V. Laboratory Experiments

A. Experimental platform

During the first years of the course, a high-level language was used to illustrate theory concepts. Specifically, the ADA language and a free GNAT compiler were used. Experience showed that students spent most of their time dealing with the language's syntax instead of on real-time concepts. In addition, this platform only allowed the students to implement soft real-time systems. To circumvent these problems, we decided to use the C language[3]. This language must be combined with an RTOS in order to develop an RTS, and there are several ways of doing this. As the university laboratories are equipped with general-purpose operating systems (GNU/Linux), a solution that would enable us to combine this OS with a real-time one was needed. There are several projects that allow GNU/Linux to be combined with a hard real-time environment, the most relevant being RTAI and RTLinux. Both projects were initially based on a Ms.C. project carried out by Michael Barabanov under the supervision of Victor Yodaiken in 1997.

In 1998, V. Yodaiken founded a company (FSMLABS) with the purpose of harnessing the development and the industrial use of RTLinux. One of the consequences of this was the introduction of a software patent of the main concepts used in RTLinux. This patent received a great deal of criticism from within the community, which had until then developed

[3]All of the students have used this language in previous courses.

and used RTLinux following a GPL licensing approach. Part of this community decided to initiate a new project called RTAI (Real-Time Application Interface). This new project was headed by Paolo Mantegazza from the Politecnico di Milano. Although this project initially used part of RTLinux's original source code, this project is currently based on ADEOS (Adaptive Domain Environment for Operating Systems), a nanokernel distributed under a GPL licence. RTAI is actively supported by a number of contributors who develop new kernel versions and new applications to run over it.

RTLinux has been divided in two main projects: RTLinuxPro and RTLinuxFree. RTLinuxPro is developed by FSMLAB under a commercial licence, while RTLinuxFree is an open-source version of RTLinux. RTLinuxFree is licensed under GPL and the RTLinux Open Patent Licence and is community-supported. One of the main contributors to RTLinuxFree is the Technical University of Valencia. As an example, two projects that are relevant from an academic point of view are the porting of an ADA compiler [24] and the development of a stand-alone version that can be used without GNU/Linux over it[25].

As in the course described, only basic concepts are taught. Both approaches offer the necessary functionality and performance. Due to historical reasons, RTLinuxFree is used on the course. In practice, most example codes used in the laboratory experiments may run directly on RTAI.

From a practical point of view, the most significant drawback in using RTLinuxFree is that it implements a PSE51 POSIX profile [26]. This profile has no memory protection and, as a consequence, an improper use of the pointer may make the machine crash. RTLinuxFree uses the GNU/Linux modules, so students must be briefly introduced to them.

1) Teaching by example
   a) Periodic Tasks.
   b) Time handling. Computation Time.
   c) Multitasking.
   d) Shared Resources.
2) Project sessions
   a) Design.
   b) Implementation and test

Fig. 8. Lab experiments outline

B. Contents

The laboratory experiments are divided into two blocks. In the first (first to fourth session), a *teaching by example* approach is used, while in the second block, a project approach is used (fifth to seventh sessions). An outline of the sessions is presented in Figure 8.

In every *teaching by example* session, a new set of concepts is introduced. Each session builds on the preceding one. The implementation of the concepts is presented by means of a set of very simple programs. Each program is presented as a fully functional source code that the student must compile, run and study in keeping with a set of predefined goals. Each code includes one or two syntactic errors, which force the student to analyze the code.

Additionally, a set of changes that modify the program's behaviour is proposed to the student. By modifying the program semantically and syntactically, the students gain experience in dealing with these new concepts and their implementation. A key factor is that students do not spend time designing the program from scratch.

At the end of each laboratory session, the student is asked to write a report . Writing this report forces the students to review the concepts learned during the session and analyze a set of important facts.

The following topics are addressed in the *teaching by*

example laboratory sessions:

P1 Periodic Tasks.

Three different ways of implementing periodic tasks are presented: *Busy waiting*, relative delays and absolute delays. The same codes are executed in Linux and RTLinux, which allows the students to compare the behaviour of a traditional OS and an RTOS. During the programs' execution, the mouse and other tasks are used to analyze the system's sensitivity to this kind of disturbance.

To illustrate this topic, a square waveform generation program is used. This waveform is sent to the parallel port and the PC speaker simultaneously, so that the student can analyze the performance by watching the parallel port through the oscilloscope or by listening to the speaker.

P2 Time handling. Computation Time (C)

Time is a critical concept in real-time systems. For this reason, time representation, time resolution, time stamps and the estimation of the execution time are studied during this session. The goal of the session is to develop programs that measure the time the CPU needs to execute a certain predefined code. This is done by using the OS's timers combined with a double loop technique and by using the oscilloscope combined with the introduction of marks in the source code (the parallel port is used as a mark).

This experiment is of great relevance because it presents methods for measuring the information needed for the schedulability analysis presented in the lectures.

P3 Multitasking.

In this experiment, students analyze the execution of three independent tasks. All three tasks generate outputs that can be observed by the students, so that they can see them being simultaneously executed. The commands that are needed to handle priorities are introduced in this session.

Students are asked to analyze the schedulability of the complete system.

P4 Shared Resources.

The use of semaphores to implement mutual exclusion is presented, and all related commands for handling and customising semaphores are introduced in this experiment. The example presented in the lectures is implemented (2b section)

Once the students have learnt the basic real-time concepts and the way of implementing them, a real problem is presented to them. All proposed problems require the use of several real-time concepts seen in the previous block. The project includes designing and implementing the software and analysing the time response. By way of example, in the 2003-2004 academic year, the following problems have been proposed:

1) Servo Control System Implementation.

The students are asked to implement a position control for a DC motor. The controller is composed of two control loops: one for velocity and the other for positioning at two different rates. Both control loops use the same AD/DA card and need to share information in order to implement the system.

2) Distance Measuring.

A Sharp GP2D02 sensor is used to estimate the distance. In order to interface this sensor with the computer, a predefined waveform containing time constraints must be generated through an IO card. Once this estimation is obtained, the goal is to filter it in order to obtain a clean measure.

3) Velocity control.

The velocity measurement is obtained by means of an encoder. The interface with the encoder is made following a protocol with time constraints through the

parallel port. Once the measurement is obtained, it must be filtered and used to close a velocity control loop.

These small projects help the students to deal with the different concepts and the way they are implemented. Many other projects could be used [27]; the only constraints are that they should be solvable in a few sessions and that they should touch upon different aspects of RTSs.

VI. Main Bibliography

Since the work by Halang [28], many interesting books on real-time systems have appeared in English. Most of these books may be used as textbooks, although unfortunately none of them exactly matches this course's syllabus.

One of the books that is most widely used in real-time teaching is by Burns & Wellings [8]. It includes most topics, such as concurrent programming, real-time scheduling, fault-tolerance and programming languages. Additionally, it includes a great quantity of teaching material. Unfortunately, this book is addressed to students of CS/CE, so it may be hard to follow for students on this course. To overcome this problem, it is used in combination with another classic book in the field of OSs. This book, by Silberschatz, Galvin & Gagne [10], introduces the most important concepts in OSs. Additionally, it offers a great quantity of teaching material. During the course, several chapters of this book are recommended as an introduction to several basic OS topics. The specific ways in which these books are used are described in the Syllabus section (IV section).

In addition to these basic books, several other books have been used and may be recommended as complementary to the basic ones. A classic book, such as the one by Laplante [29], offers a comprehensive overview of fundamental hardware- and software-related topics. Scheduling policies and analysis methodologies are covered in a great quantity of books, such as the ones by Butazzo [30], Krishna Shin [31], Stankovic et al.[32] and Liu [33]. All these books may be of great interest to teachers of these subjects, but they may be out of reach for students who are not in a CS/CE environment.

Other books that may be useful to students and lecturers alike are those that relate control systems to real-time systems. In this area, books by Bennet [34] and Svrcek et al. [35] have been particularly useful to the author.

Finally, books on specific topics, such as POSIX[21], ADA[22], JAVA-RT[23] or UML[2], may be of great interest to students and lecturers alike, particularly when they need to implement a specific application by using one of these tools.

VII. Complementary Information Sources

Today, the Internet is an information source that cannot be ignored. In the area of real-time systems, there are a great number of websites that can be used to support teaching activities. In order to guide students' steps in this area, several websites are initially recommend to them, some of which include online magazines (e.g. Embedded.com, Dedicated Systems), RTOS websites (ex. QNX, VxWorks, MaRTE OS, ECOS) and documentation web pages (e.g. OCERA project, Real Time and Embedded Guide). This information allows the students to come into contact with real components and information. One of the course's main goals is for students to understand the information available of their own accord, so it is important to determine whether they really do understand the information available and to obtain feedback from them in this respect.

Some of this material might also be of great interest to the lecturers, in that it will help them to keep up to date on these technologies. A number of examples may be obtained from these sources in order to update lectures.

VIII. Discussion

In recent years, the course has been attended by about 40-50 students per year. In general, the students' response to the

lectures has been positive, especially when theory concepts are illustrated by means of the practical examples presented in this paper. These examples are, in my opinion, a key issue that introduces students to the most important concepts, and is particularly important for those students with a background in electrical engineering or electronics. Another key issue is presenting RTOSs as a real technology that is far from perfect; although this requires the introduction of OS basics, it allows students to qualitatively understand several concepts that are of great relevance in practical RTSs. Topics introduced in the lectures are evaluated by means of a traditional exam; this exam is broken down into two parts, a theory section, in which short questions about relevant concepts are introduced, and a practical section in which a case must be studied. Using this approach, up to 80% of students pass the exam and reach a good understanding of what an RTS is.

As previously explained in the section on V-B, the laboratory sessions are divided into two parts: teaching by example and the project. This approach allows the lecturer to introduce the necessary tools in a short period of time, and the use of these new tools combined with the use of real components (e.g. sensors, actuators, etc.) is effective in sparking students' interest. During the first part, it is important for the lecturer to constantly interact with the students in helping them to understand the new commands. In the second part, the students are grouped in pairs, so that they can discuss the different approaches; at this stage, it is important for the lecturer to discuss the students' proposals with them. We have found that most groups manage to successfully finish their project on schedule, although in order to this, the projects must be built like "puzzles", using previous sessions as the building parts. Typically, the size of the code developed by the students is less than 100 lines. This way of proceeding allows the student to focus their thoughts on the relevant concepts instead of spending time on coding. The approach proposed allows the students to gain confidence in their knowledge of basic implementation tools, so that they eventually reach the stage when they are ready to implement simple RTSs.

IX. Conclusions

This paper describes the content of a course that deals with real-time systems, which is included in generic engineering curricula. The way in which traditional methods and syllabus used in CS/CE curricula are adapted to the generic engineering curricula is presented. The specific topics addressed and the methodologies used are described, and the examples used to illustrate the most relevant topics are presented. Details of the course are taken from 1998 to the present.

Our experience of teaching the course, as presented in this paper, may serve as a model for similar courses offered in other engineering programs. Additional information about this course can be obtained in the course webpage: http://www.ioc.upc.es/usuaris/RamonCosta/SITR/.

References

[1] A. Kornecki, "Real-time systems course in undergraduate cs/ce programs," *IEEE Transactions on Education*, vol. 40, no. 4, p. 9pp, November 1997.

[2] B. P. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*, 2nd ed. Addison-Wesley Pub Co, October 27 1999.

[3] G. Martinovic, L. Budin, and Z. Hocenski, "Undergraduate teaching of real-time scheduling algorithms by developed software tool," *IEEE Transactions on Education*, vol. 46, no. 1, p. 185=196, February 2003.

[4] A. Kornecki, H. Wojcicki, L. Peltier, J. Zalewski, and N. Kruszynska, "Teaching device drivers technology in a real-time systems curriculum," in *Proceedings on Real-Time Systems Education III*, Poznan, Poland, 21th November 1998, pp. 42–48.

[5] J. De La Puente, A. Alonso, M. Garcia-Valls, and J. Ruiz, "Teaching real-time systems at dit/upm," in *Proceedings on Real-Time Systems Education III*, Poznan, Poland, 21th November 1998, pp. 117 – 122, available in ftp://ftp.dit.upm.es/str/software/rta/.

[6] N. G. Leveson, *Safeware : system safety and computers*. Boston: Addison-Wesley, cop., 1995.

[7] W. Halang, "Teaching device drivers technology in a real-time systems curriculum," in *Proceedings on Real-Time Systems Education III*, Poznan, Poland, 21th November 1998, pp. 156 – 159.

[8] A. Burns and A. Wellings., *Real-Time Systems and Programming Languages 3/e. Ada 95, Real-Time Java and Real-Time POSIX. Alan Burns and Andy Wellings*, third edition ed. Addison Wesley Longmain, March 2001.

[9] B. C. Kuo, *Digital Control Systems*, second edition ed. Oxford University Press, 2002.

[10] A. Silberschatz, P. B. GalvinList, and G. Gagne, *Operating System Concepts*, sixth edition ed. John Wiley & Sons, Inc, June 2001.

[11] A. Manacero, M. Miola, and V. Nabuco, "Teaching real-time with a scheduler simulator," in *Frontiers in Education Conference. 31st Annual*, vol. 2, Reno, NV, 10-13 October 2001, pp. 15–19, available in online.

[12] F. Singhoff, J. Legrand, L. Nana, and L. Marcé., "Cheddar : an open and flexible real time scheduling framework," 2004, available online.

[13] J. A. de la Puente, *RTA*, Technical University of Madrid (DIT/UPM), 1998, available online.

[14] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*. O'Reilly, October 2000.

[15] M. Russinovich, "Inside the windows nt scheduler, part 1," *Windows & .NET Magazine*, July 1997, http://www.winntmag.com/Articles/Index.cfm?IssueID=22&ArticleID=302.

[16] ——, "Inside the windows nt scheduler, part 1," *Windows & .NET Magazine*, August 1997, http://www.winntmag.com/Articles/Print.cfm?ArticleID=303.

[17] M. B. Jones, "What really happened on mars?" 1997, http://research.microsoft.com/%7embj/Mars_Pathfinder/.

[18] R. Krten, "Device drivers & real-time systems," *Dr. Dobb's Journal*, pp. 34–39, October 1998.

[19] W. Hassan, "Writing real-time device drivers for telecom switches, part 1," *Linux Journal*, September 2001, http://www.linuxjournal.com/print.php?sid=4771.

[20] ——, "Writing real-time device drivers for telecom switches, part 2 of 2," *Linux Journal*, November 2001, http://www.linuxjournal.com/print.php?sid=5438.

[21] B. Gallmeister, *POSIX.4 Programming for the Real World*, 1st ed. O'Reilly & Associates, Inc., January 1995.

[22] A. Burns and A. Wellings, *Concurrency in Ada*. Cambridge University Press, November 1997.

[23] P. C. Dibble, *Real-Time Java platform programming*, 1st ed. Prentice Hall PTR, March 2002.

[24] M. M. Tello, J. Real, I. Ripoll, and A. Crespo, "Running ada on real-time linux," *Lecture Notes in Computer Science*, vol. 2655, pp. 322–333, 2003.

[25] V. Esteve, I. Ripoll, and A. Crespo, "Stand-alone rtlinux-gpl," in *Fifth Real-Time Linux Workshop*, 2003, pp. 149–154.

[26] Portable Application Standards Committee of the IEEE Computer Society, "IEEE Standard for Information Technology - Standardized Application Environment Profile-POSIX. Realtime Application Support (AEP)," iEEE Std 1003.13-1998. ISBN 0-7381-0178-8.

[27] M. Moallem, "A laboratory testbed for embedded computer control," *IEEE Transacion on Education*, vol. (To Appear), 2004.

[28] W. A. Halang, "A curriculum for real-time computer and control systems engineering," *IEEE Transactions on Education*, vol. 33, no. 2, pp. 171–178, May 1990.

[29] P. A. Laplante, *Real-Time Systems Design and Analysis : An Engineer's Handbook*. IEEE Press, January 1997.

[30] G. C. Buttazzo, *Hard Real-Time Computing Systems. Predictible Scheduling Algorithms and Applications*, ser. The Kluwer International Series in Engineering and Computer Science, J. A. Stankovic, Ed. Kluwer Academic Publishers, 2002.

[31] C. Krishna and K. G. Shin, *Real-Time Systems*, W. McGraw-Hill, Ed., 1997.

[32] J. Stankovic, M. Spuri, K. Ramamritham, and G. C. Butttazzo, *Deadline Scheduling for Real-Time Systems. EDF and Related Algorithms*, ser. THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE. Boston: Kluwer Academic Publishers, 1998, vol. 460.

[33] J. W. S. Liu, *Real-Time Systems*, 1st ed. Prentice Hall, June 2000.

[34] S. Bennett, *Real-Time Computer Control*, second edition ed., ser. Prentice Hall International Series in Systems and Control Engineering, M. Grimble, Ed. Prentice Hall, 1994.

[35] W. Y. Svrcek, D. P. Mahoney, and B. R. Young, *A Real-time approach to process control, Solutions Manual*. John Wiley & sons, cop, June 2000.