Experiences with an Introductory Real-Time Software Course at Delft University of Technology

Jan van Katwijk
Faculty of Mathematics and Informatics
Delft University of Technology
P.O.Box 356
2600 AJ Delft, The Netherlands
J.vanKatwijk@its.tudelft.nl

Janusz Zalewski
Department of Electrical & Computer Engineering
University of Central Florida
P.O.Box 162450
Orlando FL 32816, USA
jza@ece.engr.ucf.edu

Abstract

In this paper, we report on introductory course on real-time software engineering at Delft University of Technology. We discuss the setup of the course in relation to the structure of the curriculum at Delft, briefly discuss some of our standard cases and focus on one of the case studies used in the course. Based on our experiences, we present some conclusions.

1 Introduction

Dutch academic engineering curricula, i.e. curricula leading to a Master of Science or Master of Engineering degree, take 5 years nowadays. This is a recent change from a 4-year curriculum, which was in place since the mid eighties. The structure of the programmes is as follows:

- a three year basis study, consisting of a first year, concluded with a formal exam, the *propedeuse*, and a two year part, called the *basis doctoral part*
- a two year specialization part, leading to the degree of Master of Science (an engineering degree).

In the informatics curriculum, there is a single basis part for all students in the program. This basis part contains regular computer science issues such as mathematics, logic, programming and operating systems, an introduction to software engineering, as well as information system design and some basics in regular engineering and in business administration.

The second part in the curriculum is a specialization part, offering specializations bound to the seven chairs in informatics. These chairs are labeled as follows:

- Exploitation of Information Systems,
- Design of Information Systems,
- Data Bases,
- Parallel and Distributed Systems,
- Knowledge Engineering in Technical Domains,
- Computer Graphics and CAD/CAM, and
- Software Engineering.

The *structure* of the specialization part is fixed, some courses have to be taken from the own selected specialization, a given number of credit points has to be obtained in the other specialization directions, or can be taken from other engineering programmes. The course part of the specialization takes one year, a larger assignment has to be performed in a second year.

Since the basis study is fairly broad, students coming in their fourth year have a broad, though somewhat shallow, knowledge on some of the more systems-oriented elements of computer science and they lack both knowledge and training on a number of topics, one being real-time software. Our setup of an introductory real-time software course has to take this into account.

2 An Introductory Real-Time Course

2.1 General

In 1992, we introduced an introductory course in real-time software development. The course is a specialization course for fourth year students in informatics and senior students in other engineering disciplines. The objective of the course is to give students some insight in the nature of time-dependent software and systems and to create some understanding by students of typical design and verification issues in the development of real-time software. Due to the limited time available for the course, attention is focused more upon a passive understanding of issues involved than on an active involvement in the development of real-time software. After some experimentation, we finally reached in 1996 the course structure as described in this paper.

2.2 Course Material

The number of textbooks on introductory real-time software development is surprisingly small. After a few attempts where we used existing material, often books and reprints of articles, we decided to develop lecture notes according to our own goals. Developing course material is quite a job, the rationale for undertaking the effort was simply that neither the structure of the available material nor the level was felt to be appropriate for students in our curriculum. In 1996, we started with a preliminary version of the notes. After having offered the course three times in its current form (twice for regular students, once for a commercial company) the structure and contents of the course is fixed now and we are finalizing the lecture notes. The notes, together with the example programs referred to from the notes are available to students and other interested parties on the internet from `ftp://dutian.twi.tudelft.nl/a435`[1]. The example programs are in Ada, C (with bindings to various pthread packages and to POSIX) and Java.

2.3 Structure of the Lecture Material

In developing real-time software, the ability to handle concurrency in design and implementation, together with the ability of handling time and the possibility of precise specifications are instrumental. Therefore these issues are essential in the education of real-time software developers. These issues, processes, time, specification and verification are therefore made central in the material we have been developing so far. A second important point in the development of the material is the use of standardized structures in the solutions, i.e. typical architectures.

The course takes 7 three hour lectures. The lectures are structured as described below:

1. **The notion of process.** In order to create a reasonable understanding of the notion of process, we refresh the latent knowledge on processes (tasks) by discussing a single example, the problem of the dining philosophers. We introduce the Ada language, especially its tasking capabilities and the protected record mechanism, and show the use of the features using this example. Starting with the Ada 95 encoding, we introduce concurrency and synchronization mechanisms as vehicles for modeling solutions. These mechanisms, encoded in other notations and environments are then discussed. In C we introduce the notion of thread, to implement the same example, show an implementation for threads, we show how the philosophers example can be implemented using POSIX processes, and present the example using Java. The simple example of the dining philosophers clearly demonstrates the need for synchronization and communication mechanisms. As a gadget, we even demonstrate a classical algorithm that is used to implement semaphores using file system locks and demonstrate their use in implementing the philosophers example. Since most of this part refers to latent student knowledge, teaching it takes only one block of three hours.

2. **The notion of time.** The second block of the course is completely devoted to manipulating time. By using some simple examples, we demonstrate the need for both calendar time and durations. We introduce operations for reading calendar time and reading durations, we introduce operations for delaying the execution of the current thread of control (various delay mechanisms) and the notion of an alarm timer. We introduce the need for some form of asynchronous transfer of control, demonstrate its use using the Ada 95 mechanisms and show the implementation of such a mechanism in POSIX. The guiding example in this section is what we call *the safe periodical task*, i.e. the task that ensures itself that it will take appropriate measures after deadline overrun. The structure of such a task is demonstrated in Ada, Java and in C, using threads and POSIX primitives. As a gadget, we show how to bind to DOS and BIOS calls and how to poke in the various registers of the PC (in a DOS environment). One of the examples we are giving is a microsecond clock using a standard PC under MS-DOS. Although a significant amount of material is discussed, it is basically simple and takes one block.

3. **The notion of architecture.** After having introduced some basic technology, we pay some attention to design issues in real-time software. We strongly believe that a good design process starts with the selection and instantiation of an appropriate architecture. In order to facilitate this, we introduce a simple standard architecture that can be used for a reasonable class of systems (see Section 3), and demonstrate its use by specifying some design rules and by giving some examples, one being somewhat larger. The notion of architecture, and the different implementation models take quite some time to discuss. Including a detailed discussion of the case study (specified later), this part takes approximately 4.5 lecture hours.

[1] The course number is a435.

4. **The notion of scheduling.** Introduction of concurrency introduces additional degrees of freedom and design. However, in most cases we need to constrain this freedom, especially when deadlines and start times are to be met. In this part of the course we first demonstrate the need for control over scheduling. Then we introduce scheduling as a general constraint on the execution of a program. We next introduce a simple notation to express scheduling constraints on execution sequences, and we demonstrate a simple tool to visualize the behaviour of 'programs' written in this notation, illustrated in an example below.

   ```
   --
   name First_Come_First_Served
   evaluate R  -- arrival time

   -- simple demo, two tasks running under
   -- EDF and under FCFS, but not under RM
   task t1
      within 10 * I + 1
      readyat 10 * (I - 1) + 1
      repetitive
      basicunit (5)
   endtask
   task t2
      within 15 * I + 1
      readyat 15 * (I - 1) + 1
      repetitive
      basicunit (7)
   endtask
   ```

   We then discuss at some length the classical static scheduling algorithms, including some basic transformations, e.g., from aperiodic to periodic tasks. This part also takes 4.5 lecture hours.

5. **Analysis and verification.** One of the major drawbacks in designing software with non-functional constraints is that validation w.r.t. such requirements is (too) late in the development process. Often, only after building the system, one is able to verify whether or not the system meets its non-functional (temporal) constraints. Furthermore, one should be able to make some statements on safety properties of the system. We therefore introduce a variant of timed automata (XTG's) and introduce some of the basics in verification using model checking techniques. Our standard case used in the class is an extended version of the philosophers, called the drinking philosophers, in which the philosophers share a container with liquid food that will be filled from outside whenever its level is low (for a description, see Section 3.3). This part takes one lecture of three hours.

6. **Practical cases.** It is a good habit to have a real practitioner indicate what industrial requirements are. Materials for this part are not part of the regular course notes. This part takes one block of three hours.

2.4 Assignments and Cases

Associated with the course is a short (30 clock hours) lab assignment in which students have to use some of the basics to implement a simple system. During the last two years we take the drinking philosopher's problem as case study, where the students get a specification in XTG's and the code of the 'system' and are asked to redevelop the modified case, say something on more formal aspects and demonstrate the resulting system. The final examination in the course is by a small assignment, usually having something to do with the development in the context of one of our demonstration cases.

3 Case Studies

3.1 General

Software development starts living when illustrated with examples, so during the past years we collected a number of visually attractive case studies. Typical visually attractive case studies are:

- a toy railroad and a computer-based controller,
- an extended version of the classical problem of the philosophers (called the drinking philosophers),
- a distributed system synchronizing running wheels,
- a pendulum.

The objective of the case studies is to demonstrate to students that even with very simple examples design decisions are not automatically correct. Each of these cases is visually attractive and malfunctioning of the developed systems is indeed very well visible.

3.2 The Railroad Controller

The railroad controller is a control system for a simple computer controlled toy railroad. The railroad has been used for a couple of years as a vehicle for class assignments and specifications. The assembly contains some 20 meters of rail and 5 locomotives. The objective of the controller is to control the motion of the 5 locomotives such that each of them will follow a prescribed route and enter some stations on a prescribed time, obviously without having two or more locomotives collide. The controller is able to read the status of rail segments (occupied or not) and is able to issue commands to set the speed of each of the locomotives and commands to set switches. During the last years, we developed a complete system for railroad control. Many tools were developed for supporting the development of the controllers. Furthermore, one of the tools helps in visualizing the controller state during the execution. Showing the controller state and the real-life situation is extremely helpful in having students understand the behavior of controllers.

The railroad controller has been used intensively as a vehicle for specifications. In the past four years, we developed around a dozen different specifications (most of them fairly formal ones) for the controller and its environment. The problem description and a first specification using a variant of VDM is available in [2]. Other specifications are given in Mosca, a locally developed specification notation [5], in PAISLey [4], and in Astral [3]. A comparison of different specifications is given in [6]. The software is available through `A.W.W.M.Biegstraaten@twi.tudelft.nl`.

3.3 The Drinking Philosophers

The 'drinking philosophers' is a small self contained example system used in the classroom to demonstrate the use of different kinds of timed automata in analysis and design of real-time systems. The problem is a combination of the classical problem of the dining philosophers and the water-level controller. The philosophers only take liquid food, through a reed, connected to a container. As soon as the level in the container drops below a given level, some external agent causes the container to be refilled up to another specified level. Students get a formalized specification of the problem [7], written in the XTG notation, a locally developed variant of timed automata and are asked to implement some parts of the problem in C, using a locally developed pthread package, allowing a binding of the C code to a Java program responsible for the visualisation.

3.4 Synchronizing Wheels

The synchronizing wheels case study is a small case where two different computer systems, connected through a TCP connection, cause two wheels to run synchronously. Any change in the rotation speed of one of the wheels will be followed immediately by a similar change in speed of the other wheel. This case is still under development.

3.5 The Pendulum

The pendulum case aims at letting a small bullet hit a moving pendulum under computer control. It is a typical example of a small one-shot controller that is discussed in some more detail below.

In the experiment, we use a physical pendulum, with a disk at the end (of approximately 5 cm diameter and around 1 cm thick). The user starts the experiment by invoking the program, manually lifting the pendulum, releasing it, and typing a return character on the keyboard of the controlling

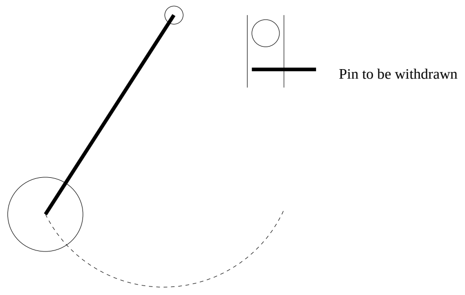

Figure 1. Schematic description

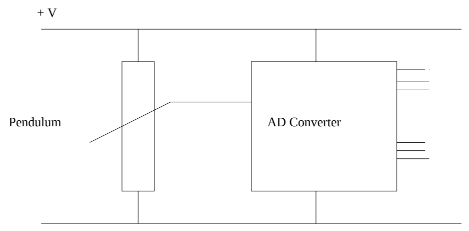

Figure 2. Electric connection

computer. The desired effect is that a small bullet is released from top, at a moment such that in its fall it hits the disc. A schematic description of the construct is given in Figure 1.

Connections between controller and controlled device are through a *sensor* and an *actuator*. A potentiometer, connected to the axis of the pendulum acts as sensor. A small deviation-dependent electrical voltage is obtained connected to an AD card in the controlling PC (Figure 2). A relay, connected to a pin, acts as actuator. This actuator is connected to a DA converter on the same card in the controlling PC.

The axis of the potentiometer is connected to the axis of the pendulum. The deviation of the pendulum causes the voltage over the output pins of the potentiometer to change. The output of the AD converter is a 12 bit value, so the potential range of values we have to deal with is $0..4095$.

The mathematics of the problem is straightforward. The interested reader is referred to the lecture notes where a detailed description is given [1].

3.5.1 Systems Design and Implementation

This problem is a real control problem, since the controller reads a time-continuous signal through a sensor and next, based on the state of our computation, issues a signal through an actuator at a computed moment in time.

In our course we put some emphasis on using standard architectural approaches rather than reinventing all wheels. For a large class of real-time control problems we provide a single 'standardized' architecture, in a generic form given in Figure 3.

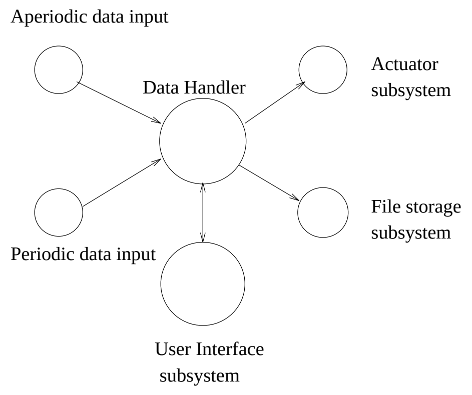

Figure 3. Standard Architecture

The architecture merely states that in many control environments we need some:

1. periodical tasks to read data (polling continuous data streams);
2. aperiodic data (reacting on events);
3. a computation module, mapping state and input to the various output signals;
4. a file storage module, ensuring that data can be stored;
5. actuator module(s), ensuring that the physical world gets stimuli from the controller;
6. a user interface module.

In the course, we spent (quite) some time showing what is the effect of different implementation styles for instantiations of this architecture, both with and without tasking. We also show the importance of selecting control in the implementation of an instantiation of this architecture.

For the pendulum controller we instantiate the architecture and decide to use a tasking implementation, with:

1. a periodical task for reading, preprocessing and buffering sensor values;
2. a periodical task computing the optimal dropping time, taking its input from the sensor reading task;
3. a main program, also acting as user interface;
4. sporadic task for controlling the relay.

The tasks exchange data through one-slot buffers, implemented using Ada 95 protected records.

The operational environment needs some further consideration. The input is through an AD converter card, output is through a DA converter card on the same port of the PC. Operations on the card, i.e., reading a 12-bit value or writing a value, take two port accesses, so locking on this level is also inevitable.

The details of the design and the encoding of the controller (implemented using GNAT3.07 under MS-DOS) can be found in the lecture notes.

3.5.2 Observations and Experiences

The controller for the pendulum designed this way is encoded using four Ada tasks and approximately 1500 lines of code. Although not extremely large, the case has a number of typical characteristics that make it interesting:

1. Time as manipulable entity becomes important. Measurements of the deflection angle are stored and the design is such that the values will be used later. This approach requires that measurements are accompanied by the time they are taken. The output of the sensor reading task is therefore a sequence of tuples *(deflection value, time)*. This in itself raises questions about uninterruptability of operations, is it really required to ensure that the sequence measuring deflection angle, measuring time is uninterruptable. If yes, how do we arrange that, if not, can we estimate the maximum error in time measurement.
2. Required accuracy in time. What is the required accuracy? The length of the trajectory is (for a full swing) 0.75 m, while the period is 2 seconds. So, the linear speed with which the disc moves is 0.75 m/s. Since the diameter of the disc is approximately 5 cm, there is an interval of app 60 microseconds providing a window where the bullet can be released and hits the disc.
3. Accuracy of measurement. The AD converter gives a 12-bit value, i.e. a resolution of $\frac{1}{4096}$. One might ask (and one should ask) how many of those bits are really meaningful. The actual obtained range of values we get from the converter is between 83 and 1555 for a swing of approximately 60 degrees (about one radian). So if we assume that the length of the pendulum is one meter (actually, it is somewhat less), the trajectory of the disc is less than 1 meter. In this trajectory, the range of values spans approximately over 1500 units, so a unit of measurement corresponds to approximately 0.6 mm in the trajectory of the pendulum. The main cause of inaccuracy is obviously the potentiometer. The length of the carbon trajectory is approximately 5 cm. Splitting that up in 4096 units results in 0.00125 cm per units. This shows, even with a non-moving pendulum, that measuring gives unstable values: values that are unstable in the last few bits.
4. The case study runs on a simple computer under MS-DOS and is implemented using GNAT Ada. Under

MS-DOS, the accuracy of the Ada clock is 0.1 second. This is insufficient when swinging with full speed. We realize, however, that measuring time can be done at the microsecond accuracy level so an improvement would be to use the microsecond timer that is discussed in the second lecture.

5. The command to release the bullet is given after a `delay_until` in the actuator task. It is interesting to have students realize that the semantics of the `delay_until` merely specify a minimum delay, not necessarily a precise delay. So, other measures have to be taken to ensure the invocation of an action within a given interval of time.

These observations show that the case provides ample opportunity to discuss a number of real-time software related issues. Students get the full code and it is simple to see the effects of modifications. We therefore feel that this case study is a worthwhile one.

4 Conclusions and Recommendations

In this paper, we briefly discussed the structure and some issues in our introductory course on real-time software development. We presented the structure of the lectures, which are available on the internet, as lecture notes together with all program code referred to in these lecture notes. The chapters covering the lectures 1 to 4 are pretty stable, the chapter on specification and verification is still under development. We welcome any kind of comments on these notes and any form of cooperation to have these lecture notes evolve. Furthermore, we are convinced that having suitable cases demonstrated during the course, and cases suitable for having students do some assignments, have a great value in the educational process. In this paper we described four such cases. We would welcome all kinds of comments on these cases and would like to exchange information on similar cases with other educators.

References

[1] J. van Katwijk, J. Zalewski, Introduction to Real-Time Software Development. Lecture Notes, `ftp://dutian.twi.tudelft.nl/a435`

[2] A. Biegstraaten, K. Brink, J. van Katwijk, W.J. Toetenel, A Simple Railroad Controller: A Case Study in Real-Time Specification. Report 94-86, Faculty of Mathematics and Informatics, Delft University of Technology, 1994.

[3] L. Bun, J. van Katwijk, An Astral Specification for a Railroad Controller. Report 95-104, Faculty of Mathematics and Informatics, Delft University of Technology, 1995.

[4] J. van Katwijk, W.J. Toetenel, A Simple Railroad Controller. A Case Study in Real-Time Specification Using PAISLey. Report 95-27, Faculty of Mathematics and Informatics, Delft University of Technology, 1995.

[5] A. Biegstraaten, K. Brink, J. van Katwijk, W.J. Toetenel, A Simple Railroad Controller. A Case Study in Real-Time Specification Using Mosca. Report 94-87, Faculty of Mathematics and Informatics, Delft University of Technology, 1994.

[6] J. van Katwijk, W.J. Toetenel, Comparing Formal Specifications by Measuring. Proc. 2nd International Workshop on Real-Time Computing Systems and Applications. Tokyo, 25-27 October 1995, IEEE Computer Society Press, 1985, pp 184-192.

[7] J. van Katwijk, R.M.C. de Rooij and W.J. Toetenel, Introduction to Real-Time Systems: Case Study and Exercises, `ftp://dutian.twi.tudelft.nl`