Real-time Lab Excercises: A teacher's Dilemma

Erik Herzog and Peter Loborg[1] and Simin Nadjm-Tehrani

Dept. of Computer and Information Science,
Linköping University
{erica,simin}@ida.liu.se

Abstract

This paper describes our design of real-time systems laboratories in an integrated theme of study which includes automatic control.The theme appears at the end of the third year of a 4,5 year master of engineering programme, which adopts Problem-Based Learning (PBL) as a main pedagogical method. We describe the rationale behind our choice of application area, the lab environment, and the operating system used. The paper concludes by giving some qualitative evaluations as well as some quantitative measures based on limited data.

1 Introduction

In this paper we describe our experiences with developing laboratory exercises for a course in the third year of a 4.5 year Master of Engineering programme: the Information Technology (IT) programme. We describe the dilemma faced by any course designer when relatively small number of hours are available, and the goal is to achieve some insight in some central aspect of real-time system development; an area wide enough for several courses, from theoretical analysis of scheduling algorithms to practical design and implementation of dedicated hardware, software and middleware.

This problem may be considered irrelevant, in the sense that the ACM 2001 curriculum does not list real-time systems as a core area for computer engineering students [1]. However, there is a philosophy behind the design of our programme which apparently goes against this recommended curriculum. The idea behind our educational programme is to use the Problem-Based Learning (PBL) methodology [3, 2], whereby three elements are central: study of realistic cases in integrated settings, self directed learning, and group discussions. The first item, in our case, has led to a choice of combining the topics of real-time systems with automatic control (which is considered as core for all engineers). In this way both topics strengthen and motivate each other. However, our design possibilities are limited by allocated time.

This choice implies that instead of a dedicated course in real-time systems we have a "Theme" which is an integration of automatic control and real-time systems. Note that themes are studied one at a time during a 4-7 week period with no other parallel courses. In fact the whole spring term (semester), the 6th in the study programme, was designed as a whole – covering four themes: concurrent programming and operating systems, followed by the theme which is the subject of this paper, a theme on control theory integrated with transform theory and linear algebra, and a theme on ethics for engineers [5]. The time allocated to real-time systems corresponds to 1,5 credit points out of 18 obligatory credit points for the integrated term. This paper was written after running the themes over three years.

2 The application example

Having an integrated theme with automatic control meant that we could concentrate on real-time issues related to this class of applications, as opposed to other fields such as real-time communication. To have an integrating example we chose a fine measurement robot from the swedish company C. E. Johansson. The robot is demonstrated for the students during a field trip at the beginning of the theme. Precision within a few micro meters and speed of several measurements per second are two aspects of the real system, making it interesting both from the control point of view and in terms of real-time constraints. Thus, the automatic control labs first treat the issues of servo control, stability etc. on one motor, and the real-time labs treat a collection

[1] Currently employed at Carlstedt Research and Technology, Linköping, e-mail: petlo@crt.se.

of computation processes including 3 motors (one for each dimension), a path generator, a supervisor and so on.

Sporadic events are present in the example in a natural manner (hitting an end-position, or appearance of an obstacle), and their effects can be considered both in terms of selection of scheduling policy and in treatment of exceptions (i.e. adding fault-tolerance).

3 The topic areas

Next it was decided that the labs should not teach programming. Thus, issues related to programming language constructs and styles were excluded. This contrasts another course which we give for other educational programmes – where Ada is the programming language for the labs, and synchronisation mechanisms and scheduling labs are implemented with emphasis on programming.

Instead we have decided upon an environment whereby the students encounter the system development stages. Matlab was chosed as a vehicle for studying design and code-generation, and a separate tool was used to visualize the effects of non real-time versus real-time scheduling principles.

First, the students simulate a simplified model for one robot arm and a servo motor steering the arm in one dimension. Then they add servo controllers for other dimensions, a (pre-prepared) path generator and a supervisor for detecting end-positions. ¿From this extended model, C code is generated using Matlab's automatic code generator.

The approach taken has enabled the creation of an appealing, intuitive environment. Students do not have to master complex real-time systems and tools[1] for completing the exercise. The goal is to focus entirely on real-time aspects of the exercise. However, before getting to our ultimate goal some obstacles had (and some still have) to be removed – namely which operating system to use as a target platform for Matlab code generation.

4 Developing the excercise environment

Tools such as Matlab/Simulink and MatrixX have code generation capabilities designed for a varity of different targets, ranging from Posix compliant real-time kernels through rather raw machines (PC/DOS and DSP's) in which multi-threading has to be simulated or implemented by the code generator, down to what is considered non real-time machines (such as Unix, Windows etc.) where the purpose is to speed up a simulation. Regardless of target system, the code generation is designed to implement the same overall behaviour (as specified by the model) regardless of the chosen target. This is, however, only true as long as computational resources are not exceeded.

4.1 Selecting a real-Time operating system

Matlab was choosen as the excercise environment since the university holds a site license and since the students are already familiar with the tool. Matlab's Real Time Workshop (RTW) [4] has built in support to generate code for several targets. They also provide an interface to a "general real-time" target, to be used as a basis for porting the code generator to new targets. The Unix system is considered a non real-time target, although many of the available Unix variants claim to have a posix compliant real-time part. The majority of the machines available for our students at our department are running such a Unix system[2].

In our case the choice of target system boiled down to a choice between buying a few special target machines (a bottle-neck situation with any decent sized group of students) or to try to utilise the real-time capabilities available on our Unix machines. Buying special target machines, e.g., VxWorks platforms from Windriver, would provide us with a good set of tools for manipulating the Matlab model under execution and tools to monitor the real-time behaviour on the remote target (WxWorks/Tornado), showing detailed information about e.g., task switches and semaphore usage. On the other hand, we already had a tool for visualizing the tasking behaviour of our Unix machines. So, we opted for the more generic, lower cost and lower development time option. We decided to try out the real-time scheduling classes available, using our own process logging facility.

4.2 Executing a Matlab/Simulink model

The Simulink model created in Matlab consists of a set of function blocks, where output from one block is fed through "wires" to one or several other blocks. Normally these models describe a continuous system, but it is also possible to describe purely discrete systems (e.g., sampling controllers) as well as mixed systems. When a system is discrete with several different sampling rates it is called a multi-rate system, and the only restriction on such systems imposed by matlab is that all the rates must be a multiple of all slower rates. In a mixed continuous and multi-rate system, the continuous part is executed together with the fastest sampled discrete part.

[1] The students have been using the Matlab/Simulink system prior to this course, and the extra part needed for generating C code is the activation of one button.

[2] The SunOS version 5.6

4.3 Porting to a new target

Porting the C-code generator for a new target is almost only a question of renaming the primitives used for spawning threads, manipulating semaphores and managing interrupts. Where and how to do this is well documented [4]. Converting a time-sharing based Unix system running in a network environment into a real-time target does, however, require some extra attention.

The SunOS we are using is equipped with not only one scheduler, but a set of schedulers implementing different scheduling policies (called *scheduling classes*). Schedulers are loaded into the OS when required, and it is possible to write and add new schedulers. Our machines are equiped with three different schedulers; time-sharing, interactive and real-time. The difference between the time-sharing and the interactive is minor, the interactive is a slightly different tuned version of time-sharing where the nice value (a sort of priority) is manipulated by the window system in order to achieve faster response for the application which has the user's attention. The real-time (RT) scheduler, on the other hand, operates strictly priority-based. High priority tasks allways interrupt tasks with lower priority, and all non RT-tasks have lower priority than any RT-task. If an RT-task enters an infinite loop, the entire Unix machine will hang. Thus, superuser privilages are required to put a task into an RT scheduling class. Once a task is running in RT-mode, all tasks spawned from that task inherit the RT capabilities, and no special privilages are required to change priority of a task within the RT-class.

The SunOS does, however, create a few extra tasks for some internal management duties as soon as a second task is created in a program. Thus, it became important for the main program to start off as a low priority RT-task, spawning tasks for the different sampling periods and thereby also the internal management task at the same low priority. These new tasks start by raising their own priority to a level appropriate to the sampling rate, and the main program would finally raise its own priority one step above the priority of the fastest sampling task.

4.4 Visualizing the real-time execution

The C code generated by Matlab's automatic code generator has been instrumented to provide logging of thread related system events, and the resulting program is parameterized with what scheduling policy to use. Every time a task is activated, preempted or restarted the timing information is recorded for later use. Finally, the log is viewed post-mortem, and the effect of the selected policy is presented graphically. Figure 1 shows a screen dump for a part of the log of a typical student session.

5 The resulting excercise

The resulting excercise is presented on the course webpage [7]. First, the students simulate a simplified model for one arm and a servo motor steering the arm in one dimension. Then they add servo controllers for other dimensions, a (pre-prepared) path generator and a supervisor for detecting end-positions. These modules were specified to run with different periodicity (the arm models being fastest at a rate of 200Hz, followed by the three controllers at 100Hz and the path generator at 50Hz).

¿From this extended model, C code is generated using Matlab's automatic code generator. The execution of the model with the chosed parameters results in data for the particular run, and the students are asked to examine the log, ponder on irregularities, detect missed deadlines, etc. The lab-time saved not producing all the code is hopefully spent on reflecting on the techniques used. Deeper understanding is expected through "what-if" questions in interaction with the lab assistant for the course.

The main robot example is run twice: once in the non-RT Unix mode and again in the RT-mode, comparing and explaining the differences. To facilitate generation of alternative scenarios which involve significant CPU usage, the Matlab model is also instrumented with special "heavy computation" blocks with a parameterised absolute execution time.

6 Conclusions and future developments

The decision to rely on models, code generation and analysis tools for the exercise was taken in order to concentrate on study of real-time behaviour of processes. This requires a predictable run-time environment. Non-real-time tasks must not be allowed to interfere with real-time tasks. Although SUN's operating system is supposed to guarantee this, the actual behaviour observed suggests otherwise. Run-time results differ somewhat from machine to machine. There is also a big variance over time. A possible explanation is that there is interference from network traffic, device drivers or other system tasks. Directing all interrupts, except for timer interrupts, to a special dedicated task with low priority might be a way to overcome some delays in future versions of this excercise.

Due to above problems, the predictable run-time environment required for an exercise in real-time computing is not yet realised. This fact does of course limit the instructive value of the exercise and causes frustration among students and supervisors — but a side-effect is that it demonstrates the importance of predictability in real-time computing.

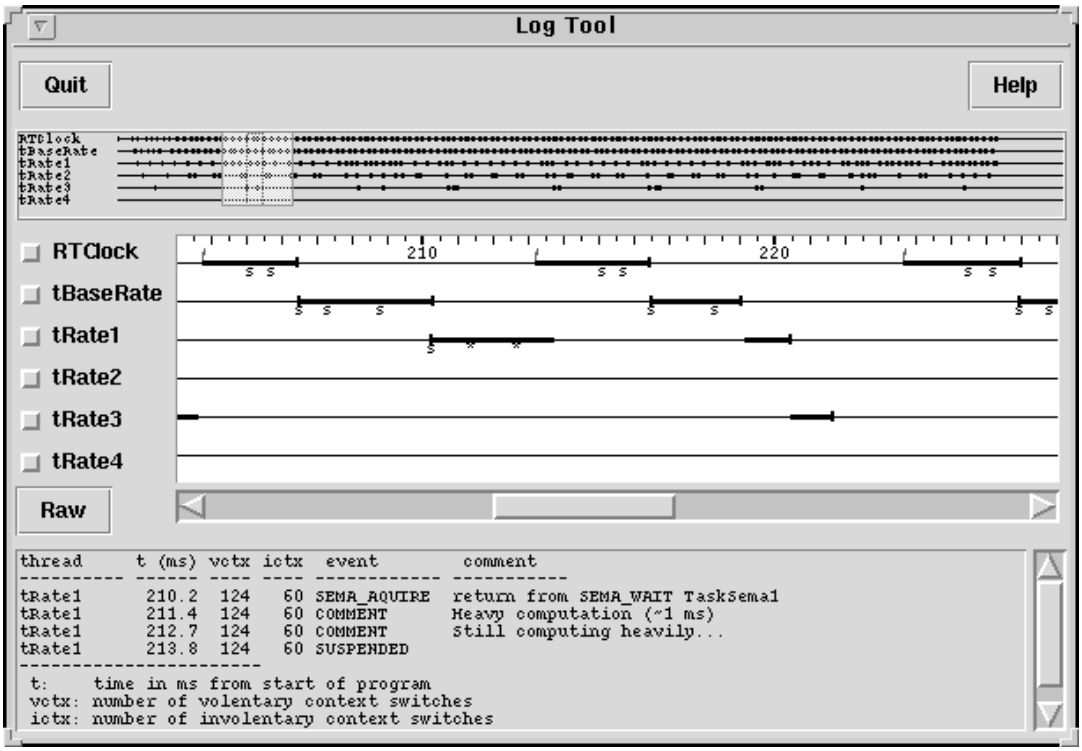

Figure 1: Window from a typical student session, examining the real-time behaviour of processes.

6.1 Results and evaluations

Despite the problems encountered, the theme as a whole is well regarded among students and feedback in the first year was expressed in terms like: "The lab is very instructive, but it is irritating with all the extra work created due to non-predictable system behaviour".

We believe the separation of programming and real-time aspects, allowing students to concentrate on real-time issues is one contributing factor why the course fares relatively well in evaluations. However, in later years we have noted a tendency among the students to consider the labs as "not so demanding". It seems that the seemless integration with the automatic control part, the lectures, and the group discussion of real-time cases provides insights early in the studies, so that the extra insight gained from the labs is getting less over time. However, as mentioned earlier, since the course has only been run three times our statistical data is still fairly limited.

As far as topic integration is concerned, one can say that it works very well (supported by student evaluations). This is one of the themes where theory and practice, and topics from two different institutions work well together. The written examination usually includes one large example reflecting both real-time and control theory aspects (e.g. distributed control in a car, autonomous helicopter, automatic drilling machine, CD player head drive). The pass rates and the numbers of top marks are higher than what we have for existing separate courses (despite the fact that the scheduling theory covered in this theme is more advanced, and the questions are sometimes more demanding).

We have also studied the pass rates for the labs compared to our course given for other programmes (considering all the involved labs here and the labs used in the traditional course). Compared to the traditional course (not PBL, no integration) we find a marked difference over three years. Between 88% and 100% of the students in this programme have passed their labs within a 3 month period after finishing the course (different %'s for different years). For the traditional course the comparative pass rate is 47% to 72%. We believe the students' conception of responsibilty, fostered by the PBL approach, and their motivation within this programme is a strong factor behind the measured result.

6.2 Generalisation to other settings

The aspects discussed in this paper can be of interest to other educators for several reasons. Our work, as well as the work in other parts of this programme, can serve to illustrate a successful attempt at incorporation of PBL into engineering educational programmes, where the focus is lifted from narrowing down into one area up to

integration of topics within an applicative setting. The price to pay is a little less room for each topic (specially since all themes here are obligatory and no electives). We have shown how to use the limited time and still not to lose the knowledge content. Reader interested in this broader area may also consult a comparative study of the student preferences and choices in two (content-wise) similar programmes, one with a traditional pedagogical approach, another with PBL [6].

Another interest may arise from the experience we gained in setting up a laboratory environment using standard equipment, standard operating system and normal educational network configuration – still finding a way to exhibit the interesting aspects of computation (with respect to real-time). This might inspire other educators to use standard settings in non-standard ways.

References

[1] Association for Computing Machinery. *Computing Curricula 2001, Strawman report.* `http://www.computer.org/education/cc2001`, March 2000.

[2] Web page for Swedish Council for the Renewal of Undergraduate Education,financed projects: `http://hgur.hsv.se/activities/projects /financed_projects/f-j/ingemarsson_ingemar.htm`, November 2000.

[3] L. Wilkerson and W. Gijselaers (Eds.). Bringing Problem-Based Learning to Higher Education. Theory and Practice. San Fransisco, Jossey Bass Publishers, 1996.

[4] Math Works Inc. *Real-time Workshop, version 2, User's Guide.* infomathworks.com, 1997.

[5] S. Nadjm-Tehrani. Towards Real-Time Systems Education with PBL. In *Proc. 2nd Int. Workshop on Real-Time Systems Education*, pages 39–48. IEEE Computer Society Press, 1997.

[6] S. Nadjm-Tehrani and L. Strömbäck. Study of educational preferences in computer engineering education. In *proceedings of second international conference on Problem-Based Learning in higher education*, page 32. Linköping University, September 2000. Extended abstract and more information currently appears on `http://www.ida.liu.se/~snt/teaching.html`, under NOT Project.

[7] The Information Technology programme. 6th Term information page (in swedish). Available from `http://www.control.isy.liu.se/student/ttit62/`, 1999-00.