Raising Motivation in Real-time Laboratories: The Soccer Scenario

Mehdi Amirijoo
Linköping University
Linköping, Sweden
meham@ida.liu.se

Aleksandra Tešanović
Linköping University
Linköping, Sweden
alete@ida.liu.se

Simin Nadjm-Tehrani
Linköping University
Linköping, Sweden
simin@ida.liu.se

ABSTRACT

Real-time systems is a topic that one cannot overlook in an engineer's education. However, teaching real-time systems in an undergraduate syllabus is a challenging experience due to conflicting constraints placed on such a course. In this paper we present a new setup for laboratories in the real-time systems course that successfully meets the constraints of mass education, stable environment management, short time span for the labs, and still enables deep involvement of students in the central topic of resource allocation with high motivation.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms

Algorithms, Languages, Performance

Keywords

Lab Environments, Pedagogy, Curriculum Issues, Real-time Systems

1. INTRODUCTION

Teaching real-time systems in an undergraduate syllabus is a challenging experience. On the one hand, given the diversity of the courses that a major in computer science or engineering has to cover, there is not enough room in the curricula with a long time interval (number of credit points) available for such a course[1]. On the other hand, over 90% of today's computing applications are in the area of embedded systems, and there is a massive growth in ubiquitous computing; many with real-time or low-power characteristics. This makes the issue of *resource allocation*, which can be considered as the heart of real-time systems, a topic that one cannot overlook in an engineer's education.

The area of real-time systems is wide enough for several courses, from theoretical analysis of scheduling algorithms to practical design and implementation of dedicated hardware, software, and middleware. This paper explains how we provide some insight in a central aspect of real-time system development within a very small time frame for a course. We also provide an indication of how considering the "fun factor" in the design of laboratories enhances the student's motivation and willingness to delve deeply in a lab assignment by choosing optional lab assignments in the limited time frame.

The paper further illustrates a major logistics issue in designing labs for this area. To gain hands-on experience in a topic like real-time systems one really needs to practice with a "real" application in which the interaction between application software, operating system, hardware, and the world around becomes concrete. However, with over 200 students taking our real-time systems courses we have had difficulties in providing environments in which interesting hand-on experience could be obtained. An added complexity is the computing environment that for most of our courses (taken by several thousands of students per year) are best administered in Unix platforms. There are some Windows machines available, but these have to run with certain restrictions and the software on these cannot be freely reconfigured or installed due to administration/security considerations. On the other hand, ubiquity of real-time computing has resulted in that many hardware interfaces, device drives etc. are Windows/PC based and languages like C provide the dominant programming environments due to early performance profiles. We describe how we have overcome the seemingly unsolvable equation that results from the following desired criteria or imposed constraints (abbreviated with C1-C4):

C1 understanding problems in a real-world setting,

C2 getting insight in a generic (not application-specific) topic,

C3 little time allocated for the lab assignments, hence no low-level programming,

C4 stable and uniform computing environment for large classes.

To the best of our knowledge, previously presented work on lab environments, e.g., [9, 10, 11], have only considered a subset of our constraints, as they have either addressed

[1] Witnessed by the fact that the topic is not included in the core part of ACM 2001 curriculum [1] for computer science.

SIGCSE'04, March 3–7, 2004, Norfolk, Virginia, USA.

courses with low student participation or low-level programming.

2. SOCCER PLAYING ROBOTS

2.1 Soccer game: a real-time scheduling problem

Typically, real-time and embedded systems can be found in control loops tightly coupling sensors with actuators. The task of a real-time application can be decomposed into the following three steps: (i) the information given by the sensors is read and processed, (ii) various computations are performed, and (iii) the result of the computations are outputed by the actuators. Step two often involves optional execution of non-critical parts, which may skipped if there is not enough time or lack of resources. The modularity of the sensor-computation-actuator loop enables the application designer to divide the application into three tasks corresponding to the steps described above. Correct scheduling of these steps (including allocating enough resources to step two) and meeting associated deadlines is very important as violating timing requirements may result in a catastrophe.

The labs we chose were inspired by the Robot world Cup (RoboCup) soccer games, where a small-size soccer game involves two teams of robots playing soccer on a green carpeted field (see the F180 league [7]). A robot moves around on the field using the power from two on-board motors. The software controlling the RoboCup soccer game is distributed between computers controlling the robots, and the robots that are being controlled. Software located on the robots consists of a low-level controller, which processes instructions sent to it via wireless communication. The processing capability of the low-level controller is limited to packet processing and motor control.

The software located on the computers consists of three major parts, namely an image processing unit, a high-level controller for each robot in a team, and an actuator. The image processing unit processes images that are fed into the computer from an overhead camera attached above the soccer field. The high-level controller consists of a planner and a reactor. For each robot a planner analyzes the output from image processing and makes various computations, including game analysis, strategy acquisition, multi-robot collaboration, and plan generation. The reactor of each robot takes a plan given by the planner and decides on concrete low-level actions, e.g., motor speeds, that should be taken to achieve the planned goal. Finally, an actuator, which is common for all robots, wraps up the decided actions into a protocol and transmits them to the robots.

Comparing the processes involved in the RoboCup soccer game with the description of real-time systems, given earlier in this section, we see that there are many similarities. RoboCup processes involve sensing, reacting, and carrying out actions, resulting in the RoboCup tasks exhibiting typical control loop characteristics. Hence, our choice of lab scenario is generic in the sense that many real-time scheduling problems can be analyzed and at the same time, it provides a setting that corresponds to many real-world scenarios. This satisfies constraints C1 and C2 (see Section 1). Finally, the problems are also fun to work with, which in turn enhances the motivation of the students.

2.2 Choice of platform and programming language

There were several problems to solve when choosing hardware and software platforms. Initially, we wanted to use Real-time Linux as operating system and ADA [8] as programming language. However, we found that, at the time of the development, there were really not many device drivers written for Real-time Linux and we were forced to focus our attention on the Windows operating system. This was also the only other computing environment available in our teaching domain besides Unix.

As the focus of the labs were to teach the students how to implement scheduling algorithms in a real environment, our goal was to minimize the time and efforts invested by the students to learn a new programming language. Our experience with students learning new programming languages just for the lab exercises was that too much time was spent on actually trying to get the syntax right, instead of solving the actual problem. After a careful investigation we found out that students are mostly familiar with Java, which is also increasing in popularity in the industry. ADA on the other hand is rarely used by the students and appears in fewer industrial sectors. Also, we wanted to avoid the students dealing with low-level programming details. This is also where Java comes in nicely, since it supports high-level programming constructs. Thus we have satisfied constraint C3.

We saw the emergence of Real-time Java (RTJ) [8] as a promising direction in real-time systems application development. As the life cycle of laboratory exercises typically spans over years, we chose RTJ as the programming language in order to keep up-to-date with the forthcoming trends. However, at the time of the lab development,the specification of RTJ was only recently released and it would take several years until a stable platform would be available. For this reason, we built an application programming interface (API), on top of Java, mimicking a subset of the RTJ programming interface. Although, the API is not complete with regard to the RTJ specification, it allows the students to become familiar with the most important constructs dealing with scheduling of tasks in RTJ.

2.3 Replacing the real world with a simulator

In Section 1 it was mentioned that the number of participants attending real-time courses typically exceeds 200 students. It is clearly unmanageable to coordinate and supervise 200 students implementing scheduling algorithms within one physical lab setup. Motors break down, batteries discharge, and all the practical hassle of getting the setup continuously functioning takes the resources away from the pedagogical efforts. We have for this reason developed a simulator, which can replace the real-world, i.e., the simulator can replace the soccer field, robots, and the ball (see Figure 1).[2] The simulator runs on the same machine as the teams. Visual information is sent to the two teams, and the result of the planner and the reactor is relayed via the actuators to the simulator. A graphical interface, called the monitor, portrays the soccer field, the robots, and the ball. This enables the students to follow their team during the progress of the game. The developed physical environment

[2]The students do not need to modify their scheduling algorithms when using the simulator.

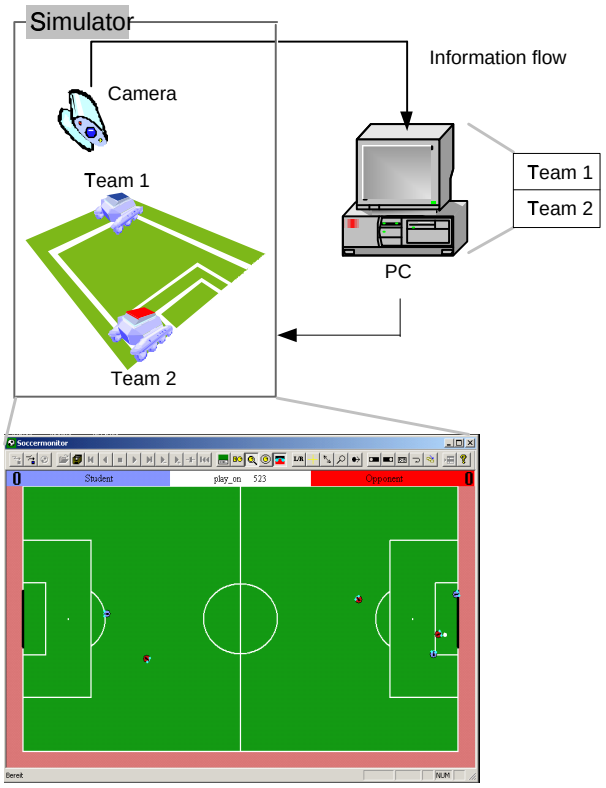

Figure 1: Simulated scenario of the RoboCup lab environment

is available as a reality reference, and acts both as a demonstration setup, and as a potential ground for rewarding high scoring teams.

By using the simulator the students can develop and simulate their schedulers in parallel. They can test and evaluate the performance of the schedulers using a variety of methods: (i) competing against a default team, where a set of schedulers are implemented by the teachers, (ii) competing against a team with schedulers implemented by themselves, and (iii) competing against other students. We have included the option (iii) to enhance the fun factor and the motivation of the students.

By comparing the performance of the schedulers, the students are able to draw some conclusions on the advantages and the disadvantages of the different scheduling algorithms. Once the students have finished implementing and testing their schedulers using the simulator, they are allowed to compete against another student team using the real-world setup. The choice of Java as a programming language allows the same resource allocation strategy to be implemented both in the simulated (Unix-based) environment and the physical (Windows-based) environment. Hence, constraint C4 (see Section 1) is satisfied.

2.4 Evaluating the quality of the soccer game

It is obvious that incorrect scheduling of the involved tasks (image processing, planner, reactor, and actuator) may result in bad robot performance. Efficient implementation of a scheduler requires a complete understanding of the function of the different tasks involved. For example, if the planners are not given enough processing time, then the robots will not be able to play sophisticated games. If the reactors are not scheduled often enough, then the result will be low reactiveness, yielding in for example collisions. The list can be made long and the bottom line is that by studying the game, we can obtain some information on where the actual errors occur. Hence, testing is done by playing and observing the game.

3. STUDENT PROGRAMMING ENVIRONMENT AND ASSIGNMENTS

3.1 The application programming interface

As mentioned in Section 2.2, we have built an API on top of Java mimicking a subset of the RTJ programming interface. Our API includes classes for handling time, task management, and sorted lists. Students are required to implement a class, called Scheduler, using the following API:

Time handling The time handling interface provides a set of classes dealing with absolute and relative time, and these consist of methods for accessing, comparing, adding, and subtracting time.

Task management The task management interface provides an interface towards the tasks. The interface allows the current status (e.g. whether the task is finished or not), periods, and deadlines of tasks to be accessed and modified using the classes ReleaseParameters and PeriodicParameters [8]. The task management interface also allows the scheduler to dispatch tasks.

Sorted lists Although sorted lists are not included in the RTJ specification, these were added to allow the students to put their efforts on the actual scheduling and not the implementation of data structure, which they have learned in earlier courses.

3.2 Coding a series of schedulers

The students are required to implement a cyclic scheduler, referred to as part (a), earliest deadline first (EDF), referred to as part (b), and as an optional part they can also implement rate monotonic scheduling (RM), referred to as part (c). For detailed description of the scheduling algorithms, we refer to [8].

The scheduling algorithms must take into consideration the precedence constraints of the tasks, i.e., the order in which the tasks must be executed. Image processing must precede the reactors, and the actuator must be executed after the reactor. The planners are less dependent on the data given by the image processing, since the planners produce long terms plans and the current status of the game does not change considerably between each sampling point. Hence, the planners can be executed whenever there is extra time over. By allocating more CPU resources to the planners, better plans are devised, but at the cost of a longer execution time and possible deadline misses of the succeeding tasks. Hence, it is vital to schedule the tasks such that they meet their deadlines and allocate as much CPU resources to the planners as possible. This is something the students need to consider as it brings forward the advantages and

the disadvantages of the scheduling algorithms, which we describe below:

Cyclic scheduler is easy to implement and causes little or negligible overhead. The schedule or the ordering of the tasks is determined off-line (before running the system), and this technique is feasible in systems where exact worst-case execution times (WCET) are available. However, in our application, the worst-case execution time of the planners is very high and allocating time corresponding to the WCET of the planners, may yield in a highly inefficient soccer game. Giving the planners less (constant) time, may yield in system underutilisation if the execution time of the other tasks is much lower than their WCETs. Hence, using the cyclic approach an optimal resource allocation cannot be done, resulting in poor planning.

EDF is a dynamic-priority scheduling algorithm, where the ordering of the tasks is done on-line according to the deadlines of the tasks. EDF is more flexible in the sense that the planners are given execution time whenever resources are available (since they have longest relative deadline), still conforming to time constraints. However, since EDF schedules the tasks on-line the overhead caused by the scheduler is greater, as compared to the cyclic scheduler.

RM is a fixed-priority scheduling algorithm, where the ordering of the tasks is done on-line according to the periods of the tasks. The planners are given time whenever resources are available, however, RM is less efficient than EDF [8].

In our lab environment, the implementations of EDF and RM need to (i) dispatch a task, (ii) check for deadline miss and call a deadline miss handler once a deadline miss has been observed, and (iii) suspend a task whenever it is finished and needs to sleep until the next period.

4. EVALUATION OF THE LABS

Development of the physical environment was essentially the effort of a team consisting one professor, one assistant professor, one senior PhD student, and six student assistants. The student assistants invested roughly two months each on the development (over an interval of two calendar years).

The labs were run for the first time in the "Integrated Theme on Real-time Process Control", a 4 credit point course that we run in cooperation with the automatic control department, and in which we follow the pedagogical ideas of problem-based learning (PBL) [2, 3]. One credit point is allocated to the real-time lab exercises. The atmosphere in these classes makes them more amenable to testing new ideas in course development. The student are familiar and keen in the idea of providing regular and constructive feedback and give us the opportunity of testing a new idea in a small scale (around 30 students) before we move over to the larger courses with over 100 students.

Despite some small initial problems in the lab's programming environment we consider the experience as very successful. The evaluation is supported by three activities: (i) evaluation using muddy cards, (ii) study of the rate of punctual completion of labs, and (iii) study of willingness to do optional assignments.

4.1 Muddy card evaluations

Muddy cards (MC) is a technique for mid-term evaluation of a course [6]. An evaluation consists of a data collection phase where students are asked for comments, a summary phase that involves clustering the comments and writing a report that describes the result, and a conclusion phase where the evaluator comments on the result. The data collection phase consists of handing out small white cards on which the students can write both positive and negative remarks about the learning environment a course provides. While this is regular practice in evaluating the state of a course for us, we increased the number of evaluation sessions during the term that the new labs were introduced. Typically, a muddy card evaluation takes part during one lecture somewhere before the middle of the course. Here, we did two additional evaluations among the first lab sessions "to feel the pulse" of the laboratory work environment specifically.

The following citations are directly taken from the student-written cards in these lab sessions or the main course MC evaluation:

> *The labs are interesting and the way to try the strength of the scheduler by letting the team play against another team is great!*
>
> *These labs are fun! Its nice to have a high abstraction level so that we can concentrate on scheduling.*
>
> *The labs are fun but you can see it is the first year for them since they seem a bit unpolished. Many of the functions seem to be like C++ functions and the result variable as a parameter and returning void, for instance.*

The third comment actually indicates a willingness of the students to criticise and provide feedback on the actual lab skeleton within which they worked; a factor that has left a specially positive impression on us. There were of course a few negative comments about the lab experience too. These were caused by not having the detailed bat files that relieved the students from having to load parts of the environment into directories with right names, etc. While we could see the negative impact of these initial problems we are confident that they are minor issues that can be remedied easily in the next run of the lab course.

4.2 Completion rates

All in all 32 students took part in the lab sessions for this course. After the deadline for delivering the lab results all 32 students had finished the compulsory part (a) and (b) of the lab, i.e., the cyclic executive and the EDF scheduler. This rate of completion (100%) can be compared to the rate of completion of the labs before the deadline for our other course in which students also learn scheduling algorithms by writing a scheduler code without any other context. Surprisingly enough, the rate of completion of these labs before the deadline was 78%, which is lower than what we experienced in the new lab course, despite the fact that the students "suffered" from some initial problems due to setting up the lab programming environment.

4.3 Optional Assignments

Another interesting metric is the number of assignments attempted by each student. In our estimation of the time needed for the labs, we considered part (a) and (b) (the cyclic scheduling and EDF lab) as the two that most likely would take the time available for the labs. We knew, however, that some ambitious students might be willing to spend more time on the labs in exchange of some "given" points in the written exam. It turned out that 22 students in fact completed the optional assignment, i.e., part (c), which amounts to 69% of the students.

5. CONCLUSIONS

We have explained a way of developing a new setup for real-time systems lab course, given the constraints of mass education, stable environment management, short time span for the labs, and still deep involvement in a central topic with high motivation.

Our detailed description of the way the environment was developed can serve as a guideline for those wanting to repeat the experience and we would be willing to share more detailed documentation and code.

The evaluation of the lab series in the first attempt (spring term in year 2003) was deemed to be successful, both in terms of student evaluations, punctual completion rate, and the exhibited interest for doing extra labs. We are confident that the latter can be associated with the increased level of motivation as a result of the "fun factor", as well as a well-thought documentation and preparation time. The completion rates are more difficult to relate to the added value of these lab series. It may simply be the case that this group of students are more willing to take own responsibility (as also observed in an earlier study [5]). However, the new statistics at least confirm that their desire to achieve on-time completion was not hampered by the new lab setup.

6. ACKNOWLEDGMENT

The authors wish to thank Dr. Jörgen Hansson and the other members of the Real-time Systems Laboratory at Linköping University for their help in the development of the lab environment. Also we would like to acknowledge Paul Scerri, Andreas Böckert, Anders Petersson, Srikanth Komarina, and Johan Eilert for their involvement in developing the lab environment.

7. REFERENCES

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

[2] Web page for Swedish Council for the Renewal of Undergraduate Education, financed projects: `http://www.hgur.se/activities/projects/financed_projects/f-j/ingemarsson_ingemar.htm`, 1996.

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

[4] 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.

[5] E. Herzog and P. Loborg and S. Nadjm-Tehrani. Real-time Systems Lab Exercises: A Teacher's Dilemma. In *Proc. 32nd Int. conf. on Computer Science Education (SIGCSE'01)*, pages 273–277. ACM, 2001.

[6] C. Kessler and S. Nadjm-Tehrani. Mid-term Course Evaluations with Muddy Cards. In *Poster session in the 7th Int. conf. on Innovation and Technology in Computer Science Education (ITiCSE'02)*, ACM, 2002.

[7] http://www.robocup.org/

[8] A. Burns and A. Wellings. Real-Time Systems and Programming Languages Addison Wesley, 2001, third edition

[9] R.A. Maulucci and J. Lentz and R.H. Eckhouse Real-time Devices at Practical Prices: Low Cost Laboratory Projects Proceedings of Real-Time Systems Education III, 1998.

[10] B. L. Kurtz and J. J. Pfeiffer A Course Project to Design and Implement the Kernel of a Real-time Operating System Proceedings of the SIGCSE Technical Symposium on Computer Science Education, 1987

[11] A Low Cost Laboratory for Teaching Embedded Real-time Systems In Proceedings of the 27th IFAC/IFIP/IEEE Workshop on Real-time Programming (WRTP), 2003.