Student Projects in Reactive and Real-Time Systems Course

Ran Lotenberg
Computer Science Department
Tel-Aviv University
Tel-Aviv 69978 Israel
ranl@math.tau.ac.il

Shmuel Tyszberowicz
Computer Science Department
Tel-Aviv University
Tel-Aviv 69978 Israel
tyshbe@math.tau.ac.il
and
Academic College of Tel-Aviv-Yaffo
Tel-Aviv, Israel

Abstract

This article presents the student assignments and the experience gained in a graduate course entitled "Reactive and Real-Time Systems", taught at Tel-Aviv University.

The course focuses on the development of provably correct reactive and real-time systems. Hence, the major projects that were given included the full life-cycle of a system development: specification, design, implementation, and verification. The software tools employed in the course are freely available from various Internet sites. Students had the freedom to choose the tools they use, and indeed many of them chose more than one tool to supply a system that was automatically verified against its requirements.

1 Introduction

The course "Reactive and Real-Time Systems" offered at Tel-Aviv University focuses on aspects of specification and development of provably correct reactive real-time systems. Reactive systems, and particularly real-time computing, have become an important discipline of Computer Science. In [14], the syllabus of the course was described. The major topics that were taught in the Spring 1998 semester were:

- `Introduction to reactive and RT systems`
- `Requirements specification (informal and formal), system analysis and design`
- `The synchronous approach for modeling reactive systems`
- `Model checking`
- `Student talks`

The next section shortly describes the assignments given at the end of each topic and their rationale. Section 3 elaborates on the major projects given in the course. Finally, Section 4 concludes the paper.

2 Course Topics and Assignments

This section elaborates on the topics taught and describes the nature of homework done by the students during the semester. Whenever possible, we add our conclusions from this work. A detailed description of the syllabus and its rationale appears in [14]. We will, however, shortly describe each topic.

2.1 Introduction

We naturally started by talking about what reactive and real-time systems are and their characterization. The major part of the introduction included the definition of the computational model of the system (i.e. the set of behaviors that are associated with the system): Fair transition systems [16] as a model for reactive systems and clock transition systems [11, 13, 1] as a model for real-time systems. The knowledge gained in this part was checked by means of theoretical questions only (e.g., from [16]).

2.2 Requirement Specification, Analysis and Design

We studied the Real-time Structured Analysis and Design (RTSAD) method [9] for the analysis and design of real-time and concurrent systems. The examples given in class included a Cruise Control System and a VCR device. In order to describe the dynamic behavior of the system under development we used Statecharts. At this point, we still hadn't covered in depth the semantics of Statecharts (e.g. is it using strong or week preemption, etc). However, after discussing some (even) very small examples, the importance of the semantics became very clear.

In order to ease the work, the students used a tool that implements a subset of Statecharts. The tool, called gE—Graphical Editor, was developed in the GMD [5]. The ultimate intention was to use the graphic specification as an input to a verification tool, as will be discussed later. At this stage, it had the advantage of enabling the inclusion of priorities on the transitions, hence mixing strong and weak preemption together, and making clear and deterministic the way the system had to act.

At this point the students received their second assignment. The question was whether to give one large project or a number of smaller ones. The latter choice was taken. The reasons were:

- The students were already experienced with the analysis of large information systems (either from their studies or from their work).
- The projects were just a means for the next phase of the study rather than the major issue.
- Based on previous experience—choosing the appropriate small or medium-sized projects exposes the students to most of the problems they have in large projects.

This semester three projects were given (to be chosen out of four), to be undertaken in teams of three students each (hoping that each project was done by all three, rather than a project per student...). The assignments were:

1. A one lane bridge (junction) that connects two sides of a two lane highway,
2. An answering machine,
3. An elevator system,
4. A juice bottle filling machine.

After the analysis phase, the students were asked to write, in a natural language, properties that the system had to fulfill. A class discussion soon revealed that many of those requirements were ambiguous and unclear, and difficult to follow in the students' analysis. The conclusion was that a more formal way is needed. Now the students were asked to write specifications in Temporal Logic, Real-Time Logic (RTL) [12, 18], RTTL [19] and MASS [7]. The intention was to use those requirements for formal proof of some (very) small examples. Nevertheless, mainly due to lack of enough knowledge in logic, they didn't succeed in proving properties (see section 2.4).

At this stage, however, it was already clear to the students that testing as the only method of program verification is not possible in large, complex systems. Due to the nature of real-time systems, they should be specified using a formal language. The properties that they wanted to prove, as written in Temporal Logic, were classified into safety and liveness properties.

2.3 Implementation: The Synchronous Model

Synchronous languages have been designed to ease the programming of reactive real-time systems. They also serve for the specification of the control part of a real-time system.

After learning what the synchronous model is, and what its advantages are, the syntax and semantics of `Esterel` were studied (as well as completing the study of the formal semantics of `Statecharts`).

We have emphasized the compilation scheme of `Esterel`, showing how it suits the computational models studied in the introduction part, and how we can take advantage of it for verification. We have used two `Esterel` compilers:

- Version V5_10 supplied by Inria [2]. The compiler is used for verification with XEVE (`Esterel` Verification Environment [3]). XEVE consists of a set of UNIX processes implementing verification functions and a graphical interface to invoke them. In order to specify properties to be verified we used the TempEst package. TempEst [15] is a tool set for the formal verification of safety properties (expressed in a variant of temporal logic) of `Esterel` programs.
- A compiler that was developed at GMD in St. Augustin, Germany, for compiling pure `Esterel` programs. This compiler produces code in the SMV (Symbolic Model Verifier) [17] language.

At this point the students implemented the three projects, in `Esterel`, as specified in RTSAD.

A reference to another language sE—Synchronous Eifel [4], developed in the GMD, was given to the students. This language enables specification of the control part using a dialect of Esterel or Statecharts specification, created with the gE tool. Data parts are written in a dialect of Eiffel. The program can easily be translated into the input format of the model checkers SMV and VIS (translations to other formats were beyond the scope of the course). Only one team, which included students who had experience with Eiffel, chose to implement one of the projects in sE.

2.4 Verification

During the course, we studied and used symbolic model checkers in order to build correct reactive and real-time systems. We also considered use of the *deductive approach* (by employing TLV [20], for example) to build a correct system from its specification. However, from the experience we had with some graduate students working on their Master Theses, the students do not have enough background in Logic, so that too much additional study would have been needed.

2.5 Student talks

The last part of the course was given by the students in the form of a seminar framework.

2.5.1 Scheduling

Based on articles and on chapters from [6], the students classified the scheduling problems, and described approaches taken in order to fulfill the timing constraints in real-time systems. Later the students had to check the feasibility of several tasks, based on Rate Monotonic Algorithm, Earliest Deadline first, Priority Inheritance, Priority Ceiling Protocol, and various servers to handle sporadic tasks.

The scheduling part of the course was covered by several non-computer assignments. The students were asked to check whether a given set of tasks is schedulable using the scheduling algorithms studied.

2.5.2 Some Other Synchronous Languages

The dataflow synchronous languages Lustre and Signal [10] were introduced. The students were free to use these languages as well.

Since we do not have a translator from Lustre and Signal to the input language of any model checker, another verification scheme was used. In order to verify properties, a module that acts as an acceptor of the program computation should be written. For any instant, if the desired property holds, the acceptor module emits a special signal (say *OK*). The module checker checks whether *OK* is constantly emitted.

3 The Projects

In this section we will describe the major projects given in class. As mentioned, the students had to choose three out of the following systems:

1. A one lane bridge (junction) that connects two sides of a two lane highway,
2. An answering machine,
3. An elevator system,
4. A juice bottle filling machine.

For each, a general informal description was given, and the students had to deliver a verified working system.

In order to gradually increase work complexity, it was advised to do the assignments in their given order, i.e. complete the development of a simple project, and proceed with more complex projects.

In the rest of this section, we will elaborate on how one team tackled the assignments. For each assignment we supply the informal specification requirements (or provide a reference to them); shortly describe the design; and elaborate on the conclusions. Throughout this description the reader may observe the knowledge acquisition process and the problems encountered during the development process.

The team chose to implement the first three projects. The design of the first two is briefly described, whereas the elevator system, which is the most complicated, is presented in detail. In all projects, RTSAD was used for analysis and design, Esterel for programming, and TempSet for verification (SMV and VIS were used as well in the junction project).

3.1 The One Lane Junction Project

Informally, the requirements of this project were as follows. Traffic is controlled by two stoplights located at the far ends of the junction. At each given time cars can cross the junction only in one direction. The green light of a lane remains on as long as there are no cars on the other side, but at least for one minute if there are cars at that lane. The light on the other side turns to green only after the junction is cleared.

For this specific task one controller for the two stoplights suffices. The team decided to supply a general solution. A "generic" stoplight module, which loops over the stoplight states, was designed. The junction executes two such modules in parallel with different phases.

The project was verified with SMV, but not with its original code: since the `Esterel` to SMV compiler does not support `Esterel` modules, the team had to flatten the code (a cut & paste and signal renaming process).

This assignment is a simple task, suitable to introduce a new environment. In this project, synchronous model understanding, `Esterel` programming, and tools usage were experienced. With this experience in mind, the team could avoid many pitfalls while working on the other two projects. In addition, at each phase of those projects, the team was able to make decisions suitable for subsequent phases.

3.2 An Answering Machine Project

This project dealt with the development of an answering machine controller. A full informal specification of the answering machine, as given to the students, can be found in [8].

A dispatcher, which calls other modules—one module per operation mode (answer incoming call, play recorded messages, and record intro message), was designed.

The specification of the controller provided experience in the use of Statecharts to specify the dynamic controllers behavior. Some of the syntactic features of higraphs (Statecharts) helped to supply a compact control flow diagram (CFD).

3.3 An Elevator Project

The third project was an elevator controller which is a complex system, and hence requires much more effort. It also manifests (what we consider to be) a "classical" problem, which we discuss below.

The control (distributed vs. centralized) and service-algorithm (which floor to move to) were the main design decisions. The team decided to distribute the control—once the elevator is detected at a certain floor, the floor controller decides what action to take (either to stop the elevator and service the passengers or to send it up/down according to given criteria). This design supports "low-cost" composition of an arbitrary number of floors. The service-algorithm used for the elevator movement keeps the lift going in the same direction until there are no floors to serve in that direction; then, if required, it changes the lift's direction.

A generic floor module, which incorporates all the logic, was programmed. The controller is composed of interconnected floor modules running in parallel. In order to shorten development time a module simulating the environment was programmed.

In the design, each floor is connected to adjacent floors and to the environment, a composition that generated many causality problems[1]. Solving these problems made the synchronous model, as implemented by `Esterel`, clearer to the team.

The elevator system could not be verified using TempEst due to lack of disk resources. The automaton file (.oc file) had more than 140M bytes when the verification process was aborted just before it crashed the department system. Some teams were unable to verify their projects using SMV, as their verification process was automatically swapped out by the operating system after 12 hours of execution.[2]

In order to verify the project, the code was simplified and the environment simulation was removed. A different problem was then encountered. Normally, the verification program does not restrict input signals, i.e., does not make any assumptions on the occurrence of those signals. In our case, it led to an exceptional "real-world environment" behavior. Consider the following example: an input signal indicates the elevator's current position: *In_floor_1*, *In_floor_2*, and *In_floor_3*, for floors 1, 2, and 3 respectively. While in real-world scenarios *In_floor_3* can not occur immediately after *In_floor_1*, in arbitrary scenarios it can. There are no limitations on the model-checker that prevent it from checking such a scenario. Furthermore, *In_floor_2* can be emitted after *In_floor_1* only when *Go_up* signal is emitted in between by the elevator controller. The fabricated unrealistic scenarios made the verification pointless. In simple projects the user can verify manually each scenario that fails for "real-word correctness", but in this case it was not attainable. Hence, the team had to reuse the environment simulation and to write new verification constraints to verify the correctness of the simulation.

This seems to be a "classical" problem and a serious flaw in the verification process. In order to verify real-world projects, specific environment behavior must be programmed into the project. Supplying the simulated environment might or might not exhibit all the potential scenarios. Further, by supplying an environment simulation, assumptions about the environment are brought into the system and hence into the veri-

[1]For example: floor n reacts to signals generated by floor n-1, and floor n-1 reacts to signals generated by floor n. It is further obvious that the environment reacts to signals generated by the controller and vice versa.

[2]The students used SunOS 4.1.4 Axil 320, with two 90 MHz hyperSPARC CPUs. The normal load average on the students computer as shown by the rup command is 5 to 7.

fication process. Also, while verifying some of the requirements, the team noticed that some "trivial" properties did not hold. At this point it became clear that hidden assumptions have to be made explicit in order to verify the correctness of the design with respect to the requirements. This means that one has to prove $Assumptions \wedge Design \Rightarrow Requirements$, rather than $Design \Rightarrow Requirements$. Some "problems" vanished by just adding fairness conditions.

4 Conclusion

The topics studied during the course were divided into two categories: theory and practical experience. The projects haven't covered all the theoretical subjects (e.g. the scheduling topic); however, they sometimes made clear aspects that seemed to be unrelated. The computational model, for example, was first seen by the students as a purely theoretical issue. While working on the practical projects, especially when combining different `Esterel` compilers to various tools (SMV, XEVE, VIS), the importance and usage of this model became quite clear. Hence, the benefit from the projects was not limited as how to develop a real-time system.

There are various software tools used for the development of RTSs. In this course, many tools are introduced and employed, unlike other courses where one, or at most two, new tools are used.

The practical experience tied all parts together giving the "look and feel" of RTS development. It emphasized the capabilities and limitations of the models and tools used. The verification process provided another reason justifying our decision to use small to medium sized projects. Though the projects were not large, the students encountered many problems during verification due to lack of computer resources. The verification process requires intense resources, which were not available. We admit that some of the problems were due to poor design. However, though correcting the design was also part of the study process, it was not always enough in order to solve problems with the verification.

In this paper we discussed how the assignments were undertaken by one team. Whereas some teams are still working out their assignments, other projects are currently evaluated by the TA of the course. Comparing the efforts expended by the teams to each project and within the projects to each phase, reveals large differences in the time spent. This can be explained by different backgrounds of the students, as it seems that their skills are similar.

The verification tools were primarily used in the projects with qualitative temporal reasoning about systems. Using abstractions, we were able to verify, with XEVE, some quantitative properties (like responsiveness within a given number of time units, etc). SMV supplies algorithms that compute the minimum and the maximum number of occurrences of a condition on any path between two given events. We are trying now to see how to use them to verify temporal properties.

Two of the student talks described the dataflow synchronous languages Lustre and Signal. Both languages are available at our site and students were encouraged (by a special bonus) to use them; however, no team chose those languages. We still cannot definitely explain the reasons for it. Is it the different programming style (not the imperative one they are used to work with), or the fact that less time was devoted in class to those languages (compared to Esterel), or other unknown reasons?

References

[1] R. Alur and D. Dill. The theory of timed automata. In *Real-Time: Theory in Practice*, LNCS Vol. 600, pages 45–73. Springer-Verlag, Berlin, June 1991.

[2] G. Berry. The Esterel v5 Language Primer. http://www.inria.fr/meije/esterel/, 1998.

[3] A. Bouali. Xeve: An Esterel verification environment. http://cma.cma.fr/Verification/Xeve/, july 1996.

[4] R. Budde. The design and programming language synchronousEifel (sE-version 1.1), 1998. GMD, St. Augustin, Germany.

[5] R. Budde. The Graphic-Editor for synchronousEifel (sE-version 1.1), 1998. GMD, St. Augustin, Germany.

[6] G. C. Buttazzo. *Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers, 1997.

[7] V. Gafni. Real-time activation oriented specification language. Technical report, Computer Science Department, Tel-Aviv University, 1996.

[8] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and design of embedded systems.* Prentice-Hall, 1994.

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

[10] N. Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer Academic Publishers, 1993.

[11] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *Real-Time: Theory in Practice*, LNCS Vol. 600, pages 226–251. Springer-Verlag, Berlin, June 1991.

[12] F. Jahanian and A. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12(9):890–904, September 1986.

[13] Y. Kesten, Z. Manna, and A. Pnueli. Clocked transition systems. Stanford technical report, Department of Computer Science, 1995.

[14] G. Koren and S. Tyszberowicz. Graduate course: Reactive and real-time systems. In D. Mossé and J. Zalewski, editors, *Proc. 2nd IEEE Real-Time Systems Education Workshop*, pages 96–103, Montreal, Canada, 1997.

[15] C. Puchol L.J. Jagadeesan and J. Von Olnhausen. Safety property verification of Esterel programs and applications to telecommunications software. In *Proceedings the Seventh Conference on Computer-Aided Verification*, pages 290–301, Belgium, July 1995.

[16] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

[17] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[18] A. Mok. Towards mechanization of real-time system design. In A. M.van Tilborg and G. M. Koob, editors, *Foundations of Real-time Computing: Formal specification and methods*, pages 1–37. McGraw-Hill, 1991.

[19] J.S. Ostroff. Formal methods for the specification and design of real-time safety critical system. *Journal of Systems and Software*, 18(1):33–60, April 1992.

[20] A. Pnueli and E. Shahar. Combining deductive with algorithmic verification. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, pages 184–195. Springer-Verlag, 1996.