Real-Time Systems, 25, 231–253, 2003
© 2003 Kluwer Academic Publishers. Manufactured in The Netherlands.

Reactive and Real-Time Systems Course: How to Get the Most Out of it

TAL LEV-AMI tla@math.tau.ac.il
School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel

SHMUEL S. TYSZBEROWICZ tyshbe@math.tau.ac.il
School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel

Abstract. The paper describes the syllabus and the students' projects from a graduate course on the subject of ''Reactive and Real-Time Systems'', taught at Tel-Aviv University and at the Open University of Israel.

The course focuses on the development of provably correct reactive real-time systems. The course combines theoretical issues with practical implementation experience, trying to make things as tangible as possible. Hence, the mathematical and logical frameworks introduced are followed by presentation of relevant software tools and the students' projects are implemented using these tools.

The course is planned so that no special purpose hardware is needed and so that all software tools used are freely available from various Internet sites and can be installed quite easily. This makes our course attractive to institutions and instructors for which purchasing and maintaining a special lab is not feasible due to budget, space, or time limitations (as in our case).

In the paper we elaborate on the rationale behind the syllabus and the selection of the students' projects, presenting an almost complete description of a sample design of one team's project.

Keywords: reactive real-time systems, course syllabus, synchronous languages, verification, scheduling

1. Introduction

Reactive and real-time systems involve concurrency, have strict time requirements, must be reliable, and involve software and hardware components (Halbwachs, 1993).

Reactive systems, and particularly real-time computing, have become an important discipline of Computer Science. Though most would agree the subject matter is interesting and important, little consensus exists on what exactly the syllabus should include.

1.1. Reactive systems

Reactive systems are computer systems that continuously react to their physical environment, at a speed determined by the environment. This class of systems has been introduced to distinguish them from transformational systems (input, process, output).

Reactive systems include, among others, telephones, communication networks, computer operating systems, man-machine interfaces, etc.

1.2. Real-Time Systems

Real-time systems (RTSs) have reactive behavior. An RTS involves control of one or more physical devices with essential timing requirements. The correctness of an RTS depends both on the time in which computations are performed as well as the logical correctness of the results. Many day-to-day gadgets embed relatively simple real-time systems; other RTSs are among the most complex systems being built today. Industrial process control systems, transportation control and supervision systems, signal processing systems, are examples of reactive system that also have strict timing constraints.

The requirements from an RTS are diverse, ranging from intricacies of interfaces to providing guarantees of safety and reliability of operation. Severe consequences may result if the requirements of a system are not satisfied.

Many RTSs are special purpose systems, require a high degree of fault tolerance, and are embedded in larger systems. Typically, an RTS consists of a controller part and a controlled part. Examples of such systems are command and control systems, flight control systems, avionic systems, communication systems, robotics, process control systems, and more.

1.3. Course Structure

The general structure of the course includes an introduction and a broad overview of reactive and real-time systems followed by in-depth coverage of some aspects of such systems.

Our initial choice of these detailed subjects includes:

- analysis and design methodology ranging from informal to formal methods,
- the synchronous approach for reactive systems,
- real-time scheduling techniques.

The course structure is modular, hence an instructor using our course plan may replace one or more of these modules according to his/her preferences and the targeted audience.

As no textbook covers this wide range of topics, the course is mainly based on research papers, as well as chapters from various textbooks.

The emphasis of the course is the construction of reliable systems that can be formally proven to meet their specifications. Such systems are said to be provably correct systems. A formal framework for the specification, implementation, and verification of (temporal) properties of reactive systems contains:

- definition of the computational model of the system (i.e. the set of behaviors that are associated with the system),
- how to specify the desired requirements (especially safety and liveness properties) of a system within the computational model,

- the language to describe the system within the model (e.g. to express the proposed implementation),
- how to automatically verify that the implementation fulfills the required properties.

Our course outline is based on this framework, and in Section 2 each of the topics above is presented in detail.

1.4. Course Goals

The course focuses on aspects of specification and development of provably correct (reactive) real-time systems and approaches to real-time scheduling. An important goal is to give the students hands-on experience with the development of such systems.
After completing the course, the students should understand:

- what reactive systems are,
- what RTSs are,
- models of reactive systems (e.g., fair transition systems) and RTSs (e.g., clock transition systems),
- models for developing RTSs; concepts, methods and tools for the specification, analysis and design of real-time systems (note that we do not deal with the low level mechanisms that are needed at the programming stage),
- what the synchronous model is and some synchronous languages,
- verification methods,
- scheduling algorithms.

1.5. General Information of the Course

The ‘‘Reactive and Real-Time Systems’’ course is an optional graduate course, which is given to M.Sc. students of the Computer Science Department at Tel-Aviv University. Being a graduate student in Computer Science is the only prerequisite of the course. Nevertheless, since the Operating Systems course is an obligatory course for the undergraduate studies, it is assumed the students have knowledge about issues such as tasking, multitasking, real-time kernels, etc.

It is a one semester course, 13–14 weeks, 3 hours weekly. The first 9–10 lectures are given by the teacher, whereas the last weeks are in a seminar framework. The number of students is therefore limited to 24.

The course is considered to be of medium level of difficulty. Students who have taken the course maintain that, the fact that it combines theoretical issues with practical experience has two immediate consequences: it is easier to follow and understand the theory, and it makes the course ''vivid''.

There is no exam at the end of the course. The final grades are based on continual assessment of the assignments given during the semester, as well as the ''technical'' quality of the students presentations. Each student also summarizes the special topic he or she has covered. This summary is provided to the other students in the class and is part of the grading.

1.6. Comparison to Related Work

In order to compare the curriculum suggested in this paper to others, it is important to note the following difficultes in directly comparing our course to others.

1. The course is optional.

2. The students are graduate computer science students.

3. This is the only course (one semester) that deals with real-time systems.

For example, Halang (1990) suggests a very good curriculum for up to two semesters. Yet, he mainly refers to students of technical subjects such as chemical, electrical, and mechanical engineering, though also to computer science students. In order to cope with the heterogeneous background knowledge, Halang's course covers its area completely, at the cost of overlapping with other courses. He emphasizes the basic requirements with regard to time behavior (timeliness, simultaneousness, and predictability), as we do in our course. The rest of the course suggested in Halang (1990) refers mainly to real-time system languages (Pearl) and operating systems (Portos).

Other courses, such as described in Calvez and Pasquier (1998); Baron et al. (1998) refer to engineering studies, and offer a comprehensive curriculum where real-time systems are part of a total discipline. The curriculums they offer intend to educate and train system design engineers, whose work will be the development of real-time systems. We too emphasize the design stage, but of course we do it in less time, hence the smaller projects. One main idea in Calvez (1998) is that it is difficult to write specifications, and easier to write programs. Hence, it is less important to study real-time languages and real-time operating systems, and the subject of system design should be emphasized.

Saden (1997) offers a course in real-time software based on the combination of patterns and entity life modeling. This combination enables a high-level discussion on design rather than using traditional step-wise approaches. We have considered using patterns in the design process but decided not to change the ''traditional'' design methods used, at least for the time-being. Design is only one issue in our course and we did not want to spend more time on this topic at the expense of other important subjects.

From our experience of teaching the course ''Object-Oriented Software Engineering'' we know that to really understand patterns and use them properly we must spend more time on this issue. Furthermore, we do not want to turn the OOSE course into a prerequisite.

Zalewski (1993) offers a syllabus for an undergraduate real-time systems course. This course assumes knowledge of operating systems, distributed and parallel programming constructs and software engineering. The course emphasizes the following topics: real-time development methodologies, real-time language constructs, real-time kernels and real-time hardware architectures. The first of these topics is covered in this course although we use other methodologies. Aspects of real-time language constructs are studied in three different contexts. Issues concerning tasks and concurrency are studied in the operating systems course. Issues of time, priorities, and scheduling are studied in the scheduling chapter of our course, and statements like delay, terminate and abort, are covered in the synchronous language part of our course. Students have learned about real-time kernels in the operating system course. The topic of real-time hardware architectures, though very important, requires a much more elaborated background lacking in computer science graduate studies.

1.7. Course Assignments

In order to meet course requirements, the students are given several assignments. We distin-guish between two kinds of assignments, stand alone assignments to conclude each topic and several ongoing projects that evolve as the course progresses and more subjects have been taught. Some assignments require computational resources, whereas others do not.

The projects are done in teams of 2–3 students each, whereas theoretical assignments are done individually. Each assignment has its own length of time for completion.

2. Detailed Course Structure

This section elaborates on the topics that are covered during the course and includes references used.

2.1. Introduction

This part introduces reactive and real-time systems. It is mainly based on the following papers: Stankovic and Ramamritham (1988, pp. 1–9); Stankovic (1998); Hoogoboom and Halang (1988); Alur and Dill (1991) and books: Halang and Sacha (1991) and Manna and Pnueli (1992).

Topics covered.

- A definition of reactive systems.
- A definition of real-time systems.
- Fundamental concepts such as timeliness and predictability.
- A characterization of RTSs; typical examples of such systems. Terms such as size, complexity, reliability, safety, faults, fault tolerance, exception handling, error recovery.
- Time-critical activities, non time-critical activities (hard vs. soft RTSs).
- Processes, tasks, concurrency.
- Formal descriptions of RTSs.
- Common misconceptions (e.g. real-time is not necessarily synonymous with fast computation).

Assignments. The knowledge gained in this part is checked by means of theoretical questions (e.g., from Manna and Pnueli, 1992).

2.2. *RT Requirement Specification, Analysis, and Design*

There are many methods for the analysis and design of real-time systems.

2.2.1. *Informal Methods*

Gomaa (1993) includes a number of (informal) methods that can be used for real-time and concurrent system design, comparing them through application to a specific common problem. Methods described include Real-time Structured Analysis and Design (RTSAD), Jackson System Development, Parnas' Naval Research Lab/Software Cost Reduction Method, Object-Oriented Design, and DARTS. We cover one method (last year, for example, we used RTSAD).

Other possible references are Deutsch (1988), Ward (1986), and Gomaa (1986, 1989).

2.2.2. *Semi Formal Methods*

In order to describe the dynamic behavior of the system under development we use `Statecharts`. At this point, we still have not covered in depth the semantics of `Statecharts` (Harel, 1987) (e.g. is strong or weak preemption being used, etc.), as the

main purpose at this stage is to have a powerful tool to clearly describe the behavior of the system under development. However, after discussing some (even) very small examples, the importance of the semantics becomes very clear to the students.

2.2.3. Formal Methods

Traditionally, testing is the main method of program verification. However, in large and complex systems, complete testing is not possible. Due to the nature of real-time systems, specifically the often severe consequences of failure to satisfy the requirements (for example, in safety-critical systems), it is generally believed that RTSs should be specified using a formal language. Formal languages, along with additional notations and logic, enable representation and verification of temporal properties.

Topics covered.

- The need for formal methods for RTSs.
- Formal specification notations and logics like Temporal Logic (Pnueli and Harel, 1988, Chapter 3, pp. 179–214; Manna and Pnueli, 1992), timed- and clock-transition systems (Henzinger, 1991; Kesten, 1995; Alur and Dill, 1991), Real-Time Logic (RTL) (Jahanian and Mok, 1986), and finite state machines. One may choose to cover the Petri Nets formalism in this part of the course.
- Safety and liveness properties.

Assignments. For the formal specification, design, and verification parts, we provide several examples that are incrementally elaborated during the semester. For a detailed description of the projects given, refer to Section 4.

We begin with an informal method to specify the system under consideration (SUC). For example, in the last semester the students used RTSAD to find the processes and `Statecharts` to specify their dynamic behavior. The duration of this assignment is 4 weeks.

2.3. The Synchronous Approach

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 real-time systems. In the synchronous model, time is considered to be discrete; i.e. the time scale is divided into an (in)finite number of non-overlapping segments. These segments are called instants. The synchronous hypothesis states that inputs, reaction (computation), and output—all happen in the same instant.

One of the advantages of the synchronous approach to reactive systems is that it has a clean mathematical model in which instants are considered to be points on a time scale. In other words, they provide primitives that enable a program to be considered as reacting in zero time (instantaneously) to external events.

Halbwachs (1993) describes the synchronous approach and its languages.

Topics covered.

- What is the synchronous model?
- The advantages of the synchronous approach.
- Types of synchronous languages (imperative; dataflow; graphical).
- The syntax and semantics of the languages `Statecharts` and `Esterel`. `Statecharts` is a graphic synchronous language. `Esterel` is both an imperative, textual language, intended for reactive systems programming, as well as a compiler that translates `Esterel` programs into finite-state automata.
- The fundamentals of reactive programming languages.
- Various compilation methods for synchronous languages and their respective computational models.

This part of the course is based, among others, on the following papers:

- Synchrony in general: A general introduction to the synchronous approach (Halbwachs, 1993, pp. 1–6; Berry and Benveniste, 1991) overall presentation of the synchronous technology (Benveniste et al., 1994)
- `Statecharts`: General description of the language (Harel, 1987, 1988) and its semantics (Harel and Naamad, 1996).
- `Esterel`: A general introduction (Boussinot and de Simone, 1991) the `Esterel` language, its semantics, and compilation methods (Berry and Gonthier, 1992; Berry 1997, 1998; Poigné and Holenderski, 1995), `Esterel` as a process calculus, and a general discussion of preemption (Berry, 1993).[1]

Assignments. After the `Esterel` language is taught, the students implement—within two weeks—the systems analyzed and designed in the previous assignment (cf. Section 2.2) using `Esterel`. The `Esterel` simulator is used to check the behavior of the systems by means of simulation.

2.4. Verification

There are two main approaches to achieve provably correct software systems. The deductive approach uses theorem provers to build a proven-correct system from its specification. The algorithmic approach uses model-checkers to verify that the system

under development fulfills its specification. Both approaches need a formal specification of the system's intended behavior. Those techniques can be applied at different levels of formality and abstraction.

Our approach is to build provably correct reactive and real-time systems using symbolic model checkers. This suits our computational model well (e.g., state-machines which are the output of the compilation of `Esterel` programs). The formulas which specify the desired properties (requirements) of a system can be checked, by means of simulation, and can be proved using a model checker.

We also considered the use of a deductive approach (by employing TLV, Pnueli and Shahar, 1996, for example) to build a correct system from its specification. However, from the experience we had with some graduate students working on their Master's Theses, the students do not have enough background in logic, thus too much additional study would have been needed.

The properties that need to be proved can be divided into two categories: safety and liveness. Safety is a property that always holds, i.e. nothing wrong will ever happen, whereas liveness means that eventually something good will happen. Another way to define the difference between the properties is the following. If one can demonstrate that the property fails by looking at some finite-length prefix of system execution, then it is safety. If all the system executions need to be examined, then it is a liveness property.

Since we do not have a translator from `Lustre` and `Signal` to the input language of any model checker, another verification scheme is 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 model checker checks whether OK is constantly emitted.

Details of the software tools used for this purpose are elaborated in Section 3.

Topics covered. The intent of this topic is to use verification tools rather than to study the verification algorithms. The topics covered are:

- What is a model checker?
- What is a deductive theorem prover?
- The BDD approach.
- The usage of least- and greatest-fixed point as means to implement model checking algorithms.
- The SMV model checker.

Assignments. The students have to formally specify safety and liveness properties of the systems they implement. The properties are written in natural language and in any appropriate formal notation.

In order to formally verify that the specified properties indeed hold, the students use some of the following verification mechanisms:[2]

- Using an `Esterel` compiler that was written in the GMD, the source code is automatically translated into an input program for SMV (McMillan, 1993). The specified assertions are added and the verification process takes place.

- In order to use `TempEst`, the assertions are written in a variant of Temporal Logic, and are translated into `Esterel` code that runs in parallel to the original `Esterel` program. Violation signals are emitted for each unfulfilled property.

- The XEVE (Bouali, 1996) environment, which locates violations of desired properties and runs a simulator that enables automatic replay of erroneous traces.

The students have to fulfill the demands of this part within 4 weeks.

2.5. Scheduling

The scheduling of real-time computation is the phase where the final temporal properties of the computation are assigned. Schedulability analysis aims to prove whether all tasks can meet their deadlines.

In this part of the course, we classify the scheduling problems, and describe some of the approaches taken in order to fulfill the timing constraints in a hard real-time system. Surveys of scheduling issues can be found in Cheng and Stankovic (1988); Burns (1991) other references are Liu (1994); Klein et al. (1994); Sha et al. (1991); Buttazzo (1997).

Topics covered.

- Problem formulation: What is special with RT scheduling?

- Optimal real-time schedulers for underloaded systems: The earliest deadline first (EDF) algorithm and its optimality (Liu and Layland, 1973; Dertouzos, 1974); the least slack first algorithm and its optimality (Mok, 1983).

- Periodic task models, Liu and Layland's rate monotonic priority assignment (Liu and Layland, 1973; Sha and Goodenough, 1990; Lehoczky et al., 1989; Leung, 1989); cyclic executive vs. fixed priority scheduling (Locke, 1992); EDF scheduling of periodic systems.

- Catering for aperiodic and sporadic tasks in periodic systems (Sprunt et al., 1989).

- The synchronization problem: Priority inversion, priority inheritance mechanism and the priority ceiling protocol (Sha et al., 1990).

- Time-triggered and off-line schedules (Kopetz, 1991).

Assignments. The scheduling part is covered by several non-computer assignments, to be done within 2 weeks. The students are required to:

- Check whether a given set of tasks is schedulable using the various schedulers studied.
- Analyze the performance of different algorithms.
- Prove theoretical issues.

2.6. Student Talks

The last part of the course is given by the students in the form of a seminar framework. Topics include, but are not limited to:

- Real-time databases.
- Real-time communications.
- More on real time logics.
- More on synchronous languages and their semantics.
- Multimedia.

3. Software Environment and Tools

To demonstrate and practice the various subjects covered in the course, we use a set of software tools. Most tools are available from the Internet. These tools include:

- `gE`—Graphical Editor, a tool that was developed in the GMD (Budde, 1998b) and implements a subset of `Statecharts`. The ultimate intention is to use the graphic specification as an input to a verification tool, as will be discussed later. The tool has 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 systems have to act.

- The `Esterel` compiler. `Esterel` is a synchronous programming language for reactive systems. `Esterel` has a precise mathematical semantics intended for programming the class of deterministic reactive systems that wait for a set of possibly simultaneous inputs, react to the inputs by computing and producing outputs, and then quiesce, waiting for new inputs. `Esterel` is based on the ''synchrony hypothesis,'' which stipulates that every reaction to a set of inputs is considered to be instantaneous. The programming model in `Esterel` is the specification of components, or modules, that run in parallel. Modules communicate with each

other and the outside world through signals, which are broadcast and may carry values of arbitrary types. Consistent with the synchrony hypothesis, the emission and reception of signals is considered to be instantaneous.

- The `Esterel` Verification Environment XEVE (Bouali, 1996). XEVE is a graphical interface environment for the symbolic analysis and verification of `Esterel` programs modeled as finite state machines (FSM). Its main features are:

  - FSM model construction from netlist descriptions of `Esterel` programs,
  - FSM minimization using a notion of bisimulation equivalence,
  - output signal status checking under input constraints,
  - verification of safety properties expressed by observers by means of output signal status checking,
  - input sequences production in case of verification failure.

- The `TempEst` package. `TempEst` (Puchol et al., 1995) is a tool-set for the formal verification of safety properties, expressed in linear time temporal logic, of `Esterel` programs.

- The model checker symbolic model verifier (SMV) (McMillan, 1993) was developed at CMU (McMillan, 1993). SMV is a program that uses a symbolic model checking algorithm to evaluate formulas of CTL (Computational Tree Logic—a branching time temporal logic) with respect to a finite state model. We use a compiler for pure `Esterel`[3] that was developed at GMD, in St. Augustin, Germany. This compiler produces code in the SMV language.

- `sE`—Synchronous Eifel (Budde, 1998a), is a language developed at the GMD. This language enables specification of the control part using a dialect of `Esterel` or a subset of `Statecharts`, 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)[4]

- Verification interacting with synthesis (VIS) (VIS group, 1996) is a model checker that has been developed jointly at the University of California at Berkeley, the University of Colorado at Boulder, and at the University of Texas, Austin.

  VIS is able to synthesize finite state systems and/or verify properties of such systems, which have been specified hierarchically as a collection of interacting finite state machines.

- We located additional tools, including the following dataflow synchronous languages:
  - `Signal` (Gauthier et al., 1994).
  - `Lustre` (Halbwachs, 1993) and its verification tool LESAR which is a model checker for verifying Lustre programs.
  - `Modecharts` (Jahanian and Mok, 1988; Jahanian and Stuart, 1988).

  We have not taught them in class, but students are free to use them and get an extra bonus.

Figure 1 shows an example of an environment in which `Esterel` programs, temporal logic, and the SMV model checker are used in concert.

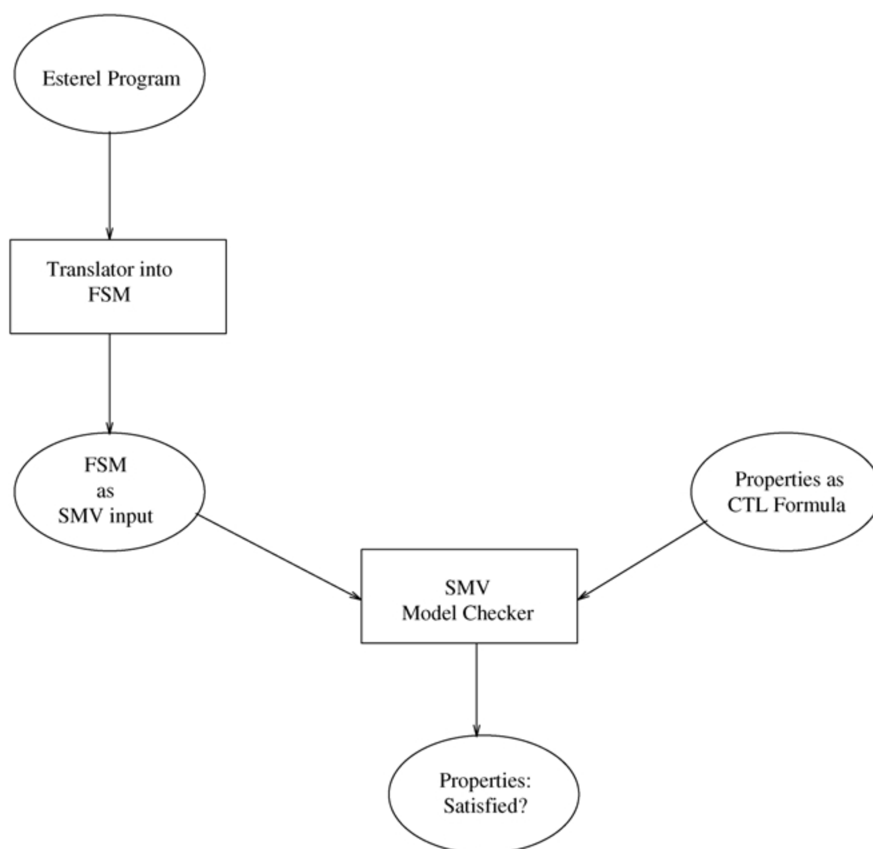

Figure 1. An example environment: the way a program is checked to meet its specifications.

4. Projects

This section elaborates on the major projects given in the course.

One of the issues we debated when choosing the projects was whether to assign one large project or a number of smaller ones. The latter choice was taken. The reasons are:

- The students are already experienced with the analysis of large information systems (either from their studies or from their work—almost all graduate students are working).
- The projects are just a means for experiencing the topics studied in class. Choosing large project introduces undesirable overheads arising from the sheer size of the code. For example, the verification process might take an unreasonably long time (even days for one verification).
- 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. Those projects are sufficiently complex to allow them to draw conclusions about realizing more complex systems as well.

Three projects are given to be undertaken in teams of three students each (hoping that each project was done by all three, rather than one project per student ...). For each project, a general informal description is given, and the students have to deliver a verified working system.

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

The projects given so far are the following.

A one lane bridge (junction) that connects two sides of a two-lane way. Informally, the requirements of this project are 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 in that lane. The light on the other side turns to green only after the junction is cleared.

This assignment is a simple task, suitable to introduce a new environment. In this project, synchronous model understanding, `Esterel` programming, and tools usage are experienced. With this experience in mind, teams could avoid many pitfalls while working on the other two projects.

A heat guided air-to-air missile controller. The missile contains a burner, a proximity trigger, a warhead, two set of wings; yaw (i.e., left, right) and pitch (i.e., up, down), a launch sensor and an infra-red camera. An algorithm is available that transfers the infra-red image of the camera to yaw and pitch angles of the target. Half a second after the missile is launched, the burner is activated. From then on, the burner remains active until the missile explodes or an error is detected. The system uses its

wings to aim the missile to the target. Each set of wings has five modes: `Centered`, `Small deviation`—one to each of the two directions, and `Strong deviation` to each direction. The missile is armed 3 seconds after it is launched. A burner malfunction, or the camera losing the target, causes the arming to be canceled and the burner to stop. If the missile is armed and the proximity detector is activated, the warhead explodes.

This is another simple project, which aims to exercise the development of the dynamic behavior of a control system.

An answering machine. This project deals 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 (Gajski et al., 1994).

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`) help to supply a compact control flow diagram (CFD).

An elevator control system. There are some "classical" case-studies requirements for this system. The students are free to implement any elevator control system located in a 3–4 story building. Some requirements are to use request buttons at each floor, sensors to locate the current position of the elevator car, door position sensors, etc. The requests from individual floors can be served in any chosen strategy, as long as it is a fair one (e.g. eventually, within predefined time limits, the request will be fulfilled).

An elevator control system has many typical aspects of an embedded system, as well as time, safety and liveness requirements.

A bottle filling system. This project is based on the example given in Ward and Mellor (1985) with several slight variations.

A cellular phone control unit. The phone contains a microphone, a speaker, an LCD display, a key pad (containing **On**, **Off**, **Send**, **End** and digits), a receiver and a transmitter. The receiver and transmitter have two modes: voice mode and signaling mode. The signaling mode is used to receive and transmit commands to the cell, and the voice mode is used for the call itself. When the transmitter/receiver are in the voice mode, the microphone's exit is transmitted and the input from the receiver is voiced on the speaker. On system startup the transmitter is waiting, and the receiver scans the spectrum trying find the best available cell (measured by signal quality). If no cell is found, a **SYSBUSY** message is printed; otherwise, the receiver and transmitter start working with that frequency. To start an outgoing call, the user enters a number, and the digits entered so far appear on the LCD display. Pressing **End** clears the number and pressing **Send** starts the call by sending a connect request with the dialed number (using the transmitter) to the cell. When a positive response is received from the cell, both the receiver and the transmitter go to voice mode. The call is terminated either by receiving a termination request from the cell or by pressing

the **End** button. If the **End** button was pressed, a termination request is sent to the cell.

Note that other issues, like incoming calls, hand-off, and speed dial, can also be addressed in the project. This project was chosen as an example of a complicated device that almost all students have. Both its dataflow and control flow parts are interesting and can be elaborated.

RFI lock system. The RFI (radio frequency identification) system is used to open a door with a contactless key and has the following major functions: register a programming-key, registering an open-key (with or without a programming-key), erasing an open-key or the programming key, opening a door.

For example, we describe here the procedure to open a door. In order to open the door, an RFI-key, which is registered as an open-key for a lock is brought close to the corresponding RFI-sensor. If the RFI-key is identified as a registered open-key for this lock, the door is opened, and a LED will light as long as the door remains open. When we move the RFI-key further away from the sensor field, the door will be locked after a delay of 3 seconds and the LED light goes off. If the key is recognized, but not identified as an open-key, another LED flashes in short intervals and the door will not open. If the LEDs remain off, this means that no RFI-key was recognized.

The main problem the students encounter with this project is to describe the behavior.

A washing machine. This is a regular washing machine, with specifications for: the washing programs; starting and stopping the machine; handling the requested temperature; activating the heating body (when water level is at a given level) and keeping the required temperature when reached; water flow into the machine (up to a given maximal value) and out; starting the machine's engine and rotating the drum once the water level is at a given position; all activities are stopped once the machine's door is open.

The production cell system. The production cell is used in a metal processing factory. It consists of two belts (a feed- and a deposit-belt), an elevating rotary table, a robot, a press, and a crane, which is only used in the toy model built in FZI. The full details including sample implementation can be found in Lewerentz and Lindner (1995).

Two kinds of properties are required from the implementation of the production-cell. The most important is the safety requirement. Thus, for example, collision between devices must not be allowed, blanks should not be dropped outside ''safe'' areas, etc. Another important requirement is the liveness of the system. This is phrased as ''Each metal blank introduced into the system via the feed-belt should eventually arrive at the end of the deposit-belt and has been forged by the press''.

A home alarm system control module. The alarm system is based on five sensors, a horn and a key pad (containing **OFF, HOME, AWAY,** and **SETUP** buttons and LEDs, a button and a LED for each of the sensors, digits, * and #). The system has three normal operation modes: **OFF**, **HOME**, and **AWAY**, and a **SETUP** mode used to configure the sensors and the secret code. At each state, the LED for that

state is turned on. Each sensor has three states: **OFF**—inactive; **HOME**—active in HOME and AWAY modes; **AWAY**—active only in AWAY mode. When the sensor detects activity, the sensor LED is turned on, unless we are in the mode **SETUP**. The system starts in the OFF mode. Entering the AWAY or HOME modes from the OFF mode is done by pressing the appropriate button for 1 s. It is only possible to enter these states from the OFF mode. Entering to the HOME state immediately activates the appropriate sensors. Moving to the AWAY state activates the appropriate state only after 15 seconds (enabling the user to leave the premises). Activity detection by one of the sensors, while it is active, starts a timer. If within 10 seconds the system is not back in the OFF mode, the horn starts sounding the alarm, until the OFF state is reached. Moving back to the OFF state from the HOME and AWAY states is done by typing the four digit secret code and then pressing the OFF button.

The SETUP state, sensor configuration, changing the secret code and more can also be addressed in this project.

A video cassette recorder (VCR) system controller. The VCR contains a tape engine, an eject engine, a TV output switch, a tape existence sensor, an EOT (end of tape) sensor and a keypad (containing **FF**, **REW**, **STOP**, **PLAY**, **EJECT**, **ON**, **OFF**, and an LCD display of the time passed in the tape). When a tape is entered, or the ON button is pressed, the system moves to the On state. Pressing the OFF button returns the system to the Off state. Pressing EJECT when a tape is inside the VCR, causes the eject engine to eject the tape. When the tape is in and the system is On, the rest of the system is activated. The tape engine has six states: **Stop**, **Play**, **FF-Play**, **Rew-Play**, **FF**, and **Rew**. In the states Play, FF-Play, and Rew-Play the TV output switch passes the tape video output to the television. Pressing the STOP button—in any state—stops the tape and move the system to the Stop state. Pressing FF or REW while the system is in the Play mode, changes the system state to FF-Play or Rew-Play, respectively. Pressing FF or REW while the system is in Stop, changes the system state to FF or Rew, respectively. Pressing a button while the system is already in the appropriate state has no effect. Pressing FF while the system is in the Rew or Rew-Play states, and REW while in the FF or FF-Play states, reverses the engine direction within 0.2 s.

Other issues --- like EOT handling, time display processing, and more --- can also be addressed in this project.

After the analysis phase, the students are asked to write, in a natural language, properties that the system had to fulfill. A class discussion soon reveals that many of those requirements are ambiguous and unclear, and difficult to follow in the students' analysis. The conclusion is that a more formal representation is needed. Now the students are asked to write specifications in Temporal Logic, Real-Time Logic (RTL) (Jahanian and Mok, 1986; Mok, 1991), RTTL (Ostroff, 1992) or MASS/PLOT (Gafni, 1996). The intention is to use these requirements for formal proof of some (very) small examples. Nevertheless, mainly due to lack of enough knowledge in logic (cf. Section 2.4), the students usually cannot manually prove properties.

At this stage, however, it is already clear to the students that testing and simulation as the only methods of program verification are not feasible in large, complex systems. Specifying the desired properties of the systems in an appropriate formal language enables the use of automated verification tools.

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

5. Case Study: An Elevator System

In this section we elaborate on how one team tackled the elevator project. The other two assignments in that particular semester (Spring, 1998) were the one-lane bridge system and the answering machine system.

Throughout this description the reader may observe the knowledge acquisition process and the problems encountered during the development process.

The control (distributed vs. centralized) and service algorithm (which floor to move to) are 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[5]. 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 140 M 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 h of execution. It is important to notice that the software we use is academic and not commercial, hence the problematic performence. It is reasonable to assume that using commercial tools for these project would have resulted in much better performance. Furthermore, the students use 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.

In order to verify the system, 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 cannot 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 verification 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.

6. Conclusion

The topics studied during the course are 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 make clear aspects that seem to be unrelated. The computational model, for example, is 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 becomes quite clear. Hence, the benefit from the projects is not limited to learning 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.

In this paper we described the Real-Time and Reactive systems course structure. We presented the students' assignments, and the final projects given at the various courses. We elaborated on the elevator project as undertaken by one team. Comparing the efforts expended by the teams on each project and within the projects on 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 projects that we have chosen to use in the course are modular. This enables us to reuse the basic specification while changing them enough to prevent the reuse of projects from previous semesters.

The dataflow synchronous languages `Lustre` and `Signal` are both available at our site and students were encouraged (by a special bonus) to use them; however, no team chose these languages. We still cannot definitely explain the reasons for this. 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?

The verification tools were primarily used in the projects with qualitative temporal reasoning about systems. Using abstractions, the teams 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.

Acknowledgments

We are grateful to Gilad Koren and to Ran Lotenberg. Gilad was involved in the process of building the course syllabus (Koren and Tyszberowicz, 1997). Ran was a student in one of the reactive real-time systems courses (Lotenberg and Tyszberowicz, 1998).

Notes

1. A complete programming example of a digital wristwatch can be found in Boussinot (1991).
2. A detailed description of the tools is given in Section 3.
3. That is, with no valued signals, no data etc.
4. The use of the `sE` language is not mandatory in the course. We noticed, however, that most of the students have been trying also this environment.
5. 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.

References

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

Alur, R., and Henzinger, T. A. 1991. Logics and models of real time: a survey. In *Real-Time: Theory in Practice, Proceedings of the REX Workshop*. LNCS Vol. 600. Mook, The Netherlands, pp. 74–106.

Baron, C., Geffroy, J.-C., and Motet, G. 1998. Dependability issues for a curriculum in real-time systems. In J.-J. Schwarz, J. Nawrocki, and J. Zalewski (eds), *Proceedings of the Third IEEE Real-Time Systems Education Workshop*. Poznan, Poland, IEEE Computer Society, pp. 16–23.

Benveniste, A., Berry, G., Caspi, P., Couronné, P., Dupont, F., Gauthier, T., Halbwachs, N., Le Guernic, P., Le Maire, C., Mignard, F., Paris, J. P., and Sorel, Y. 1994. Synchronous technology for real-time systems. In *Proceedings of Real-Time Systems'94*. Paris, North-Holland.

Berry, G., and Benveniste, A. 1991. The synchronous approach to reactive and real-time systems. *Another Look at Real Time Programming, Proceedings of the IEEE* 79: 1270–1282.

Berry, G. 1993. Preemption and concurrency. In *Proc. FSTTCS 93*. LNCS vol. 761. Berlin, Springer-Verlag, pp. 72–93.

Berry, G. 1997. A quick guide to Esterel. http://zenon.inria.fr/meije/esterel/

Berry, G. 1998. The Esterel v5 language primer. http://www.inria.fr/meije/esterel/

Berry, G., and Gonthier, G. 1992. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming* 19(2): 87–152.

Bouali. A. 1996. Xeve: An Esterel verification environment. http://cma.cma.fr/Verification/Xeve/

Boussinot, F. 1991. Programming a reflex game in Esterel v3_2. Rapport de recherche 07/91, Centre de Mathématiques Appliquées, Ecole des Mines de Paris, Sophia-Antipolis.

Boussinot, F., and de Simone, R. 1991. The Esterel language. *Another Look at Real Time Programming, Proceedings of the IEEE* 79: 1293–1304.

Budde, R. 1998. The design and programming language synchronousEifel (sE-version 1.1). GMD, St. Augustin, Germany, http://ais.gmd.de/ ~ budde/

Budde. R. 1998. The Graphic-Editor for synchronousEifel (sE-version 1.1). GMD, St. Augustin, Germany, http://ais.gmd.de/ ~ budde/

Burns, A. 1991. Scheduling hard real-time systems: A review. *Software Engineering Journal* 6(3): 116–128.

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

Calvez, J.-P., and Pasquier, O. 1998. Training engineers in real-time systems design: an integrated curriculum. In J.-J. Schwarz, J. Nawrocki, and J. Zalewski (eds), *Proceedings Third IEEE Real-Time Systems Education Workshop*, Poznan, Poland, IEEE Computer Society.

Cheng, S.-C, and Stankovic, J. A. 1988. Scheduling algorithms for hard-real time systems—a brief survey. In J. Stankovic and K. Ramamritham (eds), *Hard Real-Time Systems*. IEEE Computer Society Press, pp. 150– 173.

Dertouzos, M. L. 1974. Control robotics: the procedural control of physical processes. In *Proceedings IFIF Congress*, pp. 807–813.

Deutsch, M. S. 1988. Focusing real-time analysis on user operations. *IEEE Software* 4(5): 39–50.

Gafni, V. 1996. Real-time activation oriented specification language. Technical Report, Computer Science Department, Tel-Aviv University.

Gajski, D. D. Vahid, F. Narayan, S., and Gong, J. 1994. *Specification and Design of Embedded Systems*. Prentice-Hall.

Gauthier, T., Le Guernic, P., and Maffeis, O. 1994. For a new real-time methodology. Research Report 2364, INRIA.

Gomaa, H. 1986. Software development of real-time systems. *Communications of the ACM* 29(7): 657–668.

Gomaa, H. 1989. Structuring criteria for real-time systems design. In *Proceedings of the 9th International Conference on Software Engineering*. Pittsburgh, pp. 290–301.

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

The VIS Group. 1996. VIS: A system for verification and synthesis. In R. Alur and T. Henzinger (eds), *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*. LNCS Vol. 1102, Berlin: Springer-Verlag, pp. 184–195, http://www-cad.eecs.berkeley.edu:80/Respep/Research/vis/

Halang, W. A. 1990. A curriculum for real-time computer and control systems engineering. *IEEE Transactions on Education* 33: 171–178.

Halang, W. A., and Sacha, K. M. 1991. *Constructing Predictable Real-Time Systems*. Boston: Kluwer Academic Publishers.

Halbwachs, N. 1993. *Synchronous Programming of Reactive Systems*. Dordrecht: Kluwer Academic Publishers.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8(3): 231–274.

Harel, D. 1988. On visual formalisms. *Communications of the ACM* 31(5): 514–530.

Harel, D., and Naamad, A. 1996. The Statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology* 5(4): 293–333.

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

Hoogeboom, B., and Halang, W. A. 1988. The concept of time in the specification of real-time systems. In Krishna M. Kavi (ed.), *Real-Time Systems—Abstraction, Languages, and Design Methods*. IEEE Computer Society Press, pp. 19–38.

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

Jahanian, F., and Mok, A. 1994. Modechart: A specification language for real-time systems. *IEEE Transactions on Software Engineering* 20(12): 933–947.

Jahanian, F., and Stuart, D. 1988. A method for verifying properties of Modechart specifications. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*. Los Alamitos, CA: IEEE Computer Society Press, pp. 12–21.

Klein, M. H. Lehoczky, J. P., and Rajkumar, R. 1994. Rate monotonic analysis for real time industrial computing. *IEEE Computer* 27(1): 24–33.

Kesten, Y., Manna, Z., and Pnueli, A. 1995. Clocked transition systems. Stanford Technical Report, Department of Computer Science.

Kopetz, H. 1991. Event-triggered versus time-triggered real-time systems. In *Proceedings of International Workshop on Operating Systems of the 90s and Beyond*. LNCS, Berlin: Springer Verlag, pp. 87–101.

Koren, G., and Tyszberowicz, S. 1997. Graduate course: Reactive and real-time systems. In D. Mossé, and J. Zalewski (eds), *Proceedings of the 2nd IEEE Real-Time Systems Education Workshop*. Montreal, Canada, pp. 96–103.

Lehoczky, J. P., Sha, L., and Ding, Y. 1989. The rate monotonic scheduling algorithm – exact characterization and average case behavior. In *Real Time Systems Symposium*. pp. 166–171, IEEE.

Leung, J. Y.-T. 1989. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica* 3(1): 209–219.

Lewerentz, C., and Lindner, T. (eds). 1995. *Formal Development of Reactive Systems. Case Study Production Cell*. LNCS Vol. 891, Springer Verlag.

Liu, C. L. 1994. Fundamentals of real-time scheduling. In W. A. Halaang and A. D. Stoyenko (eds), *Real-Time Computing*. Springer-Verlag.

Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real time environment. *Communications of the ACM* 20(1).

Locke, D. C. 1992. Software architectures for hard real-time applications: Cyclic executive vs. fixed priority executives. *Real Time Systems Symposium* 4(1): 37–53.

Lotenberg, R., and Tyszberowicz, S. 1998. Student projects in reactive and real-time systems course. In J.-J. Schwarz J. Nawrocki and J. Zalewski (eds), *Proceedings of the Third IEEE Real-Time Systems Education Workshop*. Poznan, Poland, pp. 61–66.

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

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

Mok, A. K.-L. 1993. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

Mok, A. 1991. Towards mechanization of real-time system design. In A. M. van Tilborg and G. M. Koob (eds), *Foundations of Real-time Computing: Formal Specification and Methods*. McGraw-Hill, pp. 1–37.

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

Pnueli, A., and Harel, E. (1998). Applications of temporal logic to the specification of real time systems. In M. Joseph (ed.), *Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*. LNCS Vol. 331, Springer-Verlag, pp. 84–98.

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

Poigné, A., and Holenderski, L. 1995. Boolean automata for implementing pure ESTEREL. Technical Report, GMD, Schloß Birlinghoven, D-53754, St. Augustin, Germany.

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

Sanden, B. 1997. A course in real-time software based on concurrent design patterns. In D. Mosse and J. Zalewski (eds), *Proceedings of the Second IEEE Real-Time Systems Education Workshop*, Montreal, Canada: IEEE Computer Society, pp. 24–29.

Sha, L., and Goodenough, J. 1990. Real-time scheduling theory and ADA. *IEEE Computer* 23(4): 53–62.

Sha, L., Lehoczky, J. P., Strsnider, J. K., and Tokuda, H. 1991. Logics and models of real time: A survey. In A. M. van Tilborg and G. M. Koob (eds), *Foundations of Real-Time Computing: Scheduling and Resource Management*. Kluwer Academic Publishers, pp. 1–30.

Sha, L., Rajkumar, R., and Lehoczky, J. P. 1990. Priority inheritance protocols. *IEEE Transactions on Computers* 39(9): 1175–1185.

Sprunt, B., Sha, L., and Lehoczky, J. P. 1989. Aperiodic scheduling for hard real-time systems. *Real Time Systems Journal* 1(1): 27–60.

Stankovic, J., and Ramamritham, K. (eds). 1988. *Tutorial: Hard Real-Time Systems*. IEEE Computer Society Press.

Stankovic, J. 1988. Misconceptions about real-time computing. *IEEE Computer* 21(10): 10–19.

Ward, P. T. 1986. The transformation schema: An extension of the data flow diagram to represent control and timing. *IEEE Transactions on Software Engineering* 12(2): 198–210.

Ward, P. T., and Mellor, S. J. 1985. *Structured Development for Real-time Systems*. Prentice Hall, Englewood Cliffs, 1985.

Zalewski, J. 1993. Real-time systems. Undergraduated course syllabus. http://cs-www.bu.edu/ftp/IEEE-RTTC/public/syllabus.ps.

Tal Lev-Ami received his B.A. in Computer Science from the Open University of Israel in 1995 and his M.Sc. in Computer Science from Tel-Aviv University in 2000 both Summa Cum Laude. He is currently working as a senior software engineer in an Israeli software company.

Shmuel S. Tyszberowicz received his Ph.D. degree from Tel-Aviv University in 1990. He studied mathematics and computer sciences. His current research interests include specification and verification of real-time reactive systems, and object-oriented analysis and design, especially for developing reactive systems.