Teaching Software-intensive Embedded Systems at Tallinn Technical University

Leo Motus
Real-time Systems
Department of Computer Control
Tallinn Technical University, Estonia, 19086
leo.motus@dcc.ttu.ee

Abstract

The experience obtained by developing, delivering and modifying three courses on embedded software is discussed. Also the goals and reasons for developing these courses are explained.

The courses are – "Introduction to real-time software engineering", "Software dynamics" meaning timing analysis of software, and "Real-time software engineering environments". All the courses address the starting stages of software development and are biased towards studying timing correctness of systems and software.

Keywords: *embedded systems, software engineering, object-oriented design, formal and informal timing analysis, UML*

1. Introduction

The name "embedded systems" is often interpreted in two, slightly different ways. It may denote an in-built dedicated microprocessor based computing system that forms an inseparable part of a device or its part, or any time-critical computing system. The first case is illustrated, for instance, by applications in automotive industry, medical and biomedical engineering, robots and autonomous moving vehicles, advanced measuring and monitoring devices, advanced household devices. The second case covers process control applications, traffic control, environment monitoring, many banking systems, and others.

In spite of differences in hardware and implementation, the software for both classes of embedded systems should meet similar basic requirements. These requirements are substantially influenced (see, for example [1]) by forced concurrent execution of tasks. Forced concurrency implies, for instance:

- constraints on activation instants of tasks, often determined by the surroundings of embedded computer system
- strict deadlines on execution times of software processes, often imposed by the dynamics of surrounding physical and/or technical processes
- possibly strict timing constraints on interprocess communication and on the validity intervals of data and events
- increased demands on data integrity in time domain
- rather advanced exception handling procedures to meet the safety and reliability requirements.

Consequently, the starting stages of embedded systems and software development are guided, independently of the interpretation given to the word "embedded", by the same principles. Hence, similar basic development methodologies and tools can be used. Here, the starting stages mean requirements' specification, software specification, and logical design. It is essential that each of those stages ends with formal (if possible) verification. Formal verification at the starting stages reduces the rather large percentage of errors that are normally detected at implementation and integration test stages and are caused by inconsistencies and/or contradictions implanted at starting stages.

Another feature that needs explicit addressing when teaching real-time systems is the catastrophic under-estimation, by students, of the importance of feedback in embedded software. Students, and graduates, with conventional computer science background are used to reliable and obedient computers and tend to forget the, not so obedient, real world that is immediately interacting with the embedded computer system. Students from systems engineering, computer engineering, telecommunication, and electronics are remarkably better

aware of the feedback role than the computer science students.

In embedded applications the response to any command issued by the computer system, and the reactive behaviour of sensors and actuators, should be checked periodically. The checking procedures are not trivial, consume processor time, increase complexity of exception handling and, of course, influence the timing correctness of the embedded system. An example of sad effects caused by a not quite sufficient checking of the actuator's response is described in [2]. The reference describes an oil refinery explosion because a valve had stuck and this remained undetected by computer system.

The author of this paper is convinced that a student, specialising in the real-time systems should be aware of the essential difficulties emerging in development of embedded software when using conventional software development tools and methodology. These difficulties stem from the basic differences in paradigms (see, for example [1]) applicable to embedded systems and to conventional information processing systems. At the same time, tools, and huge amount of knowledge obtained in specification, design and development of conventional computer systems should find maximum usage. The set of courses discussed in this paper start from a widely used (e.g. object-oriented) design methodology, explain the basic ideas and give some practical experience. After that the deficiencies of the used design methodology are discussed from the viewpoint of (hard) real-time systems, the need for alternative solutions is explained and some alternative solutions are discussed. The major mentioned deficiencies, in a nutshell, are :

- timing analysis is introduced too late in the system life cycle (not before physical architecture design stage, see [3,4])
- even then, focus is on peformance-type integral timing characteristics, neglecting subtle and potentially more malicious characteristics (see section 3.2 of this paper)
- difficulties in applying formal verification technique to timing properties [1].

This paper introduces a set of three courses that build up from the conventional software tools, explain the essence of real-time embedded software, and suggest how to combine formal verification of timing properties with conventional software tools. These courses have been evolving during the last six years. This set of courses is to provide basic understanding and experience in the starting stages of developing software-intensive embedded systems, with special attention on time-critical systems. By now, the theoretical content of the courses has reached certain maturity level. The laboratory environment is still under development – partly because of the difficulty in forcing various real-time software engineering tools to co-operate conveniently. The ambitious future plan is to achieve a better co-operation between the practical exercises on these courses with practical exercises performed in courses on intelligent control, industrial data communication, distributed systems, and programmable logical controllers.

The paper proceeds with explaining the overall context of the embedded systems education at Tallinn Technical University. Each of the three mentioned courses is described superficially, concentrating on the main ideology and software tools used. The paper concludes with some observations collected and experiences obtained, and with the presentation of future plans.

2. Background of courses on developing embedded systems.

Estonian universities have gone through a dynamic period of reorganisations after Estonia regained its independence, lost during the World War II. By now the university system has reached a period of stable evolution. The universities give Bachelor's, Master's and PhD degrees based on credit point system. Tallinn Technical University is the main provider of engineering education in Estonia. Estonia has returned to procedures where universities give only education and academic degrees. The title of professional engineer (called Estonian Chartered Engineer) is granted by the Estonian Engineering Association on the basis of accredited courses taken at the University and practical experience obtained in industry. The engineering courses have been accredited by FEANI (European Federation of National Engineering Associations), are included into the 1997 FEANI index, and are being accredited at the national level. It should also be noted that continuing professional development is a compulsory precondition for maintaining the title of Estonian Chartered Engineer over a period of five years.

The courses on developing software-intensive embedded systems are delivered as part of the major module for systems and computer engineering, and as part of the minor module for electronics and biomedical engineering, telecommunication, and informatics. Two courses -- "Introduction to real-time software engineering" (3.5 CP) and "Software dynamics" (3.5 CP) -- are delivered at the bachelor's level. Both courses are supported by practical exercises on CASE tools and by a one-term project on design of software architecture for a real-time system.

Software dynamics denotes here formal and informal timing analysis of software. One course -- "Real-time Software Engineering Environments" (4 CP) – is at the master's level. All these courses focus at the specification and design stages and emphasise the methods applicable to formal and informal correctness study of specifications and designs.

From the systems development point of view, it is important to link the abovementioned courses with those focusing on systems modelling, analysis, and development of algorithms. Efforts have been made to maintain a close link with courses on automatic control theory, intelligent control, artificial intelligence in real-time. At this time, simply by providing examples, and explaining the potential problems invoked by transition from systems modelling and analysis to designing and implementing embedded systems.

The abovementioned three courses are, in a similar way, linked to courses on programmable controllers, industrial data communication, distributed systems, software engineering, and electronic design that partly cover the physical design, implementation and testing stages.

So far the weakest point in the education of embedded systems in Tallinn is the incomplete coverage of hands-on experience in systems development life-cycle. Reasons are pretty usual – insufficient number of good specialists, and insufficient laboratory equipment. The main focus in Tallinn is on the starting stages of the systems life-cycle, since they need more brainpower and slightly less expensive software tools and hardware. The starting stages are, in general, less thoroughly studied and taught in the universities – obviously for pragmatic reasons. Therefore they pose a greater challenge to researchers (see, for example, [5]). They also promise remarkable reduction in the number of errors that are usually detected and eliminated at implementation and testing stages -- since by applying proper tools and methodology these errors can be detected and eliminated at the starting stages of development.

3. Survey of course contents

The courses depart from the existing deficiencies (see section 1) of conventional software engineering tools and gradually introduce alterations required by the paradigms and mathematical models leading to practically usable formal methods for verification of timing properties. The joint influence of three courses is expected to teach the students:

- to consider embedded real-time systems as consisting of repeatedly (cyclically) executed, interacting, terminating processes, whereas some of the processes reside in a computer system, some reside outside of the computer system;
- to understand why are time constraints imposed on some of the processes and what are the main sources of those constraints (see the remark at the end of this list)
- to understand the essence of theoretical preconditions and methods for formal verification of time correctness of systems' behaviour, and the necessity of combining formal and informal timing analysis
- to apply the verification of timing properties in practice
- to understand the functioning principles and use in practice a variety of CASE tools and methods, suitable for specification and design of real-time systems.

Remark: Detailed discussions on this issue can be found in [1, 5]. Time constraints are needed to match the behaviour of interacting processes that form the system, to ensure the static axiomatic base for algorithms, to cope with the incomplete information (e.g. partially known causal relations, approximation of non-linearities) and various uncertainty factors (e.g. unpredictable number of iterations, length of queues, instant traffic load in networks) present in the controlled and/or monitored environment and computer system. Major source of the time constraints is the environment, additional constraints can be derived from mathematical models (theories) used in the computer systems. The difference between embedded (real-time) systems and conventional information processing computer systems is caused by the same reason as the difference between closed loop control theory and open loop control theory -- behaviour of the system changes when closing the loop, the character of changes is often impossible to predict analytically.

Getting too deeply involved with commercial CASE tools is a temptation and serious danger that one should avoid when teaching students -- one should focus on invariant features of the software process. It is probable that during the five years (that is the average period required for a student to become a mature specialist) the choice of commercially available CASE tools may change substantially. Therefore the courses discussed in this paper attempt to focus on invariant properties of embedded systems, defining preconditions for applying formal methods and various tools, explaining the basic functioning principles of the tools. It is true, that such orientation of courses puts a pressure on students' home-work – to a large extent, they have to become acquainted

with the available laboratory tools on their own, just guided by mentors.

3.1 A course "Introduction to real-time software engineering"

This is an autumn term course (16 weeks), with three lecture hours and one hour of laboratory training per week for third year students. Students have a possibility to get more training time, provided the laboratory resources are available – the resources for additional training usually can be accessed on the first-come-first-served basis. The CASE tool used in the laboratory is an UML based ARTISAN Real-time Studio, released by ARTISAN Software Tools Ltd in the beginning of 1998.

The course is roughly partitioned into two equal halfs. The first half of the course comprises a part explaining the essence of real-time systems and embedded software [1,6], and a part reminding the basics of project management for real-time systems [7]. The second half of the course is dedicated to widely accepted real-time systems design methodologies. A smaller part of the course discusses a methodology based on data flow models [8], a larger part focuses on object-oriented software engineering [9], ROOM [4], OMT [10] and its extensions to UML [11]. The books are used as references for further reading, a separately written course text (in Estonian) is based on excerpts from different books.

Preconditions for the course are basic engineering and programming education, preferably some experience with software or system development. The main goal of this course is to point out why real-time software engineering is required, what are the deficiencies of the methods and tools developed for conventional information processing and later extended to cover (partially) the embedded systems domain, what are the most frequent pitfalls.

An attempt is made to reveal the continuity of CASE tool's evolution process – the UML being a logical integration of many separately evolved methods. Ideological similarity of entity-relationship, data flow, and object-oriented approaches is pointed out. Existing real-time extensions [3,4] of object-oriented approach are discussed. UML is chosen as a basic methodology, partly because of the existence of a local research group working at integrating UML and LIMITS (a tool for formal verification of software timing correctness). LIMITS tool and the underlying theory are explained in the course on "Software dynamics" (see section 3.2 of this paper).

At the end of the course the importance of timing issues at the starting stages of a project is pointed out using examples derived from practice. It is also demonstrated how cumbersome are the commercially available tools in verifying quantitative timing correctness of the specifications and designs. Many statements made in lectures are illustrated in the laboratory by a public discussion of tasks solved by students as a hands-on exercise.

3.2 A course "Software dynamics"

This course is optional and recommended only for those planning to specialise in real-time systems. It is based on a (Q-model) theory published in [1], and on a CASE tool LIMITS developed, and still evolving at Tallinn Technical University. The name "software dynamics" is used here to denote the study of software behaviour in time, verification of quantitative and qualitative timing properties of software specification, design and implementation, and development of tools and methodologies for the beforesaid.

This is a spring term course (16 weeks), with three lecture hours and one hour of practical training per week for third year students. The CASE tool LIMITS-PC, used in the laboratory is locally developed for Windows NT platform. The LIMITS tool is also implemented for UNIX platform (in a European Union Copernicus project CP-94 no.1577). From the teaching point of view both tools are equivalent. Pragmatically speaking, LIMITS-PC has a well-defined modular structure, which enables to use its analysis engine separately as an OEM component. For instance, analysis engine is used as a module for run-time checking of timing correctness, whereas LIMITS as a whole is used for analysing time correctness of specification and design during the development of a time-critical application (see, for example [12]). The UNIX version of LIMITS is more closely integrated, its parts can not function separately.

The lecture course comprises three parts. The state of art of timing analysis is critically surveyed in the first part – the message delivered claims that time model used conventionally in computationaal models is too trivial to enable complete analysis of software timing properties. Examples of properties that need more complex time models are given (e.g. time-selective interprocess communication, validity intervals of events and data), performance bound properties can, in principle, be satisfactorily estimated by using trivial time-models.

The second part introduces a new computational model (the Q-model), and illustrates its use for systems

description. The Q-model can be described in (and proven formally consistent by using) a weak second-order predicate calculus. This feature will remain hidden from the students (unless somebody is really interested in the foundations of the computational model). The third part of this lecture course demonstrates the formal verification of some timing properties, explains how and why formal and informal verification of timing properties should be combined.

Verification in the Q-model approach attempts, as a rule, to deal with generic properties. This enables to state and prove theorems beforehand. In such case, during the specification and design process it remains to check that the actual parameter values satisfy the assumptions of the already proven theorems. Normally the assumptions take a form of simple algebraic inequalities between timing parameters of involved processes, for details see [13]. Some examples of theorems are:

- non-violation of correct order of termination and message consumption of concurrent copies of the same process
- non-violation of time constraints of a process while interacting via synchronous channel
- estimation of maximum non-transport delay of a message received via asynchronous channel.

Due to the abovedescribed approach to verification, many details of the theoretical burden can be hidden from the end-user. It is important to note that timing verification is always partitioned into formal and informal. The formal verification covers generic timing properties of any system and performance bound properties of the particular system. Quite often the incomplete information about the computer system's environment causes difficulties in estimating the parameter values. In such cases informal verification (validation) procedure that is based on animation and/or simulation is applied. After the Q-model is formally verified, even if the verification is partial, a prototype can be automatically generated. The animation is based on this prototype and can help in selection of suitable values for unknown timing parameters, in checking the feasibility of system's behaviour with certain parameter values, etc. The modified specification and/or design inevitably leads to a new round of formal verification.

The lectures are accompanied by practical exercises of students on LIMITS CASE tool. The object models developed during the previous course (Introduction to real-time software engineering) can be transformed to Q-models, and then analysed for timing correctness. It is possible to start timing analysis with developing the Q-model. Many students, however, find it easier to start with transforming object model (or data-flow based model) to the Q-model. Besides, from the teaching point of view, the transformation to Q-model and, after the verification, back to the original model, facilitates the understanding of integrity of software development process and demonstrates the impact of timing on the systems structure.

3.3 A course on “Real-time software engineering environments”

This is an autumn term course (16 weeks), with three lecture hours and one hour of practical exercises per week for Master students. The course gives a state-of-the-art survey of methodologies, methods, and tools that have lead to the contemporary real-time software engineering methods applicable in the specification and design stages. Whereas the previous two courses focused on hard real-time systems, this course surveys also methods applicable for soft real-time. Again, the idea is to demonstrate the evolution of tools and methods, and not to transfer fixed practical knowledge of manipulating specific tools. As before, the goal is to create flexible thinking, indicate that a panacea is still not available, and provide an understanding of the variety and essence of alternatives of tools and methodologies from which to choose.

Practical exercises intend to put the available CASE tools and design methodologies into a wider context and encourage the critical comparison of methodologies. The comparison should be based on three pillars -- on student's own experience, on what is told in lectures, and on what they have read in books and papers.

Unfortunately there is no compact text for this course yet. The examples have been picked up from various books and research papers. The existing surveys of CASE tools have been written from rather pragmatic aspects and do not provide the features that would satisfy the goals of this course (see, for example [14,15]). The tools surveyed in this course include SADT, SREM, EPOS, CONIC, Matsumoto’s method, HRT-HOOD, ROOM, OMT all compared with the tools (Artisan real-time studio and LIMITS) in the laboratory. In addition, some formal model based approaches are discussed and compared – for instance, path expressions, Petri nets, temporal logic.

4. Conclusions

The described courses have been developed and evolved during six years. The theoretical content of the courses has

been rather stable – still, annual minor additions and updates have been necessary. The laboratory hardware platform and CASE tools have only recently become fully operational. Now the main effort is on improving the illustrative power of practical exercises solved in the laboratory and as a homework. With natural constraints on the complexity of the exercise and time available, this task is more difficult than one would have expected.

Two difficulties have been encountered from the student's side when delivering the courses. First is the unwillingness of students to spend so much time on specification and design. This is an attitude obviously amplified by the previously taught informatics (computer science) courses where examples are taken from data processing. The specification is usually pretty straightforward in data processing problems.

The second difficulty reflects the not very good interdisciplinary knowledge of students. Time constraints on software are often determined by implicit assumptions of, for example, automatic control theory on which the embedded software development problem essentially relies. The first reaction of students is that those theories have nothing to do with software development. In other words the knowledge given in different courses remains fragmented and do not form a solid base for professional evolution of a student.

This observation has lead us to the study of systems integration problem in the context of system's complete lice-cycle [5]. The cumulative effects of:

- approximations (imposed by theory, algorithms, implementation) made at different system life-cycle stages,
- not quite consistent implicit assumptions made in different theories (algorithms, etc) jointly used in the implemented system, and
- not quite consistent behavioural (time) constraints imposed at different life-cycle stages

often become visible only at the software implementation and testing stages, unless special counter-measures are taken. It is to be hoped that the results of this set of courses will slightly improve the interdisciplinary knowledge of systems engineers in the future.

5. Acknowledgements

The development of the abovedescribed three courses and the real-time software engineering laboratory has been partially supported by Estonian Science Foundation (grant 2849), European Union grant CP-94 no.1577, ESPRIT project no.22154, and by Estonian Ministry of Education (grant 0140237s98).

6. References

1. L. Motus and M.G. Rodd "Timing analysis of Real-time Software", Elsevier/Pergamon, 1994
2. M. Bransby "Explosive lessons" IEE Computing and Control Engineering, vol. 9, no. 2, 1998, 57-60
3. A.Burns and A.Wellings "HRT-HOOD: A structured design method for hard real-time Ada systems", Elsevier, 1995
4. B.Selic, G.Gullekson, P.T.Ward "Real-time object-oriented modeling", John Wiley and Sons, 1994
5. L. Motus "On integration of time-critical systems", submitted in July 1998 to IEE Proceedings on Software, special issue on Real-time Systems, (to be published)
6. H. Kopetz "Real-time Systems: Design principles for distributed embedded applications", Kluwer Academic Publishers, 1997
7. Sommerville "Software Engineering" Addison-Wesley, 1992
8. P.T. Ward and S.J. Mellor "Structured Development for Real-time Systems", Prentice-Hall, 1985, vol.1
9. S. R. Schach "Classical and Object-oriented Software Engineering", Irwin, 1996
10. J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, W.Lorensen "Object-Oriented Modeling and Design", Prentice-Hall, 1991
11. UML web site "UML Summary" (and following chapters), www.rational.com/uml
12. T.Naks, L.Motus "Handling timing in a time-critical reasoning system – a case study", IFAC Symposium on Artificial Intelligence in Real-time Control, October 1998, Grand Canyon, AZ, USA (to be published)
13. L.Motus "Timing problems and their handling at system integration", in Artificial Intelligence in Industrial Decision Making, Control and Automation, (eds.) S.G.Tsafestas and H.B.Verbruggen, Kluwer Academic Publ., 1995
14. J.P.Calvez "Embedded Real-time Systems: A specification and design methodology" John Wiley and Sons, series in software engineering practice, 1993
15. "CASE products: comparison and evolution", ButlerBloor Limited, 1992