Teaching Embedded Systems Engineering in a Software-Oriented Computing Degree

João M. Fernandes[1] and Ricardo J. Machado[2]
[1] Dep. Informática / CCTC - [2] Dep. Sistemas de Informação
Universidade do Minho
Braga, Portugal

Abstract **- Traditional software-oriented computing degrees do not include courses on embedded systems design in their syllabus, since in the past embedded applications were seen as small-sized solutions developed without the need of engineering approaches. This reality has dramatically changed in the last decade and nowadays several embedded systems are quite complex. Embedded systems present several idiosyncrasies that make their development more difficult and complex than desktop solutions, namely when considering non-functional requirements, time-related deadlines, or the correctness of the solution.**

To be well prepared for their professions, students of software-oriented computing degrees must acquire skills and competencies in embedded systems engineering. Being able to master high-level programming languages and to develop solutions only for desktop computers means that the students cannot consider numerous opportunities, after graduation.

This paper discusses which topics in embedded software design to include in a second cycle degree on Software Engineering that was structured to consider the Bologna Declaration that is now being used in Europe to recast all university degrees. The syllabus of a 15-ECTS module dedicated to teach the fundamental concepts of embedded systems engineering and embedded software development is also described.

Index Terms - Bologna declaration, Embedded systems, Master degree, Software engineering.

INTRODUCTION

Nowadays, the majority of the computer-based systems are embedded devices (i.e., hardware-software solutions quite different from desktop computers). Additionally, it was estimated or predicted that by 2010, around 90% of the programming code will be developed for embedded devices [1].

However, the dominance of embedded applications in the market was not yet accompanied by a change in computer science education [2, 3]. Currently, computer science education is mainly directed towards the limited needs of the desktop computing, where students are exposed to the classical von Neumann architecture and to high-level programming languages. This has mainly two disadvantages: (1) students do not see the complete set of potential employment opportunities; (2) graduates do not have the adequate skills for working in many application areas.

In a considerable number of universities, the typical content in an introductory course in embedded systems and microprocessor design employs an 8-bit processor, so that students learn how to program in assembly language and how to implement hardware interfacing to the outside world on a prototyping board. A modern course in embedded systems must include a 32-bit processor [4] and several topics related to the methodological development of systems, namely give great attention to the analysis and design activities [5].

Embedded systems present a set of characteristics that make their development extremely hard, especially when it is necessary to handle non-functional requirements, to meet time-related deadlines, or to guarantee the correctness of the solutions. These restrictions are not typically present during the development of desktop applications.

Additionally, small computing platforms, like PDAs (Personal Digital Assistants) or mobile phones, are approaching the capabilities of desktop computers. Overall, this implies that the level of complexity for developing an embedded system is similar to the one needed to develop a desktop application, but the former presents a bigger number of restrictions that make the development of the software to run on those devices much more difficult.

The lack of engineers, with specific skills to develop software for embedded devices, seems to be currently a common and severe problem in all industrial countries and application areas [6-8]. This fact, associated with the growing body of knowledge produced in the field, is creating a growing interest in proposing curricula on embedded software and systems for university-level education. A good overview of some current issues in embedded systems education can be found in eight articles of the August 2005 issue of the ACM Transactions on Embedded Computing Systems [9].

This paper discusses which topics in the embedded computing field must be included in a computing degree that mainly concentrates on software development, so that its graduates can approach the market better skilled to face all the software development problems, namely in the embedded arena. The focus of the discussion is clearly on the software parts of embedded systems, even though design of those systems involves both hardware and software expertise. Our experience in teaching embedded systems in software-oriented

courses has shown us that, generally speaking, students are not motivated for topics in control, electronics, and hardware design; a similar observation is also reported in [10]. Therefore, we believe that the topics to be taught must give the students the opportunity to acquire skills in developing software for embedded devices. This requires them to have the general knowledge of hardware design that is typically part of CS, CE and EE curricula.

EMBEDDED SYSTEMS

Some authors claim that defining embedded systems is not easy. In fact, embedded computing is often defined in a negative form, by stating that it is "not" application software executing on the CPU of a general-purpose or desktop computer [11].

An embedded system can be defined as a combination of computer hardware and software, and perhaps additional mechanical and others parts, developed to perform a specific or dedicated function. Typically, an embedded system is part of some larger system.

I. Categories of Embedded Systems

Embedded applications vary so much in characteristics: two embedded systems may be so different, that any resemblance between both of them is hardly noticed. In fact, the term "embedded" covers a surprisingly diverse spectrum of systems and applications, including simple control systems, such as the controller of a washing machine implemented in a 4-bit micro-controller, but also complex multimedia or telecommunication devices with severe real-time constraints or distributed industrial shop-floor controllers.

Embedded systems can be found in applications with varying requirements and constraints such as [12]:

- Different production units, from a unique system for a specific client to mass market series, implying distinct cost constraints;
- Different types of requirements, from informal to very stringent ones, possibly combining quality issues, such as safety, reliability, real-time, and flexibility;
- Different operations, from short-life to permanent operation;
- Different environmental conditions in terms of radiation, vibrations, and humidity;
- Different application characteristics resulting in static versus dynamic loads, slow to fast speed, compute versus interface intensive tasks;
- Different models of computation, from discrete-event models to continuous time dynamics.

Koopman *et al*. divide, from an application point of view, embedded computing in the following twelve distinct categories [11]: (1) small and single microcontroller applications, (2) control systems, (3) distributed embedded control, (4) system on chip, (5) networking, (6) embedded PCs, (7) critical systems, (8) robotics, (9) computer peripherals, (10) wireless data systems, (11) signal processing, and (12) command and control.

This large number of application areas, with very distinct characteristics, leads us to the inevitable conclusion that embedded systems are quite hard to define and describe. This clearly makes generalizations difficult and complicates the treatment of embedded systems as a field of engineering.

One possible solution is to agree on a division of the embedded field, and consider, for example, four main categories: (1) signal-processing systems, (2) mission critical control systems, (3) distributed control systems, and (4) small consumer electronic devices [13]. For each category, different attributes, such as computing speed, I/O transfer rates, memory size, and development costs apply. Furthermore, distinct models of computation, design patterns and modeling styles are also associated with those categories.

An alternative and simpler classification is proposed in [14]: (1) reactive, (2) interactive and (3) transformational embedded systems. The important message to retain here is that generalization about embedded systems may sometimes only apply to a specific category.

II. Characteristics and Idiosyncrasies of Embedded Systems

Even though some computing scientists consider very naïvely or arrogantly that embedded software is just software that is executed by small computers, the design of this kind of software seems to be tremendously difficult [15]. An evident example of this is the fact that PDAs must support devices, operating systems, and user applications, just as PCs do, but with more severe cost and power constraints [16]. In this section, we argue that embedded software is so diverse from conventional desktop software that new paradigms of computation, methods and techniques, specifically devised for developing it, need to be devised and taught.

The principal role of embedded software is not the transformation of data, but rather the interaction with the physical world, which is apparently the main source of complexity in real-time and embedded software [17]. The role of embedded software is to configure the computer platform in order to meet the physical requirements. Software that interacts with the physical environment, through sensors and actuators, must acquire some properties of the physical world; it takes time to execute, it consumes power, and it does not terminate (unless it fails). This clearly and largely contrasts with the classical notion of software as the realization of mathematical functions as procedures, which map inputs into outputs. In traditional software, the logical correctness of the algorithm is the principal requirement, but this is not sufficient for embedded software [18].

Another major difference is that embedded software is developed to be run on machines that are not just computers, but rather on cars, radars, airplanes, telephones, audio equipments, mobile phones, instruments, robots, digital cameras, toys, security systems, medical devices, network routers, elevators, television sets, printers, scanners, climate control systems, industrial systems, and so on. An embedded system can be defined as an electronic system that uses computers to accomplish some specific task, without being explicitly distinguished as a computer device. The term

"embedded", coined by the US DoD, comes actually from this characteristic, meaning that it is included in a bigger system whose main (the ultimate) function is not computation. This classification scheme excludes, for example, desktop and laptop computers from being embedded systems, since these machines are constructed to explicitly support general-purpose computing. As a consequence of those divergent characteristics, the embedded processors are also quite different from the desktop processors [19].

The behavior of embedded systems is typically restricted by time, even though they may not necessarily have real-time constraints [20]. As stated by Zave: "*Embedded is almost synonymous with real-time*" [21]. The correctness of a real-time system depends not only on the logical results of the computation, but also on the time at which those results are produced [22]. A common misconception is that a real-time system must respond in microseconds, which implies the need to program it in a low-level assembly language. Although some real-time systems do require this type of answer, this is not at all universal. For example, a system for predicting the weather for the next day is a real-time system, since it must give an answer before the day being forecasted starts, but not necessarily in the next second; if this restriction is not fulfilled, the prediction, even if correct, is useless from a practical point of view.

Embedded systems are also typically influenced in their development by other constraints, rather than just time-related ones. Among them one can include: liveness, reactivity, heterogeneity, reliability, and distribution. All these features are essential to guarantee the correctness of an embedded program. In particular, embedded systems are strongly influenced in their design by the characteristics of the underlying computing platform, which includes the computing hardware, an operating system, and eventually a programming framework (such as .NET or EJB) [23]. Thus, designing embedded software without taking into account the hardware requirements is nearly impossible, which implies that, at least currently, "Write Once Run Anywhere" (WORA) and the Model-Driven Architecture (MDA) principles are not easily or directly applicable.

Reactive systems, a class of systems in which embedded systems can be included, have concurrency as their essential feature [24]. Put in other words, development of embedded software requires models of computation that explicitly support concurrency. Although software must not be, at all, executed in sequence, it is almost universally taken for granted that it will run on a von Neumann architecture and thus, in practice, it is conceived as a sequential process. Since concurrency is inherent in all embedded systems, it must be undoubtedly included in every development effort.

With all these important distinctions, one clearly notices that the approaches and methods that are used for traditional software are not suitable for embedded software. Embedded software requires different methods, techniques and models from those used generically for software. The methods used for non-embedded software require, at a minimum, major modifications for embedded software; at a maximum, entirely new abstractions are needed that support physical aspects and ensure robustness [25].

The inadequacy of the traditional methods of software engineering for developing embedded systems appears to be caused by the increasing complexity of the software applications and their real-time and safety requirements [26]. These authors claim that the sequential paradigm, embodied in several programming languages, some object-oriented ones included, is not satisfactory to adequately model embedded software, since this type of software is inherently concurrent.

One of the problems of object-oriented design, in what concerns its applicability for embedded software, is that it emphasizes inheritance and procedural interfaces. Object-oriented methods are good at analyzing and designing information-intensive applications, but are less efficient, sometimes even inadequate, for a large class of embedded systems, namely those that utilize complex architectures to achieve high-performance [27].

According to Lee [25], for embedded software, we need a different approach that allows us to build complex systems by assembling components, but whose focus is concurrency and communication abstractions, and admits time as a major concept. He suggests the term "actor-oriented design" for a refactored software architecture, where the components are not objects, but instead are parameterized actors with ports. Actors provide a uniform abstract representation of concurrent and distributed systems and improve on the sequential limitations of passive objects, allowing them to carry out computation in a concurrent way [28, 29]. Each actor is asynchronous and carries out its activities potentially in parallel with other actors, being thus control distributed among different actors [30]. The ports and the parameters define the interface of an actor. A port represents an interaction with other actors, but does not necessarily have call-return semantics. Its precise semantics depends on the model of computation, but conceptually it just represents communication between components.

M.Sc. Degree in Software Engineering

The M.Sc. degree runs in four semesters and consists of a total of 120 ECTS (European Credit Transfer and Accumulation System). The ECTS system is based on the principle that 60 credits measure the workload of a full-time student during one academic year. The general structure of the degree is presented in table 1. More details, namely the contents of the first year's modules, are available in [31]

The course's first year is divided in two one-year modules. Each 30-ECTS module is composed of five curricular units, with one of them devoted to integrate the module's topics, by adopting a set of laboratorial and practical experiments. The second year includes either a professional or a scientific path. In the former, students enroll, during the first semester, in two 15-ECTS modules and, in the second semester, develop an industrial project. In the latter, students enroll, during the first semester, in just one 15-ECTS module and start a research oriented master dissertation, which they continue in full time

in the second semester. Each 15-ECTS module is composed of three 5-ECTS curricular units.

TABLE I
GENERAL STRUCTURE OF THE M.SC. DEGREE

<table>
<tr><td rowspan="2">first year</td><td colspan="2">first semester</td><td colspan="2">second semester</td></tr>
<tr><td colspan="4">Module on Analysis and Design (30 ECTS)</td></tr>
<tr><td></td><td colspan="4">Module on Quality and Management (30 ECTS)</td></tr>
<tr><td rowspan="4">second year</td><td colspan="2">Professional path</td><td colspan="2">Scientific path</td></tr>
<tr><td>first sem.</td><td>second sem.</td><td>first sem.</td><td>second sem.</td></tr>
<tr><td>1st Module (15 ECTS)</td><td rowspan="2">Industrial Project (30 ECTS)</td><td>Module (15 ECTS)</td><td rowspan="2"></td></tr>
<tr><td>2nd Module (15 ECTS)</td><td colspan="2">Master Dissertation (45 ECTS)</td></tr>
</table>

Each 15- or 30-ECTS module must obey the following issues:

- There is a unique assessment (i.e., it exists only one unique mark for the module, even though it is composed of five internal curricular units).
- The module is supported by a set of teachers, being one of them the coordinator.
- Each laboratory must cover a major part of the subjects included in the other four internal curricular units. All teachers of the module must accompany it, so that they can adjust their materials to better support the execution of the project.
- The project must develop not only the technical competences of the students, but also planning and management skills.

The 15-ECTS modules must address one application area of the main topics of the master course. Each 15-ECTS module must incorporate the methodological approaches covered by the first year modules. Examples of these 15-ECTS modules are: (1) data–oriented enterprise applications; (2) pervasive software and ubiquitous services; (3) real–time, embedded and critical systems; (4) industrial informatics and automation. Students must choose either one or two 15-ECTS modules, depending on the chosen path (professional or scientific). In the next section, we present and discuss a possible syllabus for one of those modules in Embedded Systems Engineering.

MODULE IN EMBEDDED SYSTEMS ENGINEERING

We propose embedded systems engineering topics, with an emphasis on the software parts, to be included in a 15-ECTS module, for both the professional and the scientific paths of the M.Sc. degree. This 15-ECTS workload seems to be sufficient to guarantee that students acquire the needed skills to develop embedded software at a professional level.

The emphasis of the module is on embedded software development, with hardware-related concepts being only superficially addressed whenever necessary. Thus, it is important that students become aware of the differences between desktop (or traditional) software and embedded software.

Internally, the module is divided into three courses that will run in parallel. These three courses are aimed to provide the students with:

1. The general overview of the principles and characteristics of embedded systems, as well as a solid foundation of all the techniques specifically needed to deal with the design of embedded software solutions. This course is entitled “Engineering of Embedded Systems”.
2. The technological capability of programming embedded solutions by adopting solid architectural models and patterns and efficient frameworks and development environments. This course is entitled “Embedded Software Programming”.
3. The systematic approach of creating decision support models to analyze conflicting attributes and to help the adoption of design trade-offs, within a global quality assessment of architectural and behavioral solutions. This course is entitled “Modeling and Simulation of Reactive Systems”.

After completing this 15-ECTS module in Embedded Systems Engineering, students will be able:

- to make a precise description of an embedded software system;
- to idealize different solutions to solve the same problem and evaluate (justifying) which one is the best with respect to its design quality;
- to configure best practices of software referential design processes to support the development of embedded software solutions;
- to use development tools to make the previous tasks more efficient;
- to identify the technologies that can be used to realize the specified embedded software solutions.

We present in the next subsections these courses, and for each one we introduce its topics and discuss its aims.

I. Engineering of Embedded Systems

This course aims to provide students with a general appreciation of the current trends and application domains of embedded (hardware/software) systems. It is intended to characterize the foundations of embedded systems engineering as a global approach to the development of computer–based solutions with strong non-functional constrictions to be integrated in non-computing environments. In particular, students who successfully complete the course understand the interplay between the different requirements in a complex embedded software design, involving issues such as concurrency, reliability, and timing constraints.

The course contents is organized as follows:

1. Introduction (principles and characteristics of embedded systems).
2. Target architectures (von Neumann architecture, Harvard architecture, DSPs, ASIPs, reconfigurable technologies, custom computing machines).
3. Interfacing techniques (pooling and interrupts).
4. Scheduling strategies (rate-monotonic and earliest deadline first).
5. Models of computation (data-flow, control, synchronous, reactive).
6. Quality attributes, focusing on non-functional ones (performance, real-time, dependability, inter-operability, portability, etc).
7. Application domains (telecommunications, industrial informatics, home automation).

II. Embedded Software Programming

This course aims to provide students with skills on programming embedded systems. Students who successfully complete the course master the principles underlying the development of software for embedded systems, by adopting solid methodic approaches. They also have the ability to compare different models of computation (especially data-flow and object-oriented) and apply the most adequate ones to the systems under consideration.

The course contents is organized as follows:

1. Introduction (fundamental concepts and principles of software design for embedded systems).
2. Embedded software architectures (reference models, catalogue of patterns).
3. Behavioral composition and synthesis (operational semantics, transactional and state-orientation interpretation).
4. Programming languages for embedded systems (Esterel, Ada, Lustre, C#)
5. Data-flow programming (LabVIEW framework).
6. Object oriented programming (.NET, Java).
7. Code compilation and optimization techniques.

III. Modeling and Simulation of Reactive Systems

This course intends to put students in contact with topics related to modeling and simulation of behavior-intensive software systems, which need to address issues like concurrency, resource sharing, synchronization, and performance. Students who successfully complete the course are able to produce models that can be simulated, animated, and analyzed by tool-supported processes, in order to reason about the behavioral properties of the system under development, taken also into account how the environment affects it.

The course contents is organized as follows:

1. Introduction to quality assessment (properties estimation, requirements validation).
2. Conflicting attributes and design trade-offs (design space exploration, design decisions, prioritization).
3. Evaluation of architectural solutions (performance and dependability analysis).
4. Modeling and analysis of behavior (statecharts, Petri nets, data-flow diagrams).
5. Simulation, animation and performance evaluation (Arena and CPN Tools).

CONCLUSIONS

This paper intends to propose a 15-ECTS module on Embedded Systems Engineering, integrated in a 2-year M.Sc. degree in Software Engineering. This proposal is made under the evidence that software engineers with skills in embedded systems engineering are nowadays needed (and will be even more in the future) by the industry to design and build complex embedded computing systems.

The broad competences needed in embedded computing systems demand careful integration in current degrees, namely to not unbalance the degree in other important topics.

We hope that the proposal made in this paper helps in promoting a discussion on the computer science, software engineering, and education communities about the role and position that Embedded Systems Engineering deserves in Computing degrees in general and more specifically in Software Engineering ones.

ACKNOWLEDGMENT

The authors' views on educational issues were influenced by internal discussions at their institution about possible evolution of its software engineering program. The authors are indebted to their colleagues and other discussion partners who helped to shape their perspectives. However, the views presented in this paper carry no institutional endorsement.

REFERENCES

[1] Hartenstein R, "The Digital Divide of Computing", *1st Conference on Computing Frontiers (CF '04)*, ACM Press, 2004, pages 357–62. DOI: 10.1145/977091.977144.

[2] Mordechai B-A, "The Invisible Programmers", *Conference on Methods, Materials and Tools for Programming Education (MMT '06)*. Tampere Polytechnic - University of Applied Sciences Publications, 2006.

[3] Cardoso JMP, "New Challenges in Computer Science Education", *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*, ACM Press, 2005, pp. 203–7. DOI: 10.1145/1151954.1067502.

[4] Nooshabadi S, Garside J, "Modernization of Teaching in Embedded Systems Design? – An International Collaborative Project", *IEEE Transactions on Education* 49(2):254–262, 2006. DOI: 10.1109/TE.2006.872402.

[5] Wolf W, Madsen J, "Embedded Systems Education for the Future", *Proceedings of the IEEE* 88(1):23–30, 2000. DOI: 10.1109/5.811598.

[6] Pak S, Rho E, Chang J, Kim MH, "Demand-driven Curriculum for Embedded System Software in Korea", *ACM SIGBED Review* 2(4):15–9, 2005. DOI: 10.1145/1121812.1121816.

[7] Yamamoto M, Tomiyama H, Takada H, Agusa K, Mase K, Kawaguchi N, Honda S, Kaneko N, "NEXCESS: Nagoya University Extension Courses for Embedded Software Specialists", *ACM SIGBED Review* 2(4):20–4, 2005. DOI: 10.1145/1121812.1121817.

[8] Broy M, "Challenges in Automotive Software Engineering", *28th International Conference on Software Engineering (ICSE 2006)*, ACM, 2006, pp. 33–42. DOI: 10.1145/1134285.1134292.

[9] Burns A, Sangiovanni-Vincentelli A, "Editorial", *ACM Transactions on Embedded Computing Systems* 4(3):469–71, 2005. DOI: 10.1145/1086519.1086520.

[10] Jogesh K. Muppala, Bringing embedded software closer to computer science students, *ACM SIGBED Review* 4(1): 11–16, 2007. DOI: 10.1145/1217809.1217812.

[11] Koopman P, Choset H, Gandhi R, Krogh B, Marculescu D, Narasimhan P, Paul JM, Rajkumar R, Siewiorek D, Smailagic A, Steenkiste P, Thomas DE, Wang C, "Undergraduate Embedded System Education at Carnegie Mellon", *ACM Transactions on Embedded Computing Systems* 4(3):500–28, 2005. DOI: 10.1145/1086519.1086522.

[12] Grimheden M, Törngren M, "What is Embedded Systems and How Should It Be Taught? — Results from a Didactic Analysis", *ACM Transactions on Embedded Computing Systems* 4(3):633–51, 2005. DOI: .

[13] Koopman P, "Embedded System Design Issues (The Rest of the Story)", *IEEE International Conference on Computer Design (ICCD '96)*, 1996, pp. 310–7. DOI: .

[14] Edwards S, Lavagno L, Lee EA, Sangiovanni-Vincentelli A, "Design of Embedded Systems: Formal Models, Validation, and Synthesis. *Proceedings of the IEEE* 85(3):366–90, 1997. DOI: 10.1145/1086519.1086528.

[15] Wirth N, "Embedded Systems and Real-Time Programming". *1st International Workshop on Embedded Software (EMSOFT 2001)*, Springer, 2001, pp. 486–92.

[16] Wolf W, "What is Embedded Computing?", *IEEE Computer*, 35(1):136–7, 2002. DOI: 10.1109/2.976929.

[17] Selic B, "Turning Clockwise: Using UML in the Real-Time Domain", *Communications of the ACM* 42(10):46–54, 1999. DOI: 10.1145/317665.317675.

[18] Sztipanovits J, Karsai G, "Embedded Software: Challenges and Opportunities. *1st International Workshop on Embedded Software (EMSOFT 2001)*, Springer, 2001, pp. 403–15.

[19] Conte TM, "Choosing the Brain(s) of an Embedded System", *IEEE Computer* 35(7):106–7, 2002. DOI: 10.1109/MC.2002.1016908.

[20] Stankovic JA, "Real-Time and Embedded Systems", *ACM Computing Surveys* 28(1):205–8, 1996. DOI: 10.1145/234313.234400.

[21] Zave P, "An Operational Approach to Requirements Specification for Embedded Systems", *IEEE Transactions on Software Engineering* SE-8(3):250–69, 1982.

[22] Stankovic JA, "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems", *IEEE Computer* 21(10):10–9, 1988. DOI: 10.1109/2.7053.

[23] Selic B, "Physical Programming: Beyond Mere Logic". Embedded Software, *2nd International Workshop on Embedded Software (EMSOFT 2002)*, Springer, 2002, pp. 399–406.

[24] Manna Z, Pnueli A, "The Temporal Logic of Reactive and Concurrent Systems: Specification". Springer, 1992.

[25] Lee EA, "Embedded Software", *Advances in Computers* 56, 2002.

[26] Balarin F, Lavagno L, Passerone C, Watanabe Y, "Processes, Interfaces and Platforms. Embedded Software Modeling in Metropolis", *2nd International Workshop on Embedded Software (EMSOFT 2002)*, Springer, 2002, pp. 407–21.

[27] Bhatt D, Shackleton J, "A Design Notation and Toolset for High-Performance Embedded Systems Development", Lectures on Embedded Systems, European Educational Forum School on Embedded Systems, Springer, 1998, pp. 249–67.

[28] Hewitt C, "Viewing Control Structures as Patterns of Passing Messages", *Journal of Artificial Intelligence* 8(3):323–64, 1977.

[29] Agha G, *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, 1986.

[30] Ren S, Agha G, "A Modular Approach for Programming Embedded Systems", *Lectures on Embedded Systems - European Educational Forum School on Embedded Systems*, Springer, 1998, pp. 170–207.

[31] Fernandes JM, Machado RJ, "A Two-Year Software Engineering M.Sc. Degree designed under the Bologna Declaration Principles", *1st International Conference on Software Engineering Advances (ICSEA 2006)*, IEEE Computer Society Press, 2006. DOI: 10.1109/ICSEA.2006.13.