USING RHAPSODY® FOR THE ANALYSIS, DESIGN AND MODELING OF BIOMEDICAL EMBEDDED SYSTEMS. FROM REQUIREMENTS TO IMPLEMENTATION

P. CABEZAS, S. ARRIZABALAGA, J. MELÉNDEZ, I. SANCHO, J. LEGARDA
Electronics and Communications Department. Engineering School of the University of Navarra (TECNUN). University of Navarra. Spain.
{pcabezas, sarrizabalaga, jmelendez, isancho, jlegarda}@tecnun.es

This article describes the educational methodology related to the 'Software and Protocols Engineering: Software for Medical Equipment' subject, belonging to the Engineering School of the University of Navarra (TECNUN). The integral design of embedded systems is successfully overcome through the use of techniques from Software Engineering and Systems Engineering, and the novel use of the Rhapsody® tool, applied to the biomedical environment. In this sense, the organization and contents of the subject are shown, as well as an example of a project carried out by some students.

Keywords: Software engineering, Analysis, Design, MDD, Rhapsody, Biomedical Embedded Systems

1. Introduction

The *Software* concept refers to a fundamental part of certain families of products, therefore being a key issue in their success. However, it is important to stress that, in the development of a system, it is not all about programming; *design* plays an important role, and it is necessary to count with a high-level vision over the whole system. Furthermore, students must see that there are several disciplines involved in project management and development, especially in that kind of systems that are composed by *hardware* and *software*, and that go far beyond mere programming. In this sense requirements, analysis and design are seen and recognized as key parts in *systems development*.

At this point it is complex to show, beyond pure theory and drawings in paper, that a manageable traceability exists between requirements, analysis documents and system design documents. If they are presented as disjointed docs, like simple artifacts that are abandoned when proceeding with the next phase of the system, their importance appears seriously doubtful for students. Functional requirements, non-functional requirements, use case diagrams, implementation models… If everything is done by pencil, how do I know that I am going wrong? Furthermore, what does the hardware have to do with the system? Is it a mere support, or can it be captured in the documents and their traceability, for example with the requirements?

Given the fact that the software is not enough to build a hybrid system by itself, *Software Engineering* techniques must be helped by the *Systems Engineering* discipline, providing different and complementary views at the time of describing and assembling a system composed by both software and hardware.

The lecture called '*Software and Protocols Engineering: Software for Medical Equipment*' emphasizes the development (and not only programming) of complex systems, providing a needed and

multidisciplinary vision regarding to the analysis and design of hybrid systems, composed by hardware and software, applied to the *Biomedical* environment. The use of a rich application like IBM Rhapsody®, based on the Model-Driven Engineering (MDE) approach, helps at the time of capturing all these theoretical ideas for the students in an easy, fashionable and appealing way.

2. The Traceability issue: disjointed Artifacts and Knowledge.

The Analysis, Design and Implementation disciplines need different lectures for their contents, in order to consolidate well their conceptual basis. However, either because there is no time for teaching or because they must be individually taught, there is no technical media in the university environment to express, through a informatics tool, all what is mentioned at class regarding to these subjects.

Functional requirements and non-functional requirements and use case diagrams from the Analysis phase, class and sequence diagrams from the Design phase, different files and solutions from the Implementation phase… All these elements are usually known as *artifacts* and their connection is not trivial at all, when talking about connection media different to paper. Requirements get fulfilled by use cases, while use cases are included and supported by sequence diagrams, but… What happens with the implementation? And furthermore, where can we locate the *hardware* in this overall view?

The MDE methodology [1,2] provides clearer and sharper images to capture the previous contents, in a graphical and closer to the human-being thinking way. Of course, this can be extended to all the different phases in systems and projects development.

Numerous tools can be found, in the current state of the art, that allow the design of systems through the MDE approach, even generating executable code from the models. The majority of them are based on the combination of MDE and the Unified Modeling Language (UML) [3], creating the Model-Driven Architecture (MDA) concept [4]. As UML defines a formal notation for software systems modelling, it can be considered as the common language for software engineers and, therefore, its learning gets really useful for the students.

However, and as it has been told in Section 1, UML and Software Engineering techniques can be hard to apply when talking about hybrid systems, this is, when traceability and concepts like *hardware* come into scene. Regarding this problem, tools like Rhapsody® help with personalized (and still formal) notations, with some bits coming from the software realm and other bits from Systems Engineering domain, also including notations like SysML [5].

3. Modeling Systems from scratch with Rhapsody®

Rhapsody® is a tool from the large company IBM, which provides a MDD environment for real-time or embedded systems engineering, software development, and test based on UML and SysML. In the real (non-educational) world, it helps enable embedded systems engineers and software developers to improve productivity, quality, and communication by abstracting complex designs graphically, by automating the software development process, and assisting in finding defects early through continual testing — to aid in reducing costs [6,7].

This tool is composed by different versions and modules, providing capabilities such as:

- An integrated requirements and modeling environment, using industry standard SysML or UML diagrams
- An industry standard based UML and SysML modeling environment, which helps improving team communication while maintaining consistency across different views

- Full lifecycle traceability and analysis from requirements to design, with customizable automatic documentation capabilities
- Visualize the applications using industry standard UML diagrams
- Design object oriented-based or functional applications, through full behavioral code generation for C, C++, Java and Ada
- Visual simulation, which brings diagrams to life for design level debugging and early validation. It executes the model to help validate architecture and behavior
- Operating systems supported: Linux, Windows

Rhapsody is a powerful environment for rapid prototyping and model execution, which helps validating requirements, architecture, design, and behavior early in the lifecycle. It also helps in the delivering of products that meet requirements, which sounds too close to the real (industrial) world, but provides a sharper idea to students at this point.

4. Objectives, Methodology and Contents of the subject

As it has been previously commented, the analysis, design and development of embedded systems forms part of the contents of the optional subject called '*Software and Protocols Engineering: Software for Medical Equipment*'. It is offered to the students of the Engineering School of the University of Navarra (TECNUN), that have basic knowledge in programming (C, C++ and some them in Java), microcontrollers, protocols and networks. Therefore, it is a real challenge to correctly introduce them to the *embedded systems modelling* realm.

4.1. Initial approach to software and systems development. Objectives

This subject aims at providing a focus to software development from Information Management Systems (IMS), as a transportation vehicle for training and lecturing in requirements extraction, analysis and design of systems. Later on, students will have the knowledge required to go deep into embedded systems modelling.

This subject is oriented to the acquisition of the knowledge required by an engineer for the analysis the requirements of the client, their specification in case they are not correctly defined, their capture through the current modelling standard (UML) and their interpretation if these requirements come from another person of the workgroup. Furthermore, the student will get familiarized with the Rhapsody tool. This tool provides a MMD-based development environment for (real time or conventional) embedded systems. Rhapsody allows the engineer to capture the design through standard graphical models, generating code from them. These designs can be simulated, in order to guarantee the successful behavior and to verify the fulfilment of the functional specifications, considerably reducing development times. One of the strong points of this tool is that is may avoid a direct contact with programming languages, abstracting that complexity through models.

The student will be capable of interacting with the design and execution of a software project, together with the rest of disciplines that take part in it. The development team may be formed by informatics engineers, with expertise in software development and programming languages, and by programmers that will latterly write the source code. It is a structure similar to that composed by architects, foreman builders and building contractors; in the end, all must use the same language to solve the project.

4.2. Methodology

The subject has two parts: theory and practice. Theory is taught through master classes, regarding to the topics in the book of the subject, also containing expositions with slides and practical exercises, to get familiarized with both the UML notation and the concepts from the requirements, analysis and design fields. In the practical part, the students learn how to use a design and development tool (Rhapsody®) which is a current leader in its application field, in the laboratory. Each student uses its own computer, directed by the professor through practical explanations in the projector.

A final practice will be carried out by groups (three or four people), in which the students have to apply the knowledge gained in the theoretical and practical parts, using the Rhapsody tool. The results of these practices will be publicly expounded, simulating a presentation of results from a group of engineers to the client that asked for the product initially. The percentage related to the final mark is of the sixty per cent.

The final exam consists of an exercise in which the student will have to prove that he/she has gained the theoretical and practical concepts of the subject. It is done on paper, without computer, and is the fourty per cent of the final mark.

4.3. Distribution and Contents

The total dedication of the subject is 122 hours, which is equivalent to 4.5 ECTS credits (or 6 conventional credits). The distribution of the workload, in terms of *student working hours*, could be approximated to the following numbers:

- Theory – master classes (18 hours)
- Rhapsody training (16 hours)
- Group practice (20 hours)
- Exposition (4 hours)
- Exam (4 hours)
- Personal work (40 hours) and Study (20 hours)

These dedication numbers are related to the following contents of the subject:

I. Introduction to Embedded Systems
II. Introduction to Systems Requirements: obtention, analysis, negotiation and validation
III. Introduction to the Analysis and Design of Software Systems
IV. Object Oriented Programming (OOP)
V. The Unified Modelling Language (UML)
VI. Model-oriented design and development with Rhapsody
    - Introduction to the environment
    - Integral development of a embedded system

With the previously presented information and the profiles of the students in mind, a practical example is shown in the next section.

5. Practical example

The example presented in this section is related to a '*Monitoring system for premature babies*'. This system aims to control the state of premature babies, obtaining temperature and ECG measurements by two different types of sensors and sending them to the same microcontroller. An alarm is set in the main computer, in case a strange behavior is detected, in order to finally assist them if necessary.

5.1. Introduction

A premature baby needs special care in hospitals. He/She has to be in an incubator during a determinate time according to him/her particular situation. The incubator protects the baby from germs, provides the baby with a good temperature environment and controls the behavior of the baby.

The responsibility of controlling the baby resides on the microcontroller incorporated in the incubator. There is a set of sensors connected to the baby which sends the measurements taken to the microcontroller.

The connection between the sensors and the microcontroller is flexible. Bluetooth, WI-Fi or Zigbee implementations are allowed.

The measurements are the following ones:

- Temperature
- ECG

The microcontroller processes all the measurements and detects strange behaviors checking if the measurement is within a default range of values. The normal ranges of values considered are:

- Temperature: between 35 ºC and 37ºC
- ECG: between 100 beats/min and 120 beats/min

When the measurement is not within the range, the microcontroller activates an alarm in the main computer and sends the wrong measurement and the personal information of the baby as well. The main computer shows the wrong measurement and the personal information of the baby so that the premature baby is attended immediately by a doctor.

The communication between microcontroller and main computer is flexible. Bluetooth, WI-Fi, Zigbee, USB, GPRS are allowed.

5.2. Requirements. Classification

There are there different types of requirements: functional requirements, non functional requirements and flexibility requirements.

- Functional requirements: These requirements describe what the system does.
    1. Each sensor will send the measured value to the microcontroller.
    2. The microcontroller will receive all the measurements from the sensors.
    3. The microcontroller will process the received measurements.
    4. The microcontroller will send the measurement to the main computer and activate the alarm in the main computer in case a strange behavior is detected.
    5. The main computer will receive the wrong measurement from the microcontroller.
    6. The main computer will show an advertisement with the following information:
        - Name and surname of the baby in trouble.
        - Name and value of the wrong measurement.
        - Non functional requirements: These requirements describe technical aspects related to the requirements described above.
    7. The temperature sensor will send every three seconds the information to the microcontroller
    8. The ECG sensor will send every one second the information to the microcontroller.

b) Flexibility requirements:

1. The connection between the sensors and the microcontroller must be done with ports.
2. The connection between the microcontroller and the main computer must be done with ports.

5.2. System Analysis

The analysis of system requirements is shown at this point, specifying use cases and their relations with the different actors of the system.

Regarding the design of the system, the Use Case Diagram of the system (this is, the high-level view of the system architecture in terms of requirements analysis) can be seen in Figure 1. It shows the use cases detected for this monitoring system.

In this project, four actors and seven use cases have been considered.

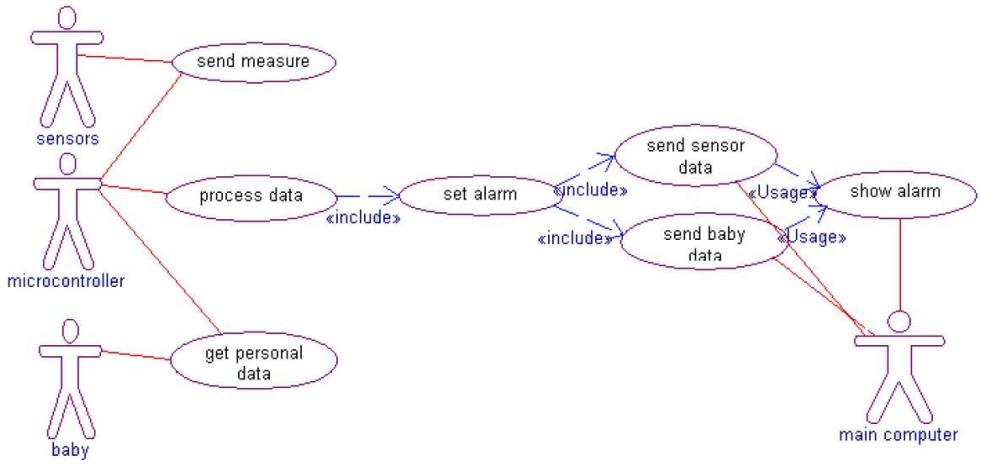

Figure 1. Use Case Diagram (Object Model Diagram)

- Send measure. It describes how sensors send measures to the microcontroller
- Process data. It describes how microcontroller processes the data and checks if a measure is wrong. This use case includes the following use case:
    - Set alarm. It describes how microcontroller generates an alarm. The alarm is set when a wrong measurement is detected while processing the data. This use case includes the following use cases as well:
        - Send sensor data It describes how a wrong measurement is sent to the main computer. This use case uses the next use case:
            - Show alarm. It describes how the main computer prints the wrong measurement.
        - Send baby data It describes how baby data is sent to the main computer in case a wrong measure is detected in the microcontroller. This use case uses the next use case:
            - Show alarm. It describes how the main computer prints the data of the baby.
- Get personal data. It describes how the microcontroller takes personal data of the baby.

With regard to the actors drawn in Figure 1:

- Sensors. This actor interacts with the system using interfaces with:
    - *Send measure*. It is responsible of sending the measurement.
- Microcontroller. This actor interacts with the system using interfaces with:
    - *Send measure*. It is responsible of receiving the measurement.
    - *Process data*. It is responsible of processing and checking the data.
    - *Set alarm*. It is responsible of setting the alarm by sending the wrong measurement (*Send sensor data* use case) and sending the personal baby information (*Send baby* data use case)
    - *Get personal data*. It is responsible of getting personal data of the baby.
- Main Computer. This actor interacts with the system using interfaces with:
    - *Send sensor data*. It is responsible of receiving sensor data.
    - *Send baby data*. It is responsible of receiving the information of the baby whose measurement is wrong.
    - *Show alarm*. It is responsible of printing the wrong measurement and the information of the baby whose measurement is wrong.
- Baby. This actor interacts with the system through:
    - *Get personal data*. It is responsible of allowing microcontroller to obtain its data.

5.2. System Design

Regarding the design of the system, the Object Model Diagram of the system (this is, the high-level view of the architecture of the system in terms of the Rhapsody tool) can be seen in Figure 2.

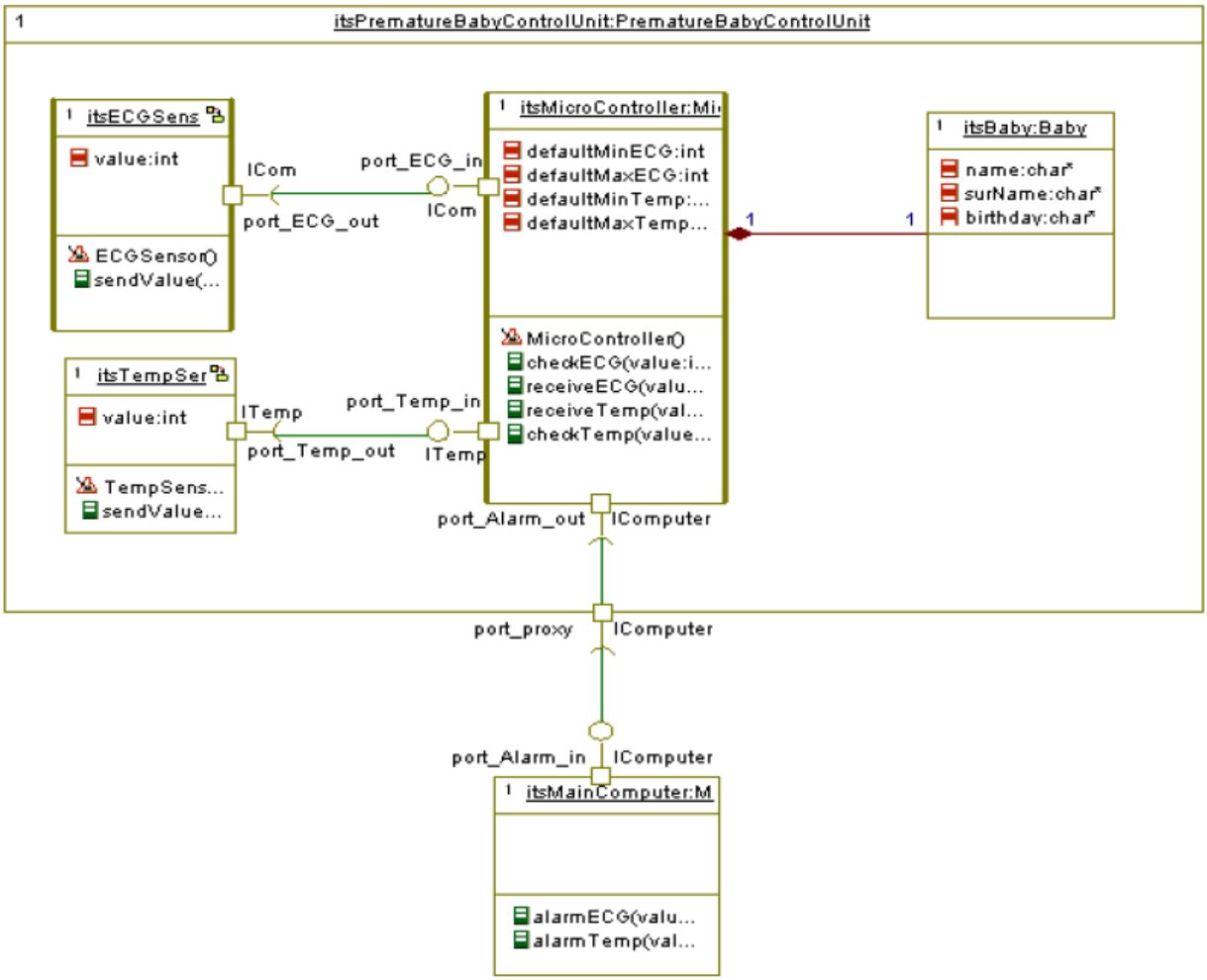

Figure 2. Class Diagram of the System (Object Model Diagram)

These models can be executed through *animated statecharts*, which give a real feel to the different people of the team working on the same system (see Figure 3). These statecharts run when the program is executed, and represent part of its internal functionality. In the example shown above, there is only one state named idle, and every three seconds the *sendValue()* function is called.

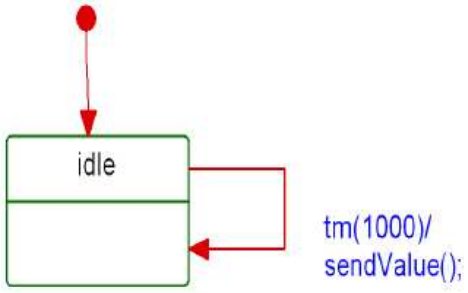

Figure 3. Simple statechart example: 'Statechart of the ECG Sensor'

Sequence diagrams can easily be created to test the system functionality, or from the execution of the system itself. Some examples can be seen in Figure 4and Figure 5.

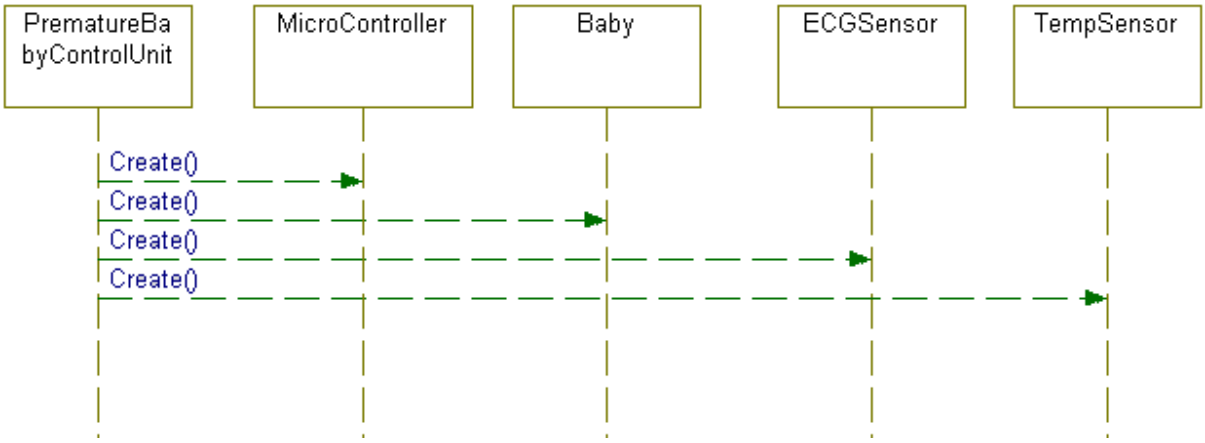

Figure 4. Sequence diagram of the creation of the composite class

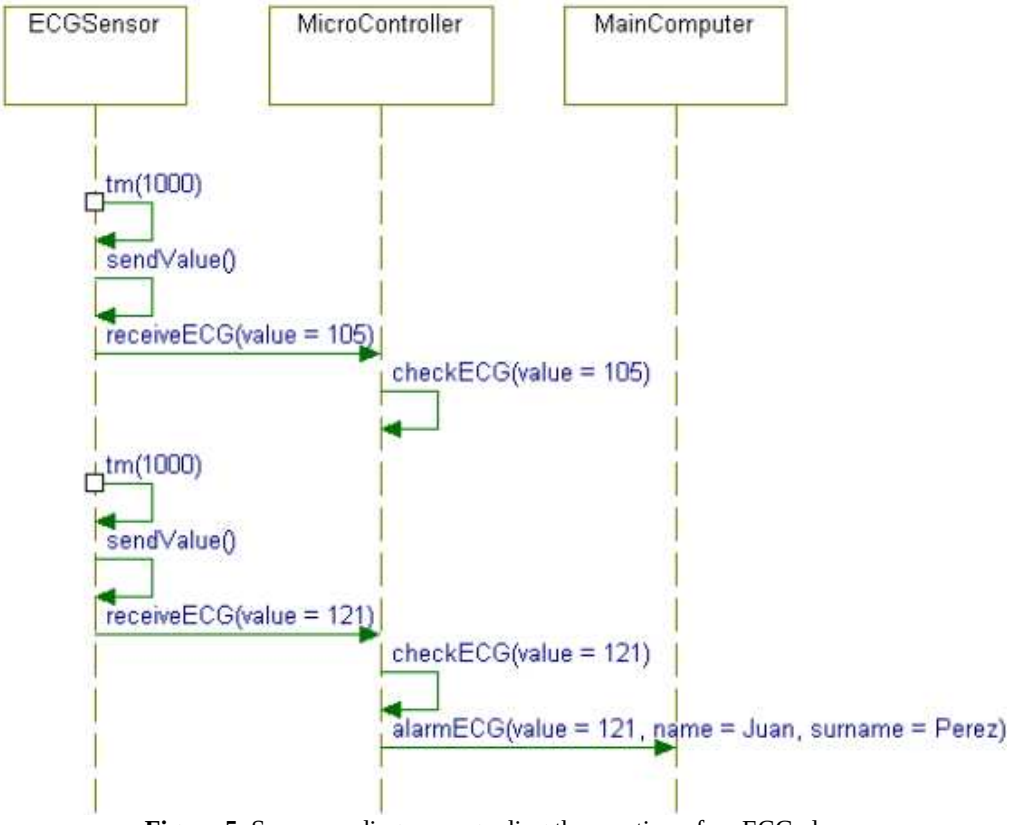

Figure 5. Sequence diagram regarding the creation of an ECG alarm

6. Conclusion

The methodological and technical solution applied to the subject called '*Software and Protocols Engineering: Software for Medical Equipment*' has been shown, providing theoretical and practical contents to students with limited knowledge in Software Engineering, Systems Engineering and programming languages.

Through the abstraction of complexity thank to the use of models, and as it can be seen in the example shown in Section 5, the students get a real and general vision from the subject, regarding to the analysis and design of a system composed by hardware and software, with its functionalities applied to the biomedical environment.

References

[1] Douglas C. Schmidt. *Guest Editor's Introduction: Model-Driven Engineering*, Computer, vol. 39, 2, 25-31 (2006)

[2] Stuart Kent. *Model Driven Engineering*, Integrated Formal Methods, vol. 2335, 286-298, Springer (2002)

[3] UML Resource Page. Web page: http://ww.uml.org

[4] AG Kleppe, J Warmer, W Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley (2003)

[5] SML Partners. *Systems Modeling Language (SysML) Specification*. Available at omg.org

[6] E Gery, D Harel, E Palachi. *Rhapsody: A complete life-cycle model-based development system*. Springer (2002)

[7] ML Crane, J Dingel. *UML vs. classical vs. Rhapsody statecharts: Not all models are created equal*. Software and Systems Modeling, Springer (2007)