Draft of the paper presented on 7th IFAC Symposium on Advances in Control Education, June 21-23, 2006, Madrid, SPAIN, session ThA03

BRIDGING THE GAP BETWEEN COMPUTER SCIENCE AND TECHNOLOGY

Zbigniew Mrozek

Cracow University of Technology (Politechnika Krakowska)
24 Warszawska Str., PL 31-155 KRAKOW, Poland,
pemrozek@cyf-kr.edu.pl, http://www.cyf-kr.edu.pl/~pemrozek

Abstract: Most of students do not have enough experience of working in interdisciplinary team. This paper shows how some interdisciplinary control problems can be stated and solved by students with some knowledge of modern CASE tools. An example of automotive ABS (Anti-lock Braking System) is given to explain graphical approach with RUP and UML in design of a control system. Copyright © 2006 IFAC

Keywords: control, mechatronics, requirements analysis, modelling, validation, RUP, UML.

1. INTRODUCTION

Design is a part of life cycle of product (figure 1). Students should understand how to minimise risk of the control project failure, risk of project delay or over-budgeting. The paper describes usability of:

- RUP (*rational unified process*) methodology
- CASE (*computer aided system engineering*) tools based on UML (*unified modelling language*)

The above tools were invented for use in area of software engineering, but they may be efficiently used in design of any control systems.

2. TEACHING THE DESIGN METHODOLOGY

Bruegge and Dutoit (1999, 2004) in their excellent book introduce UML as a base to object oriented software engineering. This approach is not limited to pure software engineering and may be successfully used in design of any system (Mrozek 2003- 2005). UML models may describe architecture and behaviour of the future system on high level of abstraction, including its software and hardware subsystems of controller.

UML is independent of programming languages and independent of physical nature of a hardware used.

Student may explore graphical UML models to verify his ideas, before source or binary code is prepared, and before physical prototype of the hardware is build. UML models form a good base for future detailed design and prototyping of the control system. It is also an important step towards concurrent design and early integration of subsystems of different nature, leading to solutions where control software, electronics and precision mechanics is integrated into one mechatronic product or device.

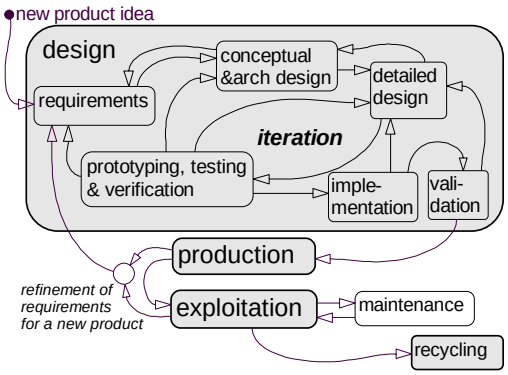

Figure 1. Design is a part of life cycle of a product

2.1. The design process

A modern design process may be presented as a sequence of (fig. 1):

early design phase, which include elicitation of requirements (*inception*) and conceptual design (*elaboration*). Off-line simulation may be used for testing the designed models,

building (*construction*) phase, which starts with detailed design of subsystems. Then simulation and prototyping is done, as well as implementation and testing of hardware and software,

deployment (*transition*) of achieved solution.

2.2. Rational Unified Process (RUP)

Planning, management and monitoring of team work may be done using Rational Unified Process (2001, 2004) or any other effective methodology. RUP is the software engineering methodology based on best practises, learned from thousands of successful projects. Some parts of this methodology are useful in design of control systems.

There are four development phases defined in RUP: *inception*, *elaboration*, *construction* and *transition*. There are also well defined conditions *(milestones)* to be fulfilled, before new phase of design is started. **Inception** corresponds well with requirements elicitation of early design phase. **Elaboration** is focussed on analysis of the problem domain, establishing future system architecture and elimination of the main risk of project failure. It extends conceptual design phase. **Construction** phase may include prototyping, detailed design and implementation. **Transition** means production and deployment of the product.

Elicitation of requirements and business modelling (with *use cases*) is mainly done during *inception* phase, and on the other hand, analysis and design of models is the main job during *elaboration* phase of design. Students should find and justify what is the main (taking most of effort) activity during each phase of design.

Preparing own projects, students will realise that design is not strictly sequential but rather an iterative effort, with many small design steps (micro-steps). After each step, the affected part of system is tested against requirements. Very often result of a step (or micro-step) is not satisfactory and the designer decides to return back to one of previous steps. Then he repeats the affected part of the design. The sequence of testing and redesigning (one or more steps) may be repeated many times in a loop (*iteration* on Fig. 1 and Fig. 2) – until satisfactory result is achieved.

2.3. The RUP methodology guidelines

In author's opinion, *best practices*, *spirit* and *essentials* of RUP (Probasco, 2003) may be concluded in few guidelines:

- Identify major risks and attack them early, or they'll attack you
- Model the system visually
- Develop interactively, make quality a way of life, not an afterthought
- Ensure that your deliver value to your customer

The above guidelines fit as well to software engineering as to designing of any control system.

2.4. Using Unified Modelling Language (UML) for modelling and inter-team communication

Inventors of RUP were motivated to create notation for their unified methodology (Jacobson at all, 1999). The result was UML, a language for specifying, visualising, constructing and documenting the artefacts. From version 1.1, the UML language is non-proprietary and open to all. It is maintained by a non-profit organisation: *Object Management Group* (OMG, 2006)

Many attempts were done to extend the software engineering methodology and UML in areas beyond informatics. McLaughlin and Moore (1998) were probably the first to describe real time control process (conveyor belt transport subsystem) using UML-like class diagram. Now UML diagrams are used for preparing different models on high level of abstraction (Mrozek, 2001-2004; Mrozek et al, 2002)

UML notation helps to describe and to understand functions, services and activities of any system, regardless of its physical nature. This is very useful during all design phases of control systems. Models are essential for communication between members of interdisciplinary team of designers.

Designing UML diagrams on computer screen is supported with CASE (*computer aided software engineering*) software tools. Best-known packages are Rational Rose (2004), Rhapsody (2004), Visual Paradigm for UML (2006) and Real-time Studio (2004). Some CASE tools offer simulation and animation of UML models. Simulation will be more realistic, if virtual console with animated dials and gauges is shown on computer screen. State diagram may be build and animated with MATLAB/Stateflow. This helps to see behaviour of the system under design. Moreover *C* or binary code for the state diagram functionality may be automatically generated with MATLAB/RTW /Stateflow.

As Brugge and Dutoit (2004) points out; it is sufficient to have a deep knowledge of a small subset of UML to use it ("*you can model 80% of most problems by using about 20% of UML*"). There is no need to use all of kind diagrams during the design. In author's opinion, *use case diagrams*, *scenarios*, *class diagrams*, *sequence diagrams* and *state diagram* are most important. Additional *system architecture diagram* (supported by Real-time Studio, 2004) may show communication links between parts of system and is very well suited for detailed design of the computerised control subsystems. Component and deployment UML diagrams are software-oriented and less useful in area of control.

3. AN EXAMPLE OF CONTROL PROBLEM

Design starts when need of new or improved solution came into sight (Figure 2). Next step is to transform the idea into specification of requirements, followed by conceptual design, as presented in paragraph 2.1. An example of automotive Anti-lock Braking System (ABS) is used in this paper.

3.1. Elicitation of requirements

During elicitation of requirements, borders and external behaviour of the system are defined and criteria of consumer satisfaction are set. This may be influenced with development strategy of a company.

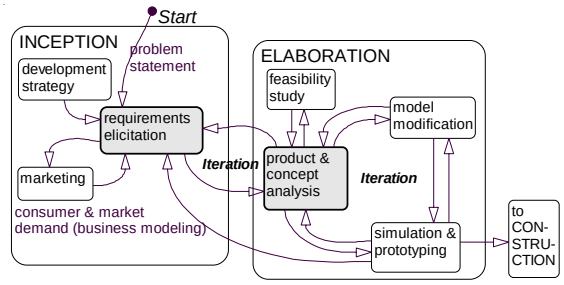

Fig. 2. Early design phases: inception and elaboration

3.2. Graphical representation of requirements on Use Case Diagram.

The goal of elicitation of requirements is to describe what the system should do and (which seems to be equally important) to agree with customer on this description.

Students should take into account that original textual description of the problem may be incomplete or some requirements may conflict with others. Even meaning of the same textual phrase may be different to some members of design team. Blaming the client for a defective problem statement is not acceptable, as consumer satisfaction is one of main objectives of design.

Use case diagram (figure 3) shows actors, use cases and interactions between them. It describes how the system may be used by user and how the system interacts with other external actors.

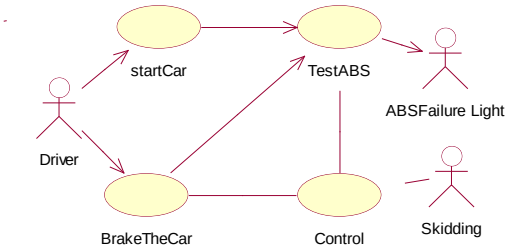

Fig. 3. Use case diagram for automotive ABS

An *actor* (human user, another system or external signal, connected to sensor or interface) is a thing outside the actual system, which interacts with the actual system. *Actor* is depicted as a simple icon of a man.

Use cases are system boundaries identifying what the system should do. They capture subsystem functionality as seen from the point of view of end user or domain expert and help to understand how the system should work. Internal structure of the system is not shown on this diagram.

Use case icon is an ellipse. *Actors* and *use cases* set the border between the system under development and its external environment.

3.3. Scenario describes the sequence of actor and system interactions

Scenario. *Scenario* is a textual description or set of messages in natural language, describing the sequence of *actor* and system interactions. It describes details of *use case* functionality.

- *Driver turns on the car engine. ABS system is self-tested*
- *Driver increases car speed to 110 km/h*
- *He presses the brake pedal to stop the car in emergency. Brakes are active and ABS is self-tested again*
- *ABS system senses non-uniform deceleration of wheels due to wet road surface. It activates modulation of pressure in affected wheels*
- *Car stops*

.

4. CONCEPTUAL DESIGN

Conceptual design (*elaboration* in RUP terminology) is the most crucial part of design process. During conceptual design, feasibility study, estimation of needed resources and business plan for development (including implementation costs) are prepared. The objective are: to establish a sound architectural foundation, to develop the project plan and to eliminate highest risk elements of the project. Preliminary architecture is refined many times, as new UML diagrams are prepared and then iteratively analysed, modified and tested.

4.1. Deciding on architecture of the system

At the beginning, students should analyse the *scenario* to identify candidates for *actors* (driver and road condition (e.g. wet)), objects (car, engine, brake_pedal, ABS_system, wheel) and attributes (turn-on, speed, stop, emergency, deceleration). Similar objects are generalised into class. This leads to preliminary version of *class diagram*.

An internal structure of the system is presented on *class diagram*. It shows classes and relationships that exist between them. An example of *class diagram* for automotive ABS subsystem is presented on figure 4. Students should identify classes from scenario, set the class name, attributes (variables and parameters) and operations (services or responsibilities of a class. They have also to decide on relations, shown as different lines, with or without arrows.

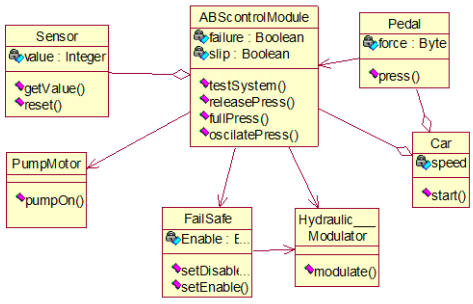

Fig. 4. Static structure of automotive ABS system is presented on class diagram

It is not a good idea to design complete diagrams sequentially. Instead, an iterative and concurrent approach is advised. Objects and classes are used to build other diagrams. Changes in object hierarchy, in naming, operations and attributes are inevitable, when other diagrams are under design. This is especially true when *sequence* or *state diagram* is prepared.

If a good CASE tool is used (e.g. Rational Rose), the same classes on different diagrams are synchronized and updated if class name, attribute, type or operation is intentionally changed on any other diagram. New classes, operations and attributes are added to respective *class diagram* – if needed for actually designed diagram. Other (if not used) are considered for deletion. All changes are performed easily on computer screen.

4.2. Verification of requirements and system architecture

Students should understand that an important reason for building UML diagrams is to mitigate possible failure of the project. During this process, student verifies specification of requirements and *scenarios* against omissions and inconsistencies *Scenario* is verified with *sequence diagram*, which shows what objects does to implement this *scenario*. *Sequence diagram* (figure 5) is a graphical model of the *scenario*.

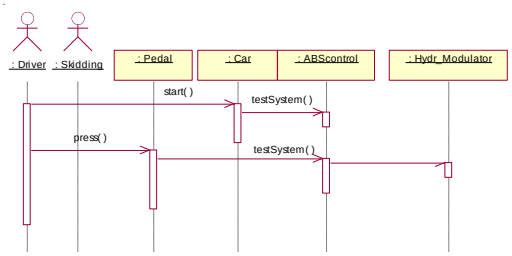

Fig. 5. Sequence diagram for an ABS system

Actors and objects are shown on top of *sequence diagram*. Time flows down the vertical lines (*life line*). Object may send messages (horizontal line with an arrow) to ask needed services from another object (e.g. "Driver" may press "Pedal"), as described in *scenario*. Student should understand that message is not free text but name of operation owned by target class. Timing marks may be added to show exact time constrains.

Collaboration diagram provides essentially the same information as sequence diagram and is not described here.

5. VERIFYING RESPONSIBILITY OF OBJECTS AND SUBSYSTEMS

Statechart diagram of Harel (1987) is used to describe and verify behaviour of the system or its part (subsystem, use case, object). Comparing with sequence diagram or single scenario, the state diagram shows all states the system or its part (class, use case) may go through during its lifecycle.

Each state represents a named condition during the life of an object (e.g. *Brake_Idle* on fig. 6). Object stays in actual state until it is fired by some event or when given condition (that must be fulfilled before the transition) will occur. Lines with arrow show possible transitions (change of state). A black ball shows a starting state. The end state (if exists) is denoted as black ball in a circle.

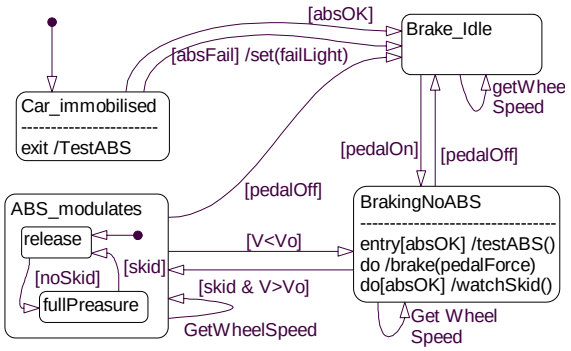

Fig. 6 State diagram for automotive ABS brake

State diagram presented on figure 6 models behaviour of the automotive ABS brake. It was prepared with Stateflow, in the MATLAB/Simulink software environment (Mrozek B, Mrozek Z. 2004). An important advantage of this environment is possibility of simulation (of-line, on-line real-time, hardware in the loop) and prototyping. Later, simulated virtual model may be automatically compiled with MATLAB/RTW/StateflowCoder and loaded into FPGA or micro controller.

If concurrent activities are needed, students may use an activity diagram. It is well suited to describe set of sequential and parallel activities.

6. EDUCATIONAL REMARKS

Lecture on RUP and UML should have 45-60 hours. In laboratory, students should be organised in groups of 2-3 persons. Each group is given the same goal to achieve: to prepare scenario and UML diagrams describing controller for modern product or device (e.g. automotive ABS or ESP[1] controller, automatic coffee machine, etc.) If laboratory equipment permits, the next step should be prototyping of controller in MATLAB/Simulink/RTW or Modelica/Dymola environment, using dSPACE, xPC, FPGA or supported microcontroller hardware. The groups work separately, but each (or every second) week they meet on seminar and discuss their progress and common mistakes. When work is finished or deadline is reached - they prepare their final reports and present them on seminar.

7. CONCLUSIONS

The main reason of teaching RUP and UML is to show that main risk of the project failure may be sufficiently mitigated during early design phase. This is important conclusion, as time delay and cost of modification will be much higher, if corrections and

[1] Electronic Stability Program (ESP), Bosh

redesign are made later, during detailed design, implementation or (it is the worst case) during final design tests.

UML model describes architecture and behaviour of the future control system. Student do not need to decide on physical nature of hardware subsystems and their details, because it is not shown on diagrams created on high level of abstraction. Preparation of diagrams automatically verifies specification of requirements and scenarios against omissions and inconsistencies. This helps to verify functionality of future system and helps to improve the system quality.

Using commercially available CASE packages, UML may greatly improve productivity of the design team by cutting down development time and improving final product quality - in accordance with ISO 9000 standards.

8. ACKNOWLEDGEMENTS:

Author wants to express his gratitude to IBM and The MathWorks Inc (USA) for free evaluation licenses for software presented in this paper.

More systematic description of RUP, UML and Stateflow may be found in cited literature

REFERENCES

Bruegge B, Dutoit A, (1999). *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, Prentice-Hall.

Bruegge B, Dutoit A, (2004). *Object-Oriented Software Engineering: Using UML, Patterns, and Java*, Pearson Prentice-Hall.

Harel D. (1987*).* Statecharts: A visual formalism for complex systems, *Sci of Comp. Programming*, , No 8, pp 231-274.

McLaughlin J. and Moore A (1998) Real-Time Extensions to UML, Timing, concurrency, and hardware interfaces, *Dr. Dobb's Journal* December

Mrozek Z. (2001). UML as integration tool for design of the mechatronic system, In: *Second Workshop on Robot Motion and Control*, (Kozlowski K, Galicki M, Tchoń K (Ed)), pp 189-194, Bukowy Dworek, Poland

Mrozek Z., Mrozek B., Osei Adjei, (2002) Teaching object oriented software engineering with UML, 13th EAEEIE Conference "Innovations for Education in Electrical and Information Engineering", April 8-10, York

Mrozek Z, Tao Wang, Minrui Fei. (2002) UML supported design of mechatronic system, *Proc of Asian Simulation Conference/5-th Int. Conference on System Simulation and Scientific Computing*, Shanghai, China, Nov. 3-6,.

Mrozek Z. (2003). Computer aided design of mechatronic systems, *International Journal of Applied Mathematics and Computer Science*, vol **13** No 2, pp 255-267

Mrozek B, Mrozek Z. (2004) *MATLAB i Simulink, poradnik użytkownika* Helion, Gliwice.

Mrozek Z (2004). Importance of early design phase in mechatronic design. *In: Proceedings of 10th IEEE International Conference on Methods and Models* in *Automation and Robotics* (Domek S., Kaszynski R (Ed)), vol **1** pp 17-28, Miedzyzdroje, Poland

Mrozek Z (2005). *An effective graphical approach to define objectives and structure of a control system* , 16-th Word Congress in Prague

OMG (2006) *Unified Modelling Language*, Resource page *http://www.omg.org/uml*

Probasco L.(2000) *The Ten Essentials off RUP, the essence off an effective development process, TP- 177 9/00, Rational Software Corporation.*

Rational Rose (2006) *Rational Rose RT* Homepage*: http://www-306.ibm.com/software/rational/*

Rational Unified Process (2001). *Best Practices for Software Development Teams*, Rational Software White Paper, TP026B, Rev 11/01,

Rational Unified Process (2006).*Rational Method Composer*, Homepage http://www-306.ibm.com/software/awdtools/rmc

Real-time Studio (2004) *ARTiSAN Software Tools, Inc*. Homepage*: http://www.artisansw.com/*,

Rhapsody and Rhapsody System Designer (2006). *i-Logic*, *http://www.ilogix.com/*

Visual Paradigm for the Unified Modeling Language (2006) Homepage: http://www.visual-paradigm.com