

UML AS INTEGRATION TOOL FOR DESIGN OF THE MECHATRONIC SYSTEM

Zbigniew Mrozek [§]

Abstract Information transfer plays an important role in the design process and can be very easily presented on UML (Unified Modelling Language) diagrams. Author believes that terminology and notation of visual modelling with UML can be used as common language for design of the mechatronic systems and as documentation tool on every design phase.

Index terms UML, mechatronics, flexible arm

I. INTRODUCTION

Different technology and systems are used in design and production. Members of interdisciplinary team have different background and skills. This gives a great opportunity to design an innovative product. But they use different methodology and their own way to describe their work. This may lead to misunderstandings and team-unifying tools should be defined to guarantee precise communication between the team members. In author's opinion, UML can be used to describe all elements of mechatronic system and will be widely used in design of innovative product.

II. UML AS LANGUAGE AND DESIGN TOOL

UML was introduced as language for modelling of the information systems. UML represents a collection of the best engineering practices that have proven successful in modelling of large and complex systems [1,2,3,6,7,15]. UML is independent of what programming language will be used for coding final software system. Many attempts were done to extend the UML applicability in the areas beyond informatics. McLaughlin [8] showed UML approach to process-control problem that contains a conveyor-belt transport subsystem. Using UML for real-time systems is presented in [4,5,8,10]. An example of robot transmission system UML model (figure 1) can be found in [9].

UML is derived from three other products: *OMT* (object modelling technique) by James Rumbaugh, *Booch method* by Grady Booch and *OOSE* (object oriented software engineering) by Ivar Jacobson. The UML language was improved many times until version UML 1.3 was accepted as proposal for standard in year 1999 [6].

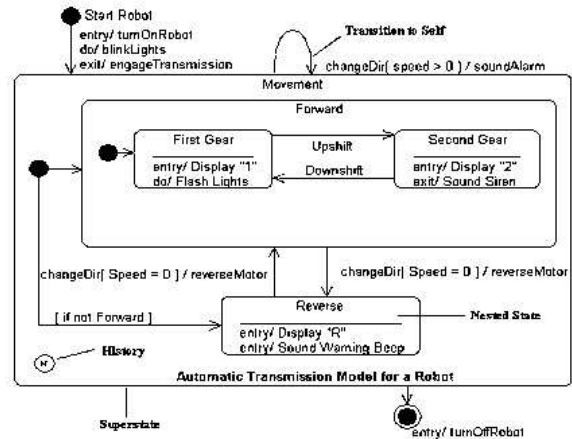


Fig 1: UML statechart diagram describes behaviour of robot transmission subsystem [9]

Any complex system can be presented by a set of nearly independent views of a model. Single view is not sufficient. Use case diagram and class diagram (there are described later) are probably used in all UML supported projects. The choice of what other diagrams are created depend on how a problem is attacked. In addition to use case and class diagrams one can create any of behavioural diagrams:

- statechart diagram
 - activity diagram
 - interaction diagrams: *sequence* and *collaboration*
- and implementation diagrams:
- component diagram
 - deployment diagram

Some CASE tools supporting UML design may offer their own extensions and diagrams which are not included in current UML specification (e.g. *system architecture*, fig. 7).

III. MANAGEMENT OF INTERDISCIPLINARY DESIGN PROJECT

The mechatronic design process can be split into three general phases:

- analysis,
- prototyping and testing
- implementation

Analysis phase includes requirements elicitation and decomposition of the future system. An important advantage of visual programming with UML is possibility of finding inconsistency and incompleteness of requirements specification on early stage of design, when project modification is relatively cheap. It means that analysis phase may decide on success or failure of the project.

[§] Cracow University of Technology, PL 31-155 Krakow, Poland, e-mail: zbigniew.mrozek@pk.edu.pl.

A virtual model of the future system is designed during **prototyping and testing phase**. Some UML diagrams can be adopted for automatic generation of virtual model of the future mechatronic system. The virtual model is then prototyped in real time, in environment representing future working conditions of the final product [11-13].

During **implementation phase** prototyping equipment should be replaced by a target system: cheaper, smaller, easier to operate and more reliable in industrial environment.

CAD tools are very useful during both: prototyping and implementation phase. For example digital electronic and computer electronic parts of the mechatronic system can be almost automatically designed in silicon as ASIC (*application specific integrated circuit*) hardware using FPGA (*field programmable gate arrays*) chips and software from Xilinx or Altera [13].

A. Use cases

The main tools used for requirements elicitation during analysis phase are *use cases* and *scenarios*. *Use case* is represented by an ellipse on use case diagram (figure 2). It captures sub-system functionality as seen from the point of view of end user or domain expert. It helps to understand how the system should work. This is an important job, as original problem description may be incomplete and some requirements may conflict with others. Collection of *use cases* describes different behaviour of the system and shows how it interacts with external *actors*.

Actor is a human user, another system, sensor or anything located outside of the actual system that will interact with the system. *Actor* is depicted as a simple icon of a man. Defining *actors* is essential to set the border between the system under development and its external environment.

Use case diagram for flexible arm test rig is presented on figure 2. It shows an operator (an *actor* in UML terminology) who may need to set a new target position of flexible arm, to stop the arm in an emergency or to finish a job and move the arm to a parking position. Calculating trajectory of a flexible arm is an extension *use case* (see dash line with arrow) to 'next position' *use case*.

Other three *use cases* are interacting with other actor who is responsible for design and maintenance. He may need to identify mathematical model of flexible arm, to run off-line simulation or to perform prototyping.

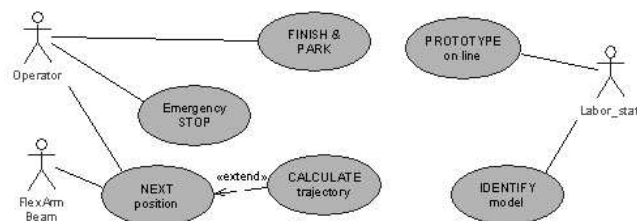


Fig.2 Use cases for prototyping controller for flexible arm. External actors may interact with the system.

The actual view of flexible arm test rig is presented on figure 3.

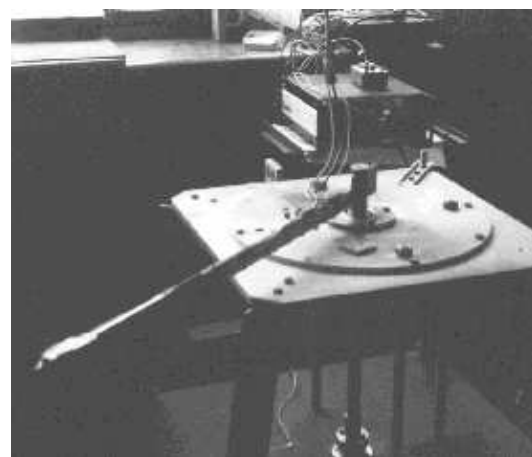


Fig. 3 Actual view of flexible arm test rig.

Use case has high level of abstraction and internal structure of the system is completely hidden on *use case* diagram. Both client and members of design staff should understand *use cases* as they must verify if all functionality of the future system is included in *use case* diagram and if some functionality overlaps different *use cases*.

Use cases are useful to prepare tests for the system. Tests should be defined on early stage of the design. Postponing this work is hazardous as objectivity can be lost. If test specification is prepared late in second or third phase of design, team members may prefer to choose tests reflecting properties of actual system under design and client may expect to include properties, which extend requirements specification.

B. Scenario

Let's assume, member of laboratory staff (*actor*) has to start the computer and to load an application software. Then he performs the desired task e.g. he identifies parameters of mathematical model of flexible arm system. Several messages will be issued during this task by actor (with a keyboard) and by the system (on a computer screen). *Scenario* is defined as set of messages (or description) in natural language, describing chosen sequence of *actor* and system interactions.

C. Implementation diagrams

Mechatronic system is combined of products and processes of different nature. On the contrary to pure informatics systems, this approach presents *implementation* diagrams on early stage of design.

UML implementation diagrams show structure of components and the run-time deployment system. They come in two forms: *component* diagram and *deployment* diagram. *Component* represents a modular, deployable and replace-able part of a software system (figure 4). It acts as a logical container for objects and classes. Small rectangles protruding from component's symbol side are interfaces. They represent services provided by the component. Elements that reside in a component can be shown inside the component symbol.

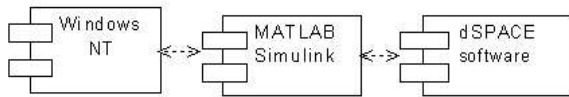


Fig.4 Component diagram for software system of FlexArm experimental rig.

Deployment diagram (not presented here) shows the configuration of run-time processing elements (nodes) and software components. Nodes are computing devices but also human resources or mechanical processing resources. Software component represents run-time instances of software (code units that execute on the node), processes and objects. Original UML *implementation* diagrams are not very useful in mechatronic design and using a *system architecture* diagram (fig. 7) is strongly suggested.

Once the system boundary of mechatronic system has been determined with *use case* diagram, physical interfaces to all actors should be identified as part of requirements for the system. Result can be presented on *system architecture* diagram. This is an extension of *deployment* diagram which is offered inside *Real time Studio* package [10]. Figure 7 shows preliminary version of FlexArm system architecture. I/O devices (monitor, keyboard, serial mouse) are connected to PC motherboard. Sensors and actuators of FlexArm test rig are connected to dSPACE board of PC_computer package. Each of three boards act as processing node. Later more details can be assigned to each element of this diagram. Using *properties* window one can assign memory address, IO address, IRQ and other important parameters. As the proposed solution is investigated further during the definition of the technical architecture more detail is added. Typically at this stage Commercial Off the Shelf (COTS) boards are added to the architecture and the requirements for any custom boards are defined. For (COTS) boards they will either be already documented as existing available board types or the hardware engineers will do it at this stage. Apart from custom hardware most of the system architecture should now be complete.

As any new hardware is designed and built during the development of the increments the System Architecture Model should be revisited to ensure that the new hardware does not require any changes to the model. The System Architecture Diagram below shows more detail.

D. Sequence and collaboration diagram

Mapping of the *use case* functionality onto object actions can derive *sequence* or *collaboration* diagram. Both diagrams are alternate representations of objects interaction. A *sequence* diagram is a graphical view of a *scenario* that shows object interaction in a time-based sequence (figure 5) while *collaboration* diagrams show how objects associate with each other (figure 6).

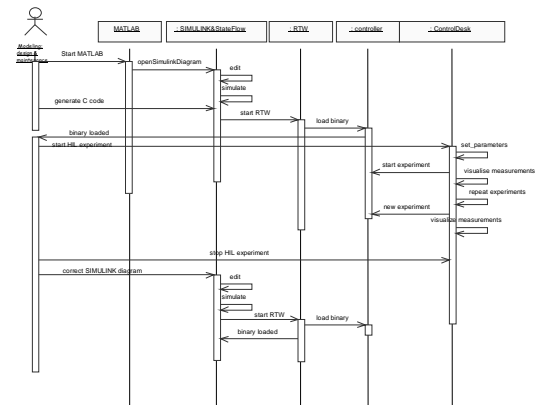


Fig. 5 Sequence diagram explaining fast prototyping methodology

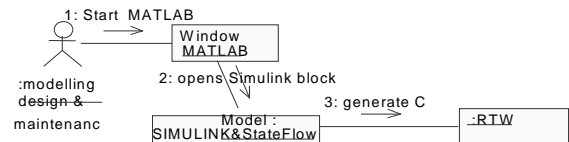


Fig. 6 Simplified collaboration diagram for part of the sequence diagram from figure 5

Sequence and *collaboration* diagrams are powerful modelling tools. They are useful during both: analysis phase and during prototyping and testing phase. They help to find incompleteness and inconsistency in specification of requirements and in *scenarios* (analysis phase of design). Later they help to verify attributes, functionality and responsibility of classes and objects. As an example, *sequence* diagram explaining fast prototyping methodology is presented on figure 5. One *actor* (design and maintenance) and several objects (software packages and hardware controller) are presented on this diagram. Similar information is presented on collaboration diagram (figure 6).

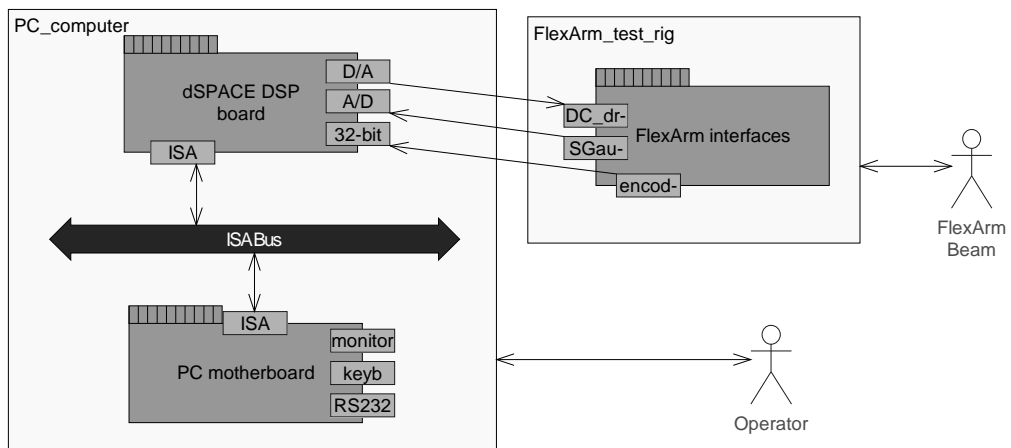


Fig.7 System architecture diagram from [10] is an extension of UML deployment diagram

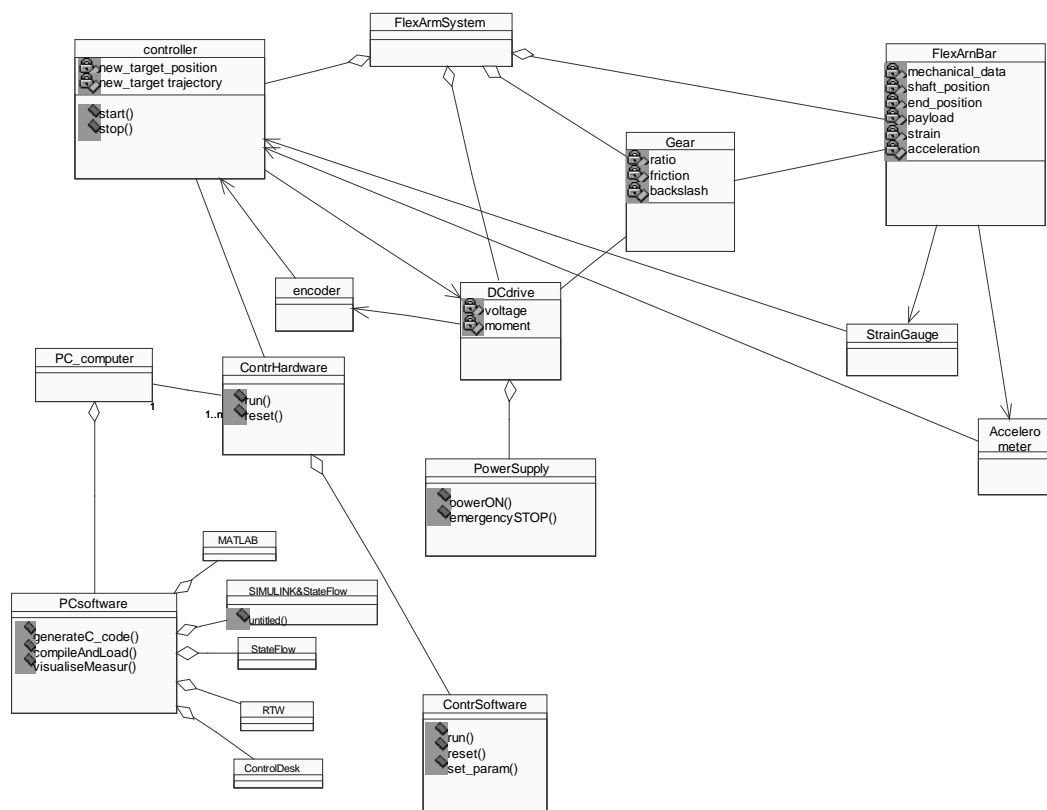


Fig. 8 Class diagram for flexible arm rig.

E. Class and object diagram

Class and object diagram show internal structure of the system. Object is an entity that has some functionality, responsibility and attributes. Class is used as template to define one or more similar objects. One can identify object as physical devices, data containers, interfaces etc. Use cases and scenarios should be analysed to find objects.

An example of class diagram for flexible arm rig is presented on figure 8. Controller, DC drive, Gear and FlexArmBar classes belong to FlexArmSystem class. They are connected to FlexArmSystem using aggregate associations (lines with diamond on the end). Class hierarchy is presented on the same figure: Power Supply belongs to DC drive and Control Hardware belongs to the Controller. Unidirectional association (line with arrow on the end) and bi-directional association (line without arrows) are used to show possible path for exchange of information (messages) between classes. Inheritance is not shown on the figure, but one can define the Sensor Class, which will be the parent class for Encoder Class, Strain Gauge Class and Accelerometer Class.

Preliminary version of class diagrams can be build quite easily. Then it should be verified to include all important properties from use cases and scenarios. Then if sequence, collaboration or any other diagram is changed or build, this may affects all other diagrams. CASE tools supporting UML programming will automatically modify all other diagrams to keep all the system consistent with new diagram.

F. Statechart diagram

Statechart diagram models dynamic behaviour of classes or objects. Comparing with sequence diagram (it shows chosen scenario), the statechart diagram shows all states the object goes through, events that cause a transition from one state to another and results (actions) of change of the state.

Each state represents a named condition during the life of an object. It stays in actual state as long as it satisfies some condition or until it is fired by some event to other state. A black ball shows a start state. An end state (if exists) is shown with black ball in a circle. Transitions (straight or curved lines with arrow) connect the various states on the diagram. Decisions, synchronizations, and activities may appear on statechart diagrams. An example of state diagram for an approach controller of flexible arm is presented on figure 9.

A statechart diagram for robot transmission (figure 1) is more complicated. Some states have an entry and exit actions that are completed instantaneously on entry or on exit from the state. An extended activity (labelled with do) is performed when entry action is finished. An extended

activity will be interrupted by any event that changes state of an object.

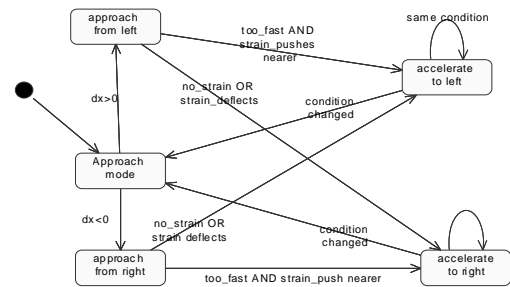


Fig. 9: Statechart diagram for approach controller

IV. HIL EXPERIMENT RESULTS

Many different packages can be used for prototyping. Author's chose is *MATLAB* (it integrates mathematical computing and visualization) and *Simulink*, an interactive tool build on *MATLAB*, used for modelling, simulation, and analysis of complex systems. *Real Time Workshop* and *dSPACE* (or other software and equipment) is used for prototyping.

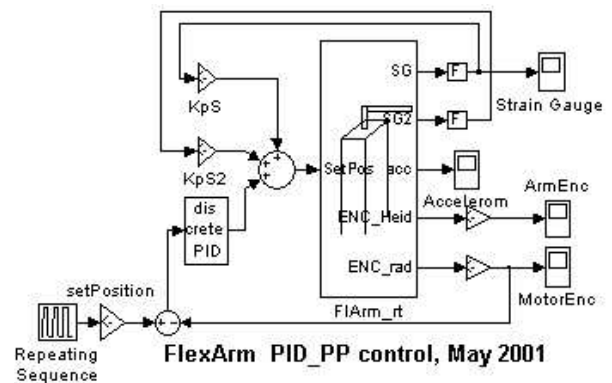


Fig 10. Simulink block diagram for HIL simulation

Simulink block diagram is presented on figure 10. Mechatronic object hardware is presented as *FIArm_rt subsystem* (rectangle with rotary beam icon) on Simulink diagram. This hardware is physically connected to dSPACE prototyping board. It includes *FlexArm* rig from figure 3 with DC electric drive and measurement equipment. Software interfaces from dSPACE library are incorporated in *FIArm_rt* subsystem.

All other icons on figure 10 are taken from Simulink block library. The only exception is *discrete PID controller* which was designed by author. The Simulink diagram structure and functionality is derived from use case diagram (figure 2) and other UML diagrams. The reference trajectory for flexible arm is memorised in *Repeating Sequence* block.

Position error signal is derived from *Enc_deg* encoder signal minus reference (polarity of original signal is reversed). This signal is fed to *discrete PID* block. Two *Strain Gauge* signals are weighted with *KpS* and *KpS2* gain ratios and added to input in order to damp oscillations of flexible arm. An approach controller from figure 9 is not implemented yet. Preliminary values for PID controller parameters and *Strain Gauge* gains were found off-line using non-linear model of test rig (described in [13]) and NCD toolbox from *The Mathworks Inc.*

The fast prototyping procedure using HIL simulation is presented below. C-language code for Simulink block diagram (including interfaces to hardware equipment hidden in *FLArm_rt subsystem*) is automatically generated with RTW package toolbox from *The Mathworks Inc.* The binary code is then loaded into memory of a prototyping board. Virtual model exists as binary code in memory of prototyping board. When ready, one may run simulation using FlexArm rig from figure 3 as a hardware in the loop.

All simulation results are carefully verified against the requirements and the prototyping procedure is repeated many times until the simulation is completely satisfactory. Frequent and effortless generation or tuning of virtual model parameters and multiple repetition of all or part of described steps (until required result is obtained) is an essential feature of prototyping methodology.

Results of HIL experiment are presented on figure 11. Upper diagram shows reference signal and outputs from two encoders. Lower diagram shows non-filtered strain gauge signals. All signals are relatively well damped.

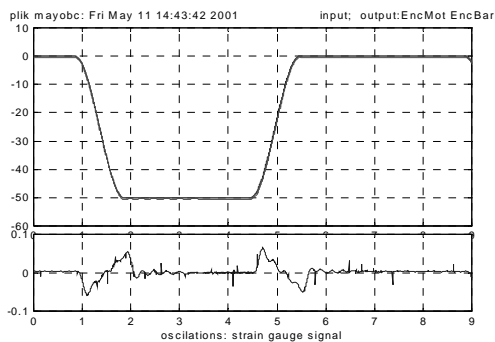


Fig 11. HIL experiment results: position tracing (upper diagram) and non-filtered strain gauge signal (lower diagram).

V. CONCLUSIONS

An advantage of UML is revealing of gaps and inconsistencies in requirements specification on very early stage of design, as well as ease of understanding and modification of visual modelling diagrams. Unification and precision of notation is important for large and interdisciplinary projects. UML tools keep track of used class and object names, its attributes and methods. User may transfers already defined classes and other elements

between different diagrams and reuse them. This accelerates work progress and helps to keep all parts of project consistent.

Using commercially available specialised CAD tools and CASE packages supporting visual UML programming may greatly improve productivity of designing team by cutting down development time and improving final product quality (according to ISO 9000 standards). UML is dedicated for large software project, but its applicability can be extended to other domains, including mechatronics.

VI. ACKNOWLEDGEMENTS:

Author wants to express his gratitude to *ARTiSAN Software Tools, Inc (GB)*; *Rational Software Corporation (USA)* and *Premium Technology Sp zoo (Poland)* for free evaluation licence for following software: *Real-time Studio*, *Rational Rose Suite* and *Rational Rose RT*.

VII. REFERENCES:

- [1] Arief L.B, Speirs N.A, *Automatic generation of distributed system simulations from UML* in Modelling and Simulation, A tool for next millennium, *13-th European Simulation Multiconference*, Vol.II pp 507-509, Warsaw, June 1-4,1999,
- [2] Booch, G. Rumbaugh, J. Jacobson I *The Unified Modelling Language User Guide*, Addison Wesley, 1999
- [3] Bruegge B, Dutoit A, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, Prentice-Hall, 1999,
- [4] Douglas, B.P. Real-time UML: *Developing Efficient Objects for Embedded Systems*. Addison Wesley, 1998.
- [5] Douglas, B.P. *Designing Real-time Systems with UML, Part I-III, Embedded Systems*, March, April, May 1998.
- [6] Kobryn Cris, *UML 2001 A Standardization Odyssey*, Communications of the ACM October 1999/Vol.42.No.10, pp.28-37
- [7] Kobryn, C. *Modelling Software Architectures with UML*. Addison Wesley Longman, 2000.
- [8] McLaughlin Michael J., Moore Alan, *Real-Time Extensions to UML, Timing, concurrency, and hardware interfaces*, Dr. Dobb's Journal December 1998
- [9] *Rational Rose Suite*, *Rational Rose RT* and other software from Rational Software Corporation,
- [10] *Real-time Studio*, ARTiSAN Software Tools, Inc. July 2001,
- [11] Uhl T (editor), *Wybrane problemy projektowania mechatronicznego*, Katedra Robotyki i Dynamiki Maszyn, AGH 1999.
- [12] Uhl T, Bojko T, Mrozek Z, Szwabowski W, *Rapid prototyping of mechatronic systems*, Journal of Theoretical and Applied Mechanics, pp.655-668, vol.38.3, 2000
- [13] Uhl T, Mrozek Z, Petko M *Rapid control prototyping for flexible arm*, Preprints of 1-st IFAC Conference on Mechatronic Systems, vol II, pp 489-494, Darmstadt, September 18-20, 2000.
- [14] *UML for Real Time System Design*, ISG 1998
- [15] *OMG Unified Modeling Language Specification (draft), Version 1.4, February 2001.* and other OMG (Object Management Group) standards, <http://www.omg.org>: