Successful company must react quickly to changing trends in the market. New products should be designed and manufactured quicker and cheaper than counter partners can do. A shorter design time is achieved by using computers to support design process. The paper describes two approaches towards designing of interdisciplinary mechatronic systems: first is visual modelling with UML, the second is physical modelling with Modelica.

Key words: mechatronics, UML, CASE, physical modelling, Modelica, Dymola, Stateflow

1 Introduction

UML (unified modelling language) is widely used in designing complex and reliable informatics systems. In mechatronics it provides means for capturing system requirements and for visual modelling of future system on high level of abstraction (see chapter 3). Modelica is freely available language for object oriented physical modelling. It may be used for modelling of system dynamics and prototyping on medium level of abstraction, as described in chapter 4 and 5.

Information transfer plays an important role in operation of mechatronic system. This can be easily presented on UML diagrams. Terminology and notation of visual modelling with UML can be adopted as common high-level object oriented language for design of the mechatronic systems and as documentation tool on every design phase [19, 20, 22]. UML provides the ease of modelling, understanding and modification of graphical diagrams of mechatronic systems. It integrates the best practices of object-oriented development.

Designers may transfer already defined subsystems and other elements between different UML diagrams and reuse them. This accelerates work progress and helps to keep all parts of project in a consistent manner [2, 4, 7, 8]. UML is supported by all major CASE (computer aided system engineering) tool vendors. The main advantage of UML is that it reveals gaps and inconsistencies in the specification of requirements, at very early stages of the design.

Many companies switch from paper blueprints to digital representation of future product. Successful examples include some modern cars and Boeing 777 plane [27]. Weak point of computer aided design process is lack of widely accepted, integrated multidisciplinary software and hardware environment for successful design, testing, prototyping, implementation and validation. Instead, a sequential design approach is traditionally used: mechanical design at beginning, then system is extended with sensors, actuators and non-mechanical subsystems. They are integrated during modelling, simulation and prototyping phase, when control system is designed and tuned [14, 32].
2 Computer aided design in mechatronics

An important objective of CAD (computer aided design) in mechatronics is integration of different disciplines by unification of design process. This helps to include mutual interactions of subsystems of different nature and to unify documenting of all actions and results obtained on each stage of project. Some other factors of using proper CAD/CAM (computer aided manufacturing) and CASE tools are:

- cutting down costs and time needed for design and to offer new product on market,
- integration of different subsystems and technologies on early stage of design,
- using virtual models (a code in computer memory) and special electronic hardware for prototyping. This includes HiL simulation, described in [31, 33],
- compatibility of design process with PDM (product data management) tools [36].

Mechatronic system integrates mechanical, electrical and electronic subsystems with microprocessor based control subsystem. An interdisciplinary approach means an equal significance of all subsystems during design, regardless of its physical nature. This is difficult, if standard design tools are used, but essential to achieve good results. Traditionally:

- specialised CAD/CAM tools are used to design mechanical, electrical and other subsystems in different domains. They are effective in their own domain and weak or inapplicable in others,
- control subsystem is modelled and designed with multipurpose software packages as MATLAB/SIMULINK [18]. They are not directly applicable for modelling of mechanical and electronic subsystems, if needed parts are not included in accompanying libraries or extensions (e.g. SIMULINK toolkits, SimMech, etc.). New approach is to use physical modelling with Modelica on this stage of design.

It is proven in practices that mechatronic approach is successful in designing of interdisciplinary products, even if effective integration of activities in different disciplines is difficult. The development process is set of partially ordered steps, which leads to desired target. Sequential approach always fixes project status before switching from one discipline specific design tool to another. Then special arrangement is needed to exchange project data between used tools. Known solutions are extra software tools, protocols and interfaces for data exchange. Main disadvantage is that above tools often work in off-line mode only. This means, multi-criteria optimisation (essential in mechatronic design) is very difficult to be implemented, as parameters imported from other packages are fixed during some steps of optimisation. Nevertheless, many interdisciplinary products as VCR (video cassette recorder), compact disk player or ABS (anti-lock brake system) were successfully designed and introduced on market [30, 31].

3 Using UML diagrams in mechatronic design

UML was originally developed in response to call for a proposal for a standardized object modelling language. Then it was improved many times until UML version 1.3 was accepted by OMG (object management group) as proposal for standard in year 1999. New UML versions are under development.

UML is the language for modelling of the information systems (supported by main CASE tools) but can also be used to describe all elements of mechatronic system on high levels of abstraction. Many attempts were done to extend the UML applicability in the areas beyond informatics. McLaughlin [16] in year 1998 was probably the first to show UML approach to process control problem. He described a conveyor belt transport subsystem. Using UML for real-time systems is presented also in [7, 8, 25].

Any complex system can be presented by a set of carefully chosen models. Single model is not sufficient to describe a real system. 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 (sequence or cooperation diagram and statechart or activity diagram) depends on experience of designing team.
3.1 Design is a part of product life time

Designing of mechatronic system is an important part of product life time (fig. 1). It is an iterative process, as designers often jump back one or more steps to redesign or tune what they have done before. Design starts with idea of product and includes requirements specification, conceptual and detail design, prototyping and testing, implementation and validation, production, exploitation and recycling of product. Redesigning a virtual model of future product (model is prepared on computer screen as UML diagrams) is supported with CASE software. Best-known packages are Rational Rose [24] and RtS [25]. Using UML helps to find and to correct errors and omissions in requirements specification on very early phase of design, when models on high level of abstraction are prepared. Some CASE tools offer simulation and animation of UML models. This helps to verify behaviour of future system. Simulation is even more realistic if some extra tools (e.g. Altia FacePlate for RtS[25]) are used to build virtual operator console with animated dials and gauges.

![Figure 1. Designing is an iterative process, an important part of product life time.]

Later CAD/CAM and CAE tools are used in detailed design of subsystems. Parameters from detail design are used in simulation models. Modelica, MATLAB, SIMULINK and other software is used to build models for virtual and HIL (hardware in the loop) simulation. Simulation is used to tune and verify behaviour of designed product on prototyping stage, before its physical model is prepared.

3.2 Suggested order of building different UML diagrams

A starting point of design is requirements elicitation (fig. 2). The main tools used for requirements elicitation are use cases and scenarios. Use case diagram describes the functional behaviour of the system as seen by external user (an actor). Scenarios and use case diagram (fig. 3) are prepared with help of client or user of the future system.

![Figure 2. UML diagrams are supposed to be build in predefined order. Other (domain specific) tools are used for detail design and prototyping]

The next step is to analyse use cases and scenarios to identify objects. This leads to preliminary version of class and object diagrams. Later objects and classes are used to build other diagrams. Sequence of actions described in scenarios is graphically presented on sequence (fig. 5) or collaboration diagram (fig. 6). Both are called interaction diagrams. All states that an object may go through are presented on statechart diagram.
(fig. 7, 8). It describes the dynamic behaviour in response to events and fulfilled conditions. Parallel activities may be shown on activity diagram (fig. 9). System architecture diagram (fig. 10) should be used in mechatronic design instead of UML implementation diagrams. It is supported by RiS [25], a CASE tool from Artisan. Next steps, on medium level of abstraction, are presented in chapter 4 and 5.

### 3.3 An actor and use case diagram

An actor is a human user, another system, sensor or anything located outside of the actual system that will interact with the system. When actor (e.g. "user" on fig. 3) communicates with the system, it means the actor sends messages to the system or receives messages from the system. Actor is depicted as an icon of a man. Defining actors is essential to set the border between the system under development and its external environment. Actors are used on use case, sequence and collaboration diagrams.

Figure 3. Use case diagram for automatic toaster

Single use case is a named oval on use case diagram, e.g. "adjust_heating_time". Use case diagram is collection of use cases and actors (fig. 3). Use case captures subsystem functionality as "black box" seen from the point of view of external user (e.g. "user"). As internal structure of use case is completely hidden; it has the highest level of abstraction of UML diagrams. Use case diagram helps to understand how the system should work. It describes different behaviour of the system and shows how it interacts with external actors.

Preparing use case diagrams is an important job, as original problem description may be incomplete and some requirements may conflict with others. Both client and members of design staff should understand use cases. User should verify if all needed functionality of the future system is included in use case diagram and if all actors do communicate with respective use cases.

Any extra requirements which are not shown in use cases or scenarios may be included as constrains (restrictions or rules applied to various elements of a model). This is especially useful in real time systems, where timing constraints for latency of messages and processing time limits for operations should be followed. OCL (object constraint language [38]), pseudocode, annotations or text (inside a notes icon) can be used to show constrains in UML models.

### 3.4 Scenarios and tests

Scenarios are instances of use cases. A scenario is systematic description of sequence of messages sent between actors and the system. Following scenario describes how Mr. Brown (an actor in UML notation) prepares a toast for breakfast:

> Mr. Brown puts fresh toast into toaster and moves it down with a slider. Toast goes down and electric heater is switched on. Mr. Brown adjusts heating time by turning a knob on toaster’s side. When heating time is elapsed, heater is switched off and toast jumps out of the toaster. Toast is ready to eat.

There is large number of possible scenarios corresponding to a single use case. It is important to prepare at least one non-trivial scenarios including exception handling and error recovery e.g. if something goes wrong, power can be switched off manually and toast is released immediately.

Use cases and scenarios are useful to prepare tests for the system. Tests should be prepared on early stage of the design. Otherwise members of design team may prefer to choose tests reflecting properties of actual
system under design and client may expect to include properties, which extends agreed requirements specification.

### 3.5 Class and object diagram

Class and object diagram shows internal structure of the system. It defines the system structure by identifying objects, defining their classes and relationships that exist between classes. Figure 4 shows final version of class diagram for automatic toaster [9].

![Class diagram for automatic toaster](image)

**Figure 4. Class diagram for automatic toaster**

Use cases and scenarios should be analysed to find objects, their names, responsibility, activities and parameters. One can identify object as physical devices, sensors, actuators, interfaces etc. If objects are known, preliminary version of class diagram can be built quite easily. The name of class is given in upper compartment of rectangular icon, e.g. “timer”. Values of variables and parameters are kept in attributes field, in middle compartment, e.g. “time_to_release”. Methods (services and responsibilities of class) are given in bottom compartment, e.g. “start_timer”.

### 3.6 Sequence and collaboration diagrams

Sequence diagram and collaboration diagram show graphically information from scenarios. Figure 5 shows how typical sequence diagram looks like. Actors and objects are shown as icons on top of vertical time lines. Time flows down the line. Object may send messages (shown as horizontal line with an arrow) to ask some services from another object (e.g. “user” asks “toast_slider” object to “move_toast_inside”). Activities shown on the sequence diagram may be annotated with text. Timing marks can be added to show exact time constrains.

![Sequence diagram for automatic toaster](image)

**Figure 5. Sequence diagram for automatic toaster.**
Collaboration diagram (fig. 6) shows structural view of scenarios but provide essentially the same information as sequence diagram. In absence of time lines, the sequence of messages is given by numbers.

### 3.7 Statechart and activity diagram

An important subset of all classes can be modelled using finite state machines. Such classes are called reactive as they react in specific way to incoming events. Control engineer will find it helpful to describe reactive behaviour of system states and to map system logic to physical architecture e.g. to FPGA chip.

Statechart diagram models behaviour of reactive entities by specifying its response to events received. It is used to describe behaviour of class instances, but statechart may also describe the behaviour of use case, actor, subsystem, etc [2, 38]. Comparing with sequence diagram (it shows chosen scenario in time-based sequence), the statechart diagram shows all states the object may go through.

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 as black ball in a circle. Transitions (lines with arrow, fig. 7) connect the various states on the diagram. Toaster has only two stable states and its statechart diagram is very simple. If toast is put inside the toaster, it switches the power on and the toaster transits to heating state. When heating time is elapsed (“time_to_release<=0”) or if eject button is pressed, toast is ejected and heating power is switched off. The toaster transits to idle state and a new toast can be started.

Statechart diagram is very useful to verify functionality of more complicated products. Figure 8 presents subsystem for positioning of arc welding gun. This statechart diagram is prepared with Stateflow [15, 18, 22] software in MATLAB environment. If the gun is in “follow welding path” state, the welding wire if fed into gun (do: feed_wire) and gun moves along welding path (do: move_gun). It finishes if end of trajectory is reached. If welding arc die or if welding wire is pinched, actual state transits to “strike_welding_arc” state or to “tear_of_wire” state, respectively. All exit conditions (here: “exit: move_gun(0)” and “exit: feed_wire(0)”) are fulfilled before leaving “follow welding path” state.
Activity diagram (fig. 9) is used to visualise parallel activities and to show sequence of internal states. It is well suited to describe set of sequential and parallel actions when preparing the welding gun to work in MIG/MAG welding mode. Short horizontal or vertical thick lines are used for synchronisation of actions.

3.8 Implementation diagrams

Original UML implementation diagrams (component and deployment diagrams) are dedicated to information systems and they are not very useful in mechatronic design. Using a system architecture diagram instead is strongly advised.
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 UML included in RtS package [25]. Figure 10 shows all
main subsystems of arc welding system. They are connected with CAN-bus communication network. This
include operator console with I/O devices (keyboard, joystick, LCD display), work piece positioners, power
source for arc welding gun, robot controller, wire and gas feeding system. Later more details can be assigned
to each element of this diagram.

3.9 Remarks on using UML

Preparing UML diagrams is an extra work, comparing with classic design methodology. But time used is not
lost. It is not possible to build UML diagrams if there are gaps or inconsistencies in requirements
specification. So problems with requirements are identified and corrected on high abstraction level of UML
diagrams. Then statechart and activity diagrams are used to verify behaviour of future system against use
cases and scenarios. Verification of behaviour is done before more detailed models are prepared (with
CAD/CAM, SIMULINK or Modelica software) and well before physical prototypes are built. As result,
detailed models and physical prototypes will probably need much less alteration than without using UML.

On the other hand, even little knowledge of UML languages is sufficient to read and understand most of
UML diagrams. This gives opportunity to all members of designing staff, regardless of their speciality, to
actively cooperate in conceptual phase of design, when subsystems are defined and technology to build them
is chosen. This helps to integrate different technologies in mechatronic design.

If CASE tools (e.g. RtS [25]) are used, double clicking on any UML diagram item, shown on a computer
screen, opens its properties window. One can memorize parameters and other information using this
window. This information is kept inside internal database of CASE tool and should be accessible to
members of design team. As the project is further investigated, requirements for any custom subsystems are
defined and COTS (commercial off the shelf) boards and subsystems (mechanical, electrical, etc) are chosen.
Their parameters are also added into this database, using respective property windows. The result is that
actual documentation of the product may be automatically generated by CASE tool on any stage of design.

UML diagrams are often modified during design. This includes adding or deleting attributes or methods of
UML classes. As a result, other diagrams, which use affected classes, may become invalid. CASE tools
supporting UML programming [e.g. 24, 25] will try to modify automatically all other diagrams to keep them
consistent. If automatic update of diagrams is not possible, the CASE tool will warn the designer.

Brainstorming may be used to identify possible solutions of problems and to find potential opportunities for
improvements. This technique can be used to prepare scenarios and to identify states and transitions for
statechart diagram [21].

4 Physical modelling, simulation and prototyping with Modelica

Modelica [10, 11, 17] is freely available language for object oriented physical modelling, which is
developed through an international effort. The work started in 1996 under ESPIRIT project: Simulation in
Europe, Basic Research Working Group (SiE WG). The author of this paper was involved in preliminary
work of this group during his temporary stay in Ghent University. Modelica language unifies and
generalizes previous object-oriented modelling languages and is intended to become a de facto standard. It
complements high level of abstraction models in UML language.

4.1 Integration of models of different nature

In modelling and simulation environment, it is desirable to integrate models specified in different modelling
formalisms (ODE, DAE, PDE, statechart diagrams, reusable library models, etc) and to support multi-
Modelica supports modelling on medium level of abstraction by composition of library components into block diagrams. Missing library components may be designed by connecting library components into subsystems or designing them from scratch using their description by ODE (ordinary differential) and DAE (differential algebraic) equations.

4.2 Physical and acausal modelling

Physical modelling is based on relations between physical quantities. This may be achieved by cutting the system into subsystems and defining equations for bi-directional connections between those subsystems. Connections specify interactions between subsystems and they are shown as lines connecting subsystems on block diagram. A connector describes all physical quantities involved in cooperation of subsystems.

Flange_a of mechanical gear (fig. 13) is defined as (comments are given between “quotation marks”):

```modelica
  " 1D rotational flange (filled square icon)"
SIunits.Angle phi       "Absolute rotation angle of flange";
flow SIunits.Torque tau          "Cut torque in the flange";
end Flange_a;
```

Here each connection is used to generate equations for across (equal) variables and flow type (through, zero-sum) variables

\[
a_1 = a_2 = \ldots = a_n; \quad \text{“across variables, e.g. absolute rotation angle”}
\]

\[
z_1 + z_2 + \ldots + z_n = 0; \quad \text{“through variables of flow type, e.g. torque”}
\]

There are two variables from SIunits library describing Flange_a connector. The tau variable (torque in the flange, class name: SIunits.Torque) is of flow type. This means that sum of torque values on connector are equal zero. On the other hand, phi variable (the rotation angle, class name: SIunits.Angle) is not of flow type and rotation angle values are equal on both sides of flange connector. In acausal modelling language (as Modelica), direction of energy flow is not predefined and connector may be bi-directional. If connector definition is properly done, its power balance is satisfied.

Physical modelling offers great improvement comparing to traditional approach of block diagram, where artificial unidirectional signals on input and output of block diagram element are considered. Connector variables are physical quantities, which have to fulfil respective physical phenomena. This idea is used also in Bond Graphs, where efforts variables (e.g. voltage or rotation angle) are equal each other, and sum of flow variables (e.g. current or torque) are equal zero. Similar approach is known in electrical engineering as current and voltage Kirchoff’s law.

Any extra library component may be build from set of ordinary differential equations (ODE) or differential algebraic equations (DAE), preferably in state space form:

\[
dx / dt = f(x,u)
y = g(x,u)
\]

where
t - time,
x, y - vectors of unknown variables (state and control).

Modelica language accepts more general declarative equations:

$$f\left(\frac{dx}{dt}, x, y, u\right) = 0$$  \hspace{1cm} (4.2)

rather than assignments (4.1). An efficient code for equations will be generated automatically by simulation package e.g. Dymola [9]. No particular variable needs to be solved manually. The modelling effort is reduced considerably and tedious and error-prone manual manipulations are avoided. The elements of x are dynamic variables since their time derivatives $\frac{dx}{dt}$ appear in the equations. The elements of y are algebraic variables since none of its derivatives appears in the equations. Modelica is acausal and any variable may act as input, output or control. It means, same model can be used regardless of which terminal of the model is chosen to act as input or output.

Solving a DAE problem involves more than integrating to obtain x. The solution procedure involves also differentiation if the Jacobian with respect to $\frac{dx}{dt}$ and y is structurally singular. In order to be able to solve for y and $\frac{dx}{dt}$ it is then necessary to differentiate some equations.

### 4.3 Standard library and additions library of Modelica models

Modelica syntax is normally hidden from the user. Models of standard components are typically available from libraries received with Modelica package or obtained elsewhere. There are two free libraries distributed with Modelica: standard library and additions library. Both have a hierarchical set of models (fig. 11).

![Figure 11. Sublibraries of Modelica standard library and additions library of models](image)

Rotational sublibrary (fig. 12) belongs to Mechanics subset of standard Modelica library of models.
Figure 12. Rotational sublibrary, one of subsets of Modelica standard library of models

Each library model has fields, where all parameters needed for simulation are kept. User can assign actual value to each field or default values will be used.

For example, model of gear (fig. 13) from rotational sublibrary (fig. 12) has following fields:

- transmission ratio defined as $\text{gearRatio} = \frac{\text{flange}_a.\phi}{\text{flange}_b.\phi}$, default value is 1
- $\eta$ - gear efficiency due to friction between the teeth
- $\text{friction}_\text{pos}(w,\tau)$ - positive sliding friction torque function $\tau$ versus velocity $w \geq 0$
- $\text{peak}$ – maximum friction torque at zero velocity
- $d$ - (relative) gear damping [N.m.s/rad]
- $\text{elasticity}$ of gear (spring constant) [N.m/rad]
- total $\backslash$ [rad]

Input/output variables are defined on both sides of gear. There are: flange angle ($\text{flange}_a.\phi$; $\text{flange}_b.\phi$) and flange torque ($\text{flange}_a.\tau$; $\text{flange}_b.\tau$). Description of connector $\text{flange}_a$ was given above in chapter 4.2.

HyLib (Hydraulics library) and Powertrain library are options offered with Dymola package [9]. Other Modelica libraries (e.g. SystemDynamics, ThermoFluid, ExtendedPetriNet, FuzzyControl and many more) are advertised in Internet and may be downloaded free from http://www.modelica.org/.

4.4 Dymola

Dymola (dynamic modelling laboratory) [9, 22] is suitable for modelling of various kinds of physical objects. New modelling methodology is based on Modelica as standard modelling tool. The usual need for manual conversion of equations to a block diagram is removed by the use of automatic formula manipulation. Dymola handles large (e.g. 100 000 equations), multi-domain models and runs on Windows, Linux and Unix. Dymola supports:

- modelling by graphical model composition,
- simulation with symbolic pre-processing,
- user defined model components,
• interface to other programs including MATLAB, SIMULINK, Real Time Workshop and dSPACE,
• 3D Animation,
• real-time and HiL simulation.

Dymola may read and generate models and data in MATLAB format. Simulation models can be exported to SIMULINK and compiled by Real Time Workshop for on-line simulation using dSPACE equipment. Models from Dymola and SIMULINK can be merged to exploit advantages of both environments if large and complicated systems are designed.

5 Management of interdisciplinary design project

The mechatronic design process can be split into few general phases:
• requirement specification,
• analysis, conceptual and architecture design on high level of abstraction
• detail level of design, prototyping and testing
• implementation

5.1 Requirement specification, analysis, conceptual and architecture level design

UML diagrams and scenarios are very well suited to support design on high level of abstraction: requirements, analysis, conceptual design and architecture. Requirements elicitation phase may decide on success or failure of the project. Inconsistency and incompleteness of requirements specification are found on early stage of design, when project modification is relatively cheap and easy.

Analysis phase includes decomposition and architecture level design of the future system. Number and responsibility of subsystems and how are they connected is decided here. Those large-scale strategic decisions will affect all later steps of design.

5.2 Detail level of design, prototyping and testing

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 [12-14]. AutoCAD, Mechanical Desktop, ADAMS and other CAD/CAM software is used to design mechanical subsystems [5, 27]. Other specialized CAD tools are used to design electrical, hydraulic and other subsystems. MATLAB, SIMULINK and STATEFLOW are used for modelling, simulation and prototyping [30-36]. New possibilities of physical modelling are given by Modelica and Dymola, as described in this paper.

5.3 Integration example

An interesting example of integration CAD/CAM software (AutoCAD, Mechanical Desktop, ADAMS) with modelling and simulation environments based on Modelica is described in [27]. A mechanical model designed in the Mechanical Desktop saved in the DWG format. It contains all the information related to the geometrical properties of the parts and their mechanical assembly. Translator described in [4] uses this information to generate Modelica block diagrams. This model is edited in Dymola or MathModelica environment and simulated. Any extra blocks from Modelica or SIMULINK library, as well as custom block prepared by user, may be added to the model.

This approach solves problem of multi-domain design and simulation as models from other disciplines (DC motors, hydraulic elements, control) can be easily added when simulation model is edited. If simulation results do not satisfy requirements, part of the system is redesigned using respective tools and new model is simulated again (fig. 1). This procedure is repeated until all requirements are satisfied. Simulation with virtual model (as above) does not guarantee that final product will satisfy the requirements. This is mainly due to limited accuracy of models. Next step is on-line HiL (hardware in the loop) simulation, using physical
prototypes of chosen subsystems connected to special prototyping hardware (e.g. in dSPACE environment) and virtual model of rest of the system in computer memory. This important step of design is described in [23, 26, 30-35]. This example can be extended by using UML language in early steps of design.

5.4 Implementation

During implementation phase, prototyping equipment should be replaced by a target system which is normally cheaper, smaller, easier to operate and more reliable in industrial environment. Domain specialised CAD/CAM and CAE tools are very useful during both: detail design 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 [23, 33].

6 Conclusions

Generally accepted tools for modelling, simulation and design of interdisciplinary product are not known. Good candidate tools described in this paper are: UML language for design on high level of abstraction and Modelica for modelling, simulation and prototyping of interdisciplinary products on medium level of abstraction. This permits concurrent designing of all subsystems of future mechatronic product. It is important in mechatronic design, that all elements forming final product are treated as equally important during all design process, irrespectively of their physical nature. This helps to achieve synergy when several parts of different nature are integrated in one product [22, 30-312].

UML provides means for capturing system requirements and to design on high abstraction level of visual modelling. Unification and precision of notation is important for large and interdisciplinary projects. UML has facilities to help with capturing structure and relationships and rigorous definition of objects’ behaviour. UML reveals gaps and inconsistencies in the requirement's specification and description of dynamic behaviour of the future system at earlier stages of software design – when it is cheaper and less time consuming to correction the design. Using commercially available CASE packages, UML may improve productivity of design team by cutting down development time and improving final product quality (in accordance with ISO 9000 standards).

Modelica and Dymola form advanced environment for modelling, simulation and prototyping for complex physical systems. They offer several advantages over other packages:

- acausal modelling based on ordinary differential equations (ODE) and differential algebraic equations (DAE).
- multi-domain modelling capability, which provides the user with the possibility to combine electrical, mechanical, thermodynamic, hydraulic and other model components.
- object orientation and multiple inheritance facilitates reuse of components and evolution of models.
- virtual component models are designed by creating and connecting library and own components
- possible integration with MATLAB, SIMULINK, STATEFLOW and dSPACE.

Modelica is used for modelling and prototyping on medium abstraction level of design. Concurrent design of physical systems regardless of their nature permits synergic integration of all subsystems of the mechatronic product.

6.1 Acknowledgements:

The author wants to express his gratitude to ARTiSAN Software Tools, Inc (GB); Dynasim AB, Premium Technology Sp z o.o (Poland), ONT (Kraków, Poland) Rational Software Corporation (USA) and The MathWorks Inc(USA) for free evaluation licenses for computer software presented in this paper.
7 References:

15. MATLAB, SIMULINK, STATEFLOW, Real Time Workshop, Homepage http://www.mathworks.com/
22. Mrozek Z, Komputerowo wspomagane projektowanie systemów mechatronicznych, Zeszyty Naukowe Politechniki Krakowskiej, seria Inżynieria Elektryczna i Komputerowa, nr 1, 2002
23. Petko M. Implementacja układów sterowania w układachASIC/FPGA, Pomiary Automatyka Kontrola Nr 1, pp.18-21, 2002,
30. Uhl T System komputerowego wspomagania projektowania mechatronicznego, Pomiary Automatyka Kontrola Nr 1, pp.14-17, 2002
37. UML for Real Time System Design, ISG 1998