Int. J. Appl. Math. Comput. Sci., 2003, Vol. 13, No. 2, 255–267

COMPUTER AIDED DESIGN OF MECHATRONIC SYSTEMS

ZBIGNIEW MROZEK*

* Cracow University of Technology, Faculty of Electrical and Computer Engineering
ul. Warszawska 24, 31–521 Cracow, Poland
e-mail: zbigniew.mrozek@pk.edu.pl

Any successful company must react quickly to changing trends in the market. New products should be designed and manufactured quicker and cheaper than counter partners do. A shorter design time provides a distinct competitive advantage. The paper describes two approaches towards designing interdisciplinary mechatronic systems: the first is visual modelling with the UML, the second is physical modelling with Modelica.

Keywords: mechatronics, UML, physical modelling, Modelica, Dymola, Simulink, Stateflow

1. Introduction

UML (*unified modelling language*) is widely used in designing complex and reliable computer science. In mechatronics it provides means for capturing system requirements and for the visual modelling and design of systems on a high level of abstraction. Modelica is a freely available language for object oriented physical modelling. It may be used for modelling and prototyping a medium level of abstraction, as described in Sections 4 and 5.

Information transfer plays an important role in the operation of mechatronic systems. This can be easily presented on UML diagrams. The terminology and notation of visual modelling with UML can be adopted as a common high-level object oriented language for the design of mechatronic systems and as a documentation tool in every design phase (Mrozek, 2002b; Mrozek, 2002c; Mrozek, 2002a). An advantage of UML over other tools is that it reveals gaps and inconsistencies in the specification of requirements, at very early stages of the design. 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 the work progress and helps to keep all parts of the project in a consistent manner (Booch *et al.*, 1999; Bruegge and Dutoit, 1999; Douglas, 1998b; Douglas, 1998a; OMG, 2003). UML is supported by all major CASE (*computer aided system engineering*) tool vendors. Using UML notation, an experienced developer may design systems with fewer defects and quicker. The shorter design process (the time from the idea to the market) and better product quality provides a distinct competitive advantage. This is important, as a successful company should react quickly to changing demands of the market.

Many companies switch from paper blueprints to a digital representation of future products. Successful examples include some modern cars and the Boeing 777 plane (Sinha *et al.*, 2000). A weak point of the computer aided design process is the lack of widely accepted, integrated multidisciplinary software and hardware environments for successful design, testing, prototyping, implementation and validation. Instead, a sequential design approach is traditionally used: mechanical design at the beginning, then the system is extended with sensors, actuators and non-mechanical subsystems. They are integrated during modelling, simulation and the prototyping phase, when the control system is designed and tuned (Gawrysiak, 2002; Uhl *et al.*, 1999).

2. Computer Aided Design in Mechatronics

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

- cutting down cost and time needed to design and offer new products on the market,

- the integration of different subsystems and technologies at an early stage of the design,
- using virtual models (a code in a computer memory) and special electronic hardware for prototyping; this includes HiL simulation, described in (Uhl *et al.*, 2000; Uhl *et al.*, 2000),
- the compatibility of the design process with PDM (*product data management*) tools (Uhl and Śliwa, 1996).

A mechatronic system integrates mechanical, electrical and electronic subsystems with a microprocessor based control subsystem. An interdisciplinary approach means equal significance of all subsystems during the 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,
- the control subsystem is modelled and designed with multipurpose software packages such as MATLAB/SIMULINK (Mrozek and Mrozek, 2001). They are not directly applicable to the modelling of mechanical and electronic subsystems if the needed parts are not included in accompanying libraries or extensions (e.g. SIMULINK toolkits, SimMech, etc.). A novel approach is to use physical modelling with Modelica at this stage of design.

It is proved in practice that the mechatronic approach is successful in designing interdisciplinary products, even if the effective integration of activities in different disciplines is difficult. The development process consists of a set of partially ordered steps which lead to a desired target. The sequential approach always fixes the project status before switching from one discipline specific design tool to another. Then a special arrangement is needed to exchange project data between the tools used. The known solutions are extra software tools, protocols and interfaces for data exchange. The main disadvantage is that the above tools often work in the off-line mode only. In other words, multi-criteria optimisation (essential in mechatronic design) is very difficult to implement, as the parameters imported from other packages are fixed during some steps of optimisation. Nevertheless, many interdisciplinary products such as VCRs recorders, compact disk players or the ABS (anti-lock brake system) were successfully designed and introduced on the market (Uhl, 2002; Uhl *et al.*, 2000).

3. Using UML Diagrams in Mechatronic Design

UML was originally developed in response to a call for a proposal for a standardized object modelling language. Then it was improved many times until UML version 1.3 was accepted by the OMG (*object management group* (OMG, 2003)) as the proposal for a standard in the year 1999. New UML versions are under development.

UML was introduced as a language for the modelling of information systems but can be used to describe all elements of mechatronic systems on different levels of abstraction. Many attempts were made to extend UML applicability in areas beyond computer science. McLaughlin (McLaughlin and Moore, 1998) was probably the first to apply the UML approach to a process control problem that contains a conveyor belt transport subsystem. Using the UML for real-time systems is presented also in (Douglas, 1998b; Douglas, 1998a; Real-time Studio, 2003).

Any complex system can be represented by a set of carefully chosen models. A single model is not sufficient to describe real system. Use case and class diagrams (they are described later) are probably used in all UML supported projects. The choice of other diagrams to be created depends on how a problem is attacked.

3.1. Design is Part of the Product Life Time

The designing of mechatronic systems is an important part of the 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 an idea of the product and includes requirement specification, conceptual and detail design, prototyping and testing, implementation and validation, production, exploitation and recycling of products. Redesigning the model of a product (the model is prepared on a computer screen as UML diagrams) is supported with the CASE software. The best-known packages are Rational Rose (Rational Rose, 2003) and RtS (Real-time Studio, 2003). Using the UML helps to find and correct errors and omissions in requirement specification in a very early phase of the design, when models on a high level of abstraction are prepared. Some CASE tools offer simulation and animation of UML models. This helps to verify if all requirements are fulfilled. Simulation is even more realistic if some extra tools (e.g. Altia FacePlate for RtS (Real-time Studio, 2003)) are used to build a virtual operator console with animated dials and gauges.

Later CAD/CAM and CAE tools are used in a detailed design of subsystems. The parameters from detailed design are used in simulation models. Modelica, MATLAB, SIMULINK and other software are used to build

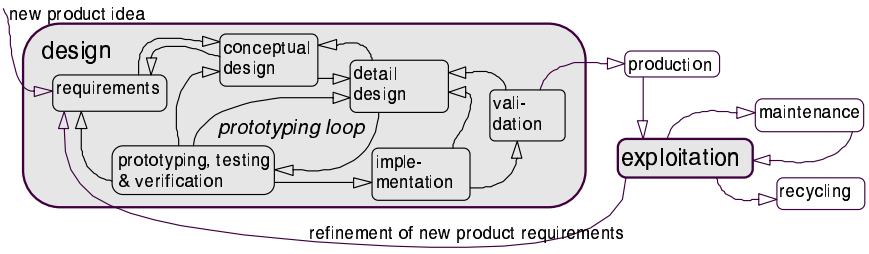

Fig. 1. Designing is an iterative process, an important part of the product life time.

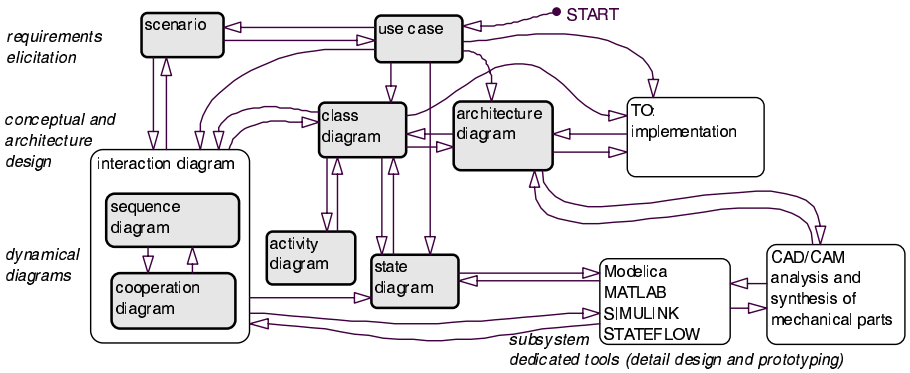

Fig. 2. UML diagrams are supposed to be built in predefined order. Other (domain specific) tools are used for detailed design and prototyping.

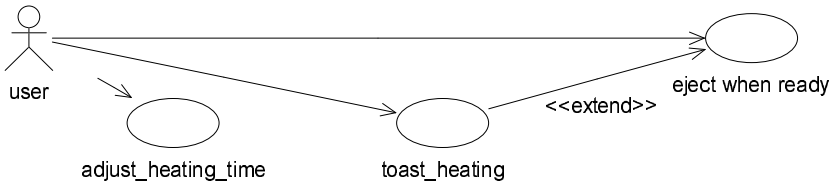

Fig. 3. Use case diagram for an automatic toaster.

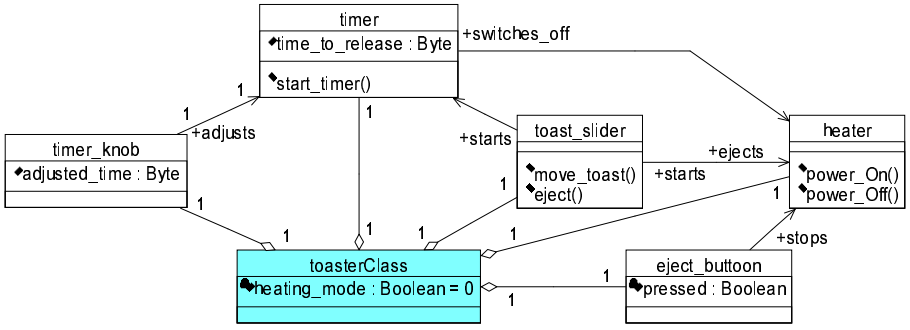

Fig. 4. Class diagram for the automatic toaster.

models for virtual and HIL (hardware in the loop) simulation. Simulation is used to tune and verify the behaviour of the designed product at a prototyping stage, before its physical model is prepared.

3.2. Suggested Order of Building Different UML Diagrams

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

The next step is to analyse use cases and scenarios to identify objects. This leads to a preliminary version of class and object diagrams. Later, objects and classes are used to build other diagrams. Class diagram may be redesigned if a new object, new method or new attribute is needed to build other diagrams. The sequence of actions described in scenarios is graphically presented on a sequence (Fig. 5) or a collaboration diagram (Fig. 6). Both are called interaction diagrams. All states that an object may go through are presented on a statechart diagram (Fig. 7 and 8). It describes the dynamic behaviour in response to events and fulfilled conditions. Parallel activities may be shown on the activity diagram (Fig. 9). The system architecture diagram (Fig. 10) should be used in mechatronic design instead of UML implementation diagrams. It is supported by RtS (Real-time Studio, 2003), a CASE tool from Artisan. Next steps, at a medium level of abstraction, are presented in Sections 4 and 5.

3.3. An Actor and a Use Case Diagram

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

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

Preparing use case diagrams is an important task, as the original problem description may be incomplete and some requirements may be in conflict with others. Both the client and members of the design staff should understand use cases. The user should verify if all the required functionality of the future system is included in the use case diagram and if all actors do communicate with the 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 the model). This is especially useful in real-time systems, where timing constraints for the latency of messages and processing time limits for operations should be followed. The OCL (*object constraint language* (OMG, 2003)), pseudocode, annotations or a 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 a systematic description of the sequence of messages sent between the actors and the system. Following the scenario describes how an actor prepares toast:

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

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

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

3.5. Class and Object Diagram

The class and object diagram shows the internal structure of a system. It defines the system structure by identifying

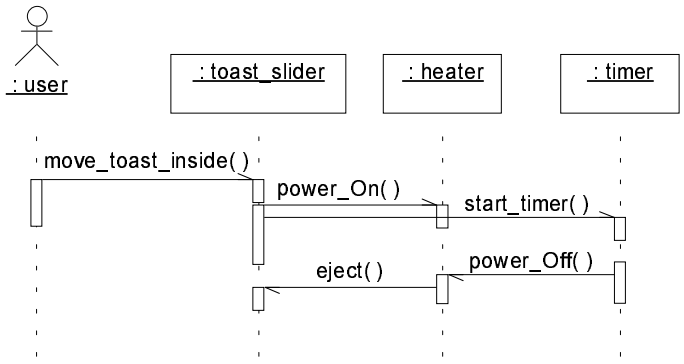

Fig. 5. Sequence diagram for the automatic toaster.

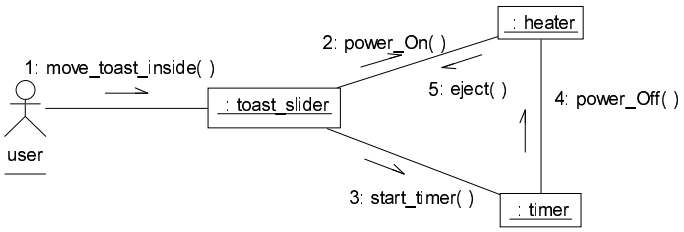

Fig. 6. Collaboration diagram for the automatic toaster.

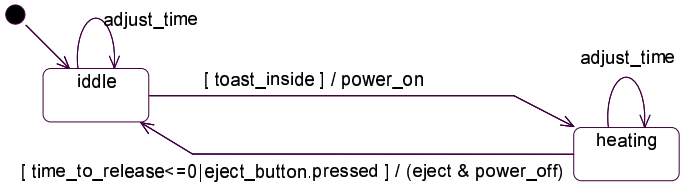

Fig. 7. Statechart diagram for the automatic toaster.

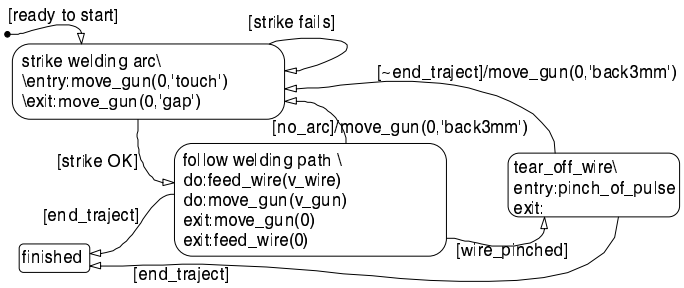

Fig. 8. Statechart diagram for the movement of a welding gun in a robotized arc welding system.

objects, defining their classes and relationships that exist between classes. Figure 4 shows the final version of the class diagram for the automatic toaster (Mrozek, 2002c).

Use cases and scenarios should be analysed to find objects, their names, responsibility, activities and parameters. One can identify objects as physical devices, sensors, actuators, interfaces, etc. If objects are known, a preliminary version of the class diagram can be built quite easily. The name of a class is given in the upper compartment of a rectangular icon, e.g. 'timer'. The values of the object's variables and parameters are kept in the attribute field, in the middle compartment, e.g. 'time_to_release'. Methods (services and responsibilities of a class) are given in the bottom compartment, e.g. 'start_timer'. Then if sequence, collaboration or any other diagrams is build or changed, new classes may be defined, or the existing classes may be updated. As a result, other diagrams, which use affected classes, may become invalid. CASE tools supporting UML programming (e.g. Rational Rose, 2003; Realtime Studio, 2003) 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.

3.6. Sequence and Collaboration Diagrams

UML provides sequence diagram and collaboration diagram to show graphically information from scenarios. Figure 5 shows a typical sequence diagram.

Objects are shown as horizontal rectangle icons on top of vertical time lines. Time flows down the line. An object or actor may send messages (shown as a horizontal line with an arrow) asking some services from another object (e.g. 'user' asks the 'toast_slider' object to 'move_toast_inside' the toaster). Activities shown on the sequence diagram may be annotated with text. Timing marks can be added to show exact time constraints.

The collaboration diagram (Fig. 6) shows a structural view of scenarios, but it provides essentially the same information as the sequence diagram. In the 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 because they react in a specific way to incoming events. A control engineer will find it helpful to describe the reactive behaviour of system states and to map the system logic to a physical architecture, e.g. to an FPGA chip.

A statechart diagram models the behaviour of reactive entities by specifying their response to receipt of events. It is used to describe the behaviour of class instances, but the statechart may also describe the behaviour of a use case, an actor, a subsystem, etc. (Booch *et al.*, 1999; OMG, 2003). When compared with the sequence diagram (it shows the chosen scenario in a 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 remains in the given state as long as it satisfies some condition or until it is fired by some event to another state. A black ball shows a start state. An end state (if it exists) is shown as a black disc in a circle. Transitions (lines with arrows, Fig. 7) connect various states on the diagram. The toaster has only two stable states and its statechart diagram is very simple. If the toast is put inside the toaster, it switches the power on and remains in the heating state unless conditions to transit to idle are fulfilled. If the heating time has elapsed ('time_to_release<=0') or if the eject button is pressed, the toast is ejected and the heating power is switched off. The toaster is idle and a new piece of toast can be started.

Statechart diagrams are very useful in verifying the functionality of more complicated products as arc welding system. If the welding gun is in the 'follow welding path' state (Fig. 8), it feeds welding wire (do: feed_wire) and moves along welding path (do: move_gun). It finishes if the end of the trajectory is reached. All exit conditions (here: 'exit:move_gun(0)' and 'exit:feed_wire(0)') should be fulfilled before leaving a given state. If the welding arc dies or if the welding wire is pinched, the state transits to the 'strike_welding_arc' state or to the 'tear_of_wire' state, respectively. This statechart diagram was prepared with the Stateflow (MATLAB, 2003; Mrozek and Mrozek, 2001; Mrozek, 2002a) software in the Matlab environment.

The activity diagram (Fig. 9) is used to visualise parallel activities and to show the sequence of internal states. It is well suited to describe the set of sequential and parallel actions when preparing the welding gun to work in the MIG/MAG welding mode. Short horizontal or vertical thick lines are used for the synchronisation of actions.

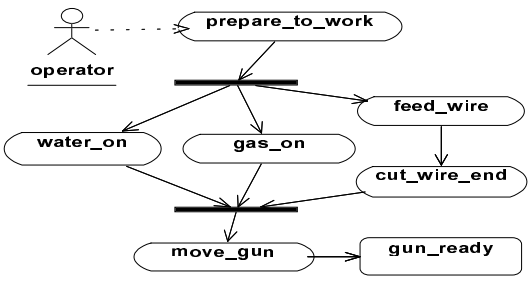

Fig. 9. Activity diagram for the preparation of the welding gun to work in the MIG/MAG mode.

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 a mechatronic system has been determined with a use case diagram, physical interfaces to all actors should be identified as part of the requirements for the system. The result can be presented on a system architecture diagram. This is an extension of the UML included in the RtS package (Real-time Studio, 2003). Figure 10 shows all main subsystems of the arc welding system. They are connected with the CAN-bus communication network. This includes the operator console with I/O devices (keyboard, joystick, LCD display), work piece positioners, a power source for the arc welding gun, a robot controller, wire and gas feeding systems. Later, more details can be assigned to each element of this diagram.

3.9. Remarks on Using UML

Generally accepted tools for the modelling, simulation and design of multidisciplinary product are not known. Good candidate tools described in this paper are: a UML language for design on a high level of abstraction and Modelica for the modelling and simulation of interdisciplinary products on a medium level of abstraction. If CASE tools (e.g. RtS (Real-time Studio, 2003)) are used, double click on any item of UML diagram opens its propert window. One can memorize its parameters and other information using this window. As the project is further investigated, COTS (*commercial off the shelf*) boards and subsystems (mechanical, electrical, etc.) are chosen and their parameters are added into the fields of properties window. When the requirements for any custom subsystems are defined, its data should also be memorised in properties window fields of a respective subsystem.

Brainstorming may be used to identify possible problem solutions and to find potential opportunities for improvements. This technique can be used to find all use cases and to prepare scenarios. Brainstorming is also useful to identify states and transitions for the statechart diagram (Mrozek *et al.*, 2002).

4. Physical Modelling, Simulation and Prototyping with Modelica

Modelica (Elmqvist *et al.*, 1998; Elmqvist *et al.*, 2001; Modelica, 2003) is a freely available language for hierarchical object oriented physical modelling, which is developed through an international effort. The work started in 1996 under the ESPIRIT project *Simulation in Europe, Basic Research Working Group* (*SiE WG*). The author of the present paper was involved in preliminary work of this group during his temporary stay at the Ghent University. The Modelica language unifies and generalizes previous object-oriented modelling languages and is intended to become a *de-facto* standard. It complements design with the UML on a high level of abstraction.

4.1. Integration of Models of Different Natures

In a modelling and simulation environment, it is desirable to integrate models specified in different modelling formalisms (ODEs, DAEs, PDEs, statechart diagrams, reusable library models, etc.) and to support multi-domain modelling (Astrom *et al.*, 1998). Modelica was designed with the objective of facilitating the exchange of models, model libraries, and simulation specifications. It allows us to define modularly the hierarchy of simulation models. The multi-domain capability of Modelica combines electrical circuits, multi-body mechanical systems, drive trains, hydraulic, thermodynamic and other domain model components. Interactions between components of different nature may be studied with Modelica. The capabilities of Modelica have been demonstrated by modelling and simulating mechanical systems (Bunus *et al.*, 2002; Clauß and Beater, 2002; Sinha *et al.*, 2000; Schlegel *et al.*, 2002), thermodynamic systems (Felgner *et al.*, 2002; Sinha *et al.*, 2000), automotive systems (Bowles *et al.*, 2001; Schlegel *et al.*, 2002), electrical (Torrey and Selamogullari, 2002) or hydraulic systems (Ferreira *et al.*, 1999; Schlegel *et al.*, 2002). The advantage of modelling is that the user can concentrate on the logic of the problem, rather than on detailed implementation of the simulation model.

Modelica supports modelling on a medium level of abstraction by composing library components into block diagrams. Missing library components may be designed by connecting library components into subsystems or by designing them from scratch using their description by ODEs and DAEs.

4.2. Physical and Acasual Modelling

Physical modelling is based on the relations between physical quantities. It 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 the block diagram. A connector describes all physical quantities involved in the cooperation of subsystems. The connector class `Interfaces.Flange_a` of a mechanical gear (Fig. 13) is defined with *Modelica* code given in Figure 14 (comments are given between the ‘quotation marks’).

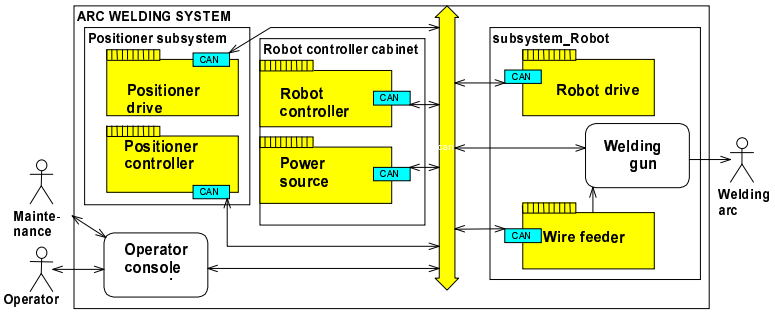

Fig. 10. System architecture diagram for a robotised arc welding system.

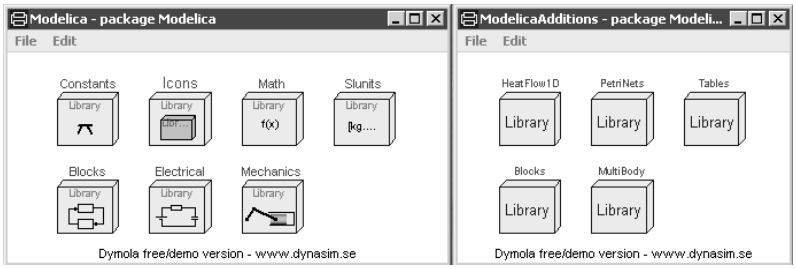

Fig. 11. Sublibraries of Modelica standard and additions libraries of models

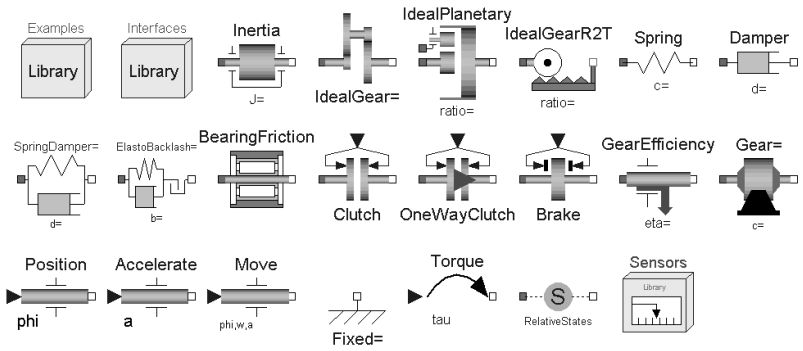

Fig. 12. Rotational sublibrary, one of the subsets of the Modelica standard library of models.

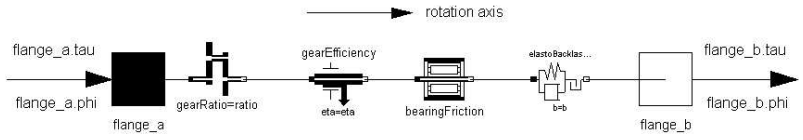

Fig. 13. Gear model from the rotational sublibrary.

```
connector Modelica.Mechanics.Rotational.Interfaces.Flange_a
  "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;
```

Fig. 14. Description of the connector class `Interfaces.Flange_a`.

Here each connection is used to generate equations for across (*equal*) variables

$$a1 = a2 = \cdots = an;$$

and the flow type (*through, zero-sum*) variables

$$z1 + z2 + \cdots + zn = 0.$$

Physical modelling offers a great improvement when compared with the traditional approach based on block diagrams, where artificial unidirectional signals at the inputs and outputs of block diagram elements are considered. Connector variables are physical quantities which have to satisfy respective physical phenomena. This idea is used also in Bond Graphs, where effort variables (e.g. voltages or rotation angles) are equal each other, and the sum of flow variables (e.g. currents or torques) equals zero. A similar approach is known in electrical engineering as Kirchoff's current and voltage law.

There are two variables from the *SIunits* library describing the *Flange_a* connector (Fig. 13, 14). The *tau* variable (torque in the flange, class name: *SIunits.Torque*) is of the *flow* type. This means that the sum of torque values on the connector is equal to zero. On the other hand, the *phi* variable (the rotation angle, class name: *SIunits.Angle*) is not of the flow type and the rotation angle values are equal on both sides of the flange connector. In an acausal modelling language (such as Modelica), the direction of the energy flow is not predefined and a connector may be bi-directional. If a connector definition is properly introduced, its power balance is satisfied.

The domain specific modelling tools are only strong in their own discipline and do not support components from other domains. Extra components may be often derived as a set of ordinary differential equations (ODEs) or differential algebraic equations (DAEs), preferably in a canonical form:

$$\mathrm{d}x/\mathrm{d}t = f(x,u), \quad y = g(x,u), \tag{1}$$

where t is time, x and y are vectors of unknown variables (state and output, respectively).

Modelica accepts more general declarative equations:

$$f(\mathrm{d}x/\mathrm{d}t, x, y, u) = 0, \tag{2}$$

rather than assignments. An efficient code for equations will be generated automatically by a simulation package, e.g. Dymola (2003). No particular variable needs to be determined 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 $\mathrm{d}x/\mathrm{d}t$ 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. This means that the same model can be used regardless of which terminal of the model is chosen to act as an input or an output.

Solving a DAE problem involves more than integrating to obtain x. The solution procedure involves also differentiation if the Jacobian with respect to $\mathrm{d}x/\mathrm{d}t$ and y is structurally singular. In order to be able to find a solution for y and $\mathrm{d}x/\mathrm{d}t$, it is then necessary to differentiate between some equations.

4.3. Standard and Additions Libraries of Modelica Models

The Modelica syntax is normally hidden from the user. The models of standard components are typically available from the libraries received with the Modelica package or obtained elsewhere. There are two free libraries distributed with Modelica: the standard library and the additions library. Both of them have a hierarchical set of models (Fig. 11). HyLib (Hydraulics library) and Powertrain library are options offered with the Dymola package (Dymola, 2003). Other Modelica libraries (e.g. SystemDynamics ThermoFluid, ExtendedPetriNet, FuzzyControl and many more) are available on the Internet and may be downloaded free from http://www.modelica.org/. Below are some examples from the hierarchical set of standard and additions library.

The rotational sublibrary (Fig. 12) belongs to the Mechanics subset of the standard Modelica library of models.

Each library model has fields where all parameters needed for simulation are kept. The user can assign a proper value to each field or default values will be used.

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

- transmission ratio defined as *gearRatio = (flange_a.phi/flange_b.phi)*, default value being 1,
- *eta*—the gear efficiency due to friction between the teeth,
- *friction_pos(w,tau)*—a positive sliding friction characteristic at the velocity $w \geq 0$,
- *peak*—the maximum friction torque at the zero velocity,
- d—the (relative) gear damping [N·m·s/rad],
- *gear elasticity* (spring constant) [N·m/rad]
- total *backslash* [rad].

Input/output variables are defined on both sides of the gear. They are the flange angle (*flange_a.phi*; *flange_b.phi*) and the flange torque (*flange_a.tau*; *flange_b.tau*). The description of the connector *flange_a* was given above in Section 4.2 (Fig. 14).

4.4. Dymola

Dymola (*dynamic modelling laboratory*) (Dymola, 2003; Mrozek, 2002a) is suitable for modelling various kinds of physical objects. The new modelling methodology is based on Modelica as a standard modelling tool. It supports hierarchical model composition and composite acausal connections. 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), complex 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,
- an 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 the MATLAB format. Simulation models can be exported to SIMULINK and compiled by the Real Time Workshop for on-line simulation using the dSPACE equipment. Models from Dymola and SIMULINK can be merged to exploit advantages of both the environments if large and complicated systems are designed.

5. Management of Interdisciplinary Design Projects

The mechatronic design process can be split into the following general phases:

- requirement specification,
- analysis, conceptual and architecture design on a high level of abstraction,
- detail level of design, prototyping and testing,
- implementation.

5.1. Requirement Specification, Analysis, Conceptual and Architecture Level Designs

UML diagrams and scenarios are very well suited to support requirements analysis as well as conceptual and architecture level design. Diagrams are used then to prepare a detail level of design with other tools and to test and verify prototypes and the final product. The requirement elicitation phase may decide against a success or a failure of the project. Inconsistency and incompleteness of the requirement specification should be found at an early stage of the design, when project modification is relatively cheap.

The analysis phase includes decomposition and the architecture level design of the future system. Large-scale strategic decisions about system implementation affect all later steps of the design. The number and responsibility of subsystems is decided, how they are interconnected, the concurrency model used, etc.

5.2. Detail Level of Design, Prototyping and Testing

A virtual model of the future system is designed during prototyping and the testing phase. Some UML diagrams can be adopted for the automatic generation of a virtual model of the future mechatronic system. The virtual model is then prototyped in real time, in an environment representing future working conditions of the final product (Felgner *et al.*, 2002; Ferreira *et al.*, 1999; Gawrysiak, 2002). AutoCAD, Mechanical Desktop, ADAMS and other CAD/CAM software is used to design mechanical systems (Bunus *et al.*, 2002; Sinha *et al.*, 2000). Specialized tools are used to design electrical, hydraulical and other subsystems. MATLAB, SIMULINK and STATEFLOW are used for modelling, simulation and prototyping (Uhl, 2002; Uhl *et al.*, 2000; Uhl *et al.*, 1999; Uhl *et al.*, 2000; Uhl *et al.*, 1999; Uhl *et al.*, 1998; Uhl and Śliwa, 1996). New possibilities of physical modelling are given by Modelica and Dymola, as described in this paper.

5.3. Integration Example

An interesting example of integrating the CAD/CAM software (AutoCAD, Mechanical Desktop, ADAMS) with modelling and simulation environments based on Modelica is described in (Sinha *et al.*, 2000). A mechanical model designed using the Mechanical Desktop is saved in the DWG format. It contains all the information related to the geometrical properties of the parts and their mechanical assembly. The translator described in (Bruegge and Dutoit, 1999) uses the information to generate Modelica block diagrams. This model is edited in the Dymola or MathModelica environments and simulated. Any extra blocks from Modelica or the SIMULINK library, as well

as custom blocks prepared by the user, may by added to the system.

This approach solves the problem of multi-domain design and simulation as models from other disciplines (DC motors, hydraulic elements, control) can be easily added when the simulation model is edited. If simulation results do not satisfy the requirements, part of the system is redesigned using respective tools and a new model is simulated again. This procedure is repeated until all requirements are satisfied. Simulation with a virtual model (as above) does not guarantee that the final product will satisfy the requirements. This is mainly due to limited accuracy of the models. The next step is on-line HiL (hardware in the loop) simulation, using physical prototypes of the chosen subsystems connected to special hardware (e.g. in the dSPACE environment) and the virtual model of the rest of the system in a computer memory. This important step of the design is described in (Petko, 2002; Schlegel *et al.*, 2002; Uhl, 2002; Uhl *et al.*, 2000; Uhl *et al.*, 1999; Uhl *et al.*, 2000; Uhl *et al.*, 1999; Uhl *et al.*, 1998). This example can be extended by using the UML language in early steps of the design.

5.4. Implementation

During the implementation phase, prototyping equipment should be replaced by a target system, which is normally cheaper, smaller, easier to operate and more reliable in industrial environments. Domain specialised CAD/CAM and CAE tools are very useful during both the detail design and the implementation phase. For example, digital electronic and computer electronic parts of the mechatronic system can be almost automatically designed in silicon as the ASIC (*application specific integrated circuit*) hardware using the FPGA (*field programmable gate arrays*) chips and software from Xilinx or Altera (Petko, 2002).

6. Conclusions

Generally accepted tools for the modelling, simulation and design of multidisciplinary products are not known. Good candidate tools described in this paper are the UML language for the design on a high level of abstraction and Modelica for the modelling, simulation and prototyping of interdisciplinary products on a medium level of abstraction. This permits concurrent designing of all subsystems of future mechatronic products. It is important in mechatronic design that all elements forming the final product be treated as equally important during the entire design process, irrespectively of their physical nature. This helps to achieve synergy when several parts of different nature are integrated in one product (Mrozek, 2002a; Uhl, 2002; Uhl *et al.*, 2000).

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

Modelica and Dymola (it uses models from Modelica) are advanced modelling languages for complex physical systems. They offer several advantages over other languages:

- acausal modelling based on ordinary differential equations (ODEs) and differential algebraic equations (DAEs),
- multi-domain modelling capability, which provides the user with the possibility of combining electrical, mechanical, thermodynamic, hydraulic and other model components,
- object orientation and multiple inheritance facilitates the reuse of components and the evolution of models,
- virtual component models are designed by creating and connecting a library and their own components,

The UML is used on high abstraction level of design and Modelica (probably integrated with SIMULINK and the dSPACE environment) for modelling and prototyping on a medium abstraction level of design. Concurrent design permits the integration of all subsystems of a mechatronic product.

Acknowledgements

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

References

Astrom K.J, Elmqvist H. and Mattson S.V. (1998): *Evolution of continuous-time modelling and simulation*. — Proc. 12-th European Simulation Multiconf. ESM'98, Manchester, UK, pp. 9–18.

Booch G., Rumbaugh J. and Jacobson I. (1999): *The Unified Modelling Language User Guide*. — Addison Wesley.

Bowles P., Tiller M., Elmqvist H., Brück D., Mattsson S.E., Möller A., Olsson A. and Otter M. (2001): *Feasibility of detailed vehicle modeling*. — Proc. SAE World Congress, Detroit, paper No. 01P-321.

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

Bunus P., Engelson V. and Fritzson P. (2002): *Mechanical Models Translation, Simulation and Visualization in Modelica*. — Proc. 2nd Int. *Modelica Conf*., DLR, Oberpfaffenhofen, Germany, pp. 199-208.

Clauß C. and Beater P. (2002): *Multidomain systems: Electronic, hydraulic, and mechanical subsystems of a universal testing machine modelled with modelica*. — Proc. 2nd Int. *Modelica Conf*., DLR, Oberpfaffenhofen, Germany, pp. 25–30.

Douglas B.P. (1998a): *Designing Real-time Systems with UML, Part I-III*. — Embedded Systems, No. 03-05, available at: http://www.embedded.com.

Douglas B.P. (1998b): *Real-time UML: Developing Efficient Objects for Embedded Systems*. — Reading, MA: Addison Wesley.

Dymola (2003): *Dynamic Modelling Laboratory*. — Available at http://www.dynasim.se/

Elmqvist H., Mattsson S.E. and Otter M. (1998): *Modelica—the new object-oriented modeling language*. — Proc. 12th Europ. Simulation Multiconf. ESM'98, Manchester, UK, pp. I–X.

Elmqvist H., Mattsson S.E. and Otter M. (2001): *Object-oriented and hybrid modeling in modelica*. — J. Européen des Systèmes Automatisés, Vol. 35, No. 1, pp. 127–131.

Felgner F., Agustina S., Cladera Bohigas R., Merz R. and Litz L. (2002): *Simulation of thermal building behaviour in modelica*. — Proc. 2nd Int. *Modelica Conf*., DLR, , Oberpfaffenhofen, Germany, pp. 147–154.

Ferreira J.A., de Oliveira J.E. Costa, V.A. (1999): *Modeling of hydraulic systems for hardware-in-the-loop simulation: A methodology proposal*. — Proc. Int. *Mechanical Eng. Congress & Exposition*, Nashville, USA, Vol. 6, p. 33.

Gawrysiak M. (2002): *Stages of mechatronic design according to R. Isermann*. — Proc. Workshop on Mechatronic Design, Cracow, pp. 7–15, (in Polish).

MATLAB, SIMULINK, STATEFLOW (2003): *Real Time Workshop*. — Available at http://www.mathworks.com/

McLaughlin M.J. and Moore A. (1998): *Real-time extensions to UML, timing, concurrency, and hardware interfaces*. — Dr. Dobb's J., No. 12, pp. 7–15.

Modelica®(2003): *A unified object-oriented language for physical systems modeling, language specification, version 2.0*. — Available at http://www.Modelica.org/

Mrozek B. and Mrozek Z. (2001): *MATLAB 6, User's Guide*. — Warsaw: Wydawnictwo PLJ, (in Polish).

Mrozek Z. (2002a): *Computer-aided design of mechatronic systems*. — Sci. Fasc., Cracow Univ. of Technol., Series: Electrical and Computer Eng., No. 1, (in Polish).

Mrozek Z. (2002b): *Design of the mechatronic system with help of UML diagrams*. — Proc. 3-rd Workshop *Robot Motion and Control*, Bukowy Dworek, Poland, pp. 243–248.

Mrozek Z. (2002c): *Methodology of using UML in mechatronic design*. — Pomiary Automatyka Kontrola, No. 1, pp. 25–28, (in Polish).

Mrozek Z., Mrozek B., Adjei O. (2002): *Teaching object oriented software engineering with UML*. — Proc. 13-th Ann. Conf. *Innovations in Education for Electrical and Information Eng*., York, (in English).

OMG Unified Modeling Language Specification v.1.4, (and other OMG documents), Available at http://www.omg.org, 2003.

Petko M. (2002): *Implementation of control systems in ASIC/FPGA circuits*. — Pomiary Automatyka Kontrola, No. 1, pp. 18–21.

Rational Rose (2003): Rational Rose RT (and other software from Rational Software Corporation), Available at http://www.rational.com/, 2003.

Real-time Studio (2003): ARTiSAN Software Tools, Inc., Available at http://www.artisansw.com/, 2003.

Schlegel C., Bross M., Beater P. (2002): *HIL-Simulation of the hydraulics and mechanics of an automatic gearbox*. — Proc. 2nd Int. *Modelica Conf*., DLR, Oberpfaffenhofen, Germany, pp. 67–76.

Sinha R., Paredis C.J. and Khosla P.K. (2000): *Integration of mechanical CAD and behavioral modeling*. — Proc. Int. Workshop *Behavioral Modeling and Simulation*, Orlando, Florida, pp. 31–36.

Steinmann W.D. and Zunft S. (2002): *A library for modelica applications in technical thermodynamics*. — Proc. 2nd Int. *Modelica Conf*., DLR, Oberpfaffenhofen, Germany, pp. 217–224.

Torrey D.A. and Selamogullari U.S. (2002): *Modelica implementation of field-oriented controlled 3-phase induction machine drive*. — Proc. 2nd Int. *Modelica Conf*., DLR, Oberpfaffenhofen, Germany, pp. 173–182.

Uhl T. (2002): *A system of computer-aided mechatronic design*. — Pomiary Automatyka Kontrola, No. 1, pp. 14–17, (in Polish).

Uhl T., Bojko T., Mrozek Z. and Szwabowski W. (2000): *Rapid prototyping of mechatronic systems*. — J. Theoret. Appl. Mech., Vol. 38, No. 3, pp. 655–668.

Uhl T., Bojko T., Mrozek Z., Petko M., Szwabowski W., Korendo Z. and Bogacz M. (1999): *Selected problems of mechatronic design*. — Techn. Report 7T07B 01414, Chair of Robotics and Machine Dynamics, Acad. Mining & Metallurgy, Cracow, (in Polish).

Uhl T., Mrozek Z. and Petko M. (2000): *Rapid control prototyping for flexible arm*. — Prepr. 1st IFAC Conf. *Mechatronic Systems*, Darmstadt, Vol. II, pp. 489–494.

Uhl T., Bojko T. and Mrozek Z. (1999): *Mechatronic approach towards designing and implementing of control systems*. — Proc. 1st Workshop *Robot Motion and Control*, Poznań, Poland, pp. 135–140.

Uhl T., Bojko T., Mrozek Z. and Szwabowski W. (1998): *Control of a flexible robot arm – fast prototyping and implementation*. — Proc. 1st Nat. Workshop *Rapid Prototyping Methods*, Acad. Mining & Metallurgy, Cracow.

Uhl T. and Śliwa Z. (1996): *Selected topics in CAD/CAM systems and their applications*. — Cracow Centre for Advanced Training in Information Technology, CCATIE, Cracow.

UML for Real Time System Design, ISG 1998, available at: http://www.isg.de/people/marc/UmlDocCollection/UMLFrReal-TimeSystemsDesign/umlrt.html/

Received: 4 September 2002
Revised: 16 January 2003