Conceptual Design for Controller software of mechatronic systems

Dr. ir. Jan F. Broenink, ir. Gerald H. Hilderink and prof. ir. André W.P. Bakkers
University of Twente, Faculty of Electrical Engineering, Enschede, Netherlands
e-mail: (J.F.Broenink, G.H.Hilderink, A.W.P.Bakkers}@el.utwente.nl

Abstract

The method and software tool presented here, aims at supporting the development of control software for mechatronic systems.

Heterogeneous distributed embedded processors are considered as target hardware. Principles of the method are that the implementation process is a stepwise refinement from control laws to efficient computer code and that all phases are verified by simulation. Simulation is also used as verification tool during physical–system modelling and control law development.

Data flow diagrams are used as description language for controller implementations, whereas the physical–system models are expressed in bond graphs. Future work comprises the building of the complete tool, although individual parts are ready.

1. Introduction

Present-day requirement for reliable and easy to change software for embedded systems increases the need for software development tools. Especially when the computer power is distributed among different kinds of processors. Currently, computer tool support exists on one hand for system design, for modelling and for controller design and on the other hand real-time software design. Unfortunately, these two groups of methods are too far apart to be easily integrated, and they do not really support heterogeneous distributed computer architectures.

With the tool we are developing, we aim at protecting the user (mechatronic engineer) from needing skills on programming the target hardware and interface devices (e.g. parallel programming, interrupt programming, inter-processor communication, real-time and priority scheduling), and yet get results equivalent to those obtained by an experienced software design engineer. Furthermore, the transformation process to convert the algorithms for the control,
i.e. mathematical formulae, to efficient computer code and possibly the concurrent implementation of the algorithms, is guided via a stepwise refinement procedure. After each refinement step, the results are verified by simulation. This refinement procedure has been prototyped in [1, 2].

As target hardware, a heterogeneous distributed embedded computer system is considered. *Heterogeneity* is necessary, since on the one hand hardware developments are such fast that ongoing adaptation of the software tool is necessary. On the other hand, it is not sure that promising chips or architectures will stay available in the market for the complete software tool lifetime. Hence, sophisticated control computer systems often comprise of different computer types. It is necessary to study *distributed* computer systems, because control systems of industrial size are such complex that sophisticated support tools are inevitable. Furthermore, systems must be easily *scalable* and *adaptable*, to support the ever changing functional specifications.

The tool described here will be implemented as a part of the 20-SIM modelling, analysis and simulation tool [3, 4, 5]. Thus, the user will have the assistance for the stepwise refinement from control algorithms to computer code for the target hardware.

In the next section, we will discuss the objectives of our method. In the sections thereafter the following parts are treated: Physical systems modelling and Control law design in section 3. Embedded system implementation, the main theme of this paper, is treated in section 4. Realisation of the system is discussed in section 5 and a description language in section 6. We conclude with an example (section 7) and conclusions (section 8).

### 2. Objectives

To provide adequate support during the design process, a design method should be tailored to the type of systems for which it is intended. In our case, the dynamic properties of the total system, and not just the controller to be designed, plays a central role. The controller being designed should contain all aspects that influence the behaviour of the total system. Thus, a correct implementation of the controller’s functional description on the control computer will display the desired behaviour. A major part of the design process is to be automated to avoid errors in the implementation.
The main objective of our tool is to support the user (mechatronic engineer) at implementation of control systems, to eliminate design and coding errors and to diminish development costs. This is accomplished by [2]:

- **Verification and validation.** After each refinement step in the implementation procedure, the result is checked by simulation. This implies that all intermediate system descriptions must be executable.

- **Mechatronic design approach.** The system is no longer developed sequentially in separated parts that are integrated in the end, but the system is treated as a whole. Design decisions may influence different parts simultaneously.

- **Separate development of reusable parts.** Complete controllers can be built by assembling existing parts, thus accelerating development. This currently rather common feature stems from object–oriented design.

As is quite common for contemporary Computer–Aided Control System Design software methods, **abstraction, partitioning** and **hierarchy** can reduce the complexity of a complete system. The system is partitioned into a hierarchical set of modules. During the design process, the level of detail will change: during control law design, the A/D converter is assumed to be ideal, while during the implementation phases, extra detail is added, to more precisely describe the behaviour.

The complete design trajectory of controllers comprises the following four parts (see fig1)

- **Physical Systems Modelling.** The dynamic behaviour of the system is object–orientedly modelled, using bond graphs as a main modelling paradigm.

- **Control law Design.** Using the model acquired in the previous step or a simplified version of it, control laws are designed. Currently, external software (Matlab) is used.

- **Embedded System Implementation.** Transforming the control laws to efficient concurrent algorithms (i.e. computer code) is guided via a stepwise refinement process. After each step, the results are verified by simulation. [1, 2].

- **Realisation.** The realisation of the control system is also worked on as a stepwise sequence. Parts of the system stay as models while other parts are coded on their target hardware. Besides catching variation in development time of parts of the system, also additional verification can be done.
In the following sections, these four parts are discussed in more detail. Since the main theme of this paper is on controller software design, the *Embedded System Implementation* and *Realisation* parts are treated more thoroughly.

## 3. Physical Systems modelling & Control law design

### 3.1. Modelling paradigm

The physical system, which is to be controlled, is modelled in an *object-oriented* way, using bond graphs as a main modelling paradigm [6, 7, 8, 9]. Since we focus on complex engineering systems, we need a description method, which can span multiple engineering domains and the information domain. Furthermore, hierarchically structuring of models is necessary, since models are of non–trivial complexity. This also implies that encapsulation of model details should be provided.

Furthermore, hierarchically structured models in general require that the contents of the submodels need be known *exactly*, because these contents and the interconnection of the submodels mutually influence each other. Encapsulation of the internals of a submodel can only be granted if such an influence is *not* present. For physical systems models, encapsulation may be granted when:

- The interface connections (*ports*) are defined as bilateral pairs of variables. Using power conjugate pairs of variables converts these interconnections to physical connections. Examples are voltage and current for the electrical domain and force and velocity for the mechanical domain.
• The submodel formulae are written in a declarative style, i.e. as real equations in the mathematical sense. Equations define relations and not a procedure for computation. This is often called a non-causal model description. This implies that a model-processing phase is required to convert the equations to assignment statements, which can be done after a model is constructed completely.

With models for information processing, like digital controllers, that consist of just signal inputs, signal outputs and computable algorithms, one does not have problems using encapsulation, since there is no choice for input or output.

Bond graphs [6, 7, 8, 9] are used for the physical systems modelling part, because both prerequisites mentioned above are fulfilled. Bond graphs are a direct representation of physical processes, domain independent, and they may be mixed with block diagrams in a natural way to cover the information domain part. Bond-graph submodel interfaces use power conjugate, bilateral pairs of variables, such that the interconnections denote physical connections. Model processing from equations to assignment statements is done efficiently, using a computational causal analysis on graph level (i.e. the graphical representation of the structured model). Before generating any assignment statement, feedback to the modeller can be given on the presence of dependent state variables (sometimes called semi-states) or algebraic loops, which are in fact hints on having abstracted too much during modelling. Another advantage is that the resulting simulation model is of less or equal numerical complexity then when using general assignment statements generation techniques [10]: The index of nilpotency of the resulting set of differential and algebraic equations (DAE) is either 0 or 1.

3.2. Modelling method

Obviously, the purpose is to create a competent model of the system under study. So, only relevant and dominant aspects with respect to the modelling goal, i.e. controller software design, need to be considered. This makes the modelling process iterative, putting an extra demand on model extendibility and maintenance. Feedback does not only come from verification by simulation, but also from the other design phases (Figure 1).

Note that in this phase, models must be created for two different purposes, namely understanding the dynamics of the physical system and deriving control laws. Sometimes one
model can serve both goals, but mostly the model used to derive a control law from, is a simplified, often linearised version of the model used for understanding the dynamics.

We therefore phrase the following, rather common, procedure:

1. **Generate a detailed model**
   This model is a rather detailed model of the physical system, in order to understand the dynamic behaviour of that system. It is a kind of physical–system replacement: It is later used as a substitute for the physical system when the control laws and control algorithms are tested.

2. **Verify the detailed model**
   Simulate the detailed model to check whether the model satisfies its goals. If the total model is rather complex, first parts of it may be tested separately. If possible, the model can be validated (i.e. compared with measurements on the real system), to check the correctness, and also whether the assumptions are applicable in the particular situation.

3. **Derive a simplified model**
   Starting from the detailed model, either reduce it automatically (p.e. by linearisation and/or order reduction) or diminish it by hand to obtain a model suitable for control law design. It may be necessary that more than one simplified model is needed to cover the whole workspace of the control system.

4. **Verify the simplified model(s)**
   Verify the simplified model by performing the same simulations as with the detailed model. Results should not differ significantly.

5. **Derive the control law(s)**
   Derive control laws, using the simplified model(s) acquired in the previous steps. This is common practice in controller design. Currently, external software such as Matlab is used.

6. **Verify the control law(s)**
   Construct a test bed in which the control law is connected to the detailed model. Verify the control laws by performing simulations. Run such experiments that the demands on the controller performance can be checked.

Arriving at this stage, the control law(s) together with the detailed model can be used in the process of embedded system implementation.
4. Embedded system implementation

After the control law has been designed and verified by simulation, it needs to be implemented on the embedded control computer. Classically, this work is one single step in the design process, thus resulting in a gap between the control law design and the final product. Here, we propose to structure this implementation process by introducing the following procedure [1, 2]:

1. **Integrate control laws and user requirements**
   
   Control laws for different situations are combined i.e. the necessity of mode switching is determined. Furthermore, reaction to external events is added. Note that the source of the trigger can be either a user-driven action or some internal constraint being met, e.g. a robot hitting its own base.

   The sensors, actuators and algorithms are assumed to be ideal.

2. **Add technology–independent functionality**
   
   The specification is augmented with the non–idealness of sensors and actuators. For example, if not all internal process variables can measured directly, estimators may be added to determine the variables. Furthermore, facilities for safety and maintenance processing are added. Also the operation of sensing and actuating is no longer considered ideal and faultless.

3. **Add technology–dependent functionality**
   
   Characteristics of the selected devices are added to the description, e.g. delays in AD-converter and finite resolution of interfaces.

   After this step, the dynamic behaviour of the system is completely described provided that the real–time requirements can be fulfilled.

4. **Add timing characteristics**
   
   The control computer hardware architecture and the software architecture are added. Scheduling techniques and/or algorithm optimisation techniques maybe used to obtain a viable realisation.

   Note that when algorithms are optimised for performance, the computer hardware architecture should be known in order to really optimise the algorithms for the embedded computer.

This procedure does not prescribe the order in which the development must take place, but instead the designer has the freedom to tackle the individual subproblems in any order. This is
a major difference with the traditional waterfall lifecycle paradigm. For example, a top–down decomposition may be applied first to define the global architecture of the system, after which those control algorithms in which problems are expected may be developed. Also parts of the controller can be developed incrementally and combined to obtain the description of the total controller. In short, the designer has the option to apply the most appropriate technique to each problem.

The implementation of the embedded system is a process of stepwise refinement from control laws to control algorithms, a specification from which computer code for the target processors (i.e. embedded computers) can be generated automatically and straightforwardly.

5. Realisation

After the control algorithms have been derived as a result of the refinement procedure of the previous step, one can work towards realisation on the target computer and physical device.

In order to make a stepwise approach possible the real embedded system is divided into three main parts:

- The embedded computer and its software
  This consists of all computer functionality, including user interfacing with control panels.

- I/O interfacing
  The interface boards to connect the physical process to the computer. Specific operating–system resources like device drivers running on these boards belongs also to this part. The software running on these boards specific for the embedded system implementing a part of the control algorithms is allotted to the control software part.

- The device itself
  The physical process to be controlled: actuators including power amplifiers, the plant itself and the sensors with their specific signal processing hardware.

In the procedure of transforming the results (i.e. models) of the implementation step to the real embedded system, any of the three parts can be realised first, leaving the other two as simulated models. Which one to chose first depends on the availability of the target hardware, the complexity of the part, the quality of the model, etc.
A next step is to have two parts as real system parts and only one simulated. It depends on progress of the realisation phase whether this intermediate step is useful. Its must contribute to the realisation phase either as an efficiency enhancement or as an extra check to be sure that the complete system will work right the first time.

We illustrate this with two examples:

- In a concurrent design setting, the device might not be ready when the embedded software is available for testing. Using the model of the device instead of the device itself allows the control computer hardware and software to be tested before the device is available.
- In a situation where the embedded computer will be a dedicated processor, or even an ASIC, the embedded software needs to be optimal. Testing the embedded software running on a ‘normal’ computer coupled to the real I/O and the real device, can be a step in fine-tuning the embedded software before it actually gets ‘burned’ into the embedded computer. Often, optimisation of the code towards minimal size is necessary to be able to use the smallest processor possible.

Note that the test set up as used in the first example can also be used in a training situation.

6. Description language

The way to express all the necessary information in an efficient manner implies that a graphical language, showing the aspects shown in the model concurrently, is favourite. In principle, the structure of the embedded system can best be explained in a graphical form, while the algorithms may better be specified in textual form. The following aspects should be distinguished:

- Information flow and information processing. Information flow shows relations between elements, and is therefore shown by the edges in a graph model. Information processing is a property of an element: the vertices of the graph represent them.
- Data processing versus mode-switching. Data processing is the normal operation of a controller: a controller processes its input data until an event occurs. This event may lead to a model switch, representing a discontinuity in the operation that changes structure of the data processing. Since the influence of mode switching is extensive, separate symbols are needed.
The graphical notation of Modern Structured Analysis [11] was chosen because the distinctions mentioned above are covered and it is a well-known method. The notation shows a controller as hierarchical graphs called activity diagrams. These define the relationships between the particular components. At the end of the hierarchy, when further decomposition does not help understanding, the processes are not decomposed and they are now described by element descriptions. Basically, two forms of element descriptions exist: process specifications for basic data processing and control specifications for the mode switching. At the top of the hierarchy, the connections to the environment are specified in the context diagram, a graph with terminators symbolising external connections. See Figure 2.

![Figure 2: Overview of the description language](image)

The use of this language for controller specification will be shown in the next section.

### 7. Example

This example illustrates the use of both the implementation method and the description language. The example concerns the redesign of the safety layer of OSCAR-6, a 6 DOF anthropomorphic rigid industrial robot manipulator for research activities [12] (i.e. design and implementation of control algorithms and path planners). The OSCAR-6 protection should provide a fail-safe robot system, so that it will not be damaged by faults in one or more subsystems or by misuse of its functionality. A redesign of the protection system was
necessary. It had to be fail–safe for system faults like broken cables and malfunctioning hardware and software and it had to be fail–soft for faults in user controllers.

The protection software was partitioned using the general architecture of protection systems as a guide. The result is shown in Figure 3. The design process consisted of creating successive activity diagrams and the corresponding control specifications. The final elementary processes were so simple that they were described by a few lines of pseudo code. Simulation of the specification was done in the standard simulator of 20-SIM, since no special tool is available yet. Via these simulations, the parameters of the safety system were tuned to obtain the required behaviour.

Figure 3 OSCAR-6 protection, activity diagram top level

In Listing 1 the automatically generated Java code has been illustrated, using the *Communicating Java Threads* library [13]. The program shows the definition of the flows as channels and the start up of the individual processes as Java threads. The channels synchronise the execution of the different processes.

This case study led to the following results (see also [2]):

- A well–structured protection system was created.
- The design method provided a good overview during the complete design process, according to the student who carried out the work.
- During the design process, simulations helped to determine the effect on specific design options.
The translation from the graphical description to the target language was straightforward yielding good code quality and efficiency.

Listing 1 OSCAR-6 protection Java code

8. Conclusions

A method for design and implementation of controller software for embedded mechatronic systems has been elaborated. Furthermore, a graphical description language to specify controller implementation has been presented.

The software tools to support this design procedure are partly available. For modelling the physical system, simulation of the physical system and embedded system the tool 20-SIM has been used [3, 4, 5]. For control law design existing tools (Matlab, MatrixX) are used. A sophisticated interface between Matlab, MatrixX and 20-SIM is being developed. The software
support for the embedded system implementation part needs to be built. However, some essential procedures were prototyped. Facilities to support hardware–in–the–loop simulation for the realisation phase need to be developed completely.

Obviously, future work is to build the complete tool. Furthermore, we plan to mature the method via extensive use and testing. Extensions to the tool to support the design of a mechatronic system completely (i.e. also including the functional design and device design) are to be considered when the software tool as presented here is mature enough.

References


