Proceedings PROGRESS 2000 Workshop on Embedded Systems, pp 27-33, 13-10-2000, Utrecht, Technology Foundation (STW).

Building Blocks for Embedded Control Systems

Jan F. Broenink, Gerald Hilderink, André W.P. Bakkers

Control Laboratory, Electrical Engineering Faculty, University of Twente
P.O.Box 217, 7500 AE, Enschede, the Netherlands

Phone: +31 53 489 2793 Fax: +31 53 489 2223
E-mail: J.F.Broenink@el.utwente.nl

Abstract – **Developing embedded control systems using a building-block approach at *all* the parts enables an efficient and fast design process. Main reasons are the *real* plug-and-play capabilities of the blocks. Furthermore, due the simulatability of the designs, parts of the system can already be tested before the other parts are available.**

We have applied an object-oriented approach for modeling *all three* parts of embedded control systems: *compositional programming* for the embedded software parts; *VHDL* for the specific I/O computer hardware parts; *and bond graphs* for the appliance (i.e. the device to be controlled).

Due to the simulatability, our building-block method is suitable for a concurrent engineering design approach, and thus supporting hardware-software co-design.

Currently, our research focuses on the software description part. The appliance description part has been worked on, while we just use VHDL descriptions for the computer hardware. This is the reason why the emphasis in this text is on the building blocks for the embedded software part.

Keywords – **Embedded Systems, Building Blocks, Compositional Programming, Control.**

I. INTRODUCTION

Today's embedded systems are widely used from household appliances to control of chemical plants, nuclear power stations, command and control systems or the control of laboratory experiments. Applications spread rapidly to the control of automated production machines, robots, numerically controlled machines and also to transportation such as space shuttles, airplays, trains and automobiles (fly/drive-by-wire). The applications have as such an impact on everyone's daily life. However, the available technology to implement these systems is not advancing at equal pace. Computer-based control systems are still being implemented using the same techniques as ten years ago.

The complexity of modern embedded systems together with the absence of appropriate software tools, is one of the main reasons for the large number of errors in the design and implementation of these systems. Moreover, exhaustive testing of these systems is impossible, because of the combinatorial explosion of the possibilities.

Industry asks for a short time to market and easy product diversification and updation, in order to stay in competition. Flexible system specification with easy modification and reuse facilities are therefore necessary. Using building blocks at all parts of embedded systems can provide for these demands. We focus on using building blocks for embedded system design that can meet these demands from industry.

In section two, we elaborate on embedded systems, and distinguish between embedded control systems and embedded data systems. Our work focuses on the former. Section three deals with the building blocks, and closes with a case. Section four shortly mentions building block libraries. Section five deals with simulation as a verification tool in embedded system implementation, and discusses a development procedure using stepwise refinement.

II. EMBEDDED SYSTEMS

Embedded systems (ES) have their computer power completely integrated and dedicated to the specific functionality of its appliance, and have specific interfacing hardware to connect the appliance to (see Figure 1). We distinguish two types of embedded systems, namely *embedded control systems* (ECS) and *embedded data systems* (EDS).

Figure 1: Architecture of an Embedded Control System

At Embedded Control Systems, the dynamic behavior of the appliance (i.e. the 'machine'-part of the embedded system) is essential for the functionality of the ES. Computational latency must be small compared to the time constants of the appliance. The central control loop is *hard real-time*, because missing dead lines means a system failure. Examples are robots, production machines like wafer steppers.

At Embedded Data Systems, the behavior of the appliance can competently be described by waiting times between subsequent commands from the software. Com-

putational latency must be small compared to the reaction time of a (human) user. Missing deadlines decrease the quality of service, but are not fatal. Examples are (cellular) telephones, and other telecom systems.

In this research, we focus on Embedded *Control* Systems.

Embedded Control Systems

Due to the specific character of embedded control systems, the properties of the constituting parts are as follows:

- Software
  Principal functions are user interfacing, data processing and appliance control. Especially for appliance control, the software needs to be reliable and safe. Furthermore, its timing needs to be fully guaranteed.
- Hardware
  Here we mean both the computer hardware and the I/O interfacing. Often, specific processors are used (ASICs, DSPs, MCUs), and also sensors & actuators get integrated with the processor on one chip. There is a trend towards applying more programmable devices such as FPGAs to be more flexible during the design, but also to create possibilities for upgrading.
- Appliance
  The machine part of the ECS, e.g. a robot including its actuators (motors). As said before, the dynamic behavior of the appliance is a crucial part of the overall behavior of the ECS.

In fact, the ECS embodies a *closed-loop* control system, where the control loop is spread out over the embedded computer and appliance (Figure 2). The time constants of the appliance dictate the timing constraints of the software.

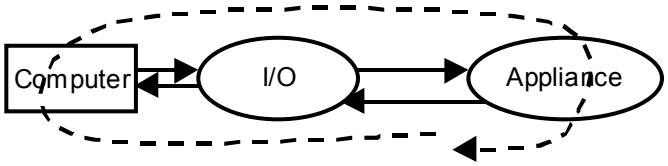

Figure 2: Control loop in an ECS

This specific character of ECS means, that for optimal software, the *complete* ECS needs to be considered. That is why we have applied a building block approach to *all* parts of an ECS.

III. BUILDING BLOCKS

Building blocks for complex systems need to comply with the following demands:

- *Overview* of the system description consisting of building blocks must be guaranteed. By allowing hierarchy, and indicating that as such, overview can be maintained.
- *Reusability* of the blocks need to be sufficiently high, to allow for competitive fast development. This requires well-designed interfaces, and the connection of blocks may not influence the description of the blocks itself.
- *Simulate-ability* of the total description, to allow for checking alternative solution proposals during design. In such a way, a real system's approach is possible, such that the system can be designed to function optimal in a global sense. Furthermore, a concurrent engineering attitude is possible.

In order to accomplish these demands, we use an object-oriented approach for *all* three parts of the ECS. This is possible since object-orientation allows for hierarchy and encapsulation. However, the description methods specific for the three ECS parts need to support this, and it has to be possible to combine those descriptions in order to reason about the complete ECS.

The object-oriented approaches for modeling *all three* parts of embedded control systems are:

- *Compositional Programming Techniques* for the embedded software parts, using *CSP-based channels* for information exchange between processes [1].
- *VHDL* for the specific I/O hardware parts, which remain configurable when using FPGA's
- *Bond Graphs* (directed graphs describing both the dynamic structure and dynamic behavior of the device) for the appliance to be controlled [2].

It appeared that these three description methods do support hierarchy and encapsulation. Furthermore, combination of the methods is possible. In the following subsections, we will discuss the three methods. Note that our research currently focuses on the software description part. The appliance description part has been worked on [3, 4], while we just use VHDL descriptions for the computer hardware.

A. Software building blocks

To describe the software, we use *Data Flow Diagrams*, and draw them as directed graphs. The vertices denote the processes, and the edges denote the communication of data. Note that this communication also performs the synchronization between the processes. Such a data flow diagram shows the structure of the software, and allows for hierarchy, i.e. different levels of nesting can be used.

For the data communication, we exclusively use *channels*. Channels control synchronization and scheduling of processes. Channels are one-way, fully synchronized and basically unbuffered. However, buffers may be added to make the communication asynchronous.

Using channels encapsulates thread programming. Furthermore, priorities need *not* be specified anymore, since this is also handled by the channel. Moreover, scheduling is no longer a part of the operating system but is hidden in the channels, and thus has become part of the application instead. All these facilities alleviate the distributed software writing problem [1].

Since the processes and their communication via channels can be specified in the formal process algebra CSP, reasoning about correctness can be done. So, analyzing the CSP description of the software part of an ECS allows for formal checking on deadlock, starvation and life-lock. This gives opportunities to verify the software before it is tested on the real appliance.

Besides channels, we use special control flow constructs to control the parallelism of the processes and its communication. Also the priority of the statements can be specified.

We have developed the CTJ library (Communicating Threads for Java™ [5]) delivering fundamental elements for creating building blocks to implement a communication framework using channels.

CTJ channel concept

Processes may *only* communicate via channels, using read and write methods, see Figure 3. When both processes are ready to communicate, a communication event occurs; otherwise one of the processes waits (gets blocked). Synchronization, scheduling and the actual data transfer (i.e. copying the whole data object) are encapsulated in the channel. Thus, the programmer is freed from complicated synchronization and scheduling constructs.

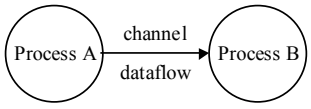

Figure 3: Data flow at channel communication.

Since the channel is an object itself, it is shown as a bubble in the implementation diagram, see Figure 5. In order to separate the hardware-dependent details of the communication, a device-driver framework for communication channels has been developed. These device drivers, so-called *link drivers*, are hardware-dependent objects that can be plugged into the channel. When a channel communication occurs between processes on different processors, channel and link-driver objects are present on both processors; the link drivers implement the specific communication protocol used.

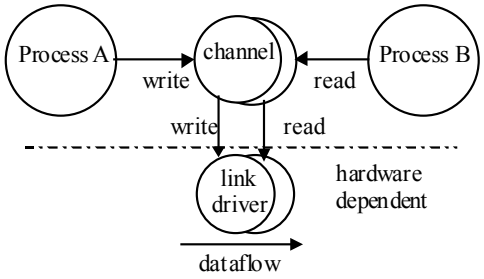

Figure 5: Channel implementation

Example

A data flow diagram of safety controller software of an industrial robot is shown in Figure 4 [6]. This safety controller resides between the actual controller and the robot itself. A so-called *safe robot* arises from this combination. Note that in this data flow diagram, control flow and the actual data flow are distinguished by dotted respectively solid lines. It is the real-time Yourdon convention.

B. Hardware building blocks

We just use VHDL descriptions of the computer hardware. Either realization can be done in specific circuits (ASIC) or, to be more flexible, using FPGA chips (Field Programmable Gate Arrays). The development of these hardware components is like software: updates can easily be made. Especially in the design phase, this is a real advantage. Furthermore, it is the solution when the specific chips are not available on the market anymore. However, the performance of FPGA chips need to comply with the demands.

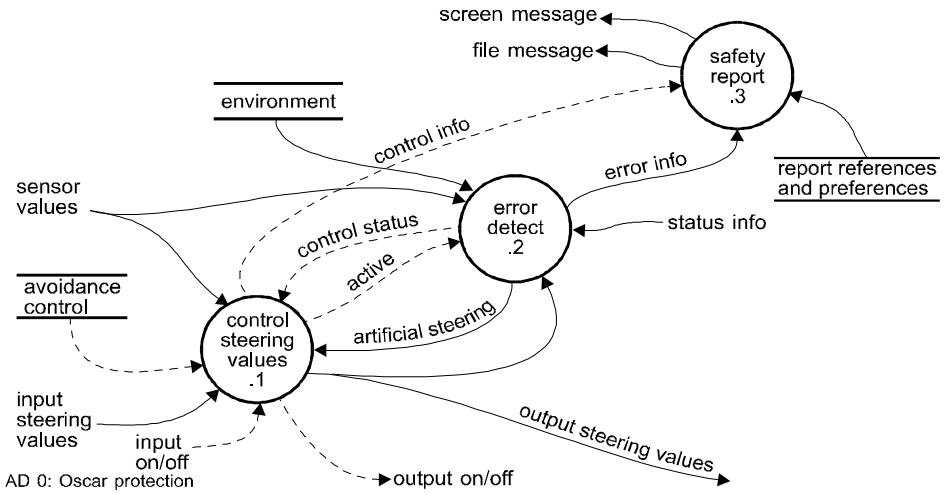

Figure 4: Data flow diagram of the safety controller

C. Appliance building blocks

For modeling the machine-part of the embedded system, i.e. the appliance, we use *Bond Graphs* [2, 7, 8]. Bond Graphs are directed graphs, showing the relevant dynamic behavior. Vertices are the submodels and the edges denote the ideal exchange of energy. They are physical-domain independent, due to analogies between these domains on the level of physics. Thus, mechanical, electrical, hydraulic, etc system parts are all modeled with the same graphs. Since the amount of basic physical concepts is limited, the number of basic elementary bond graph models is limited too.

Encapsulation is granted because:

- The interfaces of bond-graph submodels consist of so-called *ports*, consisting of two variables, whose product is the power exchanged through the port. For each physical domain, such a pair can be specified, for example voltage and current, force and velocity.
- The submodel equations are specified as real equalities, and not as assignments.

Differential equations are generated after model processing, where the port variables obtain a computational direction (one as input, the other as output) and the equations are rewritten to assignment statements.

Simulation of bond-graph models to study the dynamic behavior is in fact repeatedly executing the model statements.

Background

In this subsection, we give some background information on bond graphs.

Bond graphs are used to model the dynamic behavior of physical systems in a *domain independent* way. Domain indepence has its basics in the fact that *physical* concepts are analogous for the different physical domains. Six different elementary concepts exist: storage of energy, dissipation, transduction to other domains, distribution, transport, input or output of energy. Analogies between the electrical and mechanical domain are depicted in Table 1.

Item	Electrical	Mechanical
Variable pair	Voltage u Current i	Force F Velocity v
Energy storage	Capacitor Inductor	Spring Mass
Dissipation	Resistor	Friction
Energy input / output	Voltage source Current source	Force source Velocity source

Table 1: Analogies between electrical and mechanical domains

Another starting point is that it is possible to write models as *directed graphs*: parts are interconnected by bonds, along which exchange of energy occurs. A bond represents the energy flow between the two connected submodels. This energy flow can be described as the product of two variables (effort and flow), letting a bond be conceived as a *bilateral signal* connection. During modeling, the first interpretation is used, while during analysis and equations generation the second interpretation is used.

D. Case

This case shows the integration of building blocks in all three parts of an embedded system. It is an industrial robot controlled by a digital controller, often used for pick and place tasks [9]. The I/O is embodied by standard boards.

The robot has two vertical revolute joints and one vertical translational joint. The axes of the three joints are vertical (parallel to the z-axis). It is driven by three servomotors. A basic, single-axis control scheme is used to control a point-to-point motion, whereby the steering voltages are limited to resemble the real situation.

A photo of the robot and a sketch of the robot axes are shown in Figure 6. The breakdown into the ECS parts is shown in Figure 7. The bond-graph / block diagram model of the system is shown in Figure 9. In here, both the physical system (robot and motors) as well as the software part (controller) are described.

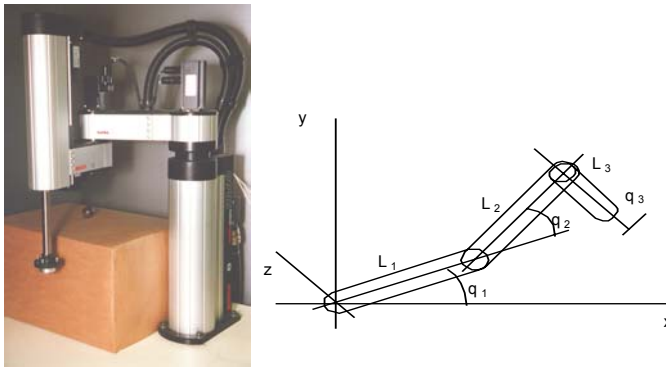

Figure 6 : Photo and axes layout of the robot

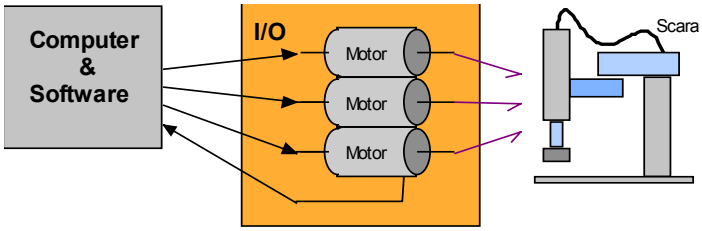

Figure 7: Breakdown into ECS parts

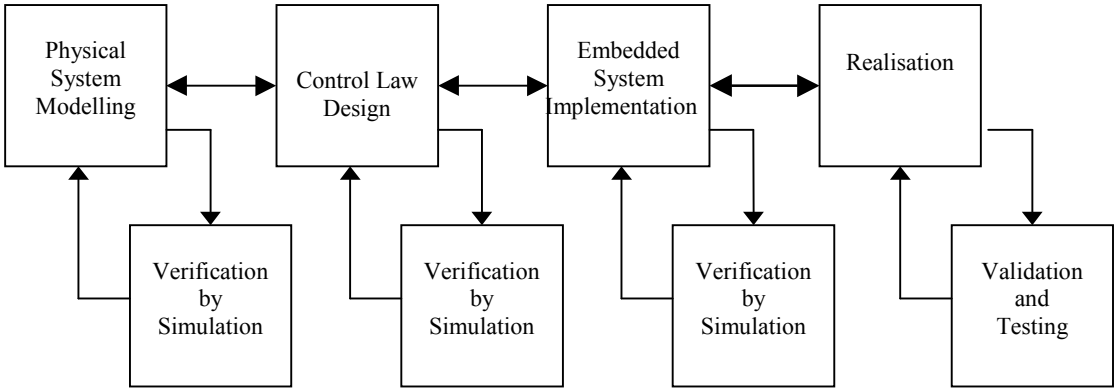

Figure 8: Design trajectory working order

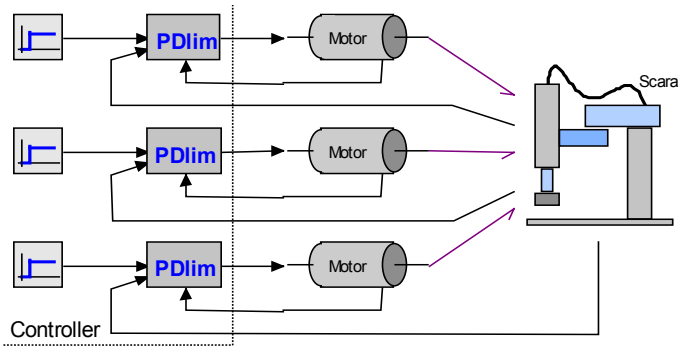

Figure 9: Simulation model of the robot with controller

IV. Building Block Libraries

Building blocks can speed up design processes, since existing building blocks can be reused.

Essential criteria to allow for reusability, namely well defined interfaces and proper encapsulation of the block's contents, are satisfied in our building block approach, because we use a true object-oriented approach. Still, classification and structuring of building blocks into libraries is a serious issue [10].

Fortunately, the underlying theories for the different ECS parts help us to find classification criteria. Especially, we have experience with the software and device parts, namely the CTJ library and the bond-graph submodel libraries as implemented in 20-SIM.

V. Simulation and ECS-Implementation

Since the ECS descriptions presented here, are simulatable, we use simulation to verify our designs. Furthermore, we advocate a stepwise refinement from specification to (software) implementation (see Figure 8). This way, checking design alternatives can be done efficiently [11].

Furthermore, different parts of an ECS can be developed separately, provided that the overall model is competent for testing. This implies that development design process can be organized as a concurrent engineering activity. For modern system development, this is an essential feature [12].

The design trajectory of ECS is as follows:

- *Physical Systems Modeling.*
  The dynamic behavior of the system is *object–orientedly* modeled, using bond graphs as a main modeling paradigm.
- *Control law Design.*
  Using the model acquired in the previous step or a simplified version of it, control laws are designed.
- *Embedded Control 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.
- *Realization*
  The realization of the ECS 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.

The stepwise refinement procedure for the embedded software consists of the following steps:

- Control laws only
  The implementation is assumed to be ideal: sensors, actuators and algorithms do *not* have any effects on the performance of the ECS.
- Non-ideal components
  Those components, being considered ideal in the previous step, are modeled now more precisely by considering their relevant dynamic effects.
- Safety, and command interfacing
  Reaction to external commands, like from the operator or from connected systems is specified. In addition, safety measures are accounted for (like reaction on external events from like emergency stops and end switches, etc.). Furthermore, facilities for maintenance processing can be added here.

The impact of these additions on the behavior of the ECS can be checked by means of simulation.

- Effects due to non-idealness of computer hardware
  The control computer hardware and software architecture are added. Effects of computational latency and accuracy can be checked. Scheduling techniques and / or algorithm optimization techniques may be used to obtain a viable realization.

These steps need not be performed in the order specified here. The designer has the freedom to tackle the individual subproblems in any order. This is a major difference with the traditional design methods which are basically waterfall like. 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.

In the realization step, simulation can play a relevant role, especially when the design project is set up in a concurrent engineering fashion. The first available part of an ECS can be tested together with the other parts, which are still simulated models. This verification process is a form of *hardware-in-the-loop simulation*.

VI. Conclusion

Embedded (control) systems can *completely* be described by object-oriented techniques, using a building-block approach: for all parts (software, hardware, and appliance) we use such techniques, namely bond graphs, VHDL and component based software using channels.

Advantages are the possibility to use a concurrent engineering approach, to use simulation as a means for verification, and to use a mechatronic or systems approach during design. The latter truly supports *flexible hardware-software co-design*, which becomes crucial in modern embedded system development.

Current research deals with the development of a design framework and a tool to efficiently apply the building block approach [11]. We use applications in the field of robotics and mechatronics. Currently we are focussing on the use of heterogeneous networked embedded systems.

References

[1] G. H. Hilderink, A. W. P. Bakkers, and J. F. Broenink, "A Distributed Real-Time Java System Based on CSP," presented at The third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC 2000, Newport Beach, CA, 2000.

[2] P. C. Breedveld, "Multibond-graph elements in physical systems theory," *Journal of the Franklin Institute*, vol. 319, pp. 1-36, 1985.

[3] J. F. Broenink, "20-Sim software for hierarchical bond-graph/block-diagram models," *Simulation Practice and Theory*, vol. 7, pp. 481-492, 1999.

[4] J. F. Broenink, "Object-oriented modeling with bond graphs and Modelica," presented at Proc. 1999 Western Simulation Multiconference, Conf. on Bond Graph Modeling and Simulation ICBGM'99, San Francisco, USA, 1999.

[5] G. H. Hilderink, J. F. Broenink, and A. W. P. Bakkers, "Communicating threads for Java," presented at Proc. 22nd World Occam and Transputer User Group Technical Meeting, Keele, UK, 1999.

[6] J. F. Broenink, F. v. d. Mast, K. C. J. Wijbrans, and M. J. L. Tiernego, "Design and simulation of safety measures for a 6-dof robot," presented at 1992 European Simulation Multiconference on Modelling and Simulation, Royal Park Hotel, York, United Kingdom, 1992.

[7] D. C. Karnopp, D. L. Margolis, and R. C. Rosenberg, *System dynamics, a unified approach*, 2nd ed. New York, NY: J Wiley, 1990.

[8] J. F. Broenink, "Introduction to physical systems modeling with bond graphs,".: to be published in the SiE whitebook on Simulation Methodologies, 1999.

[9] H. Ecker, "Comparison 11: Scara Robot - definition; solution in ACSL," *Eurosim - Simulation News Europe*, pp. 30-33, 1998.

[10] A. P. J. Breunese, J. F. Broenink, J. L. Top, and J. M. Akkermans, "Libraries of reusable models: theory and application," *Simulation*, vol. 71, pp. 7-22, 1998.

[11] J. F. Broenink, G. H. Hilderink, and A. W. P. Bakkers, "Conceptual design for controller software of mechatronic systems," presented at Proc. Lancaster Int. Workshop on Engineering Design CACSD'98, Lancaster, United Kingdom, 1998.

[12] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analyis*, 3 ed: Prentice Hall, 1998.