Proc. Progress 2001 Workshop, October 18, 2001, Veldhoven, Netherlands, F. Karelse (Ed.), pp. 121-126, ISBN: 90-73461-26-X.

Integrated Design Tool for Embedded Control Systems *)

Dusko S. Jovanovic, Gerald H. Hilderink and Jan F. Broenink

Electrical Engineering, Control Laboratory, Cornelis J. Drebbel Institute for Mechatronics,
Twente Embedded Systems Initiative.
P.O.Box 217, 7500 AE, Enschede, the Netherlands
Phone: +31 53 489 2788 Fax: +31 53 489 2223
E-mail: D.S.Jovanovic@el.utwente.nl

Abstract – Currently, computer-based control systems are still being implemented using the same techniques as 10 years ago. The purpose of this project is the development of a design framework, consisting of tools and libraries, which allows the designer to build high reliable heterogeneous real-time embedded systems in a very short time at a fraction of the present day costs. The ultimate focus of current research is on transformation control laws to efficient concurrent algorithms, with concerns about important non-functional real-time control systems demands, such as fault-tolerance, safety, reliability, etc.

The approach is based on software implementation of CSP process algebra, in a modern way (pure object-oriented design in Java). Furthermore, it is intended that the tool will support the desirable system-engineering stepwise refinement design approach, relying on past research achievements – the mechatronics design trajectory based on the building-blocks approach, covering all complex (mechatronics) engineering phases: physical system modeling, control law design, embedded control system implementation and real-life realization. Therefore, we expect that this project will result in an adequate tool, with results applicable in a wide range of target hardware platforms, based on common (off-the-shelf) distributed heterogeneous (cheap) processing units.

I. INTRODUCTION

In the design process of real-time embedded systems (e.g., automobiles, robots, production lines, consumer products like photo cameras or household appliances, etc.), it becomes increasingly important that prototyping be delayed as long as possible. The current design departments in automotive, machine and machine tool industry demand tools and methods by which their prototyping can move to “do it right the first time”. This is both for reasons of development costs as well as development time.

On the other hand, the current state in the industry is such that computer-based control systems are still being implemented using the same techniques as 10 years ago. Together with the complexity of modern embedded systems, it is one of the main reasons for the large number of errors in these systems.

The purpose of this project is the development of a design methodology, consisting of tools and libraries, which allows the designer to build high reliable heterogeneous real-time embedded systems in a relatively short time. The use of the proposed integrated design environment should shorten the design phase provided that its use is accurate and cost-effective. Furthermore, the design engineer can identify the weak and strong points by means of stepwise refining the design towards the realization. This obviously reduces costs in design and maintenance and safeguards against catastrophic disasters due to malfunctioning of the embedded software.

The software tool we want to develop has facilities to check and test an embedded system also on safety and reliability aspects during the design process, and thus supports the design of safe products.

By stimulating an iterative approach, which is a quite natural way of working, tool support becomes inevitable. This motivates our research on the design framework and tool development. Note that iterative ways of development is also performed in the separate areas of software development for embedded systems and controller design.

II. COMMON RECOGNIZED PROBLEMS IN THE FIELD OF PROGRESS RESEARCH EFFORTS

Although we deal with engineering of software for heterogeneous embedded systems, we find much in common with several approaches found in the field of design of embedded systems. Furthermore, in our opinion, the fact is that many research efforts under the hood of PROGRESS are faced to the most of specific

*) This research is supported by PROGRESS, the embedded system research program of the Dutch organization for Scientific Research, NWO, the Dutch Ministry of Economic Affairs an the Technology Foundation STW.

Proc. Progress 2001 Workshop, October 18, 2001, Veldhoven, Netherlands, F. Karelse (Ed.), pp. 121-126, ISBN: 90-73461-26-X.

issues we are also faced to; this should be concerned as an advantage and collaboration challenge for achieving a real-life instrument to allow (embedded) control system engineers to design *right* systems, reliable and safe, energy-efficient, intended to benefit the humanity.

Current tools for the design of real-time systems leave a gap between the detailed specification as a hierarchy of interacting processes and executable code. This gap has to be spanned by time-consuming and error-prone manual methods. Errors in design detected in coding or subsequent testing or maintenance means that this gap has to be spanned repeatedly. Current methods for the low-level production of code rely on separate concepts, a programming language and a real-time operating-system/kernel, to provide key elements of functionality. In particular, access to the clock, time-outs, interrupt handling, process creation, scheduling, synchronization and message passing are handled by calls to library routines, the implementation of which is not under the control of the designer. The interface between the language and real-time library is too complex, and usually too incomplete, to be formalized in any way that allows guarantees for system safety to be made. In addition, the run-time overheads of managing these fundamental operations through library calls are unnaturally large. This further contributes to the complexity, and hence insecurity, of both the design-support tools and the resulting products.

It is not hard to realize that dealing with various disciplines is inevitable in this project.

For instance, the specific issues in the approach of the stepwise refinement design paradigm in the terms of hybrid system design (modeled starting with highly abstract component to low-level, detailed descriptions, at the same design time) are recognized in so-called hardware / software co-design approach [1], [2]. This is more elaborated in the section IV.

Also, it is well recognized the importance of well defined, encapsulated way of communication in a concurrent environment. We believe that abstraction of channels is the best suitable to this need. Moreover, it's shown that channels, besides communication issues, perfectly and simply in the same time can address many more critical concurrent programming problems, as synchronization, scheduling, prioritizing and even hardware independence (portability) [3, 4], [5]. Hence, it is advisable that channels should be deployed exclusively in all communication jobs. It is argued more in section V.

Finally, formal model checking is recognized as the most expensive, but the most reliable methods of proving a system safety, liveness, correctness, fairness [6], [7], [8]. But, involving certain techniques of preventing the state space explosion problem by partial order reduction, there are experiences that formal methods approaches are possible [9], making them interesting enough to be taken into account from the very earliest phases of the system engineering by the design tool. In section VI we advocate inherently incorporated abilities of doing such kind of design support (besides simulations) through implementation library (CTJ) based on CSP process algebra [13], [14], trying to avoid overhead of models translations from modeling to checking suitable descriptions.

III. Main Project Objectives

With confidence in overcoming these and related problems, the following specific objectives are proposed for the project:

- To simplify the process of designing, implementing, and maintaining real-time embedded systems by providing a programming language and design tools, which directly supply all the necessary levels of abstraction (for *security, development, and maintenance*).
- To integrate real-time operating system/kernel functions within a programming language, so that the semantics can be simplified and made formal (for *security*), the high-level design tools can generate executable code directly (*for development and maintenance*), and so that efficient code can be compiled (*for performance*).
- To have an implementation strategy for the language that is open and retargetable so that it can run on a range of different micro-processor/controller architectures (*for portability*).

The embedded computer systems considered here are heterogeneous and distributed, consisting of various common off-the-shelf (COTS) computing units (MCUs,

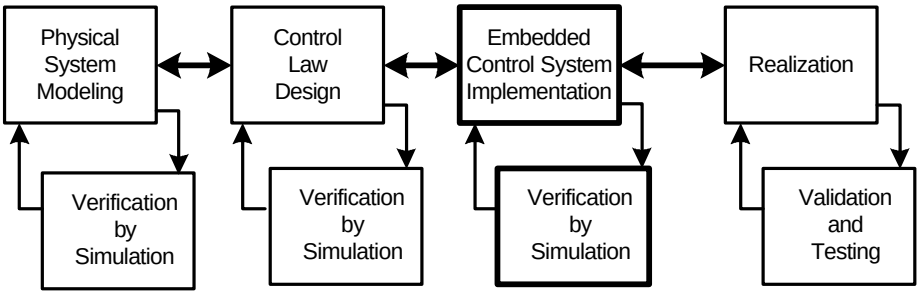

Figure 1 Design trajectory working order

Proc. Progress 2001 Workshop, October 18, 2001, Veldhoven, Netherlands, F. Karelse (Ed.), pp. 121-126, ISBN: 90-73461-26-X.

DSPs, as well as general purpose CPUs, eventually with functionality augmented by ASICs or FPGAs) deployed in parallel – hence, capable for intensive communication, also involving issues of allocation, scheduling, synchronization and interfacing.

IV. Stepwise Refinement

Stepwise refinement means that the total model (from physical system to be controlled to control laws implemented to efficient concurrent control computer code) will gradually change from a basic functional or conceptual model towards a detailed model from which the code for the control-computer system can straightforwardly be generated and downloaded.

Designing embedded control software generally starts with a dynamic model of the process to be controlled, since the process dynamics is of crucial importance for the behavior of the embedded system. To properly support the system-engineering stepwise refinement approach, verification by means of simulation (i.e. executing the models) is a key activity. Therefore, we adhere the design trajectory as shown in Figure 1:

- *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.
- *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.

After each step, the results are verified by simulation, also in the last phase (realization) when some parts are still a model.

The dynamic model of the system and the control law are considered to be ready, i.e. delivered by the mechatronic engineer. This implies that the software tool being developed should seamlessly fit to the software tool(s) used for the first two steps (see Figure 1), namely 20-SIM. Therefore, the starting point is the executable block diagrams, which are in fact data flow diagrams. The refinement process now is the extension of the block diagrams towards software specification.

Note that the *execution model* of the block diagrams in the supporting tools is based on simulation of a hybrid system (i.e. continuous parts for the process and discrete parts for the controllers), whereas the execution model of the embedded control software should be that of *efficient execution of the code on the specific target.*

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

1. *Integrate control laws*
   Combine the control law(s) with the sequence and supervisory control layers. Reaction to external commands, like from the operator or from connected systems is taken into account.
   Design and test the bumpless transfer when switching from one control law to another. Design and test protocols on machine level (e.g. homing to ensure proper repeatability).
   The implementation is still assumed to be ideal.
2. *Capture non-ideal components*
   Those components, being considered ideal in the previous step, are now modeled more precisely by augmenting the specification with their relevant dynamic effects (i.e. adding non-idealness of components).
   Also, add algorithms to process signals to obtain other signals which could not be measured directly in the practical situation (e.g. add an estimator to derive an internal variable, for which no sensor will be available).
3. *Incorporate safety, error and maintenance facilities*
   Facilities for safety of the system are specified and designed (like reaction on external events from emergency stops and end switches, etc.).
   Safety and error handling can be centralized in one module or distributed among the components. A centralized module enables easier assessment of the safety measures, as is proposed in the Safety Kernel Design Pattern [10]. Safety handling distributed among the components allow for reusable components, which are safe.
   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.
4. *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 our opinion, the building-blocks approach, due to its encapsulation, object-oriented qualities, offers capabilities to address all particular difficulties that are related with various levels of description details at different parts of overall (*hybrid*) system model hierarchy.

V. Communication, Synchronization and Scheduling

Because the data flow diagrams (DFD) are parallel running processes exchanging data via point-to-point connections, the kernel facilities to be integrated have to deal with communication, synchronization and scheduling.

For the data communication, we exclusively use *channels*, using *read* and *write* methods. Channels are simply synchronization primitives that provide communication between concurrent or distributive processes. This synchronization principle is called *waiting randevouz*. Channels are one-way, fully synchronized and basically unbuffered. However, buffers may be added to make the communication asynchronous. The basics of channels are described in CSP.

In terms of CSP, a *process* is a group of tasks, not necessarily being sequential. Processes may run in parallel, in some sequence or by some choice. CSP specifies fundamental control-flow constructs that describe the sequence of executing processes: PAR (parallel), SEQ (sequential), or ALT (alternative, a kind of case statement).

Synchronization, scheduling and the actual data transfer are encapsulated in the channel. Thus, the programmer is freed from complicated synchronization and scheduling constructs: thread programming is encapsulated.

Furthermore, priorities need *not* be specified anymore, since the channel also handles this. 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 [11].

We have developed the CTJ library (Communicating Threads for Java™ [12]) delivering fundamental elements for creating building blocks to implement a communication framework using channels. Besides the prototype in Java, which serves as a *design pattern*, implementations in C++ and C were developed. At this moment, thorough tests on real applications need to be done.

VI. Verification: Simulation and Formal Checking

Simulations are irreplaceable as helper method for designer – to mimic (animate) current design, as feedback to designer.

On the other hand, simulations are unable to prove design quality in one exhaustive way: “One of the disadvantages of simulation-based analysis is that it does not give guarantees about all possible behaviors of the system. That is, simulation *can* show that the system can behave as required, but it cannot show that it will *always* behave as required.” [9].

Being aware of such kind of experiences, from the very beginning of developing the CTJ library, we were confident that the ability of allowing formal checking could be incorporated directly in the description of concurrent control algorithms. 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.

VII. Concluding remarks

Since we are in the beginning of the project, we rely on the existing work and experience on existing tools (20-SIM), existing libraries (CTJ), existing theories (CSP, Bond graphs), and existing design trajectories.

Currently, we focus on applying UML to the issues, which are traditionally described in data flow diagram notation.

Another issue we will work on is to investigate whether RT-Linux can serve as an interface between the not-real time development environment and the hard real time embedded control system.

Furthermore, the ultimate result of this project will be the design tool and design methods for embedded controller design.

References

[1] A. D. Pimentel, P. v. d. Wolf, E. F. A. Deprettere, L. O. Hertzberger, J. T. J. v. Eijndhoven, and S. Vassiliadis, "The Artemis Architecture Workbench," presented at PROGRESS 2000 Workshop on Embedded Systems, Utrecht, the Netherlands, 2000.

[2] A. W. v. Halderen, A. Belloum, A. D. Pimentel, and L. O. Hertzberger, "On Hybrid Abstraction-level Models in Architecture Simulation," presented at

Proc. Progress 2001 Workshop, October 18, 2001, Veldhoven, Netherlands, F. Karelse (Ed.), pp. 121-126, ISBN: 90-73461-26-X.

PROGRESS 2000 Workshop on Embedded Systems, Utrecht, the Netherlands, 2000.

[3] J. F. Broenink and G. H. Hilderink, "Building blocks for control system software," presented at Proc. 3rd Workshop on European Scientific and Industrial Collaboration WESIC2001, Enschede, Netherlands, 2001.

[4] J. F. Broenink, G. H. Hilderink, and A. W. P. Bakkers, "Building Blocks for Embedded Control Systems," presented at PROGRESS 2000 Workshop on Embedded Systems, Utrecht, 2000.

[5] M. Bos, P. J. M. Havinga, and G. J. M. Smit, "Channel communication and Reconfigurable Hardware," presented at PROGRESS 2000 Workshop on Embedded Systems, Utrecht, the Netherlands, 2000.

[6] W. J. Toetenel and R. L. Spelberg, "Model Checking Real-Time Embedded Software," presented at PROGRESS 2000 Workshop on Embedded Systems, Utrecht, the Netherlands, 2000.

[7] B. D. Theelen, J. P. M. Voeten, L. J. v. Bokhoven, G. G. d. Jong, A. M. M. Niemegeers, P. H. A. v. d. Putten, M. P. J. Stevens, and J. C. M. Baeten, "System-Level modeling and Performance Analysis," presented at PROGRESS 2000 Workshop on Embedded Systems, Utrecht, the Netherlands, 2000.

[8] R. d. Vries, J. Tretmans, A. Belinfante, J. Feenstra, L. Feijs, S. Maauw, N. Goga, L. Heerink, and A. d. Heer, "Cote de Resyste in PROGRESS," presented at PROGRESS 2000 Workshop on Embedded Systems, Utrecht, the Netherlands, 2000.

[9] V. Bos and J. J. T. Kleijn, "Model Checking Manufacturing Systems: A Case Study," presented at PROGRESS 2000 Workshop on Embedded Systems, Utrecht, the Netherlands, 2000.

[10] B. P. Douglass, *Real-Time UML: developing efficient objects for embedded systems*: Addison Wesley Longman, 1998.

[11] G. H. Hilderink, A. W. P. Bakkers, and J. F. Broenink, "A distributed Real-Time Java system based on CSP," presented at Proc. Third IEEE Int. Symp. On Object Oriented Real-Time Distributed Computing ISORC'2000, Newport Beach, CA, USA, 2000.

[12] 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.

[13] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[14] A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall, 1997.