A Case Study for Tooling the Design Trajectory of Embedded Control Systems*)

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

Twente Embedded Systems Initiative,
Cornelis J. Drebbel Institute for Mechatronics and Control Engineering,
Dept. of Electrical Engineering, University of Twente,
P.O.Box 217, NL-7500 AE Enschede, The Netherlands
Phone: +31 53 489 2288 Fax: +31 53 489 2223
E-mail: D.Jovanovic@utwente.nl

Abstract – **We strive to allow a mechatronic system designer the power of designing mechatronic systems in manners of concurrent engineering in short time at a fraction of the present day costs. In our context, this means a methodology and tool support to stepwise refinement in analysis and design of the plant to be controlled, computing and I/O hardware and computer code which provide all operational functionalities.**

Nowadays, it is impossible to separate control engineering from software engineering. There is no way of implementing control strategies other than transform them in computer code for chosen processing target. Usually, there are not many "general-purpose programmers" available in control engineering research teams in companies and universities. In these cases, used software development techniques suffer from insufficiencies in knowledge in disciplines of software modeling, familiarity with concurrency in software, ways of allowing for reusability, software testing and so on. The gap between control laws design and implementing them on the targeted platform(s) is recognized as critical and not methodologically covered by existing approaches and tools.

This paper, reporting on the progress of project TES.5224, presents evolution of a system based on a 2DOF robot as a prototype of a design trajectory.

Keywords – **embedded control systems (ECS) design, system-level modeling, stepwise refinement, integrated tools, CSP/CT, simulate-ability, design portability, software quality, software specification, concurrency, robotics**

I. INTRODUCTION

The dynamic changes in technologies and markets of computers' platforms make the problems to control engineers difficult. Usually, due to existing experiences, implementation teams are stuck to certain hardware platforms, programming languages or libraries that provide software development environments the researchers are used to. Often, there is resistance to migrating to new, more advanced and more adequate hardware and software platforms. This is not hard to understand: being experienced with peculiarities of certain processing units and knowing all issues of software (code) development for those, designers are not eager to put time and energy in mastering new targets. The same holds also for updating models with new discovered modeling issues, especially if the methodology and/or the tool do not automatically reflect changes in the rest of the system properly. Usually, time costs for getting in touch with new implementation conditions have unmotivating effects. In the other hand, staying stuck to old platforms and methods introduce severe constraints to design modernization.

Notions of Stepwise refinement [1] approach in designing control systems from its conception to real-life implementation and CSP/CT [2] paradigm for dealing with concurrent behavior of computing part of those systems are recognized like promising design manners. By building tools upon these mechanisms it is believed that control engineers will be handed with a rather tractable means of managing computing aspects of modern control engineering. Since that design entries are suitable for graphical environments (bond graphs [3], 20-SIM [4] or Simulink [5] block diagrams, CSP diagrams [6]), acceptance of the methodology and tools in the embedded control systems industry segment can be expected.

Section II reports the achievements on the methodology for designing control software for mechatronic systems. Section III makes the core of the paper – a robotic case study.

*) 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.

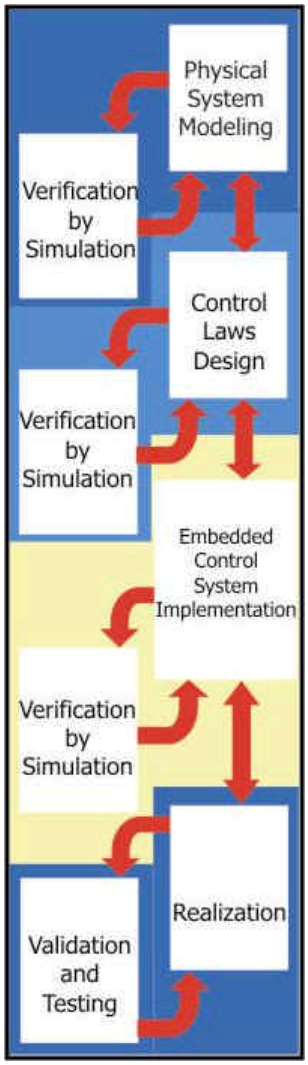

Figure 1 Design trajectory

II. METHODOLOGY

General ECS design methodology has an intention to allow a designer freedom of exploring the design space, i.e. arbitrary stepping in the (sub) phases captured by the ECS design trajectory. The four main phases in engineering an embedded control system are articulated in Figure 1, [2]. During each step, results are verified by simulation, also in the last phase (realization) when some parts are still a model. This demand of simulate-ability actually means that the design models at hand must always be completely specified, but not necessary in full detail. This might seem a strong demand, but gives opportunities to fully exploit the Stepwise Refinement approach including sophisticated tool support.

Since the focus of this research is on Embedded Control System implementation (the third, yellow-highlighted phase), only this phase is further detailed. It comprises of the following subphases:

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.

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.). 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. The inherent parallelism in control systems can be made explicit in the control software by using CSP diagrams [6]. 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.

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.

In the remainder of this section we discuss three methodological topics: Stepwise Refinement, Exploiting Parallelism and Code Generation. For all three topics, we will also indicate how tool support using the tool 20-SIM can be done.

A. Stepwise Refinement

Our *stepwise refinement* (SWR) paradigm strives for development of methodologies and tools to support mechatronics engineers in treatment (analysis and design) of a control system as a whole, allowing them to start with a sketch of overall system, and gradually refine

the model of solution in the course of understanding the problem in hand.

The SWR procedure is applied in all vital parts of a mechatronic system: controlled plant, computing and I/O hardware and computer code which provides all supervision, command and control functionalities.

Normally, mechatronic design engineers start with modeling the dynamic behavior of the plant, and derive control laws for it. These control laws are then gradually transformed towards efficient concurrent algorithms (i.e. the control-computer code). The SWR process thus transforms the control laws into an execution model that is ready to be translated into control software, whereby one can easily specify concurrency with symbols stemming from the CSP language in a block diagram [7]. The realization of the *Embedded Control System* (ECS) is also worked on as a SWR sequence. Parts of the system stay as models while other parts are coded on their target hardware.

The 20-SIM modeling and simulation tool can be used well for SWR, because it allows organizing a mechatronic model in the abstractions of high-level building blocks. Non-basic building-blocks are partitioned in hierarchies of simpler and simpler building-blocks, which can be easily accompanied with basic default dynamic behavior. In that manner, simulations can be used as verification means in very early phases of the system design.

B. Exploiting parallelism

Although embedded systems are inherently parallel, this natural parallelism need to be specified explicitly in the embedded control software. Furthermore, there are some software engineering reasons to exploit this natural parallelism:

It is straightforward to add new functionality by adding software components. Just add a new ‘block’ in the CSP diagram of the software, and connect it to the other software components. It is like adding a block in a block diagram.

The parallel structure of the software can be used to deploy the processes over different processing units, thus making a distributed implementation easier.

Since the communication with the components is explicitly through its interface, reusability is easier achieved.

Two other, more functional, reasons for exploiting this natural parallelism are:

The performance can be better, in the sense that the throughput of data can be faster.

The responsiveness to external stimuli is better, in the sense that the control software will react as quick as expected. This is especially useful when errors occur.

By using CSP diagrams, we can specify exactly what we mean and how the execution framework should fulfil our requirements [6]. CSP diagrams reflect the organization of software in two views: in data-flow and composition aspects. For the first aspect, the diagram is called a *communication graph*, and has much resemblance with a block diagram. For the second aspect, the blocks are linked with so-called composition connections, showing whether the blocks run in parallel, in sequence and if one has priority over the other. It is called a *composition graph*.

The composition graph also allows putting software execution in concurrent network. In short, by designing CSP diagrams we will have complete control over the execution framework of the controller software and we can extent the execution model with additional processes that cannot (yet) be dealt with by 20-SIM. Examples of CSP diagrams are shown when the JIWY case is discussed (section III).

Currently, 20-SIM does not support specifying the execution framework in CSP (diagrams), so for “parallelization" of the software design 20-SIM can be used to rather limited extent. Namely, using the diagram editing features (explode / implode of model parts), the model can be structured such, that the resulting blocks represent software processes, for which the parallel execution structure can be defined. Actually, the CSP diagrams are made by hand, and currently tool support is being designed.

C. Code generation

The project decision in respect to computer code engineering is to rely on the concepts of Communicating Sequential Processes (CSP) [8]. Furthermore, the code for all necessary software components (supervision, command, control, adaptation, safeguarding, communication, etc) will be generated automatically from the simulated models. This assures the quality and maintainability of the code. Besides, it is believed to allow orthogonality of additional software layers in respect to control code. For instance, we try to eliminate anomalies (impacts) of adding safeguards to the existing code as much as possible so that we end up with a clean design that is easily maintainable, extendable and reusable [1].

20-SIM is used to this end too. At the moment, code generation is performed in interplay of the models and target-specific templates. The templates, along with target-specific (device) drivers, make connection between simulation models and concrete target resources.

Unfortunately, 20-SIM does not support user’s influence in execution framework of the code that could be generated out of existing model. Namely, the execution framework is the same as simulation framework, which is simple sequential, as implied by the simulation frame-

Figure 2 Photo of JIWY

work. As indicated above, the annotation of parallelism has as yet to be entered by hand.

III. CASE STUDY: JIWY

As case study, a 2DOF robot has been chosen. This is a small positioning robot, for orienting some device, i.e. a camera, laser pointer or similar (see Figure 2 for a photo). The construction contains two revolute joints that allow mounted device to rotate on a horizontal axis and a vertical axis. The joints are equipped with DC motors and incremental encoders.

The idea is to put the robot in a typical closed position servo loop, according to the direct digital control (DDC) manner.

In the remainder, the flow of Stepwise Refinement from the closed loop towards the realization of them is explained in several iteration steps.

A. Result of the first two phases of the design trajectory

SWR starts with a coarse-grained sketch (similar to paper-sketches), able to "mimic" basic shape of the engineered system, see [2] and [9] for the JIWY-specific situation.

Tooling support for iterative steps has to allow flexibility for the models to reflect design decisions. Furthermore, the simulations have to be enabled as soon as possible.

According to certain design decisions, an analogue joystick is attached to the computer. This suggests an update of the scheme of general closed loop concept, reflected in Figure 5. Consequently, the starting 20-SIM top-level model (Figure 4) is adapted, see Figure 6.

Since the aim of this paper is to concentrate on the refinement of the software, the model has to be enhanced with enough detailed behavioral description.

The internals of the top building blocks are obtained in the first two phases of our design trajectory (Figure 1) to show the level of details necessary to start SWR of software part ("Controller") solely.

The motors in joints of the robot are modeled by bond-graphs models, which reside in the "motor" boxes of Figure 7.

The power amplifiers for the steering signals and the encoders measuring the angle of both the axes are sufficiently modeled with gains Figure 8. The joystick serves as a set-point generator, and therefore is modeled as two signal generators from the standard 20-SIM submodel library Figure 9. On the control computer, besides the control law (here a standard PID control law), also the A/D and D/A converters are commanded (Figure 10). This interfacing hardware resides on the PCI interface card from National Instruments. The choice for this card, and the development of the device drivers for it are discussed in [10].

B. Refinement in ECS implementation

This third phase in the design process is the core of the research reported here. The JIWY case is rather small, such that not all subphases of the ECS design phase are applicable here. As indicated in section II, the subphases need not be followed in the specified order, nor each subphase really needs to result in some refinement. At our case, subphase one was not applicable and subphases three and four were exchanged.

1. Integrate control laws

As yet, we only have one standard loop controller for each of the two axes working independently; integration of control laws is not applicable here.

2. Capture non-ideal components

We needed signal condition functions for both the Joystick signals and the encoder signals representing the angle of the two axes of the robot. Initial tests also indicated that filtering the Joystick signal was necessary, due to its spiky waveform. The result is shown in Figure 11.

The signal limiter symmetrizes the ranges of the potentiometers in the analogue joystick; the attenuators adjust the signal levels. The noise filter is a forth order Butterworth low-pass filter, designed for 100Hz sampling frequency with cut-off on 2Hz. The filter is designed using 20-SIM's Linear System Editor.

Actually, we did *not* model the Joystick behavior including its output signal quality in detail, but only added a software component to enhance the quality of the signal. So, with the resulting model, we cannot show what the origin of these spiky signals is. For our design context (i.e. use the given Joystick for the demo), it is not relevant to know where these spikes come from. However, it *is* relevant to understand that whenever the Joystick will be replaced, the signal conditioner should be reconsidered.

Figure 3 Closed control loop

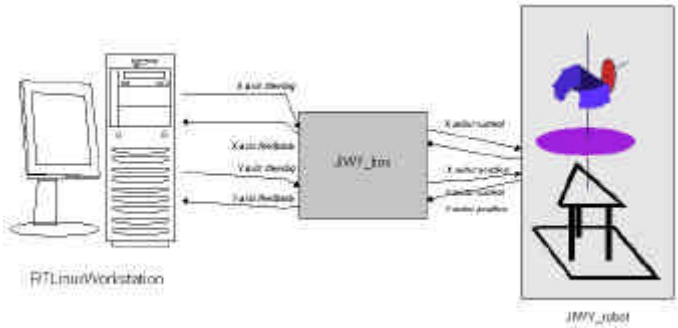

Figure 4 Top-level JIWY model

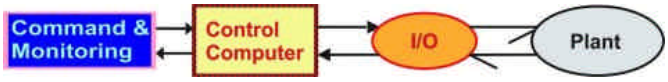

Figure 5 Closed-loop control with man-machine interface

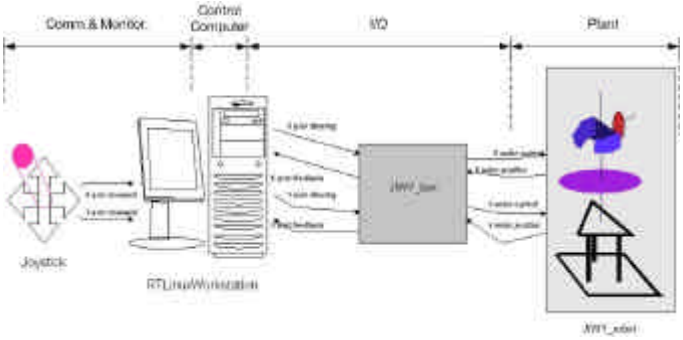

Figure 6 Top-level JIWY model in 20-SIM, with a joystick as set-point generator

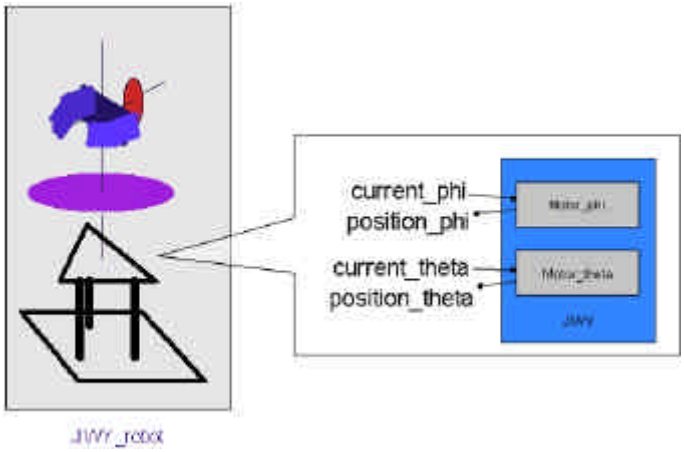

Figure 7 Model of the robot

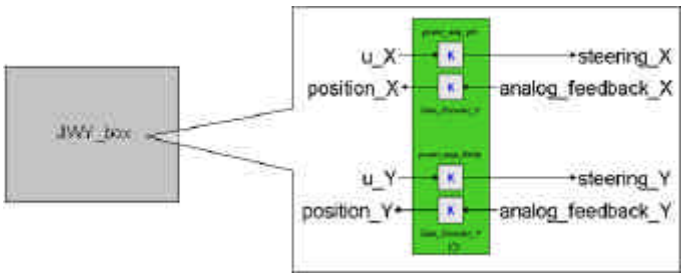

Figure 8 The internals of the amplifier I/O box

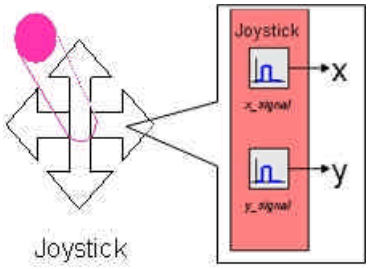

Figure 9 Sufficient model of joystick

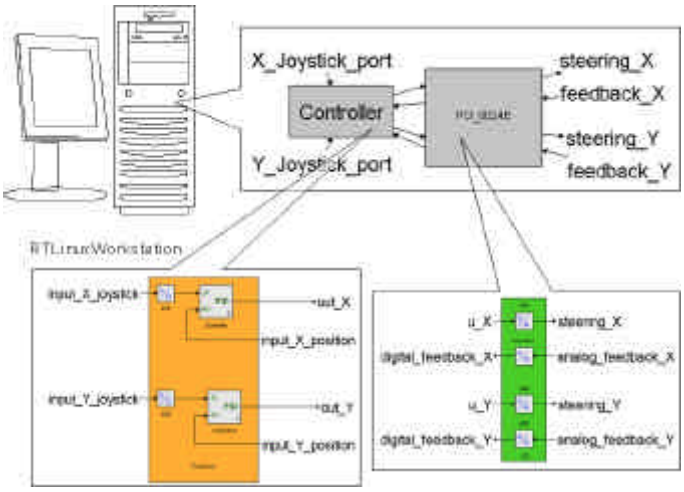

Figure 10 The control computer: the outcome of the first 2 design phases: physical system modeling and deriving control laws

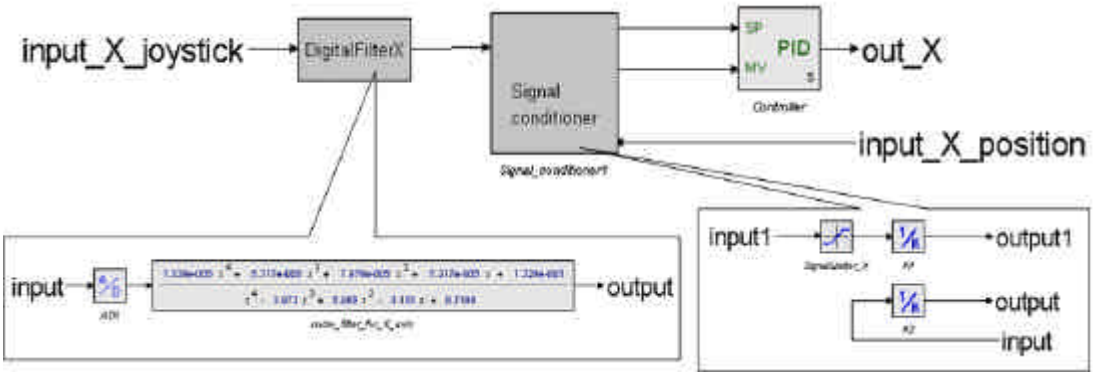

Figure 11 The refinement of the Controller building-block in 20-SIM for one axis

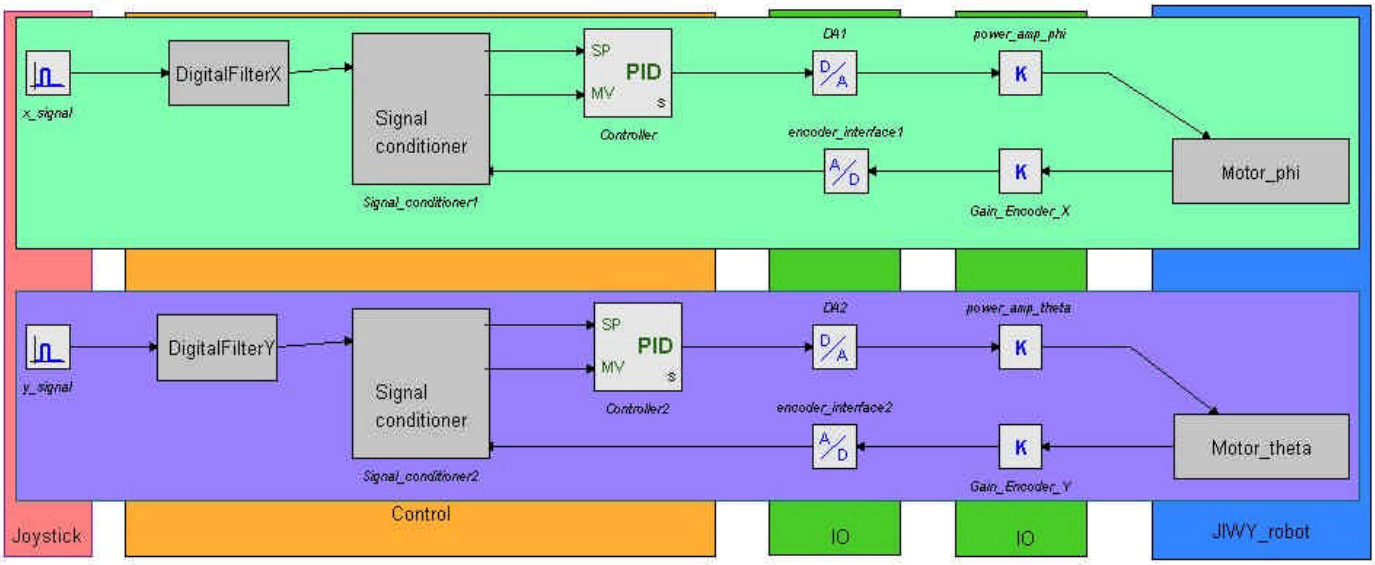

Figure 12 The total JIWY model with parallelized controller

3. *Computer Architecture*

The next step is to enhance the models of the controller part (i.e. those parts that will be implemented in software), such that a more close resemblance with the final real situation will be reached, i.e. the parallelism will be added. By using the CSP diagrams, we can specify exactly what we mean and how the execution framework should fulfill our requirements. Currently, 20-SIM does *not* support specifying the execution framework in CSP diagrams, so for "parallelization" of the software design 20-SIM can be used to a rather limited extent.

For revealing the possibilities of parallel execution of the software components, first the model of the software part should be decomposed, such that pieces appear that can be run in parallel. Except for the controller, all submodels consist of 2 groups of independent, parallel parts. The controller consist of two parallel working PID-controllers. So treating each controller as a separate block, results in a parallel description of the controller. The resulting system is now parallelizable: In Figure 12, the vertical colored boxes denote the subsystems of the previous figures, and the two horizontal boxes denote the two parallelized parts of JIWY.

At this stage, the block diagram of the control software part can be converted to CSP diagrams (Figure 13). The communication and composition graphs are drawn, whereby the topography of the originating block diagram is maintained. It is actually a notational conversion from Figure 12 to Figure 13.

The *m*-processes in Figure 13 indicate that the process at the other end of the relationship, to which it is connected, will be executed in an infinite loop. Via the parenthesis symbols '○' and their indices, it is indicated that the upper two processes will be executed independently from the lower two processes. This way, the decoupling of the two axes is indicated.

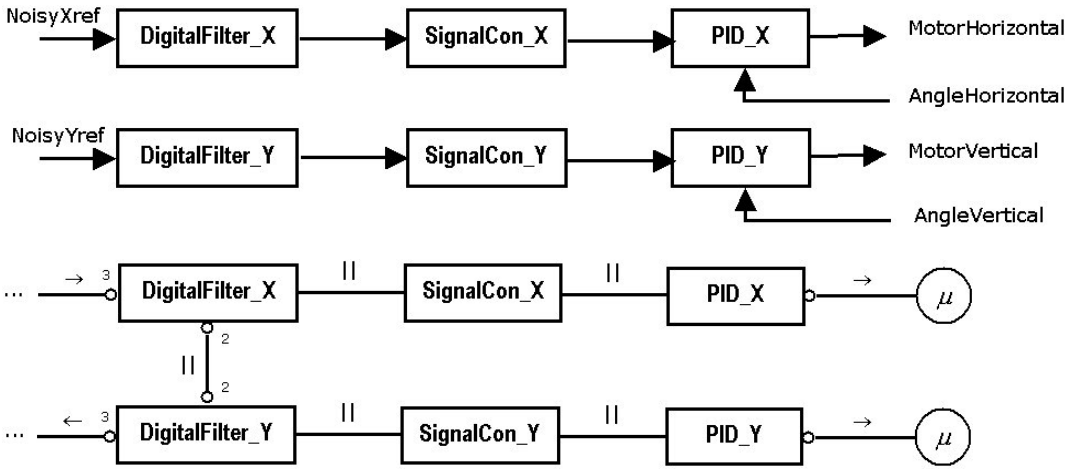

Figure 13 Communication graph (above) and composition graph (below) of the JIWY controller

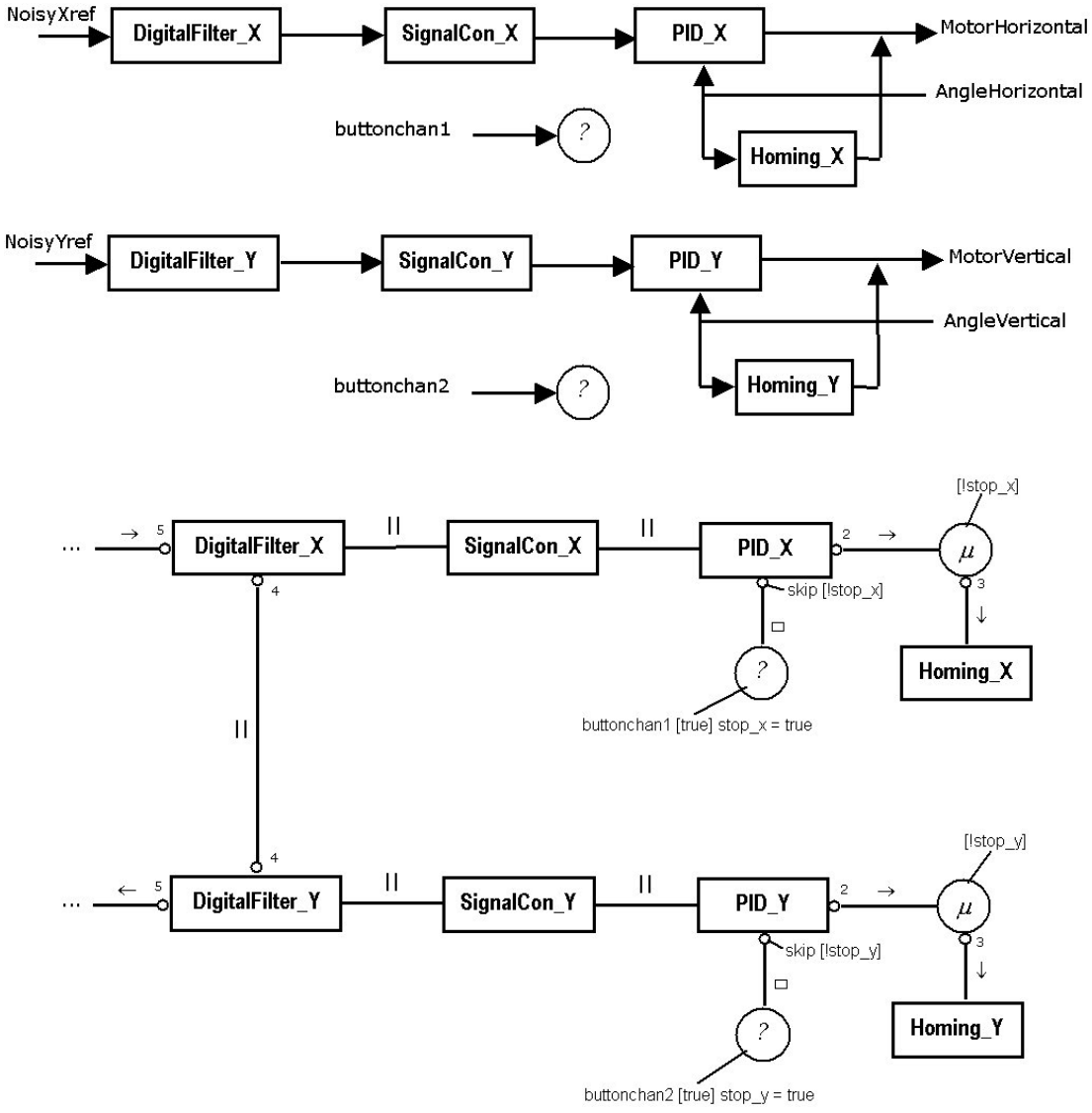

Figure 14 Stop button and homing process added

4. Incorporate safety, error and maintenance facilities

For each control loop, we designed a stop button, to stop the controller. As a reaction, we start a homing process to bring JIWY to a defined position.

The first CSP graphs (Figure 13) are extended to reflect this enhancement, resulting in Figure 14. For each loop, a choice operator (a square box) executing the control loop until a joystick button is pressed, has been added. Pushing the button stops the *m*-process via the Boolean variable *stop_x or stop_y*.

The ?-process is a primitive reader process. The reader process is conditionally related to the choice operator. The reader will read from the channel buttonchan1 when buttonchan1 is ready (i.e. a writer writes to the channel) and condition is true (i.e. [true]); otherwise the reader will not be selected. If both the channel is ready and the expression is true then the choice operator may select the reader. The reader will immediately read channel buttonchan1. Also Boolean stoph will be set to true. This is similar for buttonchan2.

IV. Concluding Remarks

The dot-framed phases of the design trajectory, as shown in Figure 15, are subject of current research and development. Hence, the main focus is put on stepwise refinement of the embedded software to be automatically generated, exhaustively verified and enabled for retargeting from one to another embedded platform.

Note that in this research, we focus on the embedded control software part. However, the CSP diagrams as shown in the last part of Section III.B can also be used to describe the communication and composition behavior of the hardware. By doing so, design space exploration on hardware-software co-design issues can be supported.

Future work mainly will with the following topics:

Analyze the scalability and effects of increasing complexity and analyzing the maintainability when new operational functionality is added to the software part.

Measure and assess real-time behavior (performance) and compliance with equipment constraints (memory footprint).

Investigate possibilities of integration of the CSP case tool under development with other graphical CAD and

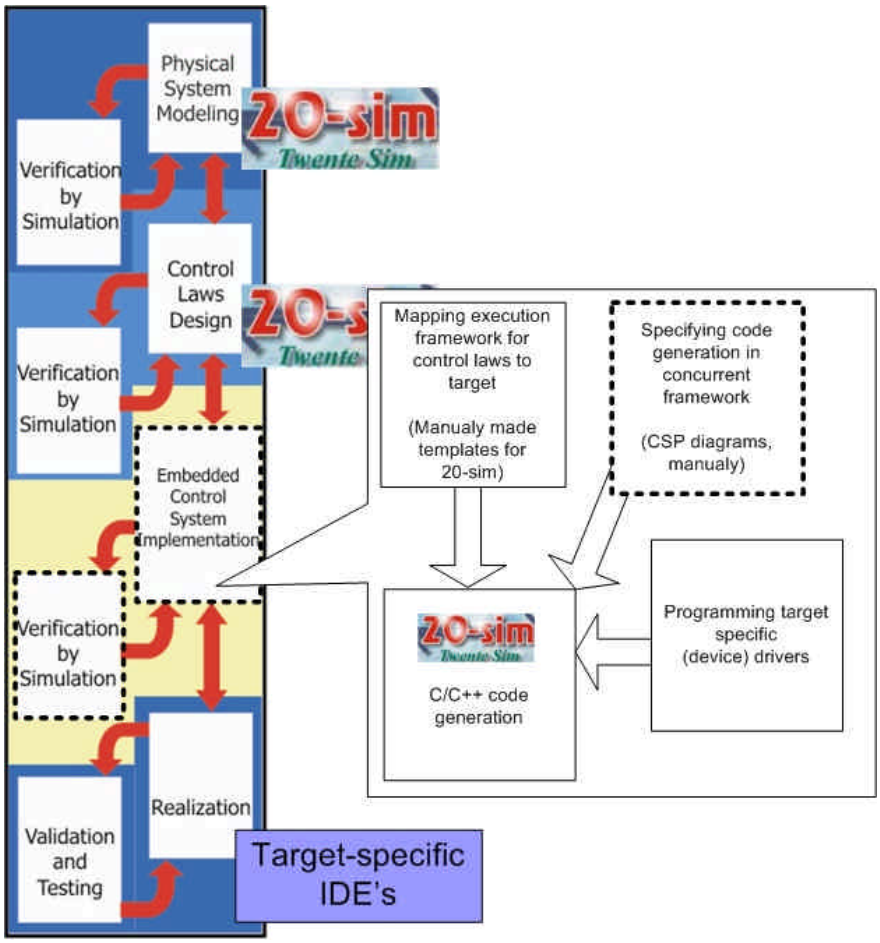

Figure 15 Current situation of the tool support

CASE tools (20-SIM, Rhapsody or RT-Rose).

Extend expressiveness capabilities of the framework to support reasoning in terms of heterogeneous distributed networked embedded systems.

REFERENCES

[1] D. Jovanovic, G. H. Hilderink, and J. F. Broenink, "Integrated Design Tool for Embedded Control Systems," presented at Progress 2001 Workshop, Veldhoven, Netherlands, 2001.

[2] J. F. Broenink and G. H. Hilderink, "A structured approach to embedded control systems implementation," presented at 2001 IEEE International Conference on Control Applications, México City, México, 2001.

[3] P. C. Breedveld and J. v. Amerongen, *Dynamische Systemen: modelvorming en simulatie met bondgrafen*. Enschede, 1996.

[4] J. F. Broenink and C. Kleijn, "Computer-aided design of mechatronic systems using 20-SIM 3.0," presented at Proc. 2nd Workshop on European Scientific and Industrial Collaboration WESIC'99, Newport, United Kingdom, 1999.

[5] -, "Matlab, Simulink" http://www.mathworks.com: Mathworks, 2002.

[6] G. H. Hilderink, "A graphical Specification Language for Modelling Concurrency based on CSP," presented at Communicating Process Architectures 2002, Reading UK, 2002.

[7] D. Jovanovic, G. H. Hilderink, and J. F. Broenink, "A communicating Threads -CT- case study: JIWY," presented at Communicating Process Architectures 2002, Reading UK, 2002.

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

[9] J. F. Broenink, D. Jovanovic, and G. H. Hilderink, "Controlling a mechanic setup using Real-time Linux and CTC++," presented at Mechatronics 2002, Enschede, 2002.

[10] R. A. Stephan, "Real-time Linux in Control Applications Area," in *Dept. Electrical Engineering*. Enschede: University of Twente, 2002.