A METHODOLOGY TO REDUCE THE DESIGN LIFECYCLE OF REAL-TIME EMBEDDED CONTROL SYSTEMS

Rémy KOCIK
ESIEE, Cité Descartes - BP 99 - 2, BD Blaise-Pascal
93162 Noisy-le-Grand CEDEX – France
E-Mail : kocikr@esiee.fr

Yves SOREL
INRIA Rocquencourt
78152 Le Chesnay CEDEX
E-Mail : yves.sorel@inria.fr

KEYWORDS

Control Systems, Real-time, Multiprocessors, Hybrid Simulation, Computer Aided Design

ABSTRACT

There are two gaps in the typical design V-process used for real-time embedded control systems. The first one is due to the manual translation of control laws into a software specification; the second one is due to the manual implementation of this specification onto distributed architecture while satisfying real-time constraints. Indeed, the correctness and efficiency of these two translations mainly rely on the skill of engineers who are subject to errors, the latter requiring many design process iterations to be corrected. We propose to fill the first gap between the modeling and the specification phases, firstly by linking the terminologies used by the two communities involved, and secondly by interfacing Scicos a hybrid simulation tool with SynDEx a specification and optimized implementation tool. The second gap is actually filled by SynDEx itself which assists the designer in implementing the control laws onto distributed architecture with the help of optimization heuristics and automatic code generation. We illustrate the proposed approach through a didactic example of inverse pendulum control system.

CONTROL EMBEDDED SYSTEMS DESIGN LIFECYCLE

Designing control embedded systems requires to solve many hardware and software problems: complex application algorithms must be implemented onto heterogeneous distributed architectures composed of processors (DSP, RISC) and specific integrated circuits (ASIC, FPGA) all together interconnected. In order to deal with this complexity, methods are often based on hierarchical design allowing to describe the system as a set of simplest, and easiest to design sub-systems. These methods are applied in the typical development lifecycle called ``V-process''. It allows to build a system starting from an abstract description to an actual product which is validated by a step by step top-down and then bottom-up design flow (Calvez 1993).

The top-down design is decomposed in three main phases: modeling, specification and implementation. The bottom-up design is decomposed in three validation phases: unit test, integration test and validation test of the system (see Figure 1). This V-process can be used both for hardware and software designs. In this paper, we focus on the software design.

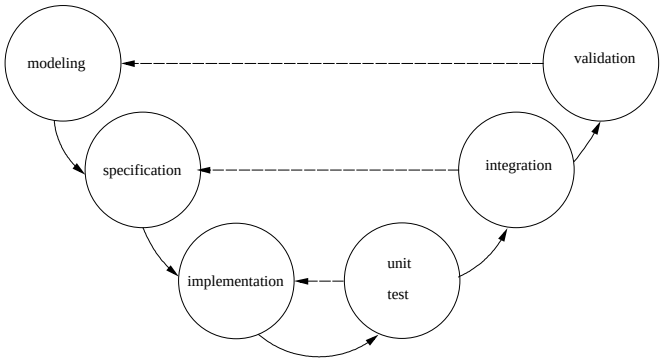

Figure 1: V-Process for Embedded Control System Design

Modeling corresponds to the mathematical description of the behaviour the system must conform to. Software *specification* consists in a high level description of the algorithms which implement the mathematical equations resulting from the modeling phase, and a description of the implementation constraints, i.e. real-time and hardware constraints. The software is actually written in the *implementation* phase in order to ensure that the algorithms will be computed on the architecture meeting the implementation constraints.

The first validation phase in the development process is the *unit test* which is often long and tedious. It consists in verifying that each software function implementing the algorithms are correctly written and perform right. When all the functions are debugged and separately validated, the *integration* phase verifies that the set of all the functions behave as defined by the model. If not, specification has to be modified. The last phase, *validation*, consists in verifying that the application behaviour is consistent with the requirements (customer's specification). If not, the model has to be modified, and usually refined. In this case the V-process is iterated as long as necessary.

GAP IN THE V-PROCESS

In the development process of a control system, control engineers define the system model and computer science engineers are in charge of the specification and implementation part. Both have to collaborate in the specification phase. This is an important phase where the mathematical model is transformed in a computing model. Problems involved by these two models are different and

errors due to misunderstanding between control engineers and computer science engineers may appear in this transformation. These errors may have impacts up to the implementation phase and then be only detected during the validation phase. Thus, development lifecycle time may be strongly increased due to these numerous backtrackings.

In this paper, we first describe, using control engineer terminology the modeling phase based on control terminology. Then, we explain using real-time computing terminology the implementation phase of the control models. Finally the links between both terminologies are made in order to reduce the gap between these two communities.

The V-process has shown a good efficiency in the design of many applications. But their growing complexity leads to an important development cost rise which became an important part of the final product cost. Thus, time spent by engineers in writing and debugging software code, is the most important part in the design process. In order to ensure competitiveness of products, it is necessary to minimize the software development lifecycle by reducing the duration of each phase, and by minimizing the number of iterations in the global V-process.

Tools based on formal languages (Halbwachs 1993) are intended to improve the global process. Indeed, these languages rely on rigorous specification based on mathematical rules. This makes possible specification, verifications, and automatic code generation such that the number of iterations in the V-process is reduced because verifications allow to detect more early specification errors. The automatic generation of a code consistent with the verified specification allows also to reduce tests and debug.

In this context, the AAA methodology (Algorithm Architecture Adequation) based on a graph formalism as been developed to optimize the implementation of application algorithms onto distributed architectures while satisfying real-time constraints (Sorel 1994). The system level CAD tool SynDEx (`www.syndex.org`) (Grandpierre and al. 1999) which relies on this methodology allows to quickly develop complex applications such that, for example, an automatic driving application for the CyCab an electrical vehicle based on a distributed architecture involving several MPC555 microcontrollers interconnected through a CAN bus (Kocik and Sorel 1998). This experience has shown the benefits that this methodology can bring for the design of complex real-time applications.

Nevertheless, this methodology takes into account only the specification and implementation part of the V-process, and there is a gap between the modeling and specification phases which may introduce errors very early in the development process. In industry this translation from modeling to specification and implementation usually relies on the skill of only few peoples. It is necessary to be vigilant because errors introduced at this level may have consequences along the development process, and are often only detected during the validation phase. Thus, correcting the error imposes a complete iteration of the development process.

Most errors usually occur during the controller model design, these errors being introduced by the translation of this model into its software specification. In order to reduce errors, to enhance ``traceability", and to minimize the number of development process iterations we propose to automatically translate models into software specifications. In the last part of this paper, we show how interfacing Scicos (`www.scicos.org`), a hybrid dynamic systems modeler and simulator, with SynDEx allows to achieve this goal.

CONTROL SYSTEM MODELING

The job of control engineers is to build a physical system able to keep under control the evolution of an other physical system called *plant*. The latter may possibly be composed of mechanical and/or electrical and/or chemical components. First the control engineer has to describe with mathematical equations th5n itecEA0 9.9(5new[eq)(Ex)4(g)6.6(o)-5.4(n)s T

moreover increase the performances of the system (Ogata 1987).

The implementation of a control law by a computer system requires to discretize input and output of the plant (Landau 1988). Thus, interactions between the computer and the plant are made using *transductors*. A feedback signal is produced by a *sensor*, which measures the amplitude of physical phenomenon and translates it into an electrical signal sampled by an analog to digital converter (ADC). The computed output is a digital signal which is converted to an electrical analog signal by a digital to analog converter (DAC) before being applied to an *actuator*. Its role consist in converting the electrical signal into an other physical phenomenon able to control the plant evolution (Figure 3) (Astrom and Wittenmark 1984).

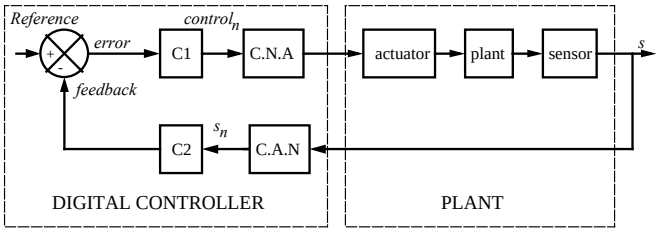

Figure 3: Computer Loopback System

The control law supplying that output signal from feedback and reference inputs is described by an algorithm implemented in a program executed on a computer. The computer seen, by the control engineer, as a simple element of the controller, is the center of interest of computer science engineers. From his point of view, the computer science engineer has to design an *application* made with a *computer system* and a physical *environment* both interacting. This system is composed of a *computer* and a set of programs, called *software*, that it will execute. The environment is defined as the set of all the physical elements which are external to the computer system. The frontier between the environment and the computer system is often difficult to establish. This is the reason why we choose here that all the physical components which can be programmed (processors, network, memory, input/output devices, ...) are parts of the computer system, whereas all the physical components which cannot be programmed are parts of the environment. The latter components, except the ADC and DAC ones, are seen by control engineers as the plant.

The computer system has to ensure that the plant behaves according to the control laws defined by the control engineer in the modeling phase. It has to interact permanently with its environment: the feedback and reference sampled input values are *input events* on which computations are performed producing *output events* which are discrete values. That is the reason why this kind of computer system is also called *reactive system* (Harel and Pnueli 1985).

The correctness of an actuation usually depends on the time elapsed between the generation of an output event and the input event which has triggered it. In this kind of systems, this time interval have to be bounded. Such applications are called *real-time applications*, the time requirements are called *real-time constraints*. In some applications, it is possible to accept sometimes that some real-time constraints are not always met, but usually these requirements are critical and have to be imperatively met. The main difficulty is to design a *predictable* real-time system guaranteeing that all the critical real-time constraints are always satisfied and the overtaking of non-critical real-time constraints will remain bounded, and casual (Le Lann 1990; Stankovic and Ramamritham 1993).

Specification

Specification corresponds to a high level software and hardware description (Calvez 1993). In the case of real-time systems these two aspects are strongly linked. In order to design the software, it is needed to specify the algorithms to be computed, the hardware architecture which will execute the algorithms, and the real-time constraints that must be satisfied (Mathai 1996).

Real-time Constraints

The discretization and digital implementation of control laws have some consequences on modeling. The control engineer has to choose sampling frequencies for each input and each output of the system (*control, s* and *reference* in Figure 3). As a general rule, in order to simplify complex computations induced by control laws discretization, the control engineer assumes that all the signals are periodically sampled. For the same reason, he assumes that input and output signals are sampled at the same time instant (Ogata 1987). Control theory books shows that this choice is empirical and relies on the skill of control engineers. The sampling frequency choice is translated during the implementation by a constraint imposed to the input events rate that the real-time system may accept. This constraint is called *input rate*. A *latency* constraint is also imposed to the real-time system. It is a boundary on the time interval between an input event arrival and the production of the corresponding output event. The goal of this constraint is to guarantee the *response time* of the system, an important quality criteria from the control engineer point of view. It has been shown, under simplifying hypothesis, that this response time is proportional to the computation latency added with a delay $T_e\backslash 2$ due to sampling performed at $1\backslash T_e$ frequency (Phillips and Troy 1984).

Algorithm Specification

Taking into account the infinite iteration due to the interactions between the real-time system and its environment (the number of iterations cannot be bounded, considered as infinite) the typical definition of an a*lgorithm* extended such as an infinite sequence of operations computed in a finite time on a finite hardware. In this way, an algorithm is seen as a finite sequence of operations computed in a finite time on a finite hardware, but infinitely repeated. The execution of every sequence of operations is triggered by input events which may be periodic or aperiodic. Complex real-time systems involve two kinds of algorithms: *data processing* and *state machine*. Data processing algorithms describe control actions applied to the plant. They represent the computations the control laws perform on data (PID corrector, filter,...) considered as periodic events

because they are sampled at the input rate. State machine algorithms define in which order data processing algorithms must be executed according to the current state and aperiodic input events.

Hardware Specification
Embedded applications are subject to strong cost constraints (financial cost, dimensions, electrical consumption). To meet these constraints, new hardware architectures have been proposed. For example, in automotive industry where competitivity is particularly hard these new technologies have contributed to reduce the amount of wiring thanks to the integration of sensors and actuators near to the processor, and to data multiplexing on serial bus. This lead to heterogenous distributed architectures with low cost components of the shelf. Computers are build with microcontrollers to perform evolving functions, and with ASICS, and/or FPGA to perform some specific functions with only few evolutions in the product life. Communications between processors are supported by low cost serial buses (like VAN, CAN, TTP, FlexRay,…) well suited for disturbed environment.

DESIGN LIFECYCLE REDUCING

In order to meet the implementation constraints and to avoid gaps between all the V-process phases and thus reduce design costs we propose a methodology based on the cooperation of two tools: Scicos for modeling and hybrid simulation, and SynDEx for specification and distributed implementation.

Specification

In computer control systems sampling and quantifying operations, needed for the discretization of analog signals, may introduce some errors which tend to degrade system performances. For the same performance level, if we want to take into account these errors, it is more complex to define the control law for a computer system than for an analog one. The modeling of a discretized system is more complex than the modeling of a continuous system for which control theory brings numerous mathematical tools.

In order to simplify the design, the control engineer usually studies a continuous model of the controller rather than a discrete one (Astrom and Wittenmark 1984). Then, this continuous model is discretized in order to allow its implementation on a computer. Thus, an hybrid simulation, i.e. the simulation of a continuous system (the plant) linked with a discrete real-time system is mandatory to validate the model used to design the real-time system. This simulation allows to tune the control laws in order to take into account approximations done in the continuous model.
Scicos (Nikoukha and Steer 1999) has been designed for this purpose, it allows an hybrid simulation taking into account the sampling frequencies of analog signals.

Specification and Optimized Algorithms Implementation

Discretized control laws and continuous plant models are described in Scicos with graphs similar to block diagram well known by control engineers. Such a graph representing a discretized control laws is extracted from Scicos and translated into a SynDEx graph (Djenidi and al. 1999). In this way, the algorithm specification conforms to the model, it is still not necessary to spent any time for the control laws specification, and only hardware architecture description is now required. SynDEx allows to perform this hardware specification, and when both algorithm (control laws) and architecture are specified it allows to execute the adequation, that is to say to execute heuristics which optimize the implementation of the algorithm onto the architecture and automatically generates the corresponding code that will be executed in real-time. Actually, interfacing Scicos with SynDEx allows to cover the complete design process.

Hybrid Simulation with Delays

Hybrid simulation, to be close to reality, needs to take into account delays introduced in the control laws by computation and communication durations (latency). These delays may impact on system stability(Torngren 1990). They depend on the implementation, that is to say on the allocation (distribution) and the scheduling of functions, associated to each block of the graph describing the control laws, onto processors.

Usually, in the specification phase each block is translated into a *task*, function with real-time execution constraints and properties (execution periods, priority, deadlines,…). Then, the schedule of this set of task is made at execution time (on-line scheduling) by an RTOS (real-time operating system) according to the properties of the tasks (Timmerman 1999). In this approach it is difficult to predict computing latency, and consequently it is not possible to easily take into account delays in the hybrid simulation. This problem can be solved by interfacing Scicos and SynDEx. Indeed, SynDEx is able to compute an adequation which is an optimized distribution (spatial allocation of CPU resources) and scheduling of the blocks (called in this context *operations*) of the algorithm onto the hardware architecture. Thanks to a good knowledge of the architecture (number of programmable and non programmable components, number of communication media) and to the knowledge of the maximum execution duration of each operation depending on the component able to execute it, SynDEx can compute and visualize a prediction of the optimized distribution and scheduling. This accurate prediction allows to verify whether real-time constraints are met or not, and also allows to provide back, in the hybrid simulation, the delays due to computations and communications. Thus, it is possible to simulate the hybrid system with Scicos in order to verify that response time, stability and others requirements are met in spite of these delays.

Example

In order to show the benefits of this approach we illustrate it with a didactic example of an inverse pendulum control system. The inverse pendulum is a system composed of a cart (mass M) on which a bar (mass m, length l) is linked. The bar can rotate around its extremity. The cart stays on a

inclined plane. The goal is to stabilize the cart at the origin position *O* (Figure 4).

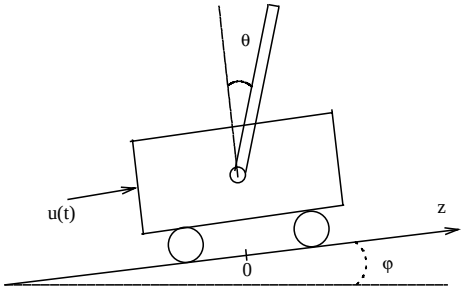

Figure 4: Inverse Pendulum Stabilization

We suppose that some sensors provide the angular position θ and the linear position *z* of the cart on the plane. The control action is a translation force *u(t)* produced by an electrical motor.

Modeling and Simulation

The first phase corresponds to the modeling and the simulation of the system in order to verify that the goal is met. Figure 5 shows how the modeling was performed with Scicos. The block named `cart` is a mathematical continuous model of the cart. This block is a function written in C or Fortran code and/or with Scicos library blocks. This model estimates *z* and θ from the angle φ (0.001 radians here) and from the control action *u(t)* applied to it. The `sensor` block models two sensors acquiring *z* and θ. The block named `actuator` is the electrical motor model. The `control` block is the discretized model of the controller. It describes the algorithms which will be executed in real-time.

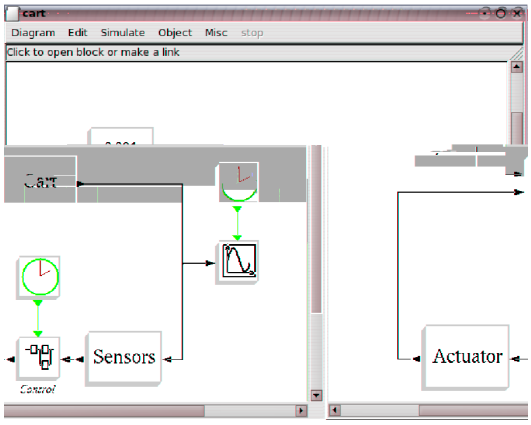

Figure 5: Scicos Graph

The sampling period of its inputs is 10ms, it is specified by a clock signal connected to its activation input port. The control engineer has used here a hierarchical description, the `control` block is a Scicos *super-block* which groups many C or Scicos blocks. Finally, a specific block (icon showing a 2D graph) allows to display a simulation graph of the *z* position and of the θ angle evolution.

Control Algorithms Extraction

When hybrid simulation shows that the control system is properly designed, the engineer may extract from Scicos the controller algorithms by selecting with the mouse the involved region (Figure 6).

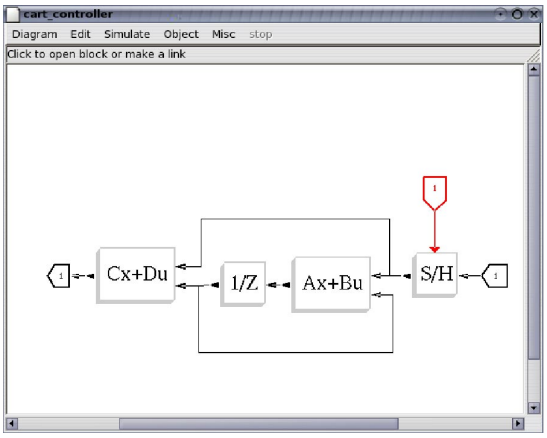

Figure 6: Controller Extraction

We can see that the control engineer chose here to design the control law as a matrix computation given by a state space representation. This Scicos graph region of the discretized control law can then be automatically translated into a SynDEx algorithm graph.

Real-time System Specification

The algorithm graph extracted from Scicos is translated in the SynDEx algorithm syntax. Now, the user must specify the architecture graph describing the hardware architecture of the real-time system and he must provide to SynDEx the worst execution duration of each operation of the algorithm graph. These durations can be, in a first step, estimated by engineers or they can be measured by SynDEx after a first execution of the automatically generated code.

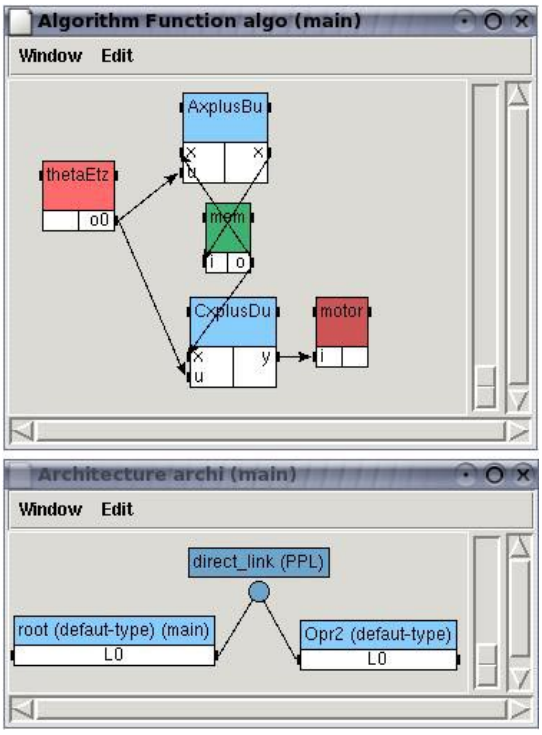

Figure 7: SynDEx Software and Hardware Graphs

The Figure 7 shows the SynDEx graphical interface where both algorithm and architecture graphs are displayed. On the algorithm graph (upper part), we can see the Scicos blocks `AxplusBu` and `CxplusDu` performing the matrix computation. The block called `thetaEtz` corresponds to the sampling of the inputs sensors while the block called `motor` drives the motor. The block `mem` specify the scicos block `1/Z`. For demonstration purpose, we have supposed that the hardware architecture is made of two processors linked together by a communication media called `direct_lnk` (Figure 7 lower part). Sensors are physically linked to the processor called `root`, the motor is physically linked through a power amplifier on the processor called `Opr2`.

Adequation and Temporal Simulation

From this specification, SynDEx can perform the adequation and display a temporal prediction of the algorithm execution onto the hardware architecture (Figure 8). Each column represents an execution sequence displaying one iteration of the operations distributed onto a processor, or of the data dependences distributed onto a communication media. In our case, there is one computing sequence on the `root` processor, another computing sequence on the processor `opr2` and a communication sequence on `direct_link`. The vertical axis represents the time evolution. It can be noticed that the complete execution of all the operations and all the communications needed is estimated to 3ms (3060). Thus, we can verify that the architecture can meet the temporal constraint of the input rate (10ms) given by the sampling period chosen for *z* and θ.

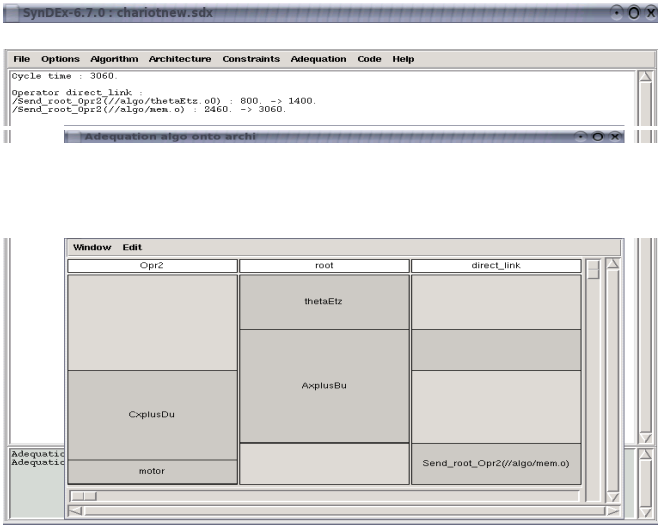

Figure 8: SynDEx Timing Simulation

Hybrid Simulation Taking into Account Delays

The computation latency (3ms) given by the SynDEx timing simulation is the delay between the sampling of *z* and θ and the *u(t)* output. This delay can be introduced in the Scicos graph in order to perform a more accurate hybrid simulation.

The Figure 9 shows the first simulation made without any delay. It exhibits that the control law discretized with a 10ms sampling period allows to stabilize the inverse-pendulum at *O*. Figure 11 and Figure 10 are the simulation results taking into account the 3ms delay. The first one (Figure 10) shows that with a 10ms sampling period, the system with the delay is no more stable. The second one (Figure 11) validates the control system stability with a control law discretized with a 8ms sampling period despite of the delay.

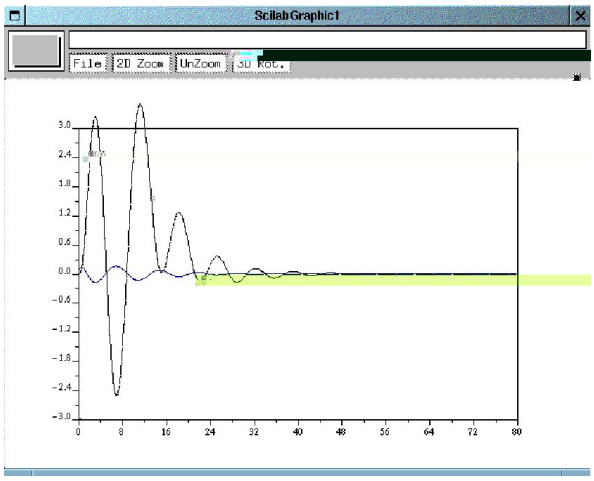

Figure 9: Simulation with 10ms Sampling Period, No Input/Output Delay

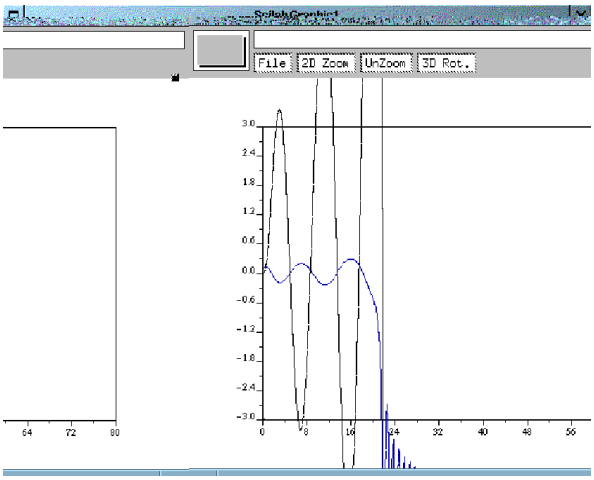

Figure 10: 10ms Sampling Period with 3ms Delay

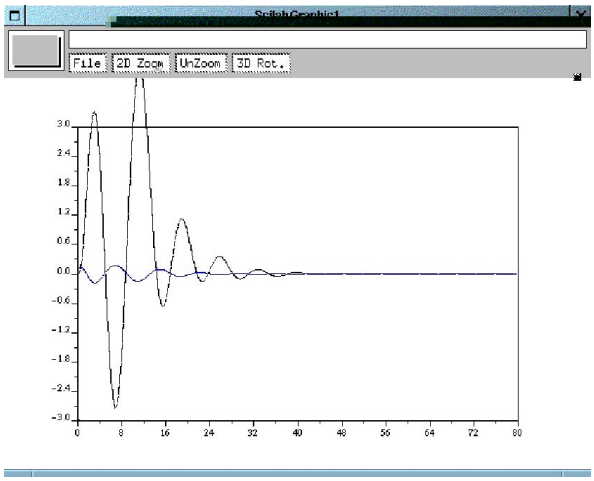

Figure 11: 8ms Sampling Period with 3ms Delay

Automatic Code Generation

When both hybrid (Scicos) and temporal (SynDEx) simulations show that the performances are satisfied, it is possible with SynDEx to generate the code that will actually be executed on each processors. SynDEx produces an executive involving inter processor communications which guarantees that the functions calls associated to each operation of the algorithm graph, and to each `SEND` and `RECEIVE` communication primitives, follows the computed schedule. The software engineer just has to provide the input and output functions (`thetaEtz` and `motor`). It is possible to reuse the C blocks used in Scicos to describe the control system: thus, the code that have been used for simulation is the same that the one used at execution time in the real-time system.

New Reduced Lifecycle

Figure 12 shows the new reduced lifecycle obtained using this methodology. Each phase is validated by simulation or verification before going to next the phase. Manual translation of models into specifications are minimized in order to avoid errors. We hope by this way to lead to a lifecycle closed to an ideal waterfall process without rise: each phase is validated before going to the next one.