Model-driven Methodology for Real-Time Software Design

Rédha HAMOUCHE, Rémy KOCIK
Université Paris-Est, ESIEE Paris, Embedded Systems Department
Cité Descartes - BP 99 - 2, Bd Blaise Pascal - 93162 Noisy-Le-Grand Cedex, France
e-mail: {hamouchr, kocikr}@esiee.fr

ABSTRACT

This paper presents a model-driven methodology and tool for Real-Time Embedded Control Software (RTECS) design. This methodology leads to evolve the RTECS development process from a classical code-oriented development to a model-driven development where the code is generated automatically. It uses a component-based and aspect-oriented approach. The component approach should add significant value to the design process, helping the software designer to produce modular and reusable system model. The aspect approach represents a significant paradigm shift from the traditional monolithic view. It provides a better ability to model extra-functional properties of RTECS. Both approaches lead to make the RTECS design significantly easier and improve the model accuracy and reduce the design time and cost. A tool called MoDEST implements this methodology and provides automated model transformations and real-time code generation. It should help designers to analyse system models.

Keywords: Model-based design, software component, aspect-oriented programming, metamodeling, embedded system design, embedded control software.

1 INTRODUCTION

Real-time embedded control systems are ubiquitous nowadays. They are used in a broad spectrum of applications, from simple temperature control in household appliances to complex and safetycritical automotive brake systems or aircraft flight control systems. The design of real-time embedded control systems tends to be more and more difficult due to the strong constraints (cost, price, energy consumption, control performance,...) and the growing complexity of the used software and hardware components. The software deals with different application domains (video processing, signal processing, telecom, ...), implemented onto heterogeneous distributed architectures composed of processors (DSP, RISC) and specific integrated circuits (ASIC, FPGA).

As shown in figure 1, designing RTECS follows usually a V-process, which allows building a system step by step according to top-down and then bottom-up design flow. A typical V-process for real-time embedded control systems design can be decomposed into three major steps [5]: functional control modeling and analysis, specification of control software and real-time implementation of control software. The result of each step is a model which is refined in the next one, until the implementation step. The functional system modeling and analysis, usually performed by control designers, is a modeling step where the behavior of plant and controller (control law) is described by a mathematical model, using dedicated languages and tools (like Matlab/Simulink or Scilab/Scicos), in order to model and analyze the plant and synthesize the control law. The specification of control SW is usually achieved by computer science engineers in order to implement the mathematical equations resulting from the modeling step, to capture the structure of the software, and to specify the high-level implementation constraints (input/output latency, jitter, etc.). The real-time implementation of control SW, realized by computer science engineers, corresponds to writing the controller as real-time software that will be executed on the hardware, and meets the implementation constraints. Usually, this implementation is described as a set of tasks, functions with real-time execution constraints (period, priority, deadline, etc.) and properties (execution duration, memory footprint, etc.). These tasks are scheduled in order to guarantee they meet

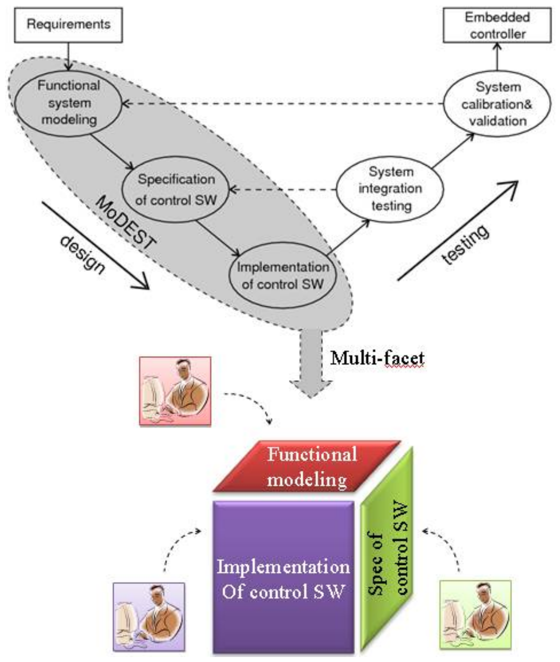

Requirements
Embedded controller
Functional system modeling
System calibration& validation
Specification of control SW
System integration testing
MoDEST
design
testing
Implementation of control SW
Multi-facet
Functional modeling
Implementation Of control SW
Spec of control SW

Figure 1. The design process using MoDEST

the specified real-time constraints.

Along the design process of RTECS, several actors belonging to different domains (control theory, signal processing, real-time software, ...) are involved. They use domain-oriented method, languages and tools. Therefore, produced models at the different steps are heterogeneous. They differ mainly by the specified functionality, real-time constraints and models of computation (MoC) [6]. This heterogeneity of models introduces lacks of consistency in the system design and leads usually to a disconnected design process where translations between the steps are needed. These translations are error-prone because they are usually performed manually by engineers. These errors may appear very early in the development process and be propagated into further steps. They can be only detected in the validation steps. Thus, correcting these errors needs numerous backtrackings (reiterations of V-cycle) in the design process, which lengthens the design lifecycle and time to market. Furthermore, the inconsistency of models may affect the implementation of the control system by disturbing its stability and reducing its performance [9][1].

Nowadays, there are research challenges to define methodology and tools which reduce the design time and cost, and ensure the consistency of models from system level to implementation level. Tools that allow a control design and real-time design are quite a few. Ptolemy, Jitterbug [3], TrueTime [7], SynDEx [10] are examples of such tools. Unfortunately, they are, in general, specialized on a certain aspect of the co-design problem or impose some restrictions [8]. Ptolemy tool, for instance, imposes some restrictions by the timed-multitasking model of computation. It only facilitates simulation of fixed priority scheduling of tasks with constant execution times [8]. Another approach like AADL (Architecture analysis and Design Language) is emerged recently for a high level design and evaluation of the architecture of embedded systems. They are powerful but the complexity of models can be higher. In fact, the different real-time/embedded concerns (safety, security, real-time constraints, scalability, etc.) are mixed in the models. When one of the concerns of the model needs to be modified, manipulating the parts of the model related to that concern can prove to be challenging since these parts are mixed with the elements from other concerns [4].

In this context, we have defined in [6][5] methodology and tool for design real-time embedded control software. Section 2 describes this methodology. Section 3 presents a model-driven tool, called MoDEST, and shows the benefits of this tool via a case study. Finally, the paper is concluded in section 4.

2 MODEL-DRIVEN EMBEDDED SYSTEM DESIGN

As depicted in figure 1, the proposed methodology offers a multi-facet design where the system model is viewed on different design facets as complementary. Each facet is well suited to the problem of each design step. It provides domain-oriented toolset for building model, using specific terminology (control, computer science or real time), by the corresponded actor (control designer or real time software designer). This design approach leads to unify the design steps into a homogeneous approach, with handling the complexity and the heterogeneity of models. It allows to link the functional system modelling step with the real-time implementation step, and it then provides an efficient real-time implementation of models with the control performance in mind. This is in order to evaluate the system performance and stability during their design. We aim to define real-time scheduling policies taking better account of control models constraints (sampling periods, input/output latencies, jitters...). On the other hand, we seek to take into account the temporal characteristics of the implementation in the hybrid simulation. The consistency between the design steps is ensured using the a model-driven development (MDD) [2] approach.

MDD approach

The main goal of the methodology is to evolve the RTECS development process from a classical code-oriented development to a model-driven development where the code is generated automatically. The automatic transformation of models improves their consistency and traceability throughout the development cycle. This consistency is achieved through the notion of metamodel. A metamodel is a description pattern which is able to capture concepts used in system models. The proposed metamodel is able to capture the three facets description by defining an unified terminology and semantics to share information without having to duplicate it. As depicted in figure 2, each change in a facet description updates another one by carrying out model transformations and reconciliation between the facets, which guarantee consistency of models in the earlier stages of design. The metamodel also allows to build bridges with external approaches and tools, as well as code generators to multiple targets. The proposed metamodel is based on two paradigms: software component and aspect-oriented programming.

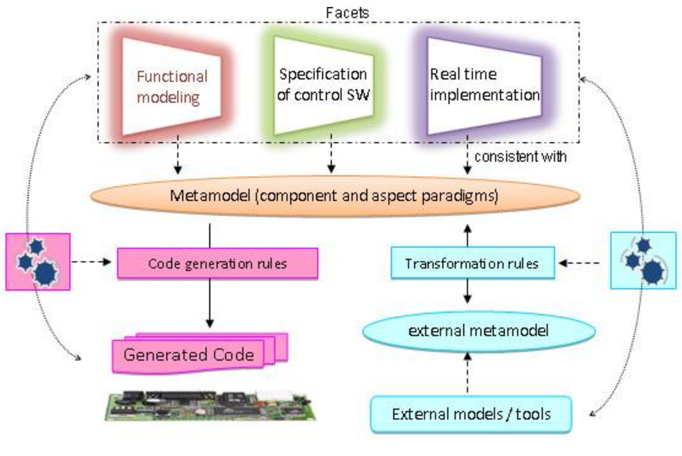


Figure 2. Metamodel and Facets

Component Approach

To offer a better level of abstraction and strengthen the construction of reusable modules, we adopt a component-based design approach [11]. This representation of the software architecture brings with it all the advantages of modern thinking in software development. Designs are instantiated largely by navigating and choosing pre-written configurable components from libraries rather than by implementing the design from scratch. The reusability of models should be increased despite the fast technological evolution of embedded platforms and the design time should be reduced.

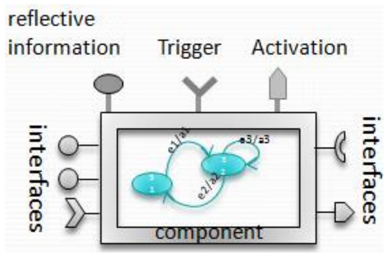


Figure 3. component model

As shown in figure 3 and 4, a new component model that is lightweight and that addresses explicitly the real-time properties of embedded systems is defined. It interacts with the environment through its interface (exit points of component interactions with its environment), it is characterized by properties stored as reflective information (such as worst-case execution time, jitters, memory footprint or location of its source or binary code), and its internal behaviour described by sub-components and a real-time automata (statechart).

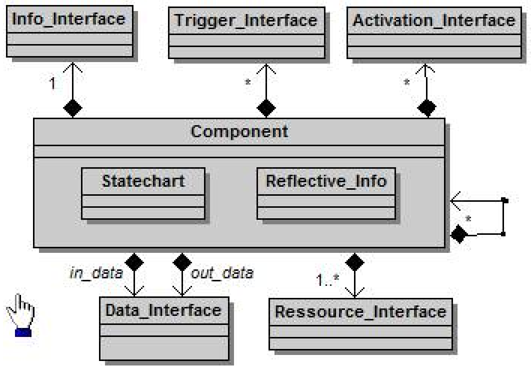


Figure 4. Metamodel of a component

Aspect Approach

A RTECS is characterized by its dependence on the execution context (the environment in which the system is operated). It reacts to the context changes (sensors, operators, . . .), and its behavior is constrained by the context (real-time, embedded, security or energy constraints, physical constraints, . . .). A major properties of a RTECS are context-dependent. Real-time, embedded, security, energy or physical constraints are an example. While designing and modelling RTECS, we have to overcome the following difficulties: (1) The major properties of RTECS, such as reliability, security, schedulability and synchronization, are global and transverse to the system and they cannot be cleanly encapsulated in a generalized procedure. Therefore, if the system analysis shows that the design is not schedulable, it is necessary to re-design and go back to models or codes to make some changes. Navigating and applying changes to models/codes are increasingly difficult as the models/codes grow more complex; (2) As we have to treat various kinds of context, it is difficult to model them from one point of view; (3) By its nature, the model for internal processing tends to depend on the model of external context, and changing the context, causes direct effects on internal model. This make reusability, modifiability and extensibility of RTECS not efficient. On the other hand, in the system design and analysis, the most difficult issues are ensuring that extra-functional properties such real-time performance are being met. What is required is a mechanism that make consistent and global changes, and offers better modularity and ability to analysis extra-functional properties. This is one value of the aspect paradigm.

The aspect-oriented approach addresses the separation of functional and extra-functional aspects in an application development, in order to avoid the usual interweaving between those aspects. Application is manipulated regarding specific aspects, rather than in its whole. Therefore, modifying any functionalities or extra-functional properties does not lead to change the whole application description. This separation gives a new dimension of modularity, reusability and maintenance, and increases configuration capacities. Unlike in monolithic development, this approach offers a better ability to analyze extra-functional properties. In the literature, an aspect is defined at the programming language level. For example, AspectJ [12] provides syntax that permits the specification of aspects and a weaver that weaves the code specified in the aspect into the base Java code. We extend the concept of aspects and apply them at the design level. Our aspects are defined as an extra-functional entities which can be applied to the facets, not to source code, in a transversal manner. An example of key aspects for embedded software systems are: security aspect, energy aspect, platform aspect, scheduling aspect, temporal aspect, profiling aspect,... By this way, designers are encouraged to describe system facets in a functional manner and then to apply extra-functional updates to the design in a global and consistent manner.

As shown in figure 5, an aspect may crosscut an facet of the system in order to affect behavior of the system model, change its semantic or its performance. It crosscuts components of a model for adding extra-functional treatment and/or imposing non-functional constraints. We distinguish four ways of crosscutting : (1) to supervise and control the behavior of components via *trigger interface*. An aspect uses this interface for sending context-sensitive

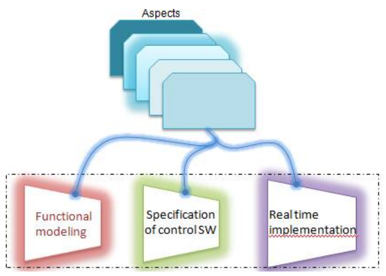


Figure 5. Aspects and facets

events (urgent event, operator command, device failure or dysfunction) and making then the statechart of components context-sensitive without a strong coupling with the context; (2) to constrain the components behavior by the context requirements. Via an interface called *activation interfaces*, the designer can introduce constraints in the component statechart as guards or invariants. Some system functionalities don't then work for example even if users try to operate it. An aspect may also introduce communication constraints on the interfaces of the component to impose a communication mode (synchronous or asynchronous) or a rate of data production/consumption; (3) to control data of components via an interface called *introspection interface*. An aspect can control access to certain data of components for managing concurrent access, verifying the data values for security reasons, and so on; (4) to define the resource constraints (via *resource interface*) associated with the target execution platform of the system. An aspect can define timing constraints such as period, deadline and jitter. It can crosscut components for defining the WCET (worst-case execution time) of component operations. This resource information is needed at design level to evaluate the system schedulability for specific execution platform. Change a target execution platform consists then in replacing only the corresponding aspect by another one specific to a new execution platform.

Component/Aspect weaving
The model of each facet is built by an assembly of components, and then by a weaving of those components with a set of aspects (see figure 6). The components are interconnected via data interfaces and they are crosscutted with the aspects via aspectual interfaces: trigger interface, activation interface, interface of reflective information, and resource interface. The weaving is described by a declarative language which allows the declaration of weaving rules as follows:

```
WR1 : IF (bool_expr) THEN
Aspect.TriggerInterface->GEN(event)

WR2 :IF (bool_expr) THEN
Aspect.ActivationInterface = bool_value

bool_expr ::= bool_expr op
IS_IN(Aspect.State)
bool_expr ::= IS_IN (Aspect.State)
op ::= and | or
```

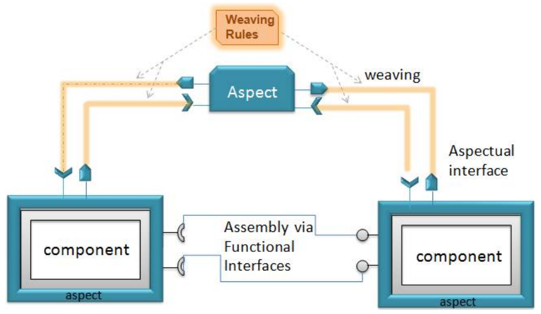


Figure 6. Aspect and components weaving

```
bool_value ::= true or false
```

An example of weaving rules is described in section 3. In our methodology, those weaving rules have the benefits to built an analysable or executable model. In fact, weaving rules allow building global conditions, based on the statecharts described in the different facets. This constitutes global invariants which serve for validating the system with external tools (model checking tools for example).

3 TOOL AND CASE STUDY

MoDEST
For evaluating the proposed methodology, a new toolkit called MoDEST (Model-Driven Embedded System design Tool) is currently under development. It implements the proposed methodology and offers a design environment ranging from control algorithm modeling to code generation on a single-processor target This tool is not doing what other domain tools done, but enabling to build links between them in order to support in a same environment the whole design process of embedded systems. The tool supports the importation of control models and provides helpful construction of embedded software specification and its real time implementation with careful consideration of model transformations. The MoDEST tool provides facilities for scheduling analysis and simulation as well as the code generation for multiple targets.

Case Study
As an example we consider the software design of a DC electrical motor speed control system, such as the control of a robot motor. Usually, the control law of such system is based on a well known PI algorithm. In this example, we will see how this algorithm can be specified and implemented with MoDEST.

Functional System modelling: The functional view of the PI controller is shown in figure 7. In the first step, the control law provided by a control engineer must be imported in the functional view of the MoDEST tool. The control law algorithm is here described using a graphical description language. The formalism of this language is well known to control engineers, since it is similar to that used by the tools for design and simulation of the control laws. The development of a gateway for an automatic import from Simulink is in progress.

The controller algorithm is here composed of 5 main functional blocks. The first block called `Input_speed` defines digital sampling of the motor measured speed. The blocks `P`, `I`, and `ADD` define the PI algorithm, while the block `motor_cmd` applies the computed voltage to the motor. For a demonstration purpose we have added 6 more blocks (`in1`, `in2`, `control1`, `control2`, `out14`, `out15`) in order to introduce temporal perturbations on the PI algorithm execution. We can see on the graph that sampling rate of `Input_speed` and `motor_cmd` is defined by a clock `CLK1`. It can also be noticed that 2 other clocks (`CLK2` and `CLK3`) are used to define sampling rate on `in1`, `in2`, `out14` and `out15`. In this example `CLK1` define a 1000ms clock, `CLK2` define a 800ms clock and `CLK3` is set to 500ms.

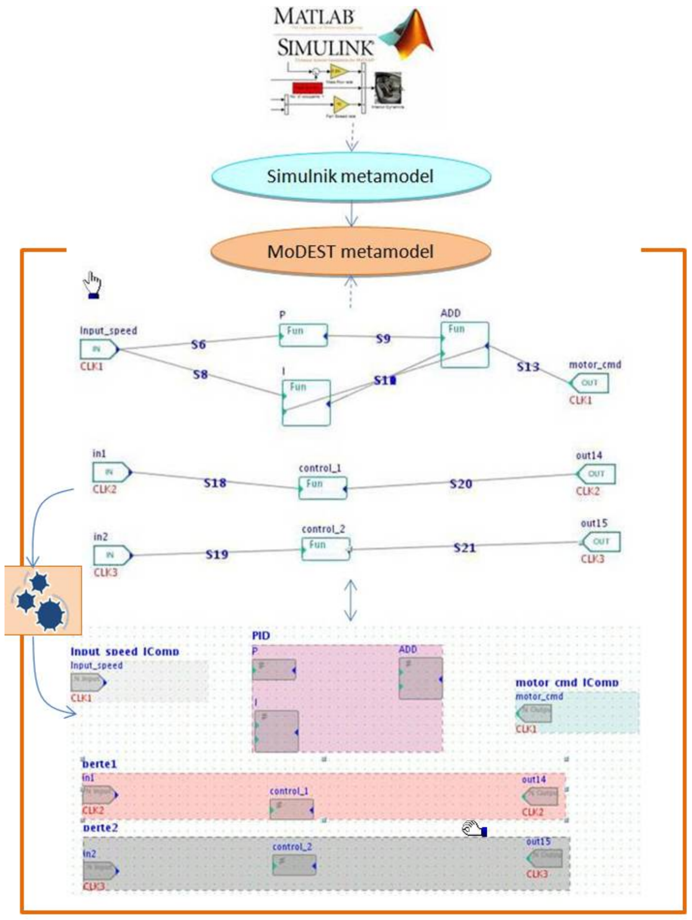


Figure 7. PI example: MoDEST functional view.

Specification of control SW: From the functional description, The computer engineer specifies the software functions which will implement the functional blocks. This is achieved using the MoDEST specification view. This view defines a mapping of functional blocks into implementation components (*IComp*) which will be executed at runtime. Figure 7 illustrates this mapping for our PI controller. Here, the input sampling block `Input_speed` will be implemented by an IComp named `Input_speed_IComp`. The 3 blocks `P`, `I` and `ADD` will be both implemented by only one IComp called `PID`.

Implementation of control SW: The third view of MoDEST allows the user to define the real-time software components and the runtime software context. First, it necessary to define the *tasks* which will support the real-time periodic execution of the *IComps*. Several *IComps* can be ordered in a *sequence* to be executed in the same task. The set of task can be defined manually by the user but it can also be automatically generated by the tool taking into account the *IComp* and clocks characteristics. On the example (see figure 8), a control task has been generated by combining the IComp of the same clock: `Input_speed_IComp`, `PID` and `motor_cmd_IComp`.

In this facet, three extra-functional aspects are defined in this view. The energy aspect constrains the functionalities of the task: if the battery level becomes low, the task should be in the limited control state. The security aspect is a watchdog which resets the task if its behaviour diverges. The platform aspect defines the runtime context of the task. The user should define a platform aspect which provides several information on the software and hardware architecture where the application should be executed (processor type, operating system, compiler, software libraries,...). The execution duration of each IComp (Input_speed_IComp, PID and motor_cmd_Icomp) depends on these parameters. When an IComp has never been used on the described architecture, the user must also provide an estimation of it. This estimation should be refined with measurements by the profiling aspect.

Analysis and Simulation

For a real-time implementation of tasks, the designer can define a scheduling aspect in order to crosscut the tasks with information related to a scheduling policy. At the design step, this aspect may adjust real-time properties such as the tasks priority. At the code generation step, it allows to add into the generated code of the tasks the OS functions which are appropriate to the selected scheduling policy. The MoDEST tool provides an automated schedulability analysis for assess the schedulability of the design without implementing the system. The result of the analysis is whether the system is feasible or not in the worst case. This analysis provides the processor charge and the response time to show if tasks meet their stipulated timing constraints. The tool implements several classical real-time scheduling algorithms such as Rate Monotonic, Deadline Monotonic and Earliest Deadline First. To reduce design time, it can automatically identify a list of applicable scheduling analysis that matches the system characteristics (Operating System, task dependencies). The results of scheduling are drawn graphically with chronogram chart as depicted in figure 8. From this simulation, MoDEST is able to perform a temporal behavior analysis by using temporal aspect. Latencies between inputs and outputs of the system, delays and jitters on tasks and IComp executions are measured and displayed as histograms. By

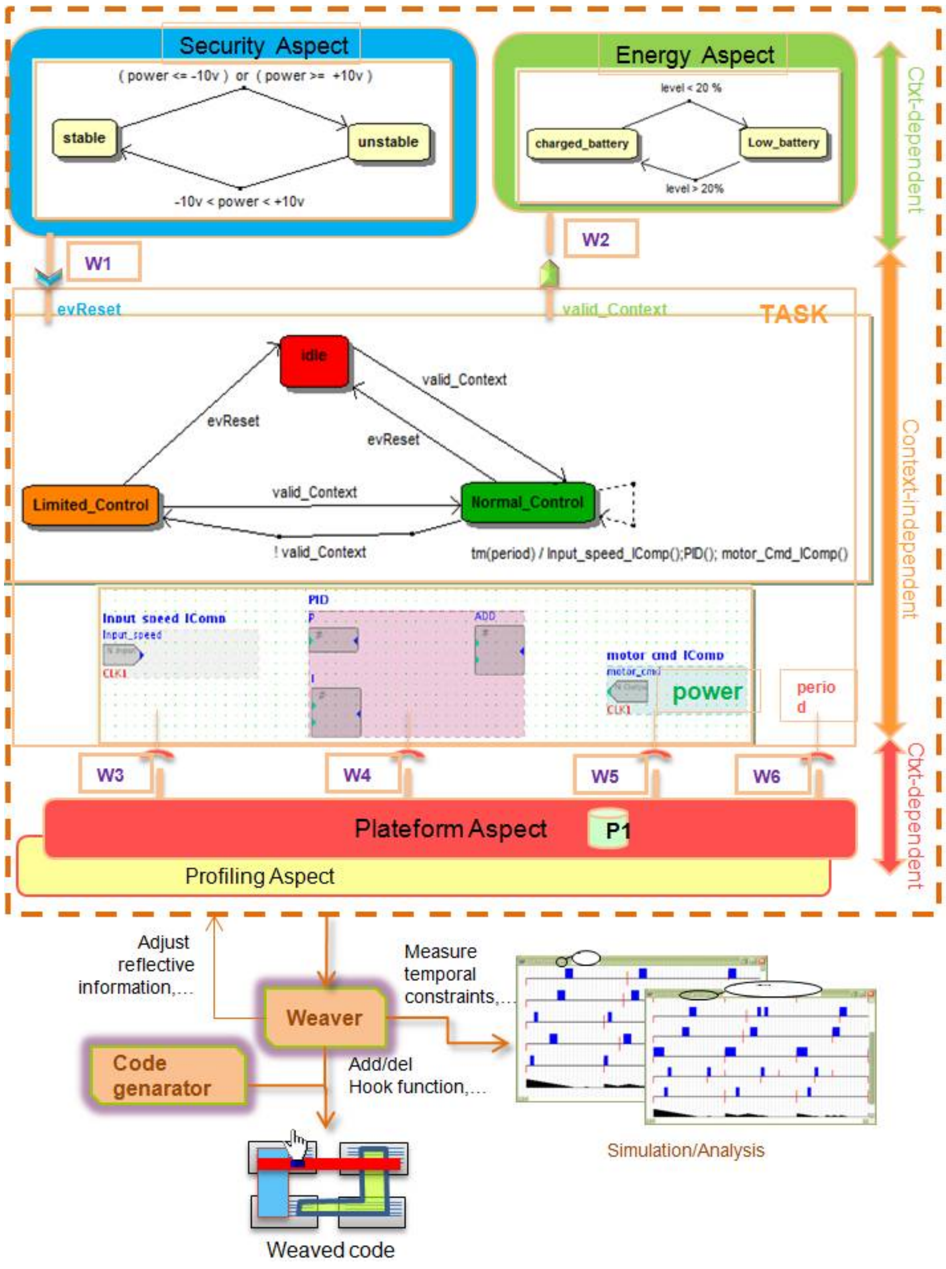


Figure 8. PI example: MoDEST implementation view

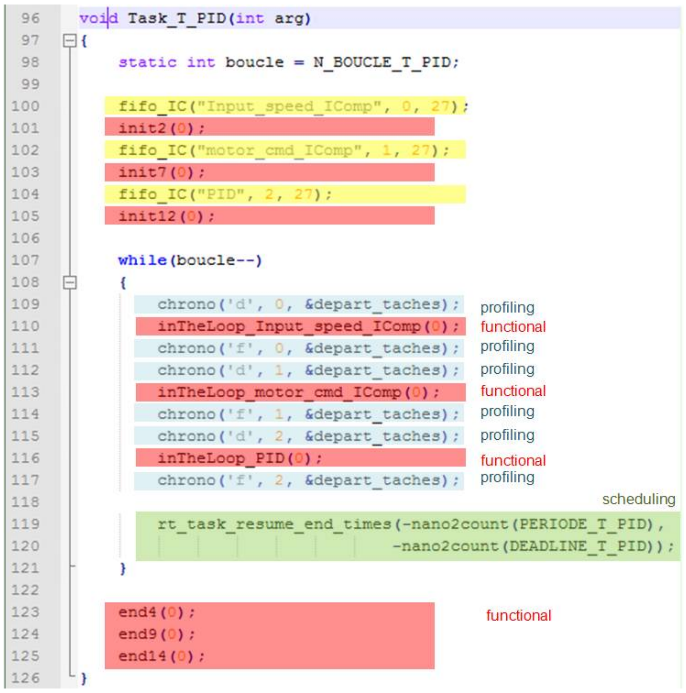
this way, the control designer can evaluate the jitters and delays induced by the real-time implementation. Taking into account these delays and jitters, he can perform some hybrid simulations to choose the scheduler which induces the best control performances.

Weaving, code generation and profiling
The weaving between the aspects and the task component is mainly described through the rules W1-W6 as follows:

```
WR1 :  if (IS_IN(unstable) then
GEN(evReset)
WR2 :  if (IS_IN(Stable) and
IS_IN(Charged_Battery) ) then
valid_Context=true
WR3 :  if (chosen(P1) then
wcet(Input_speed_Icomp) = 100ms
WR4 :  if (chosen(P1) then wcet(PID) =
200ms
WR5 :  if (chosen(P1) then
wcet(motor_cmd_Icomp) = 100ms
WR6 :  if (chosen(P1) then period = 200ms
```

The rule WR1 specifies that the task is reset (event evReset) if the robot becomes instable. The rule WR2 constrains the task to control the system only if the security constraint and the enregy constraint are met. The rule WR2 constitutes a system invariant to check the model with an external tool for model checking. The rules WR3, WR4, WR5 and WR6 define the code source path of IComp (Input_speed_IComp, PID and motor_cmd_Icomp) within the platform P1. As shown in figure 8, these rules constitute directives for the weaver and the code generator.

The MoDEST tool supports currently the code generation on a single-processor target. From the implementation facet, the designer can generate code on a target C/RTAI, real-time Java, and C on DSPBios kernel of Texas Instruments. The figure 9 shows an example of C/RTAI generated code concerning our case study. The PID task is generated as periodic task scheduled with EDF policy, and weaved with a profiling aspect. This aspect crosscuts the task code and adds into this task the profiling functions (hook functions named `chrono`) in order to measure at runtime the execution times of its IComp.



```
void Task_T_PID(int arg)
{
     static int boucle = N_BOUCLE_T_PID;

     fifo_IC("Input_speed_IComp", 0, 27);
     init2(0);
     fifo_IC("motor_cmd_IComp", 1, 27);
     init7(0);
     fifo_IC("PID", 2, 27);
     init12(0);

     while(boucle--)
     {
          chrono('d', 0, &depart_taches);           profiling
          inTheLoop_Input_speed_IComp(0);           functional
          chrono('f', 0, &depart_taches);           profiling
          chrono('d', 1, &depart_taches);           profiling
          inTheLoop_motor_cmd_IComp(0);             functional
          chrono('f', 1, &depart_taches);           profiling
          chrono('d', 2, &depart_taches);           profiling
          inTheLoop_PID(0);                         functional
          chrono('f', 2, &depart_taches);           profiling
                                                                  scheduling
          rt_task_resume_end_times(-nano2count(PERIODE_T_PID),
                                   -nano2count(DEADLINE_T_PID));
     }

     end4(0);                                        functional
     end9(0);
     end14(0);
}
```

Figure 9. PID generated code with profiling aspect

The measured times are then sending back to the simulation/analysis framework of MoDEST. Based on this collected information, a real scheduling chronogram is drawn to compare it with the theoretical scheduling chronogram. Timing measurement could be done on this result in order to get some relevant temporal information to send back to control design tools.

4 CONCLUSION AND FUTURE WORKS

This research work is undertaken by collaborations between control designers and computer sciences designers to clarify and distinguish the models handled in control performance design and those manipulated in real-time implementation. Current results are a first step towards a harmonization of these heterogeneous models. The component approach should add significant value to the design process, helping the software designer to produce modular and reusable system model. The aspect approach represents a significant paradigm shift from the traditional monolithic view. It makes the system model easy to extend/contract system capabilities with global wide changes being performed automatically, avoiding errors of forgetting to change one or more locations. This leads to make design easier, improves accuracy and reduces design time. The tool provides a way to speed up design-code-test-debug cycle through analysis/simulation of the system model, and automated transformations and code generation. These features can substantially improve the development, implementation and evaluation of embedded system software.

This research work is in progress to be fully implemented in the MoDEST tool. In the near future, we will define and implement transformation rules to improve the automatic transformation between the facets. These rules will be based on the MDA approach (Model-Driven Architecture) approach and its associated tools. Software bridges to external tools are envisaged in order to allow models validation in the earliest steps of the design process.

References

[1] M. Ben Gaid, R. Kocik, Y. Sorel, and R. Hamouche. A methodology for improving software design lifecycle in embedded control systems. In *IEEE Design, Automation and Test in Europe, DATE'08*, Munich, Germany, 10-14 March 2008.

[2] J. Bézivin and O. Gerbé. Towards a precise definition of the OMG/MDA framework. In *Proceedings of the Conference on Autonomous Software Engineering (ASE01)*, San Diego, CA, USA, 2001.

[3] A. Cervin and B. Lincoln. Jitterbug 1.1 reference manual. technical report ISRN LUTFD2TFRT-7604-SE. Departmentof AutomaticControl, Lund Instituteof Technology, Sweden, 2003.

[4] D. De Niz and P. H. Feiler. Aspects in the industry standard aadl. aspect-oriented software development. In *Proceedings of the 10th international workshop on Aspect-oriented modelling. Vol. 209, Vancouver, Canada.*, 2007.

[5] R. Hamouche and R. Kocik. Metamodel-based methodology for real-time embedded control system design. In *Forum on specification and Design Languages FDL'6*, Darmstadt, Germany, September 2006.

[6] R. Hamouche, R. Kocik, and M. E. Ben Gaid. Multi-facet design methodology for real-time embedded control systems. In *IFAC Workshop on Programmable Devices and Embedded Systems*, pages 14–20, Feb 2006.

[7] D. Henriksson and A. Cervin. Truetime 1.1 reference manual. technical report ISRN LUTFD2TFRT-7605-SE. Departmentof AutomaticControl, Lund Instituteof Technology, Sweden, 2003.

[8] D. Henriksson, O. Redell, J. El-Khoury, M.Trngren, and K.-E. Arzn. Tools for real-time control systems co-design a survey. Technical report, Department of Automatic Control, Lund Institute of Technology, Sweden., 2005.

[9] R. Kocik, M. Ben Gaid, and R. Hamouche. Software implementation simulation to improve control laws design. In *Proceedings of the European Congress SENSACT 2005*, Paris, France, 2005.

[10] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine. The syndex software environment for real-time distributed systems design and implementation. In *Proceedings of the European Control Conference, Grenoble, France*, 1991.

[11] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1999.

[12] Xerox. AspectJ site : http://www.aspectj.org, 2003.