Component-Based and Aspect-Oriented Methodology and Tool for Real-Time Embedded Control Systems Design

Rédha Hamouche
Université Paris-Est, ESIEE Paris,
Embedded Systems Department,
2, Bd Blaise Pascal BP 99
93162 Noisy-Le-Grand Cedex France
Email: hamouchr@esiee.fr

Rémy Kocik
Université Paris-Est, ESIEE Paris,
Embedded Systems Department,
2, Bd Blaise Pascal BP 99
93162 Noisy-Le-Grand Cedex France
Email: kocikr@esiee.fr

***Abstract*—This paper presents component-based and aspect-oriented methodology and tool for designing and developing Real-Time Embedded Control Systems (RTECS). This methodology defines a component model for describing modular and reusable software to cope with the increasing complexity of embedded systems. It proposes an aspect-oriented approach to address explicitly the extra-functional concerns of RTECS, to describe separately transversal real time and security constraints, and to support model properties analysis. The benefits of this methodology are shown via an example of Legway control software, a version of the Segway vehicle built with Lego Mindstorms NXT.**

***Keywords*—Model-based design, software component, aspect-oriented programming, embedded system design, embedded control software.**

I. Introduction

Designing real-time embedded control systems tends to be more and more difficult due to the strong constraints and the growing complexity of the used software and hardware components. Theses systems are context-dependent: they depend on their environment in which the system is operated, i.e. they react to the context changes (sensors, operators, ...), and their behaviors are constrained by the context [1]. Real-time, embedded, security, energy or physical constraints are an example of context-dependent properties.

A typical V-process for RTECS design can be decomposed into three major steps [2]: functional control modeling and analysis, specification of control software and real-time implementation of control software. Along the design process, we have to overcome the following difficulties: (1) 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. 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 backtracking (reiterations of V-cycle) in the design process, which lengthens the design life cycle 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 [3][4]; (2) Major RTECS properties, 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; (3) As we have to treat various kinds of contexts, it is difficult to model them from one point of view; (4) 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 makes reusability, modifiability and extensibility of RTECS not efficient.

978-3-9810801-8-6/DATE12/ ©EDAA

Nowadays, there are research challenges to define methodologies and tools which reduce the design time and cost, ensure the consistency of models from system level to implementation level, and improve modularity, reusability and maintainability of system models and/or codes. In this context, we have defined in [5][2] a methodology and a tool for designing RTECS. Section II describes the overview of this methodology. In the section III, we present the use of component and aspect paradigms for RTECS design.

II. OVERVIEW

The main goal of the methodology is to evolve the RTECS development process from a classical code-oriented development to a model-driven development (MDD) [6] where the code is generated automatically. A MDD offers a better level

of abstraction and enables handling the complexity and the heterogeneity of models. The automatic transformation of models improves their consistency and traceability throughout the development cycle. The proposed methodology offers a multi-facet modeling where the system model is viewed on different design facets. 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). To strengthen the construction of reusable modules, the methodology is based on component design approach [7], and to take into account the context dependency of RTECS, and offer a better modularity and ability to model and analysis extra-functional properties, the methodology is based on the aspect-oriented approach [8]. In the literature, the aspect paradigm is defined at the programming language level (AspectJ [9] for example). 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, cache aspect, communication, 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. In this paper, we focus only on component and aspect describing the real-time implementation facet.

In order to illustrate the proposed methodology, we consider the software design of a Legway control system (see Figure 1). A Legway is a version of the Segway (TM) [1] built with Lego Mindstorms NXT. The objective of the RTECS is the stabilization of the system on the vertical axis. When it detects a tilt angle, it acts on the wheels (motor voltage) in the direction of the change. The context of this RTECS is the user, the platform NXT, the servo-motors for controlling the wheels, and the sensor for measuring the degree of inclination. The system requirements impose a real time behavior and two major constraints: for security reasons, the system does not diverge and it should behave according to energy resources.

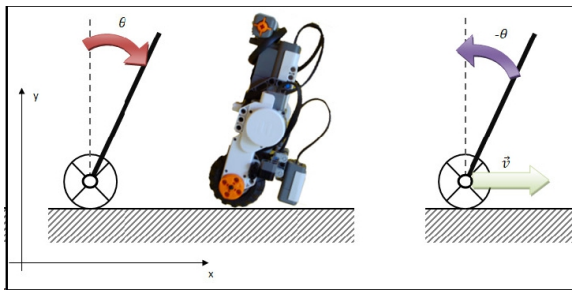

Fig. 1. Example: Legway control system

[1]Segway (TM) is a trademark of Segway vehicle: with only two wheels as points of support, he can keep his balance, forward, backward, turn under the sway of the user

III. Methodology

A. Component description

As shown in figure 2, a new component model addressing explicitly the real-time properties of embedded systems is defined. It interacts with the environment through its interface and it is characterized by reflective information and internal operations. The component behavior is reactive to the environment with its internal real-time automata (statechart). This behavior describes only the functional part of the model. For instance, the functional part of the Legway control system includes a component which behaves like a periodic task to control the Legway. This component defines the data `power` and `angle` as reflective informations, the operations `pid()` (proportional-integral-derivative algorithm) as the provided internal operation, the operations `getAngle()` and `setCmd()` as required external operations. The component statechart (see Figure 3) calls periodically the operations `getAngle()` et `pid()` for computing the servo-control command, printing some information on the system screen, and sending via bluetooth to server the power and angle informations.

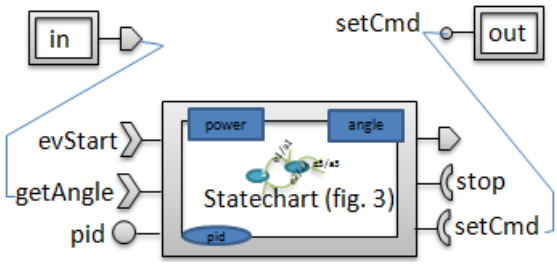

Fig. 2. The component model - Legway control component

B. Aspect description

An aspect describes the context-dependent part and the transverse properties of control systems. It may crosscut system components or set of the components for affecting the system behavior or performance. An aspect is viewed as a super-component which adds an extra-functional treatment and/or imposes non-functional constraints via interfaces called aspectual interfaces. 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 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

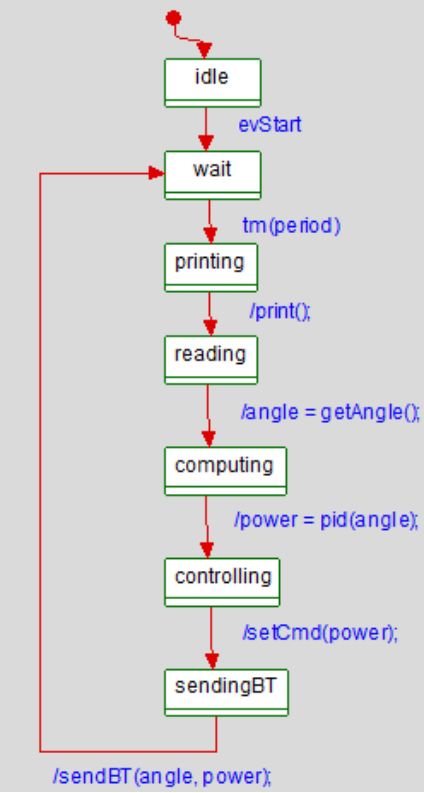

Fig. 3. The functional part of the RTECS - Legway statechart

mode (synchronous or asynchronous) or a rate of data production/consumption; (3) to control data or operation access via an interface called *introspection interface*. An aspect can control operation calls or access to data component 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 the corresponding aspect by another one specific to a new execution platform.

An example of an aspect in our case study is the cache aspect which enables to optimize access to the angle sensor. It stores the angle value in the cache. The aspect behavior (see Figure 4) allows to avoid two successive readings of the sensor in less than the "freshness timeout" delay. The aspect imposes this freshness constraint via the activation interface ICACHED and the introspection interface IANGLE. The interface ICACHED provides a guard which is true if the data is cached while the interface IANGLE allows to control data assignment (angle assignment).

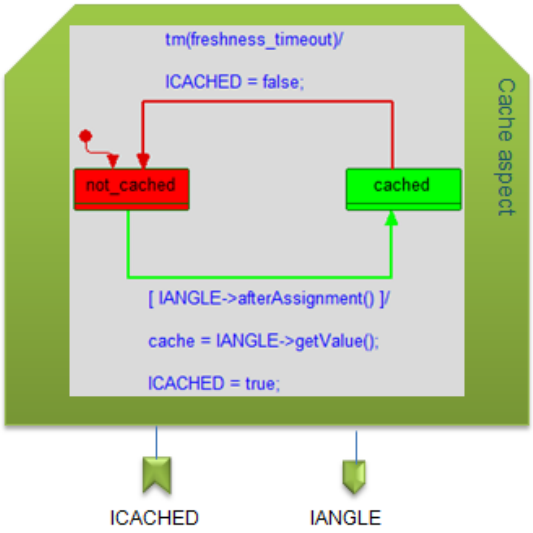

Fig. 4. Example : cache aspect

C. Model weaving

The components are interconnected via their functional interfaces and they can be weaved with aspects via aspectual interfaces. The weaving is a semi-automatic graphical step, and it is user-guided. Depending on the interface type, we distinguish four weaving forms : (1) introduction of new transition based on the guard of an activation interface; (2) introduction of a new super-state and transition based on the event of the trigger interface; (3) automatic interception of data access based on the introspection interface; (4) initialization and configuration of the functional component information via resource interfaces.

In our case study, the designer binds the IANGLE interface with the angle data of the functional component. So, after the angle assignment, the aspect capture and store the sensor value. The designer also introduces graphically in the task behavior a new transition based on the ICACHED guard (see dash-dot-dot transition in Figure 5). The angle is read from the cache until the freshness constraint is expired. This behavior optimization leads then to reduce the execution time of the task. Note that if the cache aspect is removed, the introduced transitions and the associated interfaces are automatically removed from the model.

The Legway system is extended or crosscutted by three other non-functional aspects: energy, security and platform aspects. The security aspect, shown in Figure 6, supervises as a watchdog the control task for adjusting its behaviour to the physical constraints. It introspects, via the interface `IPower` the data `power` for checking the command thresholds and supervising the system stability. If the security aspect is in an `unstable` state, it broadcasts via the interface IRESET the event `evReset` to stop the system. Thus, in the model weaving, we have introduced a super-state and a transition sensitive to the event `evReset` (see Dashed super-state and transition in Figure 5).

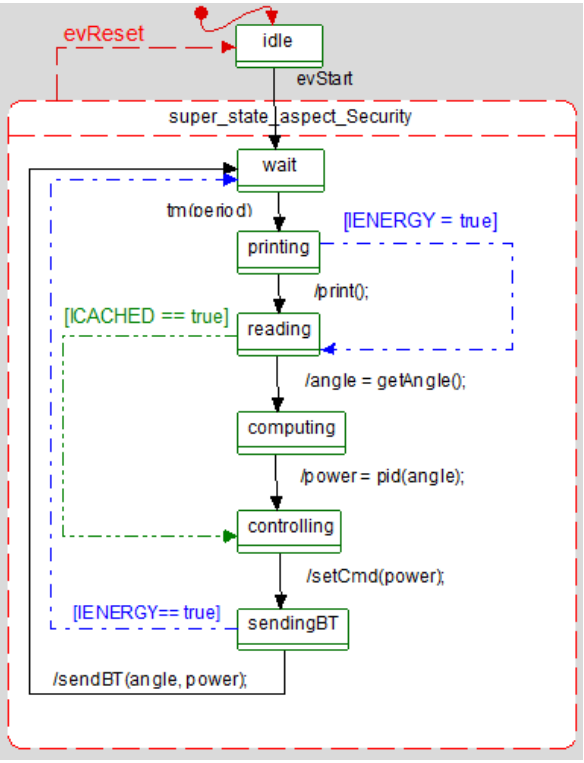

Fig. 5. Example : Model weaving (cache, security and energy aspects)

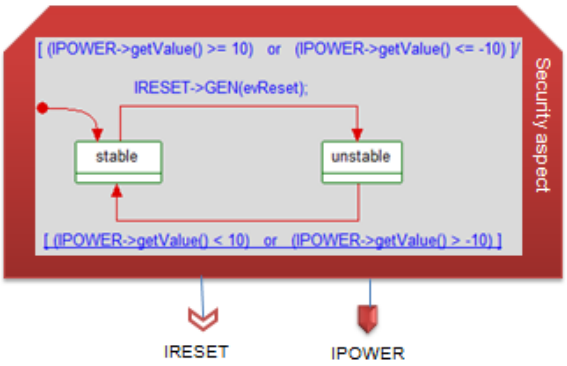

Fig. 6. Example : Legway security aspect

The energy aspect manages the battery status for constraining the control task and making it sensitive to the energy resource. As shown in figure 7, this aspect defines an activation interface IENERGY which sets its guard to true if the battery level is below 20 %. At the model weaving, we introduce two new transitions with higher priority (Dash-dot transition in Figure 5). This aspect enables then to bypass displaying operations and Bluetooth communication in order to conserve battery power.

The platform aspect specifies the NXT runtime context (the target processor, OS and implementation language). It specifies the time constraints (period, deadline) and the WCET of the task operations. The platform aspect defines timing properties of the operations (`getAngle, PID and setCmd`) within

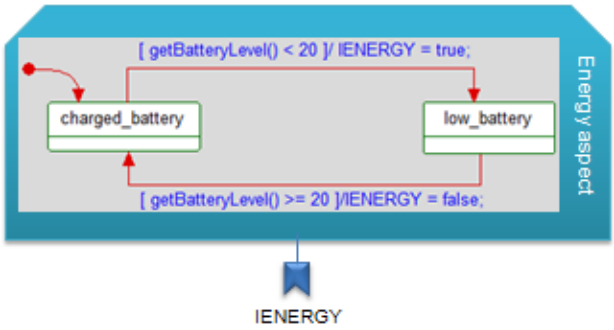

Fig. 7. Example : Legway energy aspect

the platform Lego_NXT.

IV. Conclusion and Future Works

The component approach should add significant value to the design process, helping designers 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.

This research work is in progress to be fully implemented in a computer-aided design tool. The tool will provide a way to speed up design-code-test-debug cycle through analysis/simulation of the system model, and automated transformations and code generation. This work is also in progress to take into account the interaction of aspects with similar functional targets. This interaction can be handled with a priority mechanism between aspects and between transitions.

References

[1] T. Kishi and N. Noda, "Aspect-oriented context modeling for embedded systems," presented at Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, held in conjunction with AOSD Conference, 2004.

[2] 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.

[3] 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.

[4] 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.

[5] 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*, Feb 2006, pp. 14–20.

[6] 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.

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

[8] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, 1997.

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