A Methodology for Improving Software Design Lifecycle in Embedded Control Systems

Mohamed El Mongi Ben Gaïd, Rémy Kocik, Yves Sorel, Rédha Hamouche

A methodology for improving software design lifecycle in embedded control systems

Mohamed El Mongi Ben Gaid*, Rémy Kocik†, Yves Sorel‡, Rédha Hamouche†

* NeCS Project-Team, INRIA Grenoble-Rhône-Alpes, France (email: Mohamed.Bengaid@inria.fr)
† COSI Lab, ESIEE Paris, France (emails: {r.kocik, r.hamouche}@esiee.fr)
‡ AOSTE Project-Team, INRIA Paris-Rocquencourt (email: Yves.Sorel@inria.fr)

Abstract

Control design and real-time implementation are usually performed in isolation. The effects of the computer implementation on control system performance are still evaluated on the last phases of the development cycle. It is expected that modeling the computer implementation in order to simulate its impact on control would help reducing the length and the effort of the development cycle. This paper proposes ideas towards achieving these objectives. To this end, implementation effect on control performance is first studied. Then, we describe the preliminary ideas of a methodology considering a control law designed with the Scicos simulation environment and implemented on a distributed architecture with the SynDEx system-level CAD tool. This methodology allows simulating the impact of the distributed implementation early in the design lifecycle and provides an automatic code generation of this implementation.

1. Introduction

The design of an embedded system aims at building a computing system, which is able to control the evolution of a physical system: the plant. The plant is made up of interconnected mechanical, electrical and/or chemical elements. At design time, an increasingly important time is dedicated to the writing of the software that will be executed on the computer architecture.

In the first phase of *modeling*, the control engineers describe the interactions between the components constituting the plant using mathematical equations. The built *plant* model makes it possible to predict the evolution of the process when its components are subjected to various external physical phenomena (or *actions*). Using a simplified version of this model, and starting from the desired behavior, the control engineers carry out, during the second phase, the *synthesis* of the *control laws*. The control laws are the mathematical model representing the actions that have to be applied to the plant, in order to achieve the desired behavior. They are next discretized in order to allow their digital execution. In the third step, computer science specialists implement the algorithms that are defined by the control laws, ensuring the real-time operation of the computer implementation. This third phase constitutes the software *realization* phase. Usually control engineers use CAD tools such as MATLAB/Simulink or Scilab/Scicos to model the continuous plant and make the discrete-time control law synthesis. These tools allow also the validation of the system control performance by performing a *hybrid simulation* (continuous and discrete time). However, when designing the control laws (*modeling*, *synthesis* and *simulation*), control engineers make, in most cases, the assumption that the time between the sampling of the inputs and the actuation of the outputs is zero and that inputs sampling is strictly periodic without variations of periods. The software, which is developed by computer science engineers (*realization*), cannot respect these assumptions because computations take time, and the different components of the computing architecture are shared between different activities. Moreover, in distributed architectures, these time delays can be increased due to the communications. Under these conditions, the desired performances of the system may not be achieved, and it is necessary to amend the control laws (in a calibration phase) in order to compensate for the influence of these communication and execution times.

In this paper, we propose a design methodology enabling to avoid the lengthening of the development cycle by the different iterations between the design of control laws and their implementation, which may be needed to correct the influence of the implementation on control performance. In this methodology, we propose to model the implementation in order to be able to study using simulation, the influence of the communication and computing times on the system performance, at an early stage of system development. This methodology, which was defined in the ECLIPSE RNTL

project[1] , relies on the link that was established between the tool Scicos (used for the control design) and the SynDEx[2] distributed implementation CAD software. In Scicos, the discrete-time control laws and the continuous plant model are described using a data-flow graph. After extracting these control laws from the data flow graph, an automatic translator allows their representation as a SynDEx model [3, 5]. SynDEx is a CAD software based on the AAA methodology (Algorithm Architecture Adequation). It allows the user to specify application algorithms and distributed architectures, enabling performing their best matching by exploring manually and/or automatically the possible implementations while satisfying the real-time constraints. The automatic exploration is performed by optimization heuristics, also called "adequation". Due to the automatic translation of the Scicos models into SynDEx models, the SynDEx specification of the control algorithm is thus in conformity with the Scicos model of the control law. For that reason, it is no longer necessary to specify or to code once again the control software. The specification of the distributed architecture is the only remaining task. SynDEx allows the designer to perform this specification, and subsequently makes it possible to accomplish an efficient matching between the algorithm and the architecture, to define the scheduling of the different parts of the algorithm onto the hardware components (processors, ASIC, FPGA) taking into account the required communications, and finally, to automatically generate the corresponding code. This code satisfies the real-time constraints on one hand. On the other hand, it is deadlock free in the case of distributed platforms. In our methodology, we use the result of SynDEx "adequation" to model the communications and computing times induced by the implementation and introduce them in the original Scicos data-flow graph for an accurate simulation. By this way, this methodology allows covering and reducing the entire software development cycle.

The paper is organized as follows: Section 2 defines the different factors that may degrade system performance. Based on the Scicos formalism, we propose in Section 3 a model describing the implementations generated by SynDEx, and show how this simple model allows capturing all the previously mentioned performance degradation factors. Finally, Section 4 concludes and proposes future work.

2 Factors characterizing the impact of the implementation on control performance

In this work, we assume that the control law is a Scicos data-flow graph having p inputs (measures), m outputs (controls) and whose sampling period is T_s. Usually, in simulation tools like Simulink and Scicos, the digital controllers are simulated using the hypothesis of the stroboscopic model, which assumes that sampling and actuation are performed at the same time at the beginning of each sampling period. However, this contrasts with the real behavior of computing architectures [4]. It has been shown that the impact of the implementation on the control performance may be reduced to the temporal analysis of input and output operations [2], which may be described using the notions of sampling and actuation latencies. The sampling latency at the kth sampling period of the jth controller input is defined by

$$L_j^s(k) = I_j(k) - kT_s, \qquad (1)$$

where $I_j(k)$ is the instant where the sampling of the jth input of the controller is performed. $L_j^s(k)$ represents the delay resulting from the acquisition of the jth measure with respect to the beginning of the current sampling period. In a similar way, the actuation latency at the kth sampling period of the jth controller output is defined by

$$L_j^a(k) = O_j(k) - kT_s, \qquad (2)$$

where $O_j(k)$ represents the instant where the actuation of the jth controller output is performed. $L_j^a(k)$ represents the delay that takes to update the jth control with respect to the beginning of the current sampling period. Fig. 1 illustrates these different notions.
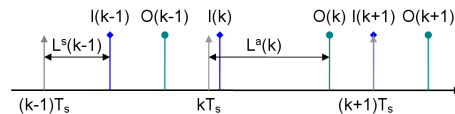
Figure 1. Implementation effect on the timing of input/output operations

3 Scicos model of the temporal behavior of SynDEx implementations

3.1 Scicos

Scicos (Scilab Connected Object Simulator) [1] is an extension of Scilab, the numerical computational software[3]. Scicos enables the creation of block diagrams in order to model and simulate hybrid dynamical systems. The main design feature in Scicos is that it was intended to be a simulation environment in which both continuous systems and discrete systems co-exist, in opposite to other hybrid simulation packages that has been constructed by extension of a continuous simulator or of a discrete simulator. Its basic

---

[1]This RNTL project was supported by the French Ministry of Industry
[2]It is distributed free of charge at www.syndex.org

[3]developed by INRIA and ENPC and distributed free of charge

blocks reflect this feature. In fact, in addition to their regular inputs and outputs, discrete blocks in Scicos have *event inputs* and may also contain event outputs. For example, consider the simple control system simulation model in Fig. 2. In this model, the controller has one regular input, one regular output and one event input (the red port located on the top of the block). The first Sampling/Hold (`S/H`) block models the sampling of plant output, whereas the second models control input actuation. Note that in this idealized simulation model (following the stroboscopic model), sampling and actuation occur at the same time (whereas in a real implementation they may occur at different moments). The activation events of the controller block, as well as those of `S/H` blocks, are received from an activation clock block, which sends activation events periodically (following the stroboscopic model). Besides the activation clock, Scicos provides also blocks for the processing of events. The existence of these activation inputs considerably simplifies the development of the proposed methodology. In fact, thanks to this feature, there is no need to modify the design that was performed by control engineers assuming an idealized implementation. It suffices to replace "the clock generator" by properly produced events matching the temporal behavior of the implementation generated by SynDEx. All these events are generated by a temporal model of the SynDEx schedule. The translation of this temporal model to Scicos is addressed in the following sub-section.
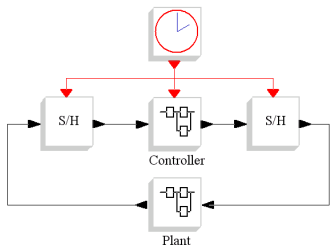
ated code includes synchronization mechanisms guaranteeing this execution order. Thus, SynDEx generates on each processor a sequence of computation operations all together synchronized by communication sequences on the communication media. This code, obtained through the "adequation", is built taking into account the worst case execution times (WCET) of the computation and interprocessor communication operations. The modeling of SynDEx implementations in Scicos requires extracting, from the generated schedule, the starting and completion time instants of the computation/communication operations. The determination of these instants enables the knowledge of the completion instants of sampling and actuation operations, which in turn determines the impact of the implementation on the performances of the control law. In the next step, this timing information is used to create a *graph of delays* in Scicos (Fig. 3). The graph of delays is a Scicos graph, built based on Scicos event processing blocks, and whose function is to send activation events to the different Scicos `S/H` blocks constituting the interconnection between the controller and the plant, as well as the controller. The graph of delays may also need some information about selected controller variables. By this way, the Scicos simulation is enabled to exhibit the impact of the SynDEx implementation on the performances of the control law. A SynDEx schedule is mainly composed of three basic constructions: sequencing, conditioning and synchronization.
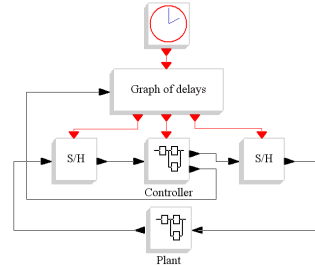

Figure 2. Plant and controller interconnection

Figure 3. Plant, controller and graph of delays interconnection

3.2 General principles for the translation of the temporal behavior of SynDEx schedules to Scicos

The optimization heuristic of SynDEx computes a distribution of the different operations of the algorithm (corresponding to the different blocks of the Scicos model) as well as the scheduling of these operations on the different components of the distributed architecture. This off-line non-preemptive schedule defines a total order on the operations described by the algorithm for each hardware component, while trying as well as possible to exploit the parallelism of the architecture. The automatically gener-

3.2.1 Sequencing

In Scicos, an event may be seen as the activation signal of a bloc. When this signal is available, the bloc begins its execution. When the execution of the block is finished, an event is generated in order to indicate to the successor blocks (which possibly consume the data that it produces) that it finished its execution. The modeling of the sequencing in SynDEx schedules follows this reasoning. The sequencing of three operations on a processor in a SynDEx scheduling as well as its translation to Scicos is illustrated in Fig. 4. The first Scicos `Event Delay` block (modeling

the execution duration of the SynDEx operation `F1`) begins its execution when it receives the event of activation. After `LF1` units of time, it finishes its execution and generates at exit an event to indicate to its successor `Event Delay` block to start its execution, which in turn begins its execution and generates an event after `LF2` time units...
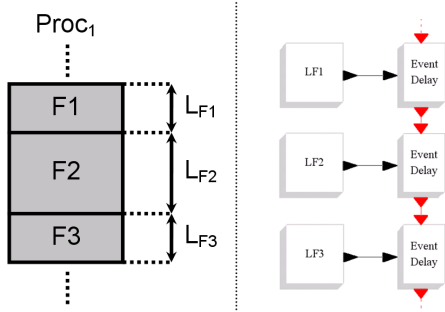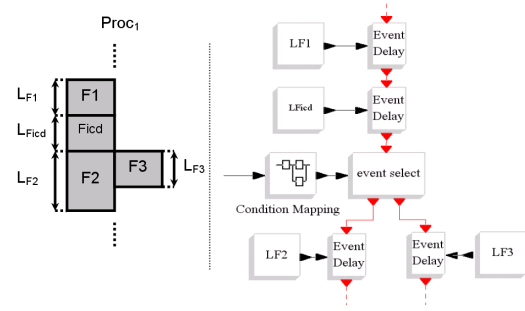

Figure 4. SynDEx schedule with sequencing (left) and its translation in Scicos (right)

3.2.2 Conditioning

A SynDEx algorithm graph may contain conditioning (equivalent to *if..then..else* in C language), which is translated in the generated scheduling by conditional branching operations. It is important to take these conditional branching operations into account in the temporal model of the SynDEx schedule, because the execution of each branch usually does not take the same execution time. This induces a temporal jitter on the input/output operations which may degrade the control performance. The modeling of conditioning in Scicos consists in appropriately switching the incoming events towards the Scicos blocks to be executed. The switching of the events may be performed using the block `Event Select`. It is necessary to create a function called "Condition Mapping", which receives in input the variable determining the choice of the Scicos block to be executed, and provides in output the identifier of the output channel of the block `Event Select` towards which the event should be forwarded. The translation of a schedule containing conditioning is illustrated in Fig. 5.

3.2.3 Synchronization

Synchronization is a fundamental operation to describe the behavior of the implementations automatically generated by SynDEx, because it allows guaranteeing the total order specified by SynDEx scheduling. The use of synchronization is necessary to describe the sending and the reception of the messages. In order to describe in a simple way the synchronization in automatically generated SynDEx schedules,

Figure 5. SynDEx schedule with conditionning (left) and its translation in Scicos (right)

it is necessary to introduce a new block in Scicos, called "Synchronization". The synchronization block has N event inputs and 1 event output. The parameter N must be customizable by the user. The block must be executed at the reception of an activation event. It generates an event in output and resets (to zero) all its internal variables (memorizing the received events) when each of its event inputs have received at least one event since the last reset.

4 Conclusion

This paper presented ideas allowing linking the semantics of Scicos and SynDEx. The main factors characterizing computer implementation influence on control performance were first reviewed. Based on existing and proposed Scicos blocks, the general principles for modeling SynDEx implementations in Scicos were then presented. In a future extension of the ECLIPSE project, this methodology will be fully implemented and evaluated on a case study.

References

[1] S. Campbell, J. Chancelier, and R. Nikoukhah. *Modeling And Simulation in Scilab/Scicos*. Springer, November 2005.

[2] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

[3] T. Grandpierre and Y. Sorel. From algorithm and architecture specification to automatic generation of distributed real-time executives: a seamless flow of graphs transformations. In *Proc. 1st ACM and IEEE Int. Conf. on Formal Methods and Models for Codesign*, Mont Saint-Michel, France, Jun. 2003.

[4] R. Kocik, M.-M. Ben Gaid, and R. Hamouche. Software implementation simulation to improve control laws design. In *Proc. of Sensors & Actuators for Advanced Automotive Applications*, Paris, France, Dec. 2005.

[5] Y. Sorel. Syndex: System-level cad software for optimizing distributed real-time embedded systems. *Journal ERCIM News*, 59:68–69, Oct. 2004.