HiFAQ's: a Language for Integrated System Level Simulation

Volker Zerbe, Ulrich Freund, Gunar Schorcht
Technical University of Ilmenau
Department of Computer Science and Automatic Control
System- and Control Theory Group
(freund,schorcht,zerbe)@theoinf.tu-ilmenau.de

Abstract

Since embedded systems are the heart of mechatronic systems, the environment of the embedded system, i.e., the rest of the mechatronic system, has to be taken into account at the early stages of the design. To deal with the complexity of the overall system, we use system abstractions methods well known in systems theory. By means of Integrated System Level Simulation techniques we close the gap between system theory and embedded systems design. As a side effect, architectural trade-offs between HW and SW can be performed.

1 Introduction

Currently, designers of embedded systems are still forced to determine the architecture at an early stage in the design cycle. Examples are Hardware/Software partitioning, type and number of microprocessors and microcontrollers, scheduling strategies, and communication protocols. A full redesign is necessary if the chosen architecture is not capable of performing the function within the performance requirements. On the other hand, it might happen that the resources of the architecture are not used completely which might result in a too expensive product.

These limitations are well recognised in the aerospace [7] and automotive industries [1, 9, 3, 4, 2] and could be overcome by means of the model based design approach. However, the coupling of two or three tools is still necessary to assess the effects of an architecture to the system dynamics. To overcome this situation we are introducing the HiFAQ modelling and simulation language.

2 Model Based System Design

Since an embedded system has to work in an environment, this imposes not only constraints on the design, but the interaction with it too. If the requirements are not met well, the interaction could even decide over the success of the system on the market.

To know at an early stage of the design in which environment the system has to work, one builds models of the environment. This is a nontrivial task to be performed by the domain experts, e.g. the electrical or the mechanical engineer. In our approach, their modelling tasks consist of providing abstract models of their systems in form of ordinary differential equations. We take these models of all components of the system for which the embedded system is designed, merge this with the requirements of the embedded system and build a functional model of it. We then augment this model with models of the architecture it has to run on. Architectural models for the early design phase are transaction/resource models, which can assess the performance properties of the system in the early design phase. These three different layers are shown in figure 1

The functional model can be mapped onto several architectural models. Architectural models represent alternative HW/SW parts. Using this approach, the designer gets an immediate feedback by means of an integrated timed simulation run to which extend the architecture will fit the function. Unsuitable architectures are ruled out instantly. This is depicted in Fig. 2

3 The HiFAQ Approach

The above approach requires a tool capable of modelling and simulating heterogeneous, em-

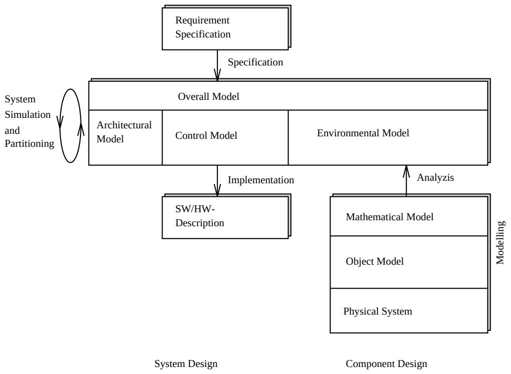

Figure 1: The Modelling Methodology

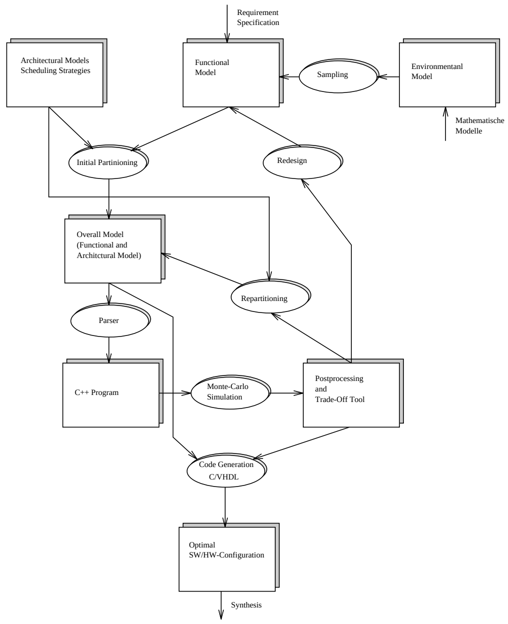

Figure 2: SW/HW Partitioning by means of System Level Simulation

bedded systems at system level. In this paper we present a modelling and simulation language for the design of those systems. The main elements of this language are the HiFAQ's (Hierarchical Functional Augmented Queuing Networks). HiFAQ's consist of three layers: the transformatorial, the reactive and the architectural layer. Elements of the HiFAQ's are functional and architectural model elements. Each element is associated to one of the three layers. After the interfaces between the layers are defined, the layers can be modelled seperately. Depending on the point of view, one gets a transformatorial functional view, a reactive functional view or an architectural/resource oriented view. Through mapping of the functional models onto the architectural models the effects of architecture modifications are unveiled via simulation.

Within a HiFAQ the reactive elements are represented by extended finite state machines, and the transformatorial layer's elements are described via difference/differential equations. The architecture is modelled by transaction/resource models. These elements themselves allow the classcial stepwise refinement approach for each layer. A HiFAQ Model defines the following couplings:

- Intralayer I/O coupling: This form is used to connect the elements within in a layer.
- State/Parameter coupling: This variante is used for the interlayer coupling. With this method, the interface between the layers and hence the different development parts of an embedded systems are defined. On the other hand, these couplings are responsible for the influences of the different layers.
- InterHiFAQ I/O coupling: HiFAQ's are coupled themselves by input/output connections. However, these couplings are supplying a closure for the different layer's coupling. Remembering that the elements within a layer are also coupled via input/output connections, the meaning of a closure becomes clear. Also the classical divide and conquer approach is preserved.

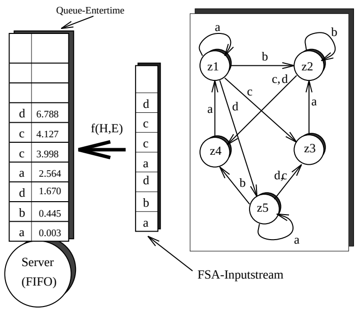

Figure 3: The different levels of a behavioural- and performance-model

Hardware parts are modelled by transaction/resource models dedicated to a functional model exclusively. A similar approach is used for software parts. Every transition of a software process is assigned to a transaction. The processor is modelled as a resource. The processor time required by a transaction is determined by the transition times of the finite state machine representing the software process. However, the process transactions are waiting in a queue for the processor, i.e. sharing the resource. This situation is depicted in Fig. 3. A systemtheoretic based desciption of HiFAQ's can be found in [5].

To some extend, the HiFAQ-Scheme resembles to the common development process. The application software is developed seperately from the system software which is also developed seperately from the Hardware. The main difference is that in the former approach the system had to be developed to the end. The results could be assessed in the end. By means of the HiFAQ scheme the classical divide and conquer approach is preserved, but the results are delivered in the early phase of the design cycle. Hence, the limitations of the current design approach are overcome.

4 The simulation class library

We developed a compiler for the HiFAQ language which translates HiFAQ models into C++ code. The C++ code uses intensively a class library capable of simulating continuous, time driven and event driven models simultaneously.

This is achieved by grouping the objects of the resulting simulation program into two categories. The first category consists of the active elements whereas the second one contains the passive elements: Active objects are responsible for the time management. Simulation time is forwarded by events and integration algorithms. Hence, active objects are events and integration algorithms. Active objects reacting to events are called discrete active objects whereas the objects doing integrations are named continuous objects The time between two consequetive events is submitted to the active continuous objects. While doing integration between the events, the time of the next event is monitored so that the system can react on state events.

Passive objects hold the functional description of the components as well as the inherent functions of the resources. These are finite state machines, differential equations, and queuing strategies. These functions are triggered by active objects, i.e., receiving a message from them. Continuous active objects are maintained in a queue with its associated step time.

5 Example

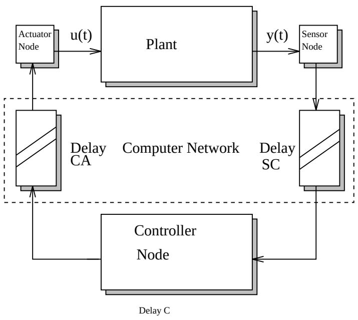

Figure 4: model of a control loop coupled by a communication network

The modelling and simulation of a simple 'Fly-by-Wire' control system by means of the HiFAQ description is shown. This system consists of a control loop implemented via a Token-Bus system. Such systems are common in automotive electronics [8]

A servo generates a signal R. Samples of this signal are transmitted to the regulator. The regulator also receives measurement values Y from the plant. Both values are compared and an error signal E is computed. The regulator calculates an actuator value U according to some control law. This value is transmitted to the plant. The plant sends a measurement value back to the regulator.

Assuming that the controller, the actuator, and the sensor are located at different locations connected via a bus system, and also assuming that the bus is also used by other stations, one comes up with the model of figure 4. The setpoint is generated in the controller. From the control theory point of view it is a real-time system with random time delays [6]. The delays *sc,ca* are simulated by the architecture of the Token-Bus model.

The plant(x_p) and the regulator(x_r) are described by second order ordinary differential equations:

Fig. 5 shows how an awkward implemention of the controller can influence the stability of the control loop.

$$\dot{x_p} = \begin{bmatrix} 0 & 1 \\ 0 & -3.3081 \end{bmatrix} x_p + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u$$

$$y_p = \begin{bmatrix} 0 & 1 \end{bmatrix} x_p$$

$$\dot{x_r} = \begin{bmatrix} 0 & -4 \\ 1 & 0 \end{bmatrix} x_r + \begin{bmatrix} -0.7473 \\ -9.88 \end{bmatrix} e$$

$$y_p = \begin{bmatrix} 0 & 1 \end{bmatrix} x_r - 0.7473e$$

With a well fitted architecture one can achieve a better performance of the overall system. This case is shown in the next figure 6 where the control and measurement values are used as early as possible for controlling the plant.

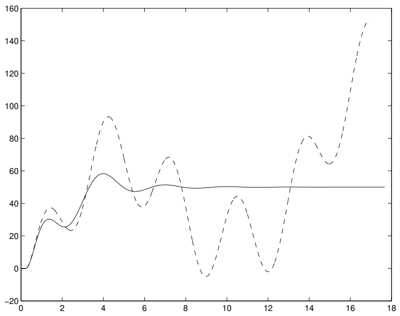

Figure 5: step response for the original (solid line) and a time driven implementation (dashed line)

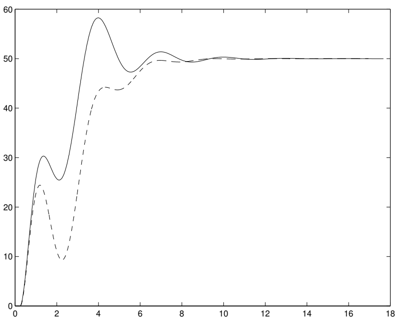

Figure 6: step response for the original (solid line) and a well fitted event driven implementation (dashed line)

6 Conclusions

In this paper we presented a modelling and simulation language for the integrated timed simulation of heterogeneous embedded systems. On a simple fly-by-wire control system the effects of architectural changes are unveiled.

Further investigations will be a database of computation time of several standard elements, the development of a framework HiFAQ's as well as the the development of control algorithms capable of dealing with variable time delay.

References

[1] J. Bortolazzi, *Untersuchungen zur rechnergestuetzten Erfassung, Verwaltung und Pruefung von Anforderungsspezifikationen und Einsatzbedingungen elektronischer Steuerungs- und Regelungssysteme*, Dissertation, Universitaet Erlangen-Nuernberg, April 1994

[2] C. Passerone et al. *Evaluation of tradeoffs in the design of embedded systems via co-simulation* Technical Report, University of California at Berkeley, Berkeley 1996

[3] FZI,*Abschlussdokumentation des Projekts MSR* FZI, Karlsruhe 1996

[4] J. Ernst *Studie zu Spezifikation und Aufbau einer Cosimulationsumgebung unter der Beruecksichtigung des Simulator-Backplane Standards der CAD Framework Initiative*, FZI, Karlsruhe 1996

[5] U. Freund, *Hierarchical Simulation Techniques for the Design of Heterogeneous, Embedded Systems*, IFIP Workshop on Modelling of Microsystems, Stirling, 1997, or http://www-sst.tu-ilmenau.de

[6] J. Nilsson, B. Bernhardsson, B. Wittenmark *Stochastic analysis of real-time systems with random time delays* Proc. 13.th IFAC World Congress, 1996

[7] R. Gerlich, *Early Validation of System Design by Behavioural Simulation*, ESTEC 3rd Workshop on 'Simulation for European Space Programs', Noordwijk, 1994

[8] U. Kiencke, T. Kytoelae, K.J. Neumann *Architectural Trends in Automotive*

Electronics 1st IFAC-Workshop on Advances in Automotive Control, Ascona, 1995

[9] W. Herden, H. Hillner, R. Mayer *Entwicklung, Modellbildung und Simulation in der MSR-Entwicklungsumgebung MESA*, Proceedings der ASIM 94, Stuttgart.