Integrated System Level Simulation Techniques for the Design of Embedded Systems

Ulrich Freund, Volker Zerbe, Gunar Schorcht
Department of Computer Science and Automation
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 in the early stages of the design. To deal with the complexity of the overall system, we use system abstraction methods well known in system 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

The design of embedded systems for mechatronic systems is still done in a somewhat haphazard manner, i.e., the dynamics of the 'plant' the embedded system has to control are not taken into account before the final stage of system integration test. Furthermore, the design of the embedded systems' software does not take into account the systems architecture. Examples for architectural trade-offs are type and number of microprocessors and micro-controllers, scheduling strategies, and communication protocols. The problem of this approach lies in the large number of iterations necessary to sort out all the synchronisation problems. A full redesign is necessary if the chosen architecture is not capable of performing the function within the performance requirements. It could also be the case that the resources of the architecture are not used completely which might result in a too expensive product.

The 'Meet-in-the-Middle' [2, 1] strategy proposes a combination of a bottom-up and a top-down approach to microsystem design - which includes embedded systems. In the bottom-up process every domain involved in the design (e.g. electrical, mechanical or optical engineering) builds up detailed models of the systems components. For the overall system model these component models are replaced by simpler behavioural models, based on parameters, look-up-tables and ordinary differential equations. In the top-down process, the requirements are translated into a system model comprising of simple component models. The requirements are checked via simulation. This means that reference models and key parameters are transfered to the component models and critical parameters are identified of the overall system.

Furthermore, reactive embedded system design is more and more supported by means of performance simulation. Reactive design methods like SDL or Statecharts are coupled with discrete event system level simulation models based on transaction/resource models. This approach is embraced in the aerospace [7] and automotive industries [2, 4, 3]. However, a seamless integrated system level design tool that supports the 'Meet-in-the-Middle' methodology for the design of embedded systems is still missing.

Hence, in this paper we present an integrated system level simulation technique capable of modelling and simulating embedded systems with its environment. This paper is organised as follows: The 'Meet-in-the-Middle' strategy is focussed on embedded systems design. Then the modelling method and its language is introduced. In section 3.3 a class library is introduced, and finally, in the last section an example is given.

2 Model Based System Design

Since an embedded system has to work in an environment, this imposes not only constraints on the design, but also on the interaction with it. If the requirements are not met well, the embedded system could even fail 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 mechanical engineer. Normally, these systems are detailed and thus very complex. But in almost all cases it is possible to find simpler models.

These simple models of all components constitute the environmental model of the system, i.e., the plant in the sense of control engineering. Taking into account the requirements of embedded systems, designing the control laws and modelling all the other functions of the system, we then augment this model with models of the architecture to be 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. The structure of the different models involved is shown in Figure 1

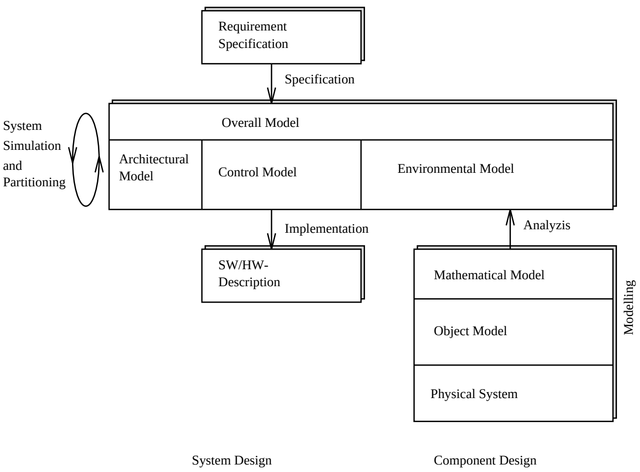

Figure 1. The Modeling Methodology

The interfaces of the embedded system to its environment have to be made. In case of a closed loop control the plant has to be sampled for the digital controller. The controller consists of a functional model based on the requirements and the constraints of the plant. The overall functional model is mapped to an initial architecture. Architectural models represent alternative HW/SW parts. This system level model, described in the HiFAQ(**Hi**erarchical **F**unctional **A**ugmented **Q**ueuing Networks)-language, is translated into a C++ program and executed on a workstation. The results of the simulation are mapped against the requirements. If they are met, one can optimize the system concerning costs (more resource sharing and SW implementation) or speed (more dedicated HW). If the requirements cannot be met by any architecture, then the functional design has to be changed, i.e., a complete redesign is necessary. After the optimal [1] HW/SW partition has been found by integrated system level simulation, one can start generating C/C++ and/or VHDL code. This process is depicted in Figure 2

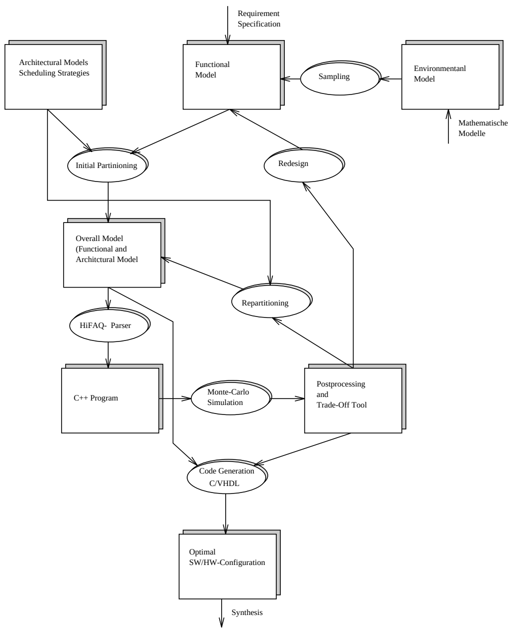

Figure 2. SW/HW Partitioning by means of System Level Simulation

3 The HiFAQ Approach

3.1 Basic Elements

The above approach requires a tool capable of modelling and simulating heterogeneous, embedded systems at system level. An appropriate language is based on HiFAQ's . 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

[1] in the sense of Pareto optimal

to one of the three layers. After the interfaces between the layers are defined, the layers can be modeled separately. Depending on the point of view, one gets either 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 modeled by transaction/resource models. The latter models have the property to behave like simple queuing networks while keeping the identity of the queueing population. Extended finite state machines are adding the concept of variables to the finite state machine definition.

These elements themself allow the classical stepwise refinement approach for each layer. A HiFAQ Model defines the following couplings:

- Intralayer I/O coupling: This kind is used to connect the elements within in a layer.
- State/Parameter coupling: This variant is used for the interlayer coupling. With this method, the interface between the layers and hence the different development parts of an embedded system is defined. On the other hand, these couplings are responsible for the influences of the different layers.
- InterHiFAQ I/O coupling: HiFAQ's are coupled themself 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.

Hardware parts are modeled 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 modeled 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. Due to space limitations here we refer the reader to the detailed

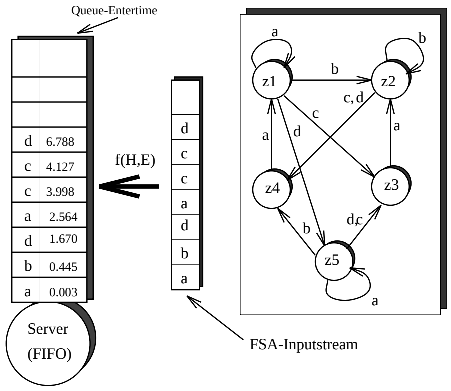

Figure 3. The different levels of a behavioural- and performance-model

theoretical description of HiFAQ's to [5]. Transition times can be determined by the processing time an assembled code segment on a particular processor. Since a single transition does not include any loops or branches, methods like [3] can be used.

3.2 Language Features

The description of a HiFAQ-system consists of three different sections:

1. the signal section
2. the HiFAQ declaration section
3. the HiFAQ system section.

All signals exchanged between the models of a HiFAQ are listed in this section. To some extend, a signal resembles a record in a third generation language. Hence, the signals are referenced by the name and the field. So called continuous signals represent multiple data streams, whereas discrete signals are only defined for a certain event. We further distinguish between internal and external discrete signals. External signals are used to exchange information between HiFAQ's whereas internal signals exchange information between the models of a HiFAQ.

HiFAQ's are declared in the second section of the description. HiFAQ's consist of an interface and a behavioural description. The interface description shows which signals are transmitted by which port. The behavioural is described by the models of the HiFAQ.

The models are nested in the description of the HiFAQ. Models themself consist of an interface and a behaviour section. Models receiving and sending external signals are mapping their ports onto the HiFAQ ports. All other ports are mapped against other models. The semantics of model coupling is implemented in the following way: A transformatorial model is assigned to an reactive model. This assignment determines the sampling of the continuous model. In an equidistant sampling case, the value of a variable is put into a signal while the automation is performing a transition, which might be caused by a triggering event of a clock. The value of the variable is determined by the transformatorial model which is assigned via parameter/state coupling to the extended finite state machine. State events are detected by the same way. However, in this case the actual value of the continuous process is checked during a transition of the reactive model. Reactive models are mapped onto architectural models, mapping the function to the architecture.

The last section of a HiFAQ system is the system itself. In a variable definition section, the instances of the HiFAQ's are named and connected with their types. The variable definition section is proceeded by the initialization section, where the instances are generated and the parameters are passed to the instances. The last section of the system section states the structure of the HiFAQ's, i.e., how a HiFAQ is connected to its neighbors.

3.3 The simulation class library

We developed a compiler for the HiFAQ language which translates HiFAQ models into C++ code. The C++ code uses a class library capable of simulating continuous, time driven and event driven models simultaneously. Simulation time is forwarded by events and integration algorithms. While doing integration between the events, the time of the next event is monitored so that the system can react on state events.

4 Example

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. Bus systems are common in aeronautical and automotive electronics [8]. Since these systems are deployed more and more in safety-critical applications, a combined assessment of behavior and performance becomes mandatory. In this chapter it is shown how different architectures can change the stability of a dynamic system.

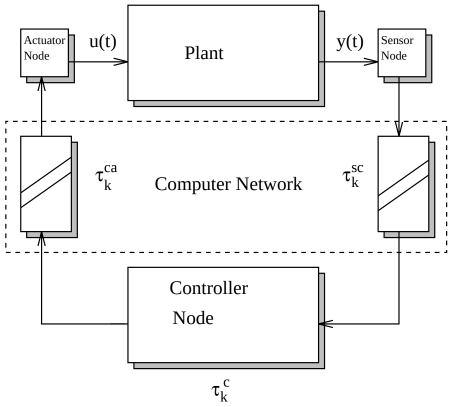

Figure 4. Model of a control loop coupled by a communication network

In a typical distributed control system the controller, the actuator, and the sensor are located at different locations connected via a bus system. If we also assume that the bus is used by other stations, one comes up with the model of Figure 4.

A controller receives measurement values from the plant, delayed by the time τ_{sc}. The measurement value is compared with the setpoint and an error signal is fed to the control algorithm. An actuator value is calculated according to some control law during time τ_c. This value is transmitted to the plant, delayed by time τ_{ca}. We simulated the delays $\tau_{sc}, \tau_c, \tau_{ca}$ by mapping the control algorithm and the plant model to an architectural model of a Token-Bus system. Since in our model there are additional stations, sending random length messages at random instants, connected to the bus, the delays are random and may exceed the duration of a sampling time step. This violates the assumption of methods trying to make the control algorithms robust to random delays, e.g. [6]. Hence, integrated simulation is currently the only means to analyse distributed control systems without the above mentioned limitations.

The plant and the controller are described by second order ordinary differential equations. We investigated two different implementations: In the first implementation the samples are queued in the controller station and the actuator values are also queued until the actuator sampling takes place. In the second implementation no queues are used. Every sample is used to produce an actuator value. The actuator puts the value onto the plant immediately. Figure 5 shows the results of the first implementation in comparison with the equidistant sampled, no queueing, delay free functional model.

With the second architecture one can achieve a better performance of the overall system due to overshooting. This case is shown in Figure 6. With these results, it is up to the design team to decide how the system should be changed, by means of a modified control-law (redesign) or a different architecture (repartitioning).

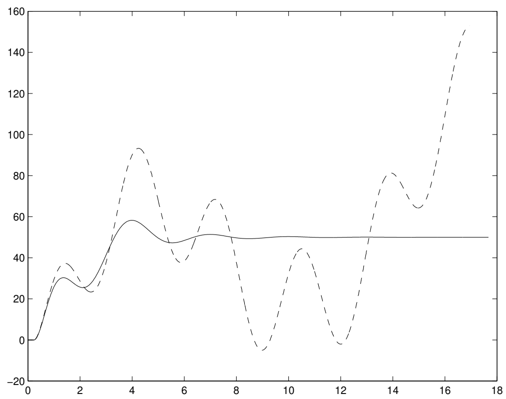

Figure 5. Step response for the original (solid line) and a time driven implementation (dashed line)

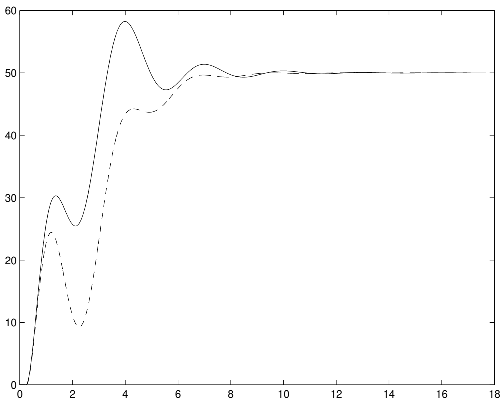

Figure 6. Step response for the original (solid line) and an event driven implementation (dashed line)

5 Conclusions

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

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

References

[1] K.D. Mueller Glaser, *Systems Engineering in Microsystems Design*, Proceedings of the IFIP Workshop on Modelling of Microsystems, Stirling, 1997

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

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

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

[5] U. Freund, V. Zerbe, G. Schorcht *Hierarchical Simulation Techniques for the Design of Heterogeneous, Embedded Systems*, IFIP Workshop on Modelling of Microsystems, University of Stirling, 1997, or http://www-sst.theoinf.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