From High–Level Specifications down to Software Implementations of Parallel Embedded Real–Time Systems

C. Rust, F. Stappert, P. Altenbernd, and J. Tacken
C-LAB
Fürstenallee 11, 33094 Paderborn, Germany
{car,fst,peter,theo}@c-lab.de

Abstract

In this paper we describe a methodology and accompanied tool support for the development of parallel and distributed embedded real–time system software. The presented approach comprises the complete design flow from the modeling of a distributed controller system by means of a high–level graphical language down to the synthesis of executable code for a given target hardware, whereby the implementation is verified to meet hard real–time constraints. The methodology is mainly based upon the tools SEA (System Engineering and Animation) and CHaRy (The C–LAB Hard Real–Time System).

1. Introduction

In the field of embedded systems development there is currently no seamless design flow from high–level modeling down to the implementation on a specified target hardware, e.g. a network of interconnected microcontrollers. Tool support exists for the graphical specification and simulation of systems. There is also tool support for the implementation of an embedded system on a target processor. However, between these design steps there is a gap in the tool chain. Thus, several design activities, like partitioning and code generation, still have to be done manually, which can be a tedious and error prone process.

In this paper we describe a methodology and accompanied tool support for the development of embedded real–time system software. Embedded systems are typically used for the control of physical devices. These controllers behave periodically: In a loop, the current sensor data is read and, according to internal computations, the values of some actuators are adjusted. The correct behavior has to be assured with respect to both the functional requirements as well as to hard real–time constraints on the computational part of the loop. These constraints are imposed by the controlled physical devices. Furthermore, embedded systems are by nature distributed or parallel, and todays applications (e.g. cars) frequently involve many micro–processors to serve different special–purpose demands (e.g. cruise control, ABS, air–bag, motor management, etc.).

The methodology presented in this paper is mainly based upon the tools SEA (System Engineering and Animation) and CHaRy (The C–LAB Hard Real–Time System), that were developed at C–LAB. The use of both tools within our methodology and their interaction is shown with the application example of an Anti Blocking System of a car.

SEA is an environment for the modeling and interactive simulation of heterogeneous systems. The tool is based on the formal model of Predicate/Transition–Nets (Pr/T–Nets). The idea is to describe the controller on a high abstraction–level using SEA, and leave the implementation to CHaRy. During the modeling phase the designer uses a high–level graphical language, in our example Software Circuits. Within SEA the model can be interactively simulated and modified until it is functionally correct. Afterwards the model is transformed into an exchange format establishing the interface between SEA and CHaRy.

CHaRy is a software system for the synthesis of periodic controller applications, where hard real–time conditions must be guaranteed for software tasks. Since this is a very complex problem, CHaRy decomposes it into the sub–problems of mapping the controller models to a number of tasks (partitioning), the extraction of their worst-case execution times (timing analysis), and their assignment to a processor network (allocation), so that all hard real–time conditions are guaranteed (feasibility analysis). The output of CHaRy is executable code for each processor in the network together with a system configuration. The latter specifies the services needed from a given real-time operating system (e.g. task scheduling, inter task communication, device drivers or memory management).

The rest of the paper is structured as follows. Section 2 gives an overview of related work. Section 3 shows the proposed design flow. Section 4 describes the modeling and simulation using SEA, and Sections 5 and 6 describe timing analysis and the scheduling/allocation performed by CHaRy. Finally, a conclusion is given in Section 7.

2. Related Work

Design Environments: Similar to the work presented here, the Ptolemy project [10] studies modeling, simulation, and design of concurrent real–time embedded systems. The focus is on assembly of concurrent components. The underlying principle in the project is the use of well–defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. Unfortunately, for the majority of the allowed computation models no synthesis is available and therefore no seamless design is possible, as in our approach.

The reduction of complexity by means of a rigorous problem decomposition is well–known in the field of hardware high–level synthesis [14]. However, for hard real–time software systems this is relatively new, and hence it is often not very well supported by tools. When using ordinary real–time design methodologies like HRT–HOOD [4] or ROOM [20], many design steps are still hand–made. Although problem decomposition usually leads to sub–optimal solutions, it is in our opinion the only way to handle very large systems.

Scheduling and Allocation: The problem of assigning the software of periodic controller algorithms to parallel embedded computers in a way that the real–time conditions are met, is very complex. Many known approaches in this field try to handle Scheduling and Allocation simultaneously or have problems to locate feasible assignments (e.g. [19]). However, following a distinct handling of Scheduling and Allocation, our efficient heuristic [2] delivers good results even for larger applications.

Timing Analysis: A tight worst-case execution time (*WCET*) estimation must comprise both a high level analysis (HLA) on the source code level as well as a low level analysis (LLA) on the assembler code level. The HLA deals with finding the longest executable path in the control flow of a program, whereas the LLA predicts the execution time of sequences of assembler instructions considering the effects of caching and pipelining. In contrast to most current approaches to WCET analysis, our work comprises both LLA as well as HLA.

Current approaches for the HLA differ mainly in the methods used for program path analysis. In e.g. [6] graph algorithms are used to analyze the control flow graph. In contrast, [18] uses ILP to specify the problem of finding the longest executable path in the control flow. Due to complexity problems, it is not feasible neither for graph algorithms nor for ILP approaches to explore all possible paths in the control flow graph. However, our graph-based approach uses a heuristic [1] with a good trade-off between accuracy and computation-time consumption.

Most publications on LLA concentrate on only one of the aspects caching or pipelining separately, e.g. [15] on caching or [16] on pipelining. Our approach integrates both aspects by using the results of the caching analysis for the pipelining analysis ([21]).

3. Design Flow

In this section we give an overview of how a parallel embedded real-time system is modeled following our design methodology (see Figure 1). The design starts with a specification of the system under development. The specification is based on extended Pr/T–Nets [12] as a formal model. We use a high-level petri net model, since petri nets are well suited for the design of concurrent systems and allow a natural modeling of reactive systems due to their asynchronous characteristics. Furthermore, many useful analysis methods for petri nets already exist. Due to our extensions to the basic model of Pr/T–Nets it is also possible to use other specification languages. They can be transformed into our unique formal model in several ways [13]. For the work

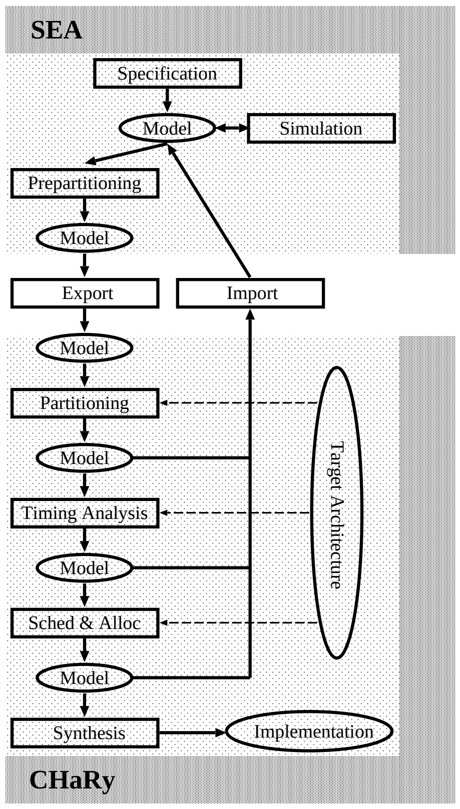

Figure 1. Design flow

presented in this paper we use a library for the specification of so called Software Circuits [8]. The library (see Section 4) addresses the problem of modeling periodic processes. Thus it is easier and thereby less error-prone to use this library for specifying the systems we have in mind with our methodology.

A model specified by the user contains the functional behavior of the periodic processes as well as their periods and deadlines. As Pr/T–Nets form an executable specification method, system evaluation is supported from the very beginning. That means, just after a specification of the system under construction has been defined, this model can be executed within the SEA Environment using its interpretative simulation component or a C++ prototype, that can be generated from the model. Primarily, the simulation allows to check whether the computations of the different model parts are functionally correct. Furthermore, it is possible to observe the temporal behavior assuming that it is possible to guarantee the specified periods and deadlines for an implementation. Since neither the execution order of the different tasks nor their execution times are known, the simulation of the temporal behavior is rather imprecise at this stage of our design flow.

In order to generate an implementation from the specification the model is given to CHaRy. The model is transformed into an exchange format defined for this purpose. The thereby generated input for CHaRy comprises a set of tasks described in a C–like imperative language and the periods and deadlines for these tasks. Before exporting the model, SEA performs a partitioning of the user specified tasks into smaller units. This prepartitioning [22] defines the smallest units for the final partitioning performed by CHaRy. In contrast to the final partitioning the prepartitioning is computed with respect to characteristics of the underlying Pr/T–Net model.

CHaRy first approximatively estimates the execution times of all units produced by the prepartitioning. With respect to these values as well as to the expected communication times between different processes the final partitioning for the implementation of the system is determined. This is done by clustering the atomic units given as input. For the partitioning as well as for the following steps CHaRy needs a specification of the target architecture (e.g. a set of microcontrollers connected via a CAN-bus). Having defined the partitioning, an accurate worst-case execution time analysis is performed. This step is described more detailed in Section 5. Afterwards, CHaRy determines an allocation of the tasks to processors and computes a schedule for tasks assigned to the same processor as described in Section 6. This step yields information for the configuration of the target architecture, which - together with the object code for all tasks - forms an implementation of the modeled system. The implementation is created in the final step of synthesis. This step includes the generation of communication components for distributed target architectures (cf. [25, 17]). For this purpose we rely on the library operating system DReaMS. It provides customized communication support for distributed embedded applications [7]. Besides the implementation, CHaRy also yields several informations about the implemented system, e.g. the worst-case execution time of all tasks, their schedule and thereby also their release and response times. The response times are compared with the specified deadlines in order to check the real–time conditions. All informations may be integrated into the model specified by the user. This enables the engineer to simulate the behavior of the final implementation within the SEA Environment in order to check whether the implementation fulfills all requirements.

4. Modeling and Simulation

The basis for our modeling approach is the SEA language [12]. It allows to hierarchically specify the structure and the behavior of a heterogeneous system in an application oriented way. Freely defined graphical elements as well as predefined elements e.g. lend from existing graphical languages can be used. As a unified formal semantical basis extended Predicate Transition Nets (Pr/T–Nets) are used. They build the bottom level of hierarchy in each specification. Via this unified formal semantics the integration of specification elements from different application domains, as it is needed for the engineering of heterogeneous systems, can be reached. Furthermore, the underlying Pr/T–Net allows a simulation/animation of the graphical system specification at arbitrary levels of hierarchy. Even the freely defined graphical elements like pictograms can be animated.

In this paper we use a library of predefined elements for the specification of Software Circuits [8]. The library supports a simple specification process comprising three main steps. First, the engineer specifies the different computational parts of the system by block diagrams. For the specification he can either use predefined elements, e.g. for arithmetic operations, or easily define own elements for more complex operations. An example for such elements, that consist of a Pr/T–Net and a graphical interface, is depicted in Figure 2. Besides the operations the diagrams may contain inputs and outputs, which form the interface of the specified element, as well as actuators and sensors. Hierarchical definitions are also supported.

The specified diagrams form the basis for the second step of the specification process, the definition of tasks. A task is defined by instantiating a diagram and defining a period as well as a deadline for the task. For each specified task, our modeling tool SEA automatically creates an extended Pr/T–Net, that has the functional behavior defined by the diagram

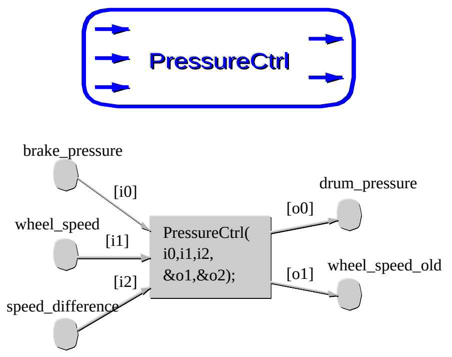

Figure 2. Software Circuit library element

as well as the temporal behavior specified by period and deadline. Furthermore SEA generates a graphical interface for the diagram according to its inputs, outputs, sensors and actuators.

In the final modeling step the engineer connects the previously defined elements to the top level model. Figure 3 shows the top level view of our application example. It contains one task for observing the pressure put on the brake and four tasks realizing the anti blocking system at each wheel. The graphical interfaces of the tasks visualize their status (computing/waiting), the last input values sent to the respective component and the sensor and actuator values.

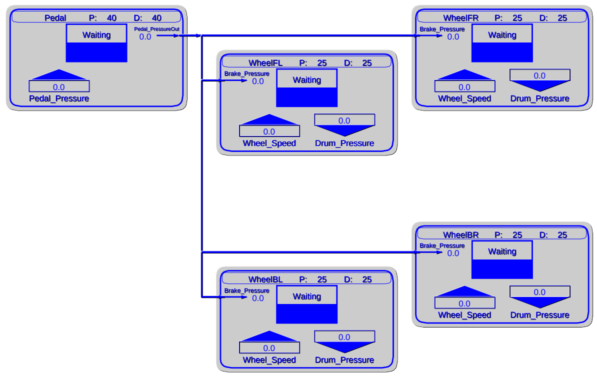

Figure 3. SEA specification of application example

Modeling an embedded real-time system as described above, the engineer has not to deal with petri nets at all. Nevertheless, he defines a hierarchical Pr/T–Net, that can be executed by means of the simulation facilities of our modeling environment. Simulation runs are supported on all hierarchy levels. Thus the user can investigate the computational behavior of single components as well as the behavior of the whole system over the time. Sensor values and - on lower hierarchy levels - input values for single components can be set interactively during simulation. As described in Section 1, the simulation reflects the real behavior of the final implementation, since all relevant informations about the implementation are collected by CHaRy during the analysis and synthesis steps and integrated into the model.

5. Timing Analysis

The WCET analysis is initiated after clustering the units produced by the prepartitioning. In contrast to the timing assumptions used during the clustering process (where average values are needed), the WCET analysis tries to give an *exact* prediction of the WCET of the tasks. The WCET analysis is mainly divided in two modules: the Low-Level Analysis (*LLA*) which works on the level of machine instructions, and the High-Level Analysis (*HLA*) which analyses the control flow of a task on the source code level.

The LLA covers all speed-up mechanisms used for modern superscalar processors [9]: pipelining, instruction-level parallelism and caching. The pipelined and parallel execution of assembler instructions is analyzed for each basic block (i.e. each node of the control flow graph) of a task. Also, the LLA predicts whether memory accesses will hit the cache at run-time.

The HLA uses the results from the LLA to compute the final estimate on the WCET. This is done by a heuristic for searching the longest really executable path in the control flow, i.e. by taking into account functional dependencies between various program parts [1]. The LLA computes a representation of the state of pipeline and cache at the beginning and at the end of each basic block, i.e. before and after execution of its machine instructions. The HLA then computes the length of a path in the control flow graph by concatenating the state representations of the basic blocks on this path. The concepts used in our WCET analysis are described in more detail in [21].

Obviously, the LLA is strongly hardware dependent, i.e. the cache and pipeline architecture have to be taken into account. However, while a cache configuration can easily be parameterized, pipeline structures of various processors differ considerably. Thus, the timing analysis has to be adapted to each new target processor. Within CHaRy, the pipeline architecture is modeled as a set of C++ base classes which can be configured for a given processor architecture with reasonable programming effort. The current implementation of CHaRy supports the PowerPC 604 architecture.

The result of the timing analysis process is a tight estimate on the worst-case execution time of each task. This estimate is then used by the scheduling and allocation module in order to compute the final implementation of the model so that all real-time constraints are met.

Concerning the application example a WCET of 180 clock cycles for the task Pedal and a WCET of 1128 clock cycles for the tasks running at the wheels was computed by CHaRy. Actually we have not compared these value to the real execution times. But as earlier experiments with the PowerPC 604 have shown, CHaRy typically overestimates the WCET about 4 % up to 13 % [21].

6. Scheduling and Allocation

During scheduling and allocation the analyzed tasks are assigned to hardware devices resulting in a system configuration, telling which task is to be run where and when.

The problem to overcome is to map the tasks to a number of interconnected microprocessors such that

- all timing constraints are satisfied
- some objective function, expressing the resources used, is optimized

Because this problem is known to be NP hard [24], use of heuristics is preferred. In particular, the process is divided into the sub-problems allocation for the overall mapping, and the actual local resource scheduling.

The allocation is an iterative process based on Simulated Annealing [11], a well-used general optimization technique. Within each iteration, a new mapping is produced which is then checked for feasibility w.r.t. real-time constraints, and a value for the objective function is determined. If the assignment is both feasible and cheaper, it is accepted as a new optimum, otherwise, it is rejected. As shown in [2] efficiency is drastically improved when pre-processing the iterative simulated annealing approach. However, other allocation techniques may be applied as CHaRy is open for any other method, as well.

For the scheduling of each local resource (processor or network link) several properties can be distinguished. The scheduling can either be performed on-line (i.e. at run time) or off-line (i.e. computing a static cyclic schedule). In any case the scheduling is based on fixed priorities assigned by policies, like RMS, DMS, or LLF. The scheduling algorithm can be pre-emptive or non pre-emptive. Task invocation can be time triggered (each task is started at a specified instant in time), or event driven (each task is started when its input data is complete). During the check on feasibility [3, 23] the properties of all used resources are correspondingly taken into regard, as each of them has a different impact on the global behavior. Furthermore, a combination of caching and pipelining analysis and preemptive scheduling is possible, by adding task-switch overhead for refilling caches and pipelines into the analysis [5]. Communication calls are synthesized during code generation, resulting in synchronous mechanisms for tasks of the same period length, and in asynchronous mechanisms for tasks with different periods lengths (using semaphore locking in both cases) [8]. Consequently, corresponding worst-case blocking delays are incorporated into the analysis. Note, that scheduling (and the check on feasibility) is given for both micro controllers and network links.

The following table shows the results of the WCET analysis and scheduling/allocation computed by CHaRy for the application example.

Task	**WCET**	**Processor**	**Period/ Deadline**
Pedal	180 cycles	driver	40 ms
WheelFL	1128 cycles	front_left	25 ms
WheelFR	1128 cycles	front_right	25 ms
WheelBL	1128 cycles	back_left	25 ms
WheelBR	1128 cycles	back_right	25 ms

The final scheduling and allocation of the example is rather trivial. An elaborate test-case evaluation of CHaRy's allocation component is provided in [2].

7. Conclusion

We have presented a methodology and accompanied tool support for the seamless development of parallel embedded real-time systems. The seamless design flow is accomplished by the interaction of the tools SEA and CHaRy, iteratively refining and simulating a Pr/T–Net model of the system under development. The final implementation of the model on a distributed controller network is synthesized automatically. Our methodology was presented using the rather small application example of an Anti Blocking System. Currently we are evaluating our approach performing more complex case studies.

References

[1] P. Altenbernd. On the false path problem in hard real-time programs. In *8th Euromicro Workshop on Real Time Systems (WRTS)*, pages 102–107, L'Aquilla, Italy, June 1996.

[2] P. Altenbernd and H. Hansson. The Slack Method: A New Method for Static Allocation of Hard Real-Time Tasks. *Kluwer Journal on Real-Time Systems*, 15:103–130, 1998.

[3] N. C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times. Report YCS164, Department of Computer Science, University of York, 1991.

[4] A. Burns and A. J. Wellings. HRT-HOOD: A Structured Design Method for Hard Real-Time Systems. *Journal of Real-Time Systems*, 6(1):73–114, 1994.

[5] J. Busquets-Mataix, J. Martin, R. Carot, and A. Wellings. Adding Instruction Cache Effect to an Exact Schedulability Analysis of Preemptive Real-time Systems. In *Proceedings of the 8th Euromicro Workshop on Real-time Systems*, 1996.

[6] R. Chapman. *Static Timing Analysis and Program Proof.* PhD thesis, Computer Science Department, University of York, 1995.

[7] C. Ditze and C. Böke. Supporting software synthesis of communication infrastructures for embedded real-time applications. In *Proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems (DCCS)*, 1998.

[8] H. Hansson, H. Lawson, M. Strömberg, and S. Larsson. BASEMENT: a Distributed Real-Time Architecture for Vehicle Applications. *Real-Time Systems*, 11(3):223–244, November 1996.

[9] J. L. Hennessy and D. A. Patterson. *Computer Architecture:A Quantitative Approach*. Morgan Kaufman Publishers, 1996.

[10] J. D. II, M. Goel, C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong. Overview of the ptolemy project. ERL Technical Report M99/37, Dept. EECS, University of California, Berkeley, CA 94720, July 1999.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimisation by Simulated Annealing. *Science*, 220(4589):671–680, 1983.

[12] B. Kleinjohann, E. Kleinjohann, and J. Tacken. The sea language for system engineering and animation. In *Applications and Theory of Petri Nets*, LNCS 1091, pages 307–326. Springer Verlag, 1996.

[13] B. Kleinjohann, J. Tacken, and C. Tahedl. Towards a complete design method for embedded systems using predicate/transition-nets. In *Proc. of the XIII IFIP WG 10.5 Conference on Computer Hardware Description Languages and Their Applictations (CHDL-97)*, pages 4–23, Toledo, Spain, Apr. 1997. Chapman & Hall.

[14] M. C. McFarland, A. C. Parker, and R. Camposano. The High-Level Synthesis of Digital Systems. *Proceedings of the IEEE*, 78(2):301–318, February 1990.

[15] F. Mueller. Timing Predictions for multi-level caches. In *Proc. ACM SIGPLAN Workshop on Languages,Compilers and Tools for Real-Time Systems (LCT-RTS'97)*, pages 29–36. ACM, June 1997.

[16] K. Narasimhan and K. D. Nilsen. Portable Execution Time Analysis for RISC-Processors. In *ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*. ACM, 1994.

[17] R. Ortega and G. Borriello. Communication synthesis for distributed embedded systems. In *Proceedings of the International Conference on Computer Aided Design*, 1998.

[18] G. Ottosson and M. Sjödin. Worst-Case Execution Time Analysis for Modern Hardware Architectures. In *Proc. SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (LCT-RTS)*. ACM, June 1997.

[19] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 108–115, 1990.

[20] B. Selic, G. Gullekson, and P. T. Ward. *ROOM – Real-Time Object-Oriented Modeling*. Wiley & Sons, Inc., 1994.

[21] F. Stappert and P.Altenbernd. Complete Worst-Case Execution Time Analysis of Straight-line Hard Real-Time Programs. *Journal of Systems Architecture*, 46:339–355, to appear 2000.

[22] J. Tacken, C. Rust, and B.Kleinjohann. A method for prepartitioning of petri net models for parallel embedded real-time systems. In *Proc. of the 6th Annual Australasian Conference on Parallel And Real-Time Systems (PART '99)*, Melbourne, Australia, Nov. 1999.

[23] K. Tindell. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Report YCS197, Department of Computer Science, University of York, 1993.

[24] K. Tindell, A. Burns, and A. Wellings. Allocating Real-Time Tasks (An NP-Hard Problem made Easy). *Journal of Real-Time Systems*, 4(2):145–165, 1992.

[25] T.-Y. Yen and W. Wolf. Communication synthesis for distributed systems. In *Proceedings of the International Conference on Computer Aided Design*, 1995.