Task Set Design Tools for an Embedded Distributed Control System*

Emmerich Fuchs, Dietmar Millinger
Real-Time Systems Group,
Vienna University of Technology
Treitlstr. 3/182/1, A-1040 Vienna, Austria
E-mail: {emmerich,dietmar}@vmars.tuwien.ac.at

Abstract

The design of an advanced distributed embedded control system, e.g., in the automotive or aerospace industry, is often approached cooperatively by a system manufacturer and several subsystem suppliers. Therefore a decomposition of the system into smaller subsystems with well-defined, stable interfaces among them must be developed. The internal design of these subsystems can then be based on the definition and the properties of the subsystem interfaces.

In this paper we present the Time-Triggered Architecture (TTA) supporting such a design approach and concentrate on the design and implementation of a subsystem developed by one supplier. We propose a set of local design tools for the application developer that supports the systematic and efficient development of application tasks on top of a time-triggered operating system.

1 Introduction

The design of an embedded distributed control system is often carried out by the system manufacturer in close cooperation with the subsystem suppliers, e.g., in the automotive or the aerospace industry. A well known example from the aerospace industry is the Integrated Modular Avionics (IMA) where the modules provided by different suppliers are integrated into a cabinet by the SAFEbus [HD92] (Honeywell's implementation of ARINC 659[Inc93]). This manufacturer/supplier relationship is the main reason why we have structured the design process in the Time-Triggered Architecture (TTA)[SHS+97] into two phases and put special emphasis on the design of the interfaces between the system manufacturer and the subsystem suppliers. More information concerning the Time-Triggered Architecture can be found in the book by Kopetz [Kop97] and concerning the communication subsystem in [KG94, KHK+97].

We will first describe the *global design* issues as they are the basis for the *local design* on which we will concentrate in the rest of the paper. The global design deals with all issues that are relevant for more than one subsystem and where the activities at the different subsystems must be coordinated to operate in a consistent and highly synchronized manner. The tool set described in this paper is based on an adapted and extended version of TRAPPER, a software engineering environment for parallel applications [SSKF95].

In Section 2 the main activities and tools required for the design of all system-level issues are briefly addressed. The main topic of this paper, the task model and the local design tools are addressed in Sections 3 and 4. The paper concludes in Section 5.

*This work was supported by the Esprit OMI Project 23396, Time-Triggered Architecture (TTA).

2 Global Design

The design process for a distributed real-time system is to a large extend constrained by the architecture chosen for the system.

2.1 Time-Triggered Architecture

The Time-Triggered Architecture (TTA) is an architecture for distributed real-time systems. In the TTA all control actions (e.g., sending of messages, activation of tasks) are triggered by the progression of a globally synchronized time base. A TTA system is decomposed into a set of subsystems called *clusters*, with each cluster realizing a particular application function. A cluster is a distributed computing system consisting of a set of *nodes* that are interconnected by a communication subsystem. A node in the TTA consists of the communication controller and the host subsystem. For the constructive design of a TTA cluster, the communication network interface (CNI) between the host subsystem and the communication controller is from utmost concern. Due to its time-triggered nature, the communication subsystem provides a synchronous communication service with bounded communication delays between the sending and receiving CNIs. Furthermore, the exact points in time, when messages are delivered in the CNI and fetched from the CNI can be planned exactly before the run-time of the system. These stable properties of the CNI are used to separate the global design of interactions between the nodes in the cluster from the local design of task activities within one node (set of nodes).

2.2 Design Process

The global design for our type of architecture is concerned with a list of activities described below, until all configuration and scheduling tables for all nodes of the distributed system can be generated, see Figure 1:

- Specification of attributes common to all nodes within the distributed system.
- Partitioning into a set of hardware nodes connected by a communication network (broadcast bus).
- Configuration and replication of critical hardware nodes to achieve fault-tolerance.
- Specification of operational modes in which the distributed system may operate.
- Specification of communication relations between the individual nodes of the distributed system including history state, redundant messages, and the timing constraints for the communication relations.
- Scheduling of the messages on the communication network under the given timing constraints.
- Manual editing of the automatically generated communication schedules.
- Generation of the configuration tables for all nodes of the distributed system.

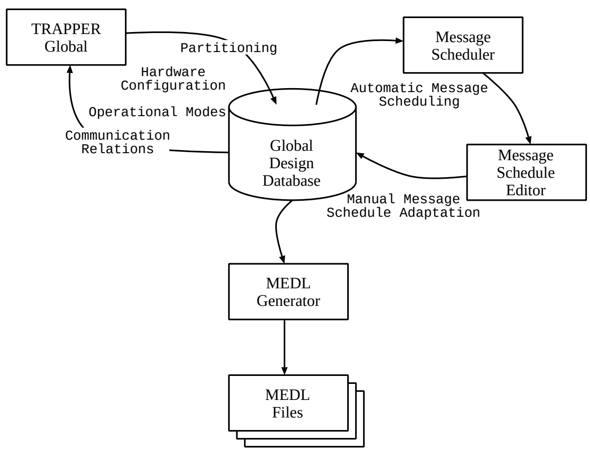

Figure 1: Overview on the global design activities.

The different design steps listed above describe an iterative design approach where the final global design evolves over time until all participants, the system manufacturer and the subsystem suppliers, agree on a design of the global design data.

After such a point of unanimous agreement has been reached the subsystem suppliers can proceed with their application design (local design) independently and independently from the system manufacturer. If however, during the local design of the suppliers application any problems arise that make it impossible to achieve the desired functionality with the current global design then a re-iteration of the global design must be carried. Of course such a re-iteration can become a very costly endeavor since the changes requested on the global design e.g., communication schedule, by one subsystem supplier may adversely affect the local design of another supplier.

Again all participants must reach an unanimous agreement on the global design data before they can proceed with their local design. Therefore, changes to the global design data after work on the application design has started should be handled with extreme care and may become prohibitively expensive in case substantial changes are required.

2.3 Global Transactions

The agreed global design of a system consists of a structural and a behavioral description of the system. The structural description defines the number and purpose of the computation nodes within one cluster. The behavioral description consists of a list of *global transactions* (GT). A GT is a directed graph consisting of vertices that describe computational actions and directed connection lines between vertices that describe communication interrelationships (see Figure 2). Computational actions are called *local transactions* (LT), which are assigned to distinct host subsystems.

A global transaction has a defined period of activation and a defined deadline. The phase relations between the different GTs of a system are defined. During the global design phase for every GT a time division defining the time intervals reserved for LTs on computation nodes and communication actions between the LTs is established.

The time divisions for the communication activities are the basis for message scheduling on the time-triggered communication subsystem.

The time divisions of all LTs on a distinct node result in activity patterns, which are determined by the properties of the LTs. An LT on a node has a defined starting point in time, a defined deadline and defined input/output messages. Resulting from the plurality of GTs on a system, a plurality of LTs compete for processing time on a computation node (see Figure 3).

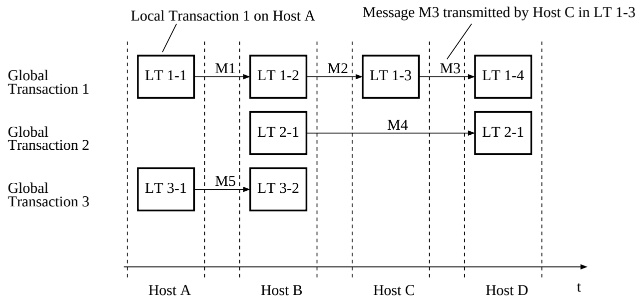

Figure 2: Global Transactions.

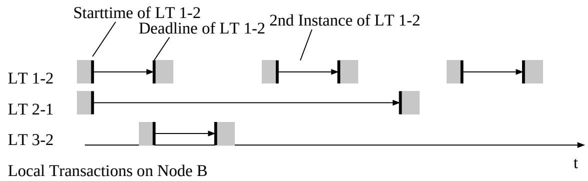

Figure 3: Local Transactions on a Node.

3 Task Model

In the scope of the TTA project [SHS+97] the local design tools were implemented for a time-triggered operating system (TTOS). TTOS was evolved from the MARS operating system [KFG+93].

A task in TTOS is an executable unit of program code. If a memory management unit is available, a task is assigned to a process. In the context of the process the access to memory resources is defined for a task. All memory resources required by the task are allocated before the runtime of the system.

3.1 Dispatching

The task model supported by TTOS is completely based on the time-triggered paradigm. The invocation of every single task is triggered by the passage of time rather than by external or internal events. There are only non-blocking communication and synchronization mechanisms available during the task execution by the processor. Once the task execution is started, the task cannot be blocked until it finishes execution or the task is preempted by another task (see Figure 4). Every task must have a known worst case execution time (WCET)[PK89, PS97a] that determines the maximum execution time of the task at a given host subsystem without preemption. The real maximum run time of a task is determined by the number and length of preemptions of a task. This time is called the worst case run time of a task (WCRT, see Figure 5).

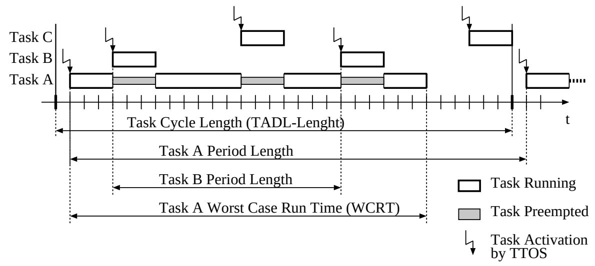

Figure 4: Task Dispatching Model.

The points in time when the execution of a task is started are determined by a time table, the task definition list (TADL). The TADL is generated before runtime and stored in the local non-volatile memory of the host subsystem. It is the generated output of the local design process. The TADL consists of a list with a finite number of entries, which wraps around after a certain time interval, the *TADL-length*.

The time base used by the TTOS dispatcher is derived from the globally synchronized time base of the TTP/C communication system. Therefore all time-triggered nodes in a TTP cluster execute their task sets in tight synchronism.

3.2 Synchronization

All synchronization requirements of a task set, like mutual-exclusion and precedence constraints must be fulfilled by the task activation pattern stored in the TADL. The pre-runtime scheduling tool that generates the TADL for a given node obeys the synchronization requirements of the task set by finding task execution intervals (from task start time to task start time plus WCRT) that fulfill the synchronization requirements.

3.3 Message Communication

Message communication between tasks is not performed during task execution, input messages are received before the start time and output messages are transmitted after the worst case execution time of the task (see Figure 5). This ensures a deterministic and jitter-free communication behavior of a task. The TADL determines the set of messages a task receives and transmits.

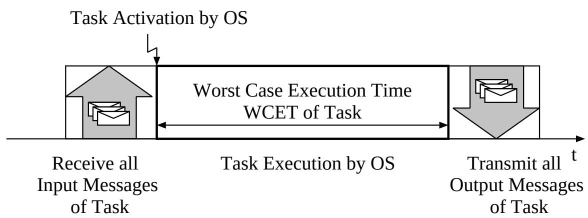

Figure 5: Task Communication Model.

4 Local Design Tools

Similar to the message scheduling in the global design we take an interactive approach during task scheduling, because experience in the last couple of years at our department has shown that such an approach suits the needs of an application developer better than a fully automated approach. The local design of a subsystem developed by one supplier is concerned with the activities depicted in Figure 6:

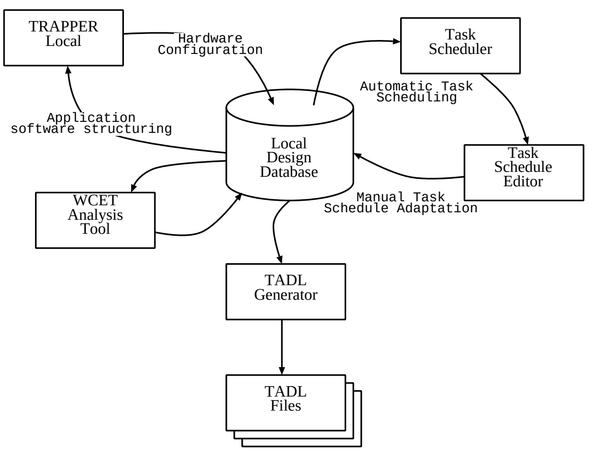

Figure 6: Overview on the subsystem-local design activities.

4.1 Application Structuring

During this first design step the application structure of a host subsystem is defined. For all local transactions (LT) located at the subsystem, tasks are defined, which provide the services requested by the LT. Between the tasks of an LT, communication relations and precedence constraints (mutual exclusion, task precedence) are defined. The tasks of an LT inherit the timing parameters of the LT, the earliest possible starting time and the deadline.

4.2 Worst Case Execution Time

After coding and compilation of the tasks, a worst case execution time analysis is required. The resulting WCET times for tasks are an additional input for the task scheduler. At the present there are no fully functional tools for automatic WCET analysis available. An overview of the current state of research is given by Puschner in [PS97b, Pus98]. In this phase of the local design process we go back to timing measurements. Based on the execution time measurements, an estimation for the WCET is generated. Violations of the WCET are detected by the operating systems deadline monitor.

4.3 Automatic Task Scheduling

In this phase of the local design sufficient information is available to schedule the tasks of a host subsystem. The constraints for the scheduler are:

Task Set The number of tasks is static and known.

Timing The WCET, the minimal allowed start time and the deadline is given for every task in the task set.

Synchronization The precedence constraints for synchronization purposes are defined by the means of attributed relations between tasks.

Communication The external communication constraints are not regarded by the scheduler, because they are resolved implicitly by the timing parameters of the tasks. The internal communication constraints are regarded by implicit mutual exclusion constraints between sender and receivers of local messages.

Before the scheduler is activated a schedulability test is conducted. Conditions which are tested are the maximum expected processor load and the processor load within the time bound of LT's.

The scheduling strategy of the automatic scheduler is based on the *least laxity* strategy. The processor time is divided into equidistant granules, called *macro-ticks*. For every macro-tick the task set is examined, and the task with the smallest difference between the estimated end time and the deadline is selected for execution in this macro-tick. The automatic scheduler assigns start times for all tasks of the task set.

4.4 Manual Schedule Editing

The task schedule editor is a tool that displays all relevant parameters of the local design. The tool allows the user to manually edit the results of the automatic scheduler. The editor provides three views with different editing facilities (see Figure 7).

List View This view contains a list of all task invocations, including the start time and the scheduling status of the tasks. The scheduling status informs the engineer, whether all constraints are met for the task by the current schedule, or if a constraint is violated, a description of the violation is listed. In this view the task start times can be edited by defining a numerical value.

Time Bar This view shows the execution patterns of all tasks over the complete length of the TADL. The task start times can be changed by dragging the start line of the task over the time axis. With every change of a task start time, the schedule is recalculated. To support a constructive generation of a task schedule, groups of tasks can be frozen, excluding them from rescheduling. Frozen tasks can be grouped horizontally defining time intervals which are locked, or vertically, defining local transactions for each of which the tasks are frozen. By this means a designer is able to refine a task schedule step-wise.

Tasks for which the current schedule does not meet the scheduling constraints are highlighted graphically.

Precedence View This view shows all explicit precedence constraints defined for the local task set. The set of constraints can be edited by the means of a special dialog box.

4.5 Generation of the TADL

The TADL generator is a front-end tool that extracts the scheduling information from the local design data-base and generates C-code files that describe the static TADL data. For every task a framework is generated that provides the task with it's input messages and that fetches the output messages. In the source file of the tasks code, a header with a comment is generated for every task, informing the programmer about the input and output messages designed for the task.

4.6 Discussion

The decision for the presented methodology is based on experiences made at our department with approaches on fully automated scheduling tools. A fully

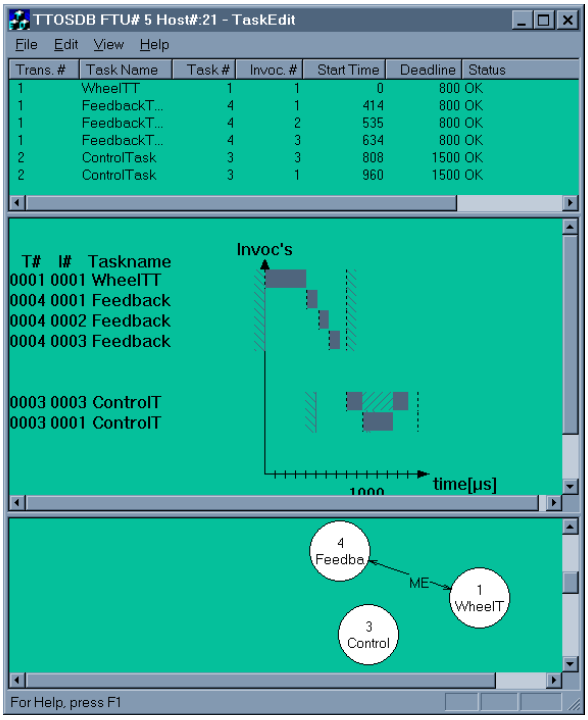

Figure 7: Screen shot of Task Schedule Editor.

automated scheduling tool requires an exhaustive description of the desired system properties, otherwise the scheduler delivers unwanted results. The turnaround times for highly sophisticated scheduling tools are rather poor and finally, the efforts (cost) required to build a sophisticated scheduling tools are significantly higher compared to our approach.

5 Conclusion

In this paper we presented a set of design tools for the development of distributed embedded control applications based on the Time-Triggered Architecture. The main focus of this paper lies on the design and implementation of a subsystem developed by one supplier. We propose a set of local design tools together with an incremental scheduling approach for the application developer to support the systematic and efficient development of application tasks on top of a time-triggered operating system.

The local design tools presented in this paper support an interactive scheduling process that gives the designer a clear insight in the intermediate results of the scheduling process and allows the designer to interact with the scheduling tool.

Our future research will focus on the optimization of the scheduling strategy and the optimization of the user interaction with the scheduler.

References

[HD92] K. Hoyme and K. Driscoll. SAFEbus (tm). In *Proceedings of the 11th AIAA/IEEE Digital Avionics Systems Conference*, pages 68–73, Seattle, WA, USA, Oct. 1992.

[Inc93] Aeronautical Radio Inc. Backplane Data Bus. ARINC specification 659, ARINC, Dec. 1993.

[KFG+93] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schütz, A. Vrchoticky, and R. Zainlinger. Real-time system development: The programming model of MARS. In *Proc. of the International Symposium on Autonomous Decentralized Systems*, pages 290–299, Kawasaki, Japan, Mar/Apr 1993.

[KG94] H. Kopetz and G. Grünsteidl. TTP — A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14–23, January 1994.

[KHK+97] H. Kopetz, R. Hexel, A. Krüger, D. Millinger, R. Nossal, A. Steininger, Ch. Temple, T. Führer, R. Pallierer, and M. Krug. A Prototype Implementation of a TTP/C Controller. In *Proc. of the SAE Congress 1997*, Detroit, MI, USA, Feb. 1997. Society of Automotive Engineers, SAE Press. SAE Paper No. 970296.

[Kop97] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publishers, 1997.

[PK89] P. Puschner and Ch. Koza. Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, 1(2):159–176, Sep. 1989.

[PS97a] P. Puschner and A. Schedl. Computing Maximum Task Execution Times — A Graph-Based Approach. *Real-Time Systems*, 13(1):67–91, July 1997.

[PS97b] P. Puschner and A. Schedl. Computing Maximum Task Execution Times — A Graph-Based Approach. *Real-Time Systems*, 13(1):67–91, July 1997.

[Pus98] P. Puschner. Worst-Case Execution Time Analysis at Low Cost. *Control Engineering Practice*, 6(1), Jan. 1998.

[SHS+97] C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, and C. Temple. Time-Triggered Architecture (TTA). In J.-Y. Roger, B. Stanford Smith, and P.T. Kidd, editors, *Proceedings of EMMSEC'97, Advances in Information Technologies: The Business Challenge*, pages 758–765. IOS Press, Nov. 1997. public project information is available at http://www.vmars.tuwien.ac.at/projects/tta/index.html.

[SSKF95] C. Scheidler, L. Schäfers, and O. Krämer-Fuhrmann. Software engineering for parallel systems: The TRAPPER approach. In *Proceedings of HICCS-28*, Hawaii, USA, January 1995. IEEE CS Press.