ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems, La Jolla, California, June 1995.

The Cluster Compiler – A Tool for the Design of Time-Triggered Real-Time Systems

Hermann Kopetz, Roman Nossal

Institut für Technische Informatik, Technical University of Vienna,
Treitlstr. 3/182/1, A-1040 Vienna, Austria
{hk,nossal}@vmars.tuwien.ac.at

May 12, 1995

Abstract

An off-line planning tool that supports the programmer in developing his real-time application is mandatory in the design of time-triggered real-time systems. This paper describes the architecture and the functions of such a tool, the Cluster Compiler, that is in development at our institute. We emphasize on the principle of a strict separation of the local from the global parts of a distributed system and on the consequences for the structure of the design tool arising from this principle.

Introduction

At present, real-time systems are often designed unsystematically. Conventional software modules are integrated by "real-time specialists" who tune the system parameters (e.g., task priorities, buffer sizes, etc.) during an extensive trial and error period, consuming more than 50% of a project's resources. Why the system performs its functions at the end is sometimes a miracle, even to the "real-time specialists".

To change this deplorable situation we need a proper real-time system architecture, a systematic design methodology, and a set of tools that support the system designer. In the last ten years we have focused our research efforts on the systematic design and developments of a particular class of real-time system architectures, the time-triggered real-time systems [Kop91]. Although we are aware that the time-triggered approach is limited to systems of regular behavior, there is a growing number of applications, e.g., in the domain of automotive electronics, that fit very well to this design paradigm.

In a time-triggered real-time system all computation and communications actions are triggered by the progression of a global time. The recurring global clock tick is the only event which may initiate an action in such a system. As the occurrence of this event is known a priori the system's actions can be planned off-line.

This off-line planning of communication and processing actions requires extensive tool support. We have realized this during the implementation of time-triggered applications on our MARS architecture. The most important tool support is required for finding the static schedules for the message transmission in the communication system and the task execution in the nodes of the distributed system, and for automatically generating the appropriate data structures for the control of the communication and the operating system in each node. In this paper we present such a planning tool that we call the Cluster Compiler.

The Cluster Compiler combines both aspects of scheduling, i.e., it integrates task and message scheduling into one off-line scheduling tool. As the scheduling problem is known to be NP-hard it is impossible to enumerate all solutions and to design good schedules manually. In the past we have focused on the heuristic search technique IDA* [Kor85] to find appropriate schedules [Foh94]. The cluster compiler is a flexible tool that will also implement other optimization techniques, like genetic algorithms [Hol75, Hou94, Sys91], simulated annealing [Kir83, Nan92, Che95] or tabu search [Glo93, Moo93, Por93].

The Cluster Compiler is the central tool for system design. It has to produce data structures that are consistent within all parts of a time-triggered system implementation, i.e., the protocol software in the communication controllers and the operating system in the nodes. As these data structures control the run-time system functions, they determine the behavior at the interfaces and thus put a number of requirements on the Cluster Compiler.

This paper is organized as follows. After the introduction we give an overview of the time-triggered system architecture that is the target for our Cluster Compiler. In the third section the functions of the Cluster

Compiler are explained. In section four we describe the structure of the Cluster Compiler. The paper concludes with section five.

A Time-Triggered Architecture

As an example for a time-triggered (TT) architecture we give a short overview of the architecture of MARS (MAintainable Real-time System) in this section. We start by discussing the architectural levels that have been introduced in MARS. We then continue with a description of the nodes and the communication system.

Architectural Levels

The architectural levels introduced in MARS are levels of abstraction. At the top of the hierarchy is the complete MARS system that includes the controlled object and the computer system.

System Level

A distributed real-time application can be decomposed into a set of communicating subsystems, called clusters. We distinguish between computational clusters and environmental clusters. In the example of Figure 1 the controlled object or the human operator are environmental clusters, whereas the distributed computer system can be decomposed into one or more computational clusters.

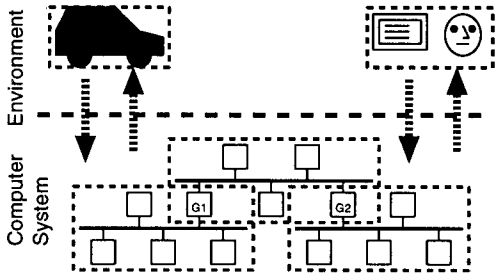

Figure 1: Overview of MARS

In MARS it is assumed that all computational clusters have access to a global timebase of known precision [Kop95b]. An intercluster gateway implements an abstraction function between the two clusters, i.e., only the information that is relevant for one cluster is passed across the interface from/to the other cluster.

Cluster Level

A cluster can be decomposed into a set of *fault-tolerant units (FTU)* that communicate with each other by the periodic exchange of TT messages (see Figure 2). The main concern at the cluster level is thus the correct and timely exchange of messages between the FTUs. There are, however, additional functions that are assigned to the cluster level: the synchronization of the clocks, and the membership service, i.e., the determination which FTUs are operational and which FTUs are faulty at a particular point in time.

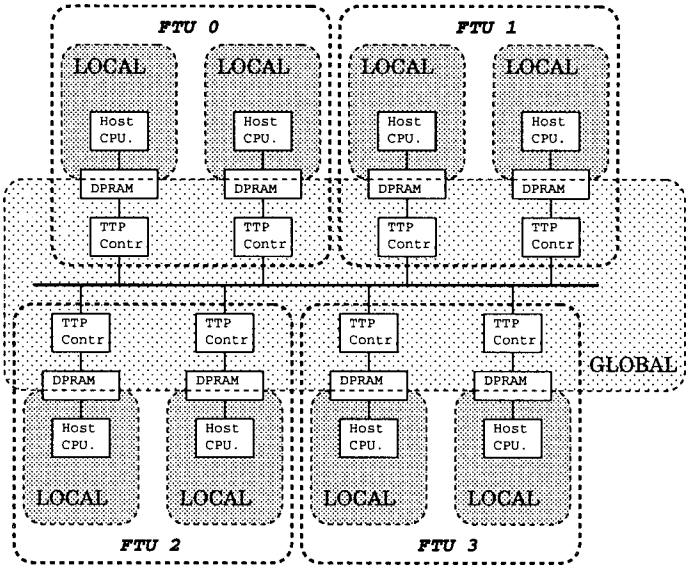

Figure 2: A MARS Cluster

Fault-Tolerant Unit / Node Level

A Fault Tolerant Unit (FTU) is a set of replica determinate fail-silent nodes that perform specified communication and computational functions within the given time constraints. As long as at least one of the nodes of an FTU is operational, the FTU as a whole is considered operational. The fact that an FTU can consist of more than one node is only relevant from the point of view of dependability. From the functional point of view, an FTU acts just like a single node.

A node consists of two parts that communicate via a node internal interface, the *message base interface (MBI)* [Kop95a]. One part of the node, the communication controller, implements the cluster communication and is logically part of the cluster level. The other part, the host CPU with the application software, implements the specified node local application. The MBI within a node is thus the dividing line between the cluster wide communication system and the node local processing of tasks. The MBI is a firewall that protects the local processing functions within a node from failures of the communication system and vice versa. The MBI thus also hides the communication between nodes from the application software.

A node is the smallest replaceable unit of this architecture. At any point in time, it is assumed that a node

is either operating correctly or it is silent. The implementation of a node must guarantee that the fail-silent abstraction is realistic, i.e., the node must have a very high error detection coverage.

The Node Architecture

Hardware Architecture

Figure 3 depicts the hardware structure of a MARS node. In the lower part of Figure 3 we see the communication controller with its own memory and with one or two bi-directional communication ports, connected to the (possibly replicated) broadcast channel, i.e., the cluster wide communication system. In the upper part we see the host processor with the interfaces to the local sensors and actuators. The host processor and the communication controller communicate via the Message Base Interface, which is implemented in the dual ported RAM, shown in the middle of the picture.

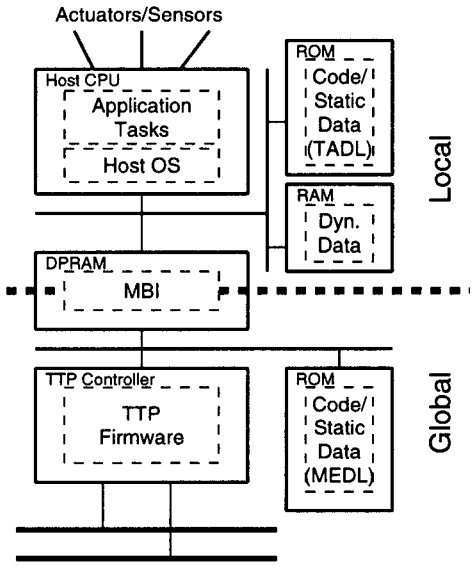

Figure 3: Hardware structure of a MARS node

The MBIs are the most important interfaces within a cluster. They separate node local activities that take place in the host processor from the cluster wide activities that take place in the communication controller. This interface is a strict data sharing interface. There is only one control signal, the periodic global clock tick, crossing this interface in the direction from the communication controller to the host processor. No control signal is crossing the MBI interface in the other direction.

The communication controller derives its control signals, i.e. when to send the next message, autonomously from the progression of the global time. It has access to a local data structure, the message descriptor list (MEDL), that contains the required information about the messages, e.g., what message has to be received at a particular point in time, where to locate the data in the DPRAM, when to send the next message, etc.. The Time-Triggered Protocol (TTP) that is implemented in the communication system is described in a later section of this paper.

Additionally, the hardware contains a number of mechanisms to increase the self-checking coverage of the nodes. The description of these mechanisms is beyond the scope of this paper.

The Application-Software Architecture

The application software is executed on the host processor of the node. The application software consists of a set of sequential tasks that read their input data (and their previous internal state), execute the specified data transformation and output the results (and the new internal state). In a TT system the synchronization requirements of the tasks (e.g. mutual exclusion or precedence) are considered during the generation of the static task schedules. It is therefore not necessary to allow internal synchronization points (e.g. a wait statement) within a task. The maximum execution time of a task can thus be determined without considering the other tasks in the node.

The task execution is controlled by a node local TT operating system. The control information for this operating system, i.e., when to execute a particular task, is contained in a static data structure, the task descriptor list (TADL) that has to be generated by the Cluster Compiler at compile time.

The Communication Protocol TTP

The exchange of messages within a cluster is controlled by the time-triggered protocol (TTP) [Kop94] that provides all services needed for the implementation of fault-tolerant hard real-time systems in an integrated manner. TTP is executed by the cooperating set of communication controllers within a cluster.

The TTP Services

TTP provides the following services:

- Guaranteed real-time response for all messages
- Fault tolerant clock synchronization
- Distributed membership service as the basis for atomic broadcast
- Temporal encapsulation of the nodes to support a constructive design and test methodology.
- Rapid fault detection at the sender and receiver and an end-to-end error detection mechanisms to increase the selfchecking coverage of the nodes.

- Consistent mode change to realize a data dependent change in the temporal task structure (e.g., a fast switch over to an emergency mode).
- Transient blackout management to bring the protocol into a defined state in case the fault hypothesis is violated.
- Support for the implementation of fault-tolerant systems consisting of duplex channel and replicated nodes.
- Distributed redundancy management

These services are implemented without a central master.

Principle of Operation

TTP controls the medium access by a synchronous time division access method (TDMA) derived from a fault-tolerant global time base that is established by the protocol itself. The protocol provides a temporal encapsulation of the nodes, i.e., it is not possible that an increased communication demand by one node affects the temporal behavior of another node. This property of the protocol makes it possible to constructively build and test complex TTP systems.

The semantics of the messages transported by TTP corresponds to the state message semantics, i.e., a new version of a message overwrites the previous version and a message is not consumed on reading. This message semantics is well suited to handle the transport of state variables in control applications. State message semantics provides an implicit flow control and eliminates the delicate problem of dynamic buffer management.

TTP supports different operational modes [Jah88, Jah94], i.e., task and message sets, within the system. Each of these modes reflects a certain state or phase of the application. For example consider an airplane. A flight consists of three phases, starting, normal flight and landing. For each flight phase different activities and thus messages are necessary in the computer system, therefore each of the phases is modeled by one mode. TTP also provides a protocol service that can switch between different operational modes dynamically. For every mode up to seven (relative) successor modes can be defined during design time. Whenever the TDMA slot of a node has arrived, the node can request a mode switch to one of these successor modes by setting the appropriate field in the message header. All nodes will switch to this successor mode consistently after the receipt of this message. The latency time for a mode switch is, at worst, a full TDMA cycle.

TTP contains mechanisms that support the rapid and consistent detection of the loss of messages or the failures of nodes. The most prominent of these mechanisms is an integrated membership protocol. This membership protocol is based on the a priori known send times of the TDMA protocol, the known points of arrival of each message and the acknowledgment bits contained in each message header. The membership protocol reports with a latency of one TDMA round which node is active and which node is inactive. The membership protocol is the core of many other TTP services: the clock synchronization service, the consistent mode change service, the redundancy management service, the atomic broadcast service, and the blackout monitoring service.

The system level fault model of the architecture assumes that nodes exhibit fail-silent failures only, i.e., they either produce correct messages or no messages at all. To support the implementation of fail-silent nodes TTP provides a "High Error Detection Coverage" (HEDC) mode of operation. In the HEDC mode an application task can append an end-to-end signature to each data field. This end-to-end CRC will be checked by the TTP hardware controller and by the receiving task. In the HEDC mode the complete path between the sending task and the receiving task is protected by a signature. Fault injection experiments [Kar95] have shown that this is an effective mechanism to implement a very high error detection coverage on standard off the shelf hardware.

The high data efficiency of TTP is achieved by taking advantage of the regularity and a priori knowledge contained in a TDMA protocol. Since it is known in advance at what point in time a node will send a message, there is no need to carry the message name or the sender name in the message, since send and receive times are common knowledge.

TTP Message Formats

It is crucial for the operation of TTP that sender and receiver never disagree about the protocol controller state (the C-state) during the exchange of normal messages. The C-state consists of the global time, the operational mode and the current membership information. The membership information is represented in a bit vector with the length of the number of nodes. TTP enforces C-state agreement by a special mechanism for the calculation of the CRC field of normal messages as is explained below.

A TTP message consists of three fields, a one byte header, a data field of up to 16 bytes and a one byte CRC. In TTP two message types are distinguished by the first bit of the header, normal messages and initialization messages.

For normal messages the CRC is calculated over the Header, the Data Field and the local C- state of the sender, as shown in Figure 4.

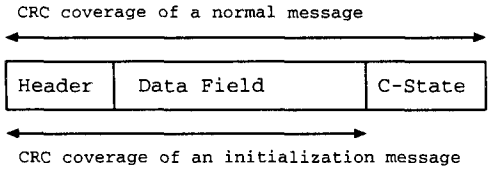

Figure 4: CRC Calculation

The receiver compares the CRC that is calculated locally over the received header, datafield, and its local C-state with the CRC contained in the last byte of the received message. If these two CRCs are different, then either the message has been mutilated during the transmission or the sender has a different C-state of that of the receiver. In both cases the message is discarded and – provided there is no correct redundant message available – the sender is eliminated from the list of active members.

An initialization message (I-message) contains in its data field the C-state of the sender and has a "normal" CRC calculated over the message contents only. Initialization messages are needed at system startup and to reintegrate repaired nodes into an ensemble. It is good practice to have at least two different nodes that send I-messages periodically (possibly with a long period).

Function of the Cluster Compiler

The Cluster Compiler is the off-line part of a MARS system. The application programmer provides his/her application specific knowledge to the compiler either directly or by using a higher level design tool. The compiler then creates the control information which is needed for protocol and task execution. A guiding principle during the development of the Cluster Compiler is that the programmer should be able to develop an application with minimal concern about the communication pattern. As long as a temporally accurate version of the data is available at the MBI, the application will perform as intended.

The function of the Cluster Compiler as the central tool in system design is determined by the other parts of a MARS system, i.e., the protocol, the communication controller, the operating system and the application. Each of these parts assumes that certain problems are solved by the Cluster Compiler off-line, so that all mechanisms that would be needed for an on-line problem solution can be eliminated.

The Cluster Compiler also reflects the main design principle of a TT system, the separation of local and global concerns. These two main concerns are on the one hand communication planning, which is a global subject, and on the other hand task scheduling, which is primarily of local interest to the application in a node. In addition to these two main tasks a number of more specific, smaller tasks are performed.

Communication Planning

Communication planning is a global issue that has to be performed simultaneously for all nodes of a cluster. It consists of two steps that are coupled tightly to each other, *data element allocation* and *message scheduling.* TTP messages correspond to the common notion of a message. They are global data structures that are exchanged via the communication medium. The contents of TTP messages are sets of data elements.

Data elements are the input/output data items consumed and produced by an application task within a node. They only exist in the local part of a node. During the development of an application the programmer just specifies the data elements, he never deals with TTP messages. Every data element has a specified validity time that depends on the dynamics of the real-time state-variable in the environment this data element is representing. If the validity time of a data element is longer than the longest time interval between the point of observation of the real-time state-variable and the point of use of the corresponding data element, we call a data element *phase insensitive.* An application task receiving a phase insensitive data element can always assume that there is a temporally accurate version of the data available at the MBI. If the above cited condition is not satisfied, then the data element is *phase sensitive.* Tasks receiving data elements of this type must be synchronized with the sending task and thus with the message schedule and/or a state estimation task must be executed at the receiver.

The data elements produced by the various application tasks are combined to TTP messages during data element allocation, considering the temporal validity of the data elements. In the following step these messages are scheduled, i.e., for each mode a period called the *cluster cycle* is designed. A cluster cycle consists of at least one TDMA cycle. In each TDMA cycle exactly one sending slot is assigned to each FTU.

This planning step is performed under a number of constraints:

- The same sequence of sending FTUs must be used in every TDMA cycle of a cluster cycle.
- The bandwidth of the communication medium must not be exceeded.
- Each data element must be scheduled observing its maximum latency as it is specified by the programmer in the application definition.

Apart from these apparent limitations other requirements have to be fulfilled:

In each cluster cycle there must be at least two nodes that send an I-frame containing their C- state periodically. The periodical diffusion of this state, that must be identical for all nodes of a cluster, is needed for reintegration of failed nodes.

When planning the mode changes the Cluster Compiler has to ensure that the order of nodes of the old mode is maintained for at least two slots. This is important because otherwise the acknowledgment scheme for received messages could fail.

There are many more requirements and limitations that reduce the degree of freedom during communication planning. The above list is by no means complete but is intended to merely give the reader an idea of some of the issues that have to be observed and why it is very difficult to do the planning by hand.

Task Scheduling

Task scheduling is required in every real-time system. There are two approaches to task scheduling: the dynamic approach where the schedules are calculated online on the basis of the actual resource requirements and the static approach where the schedules are determined off-line on the basis of the maximum resource requirements of a task [Foh94].

In our static approach that maintains the strict separation of global and local concerns task scheduling belongs to the local side. For this reason the task schedule for every node has to fit the global framework given by communication planning. Apart from the given communication framework the application specification is the basis for task scheduling. This specification must contain details on the data elements (maximum latency, etc.), details on the tasks (deadline, period etc.), and the required relationship between the tasks, the task precedence graph of the application.

The following constraints have to be observed during task scheduling:

- All tasks must be scheduled so that they meet their deadlines.
- The schedule created by the Cluster Compiler has to be consistent with the specified task precedence graph.
- Data access to the DPRAM must be synchronized with the communication controller. It has to be assured that neither read/write nor write/write conflicts between the host CPU and the TTP controller may arise.
- The Cluster Compiler has to decide which type of mode change is supported (immediate mode change, completion of all activities or completion of some of the current activities). For more details on the semantics of mode changes the reader is referred to [Jah88].

Multicluster Systems

In multicluster systems two additional tasks have to be supported by the cluster compiler, system configuration and gateway design.

System configuration is concerned with the design of the communication in a multicluster system. The planning tool is provided with the number of nodes and the data elements that have to be passed between them. Based on this information the tool chooses the appropriate number of busses, assigns the nodes to them and designs their connection via gateways. The main issue of system configuration is on the one hand the minimization of hardware cost determined by the number and length of the necessary cables and by the number of gateway nodes. On the other hand the tool must minimize the transmission delay of data elements sent from one cluster to another.

In a multicluster system messages passed between clusters have to be scheduled such that the overall latency of the information transfer is minimized, i.e., the Cluster Compiler must create systemwide communication schedules. In systems with unsynchronized clusters only a maximum delay can be guaranteed whereas in systems where clusters are synchronized and the communication is controlled by time-triggered protocols, a latency guarantee for each message can be given.

Structure of the Cluster Compiler

This section describes the structure of the Cluster Compiler and explains its parts and the techniques that will be used.

Heuristic Optimization

The scheduling problem that has been described in the previous section is known to be NP-complete. It is therefore impossible to enumerate all solutions in order to find the best. To solve a problem of this type one has to rely on heuristic techniques. For the scheduling tool for former versions of the MARS system we have used the heuristic search strategy IDA* [Foh94, Kor85]. The tool for the current version, the Cluster Compiler, will be based on heuristic optimization strategies as well. Besides IDA* we intend to evaluate genetic algorithms, simulated annealing and tabu search.

For each of the five steps of the Cluster Compiler, shown in Figure 5, the main goals of the optimization procedure are given in this section. These goals are

included into the objective function of the respective procedure.

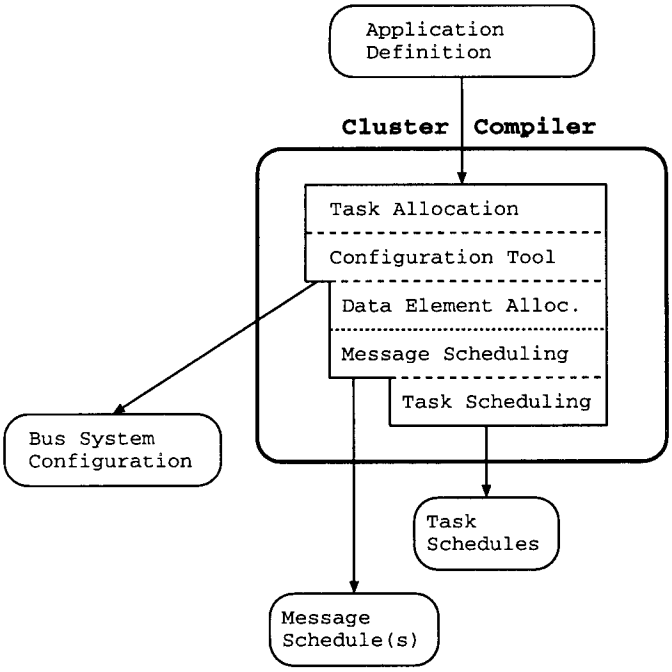

Figure 5: The structure of the Cluster Compiler

Interfaces

A database forms both, the internal interfaces as well as the external interfaces of the Cluster Compiler, i.e., the interfaces between its parts and the interfaces to the application programmer and the hardware. For the latter a translation tool that transforms the output into the form needed by the hardware will be developed. The advantages of this approach are twofold. The Cluster Compiler is not concerned with creating a certain output format and it is easier to adapt the output for different purposes, e.g., minimal memory demand, and for different target systems.

Functional Units

Task Allocation The first part of the Cluster Compiler performs the task allocation, i.e., it assigns the tasks of the application to the nodes of the system. The objective function of this step is aimed at equal load distribution among the nodes and minimization of internode communication.

In many applications this assignment is restricted by an at least partially fixed allocation. A task reading and evaluating sensor data, for example, will mostly reside on the node the sensor is connected to.

The input for this step and thus for the whole Cluster Compiler is the *application definition*. It consists of the list of data elements, the task definitions and the relations between the tasks. For each task its period and its maximum execution time are specified. In the list of data elements the size, the temporal validity, the sending and the receiving tasks of each data element are defined. Concerning task relations we distinguish three types, mutual exclusion, simple precedence and data element transmission, i.e., precedence combined with data transfer. These relations are specified in the task precedence graph.

Configuration Based on the previously fixed task allocation the configuration part chooses the appropriate number of busses, assigns the nodes to them, and connects the busses via gateways. The optimization procedure of this step is dedicated to the minimization of the overall message transmission delay and the number of busses and gateways. Moreover it takes into consideration so called locality constraints, which can be specified by the application programmer. For instance these constraints force that a set of nodes has to be assigned to a certain bus or that a fixed number of busses and gateways has to be used.

Data Element Allocation Within the third step, data element allocation, data elements produced by the various tasks are combined to TTP messages and their size and period is calculated. The constraints that apply to this step have been mentioned in the previous section.

Message Scheduling Message scheduling is coupled tightly to the previous part, in fact the separation into two logical steps is for reasons of clearness only. This scheduler assembles single TTP messages to cluster cycles and creates the control information that is needed for protocol execution.

The objective function for the steps data element allocation and message scheduling considers the constraints associated to data elements, especially the maximum latency, as penalty terms. If a constraint is not met by a solution, the "cost" of this solution is increased by the corresponding penalty term, thus making it unlikely that the solution is taken into consideration any further. The main goal implemented by the objective function is on the one hand the minimization of the required bandwidth on the communication medium and on the other hand the optimal utilization of this bandwidth requirement.

This step finishes global planning. The following step creates only local control information for each node.

Task Scheduling During the last step the Cluster Compiler performs task scheduling and creates the control information representing the task schedule for the host processor's operating system at each node. It is

based on the knowledge of the communication system, which has been defined in the previous steps. The message schedule establishes the boundary conditions for task scheduling, in the sense that the earliest release time of a task must be later than the latest arrival time of a message received by the task and that its deadline must be prior to the earliest sending time of a message sent by this task. In the section on task scheduling we have mentioned communication planning being a global issue as a reason for doing this planning step first. Yet there is a second reason: As our communication protocol is based on a strict TDMA scheme the degree of freedom is much higher in task scheduling than in message scheduling. Doing the task scheduling first would further reduce the solution space for message scheduling.

The restrictions to task scheduling mentioned above apply only to tasks that process phase sensitive data. Tasks acting on phase insensitive data solely can be scheduled without synchronization to the message schedule. The state message concept ensures that there is always valid data available at the MBI. Thus task execution for phase insensitive data is completely decoupled from the timing of any communication action.

The objective function of task scheduling forces only the fulfilling of the constraints put upon a task, i.e., its period, its relations and for tasks processing phase sensitive data the boundary conditions determined by the message schedule, by introducing penalty terms again. There is no parameter that has to be optimized explicitly, not even the overall schedule length, as this length is predetermined by the length of the cluster cycle that is designed during communication planning. Optimizing the overall schedule length or maximizing the processor idle time within the predetermined cluster cycle length makes sense only if spare capacities for sporadic tasks have to be saved. This approach will be used in a later version of the new MARS system.

Support of Different Planning Issues

The Cluster Compiler must support two different types of planning, overall planning on the one hand and partial (re)planning on the other. In overall planning the system and its control structures are designed throughout all the steps starting from the application definition. Partial (re)planning means doing only some steps of the planning process, i.e., starting somewhere in the middle, skipping the earlier steps.

To cope with these two demands two versions of the Cluster Compiler will be implemented, one integrated tool for overall planning and a single-step tool with clear interfaces between the steps for partial (re)planning.

Single-Step Tool

This version of the Cluster Compiler consists of clearly separated parts that communicate with each other via database interfaces. Each step writes its output to a database that serves as input for the succeeding step. Thus it is possible to start the planning at any step.

The single-step Cluster Compiler will be used mainly to perform local changes without influencing the global parts of the system. If for instance a task on one node changes there is no need to redesign the bus system layout and the message schedules. Only the task schedule on this node has to be recalculated.

Integrated Tool

One main goal of our research is a comprehensive integration of the main steps described above. As a "local" optimization within each function results in a suboptimal global solution an integrated optimization for all design steps must be achieved.

If planning is done in an integrated manner, it is no longer possible to distinguish the parts of the Cluster Compiler. To be able to redesign parts of the system using the single-step tool the internal interface databases have to be filled. This is done by a special tool that creates the database entries out of the Cluster Compiler's final output.

Conclusion

We have presented an architecture for a distributed time-triggered real-time system and have given an introduction to the communication protocol TTP that is used by the communication system of our architecture. This protocol provides all services needed for the implementation of fault-tolerant hard real-time systems.

We have focused on our main design guideline, the strict separation of local and global issues. This principle is realized in the hardware of each node and supported by the design system, the Cluster Compiler. We have shown the issues and concepts for this off-line design tool, especially how our main principle will be implemented in the tool.

The functions of the Cluster Compiler, i.e., message and task scheduling, and the requirements put upon these functions by the other parts of the system have been described. Based on this we have introduced a clear design structure.

The next steps in our research will be the evaluation of heuristic optimization algorithms and the definition of the interfaces between the Cluster Compiler and the remaining MARS system. As argued these interfaces determine the function of the Cluster Compiler, because they specify its input data and its required outputs. For this reason they are the basis for all future work

concerning the Cluster Compiler and the other parts of the system.

Acknowledgement

This work has been supported in part by the ESPRIT BRA PDCS 2.

References

[Che95] Sheng-Tzong Cheng and Ashok K. Agrawala. Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints. Technical Report CS-TR-3402, University of Maryland Institute for Advanced Computer Studies, Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742, USA, January 1995.

[Foh94] G. Fohler. *Flexibility in Statically Scheduled Real-Time Systems.* PhD Thesis, Technisch-Naturwissenschaftliche Fakultät, Technische Universität Wien, Wien, Österreich, 1994.

[Glo93] F. Glover, E. Taillard, and D. de Werra. User's Guide to Tabu Search. *Annals of Operations Research*, 41, 1993.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, Mass., USA, 1975.

[Hou94] E.S.H. Hou, N. Ansari, and H. Ren. A Genetic Algorithm for Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2), February 1994.

[Jah88] F. Jahanian, R. Lee, and A. Mok. Semantics of Modechart in Real Time Logic. In *Proc. of the 21st Hawaii International Conference on Systems Sciences*, pages 479–489, Jan. 1988.

[Jah94] F. Jahanian and A.K. Mok. Modechart: A Specification Language for Real-Time Systems. *IEEE Transactions on Software Engineering*, 20(12):933–947, Dec. 1994.

[Kar95] J. Karlsson, P. Folkesson, Jean Arlat, Yves Crouzet, and Günther Leber. Integration and Comparison of Three Physical Fault Injection Techniques. In *Predictably Dependable Computing Systems*, chapter V: Fault Injection, pages 309 – 329. Springer Verlag, 1995.

[Kir83] S. Kirkpatrick, C. Gelatt, and M. Veechi. Optimization by Simulated Annealing. *Science*, 220:671–680, May 1983.

[Kop91] H. Kopetz. Event-Triggered versus Time-Triggered Real-Time Systems. In A. Karshmer and J. Nehmer, Editors, *Proc. Int. Workshop on Operating Systems of the 90s and Beyond*, Lecture Notes in Computer Science, Volume 563, pages 87–101, Berlin, Germany, 1991. Springer-Verlag.

[Kop94] H. Kopetz and G. Grünsteidl. TTP — A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14–23, January 1994.

[Kop95a] H. Kopetz, M. Braun, C. Ebner, A. Krüger, D. Millinger, R. Nossal, and A. Schedl. The Design of Large Real-Time Systems: The Time-Triggered Approach. Research Report 14/95, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, May 1995.

[Kop95b] H. Kopetz, A. Krüger, D. Millinger, and A. Schedl. A Synchronization Strategy for a Time-Triggered Multicluster Real-Time System. Research Report 4/95, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, Feb. 1995. Accepted for publication at the 14th Symp. on Reliable Distributed Systems.

[Kor85] R. Korf. Depth-First Iterative-Deepening: An Optimal Admissable Tree Search. *Artificial Intelligence*, 27(3):97–109, 1985.

[Moo93] E.L. Mooney and R.L. Rardin. Tabu Search for a Class of Scheduling Problems. *Annals of Operations Research*, 41, 1993.

[Nan92] A.K. Nanda, D. DeGroot, and D.L. Stenger. Scheduling Directed Task Graphs on Multiprocessors using Simulated Annealing. In *Proc. of the 12th Int. Conference on Distributed Computing Systems*, 1992.

[Por93] S.C.S. Porto and C.S. Ribeiro. A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints. Technical Report PUCRioInf-MCC03/93, Department of Computer Sciences, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 1993.

[Sys91] G. Syswerda. Schedule Optimization using Genetic Algorithms. In L. Davis, Editor, *Handbook of Genetic Algorithms*, New York, USA, 1991. Van Nostrand Reinhold.