Developing Software with Predictable Timing Behavior

Peter Puschner
Ralph Zainlinger

Institut für Technische Informatik
Technische Universität Wien
Austria

hk@vmars.tuwien.ac.at

Research Report Nr. 5/90

February 1990

Published in Proc. of the 7th IEEE Workshop on
Real-Time Operating Systems and Software,
May 1990, Charlottesville, VA

Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3/182/1
A-1040 Wien
Austria

Phone: (43) 222-58801-8181
Fax: (43) 222-569149

Developing Software with Predictable Timing Behavior

Peter Puschner
Ralph Zainlinger

Institut für Technische Informatik
Technische Universität Wien
Treitlstr. 3/182
A-1040 Vienna, Austria

E-mail:
peter@vmars.UUCP
ralph@vmars.UUCP

April 12, 1991

Abstract

In critical real-time systems knowledge about the maximum execution times (MAXT) of tasks is of utmost importance. However, this knowledge is merely the first step towards building functioning systems that definitely meet their deadlines. Computing the MAXT of a task has to be an integral part of the application development and has to be directly incorporated into the implementation process.

This paper presents the concepts for an integrated programming environment that particularly supports the development of real-time software with predictable timing behavior.

We are grateful to Andreas Damm, Gustav Pospischil, Werner Schütz, and Alexander Vrchoticky for helpful discussions and comments on this report.

This work has been supported in part by the ESPRIT Basic Research Project 3092 "Predictably Dependable Computing Systems (PDCS)".

1 Introduction

Most computer systems for real-time process control must meet high standards of reliability, availability, and safety. In many of these real-time applications, the cost of a catastrophic system failure can far exceed the initial investment in the computer and the controlled object. To prevent such failures, system design must guarantee behavior as specified in the domains of both value *and* time during all anticipated operational situations.

One of the key aspects in designing predictable real-time systems is exact knowledge about the timing behavior of the real-time tasks involved. In order to guarantee that a real-time task can be finished before the expiration of its specified deadline, the maximum execution time (MAXT) or a reasonable upper bound have to be known in advance, i.e. before run time.

[Kli86,Pus89,Sha89,Sto87] present concepts on how MAXT can be calculated and which restrictions have to be imposed on programming languages for critical real-time systems (e.g. bounded loops).

This report suggests how these concepts could be effectively integrated into the design process of a real-time system. The major idea is that the MAXT analysis forms an integral part of the programming environment in order to provide the programmer with concise information about the maximum execution times of arbitrary pieces of code.

The proposed programming environment is conceived as part of an already existing prototype design environment for real-time system, the MARS Design System MARDS [Sen88,Sen89]. However, the concepts are not restricted to this particular design environment.

The structure of this report is as follows. Section 2 provides background information about the design principles applied in MARDS. Section 3 lists some basic requirements for the proposed programming environment, which is described in Section 4.

2 Background and Motivation

MARDS has been designed to support the development of critical real-time applications according to the MARS architecture [Kop89]. MARS (MAintainable Real-time System) is a periodic, time driven distributed real-time system architecture. A MARS system consists of a number of clusters each of which is built of a set of components interconnected by a real-time bus. Each component executes a set of tasks. Communication between tasks is realized exclusively through messages.

During design the overall system is gradually refined, first into a set of clusters (set

of components characterized by a high inner connectivity), then each cluster independently of the others into components. Concurrently, the functional behavior of the system is modeled through *real-time transactions*. A real-time transaction refers to the execution of a set of correlated actions between a stimulus from the environment and the corresponding (time-constrained) response indicating the completion of this set of actions. During system design transactions are refined into subtransactions which finally become tasks.

It is a key concept of the design system and thus the underlying design methodology that time forms an integral part and does not constitute an isolated addendum. Transactions, for example, are characterized by various time attributes, e.g. MART, the maximum response time which is actually dictated by the environment, and MINT, the minimal interval between two invocations of the same transaction. At the task level MAXT determines a crucial characteristic. Before coding activities take place, MAXT is estimated for each task (by an experienced designer) in order to verify whether an appropriate pre run-time schedule can be found for the designed software or not. If the scheduling problem can be solved the tasks are coded respecting the estimated MAXTs.

3 Requirements for the Programming Environment

The following is a list of basic requirements for the proposed programming environment.

- **General Applicability:** Two fundamentally distinct approaches towards critical real-time task development are conceivable: One starting with a predetermined MAXT, which has to be met by the implementation (as a result of a consequent top down refinement), the other starting with the coding activities (bottom up) not respecting any given timing constraints during implementation. While the second strategy only demands calculating and adequately representing MAXT, the first approach requires more sophisticated mechanisms for comparing and controlling the desired and the actual MAXT as well as support for improving unsatisfactory results. The programming environment should support both approaches.
- **Timing Information:** During program development the programmer must have access to the following (time related) information at any point in time.
  - How long does the execution of this section of the program take ?
  - How long does the execution of the overall program take ?
  - Am I adhering to the demanded MAXT ?

  - If not, what is the time difference ?

- **Traceability:** Traceability in this context refers to the global consequences of changes in a particular (local) section of the program. Questions such as "What will happen to the global MAXT if I am able to reduce MAXT of this particular block" are likely to occur and have to be answered by the programming environment.

- **Modelability:** Modelability is closely related to traceability. Modelability in our definition comprises support for

  - comparing different combinations of implementation alternatives and
  - experimenting with hypothetical time changes in different program sections.

- **Turn Around Cycles:** The time it takes to compile and evaluate a program (in the time domain) has to be kept as short as possible. It is unacceptable to pass the code to a separate tool to calculate MAXT. All components (editor, compiler, MAXT analyzer) have to be integrated into a common framework.

- **User Interface:** A highly sophisticated (graphical) user-interface is required to appropriately handle and present the various timing information.

4 The Programming Environment

The requirements listed above are the basis for the programming environment. The programming environment consists of three cooperating parts, 1) an editor which incorporates a text editor and a so-called time editor, 2) a compiler and 3) a timing analysis tool. The components of the environment exchange timing information via a *timing tree* which represents the structure of the program or program fragment being developed. At each point in time the timing tree contains a certain amount of timing information. The amount of timing information present in the timing tree depends on which part of the environment has produced or changed the tree information.

Figure 1 shows the programming environment. The tools necessary for handling the timing information are presented in the grey area. The programmer implements his programs/tasks with the syntax directed text editor and passes them to the compiler.

The compiler generates the object code as well as the timing tree. The timing tree is a simplified program syntax tree augmented with timing information. Its nodes represent the sequential parts, branches, bounded loops, markers and scopes (the latter have been introduced in [Pus89]) of programs.

The timing information produced by the compiler contains only the maximum execution times of sequential (basic) blocks, i.e. the compiler fills the leaves of the tree

Figure 1: programming environment for time guaranteed tasks

with timing information. The timing tree is forwarded to the MAXT analyzer which fills the inner nodes with the values calculated for the maximum execution time of the respective program parts represented by this subtree. Thus, the maximum execution times for all branches, loops, scopes and the whole piece of source code analyzed are available in the timing tree.

The *time editor* reads the timing tree after it has been completed by the MAXT analyzer. The time editor transforms the time information contained in this tree into a Nassi-Shneiderman diagram which is displayed in parallel with the program code. Every structural unit of the diagram provides a display field for the actual MAXT of the matching program part and one or more display fields for hypothetic timing values.

Time Editing

If the user is not satisfied with the timing behavior of his program he can set hypothetic execution times at every level of the Nassi-Shneiderman diagram and have the MAXT calculated under the assumption of the hypothetic values. In this way he can model the timing behavior of potential improvements and gets an immediate feedback about

its gains before actually changing any piece of code.

If a simulated value is input at a specific level in the Nassi-Shneiderman diagram both actual and hypothetic time values of the abstraction levels below are ignored in the MAXT calculation. A change at one specific level influences all simulated timing values of the surrounding constructs. Hypothetic timing values which have been set by the programmer are marked to inform the programmer that these values are not derived from analyzing actual program statements.

Code Variants

Often the programmer has many possibilities for writing tasks with the same functionality. It might not always be obvious which variant needs the minimal amount of time to execute. For this reason the environment supports multiple windows in which different versions of program parts can be edited and evaluated. The programmer may code different variants of some program parts or even implement different algorithms, calculate their maximum execution time and check which variant is suited best. The least time consuming one may then be used in the application being developed.

5 Summary and Conclusion

In this report we have discussed the requirements and concepts for a real-time task programming environment. When developing software with predictable timing behavior it is a waste of efforts to write programs and find out afterwards that they violate the specified timing behavior. Therefore, the timing behavior has to be a central issue throughout the whole task development process.

We have proposed a way to include timing behavior in the task design. Timing information is available during the whole implementation process. In every phase, the task's timing behavior can be calculated and modeled, thus allowing the programmer to test whether the specified requirements can be met. In addition, the programming environment allows the user to generate different variants for a piece of source code to find the implementation with the shortest MAXT.

References

[Kli86] E. Kligerman and A. Stoyenko. Real-Time Euclid: A Language for Reliable Real-Time Systems. *IEEE Transactions on Software Engineering*, SE-12(9):941–949, Sep. 1986.

[Kop89] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft, and R. Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[Pus89] P. Puschner and Ch. Koza. Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, 1(2):159–176, Sep. 1989.

[Sen88] Ch. Senft. A Computer-Aided Design Environment for Distributed Realtime Systems. In *IEEE CompEuro 88, System Design: Concepts, Methods and Tools*, pages 288–297, Brussels, Belgium, Apr. 1988.

[Sen89] Ch. Senft and R. Zainlinger. A Graphical Design Environment for Distributed Real-Time Systems. In *Proc. 22nd Annual Hawaii International Conference on System Sciences, Vol. II*, pages 871–880, Kailua-Kona, HI, USA, Jan. 1989.

[Sha89] A. C. Shaw. Reasoning About Time in Higher-Level Language Software. *IEEE Transactions on Software Engineering*, SE-15(7):875–889, July 1989.

[Sto87] A. Stoyenko. *A Real-Time Language With A Schedulability Analyzer.* PhD Thesis, Computer Systems Research Institute, University of Toronto, Dec. 1987. Technical Report CSRI-206.