

R E A L

T I M E

S Y S T E M

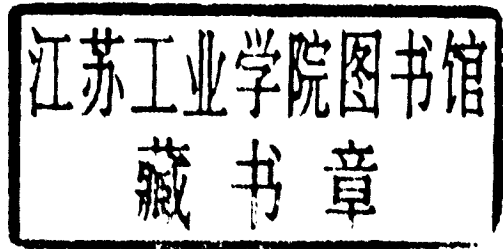
D E S I G N

SHEM - TOV LEVI
ASHOK K. AGRAWALA

REAL-TIME SYSTEM DESIGN

Shem-Tov Levi

Ashok K. Agrawala



McGraw-Hill Publishing Company

New York St. Louis San Francisco Auckland Bogotá Caracas Hamburg
Lisbon London Madrid Mexico Milan Montreal New Delhi Oklahoma City Paris
San Juan São Paulo Singapore Sydney Tokyo Toronto

This book was set in Times Roman by Publication Services, Inc.
The editor was David M. Shapiro;
the production supervisor was Friederich W. Schulte.
The cover was designed by John Hite.
Project supervision was done by Publication Services, Inc.
R. R. Donnelley & Sons Company was printer and binder.

REAL-TIME SYSTEM DESIGN

Copyright © 1990 by McGraw-Hill, Inc. All rights reserved.
Printed in the United States of America. Except as permitted under the
United States Copyright Act of 1976, no part of this publication may be
reproduced or distributed in any form or by any means, or stored in a data
base or retrieval system, without the prior written permission of the
publisher.

1 2 3 4 5 6 7 8 9 0 DOC DOC 9 5 4 3 2 1 0

ISBN 0-07-037491-0

Library of Congress Catalog Card Number: 90-60025.

LIST OF FIGURES

1.1	Embedded computer system	2
1.2	Computation interactions with operating system and environment	6
1.3	System's layers of objects	6
2.1	Point-based versus internal-based representation	12
2.2	Master-slave synchronization principle	17
2.3	Breakdown of communication delay in Tempo	21
2.4	Comparison of synchronization algorithms	27
2.5	Resynchronization example	31
3.1	Justifier-justification-justificand relations	36
3.2	Object's owner and user justification	36
3.3	Example: different contexts sharing an object	38
3.4	Relations between different types of objects	41
3.5	Recovery block versus modular redundancy	43
3.6	Interrupt service without preemption	46
3.7	"First-deadline" scheduling with preemption	47
3.8	Local versus remote servers	51
3.9	Objects involved in communication	51
3.10	Timing diagram for local and remote nodes	52
4.1	Convex interval binary relations	58
4.2	Object's owner and user temporal justification	63
4.3	A calendar expressed in C	64
4.4	Uncertainty of time	65
4.5	Variance of duration	66
4.6	Time constraint laxity "window"	66
4.7	Constraint propagation using global-time terms	69
4.8	Time constraint propagation	69
5.1	Life cycle of an ideal real-time system	76

5.2	Statechart example	80
5.3	Clustering states in statecharts	80
5.4	Zooming in statecharts	81
5.5	States orthogonality in AND decomposition	91
6.1	Hierarchical control structure	86
6.2	Information hiding concept	87
6.3	Computation graph model	90
7.1	Total and partial order conflicts	94
7.2	SPN example	96
7.3	Annotated Petri net example	99
7.4	Transition firing inhibit	100
7.5	Asynchronous subsystem	102
7.6	Independent cycle subsystem	103
8.1	PARC execution configuration	108
8.2	Paths example of weakest precondition solution	112
8.3	RTL constraint graph representation	114
8.4	Constraint graph example	114
8.5	Positive-cycle detection by node removals	115
9.1	Rendezvous in Ada typical remote call	126
10.1	Glass-box and black-box testing	133
10.2	Data-flow diagram example	140
10.3	R-net example	142
11.1	Time-driven scheduling examples	152
11.2	General time constraint in TDS	153
13.1	Verification of schedulability of TC_{in}	192
13.2	Laxity interaction of overlapping time constraints	193
13.3	Laxity computation example	194
13.4	Spreading requirement in preemptable constraint	195
14.1	Temporal (a,b) and physical (c,d) redundancy	220
14.2	Wrong use of resources: 0-resiliency	220
14.3	Forward wave of ALLOC_REQ messages	226
14.4	Backward wave of ALLOC_REP messages	227
15.1	Network interface unit (NIU)	239
15.2	Delay model for a LAN	241
15.3	Duration of semantic link	252
16.1	A single access server object	258
16.2	MARUTI: user's view of management	260
16.3	Interrupt handling and serving	262
17.1	Control transfer from on-line to off-line	269
17.2	Local versus remote relation	270
17.3	Typical joint variables in MARUTI	277
17.4	Design example: inserting time constraint	279
17.5	Design example: removing time constraint	280
17.6	Example design	282
17.7	Example design	283

Computers have been used for real-time applications for a long time. The design and implementation of such systems has usually been carried out as an extension of the system design principles used for general purpose computer systems. In particular, real-time systems have often been designed as interrupt-driven systems with priority-based scheduling. Priority structures are used to accomplish the real-time processing by assigning higher priority to critical tasks. In this approach *time* is not treated explicitly in resource management or in scheduling of tasks.

The hard real-time applications of tomorrow must provide support for reliable distributed operation. The past design methodologies may not be adequate to meet these challenges. In this book we have attempted to present a comprehensive approach to the design of the next generation of real-time systems.

We believe that it is essential for the next generation of hard real-time systems to use time directly and explicitly. The approach taken here is to make a uniform representation and use of time. Using an object-oriented approach, we consider time to be an integral property of every entity in the system. In addition, a hard real-time system has to have a deterministic and predictable behavior. The techniques useful for making the system behavior deterministic are also presented.

It has often been noted that the maintenance of a complex real-time system becomes a major problem. Any change usually requires extensive testing to assure functional as well as temporal correctness. The approach we propose is to carry out a verification of the resource allocation. In this way we can reduce significantly the testing requirements. The ideas presented in this book are realistic, they have been implemented, and they are therefore practical.

We present a detailed discussion of the current state of the art of real-time systems and application methodologies along with some major recent advances.

This includes a complete presentation of the design, implementation, verification, and testing issues of the real-time problems. Both the application level and the system level view are taken.

This book is aimed at the professional in the field who has to deal with real-time systems from different perspectives. It is useful to the researchers as it presents many novel ideas, not only for real-time systems but also for distributed, reliable systems. The complete treatment of real-time systems makes it well suited for a graduate or advanced undergraduate course on the subject.

This book reflects our experience in building and implementing real-time systems that are in use extensively, as well as our backgrounds in academic research. The main motivation for this work was to explore ways of improving the current design methodologies to meet the challenges of tomorrow. As we found no adequate text on the subject, we started compiling our notes, which resulted in this book.

While writing the book we found many issues we wanted to share with our readers. Some of these issues concern general subjects such as fault tolerance or complexity, while others concern practical issues such as debugging. A detailed presentation of all these topics would make the book unwieldy. The more information we add the more difficult it would be to focus. We therefore decided on this version of the book, in order to present a discussion of all relevant topics, with suitable references for further study.

As the book is aimed at various types of professionals in the field of real-time systems, let us suggest a structure of the book according to reading interests.

The book has sections that are common to all readers, as they include basic information as well as notation and semantic definitions. These sections are: Section 2.1, Section 2.2, Section 2.3, Section 3.1, Section 4.1, Section 5.2, Section 6.1, Section 6.2, Chapter 11, Chapter 12, Section 13.1, Section 14.1, Section 15.1, Section 15.4, Section 16.1, and Section 17.1. The information introduced in these sections is essential for understanding the rest of the book.

Readers who are more engineering-oriented may want to read the following sections: Section 2.4, Section 2.5, Section 3.2, Section 3.3, Section 4.2, Section 5.1, Section 7.1, Chapter 9, Chapter 10, Section 13.2, Section 13.3.1, Section 13.4.1, Section 14.2, Section 14.4, Section 15.2, Section 15.3, Section 16.2, Section 17.2, and Section 17.3.

Readers who are more research-oriented may find further interest in the following sections: Section 4.3, Section 6.3, Chapter 7, Chapter 8, Section 13.2, Section 13.3, Section 13.4, Section 14.2, Section 14.3, Section 14.4, and Section 17.3.

We hope that the above suggested structure will help each professional to focus easily where he or she finds interest.

ACKNOWLEDGMENTS

A project of this magnitude is clearly the result of a lot of support from many friends. Members of the Systems Design and Analysis Group of the Department

of Computer Science, University of Maryland have actively contributed to the development of the material in the book and are implementing the **MARUTI** operating system, which is based on the ideas presented in this book. We would like to gratefully acknowledge the support for our research that has contributed to the development of this book. Our research was supported by the Office of Naval Research and Rome Air Development Center through the Army Strategic Defense Command through grants and contracts to the Department of Computer Science at the University of Maryland. We would also like to acknowledge the Israel Aircraft Industries, Ltd., for contributing to the making of this work.

Shemi-Tov Levi
Ashok K. Agrawala

CONTENTS

List of Figures	xv
Preface	xvii

1	Introduction	1
1.1	Embedded Computer Systems	2
1.2	Historical Perspective	3
1.3	Distributed Real-Time System Environment	4
1.4	Real-Time Programming	5
1.5	Real-Time Operating Systems	6
1.6	Book Organization	7

Part I Real-Time Issues

2	Time Handling	11
2.1	Representation of Time	11
2.2	Time Constraints	13
2.3	Time Service and Synchronization	14
2.3.1	Definitions	14
2.3.2	Clock Synchronization	15
2.3.3	Types of Clock Systems	15
2.4	Master-Slave Algorithms	16
2.4.1	Tempo: A Master-Slave Example	18
2.4.2	Master-Slave Enhancements	20
2.5	Distributed Clock Algorithms	22
2.5.1	A Fundamental Ordering Approach	23

2.5.2	Time Intervals Approach	24
2.5.3	Fault-Tolerant Algorithms	28
3	Objects	34
3.1	Basic Concepts	34
3.1.1	Objects: Justification and Manipulation	34
3.1.2	Creation and Deletion of Objects	35
3.1.3	Accessing Objects	36
3.1.4	Object Architecture	40
3.1.5	Relations and Operations	40
3.1.6	Fault Tolerance Relations	42
3.2	Requirements for Exceptions	44
3.2.1	Interrupt-Driven Systems	44
3.2.2	Communication Service as an Agent Object	49
3.2.3	Exception Handling	51
3.3	Guarantees in Hard Real-Time Systems	54
3.4	Concluding Remarks	55
4	Adding Time to Objects	56
4.1	Temporal Relations	56
4.1.1	Time Representation	56
4.1.2	Temporal Relations	57
4.2	Calendars	61
4.3	Time Projection	64
4.3.1	Constraint Projections	65
4.3.2	Assessment	68
4.3.3	Constraint Propagation	68
4.4	Concluding Remarks	71

Part II Real-Time Applications

5	The Real-Time System Life Cycle	75
5.1	Requirement Specification	77
5.2	Statecharts	79
5.3	Concluding Remarks	82
6	Structured Design Approaches	83
6.1	Event-Based Model	83
6.1.1	Model Description	84
6.1.2	Properties of the Event-Based Model	84
6.1.3	Examples	84
6.2	Process-Based Structured Design	85
6.2.1	Description of a Theoretical Model	85
6.2.2	Structured Design Method Characteristics	85
6.2.3	DARTS	87
6.3	Graph-Based Theoretical Model	89
6.4	Concluding Remarks	92

7	Petri Net Models	93
7.1	Stochastic Petri Net (SPN) Model Analysis	95
7.1.1	Definitions	95
7.1.2	Example of SPN	96
7.2	Annotated Petri Nets	99
7.3	Time-Augmented Petri Nets	101
7.3.1	Time-Driven System Model	101
7.3.2	Concept of Relative Firing Frequency	102
7.3.3	Subclasses of Time-Driven Systems	102
7.3.4	Proving Safeness in the Presence of Time	104
7.4	Assessment of Petri Net Methods	105
8	Axiomatic Approaches	107
8.1	Weakest Precondition Analysis	108
8.1.1	Predicate Transformers	108
8.1.2	Program Time Behavior	109
8.1.3	Method and Example	111
8.2	Real-Time Logic	112
8.3	Time-Related History Variables	115
8.4	State Machines and Real-Time Temporal Logic	119
8.4.1	ESM Components	119
8.4.2	Real-Time Temporal Logic	120
8.5	Concluding Remarks	121
9	Language Support and Restrictions	123
9.1	Real-Time Programming Discipline	123
9.1.1	Language Requirements	124
9.1.2	Discipline for Real-Time Programming	124
9.2	Real-Time Programming Languages	125
9.2.1	Asynchronous Real-Time Language	125
9.2.2	Synchronous Real-Time Language	127
9.3	Schedulability Analysis	128
9.4	Concluding Remarks	130
10	Verification and Validation of Real-Time Software	131
10.1	Testing Real-Time Properties	132
10.1.1	Systematic Testing Methods	132
10.1.2	Statistical Testing	133
10.2	Simulation as Verification Tool	136
10.2.1	Classes and Aims	136
10.2.2	Problems in Simulation for Verification	137
10.3	Testing Control and Data Flow	138
10.3.1	Control-Flow Verification	138
10.3.2	Data-Flow Verification	139
10.4	Proof Systems	143
10.5	Operational Approach	144
10.6	Concluding Remarks	145

Part III Real-Time Operating Systems

11	Properties of Real-Time Operating Systems	149
11.1	Current Operating Systems	149
11.1.1	Priority-Driven Systems	150
11.1.2	Priority-Driven with Enhanced Time Services	150
11.1.3	Time-Driven Scheduling	151
11.1.4	Deadline-Guaranteeing Operating Systems	154
11.1.5	Assessment of Current Approaches	155
11.2	Resource Management/Allocation	156
11.2.1	Scheduling	156
11.2.2	Processor Allocation	158
11.2.3	Architecture Dependency	159
11.3	Time Services	160
11.4	Communication	161
11.4.1	Message Passing	162
11.4.2	Error Handling	163
11.4.3	Issues of Efficiency in Implementation	163
11.5	Name Servers	164
11.6	Data Access Strategy	165
11.6.1	Protection	165
11.6.2	Remote Storage and Directory Services	165
11.7	Fault Tolerance	166
11.8	Other Services	166
11.8.1	Service Architecture	166
11.8.2	Reconfiguration Services	167
11.8.3	I/O Device Services	167
11.9	Concluding Remarks	167
12	Allocation and Scheduling	169
12.1	Problem Definition	170
12.2	Rate Monotonic Priority Scheduling Algorithms	171
12.3	NEXT-FIT-M Partitioning for Rate-Monotonic Schedulers	172
12.4	Allocation with Minimization of IPC	174
12.5	Allocation with Bottleneck Processor Load Minimization	177
12.6	Allocation with Load Balance Optimality Constraint	179
12.7	Serving Non-Real-Time Tasks by a Real-Time Scheduler	183
12.8	Heuristic Approach in Scheduling	184
12.9	Imposing Precedence and Resource Requirements	186
12.10	Concluding Remarks	187
13	Verification of Schedulability	188
13.1	Feasible Schedule Conditions	189
13.1.1	Convex Time Constraints	189
13.1.2	Nonconvex Time Constraints	196
13.2	Algorithms Principles	199
13.3	Schedule Feasibility for Convex Constraint	200
13.3.1	The Verification Algorithm	200

13.3.2	Correctness of Schedule Feasibility Guarantee	204
13.3.3	Properties of Schedule Feasibility Verification	205
13.4	Schedule Feasibility for Nonconvex Constraint	209
13.4.1	The Verification Algorithm	209
13.4.2	Properties of Schedule Feasibility Verification	213
13.5	Conclusion	217
14	Resource Allocation	218
14.1	Definitions and Formulation	219
14.1.1	Model Description	219
14.1.2	Conditions and Formulation	221
14.2	Allocation Algorithm	223
14.2.1	Message Types Used	223
14.2.2	Principles of Allocation Initiation	224
14.2.3	Principles of Algorithm for Allocator	225
14.2.4	Local and External Variables	229
14.2.5	The Allocation Algorithm	229
14.3	Allocation Algorithm Properties	233
14.3.1	Algorithm Termination	233
14.3.2	Allocatability Correctness	234
14.3.3	Achievement of Fault Tolerance Objectives	235
14.4	Reallocation upon Failure	236
14.4.1	Rational	236
14.4.2	Algorithm for Detection Unit D_i	237
14.5	Concluding Remarks	237
15	Communication	238
15.1	Network Characteristics	238
15.1.1	The Network Interface	239
15.1.2	Communication Elements and Timing Uncertainties	240
15.1.3	A Communication Delay Model	242
15.2	Protocols for Real-Time Communication	244
15.2.1	Protocols with Contention	244
15.2.2	Synchronous Protocols	247
15.3	Heterogeneity and Representation	250
15.4	Bounded Semantic Links	251
15.4.1	Passive Links	251
15.4.2	Agents	252
15.5	Concluding Remarks	253

Part IV Operating System Implementation

16	The MARUTI Operating System	257
16.1	Introduction	257
16.1.1	Objectives	257
16.1.2	Approach and Principles	258
16.1.3	What's New?	260

16.2	MARUTI Components	261
16.2.1	Kernel Components	261
16.2.2	Application Level Components	265
17	Operational Issues and Examples	268
17.1	Execution and Distribution Considerations	268
17.1.1	Scheduling Queues	268
17.1.2	Remote Services	269
17.2	Job Acceptance in MARUTI	271
17.2.1	MARUTI after Boot	271
17.2.2	MARUTI after LOGIN	272
17.2.3	Executing a Server Object	272
17.2.4	Executing an Actor (or Agent) Object	273
17.3	Some Examples of Design	274
17.3.1	Variables of Object's Joint	275
17.3.2	Inserting Time Constraint into a Calendar	277
17.3.3	Removing Time Constraint from Object's Calendar	279
17.3.4	Loading and Unloading Time Constraint in Object's Calendar	281
17.4	Concluding Remarks	282

Part V Epilog

18	Conclusion	287
	Bibliography	289
	Index	297

CHAPTER 1

INTRODUCTION

The ever-increasing use of computer systems is clear evidence that the functional capabilities provided by them can be used very effectively for a variety of purposes and in a large number of fields. In many of these applications, the performance of the computer system is measured with metrics such as response time or turnaround time, the implication being that the faster the better with no specific requirement being placed on the timing behavior of the system. *Real-time applications* are different from this paradigm of computation in that they impose strict requirements on the timing behavior of the system. The systems that support the execution of real-time applications and ensure that the timing requirements are met are often referred to as *real-time systems*. Traditionally, the correctness of many computer systems has been taken to imply their logical and functional behavior. For real-time systems correctness depends on the temporal properties of this behavior as well.

As the price and performance of digital computers continue to improve and their size, weight, and power requirements continue to decrease, there has been a steady increase in the use of computer-based real-time systems in a wide variety of fields. Application domains such as military, industry, and medicine indicate a wide spectrum of possible implementations. Current real-time system examples include nuclear power plant control, industrial manufacturing control, medical monitoring, digital fly-by-wire avionics, weapon delivery systems, space navigation and guidance, and reconnaissance systems. As the use of real-time systems has spread, the timing requirements have become more stringent and the reliability requirements more difficult to achieve.

In general, we call a system a real-time system when it can support the execution of applications with time constraints on that execution. A variety of systems clearly meet this definition. Note that no assumptions are made about the structure or the architecture of the computer system used. A particular class of such systems comprises embedded computer systems.

1.1 EMBEDDED COMPUTER SYSTEMS

Many complex systems in use today require a very elaborate control and computational facility to support their continued proper functioning. Such systems often use dedicated hardware as controllers. Clearly, all the computations and control functions can also be carried out by an appropriate computer system. When a computer system is used in a large system to provide control and computation functions, it is often referred to as an *embedded computer system*. Currently we find such systems in almost every aspect of our lives, with computers being introduced into new systems at an ever-increasing rate.

An embedded computer system has to manage and control the rest of the system. It collects data through sensors and issues control commands to mechanical, electromechanical, and electronic actuators. Figure 1.1 illustrates a typical system of this class. Note that this figure could also depict any process control system; we are using it to convey the idea of a computer as a controller in such a system.

A distinguishing feature of embedded computer systems is that they usually provide an execute-only environment, in which no program development goes on. The processing requirements in these systems do not change as they handle a fixed and well-defined workload. Although some embedded systems are designed

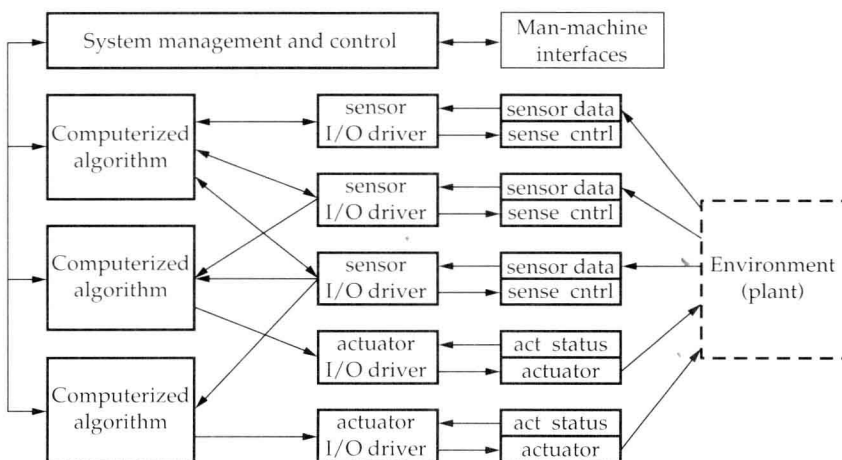


FIGURE 1.1
Embedded computer system.

to handle transient inputs, the processing requirements for such transient requests are predefined. For example, the embedded computer system used in automobiles to control the fuel injection and spark ignition executes one fixed program that may adjust its parameters as its sensors provide it with new information about environmental conditions.

When we study embedded computer systems, their close interaction with mechanical, electromechanical, and electronic components requires that such components be considered as a part of the system and that the expressions and modeling tools used in the study have the capability of representing the interdisciplinary properties of the system. We cannot describe the performance of a robot arm without reference to its mechanics, electronics, and control software. Furthermore, each discipline must maintain consistency with the others. We want the system description to resolve interdisciplinary contradictions when they exist. For example, consider a control system with high-bandwidth control software and control hardware that is very slow. The system description of this example must allow for combining the hardware and its control software. The description must reflect the overall slow response time of this system.

Specifying a system that makes use of techniques from several disciplines starts at the level of the whole system. Out of this high-level specification, we specify the subsystems up to single-discipline components. Derivation of subsystem specifications based on the high-level specification must be consistent and unambiguous. Budgeting software subsystem specifications must, therefore, not only support the performance description of the whole system; it must also be derivable from a whole system description and allow verification of the system properties.

1.2 HISTORICAL PERSPECTIVE

Real-time systems developed from embedded computer systems are still an important family of real-time systems. The use of digital computers in such systems started with the replacement of analog processing sections of control systems during the 1950s. A major step in this direction was taken in March 1956. TRW engineers were contracted by Texaco to computerize a process control system in a polymerization unit of a refinery in Port Arthur, Texas [11]. The process controller employed an RW-300 computer, which controlled 26 flows, 72 temperatures, and 3 pressures. It was announced to be operational in March 1959. In 1962, another major step was taken in the chemical industry, at ICI in England, where a single Ferranti computer replaced complete analog instrumentation that controlled 129 valves and measured 224 nodes.

Other examples of the early use of embedded systems exist. Airborne controllers were needed for missile and aircraft applications. NASA engineers developed many flight control systems during the 1960s for the Mercury, Gemini, and Apollo projects. Other military projects employed time-driven sequencers, digital comparators, and adders for flight control subsystems.