

## CONCURRENT AND REAL-TIME PROGRAMMING IN ADA

Ada is the only ISO standard, object-oriented, concurrent, real-time programming language. It is intended for use in large, long-lived applications where reliability and efficiency are essential, particularly real-time and embedded systems. In this book, Alan Burns and Andy Wellings give a thorough, self-contained account of how the Ada tasking model can be used to construct a wide range of concurrent and real-time systems. This is the only book that focuses on an in-depth discussion of the Ada tasking model. Following on from the authors' earlier title 'Concurrency in Ada', this book brings the discussion up to date to include the new Ada 2005 language and the recent advances in real-time programming techniques. It will be of value to software professionals and advanced students of programming alike; indeed, every Ada programmer will find it essential reading and a primary reference work that will sit alongside the language reference manual.

ALAN BURNS is a Professor in Computer Science at the University of York. His research activities have covered a number of aspects of real-time and safety critical systems, including the assessment of languages for use in the real-time safety critical domain, distributed operating systems, the formal specification of scheduling algorithms and implementation strategies, and the design of dependable user interfaces to safety critical applications. His teaching activities include courses in Operating Systems, Scheduling and Real-time Systems. He has authored over 370 papers and reports and 8 books, including 'Real-time Systems and Programming Languages' (3rd Edition), 'Concurrency in Ada' (2nd Edition) and 'Concurrent and Real-Time Programming in Java'.

ANDY WELLINGS is a Professor of Real-Time Systems in the Computer Science Department at the University of York. He is interested in most aspects of the design and implementation of real-time dependable computer systems and, in particular, in real-time programming languages and operating systems. He is European Editor-in-Chief for the Computer Science journal 'Software-Practice and Experience' and a member of the International Expert Groups currently developing extensions to the Java platform for real-time, safety critical and distributed programming. He has authored over 280 papers and several books, including 'Real-time Systems and Programming Languages' (3rd edition) and 'Concurrency in Ada' (2nd edition).

Cambridge University Press

978-0-521-86697-2 - Concurrent and Real-Time Programming in Ada 2005

Alan Burns and Andy Wellings

Frontmatter

[More information](#)

---

# CONCURRENT AND REAL-TIME PROGRAMMING IN ADA 2005

ALAN BURNS AND ANDY WELLINGS

*University of York*



Cambridge University Press  
978-0-521-86697-2 - Concurrent and Real-Time Programming in Ada 2005  
Alan Burns and Andy Wellings  
Frontmatter  
[More information](#)

---

CAMBRIDGE UNIVERSITY PRESS  
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo  
Cambridge University Press  
The Edinburgh Building, Cambridge CB2 8RU, UK  
Published in the United States of America by Cambridge University Press, New York

[www.cambridge.org](http://www.cambridge.org)  
Information on this title: [www.cambridge.org/9780521866972](http://www.cambridge.org/9780521866972)

© A. Burns and A. Wellings 2007

This publication is in copyright. Subject to statutory exception  
and to the provisions of relevant collective licensing agreements,  
no reproduction of any part may take place without  
the written permission of Cambridge University Press.

First published 2007

Printed in the United Kingdom at the University Press, Cambridge

*A catalogue record for this publication is available from the British Library*

ISBN 978-0-521-86697-2 hardback

Cambridge University Press has no responsibility for the persistence or  
accuracy of URLs for external or third-party internet websites referred to  
in this publication, and does not guarantee that any content  
on such websites is, or will remain, accurate or appropriate.

## Contents

<i>Preface</i>	<i>page xi</i>
<b>1 Introduction</b>	1
1.1 Concurrency	2
1.2 Real-time systems	3
1.3 Ada's time and clock facilities	6
1.4 Summary	13
1.5 Further reading	13
<b>2 The nature and uses of concurrent programming</b>	15
2.1 Uses of concurrent programming	17
2.2 Program entities	18
2.3 Process representation	20
2.4 A simple embedded system	21
2.5 Summary	30
2.6 Further reading	30
<b>3 Inter-process communication</b>	31
3.1 Data communication	32
3.2 Synchronisation	33
3.3 Deadlocks and indefinite postponements	34
3.4 System performance, correctness and reliability	36
3.5 Dining philosophers problem	38
3.6 Shared variables and protected variables	39
3.7 Semaphores	41
3.8 Monitors	44
3.9 Message-based communication	48
3.10 Summary	53
3.11 Further reading	54

vi		<i>Contents</i>
<b>4</b>	<b>Task types and objects</b>	55
4.1	Task creation	57
4.2	Task activation, execution, finalisation and termination	65
4.3	Task hierarchies	70
4.4	Task identification	75
4.5	Task creation, communication and synchronisation within task finalisation	77
4.6	Summary	77
<b>5</b>	<b>The rendezvous</b>	79
5.1	The basic model	79
5.2	The entry statement	81
5.3	The accept statement	83
5.4	The <code>Count</code> attribute	88
5.5	Entry families	88
5.6	Three-way synchronisation	90
5.7	Private entries	92
5.8	Exceptions and the rendezvous	93
5.9	Task states	94
5.10	Summary	94
<b>6</b>	<b>The select statement and the rendezvous</b>	97
6.1	Selective accept	97
6.2	Guarded alternatives	101
6.3	Delay alternative	103
6.4	The else part	107
6.5	The correct use of guards	109
6.6	The terminate alternative	111
6.7	The exception <code>Program_Error</code>	116
6.8	Summary of the selective accept statement	118
6.9	Conditional and timed entry calls	118
6.10	Mutual exclusion and deadlocks	121
6.11	The dining philosophers	124
6.12	Task states	127
6.13	Summary	127
<b>7</b>	<b>Protected objects and data-oriented communication</b>	129
7.1	Protected objects	129
7.2	Mutual exclusion	131
7.3	Condition synchronisation	133
7.4	Entry calls and barriers	135
7.5	Private entries and entry families	139

<i>Contents</i>	vii
7.6 Restrictions on protected objects	142
7.7 Access variables and protected types	144
7.8 Elaboration, finalisation and exceptions	146
7.9 Shared data	147
7.10 The readers and writers problem	148
7.11 The specification of synchronisation agents	151
7.12 Shared variables	152
7.13 Volatile and atomic data	156
7.14 Task states	160
7.15 Summary	161
<b>8 Avoidance synchronisation and the requeue facility</b>	163
8.1 The need for requeue	163
8.2 Semantics of requeue	175
8.3 Requeuing to other entities	179
8.4 Real-time solutions to the resource control problem	183
8.5 Entry families and server tasks	186
8.6 Extended example	190
8.7 Task states	193
8.8 Summary	194
<b>9 Exceptions, abort and asynchronous transfer of control</b>	195
9.1 Exceptions	195
9.2 The abort statement	198
9.3 Asynchronous transfer of control	200
9.4 Understanding the asynchronous select statement	212
9.5 A robust readers and writers algorithm	217
9.6 Task states	221
9.7 Summary	221
<b>10 Object-oriented programming and tasking</b>	223
10.1 The Ada 2005 OOP model	224
10.2 Tasks and interfaces	231
10.3 Protected types and interfaces	239
10.4 Synchronized interfaces	244
10.5 Summary	246
10.6 Further reading	246
<b>11 Concurrency utilities</b>	247
11.1 Communication and synchronisation abstractions	248
11.2 Semaphores	248
11.3 Locks	257
11.4 Signals	263

viii	<i>Contents</i>
11.5 Event variables	264
11.6 Buffers	266
11.7 Blackboards	268
11.8 Broadcasts	269
11.9 Barriers	276
11.10 Concurrent execution abstractions	277
11.11 Callables and futures	278
11.12 Executors	280
11.13 Completion services	284
11.14 Image processing example revisited	288
11.15 Summary	291
<b>12 Tasking and systems programming</b>	<b>293</b>
12.1 Device driving and interrupt handling	296
12.2 Model of interrupts	300
12.3 Task identifiers	311
12.4 Task attributes	313
12.5 Summary	316
12.6 Further reading	316
<b>13 Scheduling real-time systems – fixed priority dispatching</b>	<b>317</b>
13.1 Scheduling	317
13.2 Fixed priority dispatching	319
13.3 Priority ceiling locking	322
13.4 Entry queue policies	327
13.5 Active priorities and dispatching policies	327
13.6 Summary	329
13.7 Further reading	329
<b>14 Scheduling real-time systems – other dispatching facilities</b>	<b>331</b>
14.1 Non-preemptive dispatching	331
14.2 Round-robin dispatching	332
14.3 Earliest deadline first dispatching	335
14.4 Mixed scheduling	347
14.5 Dynamic priorities	348
14.6 Synchronous and asynchronous task control	354
14.7 Summary	359
14.8 Further reading	359
<b>15 Timing events and execution-time control</b>	<b>361</b>
15.1 Events and event handling	361
15.2 Timing events	362
15.3 Dual priority scheduling	366

<i>Contents</i>	ix
15.4 Execution-time clocks	369
15.5 Execution-time timers	371
15.6 Group budgets	374
15.7 Task termination events	387
15.8 Summary	389
15.9 Further reading	389
<b>16 Real-time utilities</b>	391
16.1 Real-time task state	393
16.2 Real-time task release mechanisms	395
16.3 Periodic release mechanisms	397
16.4 Sporadic release mechanisms	405
16.5 Aperiodic release mechanisms and execution-time servers	407
16.6 Real-time tasks	415
16.7 The cruise control system example	419
16.8 Summary	432
<b>17 Restrictions, metrics and the Ravenscar profile</b>	433
17.1 Restricted tasking and other language features	433
17.2 The Ravenscar profile	436
17.3 Partition elaboration control	439
17.4 Examples of the use of the Ravenscar profile	440
17.5 Metrics and optimisations	448
17.6 Summary	449
17.7 Further reading	450
<b>18 Conclusion</b>	451
18.1 Support for concurrency	452
18.2 Support for real-time	452
18.3 New to Ada 2005	453
18.4 Outstanding issues and the future	453
<i>References</i>	455
<i>Index</i>	457



## Preface

The development of the Ada programming language forms a unique and, at times, intriguing contribution to the history of computer languages. As all users of Ada must know, the original language design was a result of competition between a number of organisations, each of which attempted to give a complete language definition in response to a series of documented requirements. This gave rise to Ada 83. Following 10 years of use, Ada was subject to a complete overhaul. The resulting language, Ada 95, had a number of significant changes from its predecessor. A further 10 years of use has produced another version of Ada, known as Ada 2005, this time the changes are less pronounced and yet there are some key extra facilities, especially in the areas of real-time programming.

Closely linked to the development of Ada has been this book on its concurrent features. Starting out as ‘Concurrent Programming in Ada’, it became ‘Concurrency in Ada’ when the Ada 95 version of the language was defined. There were two editions of this title. With the new features of Ada 2005, it has been decided to broaden the focus of the book to include real-time issues – hence this first edition of the new title ‘Concurrent and Real-Time Programming in Ada 2005’. No prior knowledge of concurrent programming (in general) or of Ada tasking (in particular) is assumed in this book. However, readers should have a good understanding of at least one high-level sequential programming language and some knowledge of operating system principles.

This book is aimed both at professional software engineers and at students of computer science (and other related disciplines). Many millions of lines of Ada 83 and 95 code have been produced world wide, and over the next decade a wide range of new applications will be designed with Ada 2005 as the target language. It is important that Ada programmers do not restrict themselves to a sequential subset of the language on the dubious assumption that tasking is not appropriate to their work, or for fear that the tasking model is too complex and expensive. Tasking is an integral part of the language, and programmers must be familiar with,

if not experienced in, its use. Due to space considerations, books that describe the entire language may not deal adequately with the tasking model; this book therefore concentrates exclusively on this model.

Students studying real-time programming, software engineering, concurrent programming or language design should find this book useful in that it gives a comprehensive description of the features that one language provides. Ada is not merely a product of academic research (as are many concurrent programming languages) but is a language intended for actual use in industry. Its model of tasking was therefore integrated into the entire language design, and the interactions between tasking and non-tasking features were carefully defined. Consequently, the study of Ada's model of concurrency should be included in those advanced courses mentioned above. However, this does not imply that the full tasking model is free from controversy, has a proven formal semantic basis or is amenable to efficient implementation. The nature of these areas of 'discussion' are dealt with, as they arise in this book.

Unlike Ada 83, which defined a single language, the Ada 95 and 2005 definitions have a core language design plus a number of domain-specific annexes. A compiler need not support all the annexes but it must support the core language. Most of the tasking features are contained in the core definition. But there are relevant annexes that address systems programming and real-time programming.

The first chapter provides a basic introduction to concurrent and real-time systems and gives an overview of the clock facilities within Ada.

Chapters 2 and 3 look in detail at the uses of concurrent programming and the inherent difficulties of providing inter-process communication. There is, as yet, no agreement on which primitives a concurrent programming language should support and, as a consequence, many different styles and forms exist. In order to understand the Ada tasking model fully, it is necessary to appreciate these different approaches and the problems faced by the user of any language that supports multi-processing.

The Ada task is introduced in Chapter 4 and the rendezvous and the important select statement are considered in the following two chapters. The rendezvous provides a synchronous communication mechanism. Data-orientated asynchronous communication is considered in Chapter 7, together with the important abstraction of a protected object. This provides a passive means of encapsulating data and providing mutual exclusion. An effective way of increasing the expressive power of the communication primitives is the requeue facility. This is described, with many examples given, in Chapter 8. The relationship between tasks and exceptions is dealt with in Chapter 9. This chapter also covers the means by which one task can affect the behaviour of another task asynchronously.

Chapter 10 considers the interplay between tasking and the object-orientated programming features of the language. This forms the basis from which a collec-

tion of concurrency utilities can be defined. A number of these are provided in Chapter 11.

As indicated earlier, a number of the annexes deal with issues of relevance to concurrent programming. Chapter 12 considers systems programming (including support for low level programming). For real-time programmers, perhaps the most important issue is scheduling. Ada provides a comprehensive list of features that are covered in Chapters 13 and 14. In addition to scheduling, real-time programs need to have control over when events are executed and control over the resources that tasks and groups of task require at run-time. These issues are covered in Chapter 15.

Having introduced all of Ada's relevant features, Chapter 16 then provides a collection of real-time utilities that can be used to gain access to the power of the language. This is followed in Chapter 17 by consideration of the usefulness of subsetting Ada and using profiles to gain access to efficient and certifiable implementations. In particular, the Ravenscar profile is described in this chapter. Finally, in Chapter 18 conclusions are provided and a short summary of the differences between Ada 2005 and Ada 95 is given in the context of concurrent and real-time programming, together with a brief look to the future.

The material presented in this book reflects the authors' experiences in both using and teaching Ada tasking. Teaching experience has been obtained by writing and presenting courses at the University of York (UK) and by developing educational material and training.

### **Further material**

Further material on all aspects of real-time and concurrency in Ada 2005 can be found on a [www](http://www.cs.york.ac.uk/~rts/ada/CRTIA.html) page dedicated to this book:  
<http://www.cs.york.ac.uk/~rts/ada/CRTIA.html>.

### **Real-time systems research at York**

Alan Burns and Andy Wellings are members of the Real-Time Systems Research Group in the Department of Computer Science at the University of York (UK).

The aim of the group is to undertake fundamental research, and to bring modern techniques, methods and tools into engineering practice. Areas of application of our work include space and avionic systems, engine controllers, automobile control, manufacturing systems, sensor nets and multi-media systems. Specifically, the group is addressing: scheduling theories, language design, kernel design, communication protocols, distributed and parallel architectures and program code analysis.

Further information about the group's activities can be found via our www page:  
<http://www.cs.york.ac.uk/~rts>

### **Acknowledgements**

The authors would like to thank the following individuals who have, directly or indirectly, helped during the preparation of this book. In particular, John Barnes, Andrew Borg, Javier Miranda, Pat Rogers, Ed Schonberg, and Tucker Taft.

The authors also wish to thank members of ISO committee ARG and attendees of the IRTAW series for their input to some of the issues discussed in this book.

Finally, we would like to thank the AdaCore GNAT project team for all their efforts to produce a public domain Ada 2005 compiler. Writing a book for a new language is very difficult when there are no validated compilers to help test the code. Access to the GNAT system has provided us with more confidence that the code given in this book is at least syntactically correct!