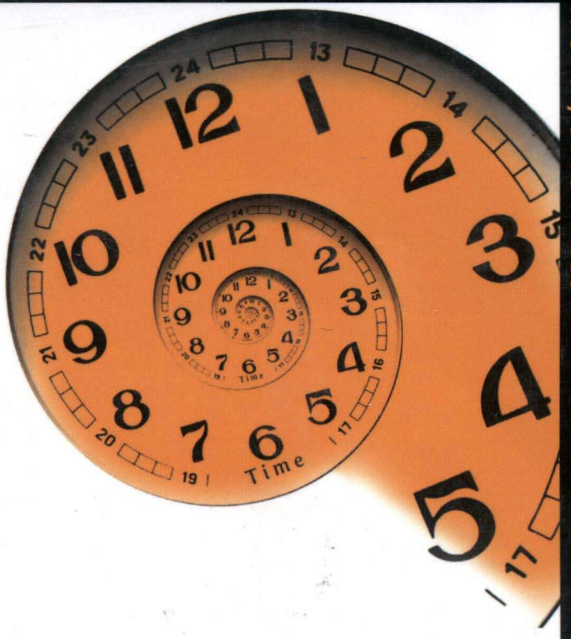
Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE

Developing Cyber-Physical Systems



The MK/OMG PRESS

Bran Selić

Sébastien Gérard

Foreword by Richard Soley

MK MORGAN KAUFMANN

OMG OBJECT MANAGEMENT GROUP

Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE

Developing Cyber-Physical Systems

Bran Selić

Sébastien Gérard

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann is an imprint of Elsevier

Acquiring Editor: *Andrea Dierna*
Editorial Project Manager: *Kaitlin Herbert*
Project Manager: *Punithavathy Govindaradjane*
Designer: *Mark Rogers*

Morgan Kaufmann is an imprint of Elsevier
225 Wyman Street, Waltham, MA 02451, USA

Notices
Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods or professional practices, may become necessary. Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information or methods described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Library of Congress Cataloging-in-Publication Data
Selić, Bran.
Modeling and analysis of real-time and embedded systems with UML and MARTE: developing cyber-physical systems / Bran Selić, Sébastien Gérard.
pages cm
Includes bibliographical references and index.
ISBN 978-0-12-416619-6 (alk. paper)
1. Embedded computer systems—Computer simulation. 2. UML (Computer science) I. Gérard, Sébastien. II. Title.
TK7895.E42S3985 2014
006.2'2—dc23 2013027695

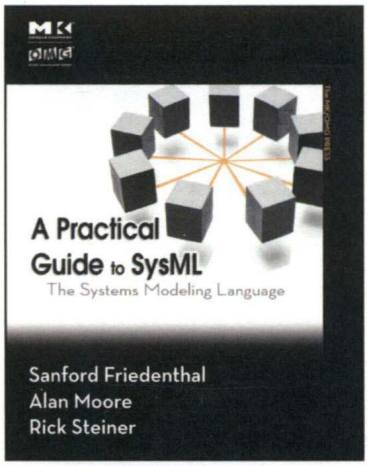
British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-0-12-416619-6

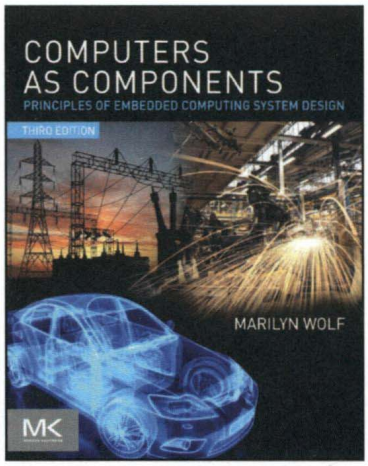
Printed and bound in the United States of America
14 15 16 17 18 10 9 8 7 6 5 4 3 2 1

For information on all MK publications visit our website at *www.mkp.com*

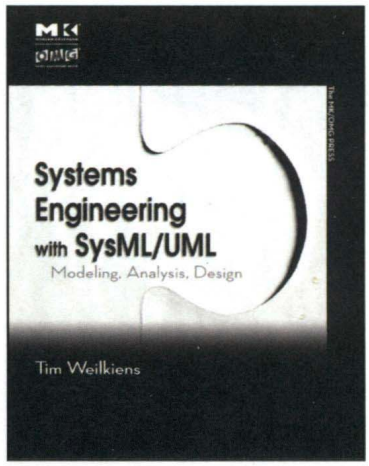
Related Titles from Morgan Kaufmann



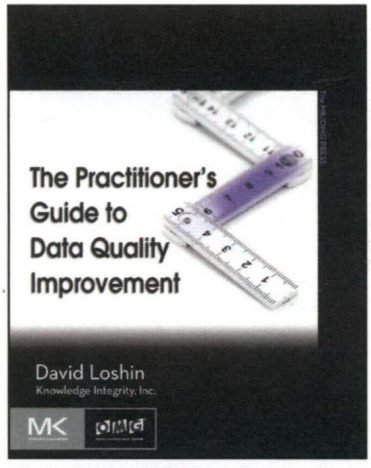
A Practical Guide to SysML
The Systems Modeling Language
Sanford Friedenthal
Alan Moore
Rick Steiner
ISBN: 9780123786074



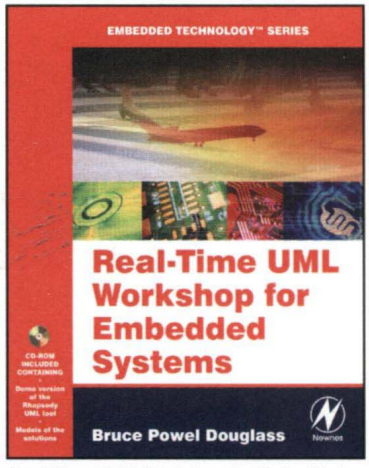
Computers as Components, 3rd Edition
Principles of Embedded Computing System Design
Marilyn Wolf
ISBN: 9780123884367



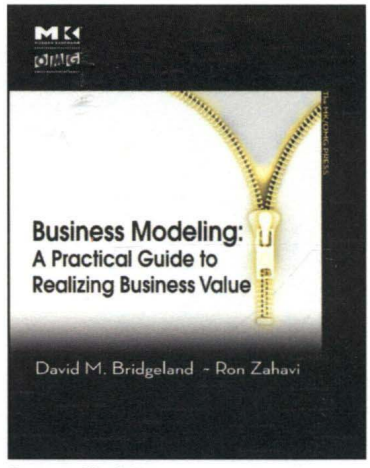
Systems Engineering with SysML/UML
Modeling, Analysis, Design
Tim Weilkiens
ISBN: 9780123742742



The Practitioner's Guide to Data Quality Improvement
David Loshin
ISBN: 9780123737175



Real-Time UML Workshop for Embedded Systems
Bruce Powel Douglass
ISBN: 9780750679060



Business Modeling
A Practical Guide to Realizing Business Value
David M Bridgeland
Ron Zahavi
ISBN: 9780123741516

MORGAN KAUFMANN
mkp.com

Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE

To Lillian, Andrew, Matthew, and Sandy
and
To Marie, Zélie, Nicole, and Jean-Marie

Acknowledgments

The authors express their gratitude and indebtedness first and foremost to Professors Murray Woodside and Dorina Petriu of Carleton University in Ottawa, Canada, who made a major contribution to the text of chapter on performance analysis (Chapter 11), to Drs. Sara Tucci-Piergovanni and Chokri Mraidha for their key contributions to the chapter on schedulability analysis (Chapter 10), and to Dr. Richard Soley, Chairman and CEO of the Object Management Group, who graciously agreed to write the foreword. In addition, we would like to thank the staff of the Laboratoire d'Ingénierie dirigée par les modèles pour les Systèmes Embarqués (LISE) of the Commissariat à l'Énergie Atomique (CEA) LIST group in Gif-sur-Yvette, France, who contributed in various ways to this effort and, in particular (in alphabetical order): Arnaud Cuccurru, Frédérique Descreaux, Hubert Dubois, Agnès Lanusse, Ansgar Radermacher, François Terrier, Patrick Tessier, and Yann Tanguy.

The authors are also extremely grateful to those hardy expert reviewers, Frederic Mallet, Chokri Mraidha, and Rob Pettit, whose detailed comments, corrections, and suggestions have greatly improved the quality of the text. Any remaining errors and rough spots are the sole responsibility of the authors. Thanks are also due to our very supportive editors at Morgan Kaufmann (Elsevier), Andrea Dierni, and Kaitlin Herbert, who provided general guidance and also helped us navigate the intricacies of preparing the text for publication. We also acknowledge the contribution of Pavel Hruby, who prepared the Visio templates for UML 2, which were used in constructing most of the diagrams in the book.

Last but most definitely not least, we both owe unconditional thanks for the support of our respective families, who had to endure our absences and distractions while we worked on this manuscript. Hence, we dedicate this book to them.

Bran Selić
Nepean, Canada

Sébastien Gérard
Gif-sur-Yvette, France

Foreword

Early in this excellent tome you will find a terminological quibble, one of the myriad quibbles over terms found in the juvenile field of computer science. Should we say model-based engineering, model-based development, or model-driven development? There are really two arguments in that quibble:

1. Is it model-based or model-driven? These are functionally the same, based on the concept that complex organizations of systems should be based on (driven by) lower cost (and generally more abstract) models of those organizations.
2. Is it engineering, development, or (dare I say) architecture? In fact modeling should be used to drive development; it should be used to specify an architecture (after all, that is precisely what is done in the building trades); and the process of developing a working organization, device, or software program should be engineered based on a model.

Obviously, as the progenitor of the term, I like the term Model-Driven Architecture (MDA), the name that in 2001 pulled together the many streams of modeling-based development of software and other complex systems. Terminological quibbles aside, the importance of modeling of complex systems—whether software, ships, buildings, weapon systems, cabbages, or kings—is quite clear:

- Abstractions of complex systems are extraordinarily valuable in engineering processes for developing predictions of systems, from the usability of those systems to the likely failure modes and useful lifetimes of those systems.
- Abstractions of complex systems allow the pinpointing of subsets or parts of complex systems for analysis, and further the likely effects of integrating systems of systems.
- Abstractions of complex systems are generally much less expensive than the actual expressions of those systems (even in the case of software), and can thus serve as risk-mitigation approaches to the problems of large systems engineering.

None of this should be surprising to even the casual observer of engineering processes. Why, then, am I asked quite often why so many software developers do not model their systems before committing code to paper? The answer, of course, has more to do with human behavior than technology. It is a matter of training, of expectations, and of misunderstandings about the full costs of software development, and most importantly, it is the same reason that buggy whips were still made in quantity in 1910, when horse buggies were already on their way out as a primary method of transportation in favor of automobiles.

Modeling is key to engineering

While engineering has been developed over thousands of years—and as the authors of this book note—it *had* to be developed. That is, over the thousands of years of the creation of the engineering discipline, proven solutions have been developed, tested, and embedded in the body of knowledge of engineering. The authors of this tome state:

> *The use of models and modern model-based engineering technologies and related industry standards as a means of reducing accidental complexity in the development of real-time and embedded software is the primary theme of this book.*

In fact, it is more than the reduction of accidental complexity that models offer (see above), and it is far more than the development of real-time and embedded software.

A pedagogical example is relevant. On August 10, 1628, the great new warship Gustav Vasa (named after one of the greatest rulers of his time, Gustav Eriksson, otherwise known as Gustav I, King of Sweden) sailed proudly into its own home harbor and promptly sank into the Baltic mud, with the loss of 53 lives. Ships had been built for time immemorial all over the world, yet the nascent naval engineering field had not discovered how to compute the center of gravity of a large complex engineering system (like a ship), although it was certainly understood that a ship's center of gravity must always be below the waterline (or it will shortly find its way below the waterline, by process of sinking). This enormous, visible, deadly, and costly failure would spur (as all engineering failures do) a great leap in naval engineering, most importantly in modeling naval vessels. Further project management issues from the design and development of the Gustav Vasa (the untimely death of Henrik Hybertsson, the shipwright; the constantly changing specifications for the ship and its armaments; and most importantly a failed simulation of seaworthiness conducted before launch) contributed to a callosal failure, which was also a tremendously valuable failure that guaranteed better naval engineering in the future.

Engineered systems have always had a rather high level of expectation of success and correctness by the public at large; as engineering contributes to a better standard of life worldwide, it also engenders those expectations. Though civil engineers are aware that all engineered structures have some probability of failure, the public continues to be surprised when bridges fail; though naval engineers are aware of the same facts, the public continues to be surprised when ships sink. This gap can only be bridged by shared expectations of quality and cost, and quality and cost can only be brought into the engineering discipline through modeling.

Cyber-physical systems

There are various names for the connected nature of engineered systems that are largely (or entirely) dependent on correct underlying software systems:

- *Cyber-physical Systems*. Though this term directly brings together the cyber and physical worlds, it is quite a complicated term for the average person.
- *Internet of Things*. Though this term sounds fun for academics and other researchers, it seems too general to be acceptable.
- *Industrial Internet*. This term brings together the term most people understand to have changed the way we read, write, entertain ourselves, listen to music, and conduct many other activities (the Internet) with the largely unchanged way in which industrial systems are developed, delivered, and measured (consider the continuing use of ladder diagrams in discrete programmable control systems as a good example).

This move of industrial systems to the Internet only aggravates the need for correctness and completeness of software systems, pushing ever higher the expectation of correctness for cyber-physical systems.

Is a domain-specific language better than UML?

Given these expectations, how do we bring better development discipline to the software world, especially the real-time and embedded software world? And how to we deal with the bewildering blizzard of modeling languages? Is the Unified Modeling Language (UML) the best choice, or should we take a more domain-specific language (DSL) approach to ensure our designs are best matched to the system under development?

The reality is, of course, much simpler: the UML is just one of a large family of related modeling languages (including at least SysML, SoaML, UPDM, BPMN, UML for Systems on a Chip, and others), and is in fact a DSL (designed for use in software development, but used in other areas as well). The subject of this book, Modeling and Analysis of Real-Time and Embedded Systems (MARTE), is a DSL for designing, analyzing, and building embedded & real-time systems.

Implementations of the MARTE language are quite easy to find today, even if this book appears to be the first tome dedicated to its explication. Just as ships should never have been developed without understanding center of gravity issues after the failure of the Gustav Vasa, software-driven real-time and embedded systems should never again be built without a thorough analysis through the use of the MARTE language; to do so would simply be a negligent approach to engineering embedded systems. This makes this book indispensable for any engineer building the Industrial Internet of today and tomorrow, based on real-time and embedded systems.

May the Gustav Vasa sail again, this time without sinking!

Richard Mark Soley Ph.D.
Chairman and Chief Executive Officer,
Object Management Group, Inc.,
Brussels, Belgium

Preface

An honourable work glorifies its master—if it stands up

Lorenz Lechler, Instructions, 1516

Regardless of which statistics we are inclined to believe, there is little doubt that software systems all too often fail to "stand up."[1] There are many reasons for this, not the least of which is the often overwhelming complexity of such systems [4]. If we make a crude comparison with mechanical systems by equating a line of program code to a component in a machine, then even a simple 5,000-line program easily qualifies as a reasonably complex system in most people's judgment. Yet, there are numerous software systems in use today that incorporate millions and even tens of millions of lines of code.

In addressing complexity, it is helpful to distinguish between two types of complexity: (1) *essential complexity*, which is inherent to the purpose and nature of the system and is, therefore, unavoidable and (2) *accidental complexity*, that is, complexity that arises from the use of inadequate or ineffective methods and tools [2].

When it comes to real-time software systems, much of the essential complexity stems from the inescapable complexity of the physical world with which these systems interact. Needless to say, this world is highly concurrent, often unpredictable, interconnected in complex ways, and extremely diverse. Yet, the system and its software are required to cope successfully with at least some of these phenomena. Compounding our difficulties are resource limitations, such as limited processing speeds and limited amounts of memory, which further constrain our ability to address these issues. We have, over time, evolved a set of techniques and technologies to mitigate some of these difficulties, such as the use of mutual exclusion mechanisms for dealing with concurrency conflicts, or various redundancy patterns for dealing with failures. But, no matter how hard we try, these are essential characteristics of the problem, which cannot be eliminated by either process or technology.

Accidental complexity, on the other hand, is something that we can and, clearly, *must* strive to eliminate. Sadly, it is the authors' experience that real-time software development abounds with accidental complexity, perhaps because it is perceived as outside the mainstream of software engineering. It is typically developed using tools and methods that do not account for the specific problems of the real-time domain. A classical and notable example of this can be seen in the desire to ignore entirely the physical aspects of computing. For example, the well-known computing pioneer, Edsger Dijkstra, once wrote: "I see no meaningful difference between programming methodology and mathematical methodology," suggesting that software development should be a branch of applied mathematics [5]. This implies an idealized and, some would say, idealistic approach to software design, where physical factors are either ignored or deemed secondary. This line of thinking has led to development methods based on the simplifying but often inappropriate assumption that the time required to execute complex algorithms is negligible (the so-called "zero-time" assumption), or that there is a infinite

[1] The website http://www.it-cortex.com/Stat_Failure_Rate.htm maintains a collection of sources focusing on studies of software failure rates. Although there is some variance in the results across the different studies, they all indicate rates that would be deemed unacceptable in more traditional engineering disciplines.

supply of necessary resources available (memory, bandwidth, etc.). In fact, entire development methodologies have evolved that actively discourage any concerns with the physics of computing, which is reflected in oft-quoted but much misunderstood phrases such as "premature optimization" or "platform-independent design." Design issues related to the physical aspects of a computing system are typically bundled into a nebulous category referred to as "non-functional" issues, to be dealt with only *after* the "main" problem of functionality has been solved.

It is, of course, essential that a design realizes the desired functionality, but, as has often been pointed out producing the right output at the wrong time is still wrong. A program that takes 25 hours to predict tomorrow's weather is not particularly useful. However, any engineering problem needs to be addressed in its full context, which, at least in the case of real-time systems, necessarily includes the physical dimension of computing. After all, no matter how far we advance our software technologies, they will always require hardware to run. In a very pragmatic sense, computer hardware constitutes the raw material out of which our software systems are constructed. And, as with any technology, the raw material used and its properties must be accounted for and proper trade-offs made during system design.[2]

This incursion of the physical into the logical world of software is what distinguishes the design of real-time software most from other types of software. It requires that we extend our tools and methods beyond mathematics and enter the realm of *engineering*. Fortunately, engineering is a long-standing and well-developed discipline; it abounds with proven solution patterns that have evolved over thousands of years. One of the most fundamental and most useful of these is the use of *models and modeling* to support design. Models are used in many different ways in engineering, including, notably, helping us *predict* the key characteristics of designs before committing full resources to implementing them. This not only improves the overall reliability of our designs but also greatly reduces engineering risk and development time.

The use of models and modern model-based engineering technologies and related industry standards as a means of reducing accidental complexity in the development of real-time and embedded software is, in a way, the primary theme of this book.

About this book

Why this book

Interest in model-based methods for designing software has grown significantly since the introduction of the Unified Modeling Language (UML) and its adoption as an industry standard in the late 1990s. Quite naturally, this interest has extended to the real-time domain, resulting in the adoption of several real-time-oriented modeling language standards based on UML. The first of these, the UML Profile for Schedulability, Performance, and Time [12], was supplanted by the more advanced MARTE standard [13], which was aligned with the most recent major revision of UML, UML 2 [14].

MARTE, which stands for Modeling and Analysis of Real-Time and Embedded systems, is a *profile* of UML. A UML profile is a domain-specific interpretation of the general UML language that

[2] It is good to keep in mind a quote from a 2,000-year-old engineering text by the Ancient Roman engineer, Vitruvius: "All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament."[18]

specializes some UML concepts to reflect domain phenomena and concerns. Because a properly defined profile is conformant to the rules and concepts of standard UML, it has the potential to reuse some of the existing UML tools, methods, and expertise. In the specific case of MARTE, the profile was not designed to displace UML but as a complementary domain-specific language.

MARTE provides broad coverage of the real-time domain and is the result of the collective work of numerous domain experts. The current version of the standard extends to almost 800 pages and is structured as a kind of reference manual with only a modicum of methodological guidance,[3] which is the standard format for all language specifications provided by the Object Management Group (OMG). Quite naturally, this presents an intimidating proposition for someone interested in learning about MARTE.

Consequently, this book is intended to provide a user-friendly introduction to the MARTE concepts and also to act as a methodological guide for their application in practice. It serves to identify and describe the key ideas behind MARTE, based on reduced and approachable (but nevertheless technically accurate) descriptions. However, beyond its core purpose as an introductory text, we expect that, because of its methodological content, the book will serve as a convenient user guide and reference even to expert MARTE users.

Modern model-based engineering for software development

The technical foundations for this book are the UML language and an approach to software development variously referred to as *model-based engineering*, *model-based development*, or *model-driven development* (we will use only the first of these in this book). Whatever we choose to call it, the essence of this approach is based on the use of models as an *essential* element of the development process. That is, it goes beyond the traditional exploitation of software models merely as a kind of power-assist and documentation facility. In particular, it means using models as an engineering tool for doing predictive analyses of key characteristics of proposed designs. In addition, given the unique nature of software, models of software systems can be used in a number of other ways, including, notably, computer-based code generation. This can greatly reduce the likelihood that design intent, which is captured in a model, will be corrupted during implementation. And, of course, it can also improve both productivity and overall design quality.

The OMG, UML, and MDA

The OMG[4] is an open not-for-profit computer industry consortium that develops and issues international standards for a wide range of computing-related technologies. It was established in 1989, focusing initially on distributed middleware standards, such as CORBA. However, with the increased interest in the object-oriented design methods that occurred in the mid 1990s, it broadened its scope to modeling and modeling languages for computer applications. The initial outcome of this expanded initiative was the Unified Modeling Language standard [14].

[3] This is by intent—specifications of this type generally avoid methodological aspects in order to remain open to a range of different approaches.

[4] http://www.omg.org