# MODELLING SYSTEMS: PRACTICAL TOOLS AND TECHNIQUES IN SOFTWARE DEVELOPMENT

Second Edition

How can we make sure that the software we build does what it is supposed to do? This book provides an insight into established techniques that help developers overcome the complexity of software development by constructing models of software systems in early design stages. It uses one of the leading formal methods, the Vienna Development Method (VDM), and combines training in the formalism with industry-strength tool support and examples derived from real industrial applications. The principles taught here also apply to many of the current generation of formal methods.

This second edition has been updated to include advanced online tool support for formal modelling as well as up-to-date reports on real commercial applications in areas as diverse as business information systems and firmware design.

# MODELLING SYSTEMS: PRACTICAL TOOLS AND TECHNIQUES IN SOFTWARE DEVELOPMENT

Second Edition

JOHN FITZGERALD

and

PETER GORM LARSEN

CAMBRIDGE UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo, Delhi

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
Information on this title: www.cambridge.org/9780521899116



First published 1998
Second edition 2009

Printed in the United Kingdom at the University Press, Cambridge

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

ISBN 978-0-521-89911-6 hardback

Additional resources for this publication at www.vdmbook.com

# Contents

*Foreword* *page* ix
*Preface* xi
1 Introduction 1
1.1 Software 1
1.2 Modelling and analysis 2
1.3 This book 3
1.4 VDM-SL 4
1.5 The structure of a VDM-SL model 4
1.6 Analysing a model 9
2 Constructing a Model 13
2.1 Introduction 13
2.2 Requirements for an alarm system 13
2.3 Constructing a model from scratch 14
2.4 Reading the requirements 15
2.5 Extracting possible types and functions 16
2.6 Sketching type representations 16
2.7 Sketching function signatures 22
2.8 Completing the type definitions 23
2.9 Completing the function definitions 24
2.10 Reviewing requirements 29
3 VDMTools Lite 35
3.1 Introduction 35
3.2 Installing VDMTools Lite 36
3.3 Configuring the alarm example 36
3.4 Syntax and type checking models 38
3.5 Interpreting and debugging models 42
3.6 Test coverage 48

| | | | |
|---|---|---|---|
| | 3.7 | Integrity checking | 49 |
| | 3.8 | Setting options | 50 |
| 4 | Describing System Properties Using Logical Expressions | | 55 |
| | 4.1 | Introduction | 55 |
| | 4.2 | The temperature monitor | 55 |
| | 4.3 | Logical expressions | 57 |
| | 4.4 | Presenting and evaluating predicates | 64 |
| | 4.5 | Using quantifiers | 66 |
| | 4.6 | Coping with undefinedness | 71 |
| 5 | The Elements of a Formal Model | | 77 |
| | 5.1 | Introduction | 77 |
| | 5.2 | A traffic light control kernel | 78 |
| | 5.3 | Union and basic types | 81 |
| | 5.4 | Basic type constructors | 83 |
| | 5.5 | Record types | 84 |
| | 5.6 | Invariants | 85 |
| | 5.7 | Explicit function definitions | 87 |
| | 5.8 | Functions for changing signals | 88 |
| | 5.9 | Reviewing the safety requirements | 96 |
| | 5.10 | Optional types: modelling failure behaviour | 96 |
| 6 | Sets | | 99 |
| | 6.1 | Introduction | 99 |
| | 6.2 | The set type constructor | 100 |
| | 6.3 | Defining sets | 101 |
| | 6.4 | Modelling with sets | 103 |
| | 6.5 | Distributed set operators | 115 |
| | 6.6 | Summary | 118 |
| 7 | Sequences | | 121 |
| | 7.1 | Introduction | 121 |
| | 7.2 | The sequence type constructor | 122 |
| | 7.3 | Defining sequences | 122 |
| | 7.4 | Modelling with sequences | 125 |
| | 7.5 | Further operators on sequences | 132 |
| | 7.6 | Abstraction lesson: choosing abstraction levels | 135 |
| 8 | Mappings | | 137 |
| | 8.1 | Introduction | 137 |
| | 8.2 | The mapping type constructor | 139 |
| | 8.3 | Defining mappings | 139 |
| | 8.4 | Modelling with mappings | 140 |
| | 8.5 | Summary | 154 |

9 Recursive Structures 157
9.1 Recursive data structures: trees 157
9.2 Abstract syntax trees 161
9.3 Directed graphs 164
9.4 An application of directed graphs: code optimisation 167
9.5 Abstraction lesson: executability 169
10 Validating Models 171
10.1 Introduction 171
10.2 Internal consistency: proof obligations 173
10.3 Visualisation of a model 180
10.4 Interfacing to legacy code 182
10.5 Systematic testing 183
10.6 Using proofs 184
10.7 Choosing a validation technique 187
11 State-Based Modelling 189
11.1 Introduction 189
11.2 State-based modelling 190
11.3 A state-based model of the explosives store controller 191
11.4 A state-based model of the trusted gateway 196
11.5 Validation of state-based models 199
12 Large-Scale Modelling 203
12.1 Introduction 203
12.2 A structure for the tracker model 204
12.3 Information hiding 212
12.4 Object-oriented structuring 214
13 Using VDM in Practice 217
13.1 Introduction 217
13.2 Development activities and processes 218
13.3 Common development problems 219
13.4 Advantages of VDM technology 220
13.5 Getting started 223
13.6 VDM and its extensions 226
13.7 Industrial applications of VDM 227
13.8 Moving on: information resources 232
Appendix A Language Guide 235
A.1 Identifiers 235
A.2 Type definitions 236
A.3 Basic data types and type constructors 236
A.4 Data type operator overview 237
A.5 Expressions 246

A.6 Patterns 248
A.7 Bindings 249
A.8 Explicit function definition 249
A.9 Implicit functions 250
A.10 Operations 250
A.11 The state definition 251
A.12 Syntax overview 252
Appendix B Solutions to Exercises 263
B.1 Solutions for Chapter 2 Exercises 263
B.2 Solutions for Chapter 3 Exercises 263
B.3 Solutions for Chapter 4 Exercises 264
B.4 Solutions for Chapter 5 Exercises 265
B.5 Solutions for Chapter 6 Exercises 266
B.6 Solutions for Chapter 7 Exercises 268
B.7 Solutions for Chapter 8 Exercises 268
B.8 Solutions for Chapter 9 Exercises 270
B.9 Solutions for Chapter 10 Exercises 273
B.10 Solutions for Chapter 11 Exercises 275
*Bibliography* 277
*Subject Index* 283
*Definitions Index* 287

# Foreword

Software engineers produce many descriptions: those of the environment or domain in which a desired computing system software is to exist; descriptions of the requirements put on the software; and descriptions of the software design that implements the requirements. Thus the descriptions span the spectrum from application domain, via requirements and software architecture, program organisation and lower level designs, to executable code. While its concerns may be general, software engineering is unique among engineering disciplines in that its primary products are descriptions that must eventually satisfy the laws of mathematical logic and metamathematics.

Other engineering disciplines have to handle a quantum leap into physical reality – the stuff of natural science. In software engineering there is a different quantum leap: that from description to execution. Software engineering is thus about structuring and relating descriptions.

Abstraction and modelling taken together are the keys to mastering the complexity of environments and systems. Formal specification is employed to express abstractions and to ensure affinity to real domains. Such specifications open up ways to establish the proper relation between domain and requirements models as well as potentially verifying the links between software architecture, requirements models and the stages of design. This increases the chance of achieving a proper fit to the environment, to user expectations and of the correctness of implementation.

VDM was first conceived at the IBM Vienna Laboratory during the summer of 1973. The quarter of a century which separates that date from the publication of this book has shown that VDM is characterised by having remarkably robust, yet simple and elegant, means of abstraction and modelling. Careful attention was paid during the 1980s to ensuring a consistent and comprehensive final version of the VDM Specification Language (VDM-SL). This was supported by a method for specification refinement (reification) including a Logic for Partial Functions Proof

System. VDM is today as powerful a tool and technique for software development as any available.

Software development is pursued in a world where the engineer is not always allowed to pursue the ideas of formal development as epitomised by VDM-SL. But anyone who is aware of the fundamental idea of building abstract models can benefit from its immense power to aid understanding and communication.

In this delightful book former students of ours bring you realistic and effective techniques for abstraction and modelling. The practical, tool-based, approach is one which should give their readers and students (present and future software engineers) the ability to employ these techniques in their everyday work.

*Dines Bjørner*
*Cliff Jones*
Hiroshima, November 1997

# Preface

For developers of computer-based systems, capturing and understanding the complex functional requirements and behaviour of software components has come to represent a considerable challenge. This book aims to equip readers with skills and techniques which will help them to address this challenge. It does so by stressing the value of abstract system models which can be analysed and tested before an expensive commitment is made to a particular design strategy. The book enables the reader to understand the role and nature of abstract models as well as gain practical experience in their creation.

In order to permit machine-supported analysis, system models must be formulated in a well-defined notation. In this text, we use a formally defined language called VDM-SL (the Vienna Development Method Specification Language). The Vienna Development Method is a collection of techniques for developing computing systems from models expressed in the language. Since its origin in an industrial environment, VDM has become one of the most widely used of a class of techniques known as *model-oriented formal methods*. The language VDM-SL was recently standardised by the International Organization for Standardization (ISO). Although VDM-SL is used as a teaching medium in this text, the principles taught apply equally well to other model-based formal methods such as B, RAISE and Z.

In this book we take a pragmatic approach to the use of formal methods. We aim to illustrate the concepts and techniques used in VDM without overwhelming the reader with mathematics. Unlike most teaching texts on formal methods, this book does not treat formal refinement or formal proof. Instead it focuses on the construction of abstract and formal models for a range of computer systems. Mastering the construction and validation of abstract models is in our view a prerequisite for entering the world of verification.

This book is unusual in two other respects. First, the majority of the examples presented are inspired by models developed in industrial projects over recent years.

We believe that examples grounded in industrial practice provide motivating illustrations of the fact that this technology can be used for development of real systems and not simply for stacks and vending machines. Second, the skills to develop abstract models can only be acquired through practice. Throughout the text, the use of an industrial tool is encouraged, in order to develop the reader's intuitive grasp of the reality of modelling.

Robust and appropriate tool support is essential for industrial application of modelling technology, so hands-on experience is stressed throughout this book. Readers will gain the most benefit if they use the freely available VDMTools tool set[1] introduced in Chapter 3. It is possible to carry out the exercises without tool support, but this will not give the reader an appreciation of what can be expected from such tools.

The web site accompanying this book (`http://www.vdmbook.com`) provides all the example models presented in the book, additional models used for exercises and links to the full VDM-SL language manuals. This book uses the (ASCII) interchange syntax of VDM-SL rather than the mathematical syntax which is used in most some of the older texts. It is our experience that this notation presents less of a barrier to the novice who does not have experience in mathematical logic.

The subset of VDM-SL used in this book includes facilities for state-based specification. However, readers already familiar with VDM will notice that the tutorial content is biased towards a functional modelling style. The functional style provides an environment in which type constructors and operators can be covered without the distraction of operation syntax, side-effects and access restrictions to external variables. Once the elements of data and functional modelling have been covered, the 'state and operations' paradigm is introduced. The text omits a discussion of explicit operations, because it is our experience that those who learn abstraction skills within the language subset we have chosen can learn to use explicit operations very easily, on the basis of experience from programming languages.

## Using this book

This text is aimed at software engineers who wish to investigate how the use of models can improve the software development process, and at university students studying software engineering or computing science. The material in

[1] The URL is `http://www.vdmtools.jp/en` or via the book's support pages at `http://www.vdmbook.com`.

the book has been used successfully on industrial training courses, undergraduate and postgraduate level [Larsen&08]. The text is designed with independent study in mind, and the support materials and discussion lists accessible via `http://www.vdmbook.com` provide further support for readers not studying this material as part of a university course.

No formal mathematical background is assumed, but the authors find that students gain most benefit when they have some familiarity with programming and with the realities of software development. In the university context, students' experience of constructing a large piece of software in a group project, sometimes suffering serious setbacks during integration, has been found to provide a valuable motivating lesson.

The objective of this book is to bring readers to a point where they are able to read, write and analyse formal models of computing systems using VDM-SL and have an understanding of the kind of problems to which these techniques can be applied cost-effectively.

The book is divided into four parts. Chapters 1 to 3 form the introductory material. The first two chapters motivate and introduce the notion of modelling using a formal language and indicate a systematic approach for using this kind of technology. On reaching the end of Chapter 2, the reader will have seen most of the elements of VDM-SL covered in the book. Chapter 3 introduces VDMTools Lite, the tool support provided for the tutorial material. Chapters 4 to 8 form the core of the book, covering the use of logic, basic data types, type constructors and functions in constructing models. Each of these chapters contains a description of the requirements for an application for which a model is developed, introducing each modelling construct in VDM-SL as it is needed. Chapters 10 to 12 are concerned with the use of models in practice, in particular validation techniques, the representation of persistent state and dealing with large-scale system models. The final part of the book examines the introduction and use of formal modelling in the commercial context. Chapter 13 discusses the introduction of modelling technology in the industrial environment. The appendices include a language guide for the subset of VDM-SL used in the text and supported by VDMTools along with solutions to exercises.

Readers using the book as an introduction to formal modelling can follow the text in the order in which it is presented. Practising software engineers may prefer to read Chapter 13 after Chapter 1 for a consideration of the costs and benefits of applying the techniques covered in depth in the remainder of the book.

Although the book is intended to embody a single course in formal modelling, Chapters 1 to 9 would be suitable for a course covering modelling only. The material in Chapter 9 on recursive structures is slightly more demanding than the preceding chapters. Chapters 10 to 13 could be used in a second and more

advanced course including a significant assignment in which students can explore the construction and analysis of a model.

Exercises are included in the flow of the text and should be attempted as they are encountered. More substantial exercises are normally included at the end of each of the central chapters. Those exercises marked with a star ($\star$) are more demanding than the average. It is our experience presenting this material that instructors are asked for large numbers of small exercises which increase familiarity with the language. Often, lecturers also require more demanding exercises for the most enthusiastic and capable students. Teaching material including samples of lecture notes, slides and exercises of both these kinds will become available from the World Wide Web Page at `http://www.vdmportal.org/`.

> The production of a formal model is much less straightforward than a textbook might lead one to suppose. By presenting particular models as solutions to problems, we do not intend to imply that they are the only, or even the best, solutions. In addition, the developer of a model runs into many "dead ends" before reaching a good solution. We are unable to present this process in all its detail in this volume, but we do record some aspects of our practical experience which we feel would be most helpful. This is done in distinguished boxes of text such as this.

## Developments since the first edition

When we published the first edition of this book ten years ago, our aim was to lower the barrier to using formal modelling techniques that were still seen as forbidding, specialised and expensive. By emphasising tool support, and by using examples derived from industry applications, we hoped to encourage readers to apply abstraction and modelling principles in a wide range of applications. Although the underlying aim remains unchanged, this second edition takes account of the significant developments in formal modelling, tool support, industry application and experience gained in teaching and training in formal methods in the last decade.

Tool support for the first edition took the form of a limited-capability version of the VDM Toolbox, then being developed by IFAD A/S in Denmark. The Toolbox technology was subsequently acquired and extended by CSK Systems in Japan, following their successful application of VDM to a major financial back-office system (TradeOne, described in Chapter 13). The tool that accompanies this second edition takes advantage of CSK's developments. VDMTools Lite now supports the full VDM-SL language instead of just the subset covered in the book. Restrictions

on the size of model have been removed. The full functionality of the tool set is now available, with the exception of the application programmer interface, the dynamic linking facility and the C++ code generator.

Since the first edition was published, both of us have worked in the software industry, in areas as diverse as business development and design of binary translation technology. The experience has reinforced our view that formal modelling technology must work with existing development tools and processes, rather than supplant them, if its benefits are to be realised. We have also become more aware of the trade-off between the effort expended in analysing models and the insights gained by doing so. As a result, we have added material on recent industrial applications of VDM (Chapter 13) and updated the industry-based examples throughout the book. Our experience teaching formal modelling to practitioner engineers and to university students has emphasised the importance of abstraction as a core skill in modelling. We reinforce this point with abstraction lessons in each of the core chapters.

This book concentrates on the core VDM-SL notation and the fundamentals of abstraction and rigorous reasoning about system models. Over the last ten years, the capabilities of the VDM formalism itself have been extended to encompass development of concurrent and object-oriented systems, and real-time and distributed systems. We have added pointers into these more specialised and advanced applications in the expanded Chapter 13.

## Acknowledgements

We would like to thank our former professors Cliff Jones and Dines Bjørner for introducing us to the subject of formal methods and to VDM in particular. Our thanks also go to our supportive and patient editor David Tranah from Cambridge University Press and to our colleagues at CSK in Japan, especially Shin Sahara.

Many colleagues have provided valuable comments on drafts of the book in both editions: Sten Agerholm, Bernhard Aichernig, Mo Ajmal, Paul Ammann, Nick Battle, Peer Bonnerup, Carsten Breum, Jeremy Bryans, Hanne Carlsen, Tim Clement, Ian Cottam, Lionel Devauchelle, Albert Esterline, Kelsey Francis, Brigitte Fröhlich, Anne Haxthausen, Niels Kirkegaard, Ole Bjerg Larsen, Janusz Laski, Yves Ledru, David Morgan, Paul Mukherjee, Anne Berit Nielsen, Erik Toubro Nielsen, Takahiko Ogino, José Nuno Oliveira, Lars Toftegaard Olsen, Richard Payne, Jan Storbank Pedersen, Marie-Laure Potet, Abd-El-Kader Sahraoui, Paul Smith, Vincent Stephan, Elliot Sullivan, Marcel Verhoef, Henrik Voss and Mark Wigmans.

The first edition was published in a Japanese translation by the Iwanami Shoten publisher in 2003. We are particularly grateful to our skilled translators: Keijiro

Araki, Takahiko Ogino, Shin Sahara and Makoto Someya. Their work has been a major stepping stone towards the adoption of formal modelling technology in Japan.

Many colleagues were kind enough to use the first edition in courses on formal modelling, and to provide us with comments and corrections. We are particularly grateful to Bernhard Aichernig, V. S. Alagar, Pascal André, Keijiro Araki, Andreas Bollin, Dan Cristea, Albert Esterline, Pascal Fenkam, Mats Heimdahl, Keith Hopper, Zarinah Mohd Kasirun, Janusz Laski, Yves Ledru, Peter Lucas, Michael Lutz, Dominique Mery, Farid Mezidane, Pascal Molli, David Morgan, Tony Moynihan, José Nuno Oliveira, Kasi Periyasamy, Steve Riddle, Angela Shiflet, Ivor Spence and Robert Topp. Our thanks are also due to the hundreds of students who have participated in our courses at Newcastle University and the Engineering College of Aarhus.

We gratefully acknowledge the support of the European Union Framework 7 IST Project "DEPLOY" on industrial deployment of advanced system engineering methods and the Erasmus programme for supporting our continued collaboration on the learning and teaching of formal methods.

As always, we reserve our deepest thanks for our closest friends and families who allowed us, once again, a decade on, to take advantage of their patience.

*John Fitzgerald*
*Peter Gorm Larsen*
Aarhus, Denmark