FORMAL METHODS FOR EMBEDDED DISTRIBUTED SYSTEMS

Formal Methods for Embedded Distributed Systems

How to master the complexity

Edited by

Fabrice Kordon
Université P. & M. Curie,
France

and

Michel Lemoine
ONERA Centre de Toulouse,
France

KLUWER ACADEMIC PUBLISHERS
NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 1-4020-7997-4
Print ISBN: 1-4020-7996-6

Created in the United States of America

Visit Springer's eBookstore at: http://www.ebooks.kluweronline.com
and the Springer Global Website Online at: http://www.springeronline.com

Contents

Preface

The development of any Software (Industrial) Intensive System, e.g. critical embedded software, requires both different notations, and a strong development process. Different notations are mandatory because different aspects of the Software System have to be tackled. A strong development process is mandatory as well because without a strong organization we cannot warrantee the system will meet its requirements. Unfortunately, much more is needed!

- The different notations that can be used must all possess at least one property: formality.
- The development process must also have important properties: a exhaustive coverage of the development phases, and a set of well integrated support tools.

In Computer Science it is now widely accepted that only formal notations can guarantee a perfect defined meaning. This becomes a more and more important issue since software systems tend to be distributed in large systems (for instance in safe public transportation systems), and in small ones (for instance numerous processors in luxury cars). Distribution increases the complexity of embedded software while safety criteria get harder to be met.

On the other hand, during the past decade Software Engineering techniques have been improved a lot, and are now currently used to conduct systematic and rigorous development of large software systems. UML has become the *de facto* standard notation for documenting Software Engineering projects. UML is supported by many CASE tools that offer graphical means for the UML notation. Moreover CASE tools are able to generate code templates and support round trip engineering between class diagrams and program code. However, all case tools and more generally the Software Engineering techniques used in practice do not well support the early phases of software development. They still lack analysis and validation means for requirements and design specifications which are easily connected to the implementation phase, even when UML is used in conjunction with some process, such as the RUP or Fusion (see references further).

Formal techniques have undergone a steep development during the last years. Based on formal foundations and deep theoretical results, methods and tools have been developed to support specification and design of software systems. Model-based and algebraic specifications, abstract state machines, CSP and CCS, temporal logics, rewriting techniques, finite automata, model checking and many other formal specification and verification techniques have been applied to non-trivial examples and are used in practice e.g. for the development of safety critical systems. Nevertheless Software Engineering practitioners claim that formal notations are not scalable enough.

This main drawback was the topic of a Dagstuhl seminar organized in May 2001 (seminar # 01221) by S. Jähnichen (Technical University of Berlin and Fraunhöfer Institute, Berlin-Ge), J. Kramer (OPEN University, London-UK), M. Lemoine (ONERA, Toulouse-F) and M. Wirsing (Technical University of München, Munich-Ge). This workshop gathered a set of selected academic people to discuss the following problem: "Can Formal Methods Cope with Software-Intensive Systems?" For these discussions, the "BART case study" (see Chapter 1) was suggested as a common basis to check how far the formal methodologies promoted by the represented teams could be efficiently applied.

Based on works presented in this seminar, a few presentations have been selected. They are the basis of the following chapters. The result is the book you are handling right now.

We would like to thank all the authors for the work they accomplished for this book. We also want to thank M. De Jong and C. Zitter, from Kluwer Academic Publisher, for their continuous help and encouragements all over the process.

F. KORDON, M. LEMOINE

Contributing Authors

François Bréant (LIP6-CNRS - Univ. Paris 6, France). After he got his Ph.D. in Computer Science in 1993 from the University P. & M. Curie (Paris, France), he developed experiments and simulations for virtual environments at the NASA Ames research center. He worked on tool cooperation for embedded systems using CORBA technology at Mentor Graphics (European project SOOM). Then, he worked on code allocation and partitioning and on data allocation and transfer for a compiler targeting a configurable multiprocessor DSP at Improv Systems Inc. He also worked on testbench environments based on a transactional model for the test of Verilog designs applied to attachment protocols at Perfectus Technologies Inc. He is currently engineer at the LIP6. His interest fields include compilers, parallel systems, simulation and verification.

Jean-Michel Couvreur (LaBRI-CNRS - Univ. Bordeaux, France). Jean-Michel Couvreur received a Ph.D. in Computer Science in 1990 from the University P. & M. Curie (Paris, France). He is currently an Associate Professor at the University Bordeaux I and he is researcher at the LaBRI Laboratory. He brings expertise in the development of algorithms for symbolic verification, and in methods for model checking using various linear temporal logics.

Adriaan De Groot (KU Njmegen, The Netherland). Adriaan de Groot studied computer science in Nijmegen and is currently doing his Ph.D. on Requirements Engineering there. He uses UML and PVS as tools to help understand natural-language specifications, leading to improved specifications. He programs for the KDE project in his spare time, and laments the lack of requirements and specifications in open source software projects.

Gary Fuehrer (Univ. of New Mexico, USA). Gary Fuehrer received a BS degree in mathematics at the University of New Mexico in 1990, a few years after beginning his career as a Windows software developer. He now serves in a consultant capacity on the latest technologies for the Windows platform. Mr.

Fuehrer is currently finishing an MS degree in computer science, also at the University of New Mexico.

Gervais Gaudière (ENAC, Toulouse, France). Gervais Gaudiere got an engineering degree in computer sciences since 1995. After 4 years of dependability studies at the French National Navigation Study Center (CENA), he teaches since 1999 Software Engineering, Quality, project management and design methods (formal and object oriented) at the French National Civil Aviation School (ENAC). He is also a class inspector of a computer sciences engineering class since 1999.

Frédéric Gilliers (LIP6 and SAGEM S.A., France). Frédéric Gilliers is currently a Ph.D. student at the university P. & M. Curie (Paris, France) where he is currently working on code generation from high-level formal specifications (the **L*f*P** language presented in this book). His research interests involve automatic generation of distributed programs from formal languages, and deployment techniques for distributed systems.

Martin Gogolla (Univ. Bremen, Germany). Martin Gogolla got his Ph.D. in Computer Science from Braunschweig Technical University in 1986 and his Habilitation in Computer science in 1993. Since 1986, he his a senior researcher at Braunschweig Technical University and for a professorship in 1994 at the Bremen University. he has numerous publications in journals and conference proceedings; published three books; organized and co-organized several workshops and conferences and is an expert referee for journals and conferences.

His research interests are: Algebraic specification; formal semantics of languages, especially data base languages; formal methods in information systems design; Object-oriented design and languages.

Jozef Hooman (KU Njmegen, The Netherland). Jozef Hooman is a senior lecturer at the University of Nijmegen since 1998. Before, he was a lecturer at the Eindhoven University of Technology. There he also received a Ph.D. degree on a thesis entitled "Specification and Compositional Verification of Real-Time Systems" which appeared as LNCS 558. The central research theme of Jozef Hooman is the formal specification and compositional verification of distributed systems, with special emphasis on real-time and fault-tolerance. Since 1993 he is an intensive user of the interactive theorem prover PVS. Recent research concerns the use of PVS for the formal specification and analysis of requirements, the study of an industrial software architecture, the verification

of concurrency control protocols, and the combination of UML and formal methods.

Deepak Kapur (Univ. of Nebraska at Omaha, USA). Deepak Kapur got his Ph.D. in computer science in 1980 from MIT. After graduation, he worked at GE R&D Center, Schenectady, New York, until 1988. In 1988, he was appointed full professor in the Department of Computer Science at the University at Albany. He also founded the Institute for Programming and Logics, and he served as its director till Dec. 1998. He won the excellence in research award at SUNY in 1998. In Jan. 1999, he moved from SUNY, Albany, to chair the Computer Science Department at the University of New Mexico. Kapur has published over 150 papers, has edited three books, and has served on program committees of many international conferences. He serves on the editorial board of four journals, including the Journal of Automated Reasoning, of which he serves as the editor-in-chief.

Fabrice Kordon ((LIP6-CNRS - Univ. Paris 6, France). Fabrice Kordon received a Ph.D. in Computer Science in 1992 from the University P. & M. Curie (Paris, France) and an Habilitation from this same university in 1998. He is currently Professor of computer science at the University P. & M. Curie were he manages a team involved in prototyping techniques for distributed systems (modelling, formal verification using Petri nets, automatic program generation). Since 1994, his team distributes on Internet `CPN-AMI`: a Petri net based CASE environment dedicated to formal verification of distributed systems which is used for teaching and research in many research institutes. He participates in the Program committee of several conferences dedicated on formal methods and software engineering. He was General co-chair for the IEEE Rapid System prototyping workshop in 2000 and 2001 prior to be program co-chair in 2002 and 2003.

Michel Lemoine (ONERA/CdT, France). Michel Lemoine was graduated in Computer Science from University Paul Sabatier in 1971. He then got a position of full-time Senior Researcher at ONERA (Office National d'Études et de Recherches Aérospatiales) in Toulouse. ONERA being strongly coupled with ENSAE (École Nationale Suipérieure de l'Aéronautique et de l'Espace) he is also professor (part-time) of Computer Science. His main domains of interest are relative to design techniques (mainly Object-Oriented), to formal methods, and to Requirements Engineering.

ONERA being in charge of many early stages of French aircraft designs he is experimented, from a System Engineering point of view, most of the new emerging techniques in Computer Science at an industrial level.

Isabelle Mounier (LIP6-CNRS - Univ. Paris 6, France). Isabelle Mounier received a Ph.D. in Computer Science in 1994 from the University P. & M. Curie (Paris, France). Since 1998, she is an Associate Professor of Computer Science at the University P. & M. Curie. Her research interests concern verification of properties of complex systems by combining reduction, abstraction and model checking techniques.

Emmanuel Paviot-Adet (LIP6-CNRS - Univ. Paris 6, France). Emmanuel Paviot-Adet received a Ph.D. in Computer Science in 1995 from the University P. & M. Curie (Paris, France). He is currently Associate Professor of computer science at the University R. Descartes (Paris, France) and he is researcher at the LIP6 Laboratory of the University P. & M Curie (Paris, France). His research interests involve verification of complex systems using model checking.

Denis Poitrenaud (LIP6-CNRS - Univ. Paris 6, France). Denis Poitrenaud got is Ph.D. in Computer Science in 1996 from the University P. & M Curie (Paris, France). Since 1997, he is an Associate Professor of Computer Science at the University R. Descartes (Paris, France) and he is researcher at the LIP6 Laboratory of the University P. & M Curie (Paris, France). His research fields are model checking techniques and theoretical aspects of formal languages.

Dan Regep (LIP6-CNRS - Univ. Paris 6, France). Dan Regep is currently a Ph.D. student at LIP6 where he is works in the area of Rapid Prototyping of safe Distributed Systems. His thesis focuses on **L*f*P**, a language for Prototyping; his main contribution was the development of the **L*f*P** software architecture style.

His research interests are mainly oriented on Formal description and modelling of Software Architecture, Evolutionary Prototyping and Automatic Code Generation techniques.

Bernhard Schätz (Tech. Univ. München, Germany). After his graduation in Computer Science, he worked on formal foundations of models for reactive systems in the project "Methods and Tools for the Use of Parallel Computer Architectures" of the German Research Community (DFG). 1998 he received his Ph.D. in Computer Science at the TU München. Currently, he works as senior researcher at the chair of Prof. Broy for Software & Systems Engineering, TU München, focusing on the application of formal techniques in the engineering process. His work aims at the construction of CASE tools for a model-based software engineering process for embedded systems. Results of his research are incorporated in the development of AutoFocus.

Grégoire Sutre (LaBRI-CNRS, France). Grégoire Sutre received a Ph.D. in Computer Science in 2000 from the École Normale Supérieure (Cachan, France). Since 2001, he is a full-time researcher at the Computer Science Research Lab of Bordeaux (LaBRI, France). His research interests are in the automatic verification of complex systems, in particular infinite-state models and C programs, using abstration/refinement and symbolic techniques.

Victor Winter (Univ. of Nebraska at Omaha, USA). Victor Winter got his Ph.D. in computer science in 1994 from the University of New Mexico. After graduation, he worked at Sandia National Laboratories as a member of the High Integrity Software (HIS) program. In 1999 he subsequently became the principal investigator of that program. In 2001, Dr. Winter accepted a position as an assistant professor in the computer science department at the University of Nebraska at Omaha. Dr. Winter is a primary developer of a program transformation system called HATS.

Paul Ziemann (Univ. Bremen, Germany). Paul Ziemann got his Diploma in 2001. He is now a researcher in the DFG-project "UML-AID" (Abstract Implementation and Documentation with UML) at the Bremen University.

His research interests are: formal semantics of UML and OCL; extension of OCL with temporal logic; graph transformation.

Introduction

F. Kordon, M. Lemoine

1. The "Traditional" development approach

Let us consider the traditional development approach for computer-based systems. This approach can be represented by the well known "V" software life cycle model, as shown in Figure I.1.

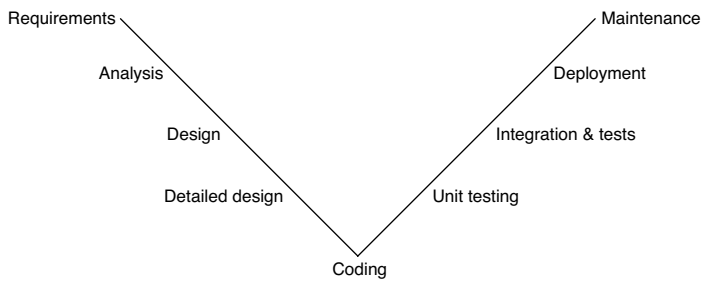

Figure I.1. The traditional V life cycle.

This software life cycle put emphasis on a tactical view of the development activity: the one that aims at producing "something" on time.[1] The first part of this approach is top-down. After having defined the requirements, their are analyzed. A solution is then elaborated: general design before the detailed one.

The coding activity corresponds to the end of the first part in a product development. Programs are then tested (unit per unit), integrated, i.e. assembled, tested again (component by component up to the complete application). Deployment to the clients' sites is possible and maintenance can start.

2. What is covered in this book

There are thousands of possible interpretations of this software life cycle (this is the same for other life cycles too). However, developing software becomes a much more delicate task since it comes to distributed applications, or embedded, or both. In that context, formal-based methods and techniques bring an increased level of security. The drawback is their complexity to set up and use.

This book is dedicated to the presentation of some techniques to be used in the context of distributed and/or embedded systems. Since formal techniques rely on models, i.e. different descriptions of the system to be designed, the presented techniques are located in the first part of software development, as shown in Figure I.2.

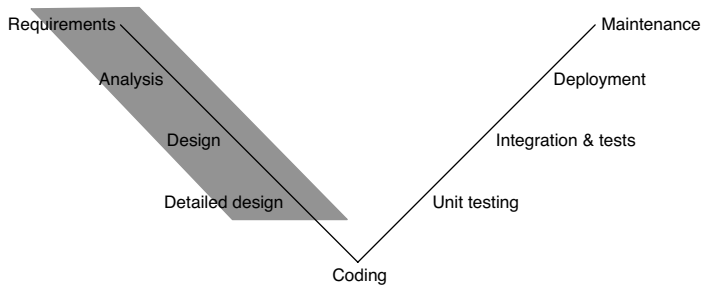

Figure I.2. Objectives of this book in the V-cycle.

We do not deal with implementation aspects. Since code generators from specifications (like with HOOD or, more recently, with UML) are now widely accepted and used (event if the generated code requires to be enriched), it is obvious that the use of such tools is required for coding. Otherwise, manual implementation may lead to a derive due to misunderstanding of the initial system (or the integration of implementation hypotheses). But this is again out of the scope of this book.

3. Organization of chapters

The problem of complex systems is to provide a safe implementation, and, prior to this, a safe design. For distributed and/or embedded systems, this mainly asks the following questions:

- Is the system complete or, in other words, does it fulfill all its requirements?

- Is the system behavior understood or, in other words, can the system behave erroneously?

This is why the book is divided in two parts. A first one deals with "Conceptual" aspects (i.e. understanding of the requirements, appropriateness of the design, etc.) and"behavioral" aspects.

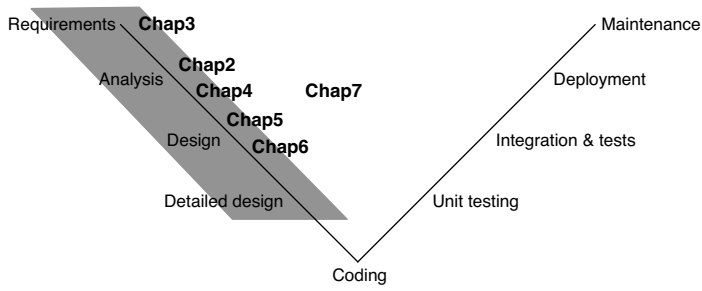

Figure I.3. Where chapters refer to in the V cycle.

Figure I.3 approximately locates which phases of the software life cycle are considered by the methods presented in the following chapters.

How to read this book? We have decided to split the book into 4 main parts.

Part I is mainly concerned with the BART Case Study. It contains only one chapter: the BART case study that is a common base to all chapters. Indeed all the presented techniques have used it as their application. Proceeding this way will allow the reader to identify where, in software life cycle, they can be (re)used, and of course to compare them, from a notational point of view.

The reader is recommended to read the first chapter as carefully as she/he can. Indeed you will find a Requirements Document as the Industry is used to deliver. Rigorous needs corresponding to end users' wishes, strong constraints corresponding to well defined obligations the system must meet are expressed in natural language. Moreover the authors of the Requirements Document have presented it as numbered paragraphes. This numbering can be used to build a traceability matrix between informal requirements and formal ones, as they are introduced in the other chapters.

Part II is intituled "Building and Validating Conceptual Aspects".

- Chapter 2 is the description of a specification language that helps developing and analyzing reactive systems that must meet constraints such as safety and optimality. Moreover, the specification can be transformed into ML codes, and then run as prototypes. Transformations can be

proved as safe, allowing then to refine it and to arrive to a more efficient implementation.

- Chapter 3 is much more concerned with Requirements Engineering. It is shown how an Informal Requirements Document can be transformed into semi formal and formal models. An incremental process is suggested to both specify rigourously the system as set of static models, and then to validate them formally. UML as semi formal notation and = Z as formal one are homogeneously integrated in a specific Evolutionary process.

- Chapter 4 is much more classic. It refers to one possible way of using UML for specifying reactive systems, and to the use of some formal notation to check with a theorem prover how statecharts are right.

Part III intituled "Building and Validating Operational Aspects" contains 2 chapters.

- Chapter 5 shows how some UML design can be checked efficiently using OCL as specification language, and USE as support tool. The result is a set of scenarios that can be played, and replayed as long as necessary, to convince end users of the quality of the formal specification of the system under consideration.

- Chapter 6, even rather similar to Chapter 5, puts the emphasis on a formal graphical Architectural Description Language, which allows both to integrate static and dynamic aspects of any reactive system. As main result, the built formal models can be both formally analyzed, and as well prototyped.

Part IV is intituled "Methodological Aspects". It contains one and only one chapter.

- Chapter 7, as expected in a methodological part, presents the AutoFocus approach, which is real methodology. In other words a dedicated process is introduced, with some notations, and some supporting tool. It can be considered as a summary of what could be done if we expect to develop a safe and secure system.

The last chapter introduces the conclusion, which is nothing more than the view of the authors, regarding the techniques and methodology used to specify the BART system.

Notes

1. Other"life cycles" put emphasis on maintenance (spiral), on components, on knowledge reuse.... Their description is out of the scope of this book.