Int J Softw Tools Technol Transfer (2012) 14:631–638
DOI 10.1007/s10009-012-0259-5

INTRODUCTION

Relating software validation to technology trends

Zhiming Liu · Abhik Roychoudhury

Published online: 5 September 2012
© Springer-Verlag 2012

Abstract Large scale software engineering is undergoing substantial shifts due to a combination of technological and economic developments. These include the prevalence of software for embedded systems, global software development across geographically distributed teams, the technological shift towards multi-core platforms, and the inevitable shift towards software being used as a service. In this overview article, we discuss some of the challenges that lie ahead for software validation, due to such technological developments. In particular, we provide a brief introduction to the papers appearing in this special issue, many of which specifically focus on validation of software running on real-time embedded systems.

Keywords Model-based development · Abstraction · Refinement · Model transformation · Verification · Tool support

1 Introduction

Software validation refers to the art of checking whether a software works as intended. Validation is a general term which captures various modes of checking—testing, analysis and verification being some of the common and well-known forms of software validation. Testing is the most commonly and widely used validation technique, which is employed on any program. Formal verification refers to a check of a given program property often against a model of the program; such a model could have been extracted from the program (typically by abstracting the program behavior). Finally, program analysis refers to inference of program properties typically by a direct analysis of the program (rather than its model).

The social importance of software validation is enormous. Today, more and more functionalities in our daily life are controlled by software. Due to the overwhelming growth of embedded systems, software-controlled devices are ubiquitous—automotive control, avionics control and consumer electronics being prominent application domains. Many of these software are safety-critical and should be validated extensively. This accentuates the importance of software debugging and validation in our daily lives.

The economic importance of software debugging is no less significant. Industrial studies on quality control of software have indicated high defect densities. Ebnau in an ACM Cross talk article[1] reports case studies where on an average 13 major errors per 1,000 lines of code were reported. These errors are observed via slow code inspection (at 195 lines per hour) by humans. So, in reality, we can expect many more major errors. Nevertheless, conservatively let us fix the defect density at 13 major errors per 1,000 lines of code. Now consider a software project with 5 million lines of code (the Windows Vista is 50 million lines of code, so 5 million lines of code is, by no means, an astronomical figure). Even assuming a linear scaling up of defect counts, this amounts to at least

$$(13 \times 5{,}000{,}000/1{,}000) = 65{,}000$$

major errors. Even if we assume that the average time saved to fix one error using an automated debugging tool as opposed to manual debugging is 1 h (this is a very modest estimate, often fixing a bug takes a day or two), the time saved is 65,000

Z. Liu (✉)
UNU-IIST, P.O. Box 3058, Macau SAR, China
e-mail: lzm@iist.unu.edu

A. Roychoudhury
National University of Singapore,
13 Computing Drive, Singapore 117417, Singapore
e-mail: abhik@comp.nus.edu.sg

[1] http://www.stsc.hill.af.mil/crosstalk/1994/06/xt94d06e.asp.

man hours = 65,000/44 = 1,477 work weeks = 1,477/50 = 30 man years. Clearly, this is a humongous amount of time that a company can save, leading to more productive usage of its manpower and saving of precious dollar value. Assuming an employee salary of 50,000 USD per year, the above translates to 1.5 million USD savings in employee salary simply by using better debugging tools. A much bigger savings, moreover, comes from the customer satisfaction. By using automated debugging tools, a software development team can find more bugs than via manual debugging leading to increased customer confidence and enhanced reputation of the company's products.

Apart from the obvious dependence of our society on software, and hence the importance of software validation, we note that software engineering has been undergoing substantial shifts due to various technological and economic reasons. One of the established shifts presently is the gradual movement from uniprocessor to homogeneous multi-processor platforms, also popularly known as multi-cores. The shift towards multi-cores is largely driven by excessive heat and power dissipation in modern day uniprocessors. Multi-cores allow for greater performance with lower energy consumption. This is all the more important with the prevalence of mobile devices (such as smart-phones), and the wide-ranging applications (popularly known as "apps") that run on these devices. We note that energy efficiency is of supreme importance in order to extend the battery life of mobile devices. As a result, such devices have increasingly adopted multi-core platforms. This brings us to the technical aspects of writing reliable software for multi-core platforms. Another significant shift is towards web-based service systems on distributed networks and cloud computing platforms. Indeed many of the papers selected for this special issue focus on such technical aspects. In the following section, we discuss the relevance of the papers selected for our track with some of the important technology trends in software engineering.

The motivation of this special issue is from the special tracks that we organized for the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation in 2006 [1], 2008 [2] and 2010 [3]. In this introduction paper, we first summarize in Sect. 2 the understanding of model-based techniques of software design and validation that has been built through the presentations and discussion at these special tracks. Then, in Sect. 3, we give an account of the contribution by selected papers presented in the special tracks. Section 4 presents an overview of the papers that consist of this special issue. Section 5 concludes the introduction paper with a discussion.

2 Model-based design and validation

The common theme of these special tracks is techniques and tools in model-driven design and verification, with an emphasis on developing countries. The key works presented here are about the inherent complexity of modern software systems being mastered by building component-based models at different levels of abstraction. The papers also deal with various techniques and tools being used for analysis and verification of safety and performance properties.

2.1 Increasing complexity of modern software

Software engineering was born with and has been growing because of the "software crisis" that refers to the difficulty of writing correct, understandable, and verifiable computer programs. The root of the crisis is the inherent complexity of software, and the major cause of the complexity is that the machines have become several orders of magnitude more powerful [4] within decades. Software complexity is characterized in terms of four fundamental attributes of software [5–8]

1. the complexity of the application domain,
2. the difficulty of managing the development process,
3. the flexibility possible to offer through software, and
4. the problem of characterizing the behavior of software systems.

The first attribute focuses on the difficulty of understanding the application domain (by the software designer in particular), capturing and handling the ever changing requirements. The second concerns the difficulty to define and manage a development process that has to deal with changing requirements for a software project that involves a large team comprising of software engineers and domain experts possibly in different geographical places. The third attribute characterizes the wide range of possible designs and implementations that have conflicting features for a given application. The design decisions also have to deal with changing requirements and aiming to achieve the optimal performance to best support the requirements of different users. To handle this challenge, rigorous techniques for incremental and compositional design are required for systematic refinement from model of requirements to models of designs and from models of design to implementations. Interactive tool support to model transformations is required to help the developer to make design and implementation decision and automate the steps of model refinements. The aim is to support the principle of *correct by construction* or *design*. The final attribute of software complexity pinpoints the difficulty in understanding and modeling the semantic behavior of software for software analysis, validation and verification for correctness, and reliability assurance. Verification and validation techniques and tools must go hand in hand with the model transformations along a tailored development process.

The software industry is now facing an even greater scale of complexity with modern software-intensive systems [9], due to the new technology trends discussed in the previous section. We see the application of these new technologies in our everyday life, such as in transportation, health, banking and enterprise applications. These systems provide their users with a large variety of services and features. They are becoming increasingly distributed, dynamic and mobile. Their components are deployed over large networks of heterogeneous platforms and thus the inter-operability of the distributed components becomes important. Consider e-Health systems, for examples, that consist of back-end networks of servers that collect data from and provide services to mobile and programmable devices. The mobile devices and the servers are connected through different kinds of communication networks. In addition to the complexity of functional structures and behaviors, modern software systems have complex aspects concerning organizational structures (i.e., system topology), adaptability, interactions, security, real-time and fault-tolerance.

A complex system is open to total breakdown, and the industry suffers from the long lasting software crisis.[2] Consequences of system breakdowns are sometimes catastrophic and very costly, e.g., the famous Therac-25 Accident 1985–1987 [10] and Ariane-5 Explosion in 1996 [11]. Discussion in the previous section shows that finding bugs in programs and fixing them are costly, and development of *verified software* is a "grand challenge" [12,13].

2.2 Formal techniques in model-based development

The model-driven approach, known as model-driven architecture (MDA) [14] among the practical software engineering community, proposes as the key engineering principles for mastering software complexity and improving dependability and predictability through building system models in all stages of the system development [8]. The approach enjoys the following key advantages [8,15,16].

1. The system architecture is decomposed into interacting components. This component-based architectural is to support compositional design and compositional assembling as well as verification and validation. This is a key engineering principle of divide and conquer. Also, formal techniques and tools for verification can only be feasibly applied to software components that are small enough. Components also encapsulate data and functionalities, thus provide localization of faults and prevent from failure propagation.
2. The architectural components are specified by their interface contracts with fully explicit context of dependencies so that components can be deployed independently and are yet subject to third party composition.
3. Models of components are constructed or automatically generated at different levels of abstraction during all stages of the development process. Thus, abstraction techniques for information hiding is essential for focusing on a design concern at a time and for supporting incremental design.
4. At any level of abstraction, the model of a component is an integration of a set of consistent models of different viewpoints, including the data types and class structures, the local data functionality of services, the interaction protocols and the dynamic behavior of the components. A model of a component at a design level or lower must be validated to ensure that the properties of the interface contract of the component are respected.

The models are required to be analyzable and verifiable and the process needs to be repeatable. Thus, formality or rigor has to be applied. For a formal model-driven method to be effective, it must provide a body of techniques and an integrated suite of tools for model construction, validation and transformation [16]. Moreover, a number of modeling notations are combined for the specification of different concerns and viewpoints of the system [17,18]. As examples, we may mention Hoare Logic for service (or component) local data functionality specification and refinement, the event-based notation such as CSP [21,22] and CCS [23] for interaction and communication between components, and the I/O automata-based models [24] of dynamic behavior components and their interfaces [25]. However, the challenge now is how to apply these notations and techniques in systems development of industrial scales. We understand that the key to a solution is a model-driven process for component-based architecture modeling, refinement and verification that can be tailored to allow these notations, and tool techniques to solve the problems occurring in activities at different phases of the process. This is what we mean by linking of theories and techniques. The combination depends on the definition of a unified semantic theory for the notations. The unified semantic is to underpin the development of tools for analysis, transformation and verification of models represented in these notations and to justify the correct and consistent use of these tools on the models of software artifacts in different stages of software development [27]. Techniques and tools for model analysis and verification are also used in combination with those for abstraction and refinement [26–28].

3 Contributions from special track

We discuss how some papers presented in the special tracks contribute to the above understanding. In general,

[2] Booch even calls this state of affairs "normal" in [6].

the presentations cover techniques and tools applicable to models at different levels of abstraction and models on different aspects of software. Techniques and tools include model transformations between models at different levels of abstraction and consistent model integration to support design, model checking, and logic and algebraic reasoning. In terms of programming paradigms and system architectures, there papers on embedded systems, multi-core, and web-services. However, the theme of discussions was possibly best reflected in the paper by Liu et al. [27] on Harnessing Theories for Tool Support. It is about the model-driven development process where different models are constructed at different stages. It presents an approach to support semantic preserving model refinement through interactive model transformations. These model transformations generate properties of models as proof obligations so that different verification tools can be called for the verification of these properties. In this way, theories, techniques and their tool support for specification, design and verification are seamlessly linked. The paper emphasizes on component-based architecture modeling for divide and conquer, and the use of multi-dimensional modeling for separation of design concerns.

3.1 Categorization of papers based on models and techniques

For requirements models, the paper by Wang et al. [29] defines a requirements modeling notation for real-time control systems. Moreover the paper by Li et al. [30] is about a tool that supports automatic generation of executable prototypes directly from requirements models. The tool analyzes and validates the consistency and functional correctness of the models of the use cases.

The largest number of presentations are on verification of program implementations. Algorithmic approach to alias analysis and null pointer detection is present in the paper of Ma et al. [34]. The work of De et al. [31] presents a consistent model of memory of programs on multi-core platforms, and the work by of Xu et al. [32] is a memory model doe static analysis of C programs. The work of by Li et al. [33] is on an optimization of well-known points-to analysis technique for analyzing pointer properties of object-oriented programs. A framework for improving the scalability of model checking concurrent C programs is presented in the work of Wang et al. [35].

Models and techniques in logic and algebraic reason are presented in the papers [36–38]. The work by Dong [36] is on a model for design and reasoning about context aware systems, and the work by Pu et al. [37] is about algebraic properties of patterns in BPEL. Zhan's paper [38] investigates the linkages between logic reasoning and algebraic reasoning.

With regards to component-based architecture modeling, the work of Liu et al. [39] shows how interface contracts can be used to model and design connectors for security control. The paper by Gomes et al. [40] is about the generation of probabilistic model from a system architecture model and fault-tree for dependability analysis.

There are two papers on modeling and verification of interactions protocols of components. The first is by Chakraborty et al. [41] and it is about analysis of inter-thread communication represented by message sequence graphs that combine combined UML sequence diagrams. The second paper [42] is by Ravn et al. and is applies the model checking tool UPPAAL to the verification of web-service standard transaction protocols. In a more general setting, the problem of model checking CSP [21,22] is revisited in the work of Sun et al. [43], and a process analysis toolkit is introduced.

A common feature of the some of the techniques for high-level model analysis [30,40–42] is that they take care of the understandability and usability to industrial practitioners by using standardized and graphic models, such as UML diagrams, message sequence charts and state/transition tables, and transforms to formal models for which formal analysis and verification are possible. In the work on program analysis, techniques for automatic construction of models from implementations through abstraction are developed.

3.2 Categorization of papers based on technology areas

The papers that were presented in special tracks also address different issues of technology/application areas. These include multi-core systems, embedded systems, hybrid systems and Web services.

3.2.1 Multi-cores

Multi-cores have found wide acceptance among processor manufacturers, thereby enabling parallel programming to gain greater acceptance as well, and moving parallel programming into mainstream. In our track, we had two papers relating to this increased emergence of multi-cores [44,45].

The first of the papers by De et al. refers to the relaxed memory consistency model semantics adopted in multi-core platforms. Memory consistency models have been used in shared-memory multiprocessors for many years. Given a number of processes accessing a shared store, a memory consistency model places restrictions on the order in which the processes can access (read/write) the shared store. This effectively restricts the values that can be returned on the read of a shared variable, and thereby provides a model of execution to the programmer. Multi-threaded programming languages like Java have a memory model of their own, and this memory model defines the semantics of multi-threading in Java. In this paper, the authors present an operational style memory model (unlike Java's existing memory model) which can serve as its multi-threading semantics. Being operational

such a model is more easily amenable to simulation as well as model checking.

The second of the papers by Chakraborty et al. refers to system specifications given as finite graphs of sequence diagrams, where each node of the graph is a UML Sequence Diagram. Such a specification notation emphasizes inter-process communication, and can be used to capture inter-thread communication in a multi-threaded program.

3.2.2 Embedded control systems

Increasingly, with the phenomenal growth of mobile platforms, more and more softwares are being run on embedded systems. Indeed in the current year (2012), the sales of smartphones and other app-enabled devices is scheduled to exceed the number of personal computers.[3] Embedded software needs to be run on platforms with less resources. Thus programmers need to be concerned about energy consumption, code size and such other issues apart from performance. Moreover, much of the embedded software are used to monitor/manage control systems which are in continuous interaction with their external physical environment. Three papers [29,32,40] in our tracks of ISoLA 2006 and of ISoLA 2010 were relevant to technology trends for embedded control systems.

The first paper by Wang et al. [29] refers to requirements modeling for embedded control systems. In particular, the paper presents a requirements modeling language to capture periodically repeating behaviors which are often common in control systems.

The second paper by Xu et al. [32] refers to low-level static analysis of C programs. C is the most widely used programming language for embedded software. In particular, the paper presents a static analyzer called Clang to capture errors due to low level programming features such as pointer dereferences/pointer arithmetic. Such errors are most common in embedded software, since it involves low level programming.

The third paper [40] refers to aeronautical systems, an important application domain for embedded control systems. Due to the extremely safety-critical nature of aeronautical software, formal methods for safety assessment are necessary. The paper presents an innovative approach using probabilistic model checking to obtain a quantitative assessment of system safety.

3.2.3 Hybrid systems

Advanced embedded system design and verification require to model and analyze dense and contentious time state spaces. This forms the area of *hybrid systems*. We have two papers on modeling and verification of hybrid systems [46,47].

The first paper by Wang presents a library called REDLIB, developed by the author, that supports model-checking LTL properties of dense-time automata. Properties can be verified for automata with multiple fairness assumptions. In this tool, a dense-time state space is represented by a BDD-like diagram. Users can use the procedures in REDLIB to quickly construct dense-time models and carry out basic Boolean and state-space operations, postcondtion/precondition calculation, state-space representation normalizations, greatest fixed-point calculation, parametric safety analysis of linear hybrid systems, speed-up techniques for greatest fixed-point evaluation, and coverage analysis of dense-time state spaces.

The second paper by Xia et al. is about the computational algebraic techniques and tool, called *discovery* for discovering invariants and ranking functions of control programs. The main finding is how generations of invariants and non-linear ranking function can be reduced to semi-algebraic systems solving. This approach is now being applied to hybrid system verification by using *differential invariants* [48].

3.2.4 Services

A number of emerging trends in business and technology are poised to bring about a paradigm shift in the way enterprise software is likely to be developed, delivered, deployed and maintained, in the next decade. From the technical perspective, the principle of "service orientation" continues to gain ground. Several practical Web services-based standards, designed to support services inter-operability over internet protocols independent of platforms and programming languages, are being developed and successfully adopted in software development. These are all signs of a technology maturing and going mainstream: there is little doubt now that the principle of loose and dynamic coupling between software components playing the complementary (and frequently dual) roles of "service providers" and "service consumers", is going to shape the core of many next generation systems. We discuss the three papers in [37,42,49] that address the problems of specification and verification of Web services.

The work by Pu et al. [37] presents a notation and analysis techniques for workflow patterns of WSBPEL-like programs. It is a denotational approach to reasoning about properties using algebraic laws.

The keynote of Jifeng [49] at ISoLA 2008 presents an abstract model of coordination and compensation of service. Exceptions, failures and recovery in long running transition services are model in a Guarded Command Language with additional combinators for coordination and compensation. It is shown that such an extension is a conservative one because it preserves the algebraic laws for designs, which can be

[3] See http://www.computerworld.com/s/article/9199918/In_historic_shift_smartphones_tablets_to_overtake_PCs for latest news.

used to reduce all programs to a normal form algebraically. Galois link between the standard design model with extended new model is investigated, showing that the embedding from the former to the latter is actually a homomorphism.

Finally, the paper by Ravn et al. [42] looks at the infrastructural support related to Web services. In particular, Web services require transactional support for reasons of consistency. The aforementioned paper studies the modeling and verification of protocols to support atomic transactions in Web services.

4 Overview of the special issue

This special issue consists of four contributions from the community of the three ISoLA special tracks to further the understanding of model-driven transformation and validation, and to address the issues of its application to solving related to industrial systems. The papers included have a focus on performance-oriented validation, with particular focus on real-time software.

4.1 Timing analysis of MSC specifications with asynchronous concatenation, by Li and Pan [50]

Industries, those in telecommunication in particular, often uses message sequence chart (MSC) for describing interactions among a set of system components. However, a MSC model can become quite complex when both basic MSCs and composite MSCs are used to describe systems with iterations and alternative branches. This paper presents a technique for timing analysis of MSCs with asynchronous concatenation. In particular, the model of *flexible loop-closed MSCs* is defined. In such a MSC, the execution time of the entry node of a loop is flexible while constraint can be enforced by the execution of the entire loop. The main result is that the problems of *reachability analysis* and *bounded delay analysis* can be solved efficiently by linear programming. The solutions have been implemented into the tool TASS and evaluated by experiments.

4.2 Formal modeling and validation of stateflow diagrams by Chen et al. [51]

Stateflow is an industrial tool for modeling and simulating control systems in model-based development. This paper presents latest work of the authors on automatic verification of Stateflow using model checking techniques. The approach is to systematically translate Stateflow diagrams to a formal modeling language called CSP# by precisely following Stateflow's execution semantics. A translator is developed inside the model checker to automate this process with the support to various Stateflow advanced modeling features. Formal analysis can be conducted on the transformed CSP# with simulation and model checking power. Using this approach, one can not only detect bugs in Stateflow diagrams, but also discover subtle semantics flaws in Stateflow user's guide and demo cases.

4.3 Constructive model-based analysis for safety assessment by Mota et al. [52]

The aerospace industry still uses fault-tree analysis (FTA) to perform reliability analysis. This is related to the ease of modeling and analysis of fault-trees when compared to Markov models, although FTA provides a weaker solution than a Markov-based analysis. This paper proposes a way of overcoming such a difficulty by automating the creation of Markov-based models as well as their analysis from Simulink diagrams, annotated with failure information. The target Markov-based models are expressed in PRISM, and the analysis is carried out by the associated PRISM model checker. The strategy is compositional and based on a comprehensive set of translation rules that define a semantics for an annotated Simulink subset in PRISM. To illustrate the application of the overall strategy, the approach is applied to a classical avionics case study: an actuator control system. This is an extended version of the paper ISoLA 2010 [40].

4.4 Compositional verification of real-time systems using Ecdar by Nyman et al. [53]

This paper presents a complete specification theory for timed systems implemented in the Ecdar tool. The operations of the specification theory is illustrated using a running example, showing the models and verification checks. The power of the compositional verification is demonstrated by an in depth case study of a leader election protocol is investigated. This case study is modeled in Ecdar as Timed Input/Output Automata and both monolithic and compositional verification of two interesting properties are performed. A huge performance improvement in execution time of the compositional is gained in comparison with classical verification.

5 Conclusion

This article gives a summary on the understanding of the key theme, feature and challenges of model transformation validation gained through the presentations and discussion of three special tracks organized for ISoLA in 2006, 2008 and 2010, respectively. This special issue is organized based on this understanding, and it consists of four papers that address problems in real-time embedded software. We believe these papers will further the understanding of the model-driven approach to software design and validation. The papers have

been carefully revised and re-reviewed after the conference. We wish the readers a very happy read.

Acknowledgments We would like to thank the authors of the papers presented at the special tracks, and all the participants for the discussion. The PC members and reviewers of the special traces too have made a great contribution through the professional work they did. Last, but not the least, we would like to thank Tiziana Margaria and Bernhard Steffen for giving us the opportunities to organize the special tracks. The work of Z. Liu was supported by Macau Science and Technology Development grants GAVES and SAFEHR and the Chinese Natural Science Foundation Grants No. 60970031, 61103013. The work of A. Roychoudhury was partially supported by a Singapore Ministry of Education research Grant MOE2010-T2-2-073.

References

1. Margaria, T., Steffen, B. (eds.): Proceedings of the Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15–19 November 2006, IEEE Computer Society (2006)
2. Margaria, T., Steffen, B. (eds.): Proceedings of the Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008 Communications in Computer and Information Science, vol. 17. Springer, Berlin (2008)
3. Margaria, T., Steffen, B. (eds.): Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010, Lecture Notes in Computer Science, vol. 6415. Springer, Berlin (2010)
4. Dijkstra, E.W.: The humble programmer—ACM Turing Award Lecture. Commun. ACM **15**(10), 859866 (1972)
5. Brooks Jr, F.P.: No silver bullet: essence and accidents of software engineering. IEEE Comput. **20**(4), 1019 (1987)
6. Booch, G.: Object-Oriented Analysis and Design with Applications. Addison-Wesley, Boston (1994)
7. Brooks Jr, F.P.: The mythical man-month: after 20 years. IEEE Softw. **12**(5), 5760 (1995)
8. Holzmann, G.J.: Conquering complexity. IEEE Comput. **40**(12), 111–113 (2007)
9. Wirsing, M., Banatre, J. P., Holzl. M. M., Rauschmayer, A. (eds.): Software-Intensive Systems and New Computing Paradigms—Challenges and Visions. Lecture Notes in Computer Science, vol. 5380. Springer, Berlin (2008)
10. Leveson, N.G., Turner, C.S.: An investigation of the Therac-25 accidents. IEEE Comput. **26**(7), 1841 (1993)
11. Robinson, K.: Ariane 5: flight 501 failurea case study. http://www.cse.unsw.edu.au/se4921/PDF/ariane5-article.pdf (2011)
12. Hoare, C.A.R.: The verifying compiler: a grand challenge for computing research. J. ACM **50**(1), 63–69 (2003)
13. Hoare, C.A.R., Misra, J., Leavens, G.T., Shankar, N.: The verified software initiative: a manifesto. ACM Comput. Surv. **41**(4), 22:1–22:8 (2009)
14. Object Managment Group. Model driven architecture—a technical perspective. Document number ORMSC 2001-07-01 (2001)
15. Szyperski, C.: Component Software, Beyond Object-Orients Programming. Addison-Wesley, Boston (1997)
16. Broy, M.: Seamless Method- and Model-based Software and Systems engineering. The Future of Software Engineering. Springer, Berlin (2011)
17. Liu, Z., He, J., Li, X., Chen, Y.: A relational model for formal object-oriented requirements analysis in UML. In: Proceedings of the 5th International Conference on Formal Engineering Methods. LNCS, vol. 2885, pp. 641–664. Springer, Berlin (2003)
18. Chen, X., Liu, Z., Mencl, V.: Separation of concerns and consistent integration in requirements modelling. In: Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science. LNCS, vol. 4362, pp. 819–831. Springer, Berlin (2007)
19. Ke, W., Li, X., Liu, Z., Stolz, V.: rCOS: a formal model-driven engineering method for component-based software. Front. Comput. Sci. China **6**(1), 17–39 (2012)
20. Chen, Z., Liu, Z., Ravn, A.P., Stolz, V., Zhan, N.: Refinement and verification in component-based model-driven design. Sci Comput. Program. **74**(4), 168–196 (2009)
21. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Upper Saddle River (1985)
22. Roscoe, A.W.: Theory and Practice of Concurrency. Prentice-Hall, Upper Saddle River (1997)
23. Milner, R.: Communication and Concurrency. Prentice-Hall Inc., Upper Saddle River (1989)
24. Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. CWI Q. **2**(3), 219246 (1989)
25. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes **26**(5), 109–120 (2001)
26. He, J., Li, X., Liu, Z.: rCOS: a refinement calculus of object systems. Theor. Comput. Sci. **365**(1–2), 109142 (2006)
27. Liu, Z., Mencl, V., Ravn, A.P., Yang, L.: Harnessing theories for tool support. In: Proceedings of the Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15–19 November 2006, IEEE Computer Society (2006)
28. Aichernig, B.K., He, J., Liu, Z., Reed, M.G.: Integrating theories and techniques for program modeling, design and verification. In: Proceedings of the 1st IFIP Conference on Verified Software: Theories, Tools, Experiments (VSTTE). LNCS, vol. 4171, pp. 291–300. Springer, Berlin (2005)
29. Wang, Z., Li, J., Zhao, Y., Qi, Y., Pu, G., He, J., Gu, B.: SPARDL: a requirement modeling language for periodic control systems. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
30. Li, X., Liu, Z., Schäf, M., Yin, L.: AutoPA: automatic prototyping from requirements. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
31. De A., Roychoudhury, A., D'Souza, D.: WOMM: a weak operational memory model. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
32. Xu, Z., Kremenek, T., Zhang, J.: A memory model for static analysis of C programs. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
33. Li, Q., Zhao, J., Li, X.: Optimize context-sensitive Andersen-style points-to analysis by method summarization and cycle-elimination. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
34. Ma, X., Wang, J., Dong, W.: Computing must and may alias to detect null pointer dereference. In: Proceedings of the Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008 Communications in Computer and Information Science, Springer, Berlin (2008)

35. Wang, J., Yi, X., Yang, X.: Towards a framework for scalable model checking of concurrent C programs. In: Proceedings of the Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15–19 November 2006, IEEE Computer Society (2006)
36. Dong, J.S., Feng, Y., Sun, J., Sun, J.: Context awareness systems design and reasoning. In: Proceedings of the Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15–19 November 2006, IEEE Computer Society (2006)
37. Pu, G., Zhu, H., He, J., Qiu, Z., Yang, H., Zhao, X.: Patterns with algebraic properties in BPEL0. In: Proceedings of the Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15–19 November 2006, IEEE Computer Society (2006)
38. Zhan, N.: Connecting algebraic and logic descriptions of concurrent systems. In: Proceedings of the Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15–19 November 2006, IEEE Computer Society (2006)
39. Liu, Z., Morisset, C., Stolz, V.: A component-based access control monitor. In: Proceedings of the Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008 Communications in Computer and Information Science, Springer, Berlin (2008)
40. Gomes, A., Mota, A., Sampaio, A., Ferri, F., Buzzi, J.: Systematic model-based safety assessment via probabilistic model checking. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
41. Chakraborty, J., D'Souza, D., Kumar, K.N.: Analyzing message sequence graph specifications. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
42. Ravn, A.P., Srba, J., Vighio, S.: A formal analysis of the web services atomic transaction Protocol with UPPAAL. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
43. Sun, J., Liu, Y., Dong, J.S.: Model checking CSP revisited: introducing a process analysis toolkit. In: Proceedings of the Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008 Communications in Computer and Information Science, Springer, Berlin (2008)
44. De, A., Roychoudhury, A., D'Souza, D.: WOMM: a weak operational memory model. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
45. Chakraborty, J., D'Souza, D., Kumar, K.N.: Analysing message sequence graph specifications. In: Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation—4th International Symposium on Leveraging Applications, ISoLA 2010. Lecture Notes in Computer Science. Springer, Berlin (2010)
46. Wang, F.: REDLIB for the formal verification of embedded systems. In: Proceedings of the Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006, Paphos, Cyprus, 15–19 November 2006, IEEE Computer Society (2006)
47. Xia, B., Yang, L., Zhan, N.: Program verification by reduction to semi-algebraic systems solving. In: Proceedings of the Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008 Communications in Computer and Information Science, Springer, Berlin (2008)
48. Yang, L., Zhou, C., Zhan, N., Xia, B.: Recent advances in program verification through computer algebra. Front. Comput. Sci. China **4**(1), 1–16 (2010)
49. Jifeng, H.: Modelling coordination and compensation. In: Proceedings of the Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008 Communications in Computer and Information Science, Springer, Berlin (2008)
50. Li, X., Pan, M.: Timing analysis of MSC Specifications with Asynchronous Concatenation (in this volume)
51. Chen, C., et al.: Formal modeling and validation of stateflow diagrams (in this volume)
52. Mota, A., et al.: Constructive model-based analysis for safety assessment (in this volume)
53. Nyman, U., et al.: Compositional verification of real-time systems using Ecdar (in this volume)