A Dependability Roadmap for the Information Society in Europe

Part 3 – Towards a Dependability Roadmap

August 2003

INFORMATION SOCIETIES TECHNOLOGY (IST) PROGRAMME

Project **Accompanying Measure System Dependability**

Work-package 1: Overall Dependability Road-mapping

Deliverable D1.1: Road-map

Contract no.: ***IST-2001-37553***

This Report "Towards a Dependability Roadmap" is Part 3 of the Deliverable 1.1 of the project AMSD.

Parts 1 and 2 can be downloaded from the project web site: http://www.am-sd.org/

Disclaimer

The information represented in this document does not represent the formal position of any particular organisation nor do views expressed represent a formal position of the European Commission.

Editors

Marcelo Masera, JRC

Robin Bloomfield, Adelard

We invite comments on this report.

Please send them to: amsd-roadmap-feedback@adelard.com

Table of contents

APPENDIX G – TRENDS IN VERIFICATION AND VALIDATION

Subtopic: Model Checking

By Mieke Massink, ISTI, Pisa

Introduction

The class of systems that require the application of the most effective and rigorous quality control measures are those with high technical complexity where management of concurrent, distributed, real-time activities in presence of faults or other rare combinations of events is critical.

This combination of factors is likely to be found in an increasing number of critical hardware and software systems as a consequence of the inevitable grow in scale and functionality and the trend to interconnect systems in networks. The likelihood of subtle, difficult to detect errors in this class of systems is much greater than in systems without concurrency and real-time requirements, in particular if also human interaction with the system is considered.

With traditional quality control measures such as peer review and testing alone it is extremely difficult and time consuming to obtain sufficient statistical information about the adequacy of software for complex and critical systems in the presence of rare but possibly catastrophic errors. This is because in order to find rare errors in general an exorbitant amount of testing or simulation is required to have a chance to detect such errors, which is extremely costly and time-consuming [6,3].

One promising way of achieving more reliable complex systems is by using FORMAL METHODS. These are mathematically based languages, techniques and software tools for specifying and verifying MODELS of systems often constituting their designs.

Formal methods can be used as a complementary quality control measure that can reveal inconsistencies, ambiguities, and incompleteness and several other shortcomings of system designs early on in the development process. This reduces significantly the accidental introduction of design errors in the development of the software [7] leading to higher quality software and cost reduction in the testing and maintenance phases of system development. Formal methods and their related software tools have been used extensively and successfully in the past in a variety of areas including protocol development, hardware and software verification, embedded/real-time/dependable systems and human safety, demonstrating that great improvements in system behaviour can be realised when system requirements and design have a formal basis.

In the following we highlight some of the more exciting recent advances and challenges in the development of MODEL CHECKING.

General advances and challenges

Model checking is a verification technique in which efficient algorithms are used to check, in an automatic way, whether a desired property holds for a finite model of the system. Very powerful logics have been developed to express a great variety of system properties and high-level languages have been designed to specify system models. Examples of the first are

various variants of temporal logic and notable examples of the latter are process algebras, known for their important compositional properties.

One of the major advantages of model checking in comparison with e.g. theorem proving is the possibility to automatically generate counterexamples that provide designers with important debugging information [5]. Another advantage is the fact that the verification is automatized so that much of the mathematical details of the verification are hidden to the designer. This reduces the amount of training that is required in order to use model checking tools. The cost effectiveness of this technique is further improved by the fact that it is used early on in the software development cycle and allows therefore early detection of errors avoiding the need for much more expensive error correction in later phases of the development cycle.

The main disadvantage of model checking is the state explosion problem, however, many techniques have been developed to successfully alleviate this problem. Most notably is the use of appropriate abstraction techniques that allow verifying models of systems with an essentially unlimited number of states.

The development of these kinds of abstractions and their automatic generation is one of the main challenges in this area. An important concern is the correctness of abstractions. Here an important combination with theorem proving techniques may be very productive.

Model checking is becoming widely used in industry (e.g. IBM, Intel, Microsoft, Motorola, SUN Microsystems, Siemens, Bell labs) and is being applied to numerous industrial case studies and standards ranging from hardware, operating systems, compilers, communication protocols to control systems, embedded real-time systems and multi-media application software. An overview of some of these applications can be found in [1]. Evidence of the high relevance of model checking for the dependability community is the presence of a workshop on model checking next year at the International Conference on Dependable Systems and Networks, the most important conference on dependability.

For an ever increasing class of systems their functional correctness cannot be separated anymore from their ``quantitative correctness'', e.g. in real-time control systems, multimedia communication protocols and many embedded systems.

The correct functioning of those applications depends critically for example on the chosen deadlines for the timers involved in those applications. For this reason, the high-level specification languages, the logics and the algorithms for model checking are being extended in order to verify real-time, performance and dependability properties as well.

Although there currently exist powerful tools supporting high-level specification languages, e.g. stochastic Petri-nets and stochastic activity networks, that allow convenient specification of performance models, the specification of the measures of interest has remained fairly informal, unstructured and restricted to some simple state-based measures such as throughput, mean time to failure and delay [2].

The powerful logic-based methods developed in the area of formal methods, and in particular in the area of model checking, allow one to specify both state based properties and properties over state sequences.

Recently proposed continuous stochastic and reward logics show how standard measures of interest can be structured and formalized and allow for the specification of more complex measures that have not yet been addressed in the literature on performability such as the interpretation of rewards as costs that allows a.o. the formal analysis of power consumption in ad-hoc mobile networks [2].

This opens up a whole new and exiting area where the synergy of different fields is likely to lead to new insights and much improved quality control methods for the area of

dependability. Another challenge will be the extension of the generation of informative counterexamples for these new logics.

A further important issue is the relation between the abstract models used for specification and verification and the final implementation of the system. Formal testing is an approach that forms a bridge between these models and the implementation. It re-uses formal models for automatically generating/synthesizing relevant test-suites for component, integration and system testing and for actually executing such tests against real system implementations [4].

Challenging issues in this area form the automated test generation as a promising ``push-button'' technology and the issues of test-case selection and the formal testing of quantitative aspects of system behaviour.

With the recent increase in efficiency of model checking techniques there is renewed interest in its direct application to software written in real programming languages such as C and Java. This could enormously enhance the capability of error detection in software code but it depends critically on the development of appropriate heuristics to guide the search for possible errors in the enormous state space generated by the software [8].

Recommendations

1. Formal methods and related tools can be used most profitable to analyse parts of systems that are most critical. The development of guidelines for the identification of those system parts, the level of formality that is required and the selection of the most appropriate method to analyse its properties need to be addressed.

2. The development of convenient and powerful logics to express relevant quantitative dependability measures in a precise, concise and formal way is key to future systematic and possible automatic, analysis of dependability aspects of critical systems. The development of logics to express in addition cost and reward based measures can greatly enhance the number of interesting measures relevant for dependable systems.

3. Counterexamples generated by model checkers provide designers with valuable information about possible problems in their designs. The extension of these examples to quantitative model checking for performance and dependability aspects of designs is one of the key issues that need to be addressed.

4. The efficiency and effectiveness of Model Checking for Performance and Dependability evaluation depends critically on the development of suitable, possibly automated, abstraction mechanisms in order to deal with very large state spaces.

5. Particular attention must be paid to the development of models that can take user interaction with the system into account. Also here quantitative models play an important role when it comes to descriptions of human performance aspects relevant for the overall evaluation of critical systems.

6. The use of different methods brings about the issue of the relation between models used for different verification purposes (e.g. simulation models, models used for model checking and theorem proving but also formalisms used for data intensive vs. those for control intensive problems). This becomes particularly relevant in the case of combined verification.

7. Training of designers, integration of tools and methods in the design process and the development of course ware to allow a broad take-up of formal methods in industry and elsewhere are fundamental for the transfer of available knowledge to the daily practice of software developers.

Sources/References

[1] Formal Methods: State of the Art and Future Directions. E. M. Clarke and J. M. Wing et al. ACM 50th Anniversary Issue, Strategic Directions in Computing Research, ACM Computing Surveys Vol. 28 (4es), No. 4, 1996. Available on line: http://www.acm.org/surveys/sdcr/

[2] Automated Performance and Dependability Evaluation using Model Checking. C. Baier, B. Haverkort, H. Hermanns, J-P. Katoen. in Tutorial Proc. PERFORMANCE 2002, Springer LNCS 2459, 2002. Available on line: http://web.informatik.uni-bonn.de/I/baier/publikationen.html

[3] Formal Methods and the Certification of Critical Systems J. Rushby, Technical Report CSL-93-7, 1993. Available on line: http://www.csl.sri.com/papers/csl-93-7/

[4] Testing Transition Systems: An Annotated Bibliography. Ed Brinksma and Jan Tretmans, MOVEP 2000 - MOdelling and VErification of Parallel processes, LNCS, Vol. 2067, Springer-Verlag, Nantes, France, pages: 187 - 195, published in 2001

[5] Tree-Like Counterexamples in Model Checking. E. Clarke, S. Jha, Y. Lu, H. Veith. IEEE symposium on Logic in Computer Science (LICS), 2002

[6] Qualitaetssicherung Software-basierter technischer Systeme - Problembereiche und loesungansaetze. P. Liggesmeyer et al. Informatik Spektrum, 21: 249-258, 1998.

[7] The theory and practice of a formal method: NewCoRe. G.J. Holzmann. Proceedings 13th IFIP World Congress, pp. 35-44, 1994.

[8] Model Checking Java Programs using Structural Heuristics. A. Groce, W. Visser, In: Proceedings of the ACM SIGSOFT 2002 Int. Symposium on Software Testing and Analysis, Software Engineering Notes Vol. 27, nr. 4, 2002.

[9] Embedded Systems Road Map 2002. Vision of Technology on the future of PROGRESS. L.D.J. Eggermont (Ed.), Technology Foundation (STW), 2002.