Electronic Notes in Theoretical Computer Science 80 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume80.html 5 pages

Formal Specification and Verification of Complex Systems

S. Gnesi [1]

Istituto di Scienze e Tecnologie Informatiche - C.N.R.
Via G. Moruzzi 1, I-56100 Pisa, Italy

1 Introduction

The application of formal methods in the rigorous definition and analysis of the functionality and the behaviour of a system, promises the ability of showing that the system is correct. Given such a promise, that is already out since several years, it is astonishing to see how little formal methods are actually used in the safety critical system industry, though the use of formal methods is increasingly required by the international standards and guidelines for the development of complex systems.

Industrial acceptance of formal methods is strictly related to the investment needed to introduce them, to the maturity of tool support available, and to the easiness of use of formal methods and tools. Nowadays, the industrial trend is directed to the adoption of formal verification techniques to validate the design, integrating them within the existing development process. Industries are more keen to accept formal verification techniques assessing the quality attributes of their products, obtained by a traditional life cycle, rather than a fully formal life cycle development, due to the lower training and innovation costs of the former. Several approaches to the application of formal methods in the development process have been proposed, differing for the degree of involvement of the method within it. Starting from rigorous specifications, formal methods can be used for the derivation of test cases, or as a validation technique aimed at proving that the specification satisfies the requirements, or as an auxiliary technique in the automated generation of code.

The Formal Methods && Tools Group of ISTI-CNR is active in the fields of development and application of formal notations, methods and software support tools for the specification, design and verification of complex computer systems. These systems often share important features like concur-

⋆ Work partially founded by Miur project*SP4*.
[1] Email: gnesi@iei.pi.cnr.it

©2003 Published by Elsevier Science B. V.

rent/parallel behaviour, physical distribution and mobility. Moreover, they often must meet real-time and security constraints and are used in safety-critical missions where also human factors play a major role.

We are currently involved in several research activities [2]. in this short paper we summarize some results in the areas of application and development of model-checking algorithms and tools for Embedded/Safety-Critical Systems, Mobile systems, Security Protocols, UML statechart diagrams

2 Model checking for Embedded Systems

Embedded computer-controlled systems often include fault tolerance techniques. Fault tolerance is the property of a system to provide, by redundancy, a service complying with the specification in spite of faults occurred or occurring [13]. The rigorous definition and verification of these systems is extremely important, since it promises the ability of showing that a system is correct also in presence of faults and failures, so that it cannot harm in any case. For this reason the use of formal specification and verification techniques is increasingly required in this field.

In recent works we have examined some specific characteristics of fault tolerant systems, such as the use of redundancy, with a twofold objective: on one hand, to show how these characteristics naturally contain the state space size, and on the other hand, to study the state space reduction techniques more suited to exploit them [3]. We have used our analysis technique to specify and verify two fault tolerant system designs. The first study is the specification and verification of the safety requirements of a *Railway Interlocking System* developed by Ansaldo Trasporti [2]. The second one is the specification and verification of fault tolerant mechanisms defined inside the project *GUARDS* (Generic Upgradable Architecture for Real-Time Dependable Systems) [19].

3 Model checking for mobile systems

Verification techniques based on finite state representations of system behaviors cannot be directly applied to those concurrent systems where behaviors may refer to past steps of the ongoing computation. In this case, even simple agents can generate infinite state systems. An illustrative example is provided by the so called *mobile* systems, i.e. systems where the communication topology among processes can dynamically evolve when the computation progresses. The π-calculus [17] gives an example of this situation. Its primitives are simple but expressive: channel names can be created, communicated and they are subjected to sophisticated scoping rules. The π-calculus has greater expressive power than ordinary process calculi, but also a much more complicated theory. In particular, the usual operational models are infinite-state and

[2] see also `http://matrix.iei.pi.cnr.it/FMT/` for futher details

infinite branching. Hence, even though the π-calculus generalizes CCS [16], the semantic-based verification tools developed for CCS cannot be directly reused for the π-calculus.

Recently a verification environment, HAL[7], for the π-calculus has been developed The construction of the verification environment takes a direct advantage of the finite representation of π-calculus agents presented in [18]. In fact, the environment includes implementations of facilities which allow a π-calculus agent to be translated into an ordinary automaton. The theory of [18] ensures that equivalent ordinary automata are associated to equivalent π-calculus agents. Hence, existing equivalence checkers for ordinary automata can be used to calculate whether or not π-calculus agents are equivalent. The environment also supports verification of logical formulae expressing desired properties of the behaviour of π-calculus agents expressed in terms of π-logic [10] formulae. The π-logic includes modalities indexed by π-calculus actions that are translated into a standard temporal logic for ordinary automata. Existing model checkers can hence be used to verify whether or not a formula holds for a given π-calculus agent.

These translation facilities have been implemented on top of the JACK environment [4]. It consists of several specification and verification tools interfaced around the FC2 format which acts as the "gluing" entity among the tools.

4 Model checking of security protocols

The wide diffusion of Internet as a commercial medium makes the guarantee of security a necessity for every distributed protocol running over it. Security protocols, known also as *cryptographic protocols*, are quite tricky: the literatures is full of significative examples of protocols that, although considered secure for long, revealed dummy flaws when formally checked.

In this field our experience has been focused in defining a verification framework for the *spi-calculus* [1], a process algebra derived from the π-calculus with operators to encrypt and decrypt messages. The spi-calculus is expressive and flexible enough to easily allow the description of a wide class of cryptographic protocols. Our approach to security verification follows a logic-based model checking paradigm [6]. This approach requires a temporal logic (for example the one defined in [8]) to be used for expressing security properties, such as secrecy and integrity, while the spi-calculus is provided with an operational semantics based on labeled transition systems on which the satisfiability relation of logic formulas is defined. In this way we are able to verify a wide class of security properties (*e.g.,* anonymity) when a finite state model of a protocols [11,12] is provided.

5 Model checking for UML statecharts diagrams

UML ([20]) Statechart Diagrams are used for describing dynamic aspects of system behaviour. UML is a semi-formal language, since its syntax and static semantics (the model elements, their interconnection and the well-formedness rules) are defined precisely, but its dynamic semantics are not specified formally. In recent papers we have presented a formal operational semantics for a behavioural subset of UML Statechart Diagrams (UMLSDs) including a formal proof of their correctness with respect to major UML semantics requirements concerning behavioural issues [9,15] basing on results exploited in [14]. In particular we have presented all the conceptual issues related to building a tool for action based branching time model-checking, for the automatic verification of formal correctness of UML Multicharts. The approach we have proposed preserves all the information necessary to report the results of model checking in terms of the original UMLSD specification. The reference verification environment used for this model checking approach is JACK [4], where automata are represented in a standard format which facilitates the use of a collection of tools for automatic verification.

Recently we have started a new project aimed at developing an on the fly Model Checker, UMC, for UML communicating state machines. UMC is essentially an experiment in the design of an integrated tool for the construction, the exploration, the analysis and the verification of the dynamic behavior of UML models described as a set of communicating state machines. The current alpha-version of the UMC prototype (which is now at version 2.3) is accessible "online" through its www interface at the address http://matrix.iei.pi.cnr.it/umc/demo.

References

[1] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols. The Spi Calculus. Technical Report 149, Digital Equipment Corporation Systems Research Center, Palo Alto, California, 1998.

[2] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, D. Romano. A Formal Verification Environment for Railway Signaling System Design. Formal Methods in System Design 12, 139-161, 1998.

[3] C. Bernardeschi, A. Fantechi, S. Gnesi. Formal Validation of Fault Tolerance Mechanisms inside GUARDS, Journal of Reliability Engineering and System Safety, Vol. 71, n.3, Feb. 2001, pp. 261-270.

[4] A. Bouali, S. Gnesi, S. Larosa. The Integration Project for the JACK Environment, bulletin of the EATCS, October (54), pp.207-223, 1994 (See also http://rep1.iei.pi.cnr.it/projects/JACK)

[5] J. Broersen and R. Wieringa. Interpreting UML-statecharts in a modal μ-calculus. Unpublished manuscript, 1997.

[6] E.M. Clarke, E.A. Emerson, A.P. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. ACM Transaction on Programming Languages and Systems, vol.8, n. 2, 1986, pp. 244-263.

[7] G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, G. Ristori, Verifying Mobile Processes in the HAL Environment. *Computer Aided Verification* (CAV'98), LNCS 1427, 1998.

[8] E. M. Clarke and S. Jha and W. Marrero, Verifying security protocols with Brutus, ACM Transactions on Software Engineering and Methodology, vol. 9, n. 4, 443–487, 2000

[9] S. Gnesi, D. Latella, and M. Massink. Model checking UML statechart diagrams using JACK. In A. Williams, editor, *Fourth IEEE International High-Assurance Systems Engineering Symposium*, pages 46–55. IEEE Computer Society Press, 1999.

[10] S. Gnesi, G. Ristori. *A Model Checking Algorithm for π-calculus agents.* In *Advances in Temporal Logic.* Kluwer Academic Publishers, pp.339 - 357, 2000.

[11] S. Gnesi, D. Latella, and G. Lenzini. A BRUTUS logic for the Spi-Calculus. In *Proc. of the ACM SIGPLAN and IFIP WG1.7 Worshop on Issues in Theory of Security (WITS 02)*, 2002.

[12] S. Gnesi, D. Latella, and G. Lenzini. Towards a model checking spi-calculus dialect. Technical Report 0000, Istituto di Elaborazione dell'Informazione - CNR, Pisa, July 2002.

[13] J.C. Laprie (Ed.). Dependability: Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems, vol. 5, Springer-Verlag, 1992.

[14] D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML statechart diagrams. IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems, Kluwer Academic Publishers, 1999.

[15] D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing. Springer-Verlag*, 2000.

[16] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[17] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100:1–77, 1992.

[18] U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In *Proc. CONCUR'95*, LNCS 962. Springer Verlag, 1995.

[19] D. Powell, et al. GUARDS: a Generic Upgradable Architecture for Real-time Dependable Systems. IEEE Transactions on Computers, 1999.

[20] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.