Formal Methods @ Runtime

Radu Calinescu[1] and Shinji Kikuchi[2]

[1] Aston University, Aston Triangle, Birmingham B4 7ET, UK
r.c.calinescu@aston.ac.uk

[2] Fujitsu Laboratories Limited, 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan
skikuchi@jp.fujitsu.com

J'aimais, et j'aime encore, les mathématiques pour elles-mêmes comme n'admettant pas l'hypocrisie *et le* vague*, mes deux bêtes d'aversion.*[3]

H. B. Stendhal, *La Vie d'Henri Brulard*

Abstract. Heuristics, simulation, artificial intelligence techniques and combinations thereof have all been employed in the attempt to make computer systems adaptive, context-aware, reconfigurable and self-managing. This paper complements such efforts by exploring the possibility to achieve runtime adaptiveness using mathematically-based techniques from the area of formal methods. It is argued that *formal methods @ runtime* represents a feasible approach, and promising preliminary results are summarised to support this viewpoint. The survey of existing approaches to employing formal methods at runtime is accompanied by a discussion of their challenges and of the future research required to overcome them.

1 Introduction

The use of rigorous logic in the design and analysis of computer programs was first proposed in the late 1960s [1, 2]. Several decades and a Turing Award [3] later, the set of mathematically-based techniques collectively known as *formal methods* comprises effective tools for the formal specification [4, 5], development [6] and verification [7] of computer systems. Already widely adopted in hardware design and verification [8], formal methods have more recently been applied to the development of software systems [9], where they are increasingly used to improve the quality of software alongside traditional approaches such as testing and simulation. Contributors to the latter advance include major software companies, who are not only using formal methods internally, but also integrating formal techniques into their software development platforms, and actively contributing to formal methods research. These developments have established

[3] "I used to love mathematics for its own sake, and I still do, because it allows for no *hypocrisy* and no *vagueness*, my two *bêtes noires*."

formal methods as an effective aid in producing high-integrity systems—a key challenge for today's developers of complex computer systems.

This paper explores the possibility to use formal methods in addressing another grand challenge of computer systems, namely runtime adaptation. Computer systems are increasingly employed—and expected to cope with limited or no human intervention—in applications characterised by continual change to system state, workload and objectives. In response to this challenge, the research community has come up with the idea of adding self-adaptation capabilities to computer systems. The impressive body of work carried out in newly emerged research fields such as autonomic, context-aware and ubiquitous computing has so far led to the development of adaptive systems based on a combination of heuristics, simulation and artificial intelligence techniques. While experimental results suggest that such solutions are often effective, they alone cannot provide the high levels of predictability and dependability that adaptive, autonomic computer systems are typically expected to attain [10, 11]. One of the most promissing approaches to achieving these necessary characteristics is to use mathematically based techniques in the runtime adaptation process, i.e., to employ *formal methods* @ *runtime* (FM@R).

This paper advocates that FM@R have the potential to contribute to the development of adaptive computer systems that offer high levels of predictability and dependability in several ways. Some of these are explored in the following sections, starting with the authors' work in Sections 2–4. Section 2 presents the use of quantitative verification to comply with non-functional requirements in adaptive computer systems. Section 3 describes how lightweight formal methods can be used to synthesise the reconfiguration operations that adaptive computer systems must undertake in order to achieve their objectives in the presence of changes in system state, workload and environment. Section 4 explains the use of model checking to verify adaptation rules in computer systems. The concluding section summarises the potential of FM@R, and suggests possible avenues for overcoming some of their current limitations.

2 Quantitative Verification @ Runtime

2.1 Description

Quantitative verification is a mathematically-based technique for establishing the correctness, performance and reliability of systems that exhibit stochastic behaviour [12]. Given a precise mathematical model of a real-world system, and formal specifications of quantitative properties of this system, an exhaustive analysis of these properties is performed. Example properties include the probability that a fault occurs within a specified time period, and the expected power consumption of a computer system under a given workload.

While quantitative verification is traditionally used for the off-line analysis of system models such as Markov chains and Markov decision processes, recent research has successfully employed an on-line version of the technique to support

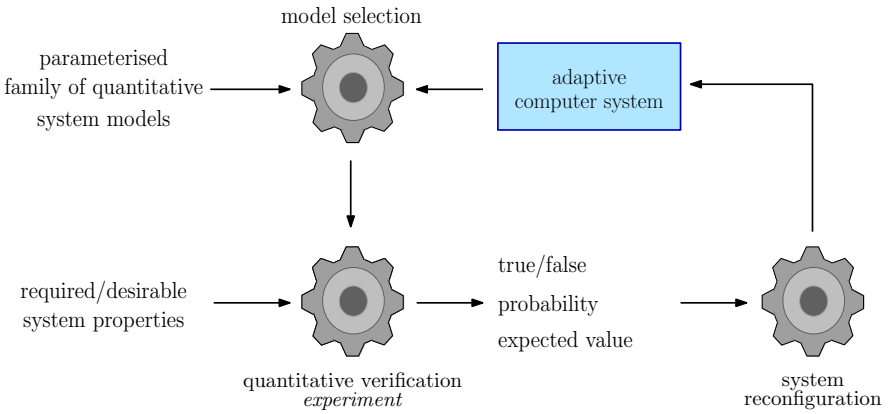

Fig. 1. Using quantitative verification in adaptive computer systems

self-optimisation in adaptive computer systems [13–15]. This approach involves using a quantitative system model parameterised by the different scenarios that the system can operate in and by the different configurations that can be selected for it (Figure 1). The system is monitored continuously to identify the scenario it operates in, and this monitoring information is used to fix the scenario-based model parameters. A quantitative verification *experiment* is then carried out at runtime, to assess which feasible configuration satisfies the system objectives best for these parameter values. Finally, this configuration is adopted automatically, thus ensuring that the system continues to achieve its objectives as it transitions from one operating scenario to another.

The approach is applicable to systems that fulfil two requirements:

1. The system behaviour that is of relevance for the planned adaptation can be modelled as a Markovian chain. Several examples of applications that satisfy this requirement are described later in this section.
2. The system objectives can be expressed as combinations of formal quantitative properties, i.e., properties specified in probabilistic computational tree logic (PCTL) [16] and continuous stochastic logic (CSL) [17] for discrete- and continuous-time Markovian models, respectively [12]. As illustrated in the remainder of the section, this ensures that non-functional properties describing reliability, performance and cost/reward system characteristics are supported by the approach.

2.2 Practical Realisation

The probabilistic model checker PRISM [18] and the general-purpose autonomic computing framework GPAC [13] were used to implement the FM@R approach from Figure 1 in a range of application domains:

– Dynamic power management [14]. The approach was used to ensure that a Fujitsu disk drive subjected to variable workload achieved predictable response time in an energy-efficient way. The disk drive was modelled as a continuous-time Markov chain, and the system objectives were specified as a combination of *reward-extended* [12] CSL properties.
– Adaptive allocation of data-centre resources [19]. Runtime quantitative verification was used to support the adaptive allocation of servers to variable-workload clusters in the presence of data-centre component failures and repairs. A continuous-time Markov chain was used to describe the system behaviour, and the required cluster reliability properties were expressed as CSL formulas. Provably optimal server allocations guaranteed to comply with the required reliability thresholds were achieved irrespective of the cluster workload.
– Dynamic QoS management in service-based systems [15]. The web services used to carry out operations within service-based systems were chosen optimally from sets of functionally equivalent services characterised by different failure rates, response times and costs. A combination of discrete- and continuous-time Markov models and of PCTL and CSL properties were used to analyse the reliability- and performance-related system properties, respectively.

In all these applications, the computation overheads associated with carrying out the quantitative analysis experiments at runtime were acceptable for realistic system sizes. For the dynamic power management of a Fujitsu disk drive, the analysis took up to a few hundreds of milliseconds. This represents, for instance, a fraction of the 1.6 seconds required for the disk to transition physically from the idle state into the sleep state. The CPU overhead was in the range 1.5%-2.5%, which was deemed acceptable for this application [14].

In the case study involving the adaptive allocation of data-centre servers, the runtime quantitative verification took between 10–30 seconds for systems comprising up to tens of servers [19]. This response time was acceptable because it represented a small delay compared to the time required to provision a server allocated to a new cluster.

Finally, using the approach for dynamic web service selection in service-based systems was feasible for workflows comprising up to eight web service invocations. Note that many of the workflows in use by the scientific community today do not exceed this size. For instance, the study carried out in [15] found that over 70% of the bioinformatics workflows from the widely used myExperiment workflow repository[4] comprise between one and eight web service invocations.

2.3 Challenges

Reducing the overheads associated with runtime quantitative verification and scaling the approach beyond small to medium system sizes represents a ma-

[4] http://www.myexperiment.org

jor challenge. The options explored in the effort to address this challenge include a combination of software engineering techniques and novel quantitative verification algorithms [14, 15]. The former category of solutions includes the pre-execution and/or caching of quantitative analysis experiments, and the execution of the PRISM experiments for different requirements in parallel, e.g., on a multicore-processor server or on multiple machines. The latter category includes efforts for the design of iterative algorithms that derive the results of an experiment from those of the previous ones.

Another challenge is related to the expert knowledge that is required to build the models employed in the runtime quantitative analysis. Building a quantitative model at the right level of abstraction represents a non-trivial task that requires a good understanding of both formal modelling techniques and the behaviour of the target system. The applications described in the previous section were implemented by research teams with significant experience in formal methods, who invested significant time in understanding the behaviour of the computer systems involved in these applications. It is expected that an increase in the teaching of formal methods by undergraduate and graduate Computer Science programmes will enable future engineers of adaptive computer systems to apply this FM@R approach with less need for expert support.

One last challenge that is worth mentioning here is the potential need for a continual update of the model used by the approach. Systems that require the ability to adapt are often affected by internal changes that modify their behaviour in ways that may not or cannot be captured by a static model. While preliminary work to learn the parameter values for parameterised system models (cf. Figure 1) has been successful [20, 15, 21], there are multiple applications in which systems undergo unpredictable changes that require structural model changes (e.g., to reflect components leaving or joining the system dynamically). Devising monitoring techniques and learning algorithms capable of dealing with this scenario represents a major challenge.

3 Lightweight Formal Methods @ Runtime

3.1 Description

Lightweight formal methods represent techniques for the (often partial) specification of computer system requirements using mathematical notation drawn from set theory and first-order logic [22]. Their benefits include expressing system requirements concisely and unambiguously, and the ability to automatically derive artifacts guaranteed to satisfy these requirements. These artifacts can be fully-fledged software components or models of the system that the specification describes. The derivation process is termed *refinement* in the former case [23] and *model finding* in the latter case [24].

An FM@R approach that employs model finding was proposed in [25] and further developed in [26]. In this approach (Figure 2), a formal specification is obtained through combining formal descriptions of the characteristics, operations,

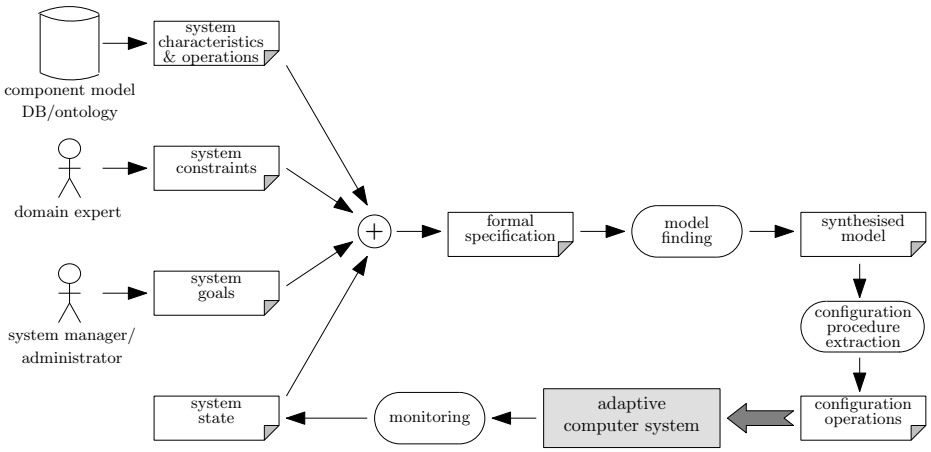

Fig. 2. Lightweight formal methods @ runtime

constraints, goals and state of a computer system. Model-finding techniques are then employed to synthesise a model of the system that satisfies this specification. The synthesised model corresponds to a system configuration that fulfils the system goals given its current state, and is used to derive a configuration procedure capable of reaching this target configuration without violating any system constraints.

The approach is applicable in system adaptation scenarios that:

1. require the synthesis of configuration change procedures (or *workflows*) capable of achieving given goal conditions;
2. involve the integration of information about the system components and constraints provided by multiple domain experts.

In such scenarios, extensive discussions among the domain experts are usually required to define a configuration change procedure that can achieve the system goal without violating any of its critical conditions. As these discussions are known to be time consuming and prone to errors [25], using the FM@R approach described in this section has the advantage of deriving a provably safe configuration procedure automatically. Furthermore, organising the information provided by different domain experts into a repository of formally and declaratively specified system constraints and objectives can encourage their reuse across applications from the same domain.

3.2 Practical Realisation

The variant of the approach presented in [25, 26] uses the formal specification framework Alloy [24]. The Alloy declarative specification language is used to define the formal specification in Figure 2, and the model synthesis is carried out

using the model finder Alloy Analyzer.[5] The Alloy formal specification comprises two parts:

1. A static part is used for the constant elements of the system. This part consists of Alloy `sig`(nature) and `fact` constructs defining the system characteristics, operations and constraints.
2. A second part of the specification is derived at runtime. This variable part consists of Alloy `sig`/`pred`(icate) and `fact` constructs that encode the system goals and state, respectively.

The model that the Alloy Analyzer tool synthesises when supplied with this specification is a sequence of value assignments to variables from the operation definitions in the specification. As a result, the model maps directly on to a sequence of configuration parameter changes satisfying all given `fact` declarations. This represents a sequence of feasible state transitions that starts with the current state of the system, terminates with a state that satisfies the system goals, and does not enter any state that violates system constraints.

The approach and its application in a case study involving virtual-machine consolidation within a cluster of physical servers are presented in detail in [26]. The experimental results in Figure 3 illustrate the computational time required to synthesise a configuration change procedure in this case study. This graph shows the relation between the size of the system (i.e., the number of system components) and both (a) the number of SAT clauses to which the system specification was reduced by Alloy Analyzer; and (b) the time spent to derive the result by the SAT solver employed by the tool. These experimental results show acceptable overheads for systems comprising up to 30-40 software and/or hardware components.

3.3 Challenges

Unsurprisingly, the main challenge of using lightweight formal methods at runtime is the limited scalability of the approach. Although systems with up to 40 components are not uncommon in real-world applications, extending the applicability of the approach to larger systems is key to its adoption.

Opportunities for taking advantage of this approach in larger computer systems do exist. One such opportunity is to improve the performance of the SAT solvers at the core of the model synthesis process. A second opportunity is to devise algorithms that translate formal specifications into sets of SAT clauses comprising significantly fewer elements than the sets produced by the algorithms currently in use within Alloy.

Some of the challenges described in Section 2.3 are also valid for the FM@R approach described here. They include the expertise required to derive an approapriate system specification, and the need to keep this specification in step with potential changes in the system characteristics and operations. The potential solutions for these challenges are those already presented in Section 2.3.

[5] http://alloy.mit.edu/alloy4/

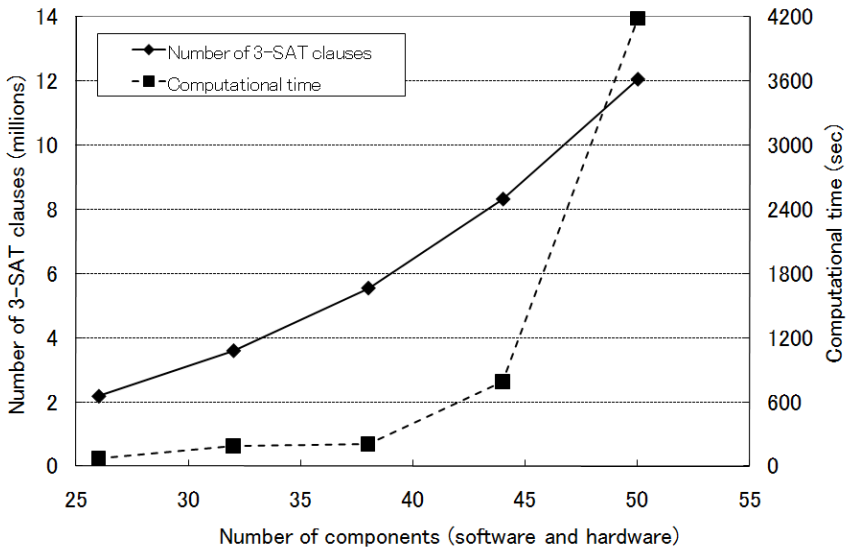

Fig. 3. The variation of the number of 3-SAT clauses and the computational time with the system size in lightweight formal methods @ runtime

4 Model Checking @ Runtime

4.1 Description

Model checking represents a formal technique for verifying whether a system satisfies its requirements [7]. The technique involves building a mathematically-based model of the system behaviour (e.g., a *Kripke structure* [7] or a *process algebra model* [27]), and checking that system properties specified formally hold within this model. For each refuted property, the technique yields a counterexample consisting of an execution path for which the property does not hold. The result is based on an exhaustive analysis of the state space of the considered model—a characteristic that sets model checking apart from complementary techniques such as testing and simulation.

Model checking @ runtime has the potential to play two key roles in adaptive computer systems. First, model checking techniques could be used to guide the adaptation process by means of an approach similar to the one described in context of quantitative verification in the previous section. Second, model checking can be used to address a major concern of adaptive computer systems, namely the correctness of the objectives specified by the administrators and users of these systems. These two applications of model checking @ runtime are detailed below.

Goal policies are often used to specify constraints, invariants or final-state conditions that an autonomic computer system is required to achieve by ap-

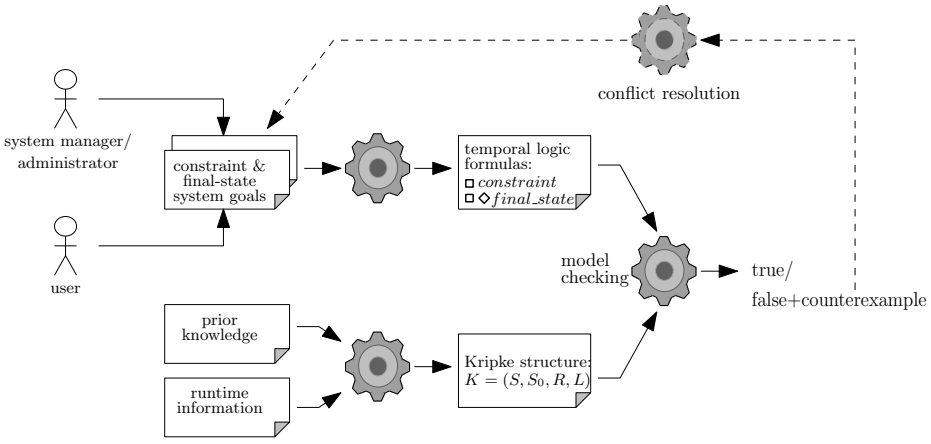

Fig. 4. Model checking for conflict detection and resolution in adaptive-system goals

propriately reconfiguring itself [13]. These policies comprise Boolean expressions similar to the formally specified requirements that model checking can verify given an appropriate model of the system behaviour. As self-adaptation requires choosing among a set of possible configuration, model checking @ runtime has the potential to confirm which of these configurations achieves the system goals. Similar to the FM@R solution described in Section 2.3, this would require the verification of the formally specified system goals against a parameterised family of system models. The feasibility of the approach—in terms of response time, overhead, scalability and expressiveness—is still to be determined.

The detection and resolution of conflicts in the objectives that developers, administrators and users specify for adaptive computer systems represents another area in which model checking @ runtime could provide effective solutions. Unlike the configuration procedure synthesis presented in Section 3, this approach is suitable for identifying the conflicts between atomic "if–then"-type rules and objectives to be achieved by an adaptive computer system. This application of model checking @ runtime is described in more detail next.

4.2 Practical Realisation

The model checking @ runtime approach in [28, 26] uses the model checker SPIN [29] to detect conflicts in the constraint and final-state goals of an adaptive computer system. Note that the model checking is performed at run time each time when the Kripke structure that describes the system behaviour changes to reflect the variable context that the system operates in. The counterexamples generated by the model checker for each identified conflict can be used to identify reachable system states that do not comply with its goals, and thus represent a starting point for resolving these conflicts as indicated in Figure 4.

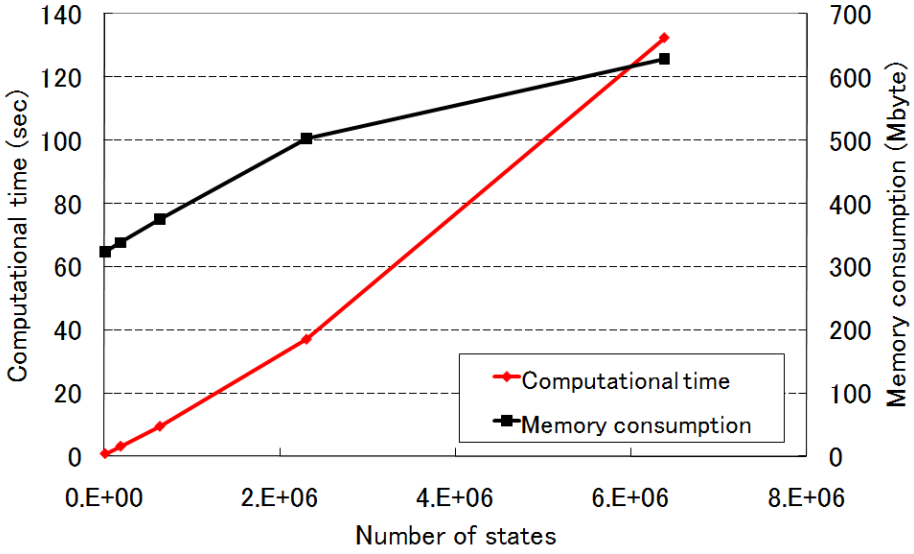

Fig. 5. Runtime detection of policy conflicts using model checking—CPU and memory overheads

Using the off-the-shelf model checker SPIN (i.e., a tool not intended for runtime use) and without effort to minimise the size of the model state space through techniques such as model abstraction [30], the approach was shown to be applicable to real-world adaptive systems comprising under ten components [28]. The experimental results from a case study involving the detection of policy conflicts in a utility resource management system [26] are shown in Figure 5.

4.3 Challenges

As for the other FM@R approaches described in the paper, the main challenge with model checking @ runtime is scalability. Work to address this challenge is still in its very early stages. The only promising result that the authors are aware of is the technique presented in [31], which aims at reducing the number of possible system configurations based on the system context or circumstances. The integration of this technique within the general FM@R approach from Figure 4 has the potential to reduce the size of the analysed Kripke structure significantly, and thus to render the approach applicable to larger adaptive systems.

5 Other FM@R-Related Approaches

Over the past decade, two research communities have carried out work that is relevant to the runtime use of formal methods.

The *Runtime Verification* research community[6] has been running annual workshops to explore techniques for the monitoring and formal analysis of program executions since 2001. The techniques developed by this community are concerned with verifying the correctness of formally-specified properties against individual execution traces or programs or *runs* [32]. This represents a powerful approach to detecting violations of correctness properties after they happen and their associated runs are recorded. As the goal is to analyse a single run at a time, these techniques scale well. However, unlike the model checking @ runtime approach presented in Section 4, runtime verification cannot guarantee the lack of constraint violations or help prevent their occurrence.

More recently, the *Models@Run.Time* community was formed to bring together researchers working on projects that involve the use of various types of models at runtime. A wide range of results including new techniques and applications have been presented at the "Models@run.time" workshops organised annually since 2005.[7] These results include the use of evolutionary computation techniques to synthesise models for evolving the configuration of systems that need to meet changing requirements [33].

A summary of the key techniques and approaches developed by the Models@Run.Time research community is available from [34]. In addition to presenting a general overview of using models in runtime environments [35], [34] describes an approach that uses an architectural model to automate configuration change in dynamically adaptive systems [31].

6 Conclusion

The computer systems of the future will increasingly face two conflicting challenges. On the one hand, they will need to adapt continually to change. On the other hand, they will be required to provide high integrity, availability and predictability. So far, the two challenges have been addressed largely in isolation, by different research communities. Adaptive computer systems have typically been produced through the application of heuristics, simulation and artificial intelligence techniques. In contrast, high integrity and predictability has been achieved through the application of formal methods—often to systems that operate in fixed scenarios, such as hardware components or communication protocols.

This paper advocates the integration of the two research areas. It is envisaged that the use of *formal methods @ runtime* (FM@R) has the potential to contribute to the solution of the dual challenge faced by future computer systems. The early FM@R research summarised in the paper suggests several ways of exploiting formal methods in the context of adaptive computer systems. Using off-the-shelf tools, these FM@R techniques were shown to be applicable to small or, in some cases, to medium-sized systems. In this respect, FM@R are in a position similar to that of model checking in the early to mid 1980s: the benefits of the approach are clear, but it is hard to confidently predict whether it can

[6] `http://runtime-verification.org`
[7] `http://www.comp.lancs.ac.uk/~bencomo/WorkshopMRT.html`

scale to large systems. Some of the avenues to explore in the search for a positive answer to this unknown have been suggested in the paper—more effective translators of formal specifications into SAT clauses (Section 3), iterative verification techniques (Section 2) and algorithms for reducing the state space of formal models (Section 4). Other potential FM@R research directions include taking advantage of assume-guarantee [36] and hierarchical [37] verification techniques in the runtime analysis of adaptive computer system models.

Pursuing these directions will require significant research effort, and achieving their objectives will take more than isolated breakthroughs. What justifies this undertaking is the promise of dependable and predictable adaptation—a major stepping stone in the human quest for non-biological systems capable of learning, adaptation, reasoning and planning.

Acknowledgements. This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1.

References

1. Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM **12**(10) (October 1969) 576–583
2. Floyd, R.W.: Assigning meanings to programs. Proceedings of the American Mathematical Society Symposia on Applied Mathematics **19** (1967) 9–31
3. US National Science Foundation: Model checking pioneers receive Turing Award, most prestigious in computing (February 2008) Press Release 08-022.
4. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)
5. Woodcock, J., Davies, J.: Using Z. Specification, Refinement and Proof. Prentice Hall (1996)
6. Lano, K.: The B Language and Method: A Guide to Practical Formal Development. Springer-Verlag (1996)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
8. Kropf, T., ed.: Formal Hardware Verification: Methods and Systems in Comparison. Volume 1287 of LNCS. Springer (1997)
9. Clarke, E.M., Lerda, F.: Model checking: Software and beyond. Journal of Universal Computer Science **13**(5) (2007) 639–649
10. Dai, Y.S.: Autonomic computing and reliability improvement. In: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05). (2005) 204–206
11. Sterritt, R., Bustard, D.: Autonomic computing — a means of achieving dependability? In: Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03). (2003)
12. Kwiatkowska, M.: Quantitative verification: Models, techniques and tools. In: Proc. 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM Press (September 2007) 449–458
13. Calinescu, R.: General-purpose autonomic computing. In Denko, M., et al., eds.: Autonomic Computing and Networking, Springer (2009) 3–30

14. Calinescu, R., Kwiatkowska, M.: Using quantitative analysis to implement autonomic IT systems. In: Proceedings of the 31st International Conference on Software Engineering (ICSE'09). (2009) 100–110
15. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimisation in service-based systems. IEEE Transactions on Software Engineering (October 2010)
16. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6**(5) (1994) 512–535
17. Aziz, A., et al.: Model checking continuous time Markov chains. ACM Transactions on Computational Logic **1**(1) (2000) 162–170
18. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In Hermanns, H., Palsberg, J., eds.: Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). Volume 3920 of LNCS., Springer (2006) 441–444
19. Calinescu, R., Kwiatkowska, M.: CADS*: Computer-aided development of self-* systems. In Chechik, M., Wirsing, M., eds.: Fundamental Approaches to Software Engineering (FASE 2009). Volume 5503 of Lecture Notes in Computer Science., Springer (March 2009) 421–424
20. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time parameter adaptation. In: Proc. 31st International Conference on Software Engineering (ICSE09), Los Alamitos, CA, USA, IEEE Computer Society (2009) 111–121
21. Calinescu, R., Johnson, K., Rafiq, Y.: Using observation ageing to improve Markovian model learning in QoS engineering. In: Proceedings 2nd ACM/SPEC International Conference on Performance Engineering. (2011)
22. Agerholm, S., Larsen, P.G.: A lightweight approach to formal methods. In: Proceedings of the International Workshop on Current Trends in Applied Formal Methods. Volume 1641 of LNCS., Springer-Verlag (October 1998) 168–183
23. Schneider, S.: The B-Method. Palgrave Macmillan (2001)
24. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)
25. Kikuchi, S., Tsuchiya, S.: Configuration procedure synthesis for complex systems using model finder. In: Proceedings of the 15th IEEE International Conference on Complex Computer Systems, Oxford, UK (March 2010) To appear.
26. Calinescu, R., Kikuchi, S., Kwiatkowska, M.: Formal methods for the development and verification of autonomic IT systems. In Cong-Vinh, P., ed.: Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification, IGI Global (2011) To appear.
27. Roscoe, A.W.: The theory and practice of concurrency. Prentice Hall (1998)
28. Kikuchi, S., Tsuchiya, S., Adachi, M., Katsuyama, T.: Policy verification and validation framework based on model checking approach. In: Proceedings of the 4th IEEE International Conference on Autonomic Computing, Jacksonville, Florida (June 2007)
29. Holzmann, G.J.: The SPIN Model Checker. Addison-Wesley (2003)
30. Wang, C., Hachtel, G.D., Somenzi, F.: Abstraction Refinement for Large Scale Model Checking (Series on Integrated Circuits and Systems). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
31. Morin, B., Barais, O., Jezequel, J.M., Fleurey, F., Solberg, A.: Models@ run.time to support dynamic adaptation. Computer **42**(10) (October 2009) 44–51

32. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming **78**(5) (May-June 2009) 293–303
33. Ramirez, A.J., Cheng, B.H.: Evolving models at run time to address functional and non-functional adaptation requirements. In: Proceedings of the Fourth Workshop on Models at Run Time, Denver, Colorado, USA, ACM (October 2009) 31–40
34. IEEE Computer: Special Issue on Models@Run.Time, **42**(10) (October 2009)
35. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10) (October 2009) 22–27
36. Pasareanu, C.S., Dwyer, M.B., Huth, M.: Assume-guarantee model checking of software: A comparative case study. In: Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking, London, UK, Springer-Verlag (1999) 168–183
37. Alur, R., Yannakakis, M.: Model checking of hierarchical state machines. ACM Trans. Program. Lang. Syst. **23**(3) (2001) 273–303