Towards usable and relevant model checking techniques for the analysis of dependable interactive systems

Karsten Loer and Michael Harrison
BAE SYSTEMS Dependable Computing Systems Centre
Department of Computer Science, University of York
York, YO10 5DD, UK
{Karsten.Loer, Michael.Harrison}@cs.york.ac.uk

Abstract

Model checking is a formal technique for the automated analysis of system models against formal requirements. Once a suitable model and property have been specified, no further interaction by the analyst is required. However, this does not make the method necessarily user friendly since the checker must be provided with appropriate and complex input data. Furthermore, counter-examples generated by the system are often difficult to interpret. Because of this complexity, model checking is not commonly used, and exhaustive exploration of system models based on finite state descriptions is not exploited within industrial dependable systems design. The paper describes the development of an integrated collection of tools around SMV, intended to make it more accessible to practicing software engineers and in particular those concerned with the human interface issues in complex safety critical systems.

1 Introduction

An obstacle to the take-up of formal methods is the incomprehensibility of the notations and tools that underly them. In practice only the originators of the methods or committed (usually academic) users will accept the cost of them. Model checking, a process that involves the exhaustive analysis of finite state descriptions appears to be a promising approach to making the benefits of formal methods more accessible to designers.

The paper focusses specifically on one such tool, the SMV model checker and its derivatives [Cimatti et al., 2002]. It describes the development of an integrated system based on SMV, intended to make it more accessible to practicing engineers (a group of engineers developing human computer interfaces within the avionics industry). The integrated system includes interfaces that make information available to designers in a comprehensible form. Although model checking has usability advantages because it is a decidable approach to analysis, there are also disadvantages.

1. The initial specification of the model is expressed as a state transition diagram using notations such as SMV. These notations must be learnt and may be counter-intuitive particularly if, as in our case, designers are human factors experts. The number of states in these models can quickly explode and techniques must be adopted to manage the number of states.

2. The notation for specifying the properties, usually modal or temporal logic, is difficult to understand and to apply. Only a subset of possible property types are available, for example representational properties are not possible.

3. The form of the result is difficult to interpret. The answer is either true or a trace is presented which is a counter-example. The answer *true* may actually mean that the property has been wrongly formulated and is therefore vacuously true. Even when false, it is difficult to make sense of the traces that arise as counter-examples. In practice because model checking is an iterative process involving a process of property refinement it is important to make these counter-examples as helpful as possible.

2 System models and property patterns

Models involved in the development or derivation of requirements in the avionics and automotive domains are often expressed as statecharts [Harel, 1987]. Specifications of system models are either validated by structured analysis, for example by simulation or testing, or verified by formal proof.

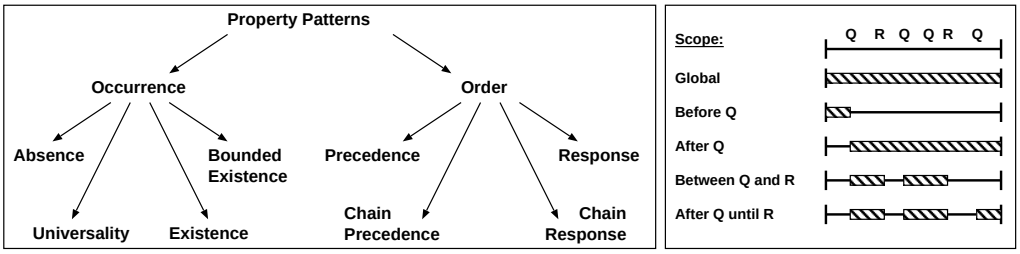

Figure 1. Property specification patterns [Dwyer et al., 1999].

Property	CTL template for property
reachability	AG EF(⟨*target_configuration*⟩)
mutual exclusion	AG !(⟨*configuration1*⟩ & ⟨*configuration2*⟩)
robustness	INVAR !⟨*triggering_condition*⟩; AG (⟨*starting_configuration*⟩ -> EF(⟨*target_configuration*⟩))
effect visibility (appropriateness)	AG(⟨*starting_configuration*⟩) -> AF(⟨*display_configuration*⟩)
(timeliness)	AG(⟨*starting_configuration*⟩ & ⟨*input_signal*⟩) -> AF(⟨*target_configuration*⟩ & *event_counter* = *n*)
recoverability	AG(⟨*starting_configuration*⟩ & ⟨*input_signal*⟩) -> EX EF(⟨*starting_configuration*⟩)
consistency	AG (⟨*input_signal*⟩) -> AX(⟨*intended target configuration*(*s*)⟩)
flexibility	AG(⟨*starting_configuration*⟩ & ⟨*reset_counter*⟩) -> EF(⟨*target_configuration*⟩ & ⟨*input_counter* = *m*⟩);

Table 1. Usability queries [Loer and Harrison, 2001].

In the context considered, Statecharts are used to explore issues such as moding behaviour. Degani in his OFAN approach [Degani, 1996] has imposed additional structure on Statecharts to make their relevance to interactive systems more explicit. This is achieved by decomposing statecharts into orthogonal sub-states modelling: *control elements*– description of the control elements; *control mechanism*– model of the device functionality; *displays*– description of the output elements; *environment*– model of relevant environmental properties; *user tasks*– sequence of user actions that are required to accomplish a certain task. Constraints are established on these models which represent critical circumstances relevant to what the user can do for example when the mode changes or where there is no possibility of simple recovery. In order to explore all possible (sensible and non-sensible) user inputs exhaustively the system is often analysed with a limited task model with the aim of analysing paths that broadly follow likely user behaviours.

Usability engineers often work with requirements that are derived from generic design principles and guidelines [Dix et al., 1998, chap.4], and usability heuristics [Nielsen, 1992]. Formulating these requirements, particularly as temporal logic formulae required by model checkers can be difficult not just for usability engineers but also software engineers. To ease this translation Dwyer *et al.* suggest "patterns" of properties of finite-state verification [Dwyer et al., 1999]. From an extensive literature review a list of 555 property specifications have been extracted with mappings to different formalisms (CTL, LTL, Quantified Regular Expressions). Most of these specifications can be assigned to one of eight patterns in the context of five different scopes (i.e. the range of a model execution where a pattern must hold) – see Figure 1. In the interactive systems domain, efforts to use the generic design principles to do this analysis more systematically include [Paternò, 1996, Campos, 1999]. Through the use of OFAN all models have the same top-level structure and generic templates can be based on the common structure. For example, [Loer and Harrison, 2001] suggest the usability templates listed in Table 1. Some of these templates are covered by, or can be derived from Dwyer *et al.*'s property specification patterns. It should be noted, that although the OFAN structure supports the instantiation of usability properties, the toolkit is applicable to statecharts in general.

3 Easing diagnosis based on output: trace visualisations

Once a model of the system and requirements have been given to the SMV model checker they can then be used to verify the requirements in every possible execution state. If a property does not hold, a trace is produced that gives a sequence of steps leading from an initial state to a sys-

tem state that violates the given property. A correctly interpreted trace can be used to assess how to modify either the model of the system or the specification of the requirement. Alternatively, the analyst might decide that the result is interesting but should be investigated from a different perspective using a different method. The raw traces produced by the model checker do not support this decision process adequately. A tool is therefore required for the visualisation of the data in the trace. In practice, usability engineers wish to consider these traces as scenarios and in order to do so add contextual information based on domain experience describing possible situations that are of interest.

SMV tools produce tables as output that present the values of variables against each step. Such tables are suitable for a mechanical analysis, but hard to read by human analysts. In a user study (using a technique [Monk et al., 1993] for getting feedback from users) a variety of visualisations were evaluated. During the session aerospace engineers and critical system researchers used prototype versions of the system focussing mainly on the output displays and provided feedback about several notations. In addition to tables other visualisations were investigated, including natural language scenario templates, scenario scripts, message sequence charts, operational sequence diagrams (OSDs) [Beevis et al., 1994], and model animation. The participants found tables and OSDs to be the most useful static scenario visualisations.

4 The IFADIS toolkit

A prototype of an integrated toolkit has been developed that performs or supports the tasks of model translation, property development, and trace visualisation. A statechart model of the system is produced (using the STATEMATE toolkit [Harel et al., 1990]). The model is imported to the IFADIS tool and a SMV model is automatically generated for both, the SMV version of Cadence Berkeley Labs[1] and NuSMV [Cimatti et al., 2002]. This translation is performed by a compiler using the algorithm described by [Clarke and Heinle, 2000].

The analyst then identifies the kind of property to be analysed using either Dwyer's patterns for the analysis of system properties or a list of templates for usability properties. Once chosen, the scope under which the property is supposed to be analysed is selected. The system retrieves the appropriate temporal logic template and asks the user to instantiate it with the appropriate variables extracted from the system model (see Figure 2). Alternatively, the analyst can choose from more abstract properties (e.g. *"Can all states be reached?"*, "*Can all events in module X be fired?*") which are then instantiated automatically by the tool.

[1] See http://www-cad.eecs.berkeley.edu/~kenmcmil/

The system then checks the property and output is presented currently using an enhanced tabular view. An optional process diagram in the top part of the tool displays where the user currently sits in the process.

5 Conclusions and future work

A framework and a prototype implementation for the analysis of dependable interactive systems has been developed. The framework supports property checking for dependable systems in general and some aspects of usability analysis in particular. It was developed with the requirements of industrial designers in mind. The proof of concept prototype has been customised to the requirements of a particular aerospace environment but we believe the ideas are applicable to a wider field.

Future work will concentrate on more sophisticated proof strategies that make use of fairness constraints and invariants. More systematic approaches that guide different user groups in property selection using "wizards" will be developed. The tool should not only support the analysis in different stages of the process but integrate effectively into the development process. For this reason a proof management system is desirable which will, among other things, keep track of the analysis history. Since many of these managerial issues depend on company procedures, or even project-specific procedures, the tool should become more customisable and be sufficiently flexible to cope with changing demands.

6 Acknowledgements

This work is funded by the BAE SYSTEMS Dependable Computing Systems Centre. We thank the BAE SYSTEMS engineers for their support.

References

[Beevis et al., 1994] Beevis, D., Bost, R., Döring, B., Nordø, E., Oberman, F., Papin, J.-F., Schuffel, H., and Streets, D. (1994). Analysis Techniques for Man-Machine Systems Design. Technical Report AC/243(Panel 8)TR/7, North Atlantic Treaty Organization, Defence Research Group.

[Campos, 1999] Campos, J. C. (1999). *Automated Deduction and Usability Reasoning*. PhD thesis, Department of Computer Science, University of York, UK.

[Cimatti et al., 2002] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV 2: An Open Source Tool for Symbolic Model Checking. In

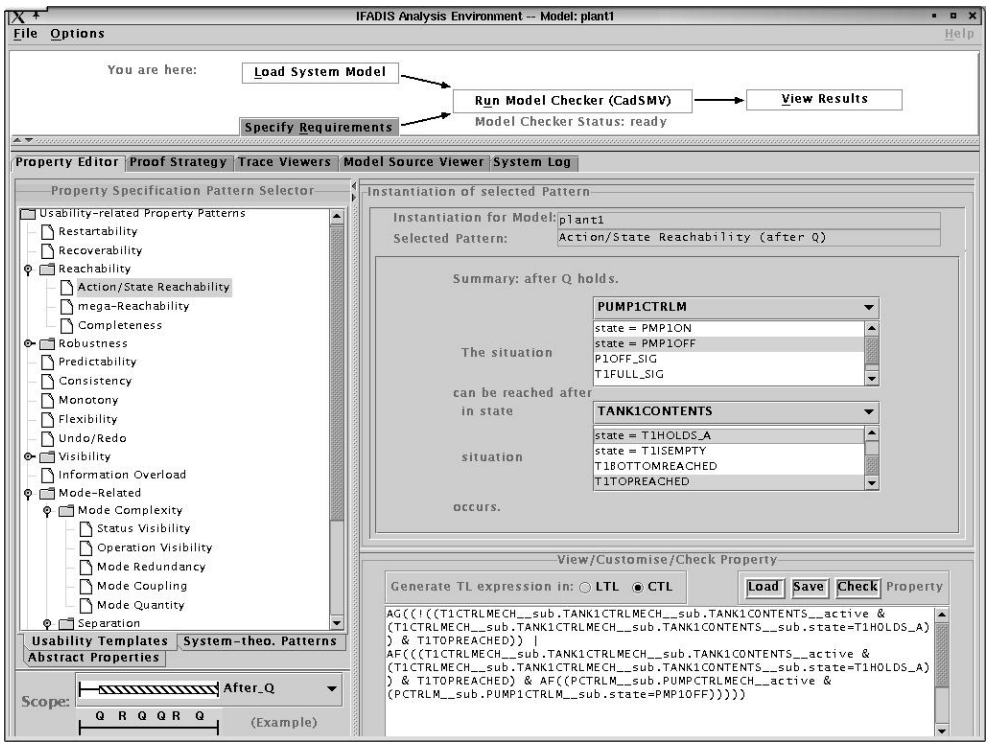

Figure 2. IFADIS **property editor (Usability templates and strategies).**

Computer-Aided Verification (CAV '02), Lecture Notes in Computer Science. Springer-Verlag.

[Clarke and Heinle, 2000] Clarke, E. and Heinle, W. (2000). Modular Translation of Statecharts to SMV. Technical Report CMU-CS-00-XXX, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

[Degani, 1996] Degani, A. (1996). *On Modes, Error, and Patterns of Interaction*. PhD thesis, Georgia Institute of Technology.

[Dix et al., 1998] Dix, A., Finlay, J., Abowd, G., and Beale, R. (1998). *Human Computer Interaction (2nd edition)*. Prentice Hall Europe.

[Dwyer et al., 1999] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1999). Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering, Los Angeles, California*.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, pages 231–274.

[Harel et al., 1990] Harel, D., Loachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., and Trakhtenbrot, M. (1990). STATEMATE: A Working Environment for the Development od Complex Reactive Systems. *IEEE Transactions on Software Engineering*, 16(4):403–413.

[Loer and Harrison, 2001] Loer, K. and Harrison, M. D. (2001). Formal interactive systems analysis and usability inspection methods: Two incompatible worlds? In Palanque, P. and Paternó, F., editors, *7th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS 2000)*, volume 1946 of *Lecture Notes in Computer Science*, pages 169–190. Springer-Verlag.

[Monk et al., 1993] Monk, A., Wright, P., Haber, J., and Davenport, L. (1993). *Improving your human-computer interface: a practical technique*. Prentice-Hall.

[Nielsen, 1992] Nielsen, J. (1992). Finding usability problems throught heuristic evaluation. In *Proc. of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 249–256, New York. ACM.

[Paternò, 1996] Paternò, F. D. (1996). *A Method for Formal Specification and Verification of Interactive Systems*. PhD thesis, Department of Computer Science, University of York, UK.