Integrating Model Checking with the Industrial Design of Interactive Systems

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

Abstract

This paper is concerned with the introduction of model checking – in particular the SMV model checking tools – into an industrial process for the design of interactive systems. Using the example of an avionics design environment, differences between actual design practice and the productive application of model-checking are explored. The paper presents a design framework which aims to bridge the gap between company practice and tool requirements. The results of a co-operative evaluation of a tool prototype that supports this framework are presented.

Keywords: Model checking, usability, industrial design, human-computer interaction

1 Introduction

A primary concern during the design of (interactive) systems for safety-critical applications is to identify and, wherever possible, eliminate errors in all conceivable circumstances. In current practice, systems engineers often use empirical analysis and scenario walk-throughs to visualise different applications of the system under design. These techniques are performed on selected instances of system use. Consequently, there is a danger that potentially hazardous situations are ignored or overlooked.

Model checking explores the execution space of a model exhaustively. In the design of interactive systems the technique can be useful to discover "latent" errors [Reason, 1990]. Within the academic community, model checking has been applied successfully, for example, to the analysis of mode confusion errors in modern "glass cockpit" aircraft [Rushby, 2002]. It has also been argued that in the field of human-computer interaction (HCI) design, model-checking analysis can support so-called "discount" usability evaluation techniques like "heuristic evaluation" [Nielsen, 1992], as is demonstrated in [Loer and Harrison, 2001].

In principle, model checking has much to offer to designers of interactive systems. However, in practice there is a gap between the artefacts used by design teams (including, for example, paper prototypes, simple models of system behaviour, and natural language requirements documents) and the inputs required by model-checking tools (i.e. finite-state system models and property specifications formulated in temporal logic or state automata). Furthermore, these inputs often need adjusting in order to achieve effective and efficient analysis. Following the solution outlined in [Loer and Harrison, 2002], a tool is presented that bridges this gap for a particular design setting in the avionics industry.

The remainder of this paper is organised as follows. In Section 2 the Integrated Framework for the Analysis of Dependable Interactive Systems (IFADIS) and a supporting tool prototype are presented. This tool provides a user interface to two contemporary implementations of the SMV model checker, Cadence SMV [Cadence Berkeley Laboratories, 2000] and NuSMV [Cimatti et al., 2002]. The results of the co-operative evaluation of the framework using the participation of industrial system engineers are presented in Section 3.

2 The IFADIS framework and tool prototype

The aim of IFADIS is to provide an interface layer (Figure 1) that makes it possible for systems engineers to apply model-checking tools without needing to know specialist notations. In order to achieve this aim, it is necessary to supply [Loer and Harrison, 2002]:

- a mechanism for the translation of system representations to SMV models,
- support for capturing natural language requirements specifications as temporal-logic formulae, and
- visualisations of model-checking results in notations that are understood by designers.

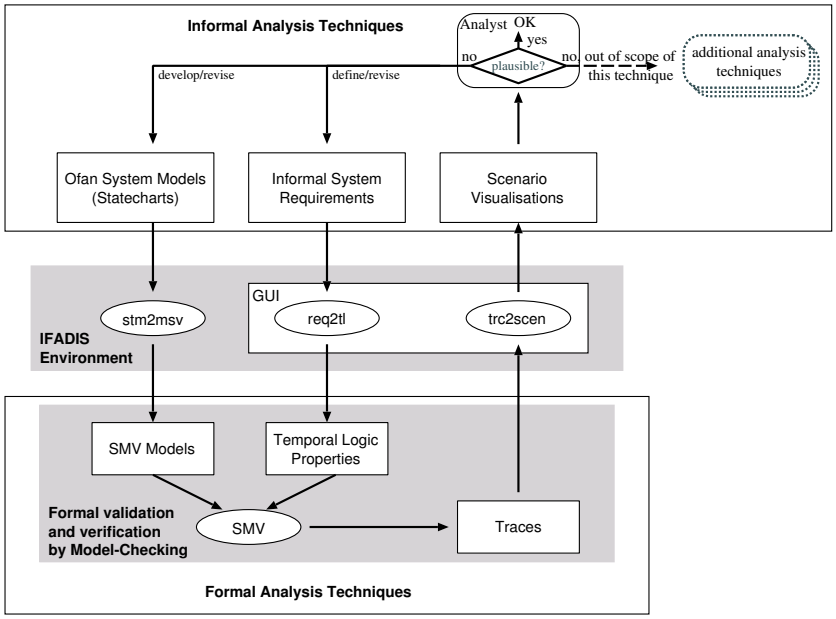

Figure 1. An implementation of the IFADIS framework for an industrial analysis process.

2.1 Model import from STATEMATE

In the automobile and avionics industries statecharts [Harel, 1987] are a commonly accepted notation for describing system behaviour [Horrocks, 1999] and even aspects of the user interface. In order to apply statecharts in interactive systems design it has been suggested that additional structure may be added by using the OFAN technique [Degani, 1996]. The IFADIS tool interfaces with the iLogix STATEMATE[1] tool via model export. The import of a STATEMATE model (Step ① in Figure 2) into the IFADIS tool involves – invisible to the tool user – a call of the `stm2smv` compiler that translates statechart models to SMV models. In this compiler an extended version of the algorithm by [Clarke and Heinle, 2000] is implemented; for a detailed description, see [Loer, 2003, Chapter 4].

2.2 Temporal logic property editor

System requirements are commonly provided as natural language documents. Many of these requirements are expressible more formally as relationships between sequences of system states and events. The IFADIS property editor supports the designer in mapping requirements to a set of specification templates that capture relationships between model states that represent "situations" in a modelled world. Once a model is loaded, the editor pane can be activated via the task tab or the process diagram (Step ②). In a third step the analyst can select the preferred domain-specific perspective (as independent "Usability Templates" or "System-theoretic Patterns"). Each pane provides a catalogue of property templates. The "system-theoretic" pane (Figure 2) implements the property specification pattern system of [Dwyer et al., 1999]. The "usability templates" pane (Figure 2) implements some of the templates given in [Loer, 2003, Chapter 5] which are based on a selection of the usability guidelines published, for instance, in [Dix et al., 1998], [Nielsen, 1992], or [Smith and Mosier, 1986]. The aim is to support the analyst in finding the template that appropriately matches the requirement to be specified. Therefore, for each pattern and template an abstract natural language summary is provided in the instantiation pane on the right. Once a template is chosen (Step ④), a pattern scope [Dwyer et al., 1999] can be selected (Step ⑤). The selection of a property and a scope clearly identifies the corresponding template, which is displayed in the instantiation pane. In the instantiation process (Step ⑥) the analyst is provided with a more detailed natural language description of the property. The user needs to define the predicates `P`, `Q`, `R` and `S` in the templates by mappings to system states ("situations") in the current system model. For each predicate, a pop-down menu is provided for selecting a non-basic parent state. On selection of a state in a list-box all states, events, and data elements are displayed that occur within transition labels of that state. As soon as one or more of these elements are selected, a corresponding boolean expression is generated. In this step the property is enhanced automatically by auxiliary variables that are introduced by the `stm2smv` compiler in order to

[1] `http://www.ilogix.com`

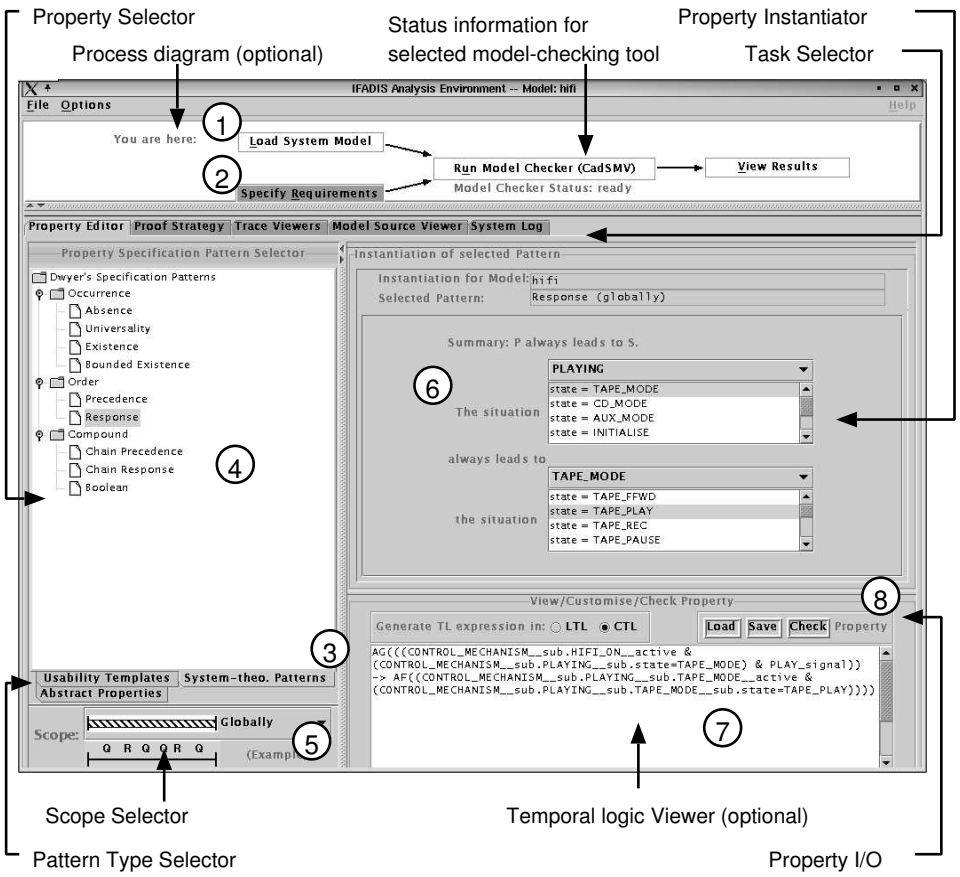

Figure 2. The user interface of the IFADIS property editor.

capture the hierarchy and activity of the statechart semantics in the SMV model. These system-internal additions are necessary in order to match the selected statechart elements in the property specification with corresponding elements from the SMV translation of the statechart. The resulting temporal-logic formula can be displayed as an LTL or CTL expression (Step ⑦). The instantiated formula can then be sent to the model checker, or saved for future use (Step ⑧).

It can be argued that this browsing process not only helps to find a desired property, but can also point the analyst to additional properties that were not considered previously.

The analysis of some requirements, such as sanity checks like state/event reachability (“*Are all states/events reachable?*”) requires the repeated analysis of the same property with different predicate instantiations. In order to save the analyst time, in Step ② the “Abstract Properties” pane can be selected. This pane offers a set of properties that automatically generate sets of basic properties by traversing the system model hierarchy.

Proof strategy editor

Cadence SMV supports LTL specifications to specify the expected behaviour of variables that are controlled outside the specified system model. Such specifications can then be used as assumptions during the analysis of further LTL properties. Analysis under assumptions is performed with the IFADIS tool by calling the “Proof Strategy” task pane (see Figure 3, Step ①). Selecting an LTL property to be proved from the property specification list (Step ②) leads to the generation of a multiple-choice list of (previously defined) LTL properties that can be used as assumptions. In the course of the selection process (Step ③) a summary of the property expression to be checked is produced (Step ④). Once the appropriate selections are made, the strategy can be stored or sent to the model checker (Step ⑤).

Analysis under assumptions is important in interactive systems design because models are used that specify the device behaviour, but are open with respect to user/environment inputs. Model checking here is not used for formal verification, rather is used to explore device response under certain circumstances but in a rather explo-

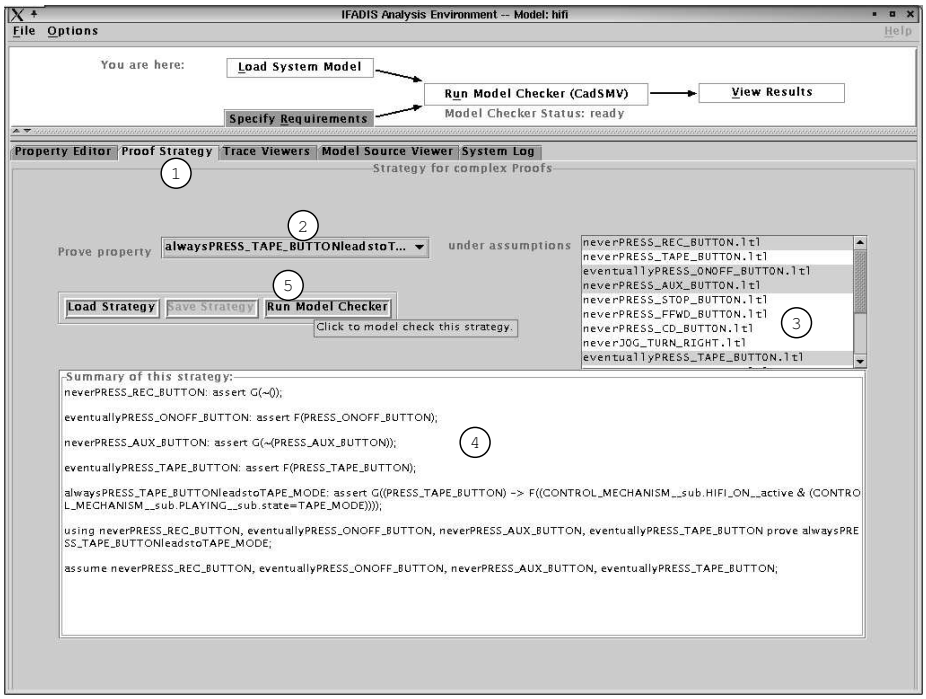

Figure 3. Screen-shot of proof strategy pane.

rative manner in order to gain an understanding for device responses under different assumptions about user input behaviour (for an application, see [Loer and Harrison, 2003]).

2.3 Trace visualisation

Model-checking traces are probably the most useful result delivered by a model checker, as they demonstrate *why* a property holds (or fails) as opposed to merely *that* it holds (fails). In this respect, traces can form the backbone for scenarios [Carroll, 1995] that capture an instance of wanted/unwanted behaviour of the modelled system. However, in order to be useful to a designer, the information that is contained in a trace needs to be visualised in a meaningful notation. A range of notations capture system behaviour and interactions (see Section 3.2). Which notation provides the most appropriate visualisation depends on the individual designer's background.

So far the visualisation tool of IFADIS only contains an engine for table visualisations, that highlights changes in the standard trace descriptions delivered by the model checkers. In [Kermelis, 2003], a further improvement of this visualisation engine is presented which implements features including the hiding of states and variables and the comparison of traces. These features were considered to be desirable by the correspondents used during the tool evaluation.

3 Co-operative Evaluation of the tool

The property editor and candidate notations for the trace visualisation were investigated separately in co-operative evaluations (for detailed results, see [Loer, 2003, Ch. 5 and 6]).

3.1 Evaluation of the property editor

Four human-factors engineers (two of whom had experience in systems engineering), one industrial systems engineer, all from the same company, and one academic HCI expert participated in a co-operative evaluation [Monk et al., 1993] of the tool. None of the participants had previous model-checking experience, but all had previously attended talks about the IFADIS framework. In a 30 minute introductory talk the prototype of the IFADIS environment and its anticipated application in the design process was described. Key features of the supporting prototype tool were also demonstrated. After this briefing, two fifty-minute evaluation sessions were held with the aim of obtaining initial feedback from potential users about the tool's usability and the support it might provide for their everyday work.

The results of the evaluation are not representative, given the short familiarisation phase, the informality of the evaluation session, the limited number of participants and the time limitations imposed by the workshop programme.

However, the received comments were valuable for future tool improvements. Issues that were raised primarily address the tool usability and work-flow. Organisational requirements about applying the tool during everyday work were also discussed.

Tool usability

All participants seemed to be comfortable with the user interface. The navigator bar on top of the screen was regarded as useful to determine where in the model-checking process the analyst was. However, during the selection and instantiation of requirements property templates several problems occurred.

The systems engineer easily identified a range of properties that were suitable to check the functional properties of interest, for example reachability, deadlock freedom, recoverability. For usability experts this task appeared to be more difficult. They prefer to work with differently oriented functional properties, such as "efficient menu structure", and non-functional properties, like "ease of use", and "intuitive operation". These properties are only partially supported by the tool. Moreover, the analysis of the properties that are supported makes it necessary to map abstract requirements to sets of specific instances in terms of model configurations. This problem can be dealt with by developing algorithms that implement more abstract requirements (for instance: "all states can be reached", "certain groups of inputs are not available once a system enters a particular mode"). All participants felt that, for an understanding of property scopes, more familiarisation with the tool would be required.

In the instantiation step, participants who regularly use state-based models worked more efficiently than participants who only used such models occasionally. Those usability experts, who had no systems engineering background, required an explanation about the meanings of expression predicates `Q`, `R`, `S`, `T` and about what they were supposed to do.

After a familiarisation phase the participants' performance in selecting elements for instantiation via the multiple-choice boxes improved. Two subjects commented that, despite the fact that the state hierarchy of the model is mapped to the sequence of states in the selector boxes, it was not trivial to retrieve model elements in the instantiation pane. As an improvement of the instantiation task, it was suggested that it should be possible to select model configurations for each predicate in a template from a graphically presented statechart.

It was also felt that a help system, and an electronic guide, like a wizard, for instance, would be a useful addition.

Finally, it was felt that the tool gives only limited feedback on model-checking progress. Technically, the progress of the model checker is hard to determine, because the tool runs in an external process that gives only limited indication of progress. In particular the time to completion of the model-checking algorithm cannot be determined in advance.

Work-flow and work context

Once the system model was imported into the tool, the usability experts had problems determining what further steps were required to perform the specification task. The participants suggested hiding parts of the interface until they become necessary for the task. It was also suggested that a display should be provided that would give hints on what to do next (for instance, a text-box displaying hints "select property template", "select scope", "instantiate selected template"). This problem seemed to be exaggerated by a lack of expertise in the use of model checking and formal analysis. The systems engineer, who had some experience of formal analysis, coped much better with this task.

The systems engineer pointed out that checking individual properties is only a part of his everyday work. In order to gain and maintain an overview of the overall analysis process a record of what properties already have been checked would be required as well as what properties remain to be checked.

One usability expert raised the concern that by introducing this tool, an engineer might "lose touch" with the model, and the ability to identify problems manually in the statechart. It can be argued that none of the previous analysis capabilities are lost, because the IFADIS environment is an additional aid only. However, this concern needs to be investigated further.

There were also questions about how to integrate the IFADIS environment with additional CASE tools. Adding support for other CASE tools would require additional compilation facilities.

Organisational requirements

It was noted that the terminology used in the tool might not be familiar to project teams. In particular, project-specific terminology sometimes differs from the standard HCI terms used in the usability template pane. The engineers suggested making the user interface more customisable by allowing analysts to rename and add property templates.

There was also discussion about how the tool could support designers in demonstrating to authorities that a product was analysed to a sufficient extent. It was suggested that a proof manager that keeps track of the properties that have

already been checked could probably provide some indication of the coverage of the analysis. This issue was not considered during design of the tool and should be explored in detail in future work.

Some of the suggested changes and features were implemented in a subsequent version of the prototype even though the participants of the co-operative evaluation are not a representative sample. Furthermore, an optimal solution is unlikely, because even within project teams in a single company, work practices vary significantly. Therefore, there might be merit in customisability though the need to provide a tool that can be shared between users is important.

3.2 Evaluation of candidate trace visualisations

A preliminary field study was conducted to find which trace visualisations supported the everyday work of a group of practitioners. Again, a co-operative evaluation technique [Monk et al., 1993] was applied. In this study, five participants (aerospace engineers with some background in human-factors engineering) were used in a co-operative evaluation where four participants were academics who conduct HCI research. The relevance of visualisations depends on the stakeholder's background and the application domain. This qualitative study was designed to elicit some sense of the orientation of these two communities. The group of participants was provided with background information about the IFADIS approach and environment. A simplified statechart model of the vertical autopilot of flight management system was then introduced. A pretty-printed excerpt of an SMV trace for a property ("*Can some target altitude always be reached, if in situation X some mode Y was used?*") was also provided. Different visualisations of this trace were offered, including data tables, scenario templates [Pocock et al., 2001], scenario scripts [Potts et al., 1994], and sequence diagrams [Rumbaugh et al., 1998, Chapter 8]. The participants were asked to assess the suitability of each notation in a questionnaire. During the ensuing discussion, particular emphasis was given to the kind of information a notation can, or should, provide. For the assessment of interactive systems the following aspects were considered to be relevant: "activity of human agents", "frequency of use of devices", "presentation of data values", "causal dependencies between activities", and "scalability of the representation".

The engineers also suggested additional visualisations which are used in their work environment. In particular, Operational Sequence Diagrams (OSDs, [Kurke, 1961]) were suggested, as well as model animation – that is, feeding the trace information back into the STATEMATE animator, for instance as offered by the tool described in [Mikk et al., 1997]. These visualisations were assessed with respect to the aspects listed above. The results of this discussion are summarised in Table 1.

From the study it was informally concluded that in this particular design context variable tables, OSDs, and animation appear to be preferrable.

4 Conclusions and future work

This paper discussed challenges concerning the integration of model-checking into industrial design processes. A tool is presented that aims to support industrial aerospace engineers. In particular, a compiler has been developed that makes it possible to check models that are formulated as statecharts, a notation that is already familiar to these engineers. The task of specifying property specifications is supported by a browser-style front-end that helps choose temporal logic specifications that capture the desired requirements for which the model is to be checked. The property editor supports the instantiation of templates with system states given by statechart configurations, so the SMV translation of the model remains hidden from the designer. A visualisation of model-checking traces as enhanced tables is provided by the tool and the suitability of selected further visualisations has been discussed.

The evaluation in co-operation with engineers and academics gives confidence that the tool improves the usability of the model checker for non-experts. However, the evaluation of the tool also suggested requirements for future work, some of which might be considered relevant by the model-checking community. Selected issues raised by the evaluation are as follows:

Improving user feedback: The feedback given by the model checkers makes it hard to judge the progress of the analysis. Some indication of progress needs to be provided, in order to make it possible for the user to schedule work. Such feedback can be based on knowledge of the model size and structure, as well as the property type.

Additional trace visualisations should be implemented to make the analysis results more accessible. In the evaluation it was suggested that OSDs and model animations would be most appropriate for this design setting. OSDs and model animation present different problems as far as producing a tool is concerned. In particular OSDs would require additional semantic information.

Improving user guidance: Rules need to be developed to guide the user with respect to the analysis schedule ("What properties have been checked already?", "Given this knowledge and an analysis goal, what properties remain to be checked?"; if there are logical dependencies between property specifications: "Which of the remaining properties should be checked next?"). Solutions to this non-trivial issue could also help to further automate the generation of property specifications.

Using the SMV tools more efficiently: So far the

representation conveys	Scenario Template	Scenario Script	Sequence Diagram	OSD	Variable Table	State-charts	Model animation
activity/workload	- -	++	++	++	-	+	+
frequency of use of devices	-	-	++	++	+	-	++
temp. progression	-	-	++	++	-	- -	++
data values	- -	+	-	++	++	- -	++*
scalability	+	- -	- -	-	++	- -	-
causality	- -	- -	++	++	-	- -	++

Key: ++ very good, + good, - bad, - - very bad, * data monitoring tool integrated in Statemate

Table 1. Summary of visualisation properties.

model-checkers are only used in batch mode. An improved performance was achieved by activating variable re-ordering (option "`-sift`" of Cadence SMV and option "`-dynamic`" of NuSMV). Consequently, the time-consuming process of producing the BDD is repeated every time an analysis is started. If the interactive mode of the tools would be used, this step would only have to be performed once at the time a new model is imported.

Improving system models: The statechart models used were developed for a requirements elicitation and validation purpose and therefore are not optimised towards model checking[2]. To avoid state explosion, the integration of state-reduction techniques, such as data abstraction, are required.

In summary, the results are promising in that they lead to a prospect that non model-checking experts have the potential to make effective use of these tools in order to analyse their systems more effectively.

5 Acknowledgements

This work was funded by the BAE SYSTEMS Dependable Computing Systems Centre (DCSC[3]). and the EPSRC-funded Interdisciplinary Research Collaboration in Dependability (DIRC[4]).

About the authors

Karsten Loer is post-doctoral researcher in the Human-Computer interaction group at the Department of Computer Science at the University of York. From 1998 to 2002 his research in the DCSC was concerned with the introduction of model-checking techniques into industrial design environments for the development of interactive systems. More recently, he worked the DIRC project on the application of model checking for the analysis of real-time aspects of user interfaces and to improve their support for multi-valued decision making.

Michael Harrison is Professor of Human-Computer Interaction at the University of York. His research interests include the development and application of formal and informal modelling and analysis techniques to support the design of interactive systems. In this context he is also concerned with allocation of function in designs, and the production and re-use of qualitative arguments to support the design and certification process.

[2] Admittedly, one could argue that a model is produced for a purpose, so the re-use of models might not even be desirable, particularly if it yields technical difficulties like state explosion.

[3] `http://www.cs.york.ac.uk/hise/dcsc`

[4] `http://www.dirc.org.uk`

References

[Cadence Berkeley Laboratories, 2000] Cadence Berkeley Laboratories (2000). Cadence SMV Homepage. `http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/`.

[Carroll, 1995] Carroll, J. M. (1995). *Scenario Based Design: Envisioning Work and Technology in System Development*. John Wiley & Sons.

[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 Larsen, K. G. and Brinksma, E., editors, *Computer-Aided Verification (CAV '02)*, volume 2404 of *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). *Modeling Human-Machine Systems: 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*. Prentice Hall Europe, 2nd edition.

[Dwyer et al., 1999] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1999). Patterns in property specifications for finite-state verification. In Garlan, D. and Kramer, J., editors, *21st International Conference on Software Engineering, Los Angeles, California*, pages 411–420.

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

[Horrocks, 1999] Horrocks, I. (1999). *Constructing the User Interface with Statecharts*. Addison Wesley.

[Kermelis, 2003] Kermelis, M. (2003). Towards an improved understanding of model-checking traces by visualisation. Master's thesis, Department of Computer Science, University of York, UK.

[Kurke, 1961] Kurke, M. I. (1961). Operational sequence diagrams in system design. *Human Factors*, 3:66–73.

[Loer, 2003] Loer, K. (2003). *Model-based Automated Analysis for Dependable Interactive Systems*. PhD thesis, Department of Computer Science, University of York, UK.

[Loer and Harrison, 2003] Loer, K. and Harrison, M. (2003). Model-Based Formal Analysis of Temporal Aspects in Human-Computer Interaction. In *Proceedings of the HCI2003 Workshop on the Temporal Aspects of Tasks*, Bath, UK.

[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. D., 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.

[Loer and Harrison, 2002] Loer, K. and Harrison, M. D. (2002). Towards usable and relevant model checking techniques for the analysis of dependable interactive systems. In Emmerich, W. and Wile, D., editors, *Proceedings 17th International Conference on Automated Software Engineering*, pages 223–226. IEEE Computer Society.

[Mikk et al., 1997] Mikk, E., Lakhnech, Y., Petersohn, C., and Siegel, M. (1997). On formal semantics of statecharts as supported by STATEMATE. In *Second BCS-FACS Northern Formal Methods Workshop*, Ilkley, UK. 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.

[Pocock et al., 2001] Pocock, S., Harrison, M. D., Wright, P., and Johnson, P. (2001). THEA: A Technique for Human Error Assessment Early in Design. In Hirose, M., editor, *Human-Computer Interaction – INTERACT '01*. IOS Press/IFIP.

[Potts et al., 1994] Potts, C., Takahashi, K., and Anton, A. (1994). Inquiry-Based Scenario Analysis of System Requirements. Technical Report GIT-CC-94/14, College of Computing, Georgia Institute of Technology.

[Reason, 1990] Reason, J. (1990). *Human Error*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU.

[Rumbaugh et al., 1998] Rumbaugh, J., Booch, G., and Jacobson, I. (1998). *The Unified Modeling Language Reference Manual (UML)*. Object Technology Series. Addison-Wesley.

[Rushby, 2002] Rushby, J. (2002). Using Model Checking to Help Discover Mode Confusions and Other Automation Surprises. *Reliability Engineering and System Safety*, 75(2):167–177. Available at `http://www.csl.sri.com/users/rushby/abstracts/ress02`.

[Smith and Mosier, 1986] Smith, S. L. and Mosier, J. N. (1986). Guidelines for designing user interface software. Technical Report ESD-TR-86-278, The MITRE Corporation, Bedford, MA. `ftp://ftp.cis.ohio-state.edu/pub/hci/Guidelines/`.