A framework and supporting tool for the model-based analysis for dependable interactive systems in the context of industrial design

K. Loer and M. Harrison

Technical Report Series

CS-TR-873

November 2004

A Framework and Supporting Tool for the Model-based Analysis for Dependable Interactive Systems in the Context of Industrial Design

Karsten Loer[1] and Michael Harrison[2]
[1]Department of Computer Science, University of York, York, YO10 5DD, UK
[2]Informatics Research Institute, University of Newcastle Upon Tyne,UK
{Karsten.Loer, Michael.Harrison}@cs.york.ac.uk

25th November 2004

Abstract

This paper describes a tool to support model checking of interactive systems within an industrial environment. The paper introduces the proposed design framework and the tool that is designed to bridge between those familiar with company practice and the less familiar approach based on model checking. The tool was developed to be used in an avionics design setting. It is currently at a prototype stage. The paper describes the experience of a small sample of designers and the design implications arising from their comments.

1 Introduction

An important concern during the design of interactive systems for safety-critical applications is to identify and, where possible, to eliminate potential design flaws that might be exhibited in system operation. It is not unusual in current design practice that system engineers subject the current state of their design to walk-throughs (for example the cognitive walk-through method [Wharton et al., 1994] in usability engineering) to explore and evaluate different applications of the system under design. Such techniques are generally performed on selected instances of system use and so there is a danger that potentially hazardous situations are ignored or overlooked.

Model checking [Clarke et al., 1999] – an analysis technique that is based on exhaustive state exploration of system models – can help to address analysis coverage. In order to achieve integration of model checking techniques, [Loer and Harrison, 2002] explored the characteristics of analysis techniques that are applied by design stakeholders with different areas of expertise during the development of interactive systems.

The purpose of this paper is to describe an integrated tool for the formal analysis of models of interactive systems (Section 4). The tool was implemented for a particular industrial design setting (described in Section 3). The application of the tool is illustrated by a brief case study in Section 5. The paper reports a user-based evaluation of the tool with developers from the targeted design setting (Section 6) and a discussion of related work (Section 7) which yields a list of suggestions to be addressed in future work (Section 8).

1.1 Characteristics and stakeholders in interactive systems design

Dependable interactive systems design has multi-disciplinary characteristics. Indeed [Preece et al., 1994] asserts that as many as eleven disciplines contribute to the process, ranging from anthropology, cognitive psychology and philosophy to computer science, engineering and design. In practice few groups get involved in the design process and two in particular can be identified, "*system engineers*" and "*usability engineers*". These communities focus on different aspects of the design. Usability engineers aim to produce solutions for:

1. understanding the work of the interactive system and the role that the designed artefact (for brevity called "*device*" hereafter) plays within this work,
2. understanding the role of the device in the context of work,
3. understanding the role of the user of the device and
4. understanding the limitations of the user and the device.

Dependable system engineers share some concerns of 1 and 2 but also deal with more system-centered concerns such as:

5. verification and validation of complex systems and
6. certification of systems.

A wealth of methods have been developed to integrate these concerns. For example, scenario-based design [Carroll, 1995] is widely used in human-computer interaction (HCI) to deal with items 1-3. Formal methods[1] such as model checking [Clarke et al., 1999] are increasingly used for verification (Item 5). Verification may be required for certification purposes (Item 6), to demonstrate that the necessary steps were taken to comply with relevant standards such as [MoD, 1996] and [IEC, 1999].

System engineers and usability engineers share analysis goals at a top level (checking the compliance of the system with respect to a set of requirements), but their perspectives and practices are different. Consequently, usability analysis and system analysis are often performed separately, with a negative effect on design quality. The goal therefore is to provide both groups of designers with a common analysis technique and with a means of exchanging analysis results early in design.

Belotti *et al.* show that analytic HCI methods can inform system design [Belotti et al., 1995]. Similarly formal modelling and analysis techniques have been used to analyse some modest HCI-related properties of systems, see [Abowd et al., 1995], [d'Ausbourg, 1998], [Butler et al., 1998], [Campos and Harrison, 2001], [Paternò, 1996], and [Rushby, 2002].

1.2 Challenges for the integration of techniques

An analysis technique that supports both groups of design stakeholders requires the integration of formal and informal analysis techniques. Model checking [Clarke et al., 1999] has merit in analysing some aspects of interactive system design because, once provided with appropriate input, the analysis is performed automatically and does not require further involvement. The technique was originally developed for hardware and protocol verification. The model (in this case of the device) is described as a state transition system and requirements are checked by means of state exploration. If no violating state is found the property holds, otherwise a trace is produced that illustrates why a given property does not hold in a particular state. The technique can be used to discover latent errors [Reason, 1990] in interactive systems. Model checking has been used to analyse mode confusion errors in flight system automation [Rushby, 2002].

[1]In this work the attribute "*formal*" is assigned to methods that provide (i) a means of modelling, (ii) a (usually mathematical) notation and (iii) a verification method. The attribute "*informal*" is assigned to methods that describe systematic procedures which do not have a mathematical foundation.

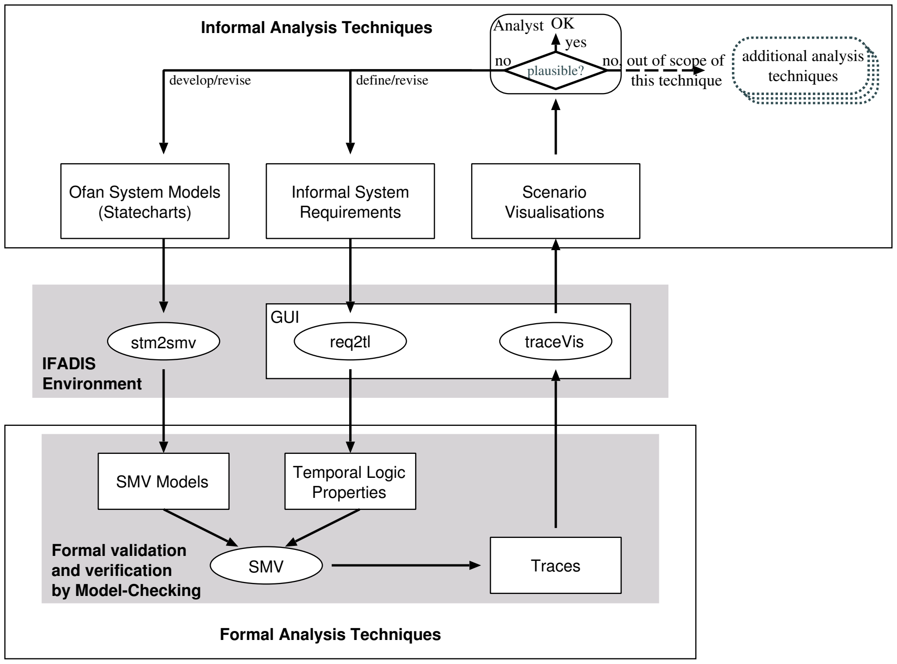

Informal Analysis Techniques
Analyst
OK
yes
no
plausible?
no
out of scope of this technique
additional analysis techniques
develop/revise
define/revise
Ofan System Models (Statecharts)
Informal System Requirements
Scenario Visualisations
GUI
stm2smv
req2tl
traceVis
IFADIS Environment
SMV Models
Temporal Logic Properties
Formal validation and verification by Model-Checking
SMV
Traces
Formal Analysis Techniques

Figure 1: An implementation of the IFADIS framework.

There are some similarities between the systematic analysis of properties in model-checking and guideline-based usability inspection methods [Nielsen, 1992] that are used by usability engineers as is discussed in [Loer and Harrison, 2001]. In principle, model checking has much to offer interactive system designers. 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).

The technical requirements for an integration of this technique into a design process were identified in [Loer and Harrison, 2002]. Three technical tasks must be addressed: (i) obtaining a model that represents relevant aspects of the interactive system (the device, the environment and user), (ii) supporting the formulation of property specifications representing the desired requirement, (iii) visualising the information provided by model-checking traces.

These are the core tasks that are considered by the IFADIS framework and tool introduced in the following sections.

2 Getting the best of both worlds: The IFADIS framework

The Integrated Framework for the Analysis of Dependable Interactive Systems (IFADIS) is designed to:

- support system engineers in applying model-checking tools, and
- make it possible for usability engineers to apply model-checking tools in usability engineering.

To do this it is necessary to identify representations that are shared between the usability engineers' and system engineers' worlds and can be mapped onto the inputs and outputs that are required and provided by the supported model-checking tools. The work was based on two well-established tools, NuSMV [Cimatti et al., 2002] and Cadence SMV [Cadence Berkeley Laboratories, 2000]. These tools appeared to be robust and well supported and preliminary work on the mapping from Statecharts to the SMV language was promising.

The IFADIS framework addresses the three challenges discussed in Section 1.2 for industrial exploitation as follows.

Task 1: Obtaining models

Using formal methods to support the analysis and design of interactive systems is only valuable if these methods can be integrated into existing practice. Hence additional special-purpose models should be avoided. General purpose models that already exist in the practice of dependable systems design are therefore desirable. Statecharts [Harel, 1987] have been used in the motor and aircraft industries. They are relatively widely used for modelling system behaviour and have also been used in the design of interactive systems [Horrocks, 1999]. Extensions such as the OFAN approach structure statecharts [Degani, 1996] to make these models equally amenable to dependability and usability analysis. The structure of OFAN models involves decomposing the system specification into separate specifications describing the behaviour of the device, the user interface, the user and the environment.

Task 2: Obtaining property specifications

System requirements are usually described, at least initially, in natural language. Model checking tools expect requirements specifications in terms of temporal logic formulae over state automata. A number of approaches including [Abowd et al., 1995], [d'Ausbourg, 1998], [Campos and Harrison, 2001], [Paternò, 1996] and [Loer, 2003] have developed usability requirements formally. Furthermore, [Loer and Harrison, 2001] describes how usability heuristics may be be translated into formal property specifications. While these activities illustrate technical feasibility they do not demonstrate that the techniques used are acceptable for use by the target audience of usability and system engineers.

Task 3: Visualisation of analysis results

Depending on how a property is formulated (for example, whether it is existentially or universally quantified), the model checker can produce a trace that either demonstrates why a property holds, or why it is violated. In the case of the SMV model checker, these traces are provided in the form of text or tables. To be useful, the results of the analysis need to be visualised in a way that the designer can use. Traces can form the backbone for scenarios that may be analysed by requirements engineers. They can be used to illustrate and communicate the implications of design decisions to and between design teams.

The remainder of this paper describes a tool that addresses these tasks. Any method will be suitable for particular kinds of analyses and less suitable for others [Fields et al., 1997]. Hence, the tool should integrate with other methods that are more suitable for analyses that it cannot handle itself.

3 An industrial design environment

The project team who were the target community for the tool, and the method supported by it, consisted of engineers with differing backgrounds and expertise in the areas of safety critical systems engineering and user interface design. The relevant aspects of this design setting can be summarised as follows:

1. In the early stages, natural language requirements documents were elaborated in collaboration with the customers of the aircraft under design.

2. From these documents, statechart models of selected systems were developed by systems engineers, using the iLogix STATEMATE[2] toolkit [Harel et al., 1990]. All team members had basic knowledge of how to read graphical state machine descriptions, but only the systems engineers were qualified to produce such model specifications.

3. The group of engineers that was responsible for the usability analysis investigated the requirements document and – as soon as they were available – statechart models with respect to a list of usability guidelines.

4. The systems engineers performed a static analysis of the model (for instance, type and completeness checks). STATEMATE's simulation tool was then used to animate the model and to explore its behaviour through a set of tests.

Both the usability analysis and the simulation/test were based on a set of mission scenarios that were chosen as most relevant through consultation with the customers. In this early phase of the design the analysis steps were performed systematically, but manually, and not exhaustively.

4 The IFADIS tool

The three tasks of the IFADIS framework mentioned in the previous section, namely model compilation, property formulation and trace visualisation, are implemented in separate tools that are integrated within the IFADIS tool.

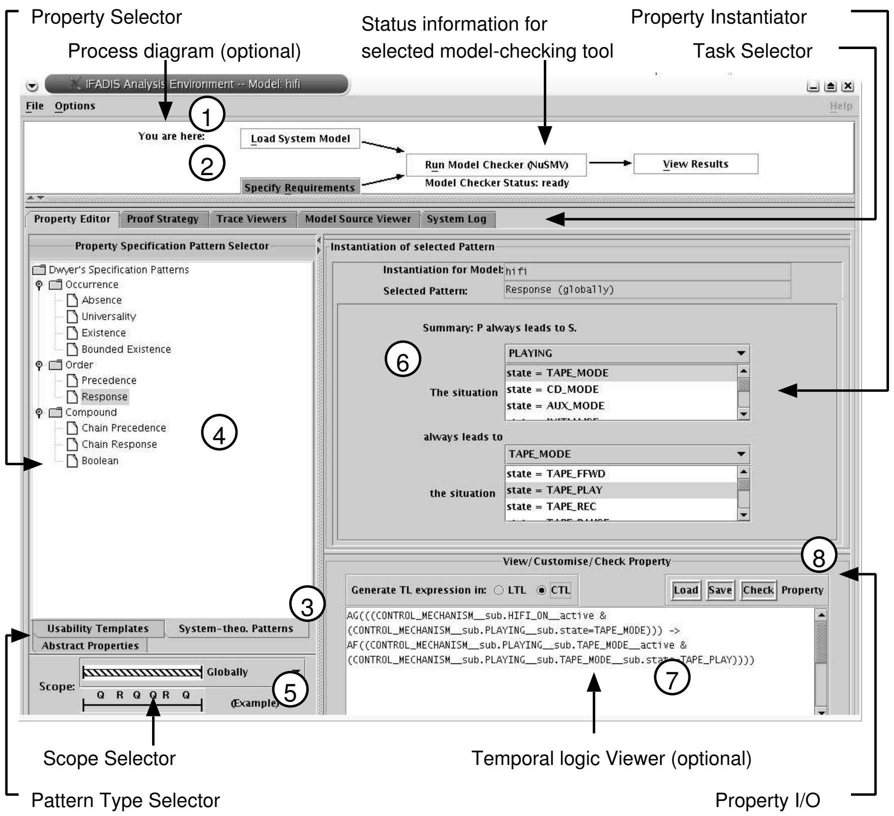
4.1 Model import from STATEMATE (`stm2smv`)

IFADIS imports a STATEMATE model (Step ① in Figure 2) by invisibly calling the `stm2smv` compiler that translates statechart models to SMV models. This compiler implements an extended version of the algorithm by [Clarke and Heinle, 2000], see [Loer, 2003, Chapter 4] for more details. The new compiler was necessary because existing tools are either (i) not automatic [Day, 1993], (ii) require additional translations via intermediate notations [Mikk et al., 1998], [Burton, 2002], (iii) are not affordable in the context of this work [Bienmüller et al., 2000], or (iv) were developed for different dialects of the statechart language [Chan et al., 1998], [Latella et al., 1999], [Lilius and Porres Paltor, 1999], [Canver, 1999].

4.2 Temporal logic property editor (`req2tl`)

In statecharts, requirements are expressible formally as relationships between (sequences of) system states and events. SMV uses linear temporal logic (LTL, [Manna and Pnueli, 1992]) or computational tree logic (CTL, [Clarke and Sistla, 1986]) as its language for specifying requirements. Dwyer and his co-authors extracted 555 property specifications from a range of sources and observed that most properties are instantiations of a limited set of patterns under a particular scope ([Dwyer et al., 1999], see Figure 3). [Loer, 2003, Chapter 5] demonstrates that usability guidelines, such as those in [Dix et al., 1998, Chapter 4], [Nielsen, 1992], or [Smith and Mosier, 1986] can be viewed similarly as patterns. The IFADIS property editor (`req2tl`) helps the designer to construct temporal logic specifications by making the patterns available and aiding the process of instantiation. To support the user in selecting the appropriate pattern and scope, a natural language summary of the semantics is provided (for example, "A situation P always leads to a situation S."). An

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



Instantiation for Model: hifi

Selected Pattern: Response (globally)

Summary: P always leads to S.

The situation: PLAYING — state = TAPE_MODE, state = CD_MODE, state = AUX_MODE

always leads to

the situation: TAPE_MODE — state = TAPE_FFWD, state = TAPE_PLAY, state = TAPE_REC

Generate TL expression in: ○ LTL ● CTL

```
AG(((CONTROL_MECHANISM__sub.HIFI_ON__active &
(CONTROL_MECHANISM__sub.PLAYING__sub.state=TAPE_MODE))) ->
AF((CONTROL_MECHANISM__sub.PLAYING__sub.TAPE_MODE__active &
(CONTROL_MECHANISM__sub.PLAYING__sub.TAPE_MODE__sub.state=TAPE_PLAY))))
```

Figure 2: The user interface of the IFADIS property editor (`req2tl`).

assumption was made that the designer would learn the list of keywords for patterns and it would not be necessary to support natural language equivalents that could be interpreted as temporal logic expressions as suggested by [Flake et al., 2000] and [Dennis et al., 2000]. Templates can be browsed from two viewpoints: "usability" and "system-engineering". In the system-engineering view, templates are arranged following Dwyer's hierarchy (see Figure 3). In the usability view, templates are listed by keywords that are commonly used in the usability literature (see Figure 4). Templates contain predicates (termed "situations" in the GUI) that the analyst must instantiate with appropriate elements of the model by selecting from a structured menu reflecting the hierarchy of the system model (see Figure 5). Hence in Step ② in Figure 2 the user selects a usability or system engineering view and selects a property (Step ④), picks a scope (Step ⑤) if required, and instantiates the resulting template (Step ⑥). The tool then generates the corresponding temporal logic property and the result is shown in the temporal logic pane (Step ⑦) if required although it is assumed that it will usually be invisible to the user. The tool automatically deals with the auxiliary variables that were introduced by the stm2smv compiler, see [Loer, 2003, Chapter 4] for details. Once a property has been selected and instantiated, it can be stored and passed on to the model-checker (Step ⑧).

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 ③ the "Abstract Properties" view offers generic options that automatically generate sets of basic properties by traversing the system model hierarchy.

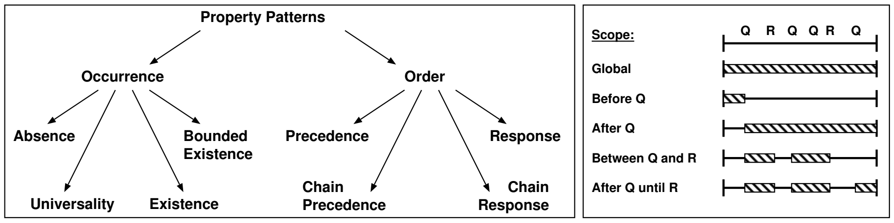


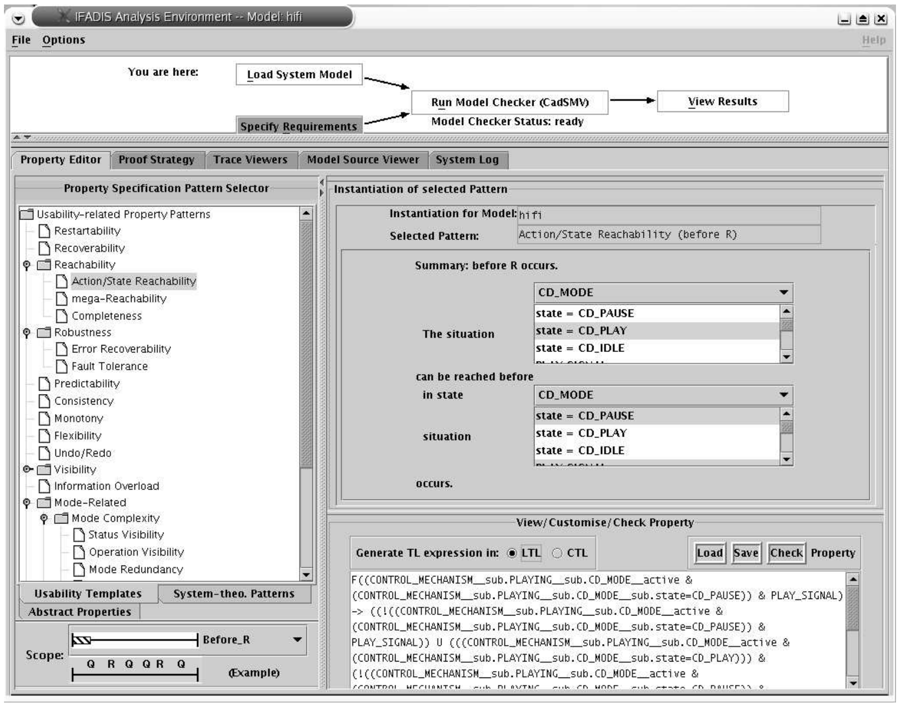
Figure 3: Property specification patterns (Dwyer et al., 1999).

The proof strategy editor for the analysis under assumptions:

At some stages of the development process complete models may not be available. Indeed, in the design of interactive systems, specifications about some behaviour that is outside the control of the interactive device (for instance, user inputs or environmental inputs) will *hardly ever* be available[3]. Consequently, the system models remain *open* with respect to the behaviour of variables that are controlled outside the specified device. In order to limit the search space, assumptions need to be made about the behaviour of the open variables (for instance: "The user of the HiFi system will not attempt to switch the system `ON`, if it is `ON` already."). Such assumptions can be specified in the same manner as the requirements specifications (see Section 4.2). In Cadence SMV 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 6, 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

[3]Unlike syndetic modelling approaches, such as [Duke et al., 1998], we do not attempt to produce detailed user models here. See Section 5 for details.



Model Checker Status: ready

Instantiation for Model: hifi
Selected Pattern: Action/State Reachability (before R)

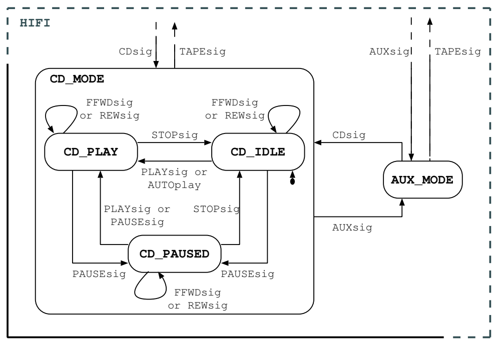
Summary: before R occurs.

The situation CD_MODE: state = CD_PAUSE / state = CD_PLAY / state = CD_IDLE
can be reached before
in state CD_MODE
situation: state = CD_PAUSE / state = CD_PLAY / state = CD_IDLE
occurs.

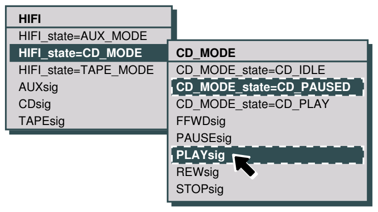
Generate TL expression in: [✓ LTL] CTL

```
F((CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__active &
(CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__sub.state=CD_PAUSE)) & PLAY_SIGNAL)
-> ((!((CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__active &
(CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__sub.state=CD_PAUSE)) &
PLAY_SIGNAL)) U (((CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__active &
(CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__sub.state=CD_PLAY))) &
(!((CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__active &
```

Figure 4: Usability-perspective of property editor (`req2tl`).



a) Excerpt from statechart Hi-Fi model.



b) Corresponding instantiation menu.

Figure 5: Example instantiation of a predicate with expression: "a `PLAY` signal is sent when the CD unit is in `PAUSED` state" — "`CD_MODE_state=CD_PAUSED & PLAYsig`".

(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).

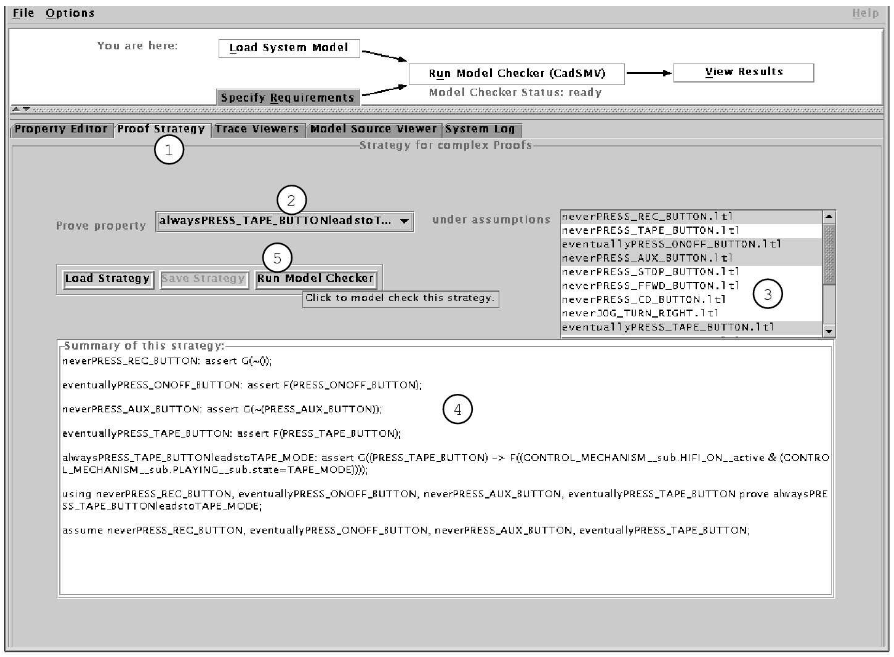
4.3 Trace visualisation (`traceVis`)

Once a system model is provided and a property is specified, the model checker can be run. Depending on the type of property, the model checker can return traces that demonstrate why a property holds (in case of an existentially quantified property) or does not hold (in case of a universally quantified property). Model-checking traces are probably the most useful result delivered by a model checker, demonstrating *why* a property holds (or fails) as opposed to merely *that* it holds (or fails). Traces can form the backbone for *scenarios* [Carroll, 1995] capturing instances 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 and a realistic context must be visualised in which that sequence could occur. A range of notations capture system behaviour and interactions (see Section 4.4). Which notation provides the most appropriate visualisation depends on the individual designer's background.

4.4 Candidate trace representations for industrial work practice

In a field study with aerospace engineers and academics, both groups having some human factors background, a range of visualisations (in addition to the tabular view that is already provided by the SMV tools) were assessed with respect to the following desirable properties:

1. **Activity of human agents:** To predict workload of future system users what *kind* of interaction is an agent engaged, and what is its *frequency*.
2. Information about the **frequency of use of devices** is required, for instance, for planning response time of hardware or the physical layout of the workspace. Furthermore, if interactions are data-driven the notation should give an impression of what kind and amounts of **data** are required, and at what time.
3. **Causal dependencies between activities:** To support the analyst in drawing conclusions it is not only necessary to indicate *that* something happened, but it also needs to be clear *why* it happened.



```
neverPRESS_REC_BUTTON.ltl
neverPRESS_TAPE_BUTTON.ltl
eventuallyPRESS_ONOFF_BUTTON.ltl
neverPRESS_AUX_BUTTON.ltl
neverPRESS_STOP_BUTTON.ltl
neverPRESS_FFWD_BUTTON.ltl
neverPRESS_CD_BUTTON.ltl
neverJOG_TURN_RIGHT.ltl
eventuallyPRESS_TAPE_BUTTON.ltl
```

Summary of this strategy:

```
neverPRESS_REC_BUTTON: assert G(~());

eventuallyPRESS_ONOFF_BUTTON: assert F(PRESS_ONOFF_BUTTON);

neverPRESS_AUX_BUTTON: assert G(~(PRESS_AUX_BUTTON));

eventuallyPRESS_TAPE_BUTTON: assert F(PRESS_TAPE_BUTTON);

alwaysPRESS_TAPE_BUTTONleadstoTAPE_MODE: assert G((PRESS_TAPE_BUTTON) -> F((CONTROL_MECHANISM__sub.HIFI_ON__active & (CONTRO
L_MECHANISM__sub.PLAYING__sub.state=TAPE_MODE))));

using neverPRESS_REC_BUTTON, eventuallyPRESS_ONOFF_BUTTON, neverPRESS_AUX_BUTTON, eventuallyPRESS_TAPE_BUTTON prove alwaysPRE
SS_TAPE_BUTTONleadstoTAPE_MODE;

assume neverPRESS_REC_BUTTON, eventuallyPRESS_ONOFF_BUTTON, neverPRESS_AUX_BUTTON, eventuallyPRESS_TAPE_BUTTON;
```

Figure 6: Screen-shot of proof strategy pane.

4. **Scalability:** Traces grow by the number of variables (= the size of the model) and by the number of states (= the length of the trace). If possible, a trace visualisation should support a sufficient abstraction, so that all relevant information can be displayed.

Trace visualisations used by existing SMV tools are limited in relation to these properties (for illustration, see the NuSMV raw format of a sample trace in Figure 7). In the remainder of this section a number of alternative notations are described, including variable tables, sequence diagrams, operational sequence diagrams, statecharts and model animation. These notations have a similar role in the design of dependable interactive systems. Note, that the primary concern here is the generation of scenario *visualisations* and not the generation of scenario *views*. The former presents the trace information in a scenario notation, whereas the latter provides perspectives on a scenario from the different stakeholders' points of view. There can be an overlap, however, since some notations are only used by certain stakeholders.

4.4.1 Variable tables

The only trace visualisations provided by existing SMV tools are tables that present the values of variables against each execution step (see Figure 8). Such tables are suitable for a mechanical analysis, but hard to read by human analysts. A problem that was mentioned by the field study participants is that for each step a lot of data is presented, even though relatively few changes typically occur in one state transition. Consequently, it is hard to identify the changes between two steps, making it difficult to analyse the activity of components and the frequency of use of devices. The readability of the table can be improved by highlighting those cells of the table that change between steps. In particular this enhancement gives an indication of the activity of agents and the frequency of use of devices. Such an improved table visualisation is implemented in the `traceVis` tool of the IFADIS environment (see Figure 9). However, this tabular view does not make causal dependencies between variable changes explicit. Table visualisations scale better than the graphical visu-

```
State 1.1:                                State 1.5:
USER.PILOT1.turn_ALT_Knob = 0             USER.PILOT2.read_FMS_DISPLAY = 0
USER.PILOT1.pull_ALT_Knob = 0             USER.PILOT1.pull_ALT_Knob = 1
DISPLAYS.FMS_DISPLAY.state = NIL          DISPLAYS.FMS_DISPLAY.state = CLMB
USER.PILOT1.read_FMS_DISPLAY = 0
USER.PILOT2.read_FMS_DISPLAY = 0          State 1.6:
CONTROL_MECHANISM.CAP_MODE.state = OFF    USER.PILOT1.pull_ALT_Knob = 0
                                          USER.PILOT1.read_FMS_DISPLAY = 1
State 1.2:                                USER.PILOT2.read_FMS_DISPLAY = 1
USER.PILOT1.turn_ALT_Knob = 1
                                          State 1.7:
State 1.3:                                USER.PILOT1.read_FMS_DISPLAY = 0
USER.PILOT1.turn_ALT_Knob = 0             USER.PILOT2.read_FMS_DISPLAY = 0
DISPLAYS.FMS_DISPLAY.state = ALT          CONTROL_MECHANISM.CAP_MODE.state = CAP

State 1.4:                                State 1.8:
USER.PILOT2.read_FMS_DISPLAY = 1          DISPLAYS.FMS_DISPLAY.state = NIL
```

Figure 7: Excerpt from sample SMV model-checking trace.

Value of variable / ... in step No.	1	2	3	4	5	6	7	8
...PILOT1.turn_ALT_Knob	0	1	0	0	0	0	0	0
...PILOT1.pull_ALT_Knob	0	0	0	0	1	0	0	0
...FMS_DISP.state	NIL	NIL	ALT	ALT	CLMB	CLMB	CLMB	NIL
...PILOT1.read_FMS_DISPL...	0	0	0	0	0	1	0	0
...PILOT2.read_FMS_DISPL...	0	0	0	1	0	1	0	0
...CAP_MODE.state	OFF	OFF	OFF	OFF	OFF	OFF	CAP	CAP

Figure 8: Visualisation of the trace in Figure 7 as table of variables.

alisations introduced in the following Sections. In [Kermelis, 2003], a further improvement of this visualisation is presented which implements features including the hiding of steps and variables and the comparison of traces. These features were developed as a result of feedback from the tool evaluation.

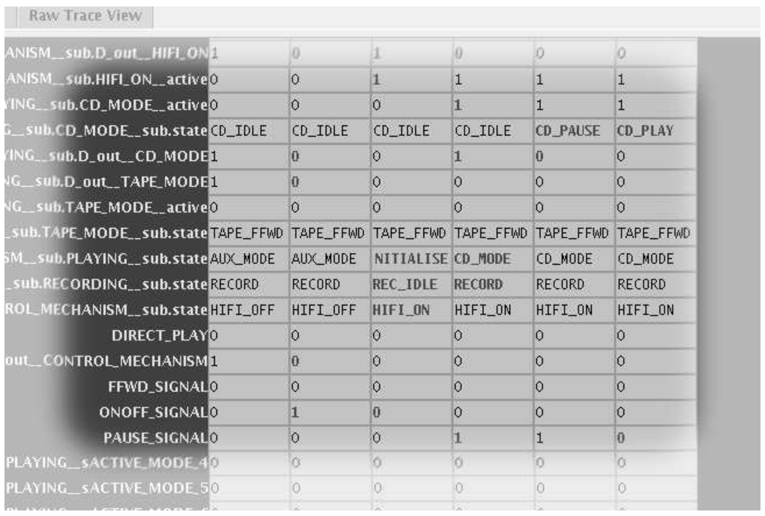
4.4.2 Sequence Diagrams

In Sequence Diagrams, as defined in the Unified Modelling Language (UML) [Rumbaugh et al., 1998], agents are denoted as boxes and have a vertical life-line attached to represent its activity over time. Interaction between agents is shown by message transfers. The number of messages sent at a time gives an indication of the workload of an agent and the frequency of use of a device. Messages can also be guarded by conditions (denoted by brackets in Figure 10). As indicated in Figure 10, an object can also send a message to itself (self-call). The scalability of sequence diagrams, as of most graphical notations, is limited.

Sequence diagrams are used already for model simulation and counterexample visualisation by some model-checking tools, for example, SPIN [Holzmann, 1997], MOCHA [Alur et al., 1998] and the most recent versions of UPPAAL2k [Amnell et al., 2001].

4.4.3 Operational Sequence Diagrams

Operational Sequence Diagrams (OSDs) have been used for behavioural analysis of human-computer interaction [Kurke, 1961]. OSDs are used in aerospace and automobile design for task representation and interaction analysis. They are similar to sequence diagrams in that they show agents as vertical columns to represent the progression of time. Interaction between agents is denoted as horizontal flow of information. Several types of information flow, operations (manual/automatic) and flow manipulation (operate, inspect, store, transmit, decide, receipt) can be specified, and OSDs can display data elements. For example, in Figure 11 in step 1.2 the `PILOT1` turns the `ALT` knob, changing the physical state of the knob. This state change, in turn leads to the transmission of an electronic



Raw Trace View

ANISM__sub.D_out__HIFI_ON	1	0	1	0	0	0
ANISM__sub.HIFI_ON__active	0	0	1	1	1	1
YING__sub.CD_MODE__active	0	0	0	1	1	1
G__sub.CD_MODE__sub.state	CD_IDLE	CD_IDLE	CD_IDLE	CD_IDLE	CD_PAUSE	CD_PLAY
YING__sub.D_out__CD_MODE	1	0	0	1	0	0
NG__sub.D_out__TAPE_MODE	1	0	0	0	0	0
NG__sub.TAPE_MODE__active	0	0	0	0	0	0
_sub.TAPE_MODE__sub.state	TAPE_FFWD	TAPE_FFWD	TAPE_FFWD	TAPE_FFWD	TAPE_FFWD	TAPE_FFWD
SM__sub.PLAYING__sub.state	AUX_MODE	AUX_MODE	NITIALISE	CD_MODE	CD_MODE	CD_MODE
_sub.RECORDING__sub.state	RECORD	RECORD	REC_IDLE	RECORD	RECORD	RECORD
ROL_MECHANISM__sub.state	HIFI_OFF	HIFI_OFF	HIFI_ON	HIFI_ON	HIFI_ON	HIFI_ON
DIRECT_PLAY	0	0	0	0	0	0
out__CONTROL_MECHANISM	1	0	0	0	0	0
FFWD_SIGNAL	0	0	0	0	0	0
ONOFF_SIGNAL	0	1	0	0	0	0
PAUSE_SIGNAL	0	0	0	1	1	0
PLAYING__sACTIVE_MODE_4	0	0	0	0	0	0
PLAYING__sACTIVE_MODE_5	0	0	0	0	0	0

Figure 9: Table visualisation of a trace (enhanced by highlighting of changes).

ALT_CHANGED_SIG signal to some external receiver. This detailed classification of operations makes the OSD notation richer than message sequence charts. However, the assignment of appropriate types to flows, operations and manipulations requires knowledge that is not contained in a trace. The production of an OSD from a trace will therefore require an interaction with the analyst. Again, scalability is a critical issue in this notation.

4.4.4 Statecharts

Statecharts are an obvious candidate notation that is already familiar to the user of the tool. [Glinz, 1995] demonstrates how (a syntactically enhanced version of) statecharts can be utilised to model scenarios. The notation and its semantics are a subset of statecharts (histories and overlapping states are omitted). By means of the standard statechart operations it is possible to compose scenarios to form richer pictures. The level of abstraction is higher than in the statechart model, so a direct mapping is hard. In this enhanced statechart no-

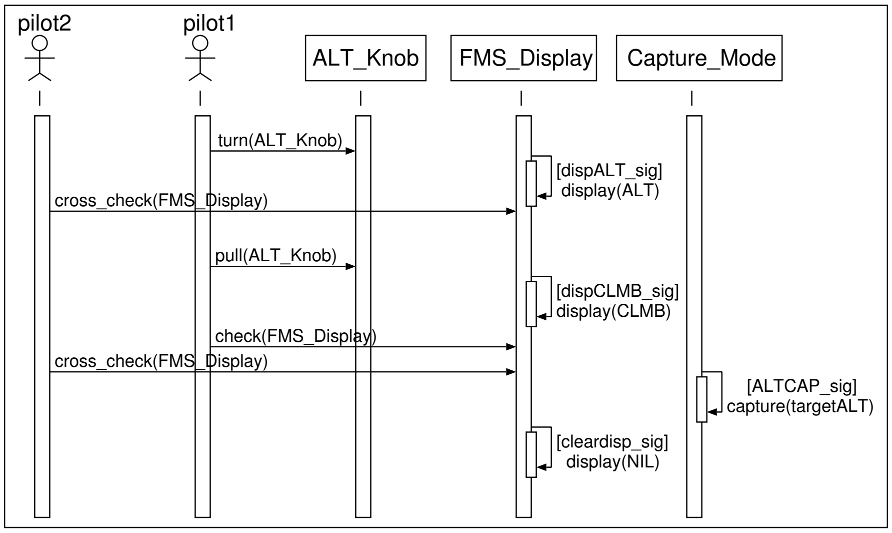


Figure 10: Visualisation of the trace in Figure 7 as sequence diagram.

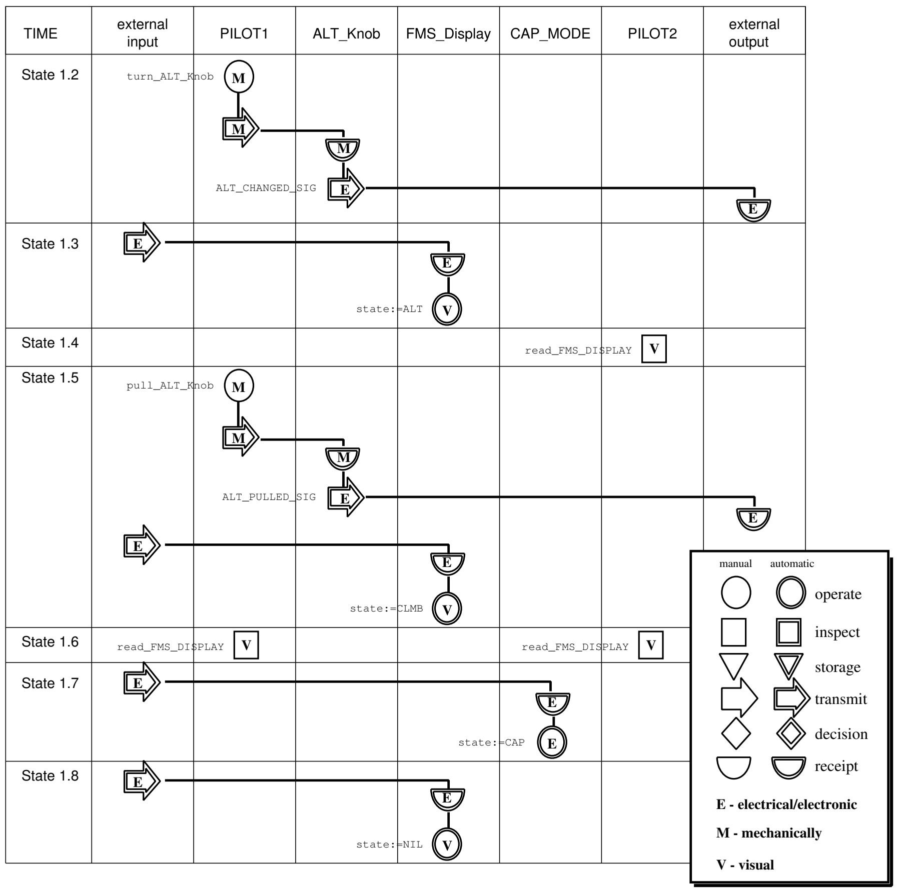


Figure 11: Visualisation of the trace in Figure 7 as an Operational Sequence Diagram.

tation causality is as hard/easy to detect as in STATEMATE statechart models. Activity is indicated by the number of steps within a scenario, but workload is hard to analyse, since time is difficult to model. Scalability is also limited.

4.4.5 Trace-driven model animation

Both UPPAAL2k [Amnell et al., 2001] and the STATEMATE tool [Harel et al., 1990] have animation features that makes it possible to "step" through a trace. Changes can be viewed in the statechart model, and states and transitions that are currently active are highlighted.

The simulation profiles that are necessary to run the animation can be created from the model-checking traces, as is demonstrated in [Mikk et al., 1998]. This is not a straightforward step though, since the simulation profile requires information on the timing of the events which is not available in SMV traces. Mikk's solution to this problem requires the engineer to step through the trace once and set the timing manually.

It is debatable how useful the animation of statechart models is in this context. They have the advantage that the OFAN statechart model will be the description of the model used by the analyst rendering the animations more familiar. However the scalability of animation is limited. In industrial scale models consisting of several Statecharts, spread across several pages, state changes will be difficult to follow in the course of the animation. In the case of long traces it is often hard to memorise the history of states and variable values and therefore to assess a current state/transition properly. This was certainly our experience working with the UPPAAL2k model checker [Amnell et al., 2001], where traces

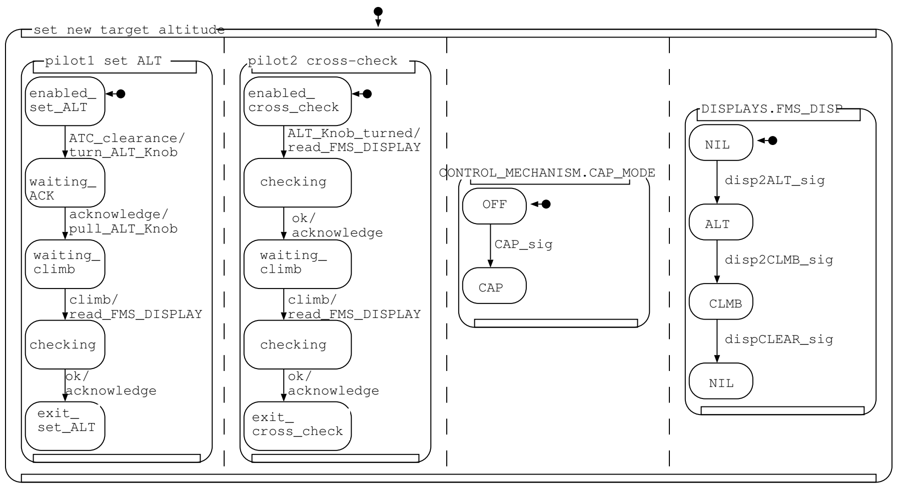


Figure 12: Visualisation of the trace in Figure 7 as a scenario statechart.

Step	Agent	Action
1	pilot1	set targetALT
2	FMS	display “ALT”
3	pilot2	check FMS display
4	pilot1 FMS	pull ALT knob display “CLMB”
5	pilot1 pilot2	check FMS display check FMS display
6	FMS	capture target ALT
7	FMS	display “NIL”

Figure 13: Scenario script describing the situation in the trace in Figure 7.

are visualised as animations of the system model[4].

4.4.6 Scenario templates and scenario scripts

Natural language scenarios might also be considered as an alternative to these formal representations. They are appealing because they do not require the analyst to learn new notations and they include narrative information about context. In the case of the Technique for Human Error Assessment (THEA) [Pocock et al., 2001] structured natural language scenarios are used to focus the analysis process. Here scenarios are described at a more abstract level than the model-checking trace, thereby improving scalability. The process of generating a natural language scenario as a visualisation of the trace itself is complicated. Parts of it can be mechanised but most of the useful information must be supplied using the rich experience of a domain expert.

Potts et al. use scenario scripts for requirements elicitation and validation [Potts et al., 1994]. These scripts display the interaction in a scenario as tables of actions along with the agents that perform these actions. General scenario scripts cannot be derived from traces and statechart models. Scenario scripts can be regarded as a more abstract and “filtered” view that displays only the changes (= *actions*) that occur between steps (see Figure 13). Furthermore, the association of actions with agents gives insight into causal dependencies. As in variable tables, the activity of the agents that are involved can be detected from scenario

[4]The UPPAAL developers have most recently (from tool version 3.4) addressed this problem by providing a second view of the trace as a sequence diagram.

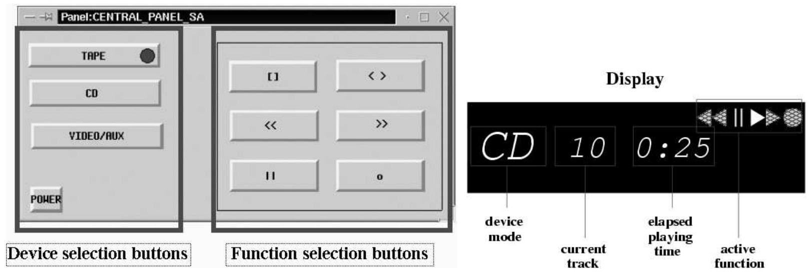


Figure 14: Input panel and display of HiFi system.

scripts. Scenario scripts are less abstract than THEA templates, but they focus on tasks and provide only limited contextual information.

5 Case study: Heuristic evaluation of a Hi-Fi system

In [Loer and Harrison, 2001] a commercially available Hi-Fi system (Figure 14) is used as an example of how a design can be analysed with respect to a number of usability properties. Although the focus in this paper is on the evaluation of the tool, this case study is used to illustrate briefly how the IFADIS tool is used to analyse usability requirements of an existing design. For instance, the specification of the CD device – see the specification excerpt in Figure 5a – shall be analysed for conformance with the consistency requirement:

> "*Users should not have to wonder whether different words, situations, or actions mean the same thing.*" [Nielsen, 1992]

This (non-functional) property was analysed by checking a set of keystroke-level functional properties. For example, demonstrating that each input element yields the output that the designer expected. The following template[5] was repeatedly instantiated by terms describing input and expected output predicates ($\langle input\_config \rangle_i$, $\langle user\_input \rangle$ and $\langle effect\_config \rangle_j$, respectively):

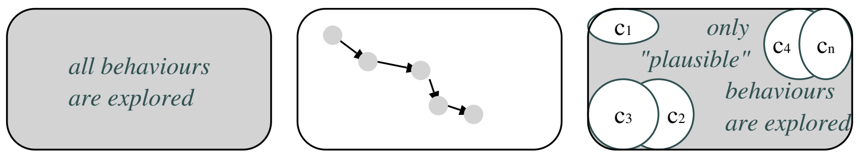
Property:	Behavioural consistency (*global* scope)
description:	Some input $\langle user\_input \rangle$ in similar configurations $\langle input\_config \rangle_i$ yields the same effect $\langle effect\_config \rangle_j$.
templates:	
CTL:	$\bigwedge_{j=1\ldots l} \bigwedge_{i=1\ldots k}$ `AG((`$\langle input\_config \rangle_i \wedge \langle user\_input \rangle$`)` `->AF` $\langle effect\_config \rangle_j$`)`
LTL:	$\bigwedge_{j=1\ldots l} \bigwedge_{i=1\ldots k}$ `G((`$\langle input\_config \rangle_i \wedge \langle user\_input \rangle$`)` `-> F` $\langle effect\_config \rangle_j$`)`
derived from:	*Dwyer et al.'s* response pattern ("global" scope), where: $l = 1$, if the response effect has to be literally the same, and $l > 1$, if a number of effects are regarded to be similar.

This analysis revealed the potential inconsistency that pressing the `PAUSE` button repeatedly while the system is in `CD_IDLE` mode can invoke the `CD_PLAY` mode (see trace in Figure 9).

How to consider the human in the system?

Initially we have avoided extensive modelling of user behaviour (that is, modelling of decision procedures that can be applied dynamically) and user tasks, as such an approach would have at least two disadvantages:

[5]For a catalogue of further templates of behavioural usability properties, see [Loer, 2003, Appendix F].



a) unconstrained input space   b) prescribed user behaviour (task)   c) constrained input space

Figure 15: Illustration of the input spaces (user/environmental behaviours) that are explored in different analysis strategies.

- While user task models help focus the analysis of interactive system behaviour in relation to a particular task, the restrictive view they provide of the user and the world might rule out desirable design alternatives [Vicente, 1999].
- The description of complex decisions (for example, as might be required in dynamic function scheduling [Hildebrandt and Harrison, 2002]) requires the development of increasingly complex user models. In the context of the model-checking analysis, the addition of user models contributes to the state explosion problem.

Avoiding models of user behaviour leaves three possibilities for the analysis of models of interactive systems (as illustrated by Figure 15):

1. No assumptions about the user are made at all: As a result the model checker will explore all possible (combinations of) user inputs, see Figure 15a. One could think of the resulting user behaviour exhibited in a model-checking trace being “random” (e.g. “monkey at the keyboard”). The advantage of this approach is that behaviours are considered that were not anticipated by the designer.
2. An instance of a particular user behaviour is modelled explicitly (see Figure 15b). This behaviour may be derived from a task description. The aim here is to focus on all possible responses of a device to a given user behaviour [Loer and Harrison, 2003], for example, as might be prescribed by an operating procedure. The disadvantage of this approach is that it is not very flexible. It is hard to capture “similar” behaviours; for example, behaviour where the user waits a little longer before performing the next step, or where the user performs additional actions that are not relevant for a given goal.
3. A set of general assumptions are made by the designer about the behaviour of under-specified components. Such assumptions can be interpreted as constraints. They “impose structure, restriction[s], or limitation[s]” [Merriam-Webster, 2003] on the model, and thereby force the model checker to explore only a sub-set of the input space during the analysis (see Figure 15c).

Note that in alternatives 1 and 3 model checking is not used for formal verification, rather is used to explore device response under certain circumstances but in a rather explorative 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]).

6 Co-operative Evaluation of the tool

Both the property editor and candidate notations for the trace visualisation were investigated in co-operative evaluations, for detailed results see [Loer, 2003, Chapters 5 and 6].

6.1 Evaluation of the property editor

Of the six designers that participated in the co-operative evaluation [Monk et al., 1993] four were human-factors engineers (two with experience in systems engineering), one an industrial systems engineer, and one an academic HCI expert. Apart from the HCI expert all participants worked for the same company. None of the participants had previous model-checking experience, but all had previously attended talks about the IFADIS framework. In a thirty-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. Two fifty-minute evaluation sessions were then 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. Each trial was recorded, using video and audio. A keystroke protocol was also taken by instrumenting the IFADIS tool. At the end of each evaluation session, the participants were asked to fill in questionnaires individually.

Though the results were not representative, the familiarisation phase short, the evaluation informal and the number of participants limited, points were raised that if implemented would improve the tool. These issues address tool usability and the way that work-flow is supported. Comments were also made about applying the tool during everyday work.

Tool usability

All participants claimed to be comfortable with the user interface. The navigator bar on top of the screen was regarded as useful to determine current position in the model-checking process. However, several problems occurred in the selection and instantiation of templates. The systems engineer easily identified a range of properties that captured functional properties of interest, for example reachability, deadlock freedom, recoverability. Usability engineers found this task more difficult and wanted properties only partially supported by the tool. Differently orientated functional properties were preferred, such as "efficient menu structure", and non-functional properties, such as "ease of use", and "intuitive operation". The formulation of such properties requires a mapping from initially presented templates (describing 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 engineers who had no system engineering background, required an explanation about the meanings of expression predicates `Q`, `R`, `S`, `T` that appear in the requirements property templates, and about what they were supposed to do. Once additional explanation was provided, they were able to perform the given tasks.

Familiarisation led to improvement of the participants' performance in selecting elements for instantiation via the multiple-choice boxes. 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. Subjects remarked that it should be possible to select model configurations for each predicate in a template from a graphically presented statechart. A help system and a wizard style guide were also requested. The subjects felt that the tool gives only limited feedback about the model-checking progress. In practice, progress is hard to present given the way that the tools were constructed. In particular the time to completion of the model-checking algorithm cannot be determined in advance.

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.

Work-flow and work context

Once the system model was imported into the tool, usability experts had problems determining what further steps were required to specify requirements of the model. It was suggested that hiding parts of the interface until they become necessary for the task would help. 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"). It seems likely that these comments were 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.

The system engineer pointed out that checking individual properties is a small part of everyday work. In order to gain and maintain an overview of the overall analysis process when a statechart model is modified it was felt desirable to (i) be able to reinvoke previously checked properties and (ii) obtain a record of what properties already have been checked as well as what properties remain to be checked.

One usability engineer raised the concern that IFADIS might lead to "losing touch" with the model, and the ability to identify problems manually in the statechart. Despite our belief that the tool is only an aid to the process 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

Terminology used in the tool might not be familiar to project teams it was claimed. Project-specific terminology sometimes differs from the standard HCI terms used in the tool's usability template pane. It was proposed that the user interface be made more customisable to allow analysts to rename and add property templates. How the tool could support demonstration to authorities that a product was analysed to a sufficient extent was also discussed. 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 revision of the prototype. An optimal solution is unlikely, because work practices vary significantly. There might be merit in customisability though the need to provide a tool that can be shared between users is important.

6.2 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. The co-operative evaluation technique [Monk et al., 1993] was again applied. Five participants (aerospace engineers with some background in human-factors engineering) were used in a co-operative evaluation as well as four academics who conduct HCI research. It is clear that the relevance of visualisations depends on the stakeholder's background and application domain. This qualitative study was designed to elicit some sense of the orientation of these two communities.

Background information was provided 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. In the ensuing discussion, participants were asked comprehension questions, but but the main focus was on their opinion of the kind of information that a notation can, or should, provide. 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 used in their respective work environments, such as the OSDs discussed in Section 4.4.3. Previous experience of model animation – that is, feeding the trace information back into the STATEMATE animator, as offered by the tool described in [Mikk et al., 1997], was also considered to be desirable.

The results of this discussion are summarised in Table 1. It may be concluded from these informal discussions that variable tables, OSDs, and animation were seen as most desirable.

7 Related work

The SACRES project[6] developed an integrated environment for the design and verification of safety-critical embedded systems provides a tool for the verification of statechart models [Brockmeyer, 1999], [Bienmüller et al., 2000]. STATEMATE models can be translated to finite state machine models for the Siemens model checker SVE [Filkorn et al., 1994] or the VIS model checker [The VIS Group, 1996]. System requirements are formulated graphically as symbolic timing diagrams (STDs), a visual formalism for real-time requirements specifications [Feyerabend and Josko, 1997]. STDs are also used by the verification tool to visualise the model checking traces. The analysis tool has become available as a commercial third-party component of the STATEMATE tool (the STATEMATE MAGNUM Model Certifier). This tool is also interesting from a usability point of view. It is packaged with a manual of state machine specifications that specify temporal logic requirements patterns visually [i-Logix, 2002]. Visual representations of property templates may be more usable for designers who use state machines than the browser-approach currently implemented in the IFADIS tool.

AUTOFOCUS is a model-based systems design and verification tool [Slotosch, 2000] which supports the modelling of systems from a structural, behavioural, interaction, and data point of view. Analyses include static checks (e.g. data consistency), model checking (using the SATO [Zhang, 1997] or SMV [Cimatti et al., 2002] tools), theorem proving (for abstraction and proofs, using VSE [Hutter et al., 1999]) and specification-based test case generation.

[6]See: `http://www.tni.fr/sacres`

Experiences in the use of symbolic timing diagrams as a means of both, requirements specification and trace visualisation, are discussed in [Schlör et al., 1998].

8 Conclusions and future work

This paper has presented a discussion of a particular tool for the support of integration of model-checking into a particular aerospace design process. A compiler is described that makes it possible to check models formulated as statecharts, a notation already familiar to these engineers. Property specification is supported by a browser-style front-end that helps choose temporal logic specifications capturing 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 also provided by the tool. A study of the acceptability of further visualisations is also discussed. The evaluation of the tool, though relatively informal, provides some confidence that the tool improves the usability of model checking for these 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: Feedback provided by the tool 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, for example based on knowledge of the model size and structure, as well as the property type.

Alternative trace visualisations: Additional trace visualisations should be implemented to make the analysis results more accessible. OSDs and model animations in particular were recommended as appropriate for this design setting.

Alternative property specification and instantiation methods: Another improvement might be achieved by adding a graphical interface for selecting the state instantiations. An alternative approach to the current hierarchical browser (illustrated in Figure 5) is the use of graphical state machine patterns, like those presented in [i-Logix, 2002]. However, it needs to be investigated how such templates can be mapped to the existing models.

Improving user guidance: Rules need to be developed to aid the analysis workflow ("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?").

Using the SMV tools more efficiently: The model-checkers are currently only used in batch mode. Improved performance was achieved by activating variable re-ordering (option "`-sift`" of Cadence SMV and option "`-dynamic`" of NuSMV). 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 requirements elicitation and validation purposes and therefore are not optimised towards model checking. To avoid state explosion, the integration of state-reduction techniques, such as data abstraction, are required.

However, in conclusion, the results are promising since they lead to expectation that non model-checking experts have the potential to make effective use of these tools in order to analyse their systems more effectively.

9 Acknowledgements

This work was funded by the BAE SYSTEMS Dependable Computing Systems Centre (DCSC[7]) and the EPSRC-funded Interdisciplinary Research Collaboration in Dependability (DIRC[8]).

References

[Abowd et al., 1995] Abowd, G., Wang, H.-M., and Monk, A. (1995). A formal technique for automated dialogue development. In *Proceedings of the First Symposium on Designing Interactive Systems, DIS'95*, pages 212–226. ACM Press.

[Alur et al., 1998] Alur, R., Henzinger, T., Mang, F., Qadeer, S., Rajamani, S., and Tasiran, S. (1998). Mocha: Modularity in model checking. In Hu, A. and Vardi, M., editors, *Proceedings of CAV 98: Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 521–525. Springer Verlag.

[Amnell et al., 2001] Amnell, T., Behrmann, G., Bengtsson, J., D'Argenio, P., David, A., Fehnker, A., Hune, T., Jeannet, B., Larsen, K., Möller, M., Pettersson, P., Weise, C., and Yi, W. (2001). UPPAAL - Now, Next, and Future. In Cassez, F., Jard, C., Rozoy, B., and Ryan, M., editors, *Modelling and Verification of Parallel Processes*, number 2067 in Lecture Notes in Computer Science Tutorial, pages 100–125. Springer–Verlag.

[Belotti et al., 1995] Belotti, V., Shum, S., MacLean, A., and Hammond, N. (1995). Multidisciplinary modelling in HCI design ... in theory and in practice. In *Human Factors in Computing Systems: CHI'95 Conference Proceedings*, pages 146–153. ACM Press, Addison Wesley.

[Bienmüller et al., 2000] Bienmüller, T., Damm, W., and Wittke, H. (2000). The STATEMATE Verification Environment – Making it real. In Emerson, E. and Sistla, A., editors, *12th international Conference on Computer Aided Verification, CAV*, number 1855 in Lecture Notes in Computer Science, pages 561–567. Springer-Verlag, Berlin.

[Brockmeyer, 1999] Brockmeyer, U. (1999). *Verifikation von STATEMATE Designs*. PhD thesis, Carl-von-Ossietzky Universität Oldenburg, Oldenburg, Germany.

[Burton, 2002] Burton, S. (2002). *Automated Generation of High Integrity Test Suites from Graphical Specifications*. PhD thesis, Department of Computer Science, University of York, UK.

[Butler et al., 1998] Butler, R. W., Miller, S. P., Potts, J. N., and Carreño, V. A. (1998). A Formal Methods Approach to the Analysis of Mode Confusion. In *Proceedings of the 17th Digital Avionics Systems Conference*, Bellevue, Washington.

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

[Campos and Harrison, 2001] Campos, J. and Harrison, M. (2001). Model checking interactor specifications. *Automated Software Engineering*, 8:275–310.

[Canver, 1999] Canver, E. (1999). Einsatz von Model-Checking zur Analyse von Message Sequence Charts über Statecharts. *Ulmer Informatik Berichte*, 99-04.

[7] `http://www.cs.york.ac.uk/hise/dcsc`
[8] `http://www.dirc.org.uk`

[Carroll, 1995] Carroll, J., editor (1995). *Scenario based design: envisioning work and technology in system development*. Wiley.

[Chan et al., 1998] Chan, W., Anderson, R., Beame, P., Burns, S., Modugno, F., Notkin, D., and Reese, J. (1998). Model Checking Large Software Specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520.

[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 et al., 1999] Clarke, E., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.

[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.

[Clarke and Sistla, 1986] Clarke, E. and Sistla, E. E. A. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *Transactions on Programming Languages and Systems*, 8(2):244–263.

[d'Ausbourg, 1998] d'Ausbourg, B. (1998). Using Model Checking for the Automatic Validation of User Interfaces Systems. In Markopoulos, P. and Johnson, P., editors, *Design, Specification and Verification of Interactive Systems '98*, Proceedings of the Eurographics Workshop, in Abingdon, UK, pages 242–260. Eurographics, SpringerWienNewYork.

[Day, 1993] Day, N. (1993). A model checker for statecharts. Master's thesis, Department of Computer Science, University of British Columbia. Available as Technical Report 93-35.

[Degani, 1996] Degani, A. (1996). *Modeling Human-Machine Systems: On Modes, Error, and Patterns of Interaction*. PhD thesis, Georgia Institute of Technology.

[Dennis et al., 2000] Dennis, L. A., Collins, G., Norrish, M., Boulton, R., Slind, K., Robinson, G., Gordon, M., and Melham, T. (2000). The PROSPER Toolkit. In Graf, S. and Schwartbach, M., editors, *Tools and Algorithms for Constructing Systems (TACAS 2000)*, number 1785 in Lecture Notes in Computer Science, pages 78–92. Springer-Verlag.

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

[Duke et al., 1998] Duke, D. J., Barnard, P. J., Duce, D. A., and May, J. (1998). Syndetic Modelling. *Human Computer Interaction*, 13(4):337–393.

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

[Feyerabend and Josko, 1997] Feyerabend, K. and Josko, B. (1997). A visual formalism for real time requirement specifications. In Bertran, K. and Rus, T., editors, *Transformation-Based Reactive Systems Development, Proc. 4th International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, ARTS'97*, volume 1231 of *Lecture Notes in Computer Science*, pages 156–168. Springer-Verlag.

[Fields et al., 1997] Fields, B., Merriam, N., and Dearden, A. (1997). DMVIS: Design, Modelling and Validation of Interactive Systems. In Harrison, M. D. and Torres, J. C., editors, *Proceedings on the 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS)*, pages 29–44. Springer-Verlag.

[Filkorn et al., 1994] Filkorn, T., Schneider, H. A., Scholz, A., Strasser, A., and Warkentin, P. (1994). SVE User's Guide. Technical report, Siemens AG, ZFE T SE 1, D-81730 München, Germany.

[Flake et al., 2000] Flake, S., Mueller, W., and Ruf, J. (2000). Structured English for Model Checking Specification. In *GI-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", Frankfurt*. Gesellschaft für Informatik.

[Glinz, 1995] Glinz, M. (1995). An integrated formal model of scenarios based on statecharts. In Schäfer, W. and Botella, P., editors, *Proceedings of ESEC'95 - 5th European software engineering conference*, number 989 in Lecture Notes in Computer Science, pages 254–271. Springer, Berlin.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274.

[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.

[Hildebrandt and Harrison, 2002] Hildebrandt, M. and Harrison, M. (2002). The temporal dimension of dynamic function allocation. In S. Bagnara, S. Pozzi, A. R. and Wright, P., editors, *11th European Conference on Cognitive Ergonomics (ECCE 11)*, pages 283–292. Istituto di Scienze e Tecnologie della Cognizione Consiglio Nazionale delle Ricerche.

[Holzmann, 1997] Holzmann, G. J. (1997). The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295.

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

[Hutter et al., 1999] Hutter, D., Mantel, H., Rock, G., Stephan, W., Wolpers, A., Balser, M., Reif, W., Schnellhorn, G., and Stenzel, K. (1999). VSE: Controlling the Complexity in Formal Software Development. In Hutter, D., Stephan, W., Traverso, P., and Ullmann, M., editors, *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, number 1641 in Lecture Notes in Computer Science, pages 351–358. Springer Verlag.

[i-Logix, 2002] i-Logix (2002). STATEMATE *Magnum Certifier Pattern Library User Guide*. i-Logix Inc., OFFIS Systems and Consulting GmbH.

[IEC, 1999] IEC (1999). *IEC 61508: Functional Safety of electrical/electronic/programmable electronic safety-related systems*. International Engineering Consortium.

[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.

[Latella et al., 1999] Latella, D., Majzik, I., and Massink, M. (1999). Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker. *Formal Aspects of Computing*, 11(6):637–664.

[Lilius and Porres Paltor, 1999] Lilius, J. and Porres Paltor, I. (1999). vUML: a Tool for Verifying UML Models. Technical Report 272, Turku Centre for Computer Science, Åbo Akademi University, Department of Computer Science, Lemmingkäisenkatu, FIN-20520 Turku, Finland.

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

[Loer and Harrison, 2001] Loer, K. and Harrison, M. (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.

[Loer and Harrison, 2002] Loer, K. and Harrison, M. (2002). Towards usable and relevant model checking techniques for the analysis of dependable interactive systems. In Emmerich, W. and Wile, D., editors, *Proceedings of Automated Systems Engineering: ASE'02*. IEEE Computer Society Press.

[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, United Kingdom.

[Manna and Pnueli, 1992] Manna, Z. and Pnueli, A. (1992). *The temporal logic of reactive and concurrent systems-specification*. Springer-Verlag, New York.

[Merriam-Webster, 2003] Merriam-Webster (2003). WWWebster dictionary. `ttp://www.m-w.com/netdict.`tm accessed 27-10-2003.

[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.

[Mikk et al., 1998] Mikk, E., Lakhnech, Y., Siegel, M., and Holzmann, G. (1998). Verifying Statecharts with Spin. In Cheng, B., editor, *Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques (WIFT '98)*, Boca Raton, FL, USA. IEEE Computer Society Press.

[MoD, 1996] MoD (1996). Safety Management Requirements for Defence Systems. Defence Standard 0056, Part 2, Issue 2, Ministry of Defence, Directorate of Standardization, Kentigern House, 65 Brown Street, Glasgow G2 8EX, UK.

[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.

[Pocock et al., 2001] Pocock, S., Harrison, M., 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 IFIP TC.13 International Conference on human computer interaction*, pages 247–254. IOS Press.

[Potts et al., 1994] Potts, C., Takahashi, K., and Anton, A. (1994). Inquiry-based requirements analysis. *IEEE Software*, pages 21–32.

[Preece et al., 1994] Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1994). *Human-Computer Interaction*. Addison-Wesley, Wokingham, UK.

[Reason, 1990] Reason, J. (1990). *Human Error*. Cambridge University Press.

[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.

[Schlör et al., 1998] Schlör, R., Josko, B., and Werth, D. (1998). Using a Visual Formalism for Design Verificatin in Industrial Environments. In Margaria, T., Steffen, B., Rückert, R., and Posegga, J., editors, *Services and Visualization: Towards User-Friendly Design – ACoS98, VISUAL 98, AIN 97 Selected Papers*, volume 1385 of *Lecture Notes in Computer Science*, pages 208–221. Springer-Verlag.

[Slotosch, 2000] Slotosch, O. (2000). Modelling and validation: AUTOFOCUS and QUEST. *Formal Aspects of Computing*, 12(4):225–227.

[Smith and Mosier, 1986] Smith, S. L. and Mosier, J. N. (1986). Guidelines for designing user interface software. Technical Report 9420, Mitre Corporation.

[The VIS Group, 1996] The VIS Group (1996). VIS: A System for Verification and Synthesis. In Alur, R. and Henzinger, T., editors, *Proceedings of the 8th International Conference on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 428–432, New Brunswick, NJ. Springer Verlag.

[Vicente, 1999] Vicente, K. (1999). *Cognitive Work Analysis*. Lawrence Erlbaum Associates.

[Wharton et al., 1994] Wharton, C., Rieman, J., Lewis, C., and Polson, P. (1994). The Cognitive Walkthrough Method: A Practitioner's Guide. In Nielsen, J. and Mack, R., editors, *Usability Inspection Methods*, chapter 2. John Wiley & Sons, Inc.

[Zhang, 1997] Zhang, H. (1997). SATO: An e-client propositional prover. In McCune, W., editor, *Proceedings of the 14th International Conference on Automated Deduction*, number 1249 in Lecture Notes in Artificial Intelligence, pages 272–275. Springer Verlag.