

# Application of Model Checking for the Model-based Analysis of Interactive Systems

(An experience report)

Karsten Loer

Karsten.Loer@gl-group.com

---

---

---

---

---

---

---

---

## Overview

- General idea: Using model-checking (MC) to discover latent errors in open system models
- Sample application: mobile appliances
  - using MC to assess safety implications of the introduction of a new mobile device (“Pucketizer”) into a processing plant
- Technical solutions:
  - Modelling interactive systems as open systems
  - Continuous analysis and model refinement
  - Introducing assumptions on behaviour of open variables
- Conclusions

---

---

---

---

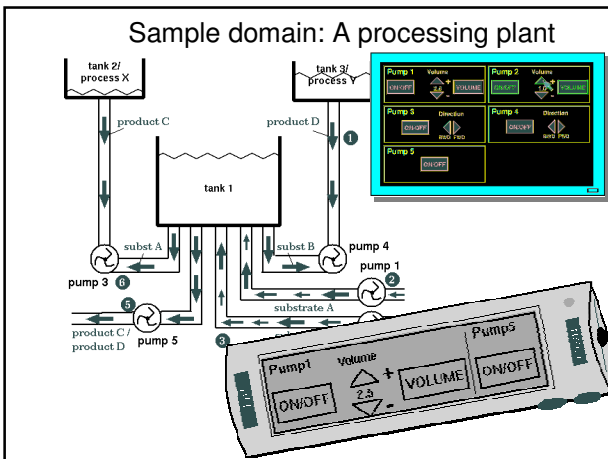
---

---

---

---

## Sample domain: A processing plant



---

---

---

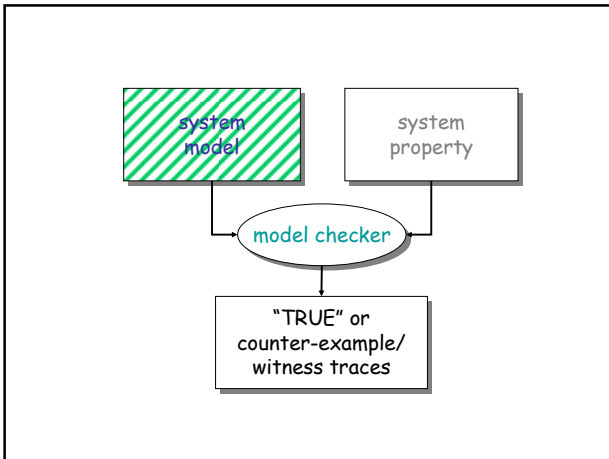
---

---

---

---

---




---

---

---

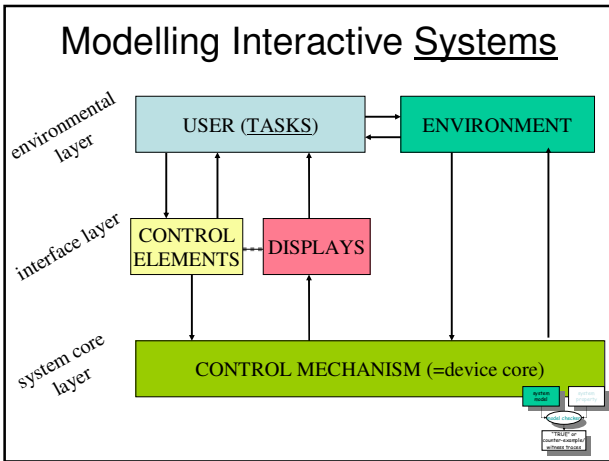
---

---

---

---

---




---

---

---

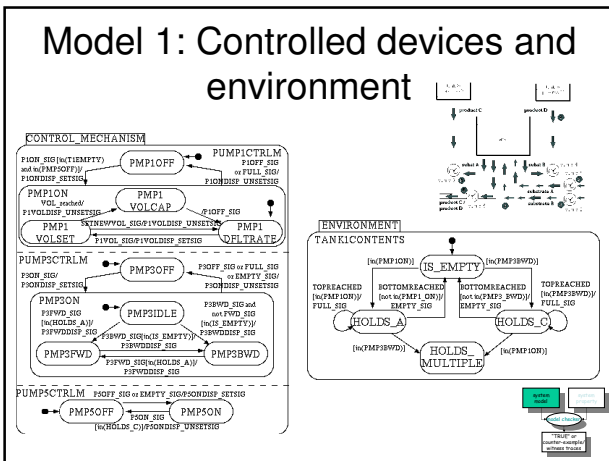
---

---

---

---

---




---

---

---

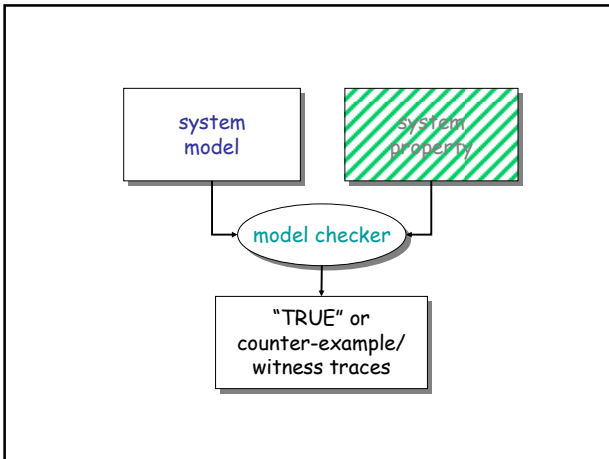
---

---

---

---

---




---

---

---

---

---

---

---

---

### Analysis: Model "plausibility"

Does the model behave as intended?

- "sanity": deadlock-freedom, state/event reachability
- "goal reachability":
  - Can product C be produced?
  - What is the easiest way to produce product C?
  - What is the "best" way to produce C under assumptions  $a_1 \dots a_n$ ?
  - Is it possible to reach unsafe states? ...

A diagram showing the flow from 'Model' to 'Set of requirements' to 'Analysis'. The 'Model' box contains a state transition diagram with two figures looking at it. The 'Set of requirements' box contains a stack of three document icons. The 'Analysis' box contains a smaller version of the model checker diagram from the first slide.

---

---

---

---

---

---

---

---

### Temporal aspects of interest:

Characteristics of user tasks\* in terms of temporal system-behaviour:

- task sequencing:
  - task interleaving, suspension and resumption
- task durations and optimisation:
  - e.g. best-case/worst-case execution times
  - multi-valued decision criteria
- task allocation
  - who needs to perform the task and when?

\*note that we do not attempt to produce user models

A small version of the model checker diagram from the first slide, located at the bottom right of the slide.

---

---

---

---

---

---

---

---

## Formulating System Requirements

- only input sequences containing "1-2-3" are accepted:
  - "all sequences containing "1-2-3" are accepted"  
 $AG ("1" \ \& \ AX ("2" \ \& \ AX "3")) \rightarrow s3$
  - "any other sequence is rejected"  
 $AG (!("1" \ \& \ AX ("2" \ \& \ AX "3")) \rightarrow !s3)$
- the accepting state can only be reached, if the inputs are made within a particular duration




---

---

---

---

---

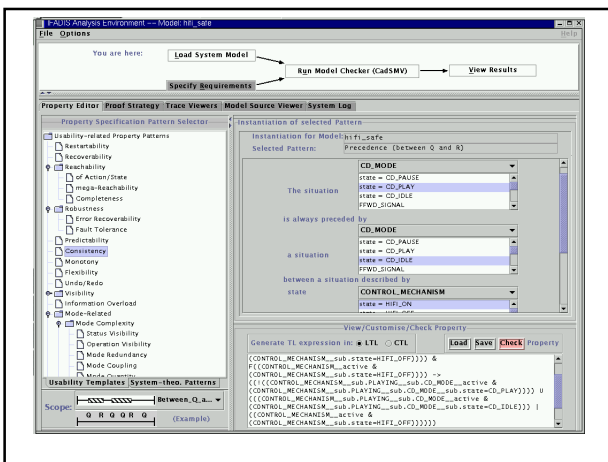
---

---

---

---

---




---

---

---

---

---

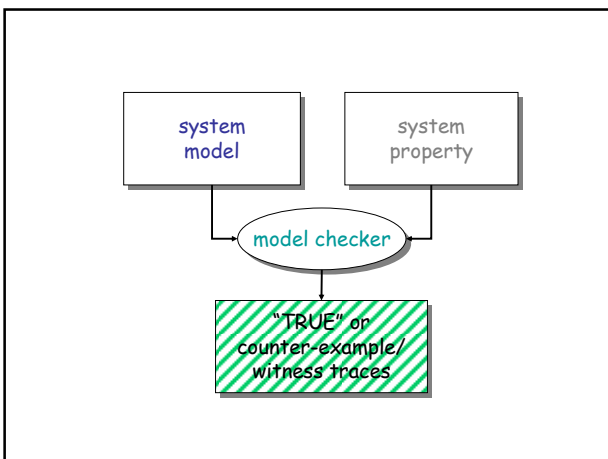
---

---

---

---

---




---

---

---

---

---

---

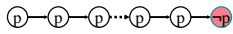
---

---

---

---

## Model-checking traces



- *trace* = sequence of execution steps that demonstrate how a state that violates (or demonstrates) a property can be reached from the initial system state.
- traces can point the analyst to:
  - violating user/device behaviour
  - task optimisations
  - recovery procedures




---

---

---

---

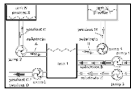
---

---

---

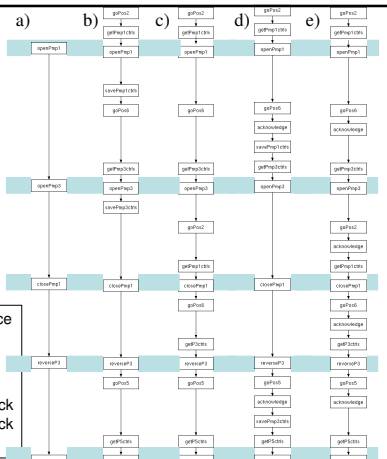
---

## Trace Comparison



Goal/Property:  
"Reachability of a state where end product C is released"

- a) Control room interface
- b) Pucketizer
- c) Pucketizer (forgetful operator)
- d) Pucketizer w. Interlock
- e) Pucketizer w. Interlock (reluctant operator)




---

---

---

---

---

---

---

---

## Explorative application of model checking

1. starting from a device-centric model  
=> all possible user inputs
  2. gradually add assumptions about user and environment behaviour  
=> sub-set of "sensible" user inputs
- formulation of assumptions:
    1. as part of the (temporal logic) property specification
    2. by model enhancements (e.g. "observer automata" or model decorations)

---

---

---

---

---

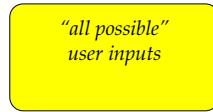
---

---

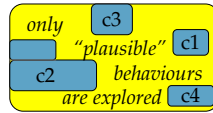
---

### Influence of user task models on explored input space

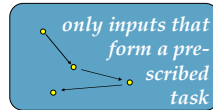
- no task model



- restricted input space



- normative task model




---

---

---

---

---

---

---

---

### Restricting the input space:



Focus of analysis:

Given:

1. a device specification and
2. a desired target "situation" (= state of the device and environment)

Question: *What assumptions can/need to be made about the user?*

---

---

---

---

---

---

---

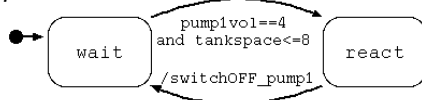
---

### Restricting the input space<sub>2</sub>



- Goal: Limit search by adding restrictions (i.e. set of state machines) on the user behaviour
- Example: Prevent tank overflow

*"Whenever the user realises that pump 1 is operating full volume while its target tank is close to full the user will switch off the pump"*




---

---

---

---

---

---

---

---

## Normative task models



Focus of analysis:

Given: A specification of

1. the device under development,
2. relevant parts of the environment and
3. a normative task model

Question: *What states of the environment can be reached?*

---

---

---

---

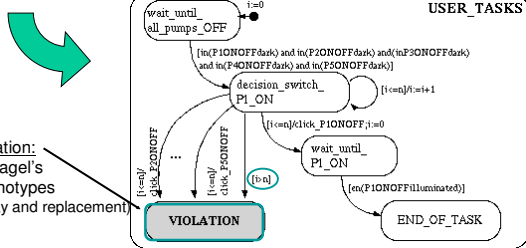
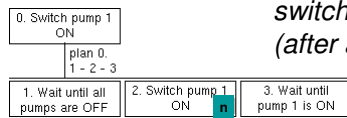
---

---

---

---

Example task: “Once all pumps are off, switch pump 1 ON (after at most  $n$  steps)”



Task violation:  
e.g. Hollnagel's error phenotypes  
(here: delay and replacement)

---

---

---

---

---

---

---

---

## Adding assumptions about operator behaviour<sub>1</sub>



- temporal logic assertions:  
“the operator always forgets to store pump controls”

```

assert SAN1:
  F ((PUMP5CTRLM.state=PMP5ON)
    & (TANK1.state=HOLDS_C));
assert alwaysForget:
  G!(savePmp1Controls| ... |savePmp5Controls);

assume alwaysForget;

using alwaysForget prove SAN1;
    
```

---

---

---

---

---

---

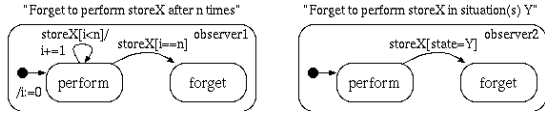
---

---

## Adding assumptions about operator behaviour<sub>2</sub>



- observer automata: the “forgetful” operator



- check properties under the assumption that violation states (“**forget**”) are absent

---

---

---

---

---

---

---

---

---

---

## Parametric search:

**Question:** “For what value(s) of variable  $x$  does property  $\pi$  hold?”

- iterative analysis:

“Does  $\pi$  hold  
for  $x==n$ ,  
for  $x==n-1$ ,  
for  $x==n+1, \dots$  ?”

- alternative: query checking

[Chan 2000, Gurfinkel et al. 2002]

---

---

---

---

---

---

---

---

---

---

## Conclusions:

Model checkers are good at...

- exhaustive and “automatic” analysis\*  
(\*provided that “appropriate input” is supplied)
- analysis of behavioural reachability properties
  - ordering/sequencing of tasks:  
e.g. Hollnagel’s error phenotypes:  
repetition, reversal, omission, delay, premature action, replacement, insertion, and “intrusion”
  - mode complexity
  - dialogue control:  
visibility of action effects, visibility of available actions, recoverability, consistency, error prevention, flexibility, efficiency of use
  - timing
  - Usability heuristics (cf. [Loer & Harrison 2000])

---

---

---

---

---

---

---

---

---

---

### Conclusions<sub>2</sub>:

Model checking has limitations...

- deliver single, sometimes “trivial”, traces
- hard/impossible to determine tendencies, e.g. certain types of user behaviour, characteristics of components that contribute to potential errors ...
- technique does not suggest corrections
- difficult/unsuitable to use for analysis of representational properties (layout, direct manipulation etc.)

---

---

---

---

---

---

---

---

### Conclusions<sub>3</sub>: Tool suitability

feature \ tool		SMV	Uppaal	HyTech
temporal properties	sequencing	+	+	+
	real-time	-	+	+
	simulation	-	+	-
	parametric search	(o) <sup>1,2</sup>	o <sup>1</sup>	+
	continuous variables	+	o	+
	scalability	+	o	-
	usability	+	+	-

<sup>1</sup> via iteration over “parameter” variable(s)  
<sup>2</sup> execution step as unit of progress

---

---

---

---

---

---

---

---

### Acknowledgements

- This work was performed in collaboration with Michael Harrison of the University of Newcastle upon Tyne.
- Many thanks for their support to the colleagues from:
  - BAE SYSTEMS Dependable Computing Systems Centre at the University of York, UK.  
<http://www.cs.york.ac.uk/hise/dcsc>
  - Interdisciplinary Research Collaboration in Dependability (DIRC project), UK.  
<http://www.dirc.org.uk>

---

---

---

---

---

---

---

---

## Further Reading

- **An Integrated Framework for the Analysis of Dependable Interactive Systems (FADIS): Its tool support and evaluation**, Karsten Loer and Michael Harrison, Automated Software Engineering Journal, to appear.
- **Integrating Model Checking with the Industrial Design of Interactive Systems**, Karsten Loer and Michael D. Harrison, Department of Computer Science, University of York, UK.  
Proceedings of ICSE'04 Workshop "Bridging the Gaps II: Bridging the Gaps Between Software Engineering and Human-Computer Interaction", Edinburgh, UK, 24/25. May, pp.9-16, 2004.
- **Towards usable and relevant model checking techniques for the analysis of dependable interactive systems**  
Karsten Loer and Michael Harrison  
Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02), Edinburgh, UK, 23-27 September, pp. 223-226, 2002.
- **Formal Interactive Systems Analysis and Usability Inspection Methods: Two Incompatible Worlds?**  
Karsten Loer and Michael Harrison  
*Proceedings of the 7th International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS)*, Limerick, Ireland, June 5-6, 2000, pp 169-190, Volume 1946, Lecture Notes in Computer Science, Springer Verlag.

---

---

---

---

---

---

---

---