Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Online Monitoring of Complex Conditions for Event-based Distributed Architectures

Prof. Dr. Holger Giese, Lucas Sakizloglou, Jochen Hänsel

Raum A-2.1

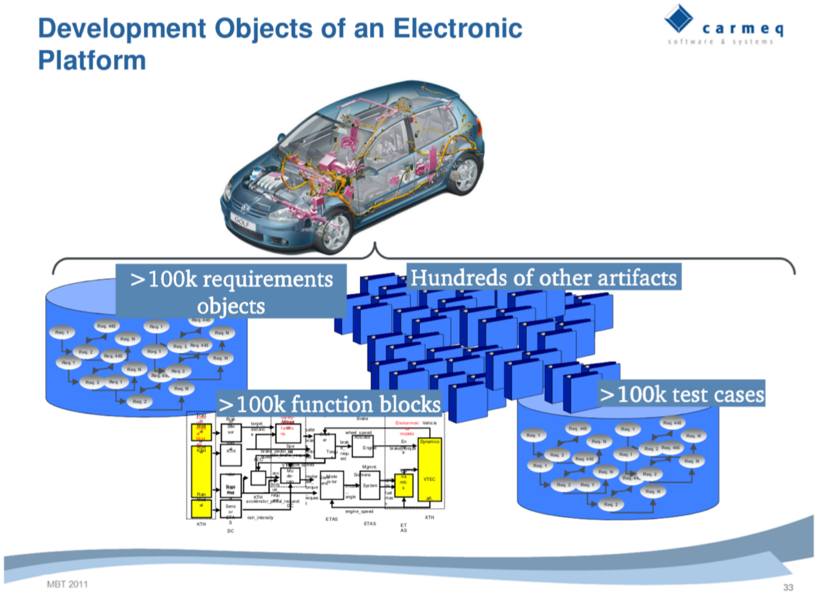
Email lucas.sakizloglou@hpi.uni-potsdam.de

Distributed Architectures

Let’s pick the title apart..

Multiple independent actors that are not collocated.

Actions taken by a component in one location require information from other locations



Development Objects of an Electronic Platform

carmeq
software & systems

>100k requirements objects

Hundreds of other artifacts

>100k function blocks

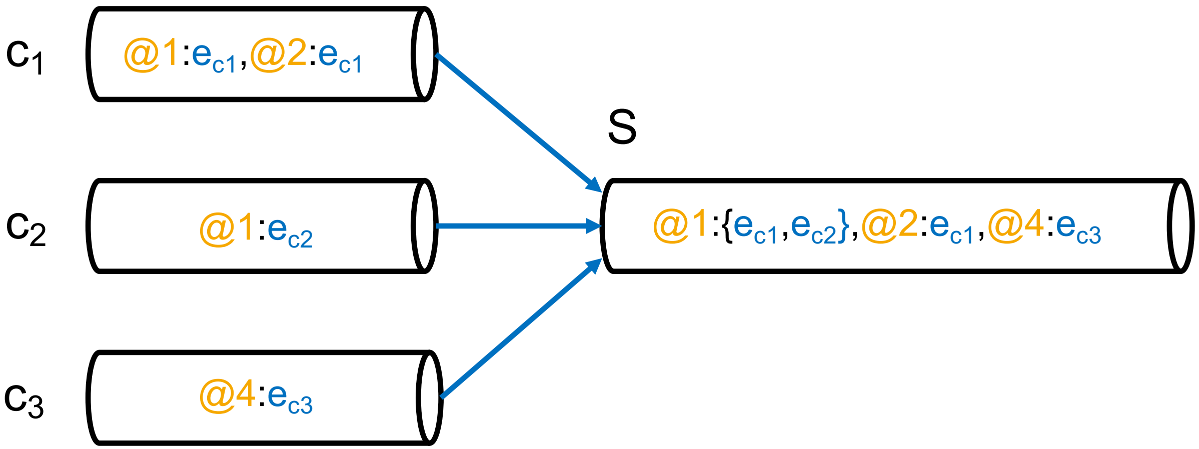
>100k test cases

MBT 2011

33

…but wait! There's more!

Event-based? *Specific type of systems that are able to emit/create an event for each change*


c_1
$@1{:}e_{c1}, @2{:}e_{c1}$
c_2
$@1{:}e_{c2}$
c_3
$@4{:}e_{c3}$
S
$@1{:}\{e_{c1}, e_{c2}\}, @2{:}e_{c1}, @4{:}e_{c3}$

In this PS not so much data processing -- We assume a system that can generate a trace of timed events

Complex Conditions

Conditions? Statements/Expectations about software behavior

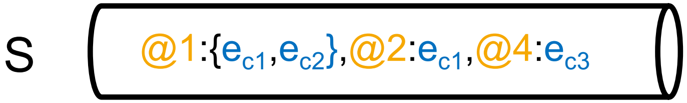
Examples:

If there is an e_{c1} then…

- there should also be an e_{c2} (proposition)
- eventually there should also be an e_{c2} (temporal)
- in the next 5 secs. there should also be an e_{c2} (metric)
- in the next 5 secs. for the same e_{c1} there should also be an e_{c2} (binding)

If there is an e_{c1} with var=10 then… (data)

Typical trade-off:
efficiency vs.
expressiveness

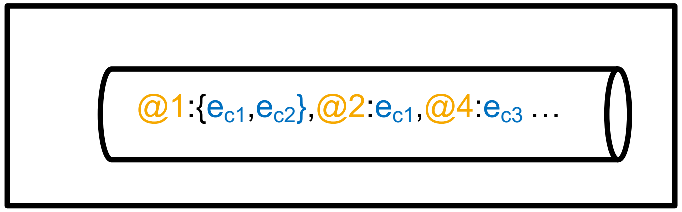


@1:{e_{c1},e_{c2}},@2:e_{c1},@4:e_{c3}

Online Monitoring

Online Monitoring of a condition can mean different things (e.g. throwing an exception is a form of online monitoring). Here:

Trace: σ



@1:{e_{c1},e_{c2}},@2:e_{c1},@4:e_{c3} …

Behavior description: m

c_1:e_{c1}(var_1)
c_2:e_{c2}
c_3:e_{c3}

Condition: q

"…in the next 5 secs. for the same e_{c1} there should also be an e_{c2} …"

magic(m,q) = monitor

monitor(σ) = {true,false,?}

Key Points for Online Monitoring

monitor(σ) = {true,false,?}

“Incoming!”

$\sigma' = \sigma + e_{c1}$,   execute: monitor(σ’)

$\sigma'' = \sigma' + e_{c1}$,   execute: monitor(σ’’)

…

What happens with millions of events..?

Incremental checking

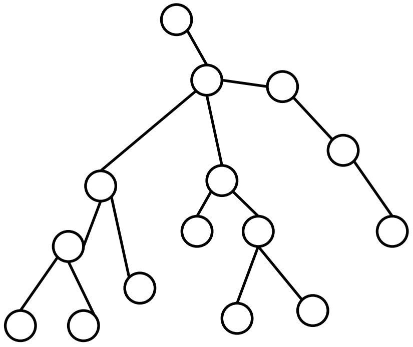
Also monitor(t) runs in parallel with the system..

Minimal (memory and runtime) overhead

Why is Online Monitoring Relevant?

- Specification: abstract model of an execution (mathematical object, hence “powered by formal methods”)
- q: (formal) statement about software behavior
- Checks whether every observed state of one behavior satisfies q

all possible system behaviors



path: one behavior

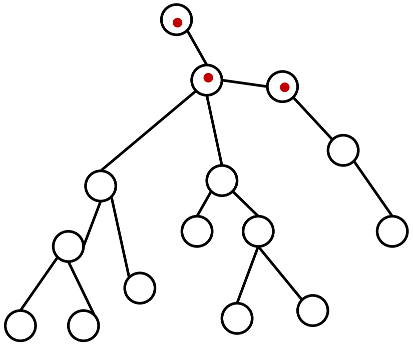
Testing

- Either an *executable* or *code-base,* at any stage of development

- t*:* test-case (executable piece of software) derived from statement

- *Testing*: Checks whether t is satisfied from one behavior

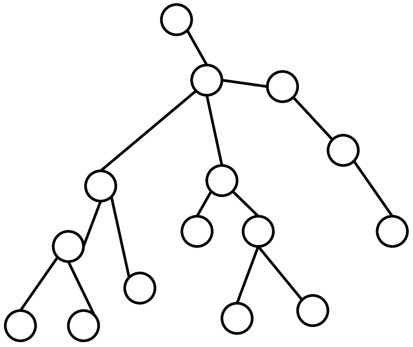
all possible system behaviors



Static Verification

- *Specification*: A description of a software functionality expressed in a language whose vocabulary, syntax, and semantics are formally defined
- q: (formal) statement about software behavior
- *Static Verification*: Checks whether q is satisfied from all possible system behaviors

all possible system behaviors



Conclusion

Online monitoring is a powerful technique that tries to accomplish an optimal trade-off between testing (incomplete) and more formal methods (often infeasible)

Due to systems becoming more and more complex, online monitoring is a very active field of research, i.e. application on safety-critical contexts and emerging domains

In this project seminar we will get hands-on experience with online monitoring tools while achieving a working understanding with theoretical concepts

What the Student can Expect

- Familiarity and intuition specification of conditions

  it is not always simple to understand what you want to test

- Familiarity with (state-of-the-art) online monitoring tools

  Nifty tools that can be used for multiple purposes and from multiple environments (command line, JAVA, etc)

  Diverse language paradigms (Python, Scala, *OCaml*, etc) usually available online

  Many use-cases (banking, OS, CERN Performance Analysis , etc)

- Extending experience/insights in testing a system

Enrollment

Start

- Today ☺ April 16
- For meetings, we will find another slot if necessary
- Until April 26: Enrollment
- April 29: Meeting to organize topics/lectures/meetings

Email: lucas.sakizloglou@hpi.de

Grading

- Project (50%)
- Report (40%):

  Introduce the approach

  Summarize tool/benchmark

  Outcomes (performance, usability)
- Presentation (10%)

  Summarize your work

Resources

Tools:

- DejaVu
- MonPoly
- BeepBeep (check documentation)

Papers: (check Repo folder named “papers”)

- Runtime Verification at Work: A Tutorial (especially the section RV-Monitor)
- Runtime Verification: From Propositional to First-Order Temporal Logic

Dates, Milestones

- Project Seminar date (SD): flexible
- No regular meetings during the semester

  2-3 offered lectures on tools/benchmarks/property specification

  Coordination meeting on project development (agile?)

  contact Teaching Assistant (TA) for specific appointments
- *Milestones*: M[1-5] – [Deadline] – [Milestone]
  - M1 – ?.? – Tool/Benchmark selection
  - M2 – ?.? – Discussion of plan with TA
  - M3 – 3 wks. before {$SD} – Discussion of project and and presentation with TA
  - M4 – 1 wk. before {$SD} – Discussion of report draft and presentation with TA
  - M5 – 1 wk. after {$SD} – Report submission