THALES

Applying TLA+ in a Safety-Critical Railway Project

Stefan Resch
Dependable Systems Architect, Thales

ENABLE-S3

This project has received funding from the ECSEL Joint Undertaking under grant agreement No 692455. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation program and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, Norway.

Overview TAS Control Platform

Safety approval according to CENELEC 50129 SIL 4

Safety layer

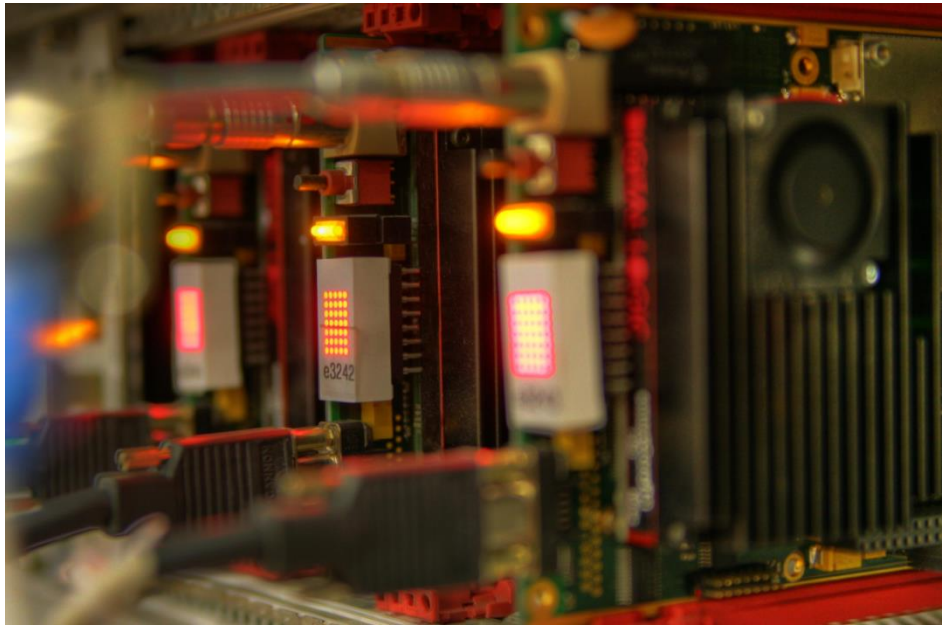
- Fault tolerance
- Health monitoring
- Various redundancy architectures

Board support package

- Communications interfaces / drivers
- Some are very specific

Based on COTS hardware / operating system

Support 25 years of application business logic (with changing underlying hardware and software)



TAS Control Platform in about 70% of new safety-critical products in Thales Ground Transportation

Formal Modelling Objectives

New Redundancy Architecture

Gain confidence in design

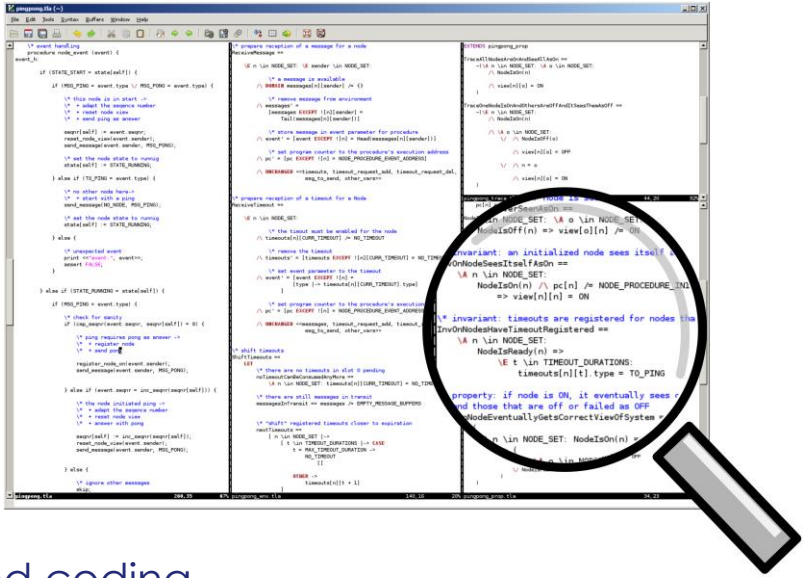
- Relevant results/insights

Feasible

- Time and effort

Close representation to implementation

- Reduce bugs introduced between model and coding
- Get feedback from experienced experts in team



Some time later …

The Chosen Approach

Formal model in TLA+ and PlusCal

- Algorithm specification in C-like language
- Linear temporal logic formulas
- Model checking with TLC

Efficient model checking through a good level of abstraction

- Prerequisite for design process

Property-driven design

- Process that provides confidence in model

C code generation from model

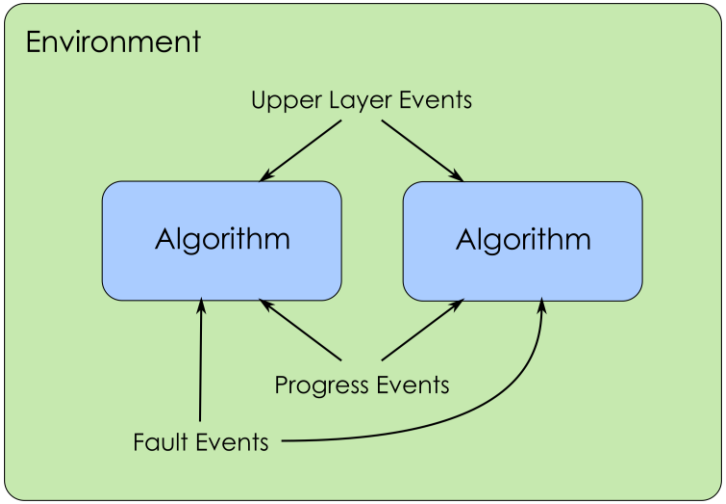
- Close gap between model and implementation

Formal Model in TLA+ and PlusCal

Algorithm defined in PlusCal

Environment in TLA+

- 3 classes of events
  - Progress
  - Upper Layer
  - Fault



Properties

- Invariants: e.g. no two masters
- Liveness: e.g. all nodes eventually get a correct view of the system

TLC model checker

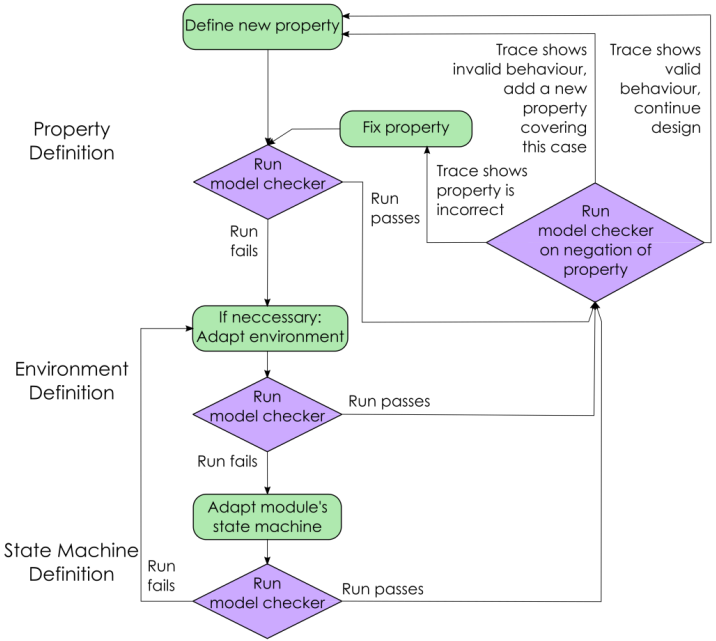
So we found a model checker …

Property-Driven Design

Analogous to test driven development

Gain confidence in model

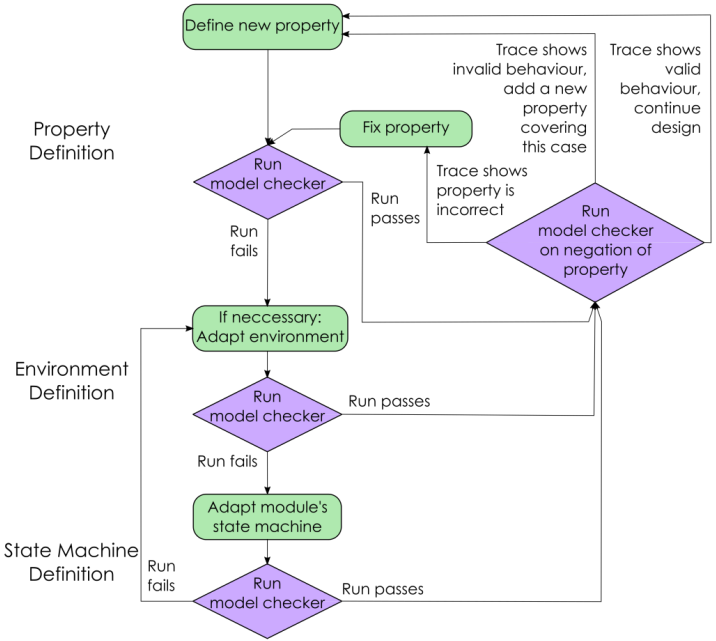
- Observe many traces of model
- Check that properties capture intended behaviour
- Fast feedback to design decisions
- Feedback from increasing size of state space



Property-Driven Design

Recap test driven development

- Write new test
- Execute test → fail
- Write implementation
- Execute all tests → pass
- Refactor

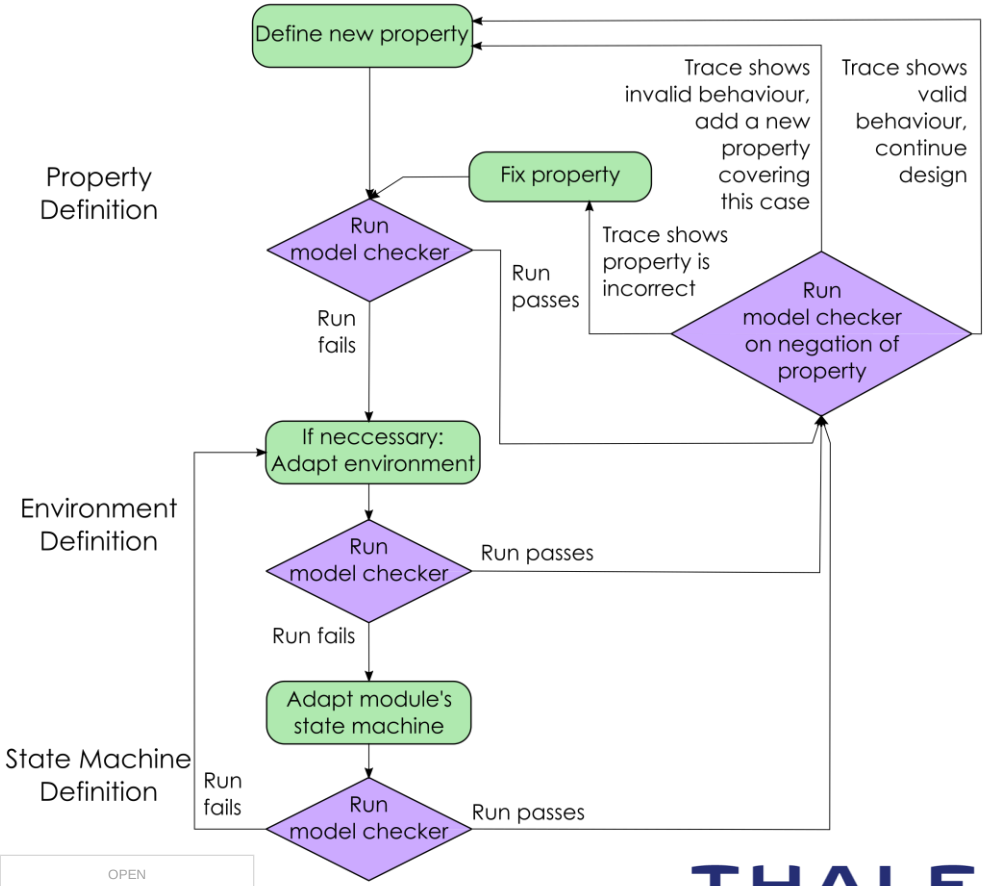


THALES

Property-Driven Design

Steps

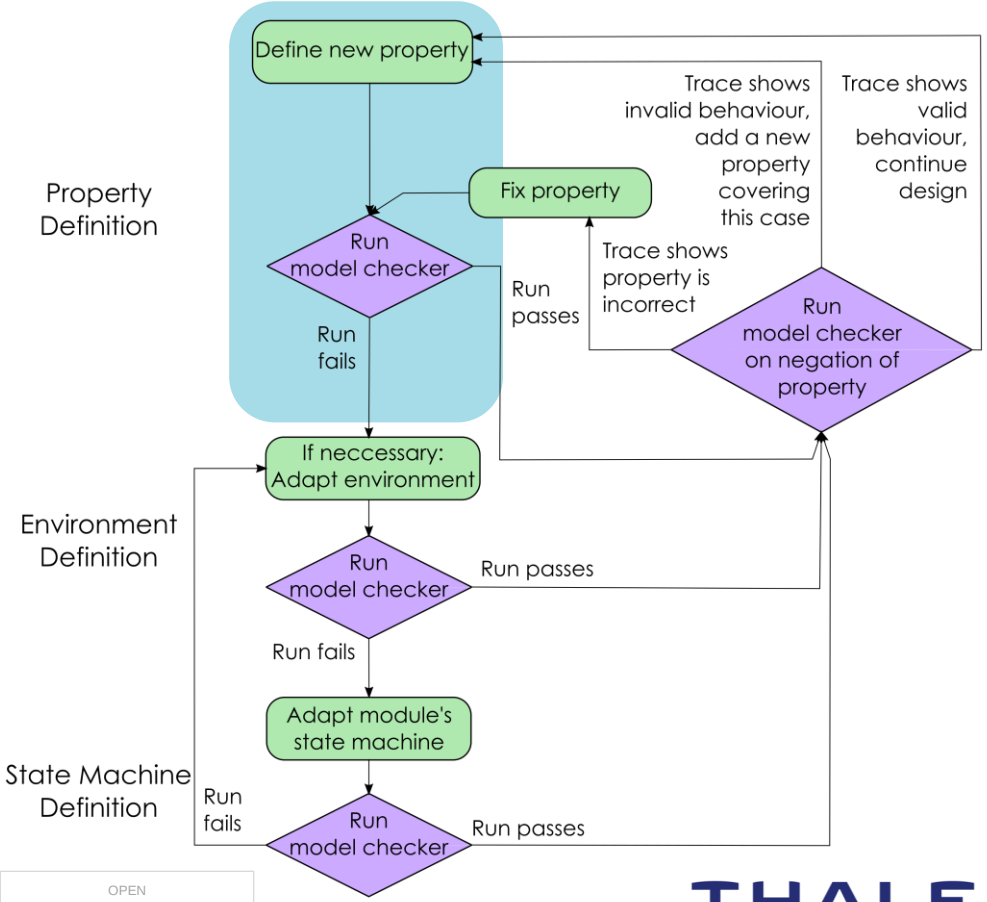
- Define property
- Enhance environment
- Adapt algorithm
- Cross check with negation of property



Property-Driven Design

Steps

- Define property
- Enhance environment
- Adapt algorithm
- Cross check with negation of property

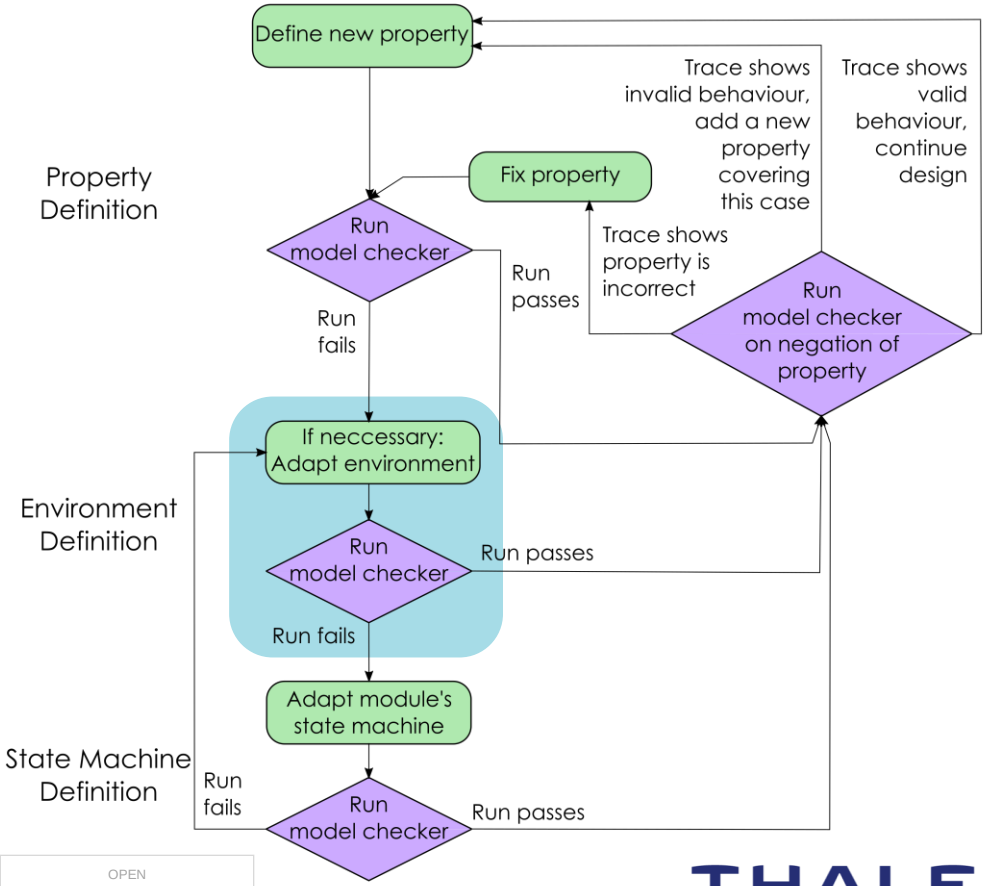


THALES

Property-Driven Design

Steps

- Define property
- Enhance environment
- Adapt algorithm
- Cross check with negation of property

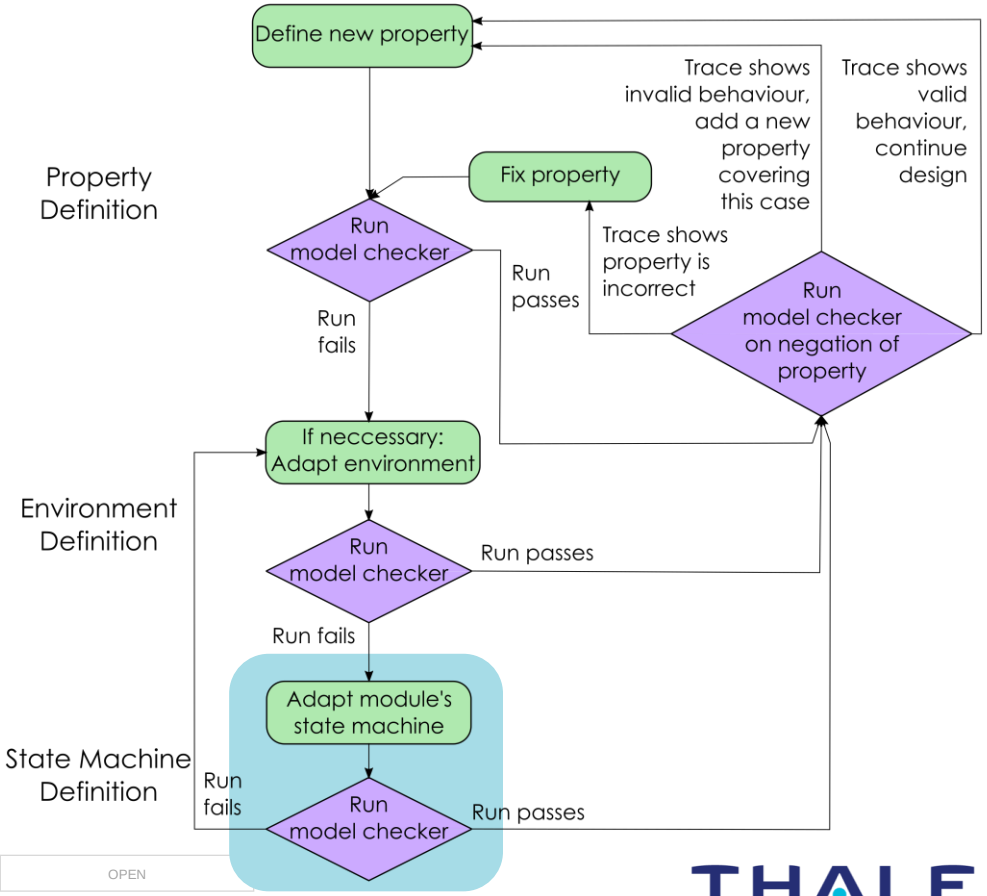


THALES

Property-Driven Design

Steps

- Define property
- Enhance environment
- Adapt algorithm
- Cross check with negation of property

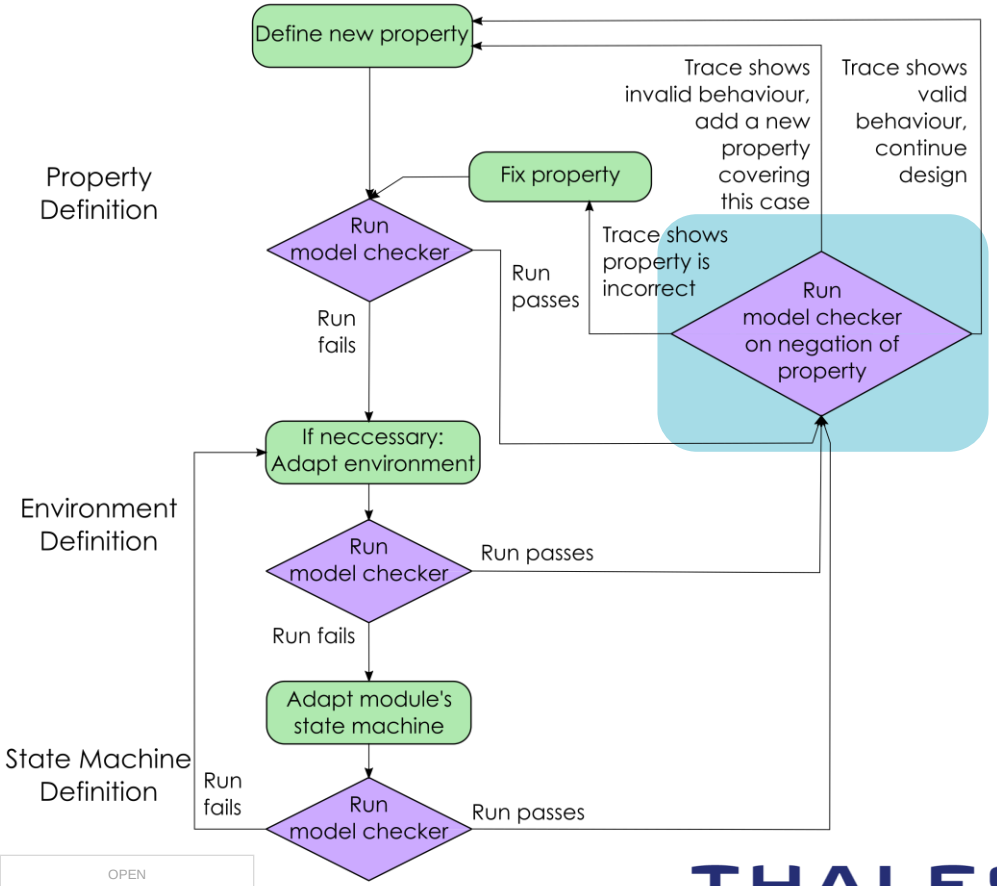


THALES

Property-Driven Design

Steps

- Define property
- Enhance environment
- Adapt algorithm
- Cross check with negation of property

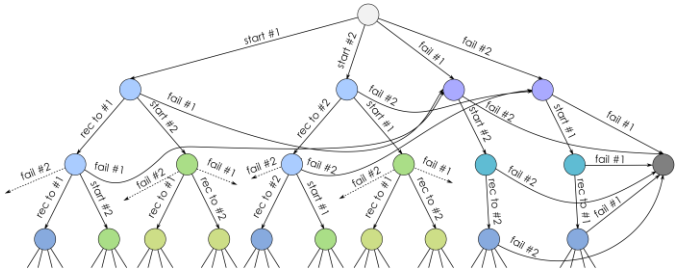


THALES

Property-Driven Design

Track size of state space between model checker runs

- Constant size but algorithm got more complex
  - Most likely added dead code
- Sudden increase of size
  - Most likely a bug in specification



Property-Driven Design

How properties drive the process

- Invariants
  - Help detecting “bugs”
- Liveness
  - Force enhancement of algorithm

Property-Driven Design

Instant feedback supports design decisions

Many traces build confidence in model

Unanticipated scenarios were found

- Longest trace consisted of more than 30 steps

Major drawback is time necessary for modelling

PlusCal to C Transformation

PlusCal

```
\* initialization
procedure node_init () {
init:
   state[self] := STATE_START;

   \* set all but own state to off
   reset_node_view(NO_NODE);

   \* start a timeout to send
   \* ping for the first time
   start_timeout(TO_PING,
      TO_PING_STARTUP_TIME);
}
```

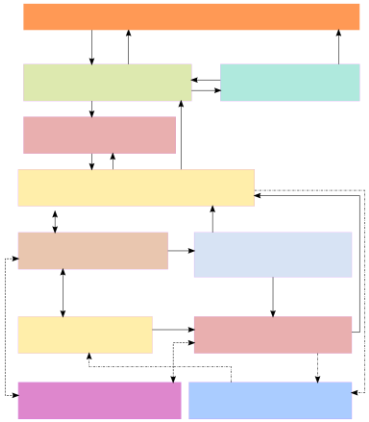
C

```
// initialization
void node_init (void) {

   state = STATE_START;

   // set all but own state to off
   reset_node_view(NO_NODE);

   // start a timeout to send ping
   // for the first time
   start_timeout(TO_PING,
      TO_PING_STARTUP_TIME);
}
```

Defining Different Modules

Specific environment definition necessary

- Avoid state space explosion
- Substantial effort



Strategy for interfaces is essential

Layering seems to be the simplest approach

Potential Improvements

Readable Traces

Animation

Faster check of liveness properties

```
trace.tladiff (~/tmp)
File  Edit  Tools  Syntax  Buffers  Window  Help
31 -     2 :>
32 +         ( 0 :> "__unused_init_value" @@
33 +           1 :> "__unused_init_value" @@
34 +           2 :> "__unused_init_value" ) )
35 +    /\ msg_to_send = [no_message |-> ""]
36 +    /\ event = (0 :> defaultInitValue @@ 1 :> defaultInitValue @@ 2 :> defaultInitValue)
37 +    /\ timeout_request_del = [type |-> "__notimeout"]
38 +    /\ stack = (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>)
39
40 step 2: <Action line 434, col 5 to line 439, col 22 of module pingpong_env>
41      /\ timeouts = ( 0 :>
42           ( 0 :> [type |-> "__notimeout"] @@
43             1 :> [type |-> "__notimeout"] @@
44             2 :> [type |-> "__notimeout"] ) @@
45       1 :>
46           ( 0 :> [type |-> "__notimeout"] @@
47             1 :> [type |-> "__notimeout"] @@
48             2 :> [type |-> "__notimeout"] ) @@
49       2 :>
50           ( 0 :> [type |-> "__notimeout"] @@
51             1 :> [type |-> "__notimeout"] @@
52             2 :> [type |-> "__notimeout"] ) )
53      /\ messages = ( 0 :> (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>) @@
54        1 :> (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>) @@
55        2 :> (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>) )
56      /\ seqnr = (0 :> 0 @@ 1 :> 0 @@ 2 :> 0)
57 +    /\ pc = (0 :> "init" @@ 1 :> "off" @@ 2 :> "off")
58 -    /\ pc = (0 :> "off" @@ 1 :> "off" @@ 2 :> "off")
59      /\ state = ( 0 :> "__unused_init_value" @@
60        1 :> "__unused_init_value" @@
61        2 :> "__unused_init_value" )
62      /\ timeout_request_add = [type |-> "__notimeout"]
63      /\ view = ( 0 :>
64           ( 0 :> "__unused_init_value" @@
65             1 :> "__unused_init_value" @@
66             2 :> "__unused_init_value" ) @@
67       1 :>
68           ( 0 :> "__unused_init_value" @@
69             1 :> "__unused_init_value" @@
70             2 :> "__unused_init_value" ) @@
71       2 :>
72           ( 0 :> "__unused_init_value" @@
73             1 :> "__unused_init_value" @@
74             2 :> "__unused_init_value" ) )
75      /\ msg_to_send = [no_message |-> ""]
76      /\ event = (0 :> defaultInitValue @@ 1 :> defaultInitValue @@ 2 :> defaultInitValue)
77      /\ timeout_request_del = [type |-> "__notimeout"]
78      /\ stack = (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>)
79
80 step 3: <Action line 448, col 9 to line 452, col 22 of module pingpong_env>
81      /\ timeouts = ( 0 :>
82           ( 0 :> [type |-> "__notimeout"] @@
83             1 :> [type |-> "__notimeout"] @@
84             2 :> [type |-> "__notimeout"] ) @@
85       1 :>
86           ( 0 :> [type |-> "__notimeout"] @@
87             1 :> [type |-> "__notimeout"] @@
88             2 :> [type |-> "__notimeout"] ) @@
89       2 :>
90           ( 0 :> [type |-> "__notimeout"] @@
91             1 :> [type |-> "__notimeout"] @@
92             2 :> [type |-> "__notimeout"] ) )
93      /\ messages = ( 0 :> (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>) @@
94        1 :> (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>) @@
95        2 :> (0 :> <<>> @@ 1 :> <<>> @@ 2 :> <<>>) )
96      /\ seqnr = (0 :> 0 @@ 1 :> 0 @@ 2 :> 0)
97 +    /\ pc = (0 :> "Error" @@ 1 :> "off" @@ 2 :> "off")
98 -    /\ pc = (0 :> "init" @@ 1 :> "off" @@ 2 :> "off")
99 +    /\ state = (0 :> "state_start" @@ 1 :> "__unused_init_value" @@ 2 :> "__unused_init_value")
100 -    /\ state = ( 0 :> "__unused_init_value" @@
101 -      1 :> "__unused_init_value" @@
102 -      2 :> "__unused_init_value" )
trace.tladiff                                                    31,1          51%
```

OPEN

THALES

Lessons Learnt

- Valuable insights to system design
- Justifiable overhead
- Very close to implementation

Further Material

Environment with example algorithm

- https://github.com/stresch/pingpong