R.Gerlich, R.Gerlich (BSSE)

Integrated Design and Testing of Safety-Critical Real-time Systems in Space

MBTUC11
Model-Based Testing

MBT User Conference 2011

19.10.2011 Berlin, Germany

Dr. Rainer Gerlich BSSE System and Software Engineering
Auf dem Ruhbühl 181
88090 Immenstaad
Germany

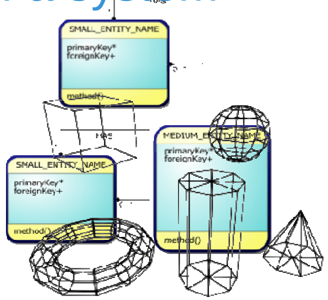
Tel. +49/7545/91.12.58
Fax +49/7545/91.12.40
Mobil +49/171/80.20.659
email Rainer.Gerlich@bsse.biz

MBT in context of MDE

Model =
Abstract representation
of a system

Test in context of V&V:

- Tests support verification
- Tests support validation



Model

Stimulation + Test of the Model

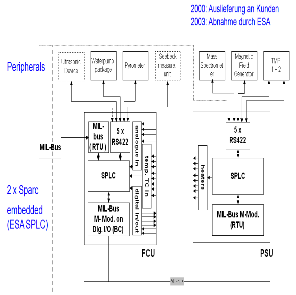
Verification

- provide input-output vectors
- support fault injection

Testing on modelling and on target level:
Are these two different things?

Validation

- provide feedback on specification
- provide performance figures



Target System

Stimulation + Test of the code

MBT in context of MDE and Automation

Code generator generates the code
+ instrumentation for recording of properties

Test generator generates the test environment
for stimulation

Root of both is the

MODEL

The whole process is driven
from the model only!

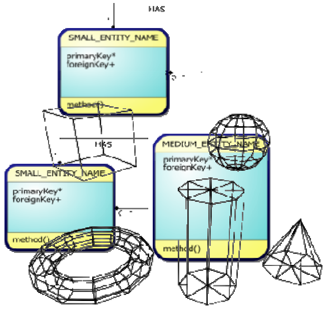
No manual intervention!

V&V on target is in focus, too

Automatic Verification
Automatic Generation
Automatic Testing

Verification

Trash on modelling level
=
More trash on target level

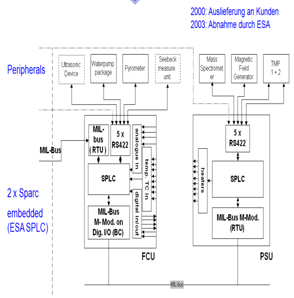


Model

Auto-Stimulation + Test of the model

Refine the Model (and the Modelling Environment) using Feedback from the System

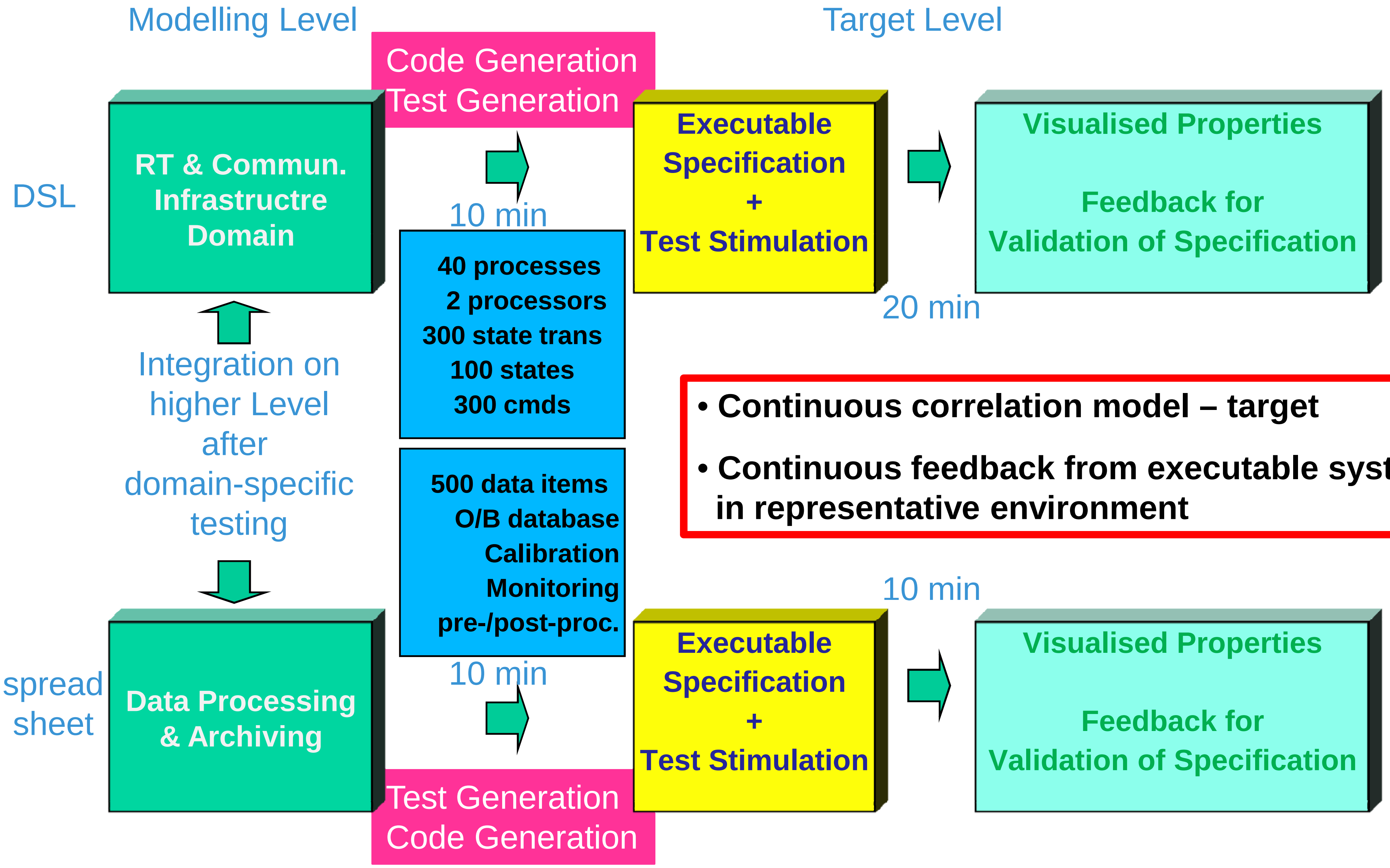
System Generation
Test Generation



Target System

Auto-Stimulation + Test of the code

The Power of Fully Automated Process Chains and Integration



Modelling Level

Target Level

DSL

RT & Commun. Infrastructre Domain

Code Generation
Test Generation

10 min

Executable Specification + Test Stimulation

Visualised Properties

Feedback for Validation of Specification

20 min

40 processes
2 processors
300 state trans
100 states
300 cmds

Integration on higher Level after domain-specific testing

500 data items
O/B database
Calibration
Monitoring
pre-/post-proc.

spread sheet

Data Processing & Archiving

10 min

Test Generation
Code Generation

Executable Specification + Test Stimulation

10 min

Visualised Properties

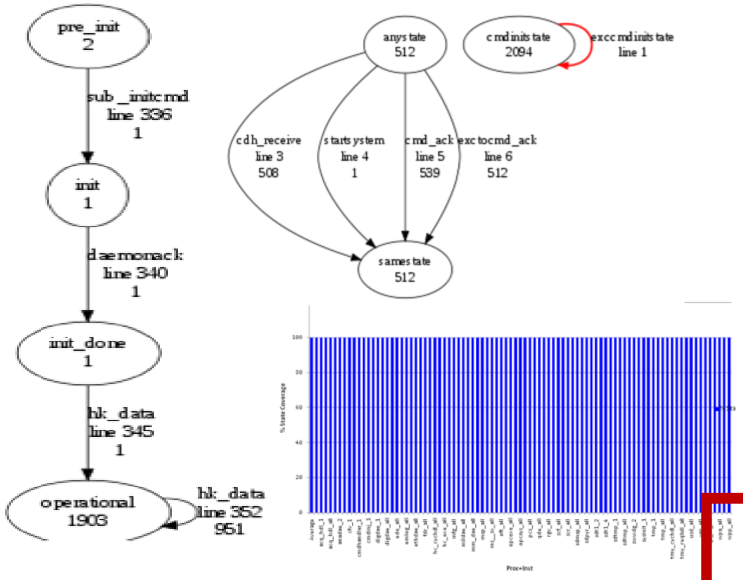
Feedback for Validation of Specification

- **Continuous correlation model – target**
- **Continuous feedback from executable system in representative environment**

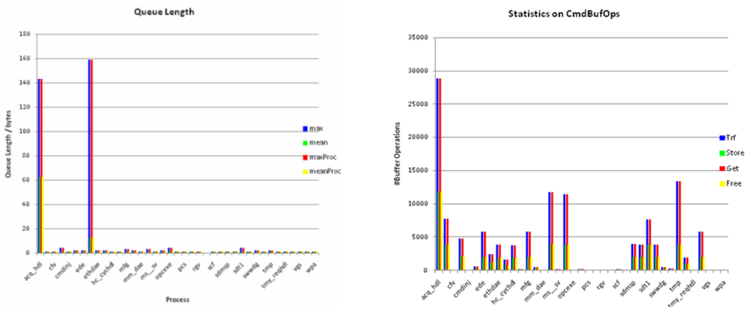
Feedback by Visualisation (examples only)

Bessere + **S**ichere **S**oftware **E**ffizient erzeugen

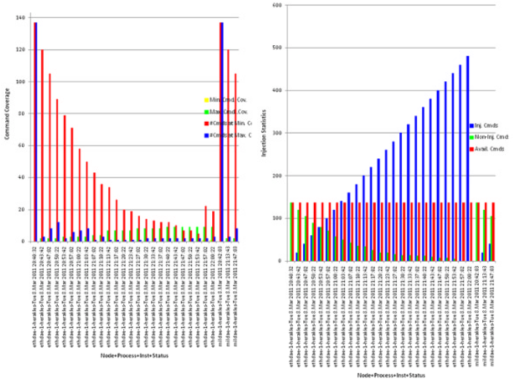
BSSE System and Software Engineering



state+trans coverage

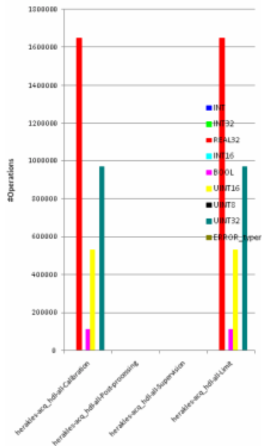


command buffer statistics



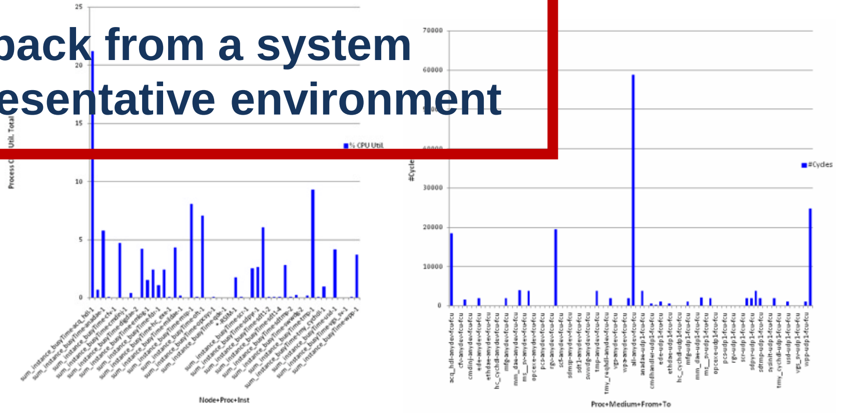
cmd inj statistics

Due to automated model-based stimulation immediate feedback from a system executable in a representative environment

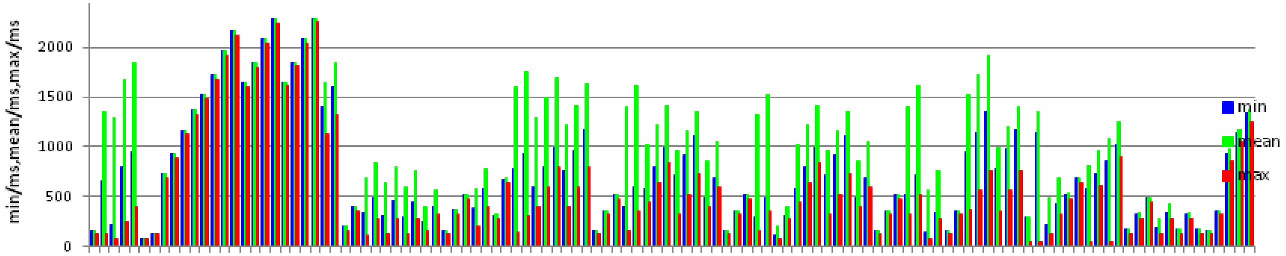


DB operations atomic data types

Command + Data Profiles



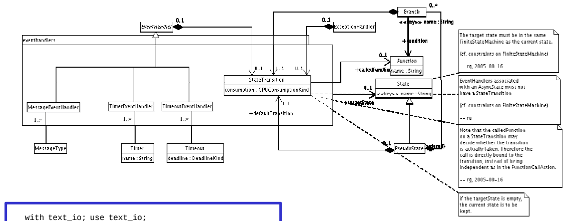
CPU + Channel Utilisation



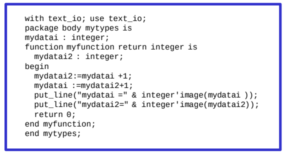
Response Times fw-bw

What is a Model?

A model represents a specification

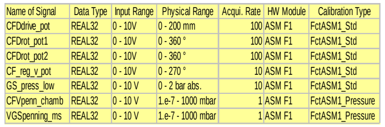

Branch
0..*
<<key>> name : String
EventHandler
0..1
ExceptionHandler
0..1
+condition
eventhandlers
0..1
Function
name : String
+calledFunction
0..1
0..1
0..1
StateTransition
consumption : CPUConsumptionKind
0..1
State
<<key>> name : String
+targetState
MessageEventHandler
TimerEventHandler
TimeoutEventHandler
0..1
+defaultTransition
1..*
1..*
1..*
MessageType
Timer
name : String
Timeout
deadline : DeadlineKind
0..1
PseudoState
The target state must be in the same FiniteStateMachine as the current state.
(cf. constraints on FiniteStateMachine)
rg, 2005_08_16
EventHandlers associated with an AsyncState must not have a StateTransition
(cf. constraints on FiniteStateMachine)
-- rg
Note that the calledFunction on a StateTransition may decide whether the transition is actually taken. Therefore the call is directly bound to the transition, instead of being independent as in the FunctionCallAction.
-- rg, 2005-08-16
if the targetState is empty, the current state is to be kept.

UML?



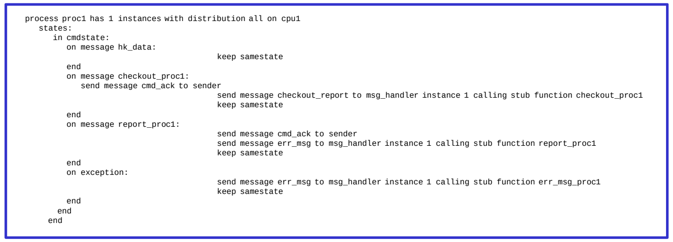
```
with text_io; use text_io;
package body mytypes is
mydatai : integer;
function myfunction return integer is
  mydatai2 : integer;
begin
  mydatai2:=mydatai +1;
  mydatai :=mydatai2+1;
  put_line("mydatai =" & integer'image(mydatai ));
  put_line("mydatai2=" & integer'image(mydatai2));
  return 0;
end myfunction;
end mytypes;
```

Ada code when re-engineering?



Name of Signal	Data Type	Input Range	Physical Range	Acqui. Rate	HW Module	Calibration Type
CFDdrive_pot	REAL32	0 - 10V	0 - 200 mm	100	ASM F1	FctASM1_Std
CFDrot_pot1	REAL32	0 - 10V	0 - 360 °	100	ASM F1	FctASM1_Std
CFDrot_pot2	REAL32	0 - 10V	0 - 360 °	100	ASM F1	FctASM1_Std
CF_reg_v_pot	REAL32	0 - 10V	0 - 270 °	10	ASM F1	FctASM1_Std
GS_press_low	REAL32	0 - 10 V	0 - 2 bar abs.	10	ASM F1	FctASM1_Std
CFVpenn_chamb	REAL32	0 - 10 V	1.e-7 - 1000 mbar	1	ASM F1	FctASM1_Pressure
VGSpenning_ms	REAL32	0 - 10 V	1.e-7 - 1000 mbar	1	ASM F1	FctASM1_Pressure

Contents of a spreadsheet?



```
process proc1 has 1 instances with distribution all on cpu1
   states:
      in cmdstate:
         on message hk_data:
                                        keep samestate
         end
         on message checkout_proc1:
            send message cmd_ack to sender
                                        send message checkout_report to msg_handler instance 1 calling stub function checkout_proc1
                                        keep samestate
         end
         on message report_proc1:
                                        send message cmd_ack to sender
                                        send message err_msg to msg_handler instance 1 calling stub function report_proc1
                                        keep samestate
         end
         on exception:
                                        send message err_msg to msg_handler instance 1 calling stub function err_msg_proc1
                                        keep samestate
         end
       end
     end
```

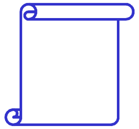
DSL code?

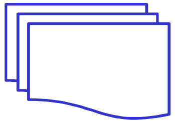
Taking a specification as base for tests

All model types we have used for

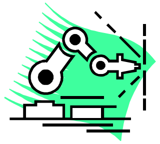
- **code generation (fully automated)**
- **test generation (fully automated)**

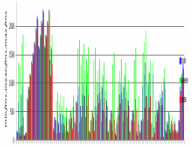
What is a Test Input on Modelling Level?



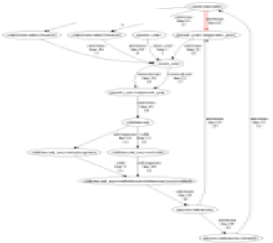


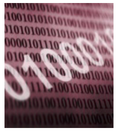
A test plan and test procedures derived from a model?



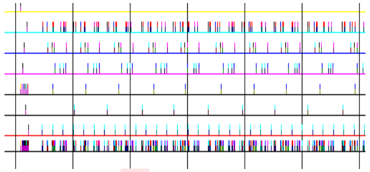


**Test stimuli automatically derived from a model
and automatically documented together with results?**





Stimulus for an FSM	valid or invalid
Stimulus for commands, msgs and data	valid or invalid, lost



A set of non-functional parameters	deadline, timeout, period
A variation of ideal parameters	time jitter, execution time

A test input may also be omitting of an expected input

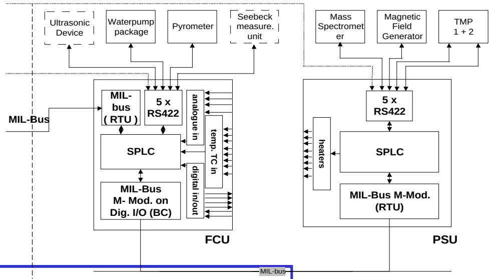
e.g. in case of fault injection: loss of data or events

important for critical, fault-tolerant systems

Automatically derived test stimuli may also support early operation of a system

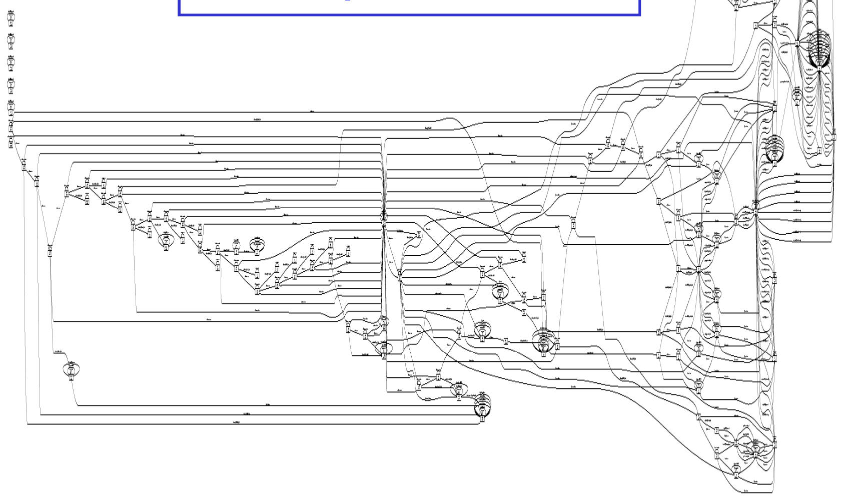
Material Science Laboratory ISS

2000: tool delivered to customer
2003: system accepted by ESA
2009: launched and put into operation



~ 40 processes
~200 states
2 processors

real-time infrastructure + TC + TM + database:
~80 KLOC when tool delivered
expanded by customer over 3 years



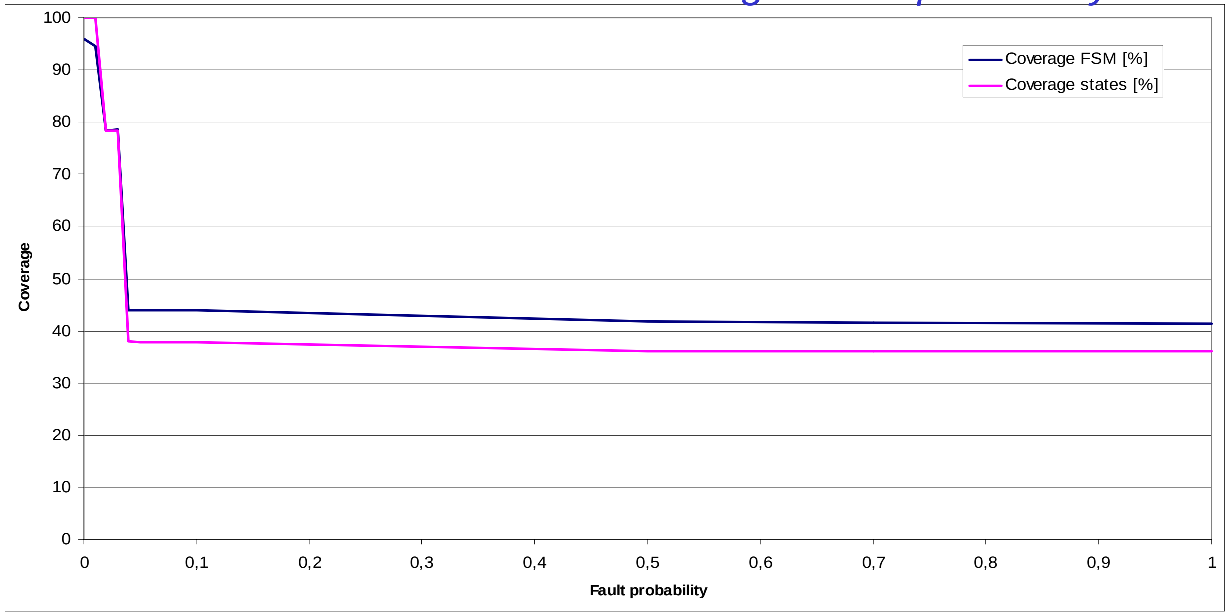
Generation:
~15 min from modelling language ISGL or Excel
automatically generated reports on properties
Input: ~500 Excel-lines

3 faults detected in first version of code and test generators from 2000 - 2003:

- limitation to 250 ground commands
- task priority list for distributed system not correct
- overflow in union (16 bit cmd counter)
- no more faults flagged from the project since 2003/2009

Early Design Validation

State and State Transition Coverage vs. FI probability



- FSM stimulation by messages
- Loss of messages / events
- varying fault probability

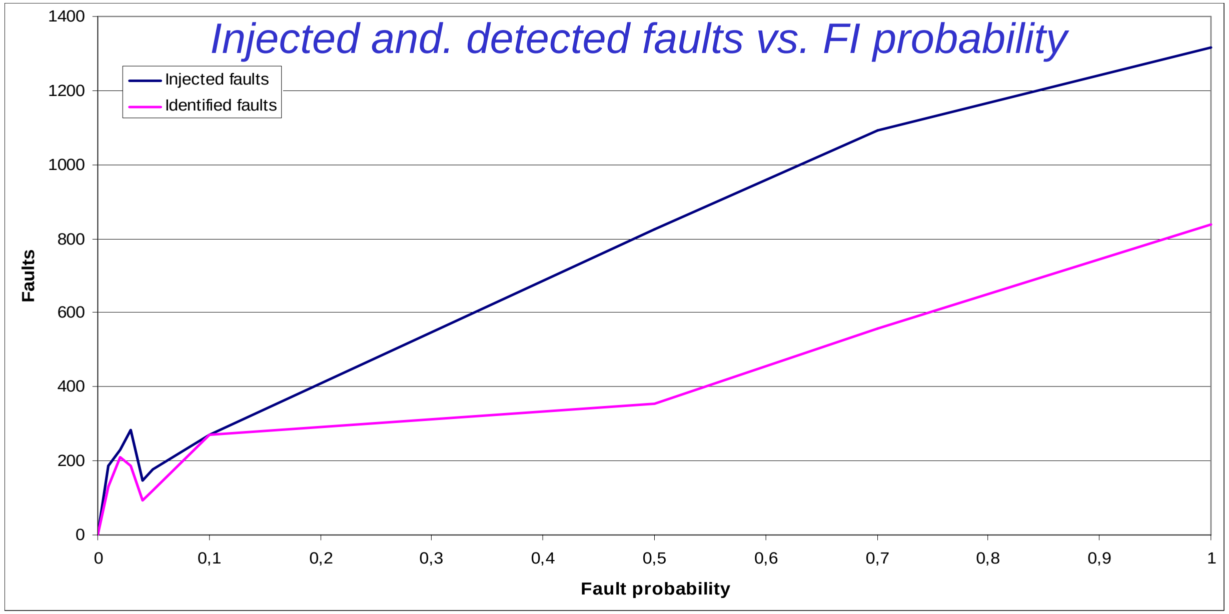
High probablity of a system collapse already at very small probablity of 0.5% for data loss

Evaluation Means
Fault Injection =
Inversion of positive functionality

System Initialisation Procedure

<Probability to reach the end> = ?

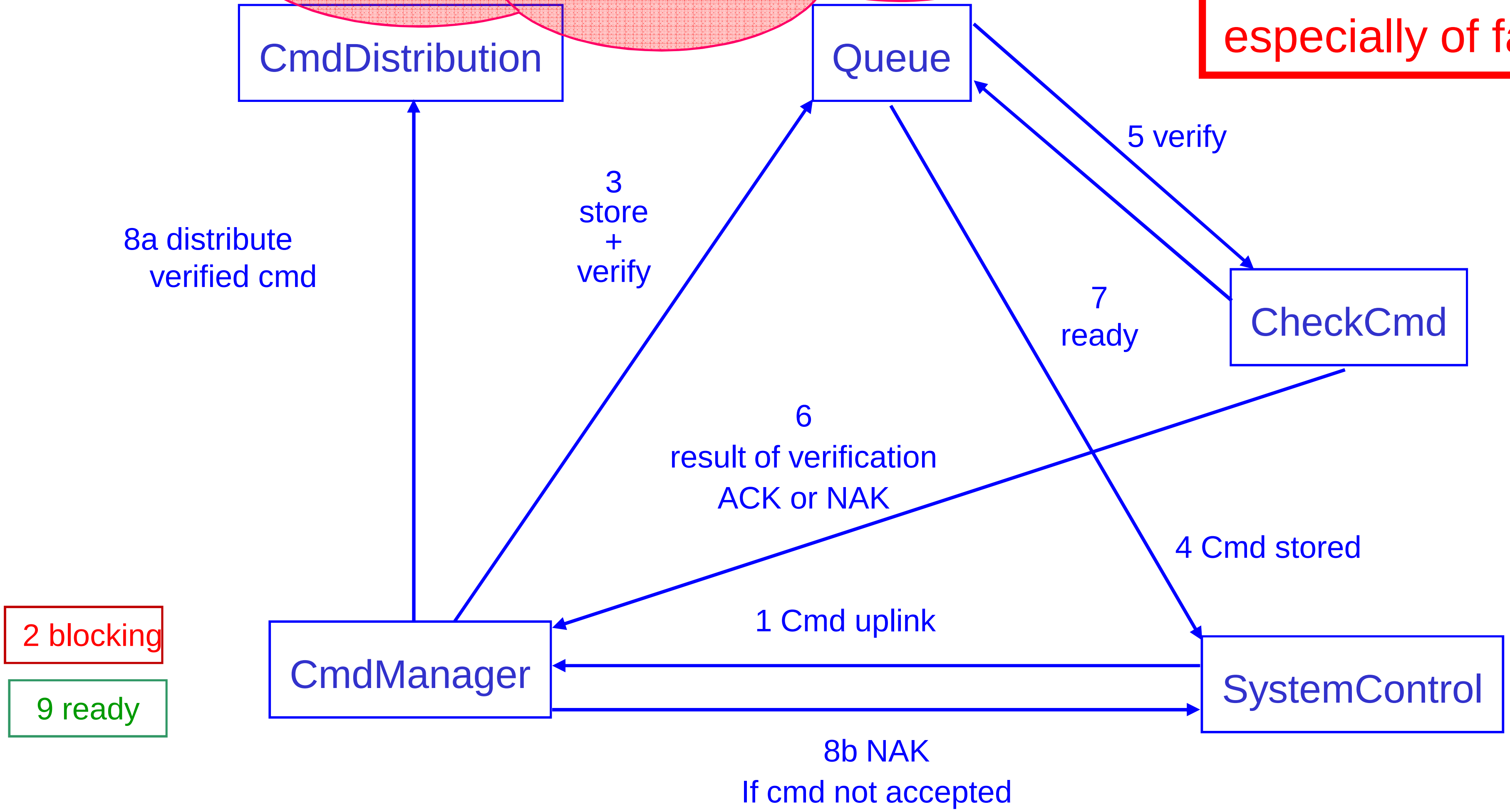
How critical is my system?



Bridge from UML: Command Manager

Pure functional verification based on UML failed:
non-functional properties were out-of-scope
and
performance and fault tolerance issues could not be detected
due to stepwise stimulation
while massive stimulation is required

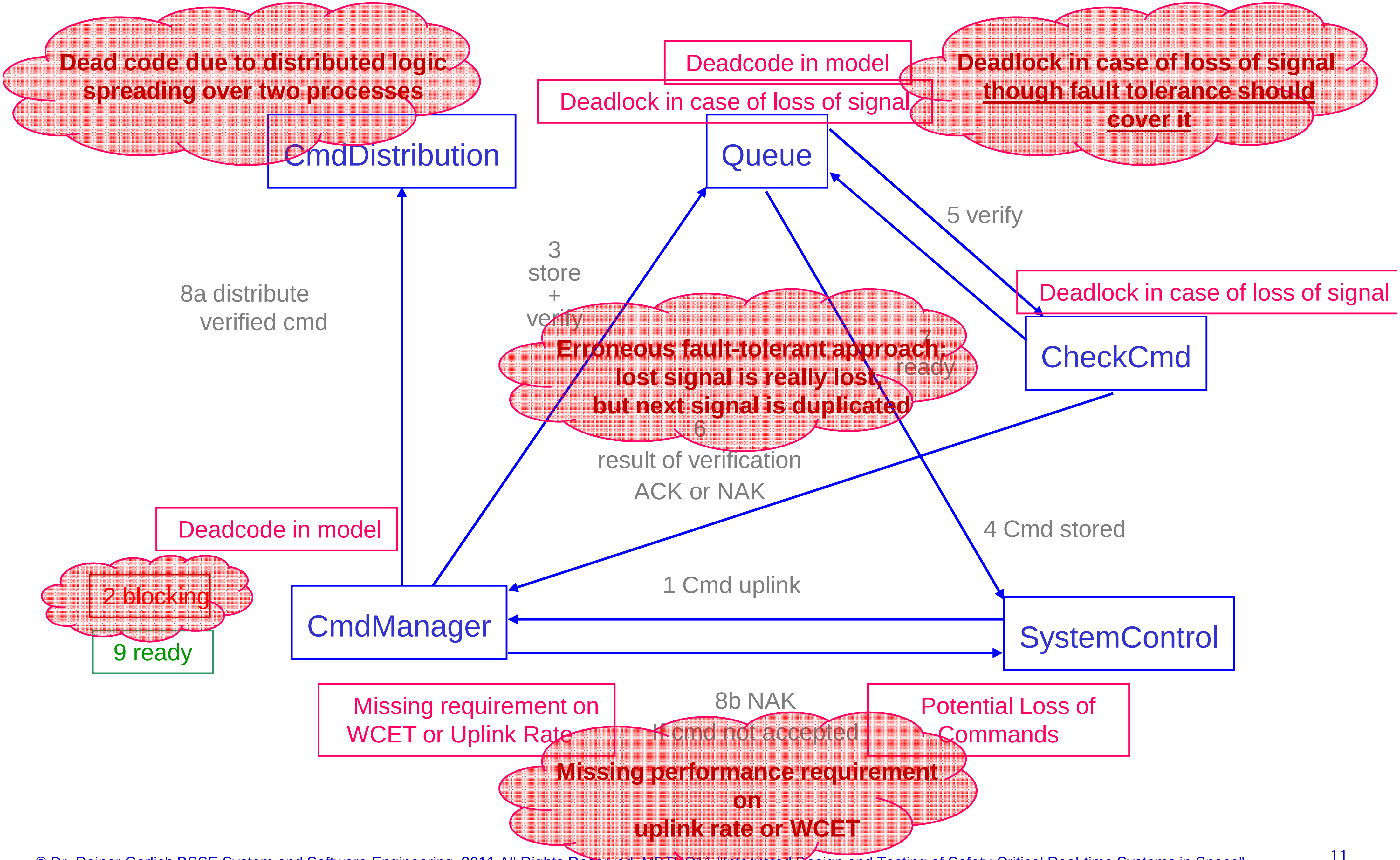
verified previously
by a UML tool
but no support of
non-functional properties
especially of fault injection



Evaluation of non-functional properties In presence of Fault Injection

Bessere + **S**ichere **S**oftware **E**ffizient erzeugen

BSSE System and Software Engineering



Dead code due to distributed logic spreading over two processes

Deadcode in model

Deadlock in case of loss of signal

Deadlock in case of loss of signal though fault tolerance should cover it

CmdDistribution

Queue

5 verify

3 store + verify

8a distribute verified cmd

Deadlock in case of loss of signal

CheckCmd

Erroneous fault-tolerant approach: lost signal is really lost, but next signal is duplicated

7 ready

6 result of verification ACK or NAK

4 Cmd stored

Deadcode in model

2 blocking

9 ready

CmdManager

1 Cmd uplink

SystemControl

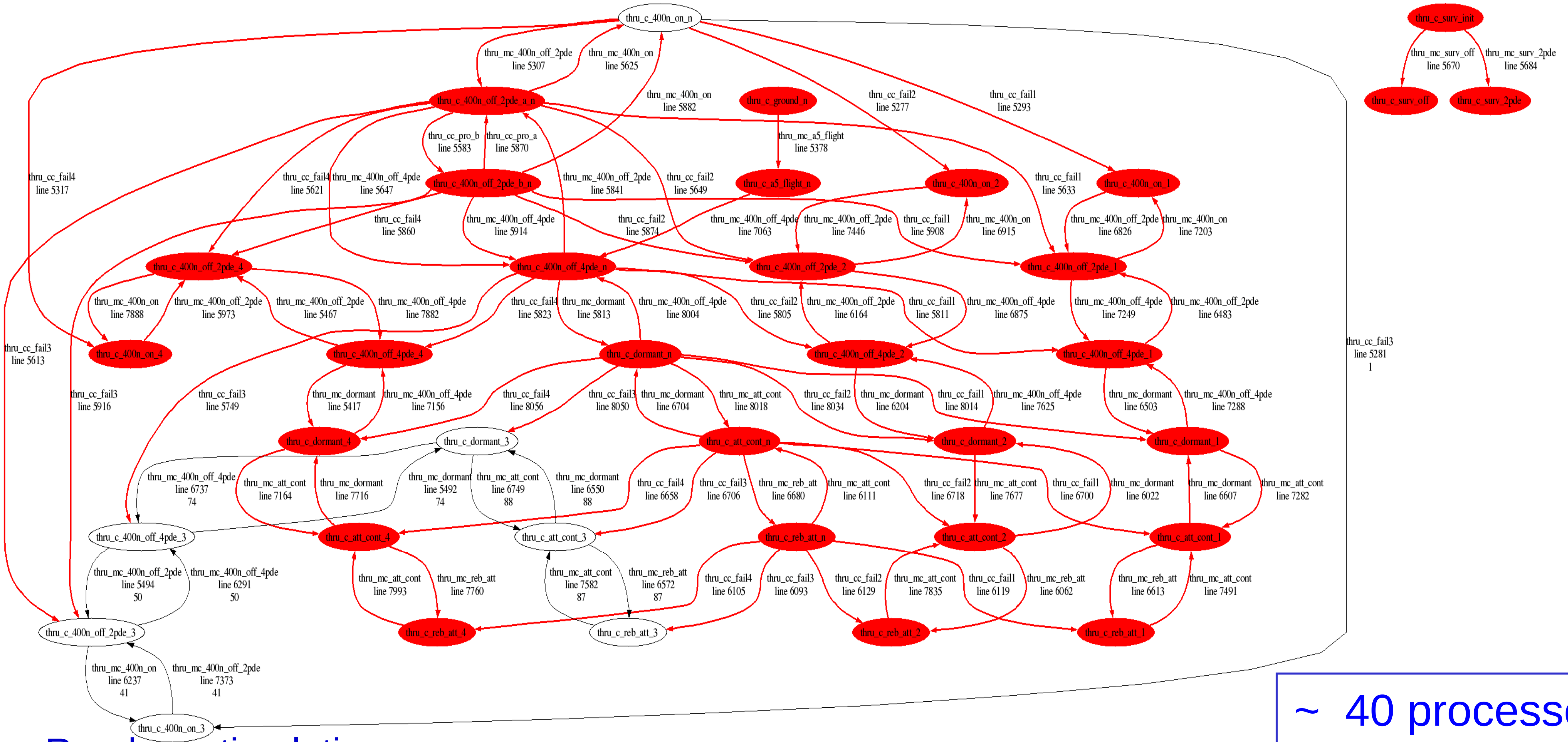
Missing requirement on WCET or Uplink Rate

8b NAK if cmd not accepted

Potential Loss of Commands

Missing performance requirement on uplink rate or WCET

Extracting a Model from Ada code: Coverage of Finite State Machines


thru_c_400n_on_n
thru_mc_400n_off_2pde line 5307
thru_mc_400n_on line 5625
thru_c_400n_off_2pde_a_n
thru_mc_400n_on line 5882
thru_c_ground_n
thru_cc_fail2 line 5277
thru_cc_fail1 line 5293
thru_cc_pro_b line 5583
thru_cc_pro_a line 5870
thru_mc_a5_flight line 5378
thru_cc_fail4 line 5317
thru_cc_fail4 line 5621
thru_mc_400n_off_4pde line 5647
thru_c_400n_off_2pde_b_n
thru_mc_400n_off_2pde line 5841
thru_cc_fail2 line 5649
thru_c_a5_flight_n
thru_c_400n_on_2
thru_cc_fail1 line 5633
thru_c_400n_on_1
thru_cc_fail4 line 5860
thru_mc_400n_off_4pde line 5914
thru_cc_fail2 line 5874
thru_mc_400n_off_4pde line 7063
thru_mc_400n_off_2pde line 7446
thru_cc_fail1 line 5908
thru_mc_400n_on line 6915
thru_mc_400n_off_2pde line 6826
thru_mc_400n_on line 7203
thru_c_400n_off_2pde_4
thru_c_400n_off_4pde_n
thru_c_400n_off_2pde_2
thru_c_400n_off_2pde_1
thru_mc_400n_on line 7888
thru_mc_400n_off_2pde line 5973
thru_mc_400n_off_2pde line 5467
thru_mc_400n_off_4pde line 7882
thru_cc_fail4 line 5823
thru_mc_dormant line 5813
thru_mc_400n_off_4pde line 8004
thru_cc_fail2 line 5805
thru_mc_400n_off_2pde line 6164
thru_cc_fail1 line 5811
thru_mc_400n_off_4pde line 6875
thru_mc_400n_off_4pde line 7249
thru_mc_400n_off_2pde line 6483
thru_cc_fail3 line 5613
thru_c_400n_on_4
thru_c_400n_off_4pde_4
thru_c_dormant_n
thru_c_400n_off_4pde_2
thru_c_400n_off_4pde_1
thru_cc_fail3 line 5281 1
thru_cc_fail3 line 5916
thru_cc_fail3 line 5749
thru_mc_dormant line 5417
thru_mc_400n_off_4pde line 7156
thru_cc_fail4 line 8056
thru_cc_fail3 line 8050
thru_mc_dormant line 6704
thru_mc_att_cont line 8018
thru_cc_fail2 line 8034
thru_mc_dormant line 6204
thru_cc_fail1 line 8014
thru_mc_400n_off_4pde line 7625
thru_mc_dormant line 6503
thru_mc_400n_off_4pde line 7288
thru_c_dormant_4
thru_c_dormant_3
thru_c_att_cont_n
thru_c_dormant_2
thru_c_dormant_1
thru_mc_400n_off_4pde line 6737 74
thru_mc_att_cont line 7164
thru_mc_dormant line 7716
thru_mc_dormant line 5492 74
thru_mc_att_cont line 6749 88
thru_mc_dormant line 6550 88
thru_cc_fail4 line 6658
thru_cc_fail3 line 6706
thru_mc_reb_att line 6680
thru_mc_att_cont line 6111
thru_cc_fail2 line 6718
thru_mc_att_cont line 7677
thru_cc_fail1 line 6700
thru_mc_dormant line 6022
thru_mc_dormant line 6607
thru_mc_att_cont line 7282
thru_c_400n_off_4pde_3
thru_c_att_cont_4
thru_c_att_cont_3
thru_c_reb_att_n
thru_c_att_cont_2
thru_c_att_cont_1
thru_mc_400n_off_2pde line 5494 50
thru_mc_400n_off_4pde line 6291 50
thru_mc_att_cont line 7993
thru_mc_reb_att line 7760
thru_mc_att_cont line 7582 87
thru_mc_reb_att line 6572 87
thru_cc_fail4 line 6105
thru_cc_fail3 line 6093
thru_cc_fail2 line 6129
thru_mc_att_cont line 7835
thru_cc_fail1 line 6119
thru_mc_reb_att line 6062
thru_mc_reb_att line 6613
thru_mc_att_cont line 7491
thru_c_400n_off_2pde_3
thru_c_reb_att_4
thru_c_reb_att_3
thru_c_reb_att_2
thru_c_reb_att_1
thru_mc_400n_on line 6237 41
thru_mc_400n_off_2pde line 7373 41
thru_c_400n_on_3
thru_c_surv_init
thru_mc_surv_off line 5670
thru_mc_surv_2pde line 5684
thru_c_surv_off
thru_c_surv_2pde

~ 40 processes
~200 states
2 processors

- Random stimulation
- Dynamic stimulation of FSMs, inputs derived from the model
- surprising, because reachability of states was not a test goal:
  Sub-sets (nets) of states, no transition possible
- Fault? in this case: hidden information ⇨ testability?
- Conclusion: difficult (*impossible*) verification of application regarding behaviour

Performance Characteristics

- Ada
  - ~1 Mio. lines of source code, ~430 KLOC
- FSM

38	FSMs / processes	
616	different commands	(inputs of FSMs, stimuli)
637	commands in total,	tuples of (FSM,cmd)
360	different states	
381	states in total	tuples of (FSM,state)
1475	transitions	(names)
4695	different tuples	(FSM, msg, initial state, final state)
9778	atomic actions in FSMs	

- Time statistics
  - ISGL model generation from Ada: < 5s
  - system code generation: ~10min
  - stimulation: 2000% coverage of input domain (20x at least) ~70 min. (~ 3cmds/s)

Evaluation of non-functional properties: Distributed Synchronous System

Synthesis of two models

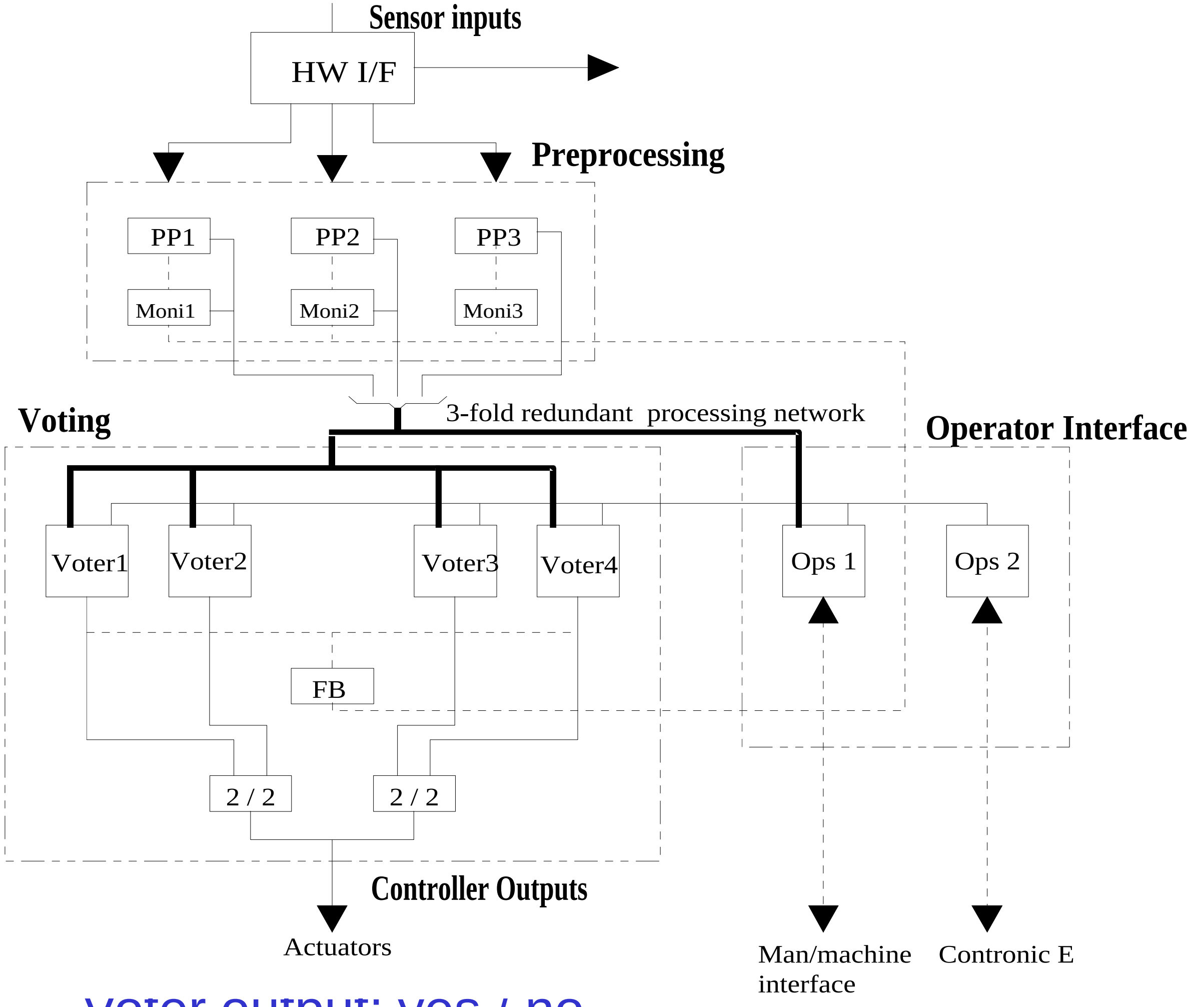
ISGL for behaviour and real-time
Scade/Lustre for control algorithms

Integration on C code level

Stimulation from behavioural model
Stimulation data provided from Scade analysis

Monitoring

Redundancy



Cyclic Data Acqusition

Pre-processing

Filtering

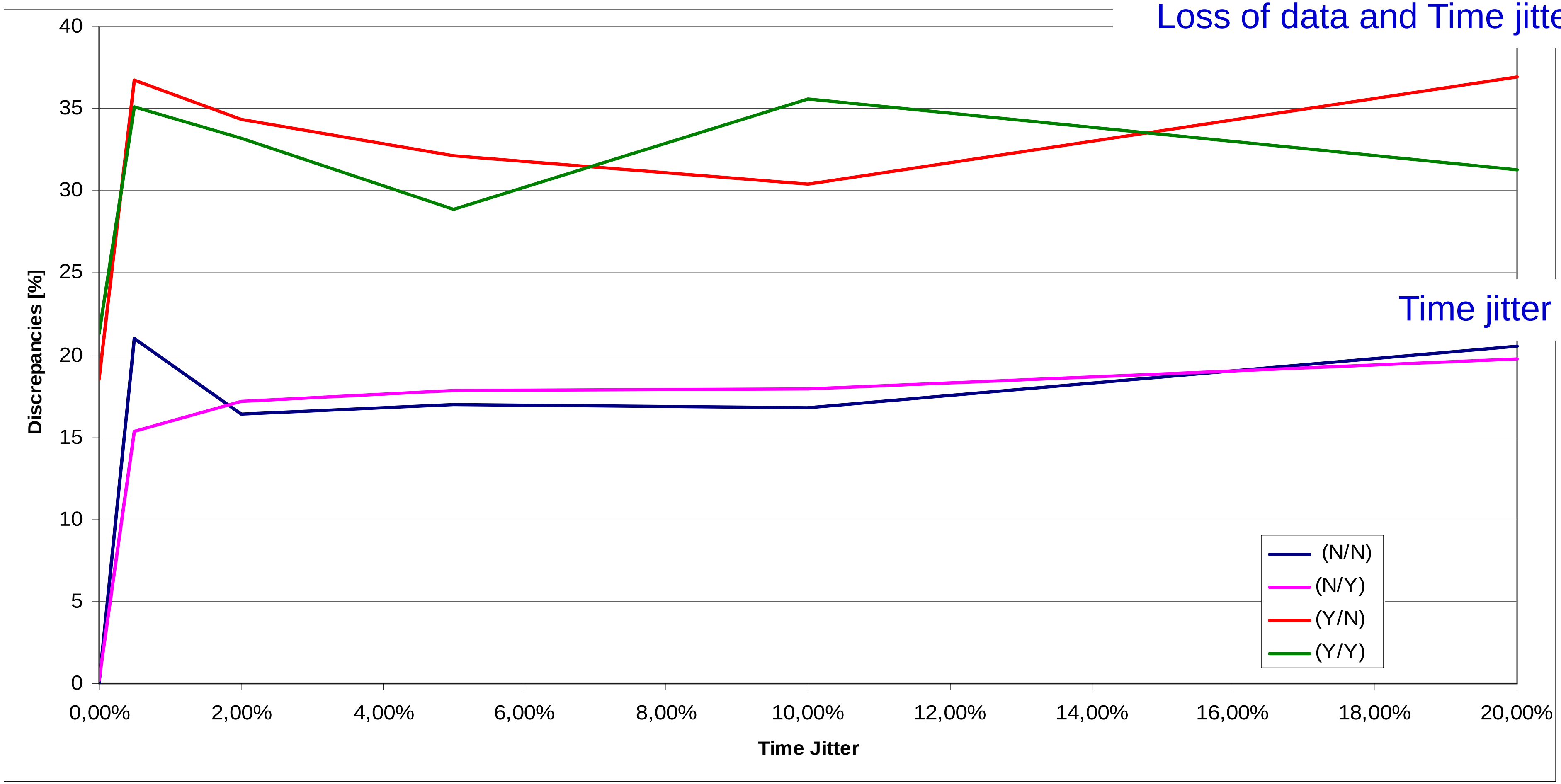
Voting

voter output: yes / no

Impact of Time Jitter and Data Loss

high fault rate at rather low time jitter and/or low rate of loss of data

↑ % faults / voter discrepancies



Loss of data and Time jitter

Time jitter only

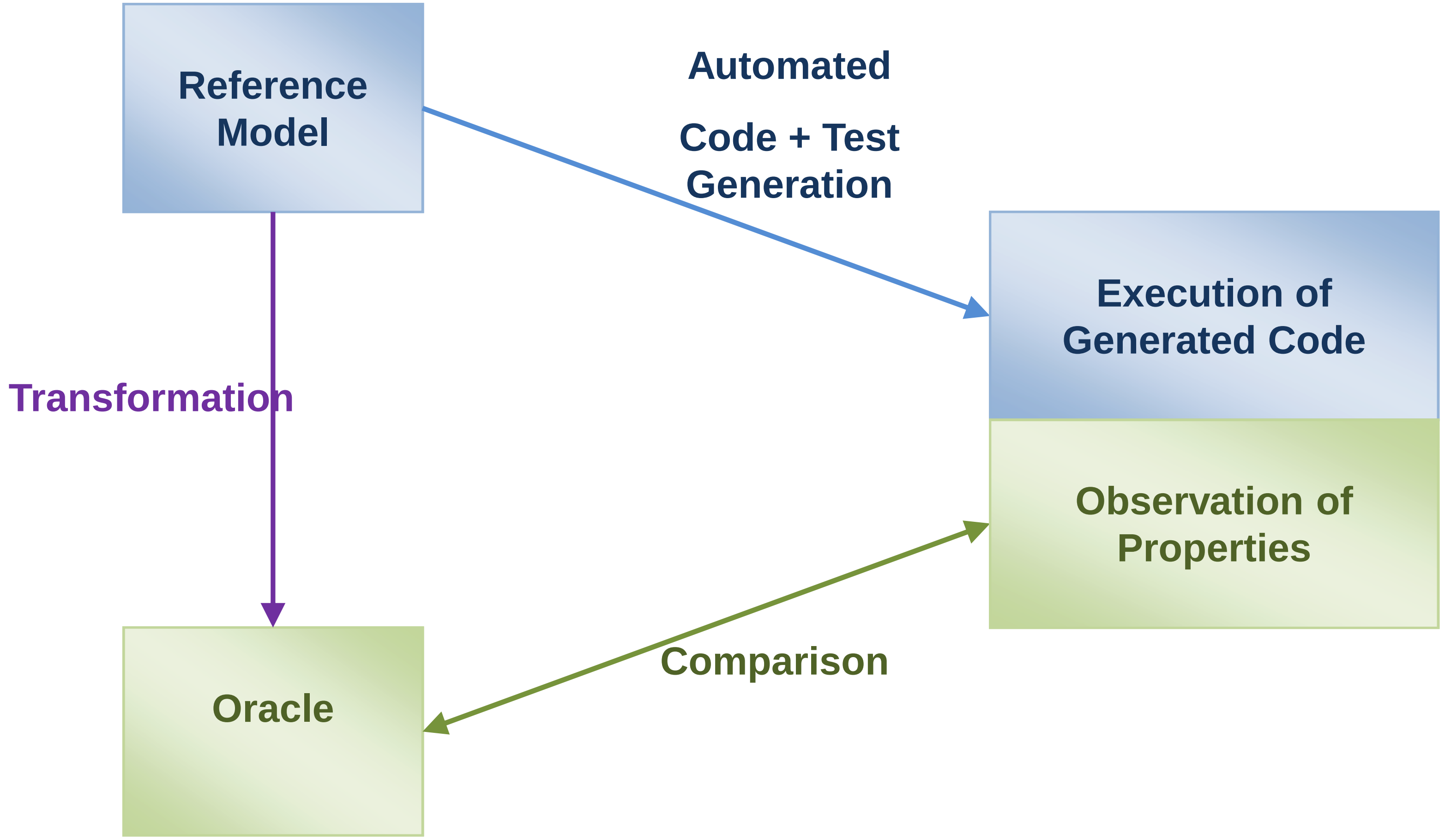
→ % Time jitter

Theoretical prediction and “confirmation” after raising doubts, but before ISG V&V:

“should be robust in case of time jitter”

Verification of the Code Generator

Tests automatically derived from a model
require the generated code
to unveil its properties



Reference Model

Automated

Code + Test Generation

Execution of Generated Code

Transformation

Observation of Properties

Comparison

Oracle