The COMPASTA Approach for MBSE

Alberto Bombardelli, Marco Bozzano, Roberto Cavada, Alessandro Cimatti,
Alberto Griggio, Massimo Nazaria, Edoardo Nicolodi, Stefano Tonetta

Fondazione Bruno Kessler, Trento Italy

MBSE 2022

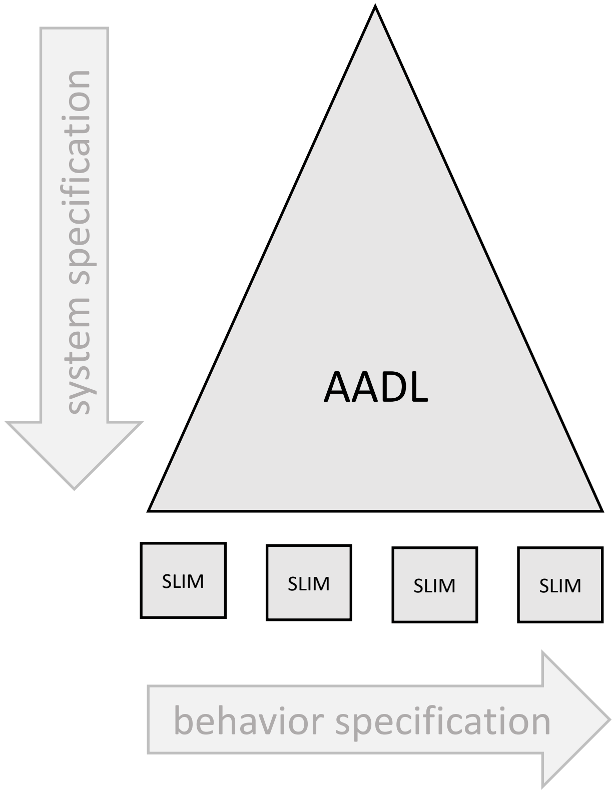
The COMPASTA Study

- The Study at a glance
  - Acronym: COMPASTA
  - Early Technology Development, funded by the European Space Agency
  - Contractor: Fondazione Bruno Kessler (FBK), Trento, Italy
  - Duration: 18 months, ending in December 2022

- FBK
  - Research Foundation (over 400 researchers)
  - Embedded Systems Unit: about 30 people

Background: the COMPASS tool

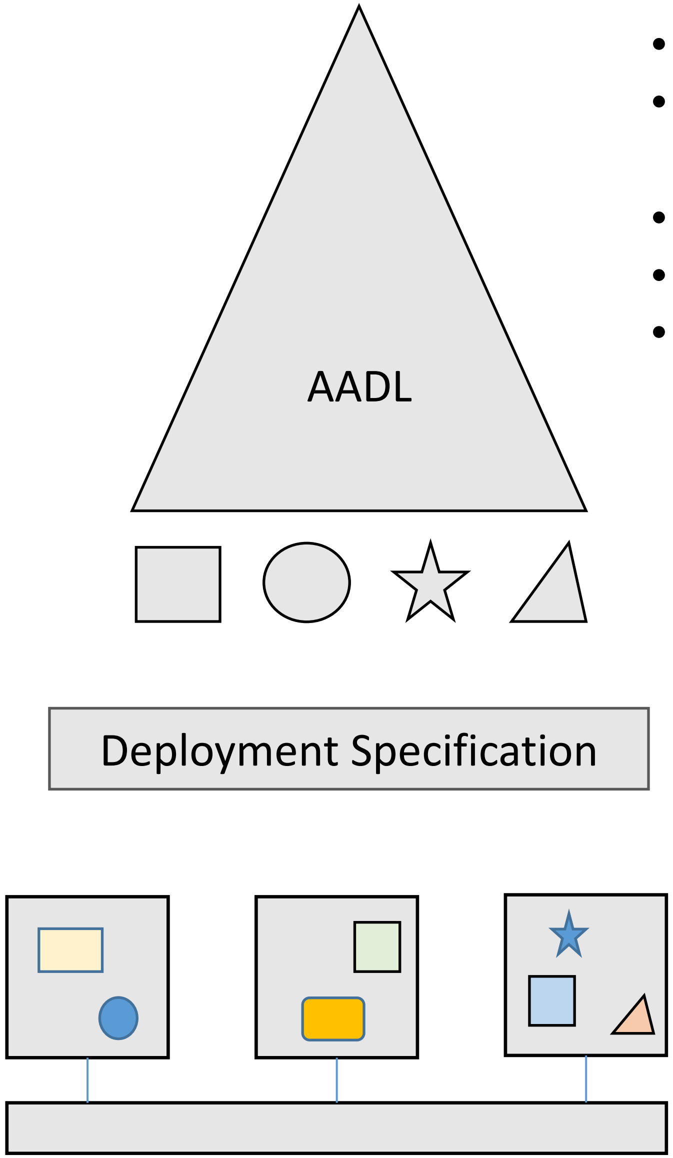
- Tool for model-based system/SW co-engineering
- Developed in a series of ESA studies (2008-2016)
- Input language is a variant of AADL (called SLIM)
- Functionality: formal design, formal V&V
- Based on model checking



- Requirements specification
- Requirements analysis
- Contract-based design
- Functional verification
- Fault injection
- RAMS Analyses: FTA, FMEA
- FDIR Analysis

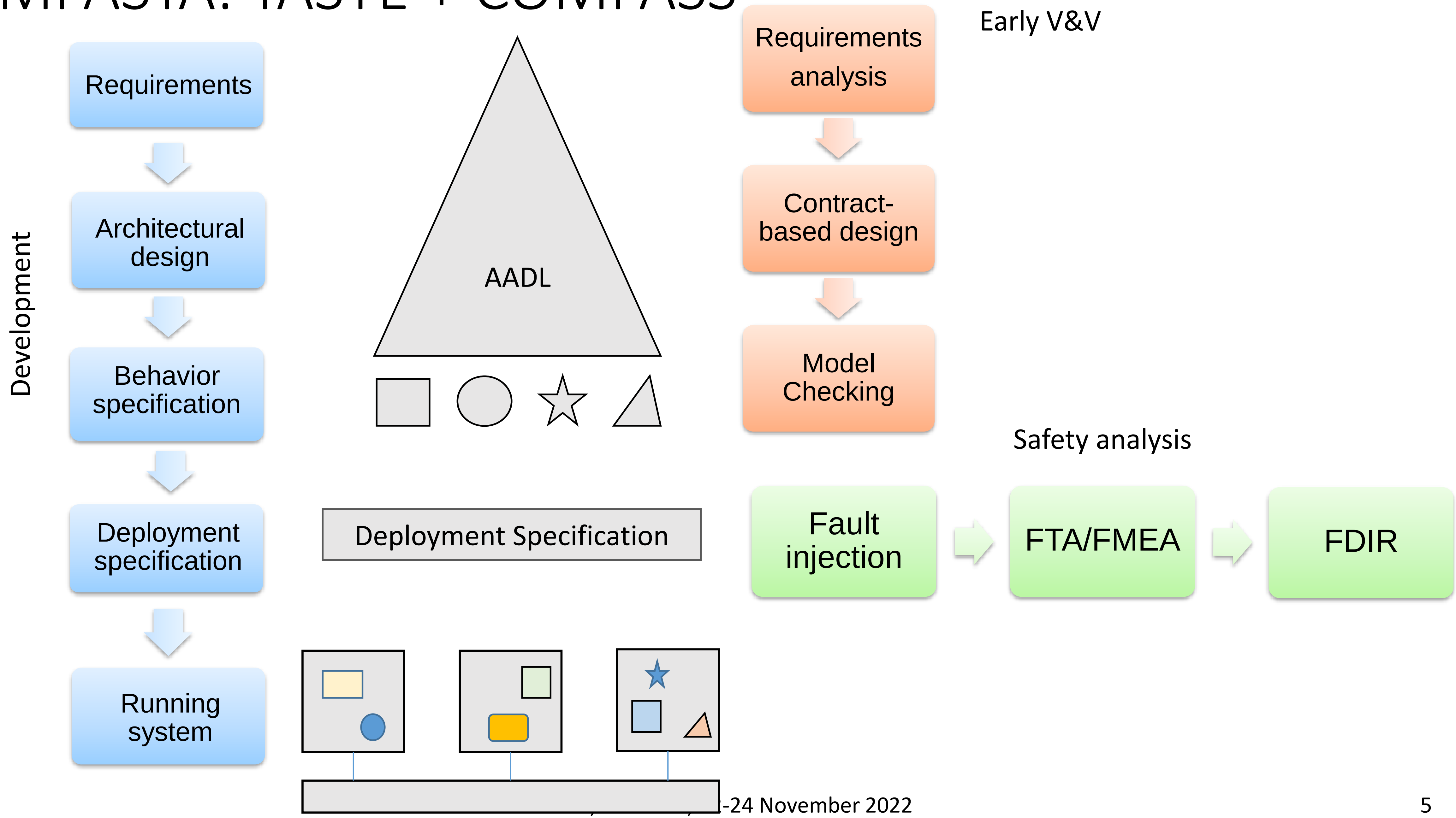
Background: the TASTE tool

- Tool for model-based design of embedded, real- time systems
- Created by initiative of ESA in 2008
- Several modeling languages
  - ASN.1, AADL, SDL, Simulink, etc.
- Ecosystem: graphical editors, visualizers, code generators



- Many languages
- Push-button compilers for deployment
- Graphical editor for AADL
- Graphical editor for SDL
- High integrability

COMPASTA: TASTE + COMPASS



Development

Requirements

Architectural design

Behavior specification

Deployment specification

Running system

AADL

Deployment Specification

Early V&V

Requirements analysis

Contract-based design

Model Checking

Safety analysis

Fault injection

FTA/FMEA

FDIR

Objectives of COMPASTA

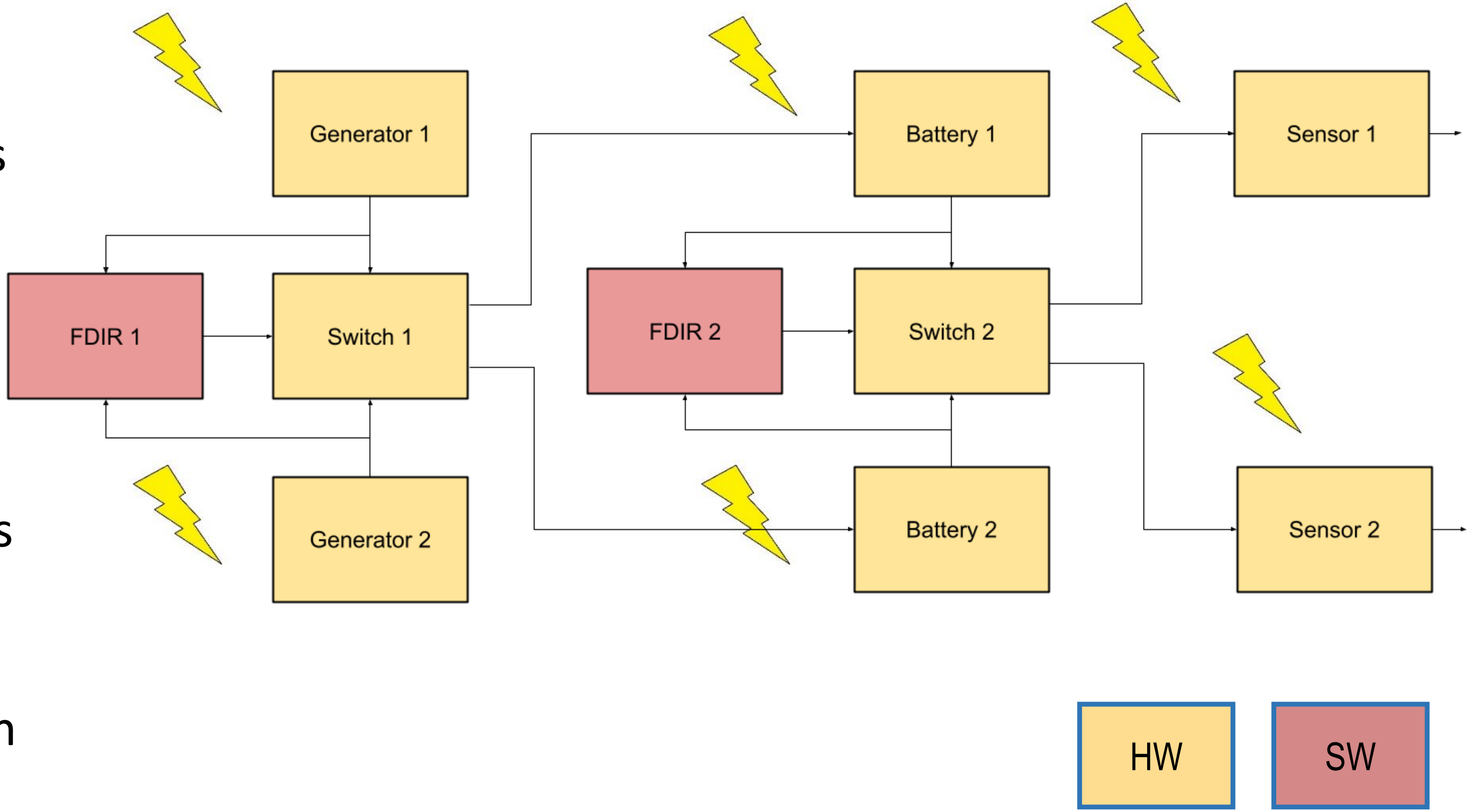
- Integration of the existing COMPASS and TASTE toolchains
- Goal: **a comprehensive, end-to-end toolchain that covers system development, early verification and validation, safety assessment and FDIR, system deployment**
  - COMPASS used to build and validate a formal model of the system (HW+SW) architecture, to specify the behavior of the HW components and their faults
  - TASTE used to model the behavior of the SW components, for deployment and code generation, and to test the final implementation
- Goal: foster the adoption and the industrial exploitation of the COMPASS+TASTE integrated toolchain

TASTE+COMPASS: Technical approach

- Specification using **AADL** and **SDL** as specification languages
  - AADL for interface view and HW components
  - SDL for SW components
- Definition of the semantics of the composition of HW and SW
- Translation of AADL/SDL into the languages of the COMPASS back-ends
- Integration of COMPASS back-ends for V&V into TASTE
- Automated formal analyses using the back-ends
- Extended editors and visualizers

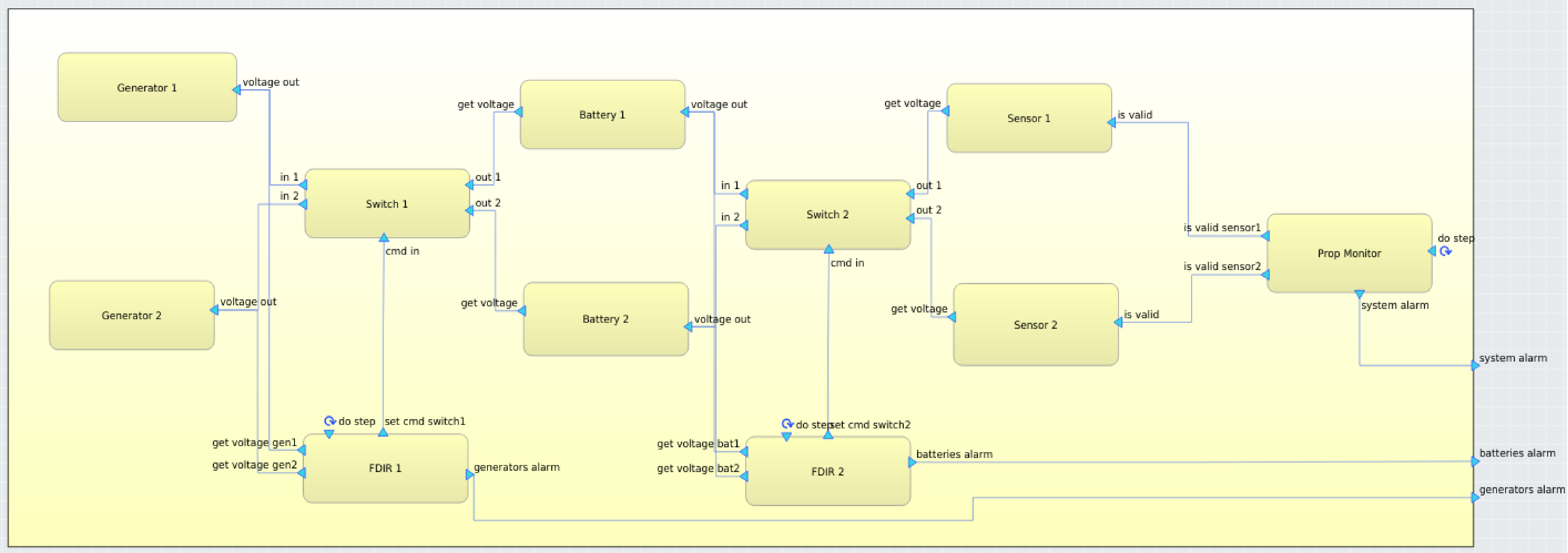
Workflow: An Example

- Redundant power system
  - Generators charging batteries
  - Batteries powering sensors
  - Redundant lines connecting generators, batteries and sensors
- HW faults
  - Generators, batteries, sensors
- FDIR components
  - They control switches and command re-configurations in case of faults
- Requirements
  - Sensors must be powered



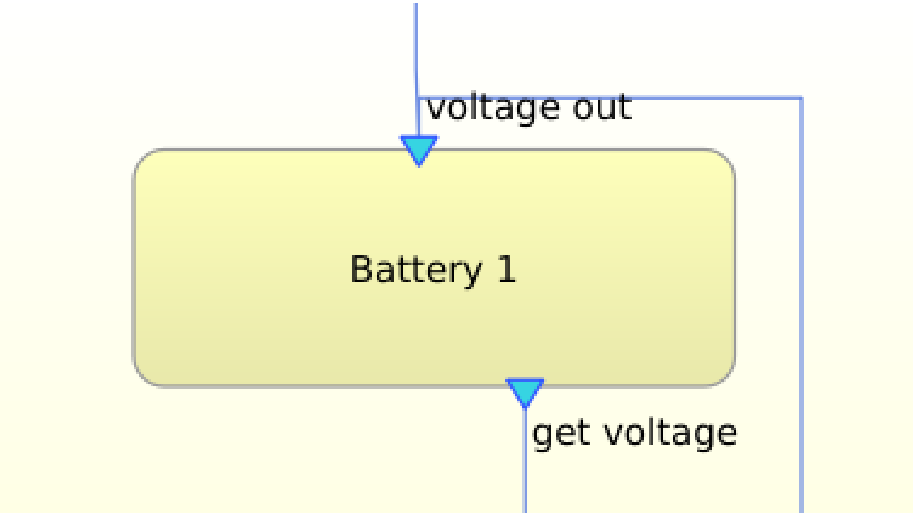
Workflow: Modeling the system architecture

- Using TASTE interface view (graphical editor + serialization in AADL)



Workflow: Modeling the behavior of HW components

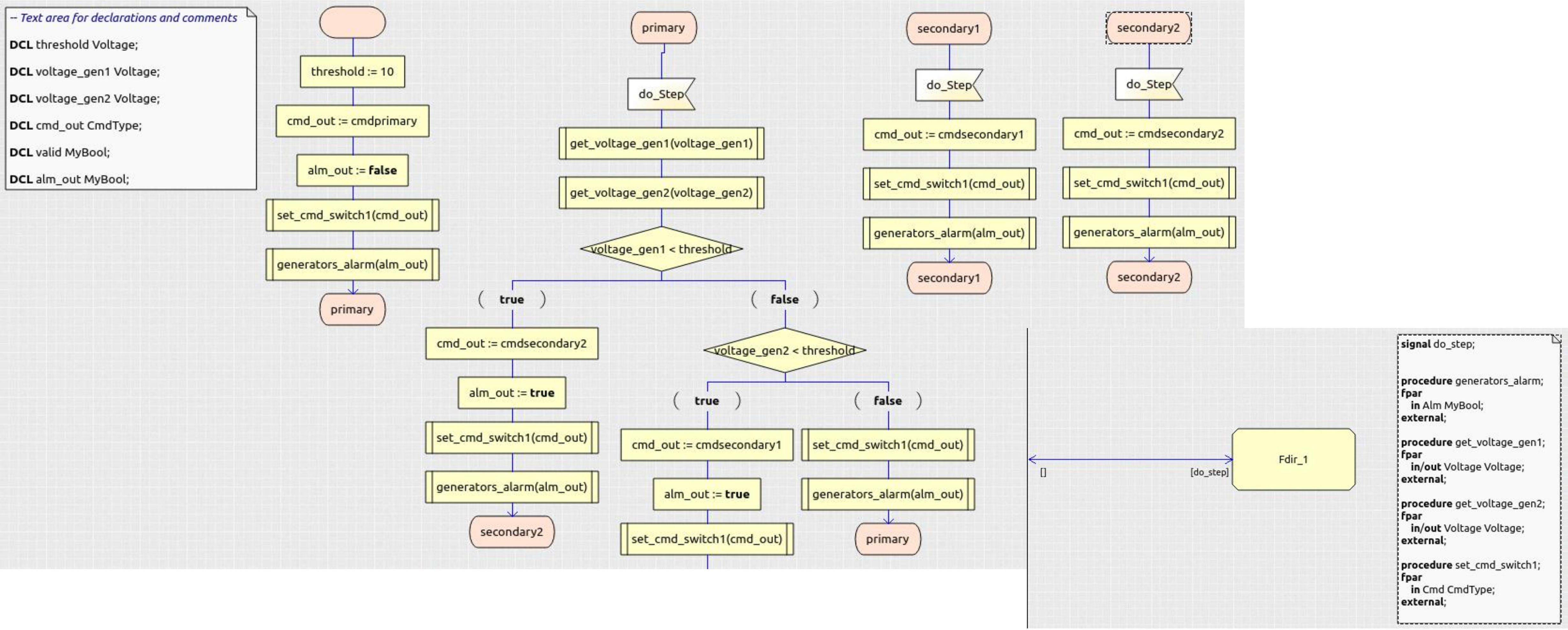
- Modeling the behavior of HW components in SLIM (an extension of AADL)



```
system implementation Battery_1.Imp
subcomponents
 delay : data clock;
states
   base: initial state while (delay <= 1);
transitions
   base -[when
                delay >= 1 and get_voltage.voltage < 10 and
                voltage_out.voltage >= 1
              then
                delay :=0 and
                voltage_out.voltage := voltage_out.voltage – 1
             ]-> base;
   …
end Battery_1.Imp;
```

Workflow: Modeling the behavior of SW components

- Modeling the behavior of SW components in SDL

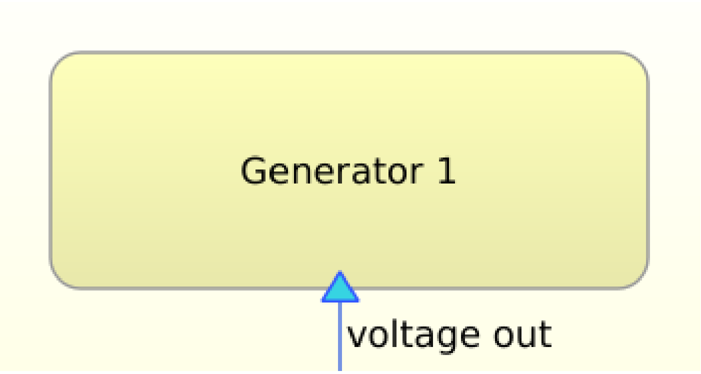


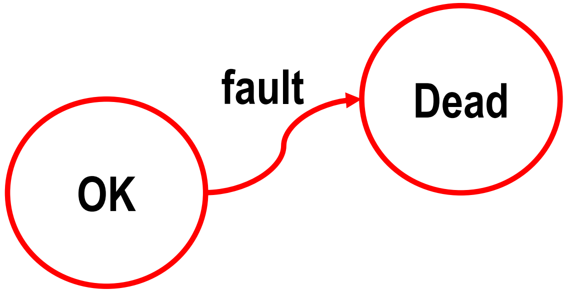
```
-- Text area for declarations and comments
DCL threshold Voltage;
DCL voltage_gen1 Voltage;
DCL voltage_gen2 Voltage;
DCL cmd_out CmdType;
DCL valid MyBool;
DCL alm_out MyBool;
```

```
signal do_step;

procedure generators_alarm;
fpar
  in Alm MyBool;
external;

procedure get_voltage_gen1;
fpar
  in/out Voltage Voltage;
external;

procedure get_voltage_gen2;
fpar
  in/out Voltage Voltage;
external;

procedure set_cmd_switch1;
fpar
  in Cmd CmdType;
external;
```

Workflow: Modeling faults

- Specifying a fault injection (generator output voltage stuck-at-zero)





```
system implementation Generator_1.others
properties
 FaultInjections => (
   [
     Description => Dead;
     Fault_Model => StuckAtByValue_I;
     Fault_Dynamics => Permanent;
     Probability => 1.e-7;
     DataInput => voltage_out.voltage;
     DataVarout => voltage_out.voltage;
     DataTerm => 0;
   ]
 );
end Generator_1.others;
```

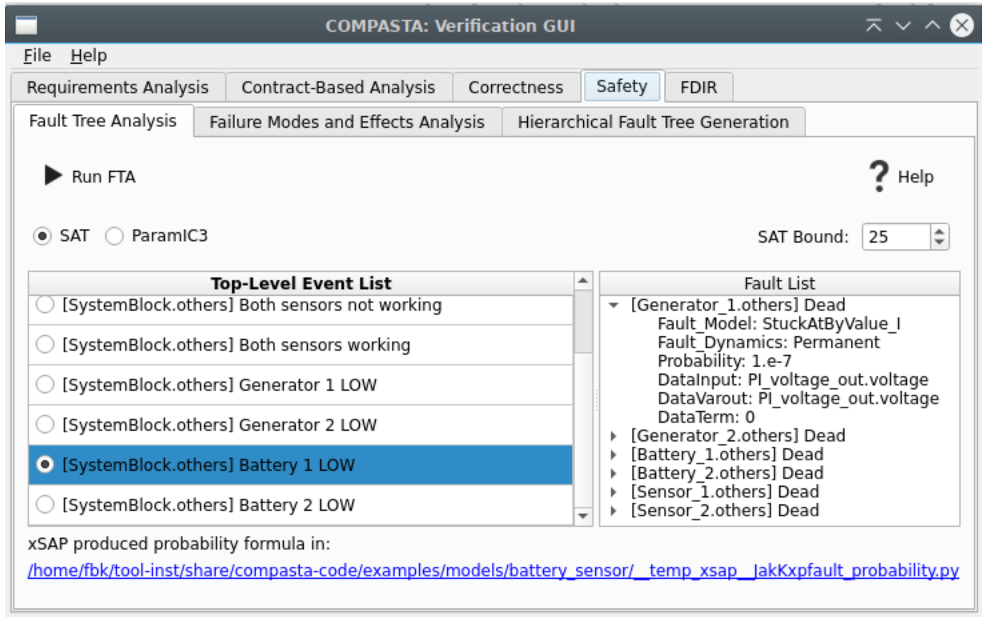
Workflow: Specification of properties and contracts

- Pattern-based modeling of properties and contracts

Name	Property
All sensors working	Globally, it is always the case that {Sensor_1.is_valid.valid and Sensor_2.is_valid.valid} holds
At least one sensor working	Globally, it is always the case that {Sensor_1.is_valid.valid or Sensor_2.is_valid.valid} holds

Component	Name	Assumption	Guarantee
Generator	power	true	always(voltage_out.voltage >= 10)
Battery	power	always(get_voltage.voltage >= 10)	always(voltage_out.voltage >= 10)
Switch	routing	true	always(cmd_in.cmd=cmdprimary -> (out_1.voltage=in_1.voltage and out_2.voltage=in_2.voltage) …)
Sensor	Power	always(get_voltage.voltage >= 10)	always(is_valid.valid)
…	…	…	…

Workflow: Formal Verification



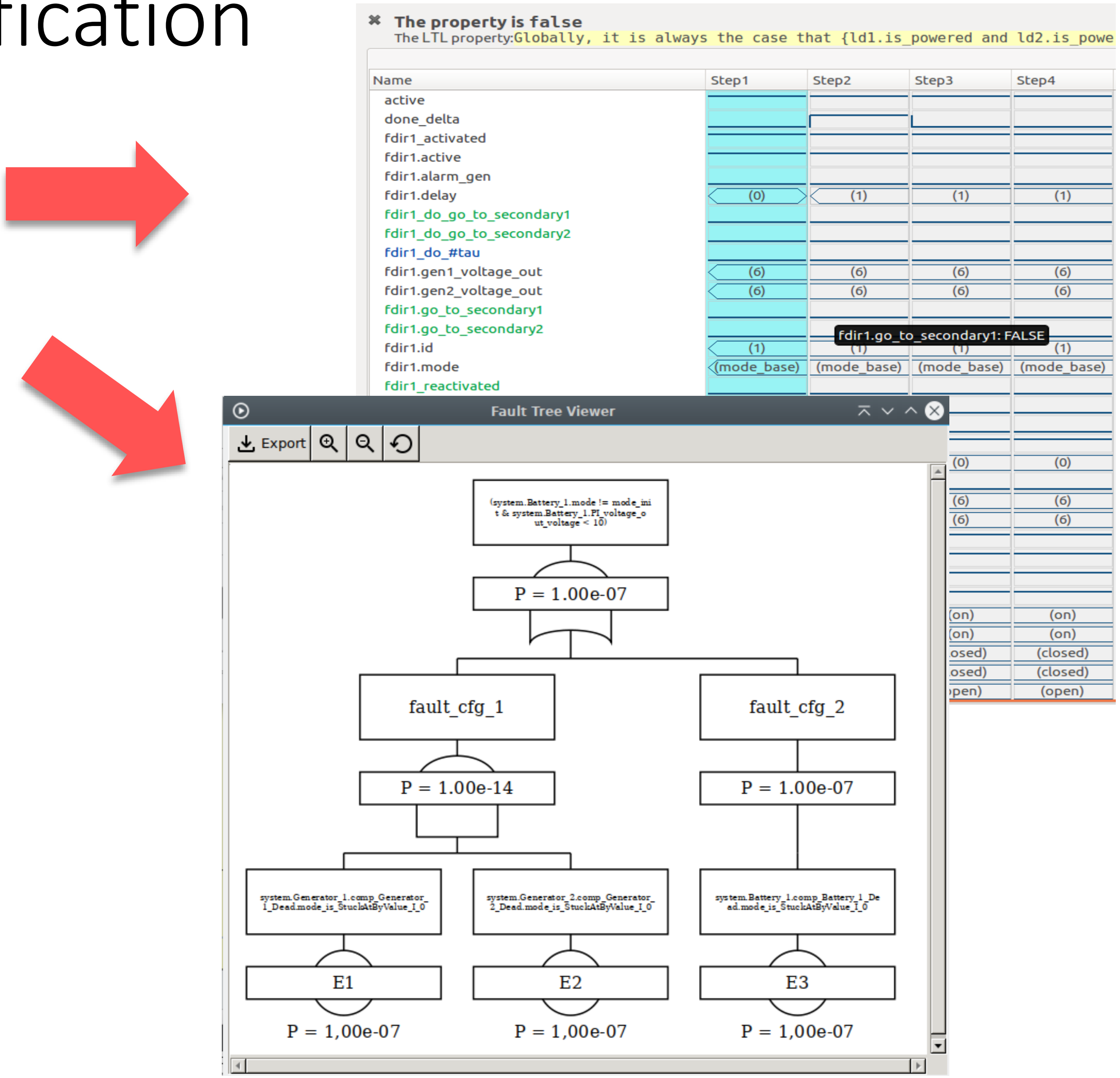
COMPASTA: Verification GUI
File Help
Requirements Analysis | Contract-Based Analysis | Correctness | Safety | FDIR
Fault Tree Analysis | Failure Modes and Effects Analysis | Hierarchical Fault Tree Generation
Run FTA
Help

● SAT ○ ParamIC3 SAT Bound: 25

Top-Level Event List
○ [SystemBlock.others] Both sensors not working
○ [SystemBlock.others] Both sensors working
○ [SystemBlock.others] Generator 1 LOW
○ [SystemBlock.others] Generator 2 LOW
● [SystemBlock.others] Battery 1 LOW
○ [SystemBlock.others] Battery 2 LOW

Fault List
- [Generator_1.others] Dead
  - Fault_Model: StuckAtByValue_I
  - Fault_Dynamics: Permanent
  - Probability: 1.e-7
  - DataInput: PI_voltage_out.voltage
  - DataVarout: PI_voltage_out.voltage
  - DataTerm: 0
- [Generator_2.others] Dead
- [Battery_1.others] Dead
- [Battery_2.others] Dead
- [Sensor_1.others] Dead
- [Sensor_2.others] Dead

xSAP produced probability formula in:
/home/fbk/tool-inst/share/compasta-code/examples/models/battery_sensor/__temp_xsap__JakKxpfault_probability.py



✖ **The property is false**
The LTL property: Globally, it is always the case that {ld1.is_powered and ld2.is_powe

Name	Step1	Step2	Step3	Step4
active				
done_delta				
fdir1_activated				
fdir1.active				
fdir1.alarm_gen				
fdir1.delay	(0)	(1)	(1)	(1)
fdir1_do_go_to_secondary1				
fdir1_do_go_to_secondary2				
fdir1_do_#tau				
fdir1.gen1_voltage_out	(6)	(6)	(6)	(6)
fdir1.gen2_voltage_out	(6)	(6)	(6)	(6)
fdir1.go_to_secondary1				
fdir1.go_to_secondary2				
fdir1.id	(1)	(1)	(1)	(1)
fdir1.mode	(mode_base)	(mode_base)	(mode_base)	(mode_base)
fdir1_reactivated				

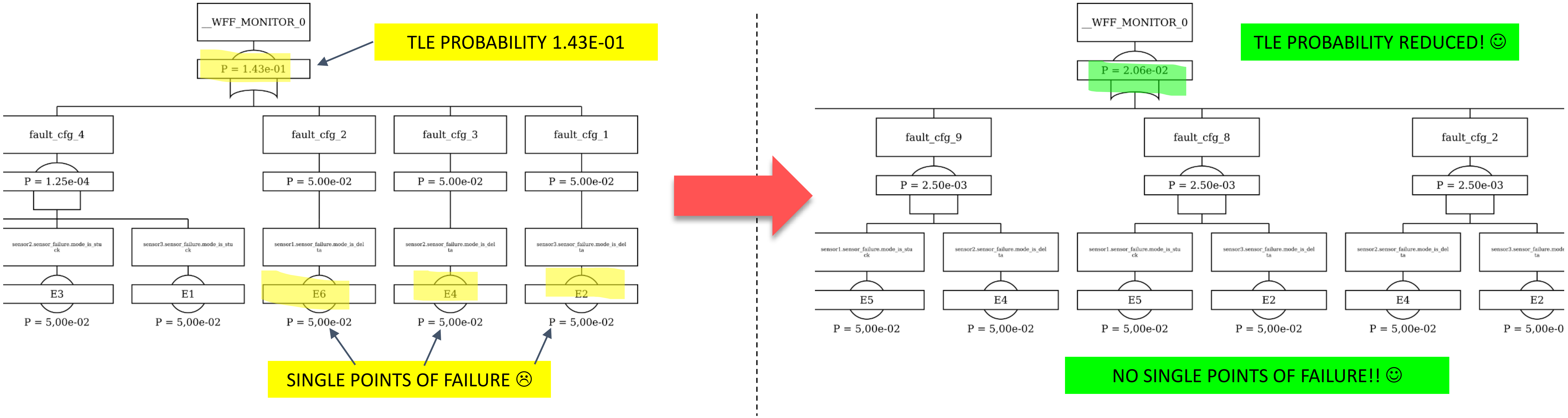
fdir1.go_to_secondary1: FALSE

(0)
(0)
(6)
(6)
(6)
(6)
(on)
(on)
(on)
(on)
osed)
(closed)
osed)
(closed)
pen)
(open)
Fault Tree Viewer
Export
(system.Battery_1.mode != mode_init & system.Battery_1.PI_voltage_out_voltage < 10)
P = 1.00e-07
fault_cfg_1
fault_cfg_2
P = 1.00e-14
P = 1.00e-07
system.Generator_1.comp_Generator_1_Dead.mode_is_StuckAtByValue_I_0
system.Generator_2.comp_Generator_2_Dead.mode_is_StuckAtByValue_I_0
system.Battery_1.comp_Battery_1_Dead.mode_is_StuckAtByValue_I_0
E1
E2
E3
P = 1,00e-07
P = 1,00e-07
P = 1,00e-07

- Formal verification
  - Functional verification
  - Dependability and safety assessment (FTA/ FMEA)
  - FDIR analysis
- Use the COMPASS back-ends to generate the results

Workflow: Iterations modeling/formal verification

- Iterate over previous steps e.g.
  - I. Refine/modify architecture and components
  - II. Refine/modify faults, properties and contracts
  - III. Re-run verification and analyze the new results


__WFF_MONITOR_0
P = 1.43e-01
fault_cfg_4
P = 1.25e-04
fault_cfg_2
P = 5.00e-02
fault_cfg_3
P = 5.00e-02
fault_cfg_1
P = 5.00e-02
sensor2.sensor_failure.mode_is_stuck
sensor3.sensor_failure.mode_is_stuck
sensor1.sensor_failure.mode_is_delta
sensor2.sensor_failure.mode_is_delta
sensor3.sensor_failure.mode_is_delta
E3
P = 5,00e-02
E1
P = 5,00e-02
E6
P = 5,00e-02
E4
P = 5,00e-02
E2
P = 5,00e-02
__WFF_MONITOR_0
P = 2.06e-02
fault_cfg_9
P = 2.50e-03
fault_cfg_8
P = 2.50e-03
fault_cfg_2
P = 2.50e-03
sensor1.sensor_failure.mode_is_stuck
sensor2.sensor_failure.mode_is_delta
sensor1.sensor_failure.mode_is_stuck
sensor3.sensor_failure.mode_is_delta
sensor2.sensor_failure.mode_is_delta
sensor3.sensor_failure.mode
ta
E5
P = 5,00e-02
E4
P = 5,00e-02
E5
P = 5,00e-02
E2
P = 5,00e-02
E4
P = 5,00e-02
E2
P = 5,00e-0

TLE PROBABILITY 1.43E-01

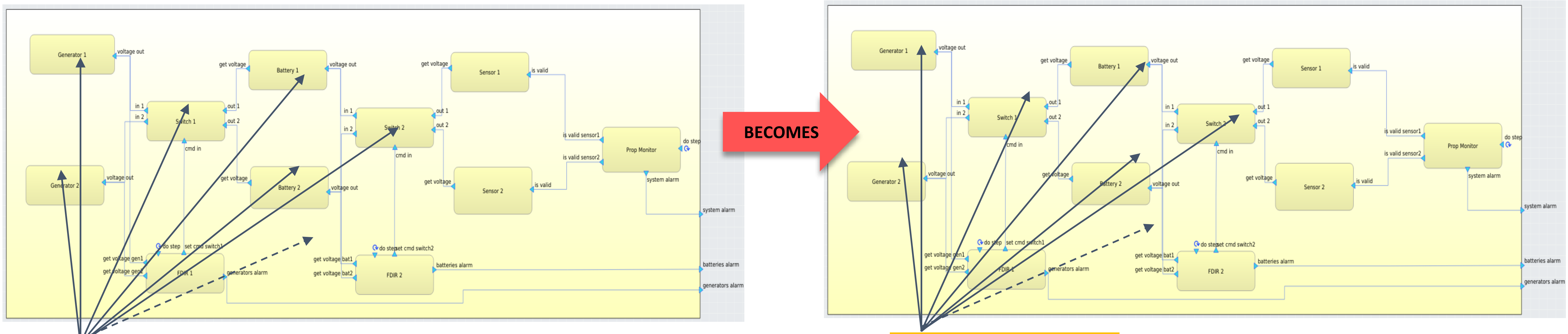
SINGLE POINTS OF FAILURE ☹

TLE PROBABILITY REDUCED! ☺

NO SINGLE POINTS OF FAILURE!! ☺

Workflow: Compilation-ready model transformation

- Replace HW components with “HW_I/O”
  - HW_IO components represent the SW interface layer between SW and HW
  - The resulting model is a native TASTE model

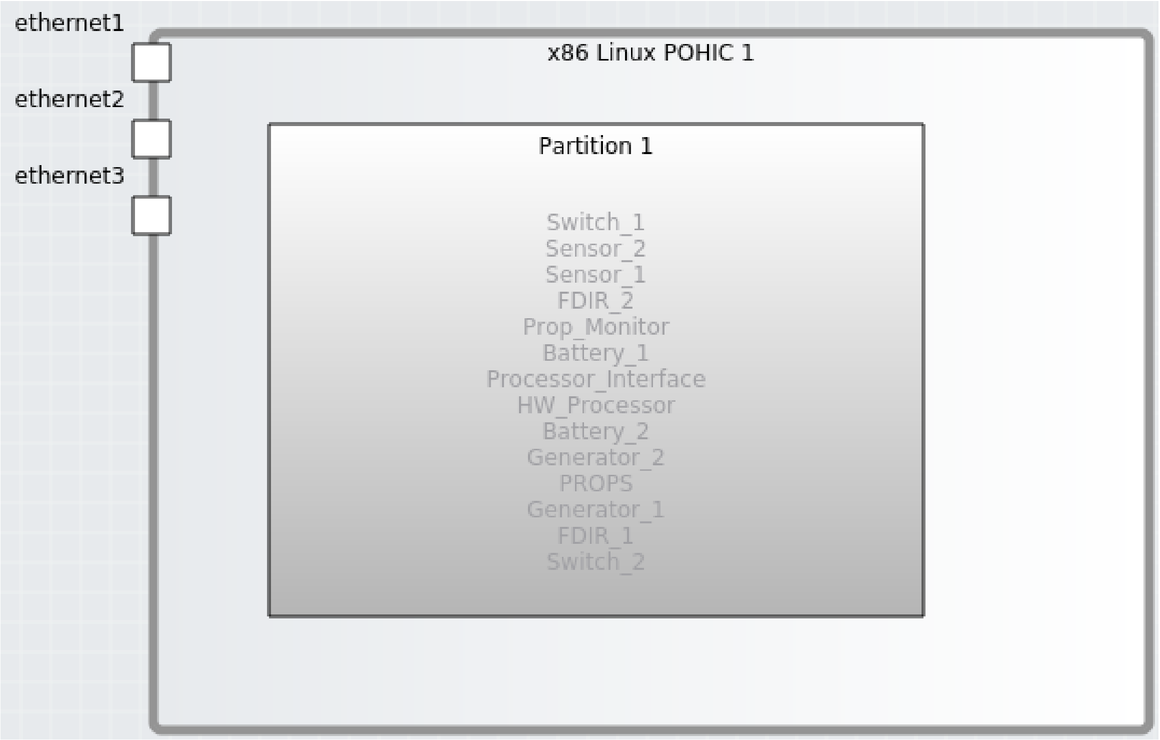


HW components

HW_I/O components (SW component)

Workflow: Deployment and code generation

- Deploy the system on the final HW, and generate code using TASTE

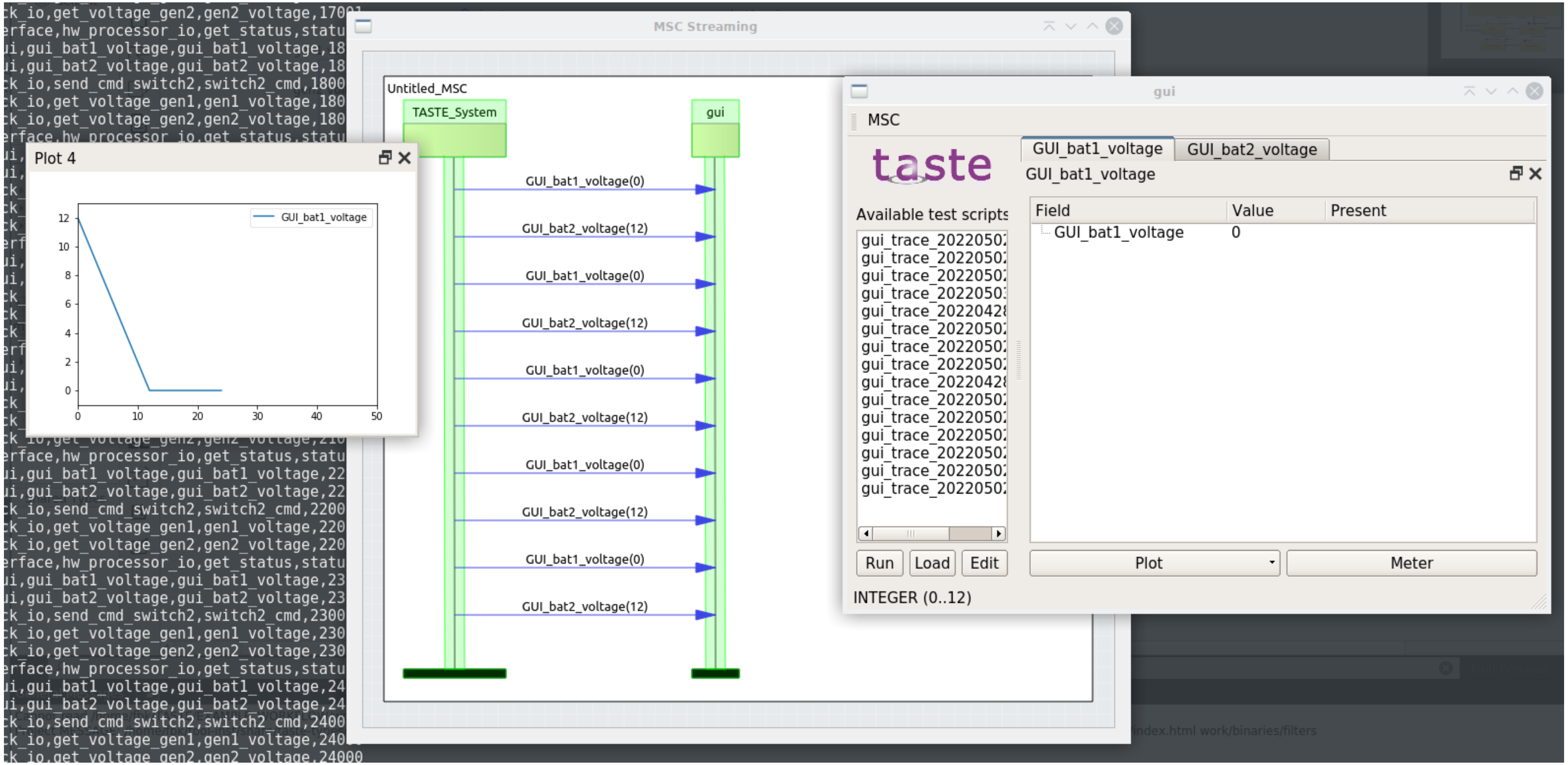


demo

Executable program

Workflow: Testing and Simulation

- Test the final implementation



```
ck_io,get_voltage_gen2,gen2_voltage,17001
erface,hw_processor_io,get_status,statu
ui,gui_bat1_voltage,gui_bat1_voltage,18
ui,gui_bat2_voltage,gui_bat2_voltage,18
ck_io,send_cmd_switch2,switch2_cmd,1800
ck_io,get_voltage_gen1,gen1_voltage,180
ck_io,get_voltage_gen2,gen2_voltage,180
erface,hw_processor_io,get_status,statu
ui,
ui,
ck_
ck_
ck_
erf
ui,
ui,
ck_
ck_
ck_
erf
ui,
ui,
ck_
ck_
ck_io,get_voltage_gen2,gen2_voltage,210
erface,hw_processor_io,get_status,statu
ui,gui_bat1_voltage,gui_bat1_voltage,22
ui,gui_bat2_voltage,gui_bat2_voltage,22
ck_io,send_cmd_switch2,switch2_cmd,2200
ck_io,get_voltage_gen1,gen1_voltage,220
ck_io,get_voltage_gen2,gen2_voltage,220
erface,hw_processor_io,get_status,statu
ui,gui_bat1_voltage,gui_bat1_voltage,23
ui,gui_bat2_voltage,gui_bat2_voltage,23
ck_io,send_cmd_switch2,switch2_cmd,2300
ck_io,get_voltage_gen1,gen1_voltage,230
ck_io,get_voltage_gen2,gen2_voltage,230
erface,hw_processor_io,get_status,statu
ui,gui_bat1_voltage,gui_bat1_voltage,24
ui,gui_bat2_voltage,gui_bat2_voltage,24
ck_io,send_cmd_switch2,switch2_cmd,2400
ck_io,get_voltage_gen1,gen1_voltage,240
ck_io,get_voltage_gen2,gen2_voltage,24000
```

Available test scripts

```
gui_trace_2022050
gui_trace_2022050
gui_trace_2022050
gui_trace_2022050
gui_trace_2022042
gui_trace_2022050
gui_trace_2022050
gui_trace_2022050
gui_trace_2022042
gui_trace_2022050
gui_trace_2022050
gui_trace_2022050
gui_trace_2022050
gui_trace_2022050
gui_trace_2022050
```

GUI_bat1_voltage

Field	Value	Present
GUI_bat1_voltage	0	

INTEGER (0..12)

Summary

- COMPASS and TASTE provide complementary functionality
- COMPASS functionality used to:
  - Model the system architecture
  - Model the HW components and their faults
  - Validate a formal model of the system
- TASTE functionality used to:
  - Model the SW components
  - Deployment
  - Code generation
  - Testing of the deployed system

Conclusions

- The goal of COMPASTA is to integrate COMPASS functionality into TASTE, and produce a comprehensive, end-to-end toolchain for system design, formal verification and validation, and deployment
- The Study is ongoing – final delivery in December 2022