COMPASTA: Integration of the COMPASS and TASTE toolsets

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

Fondazione Bruno Kessler, Trento Italy

MBSE 2021

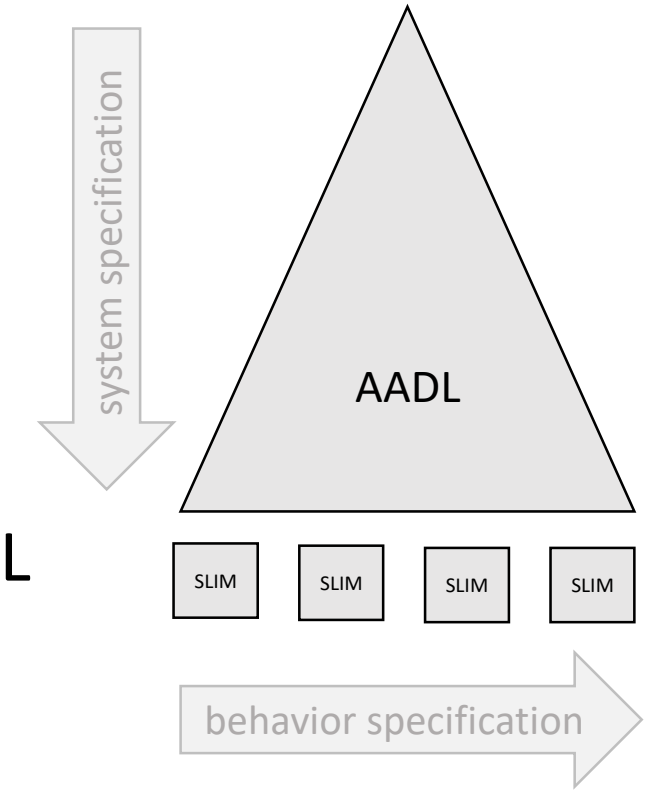
The COMPASTA Study

- The Study at a glance
  - Acronym: COMPASTA
  - Type: Early Technology Development
  - Funded as an idea in OSIP (MBSE Campaign)
  - Contractor: Fondazione Bruno Kessler (FBK), Trento, Italy
  - Duration: 18 months (April 2021 – October 2022)

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

Background: the COMPASS tool

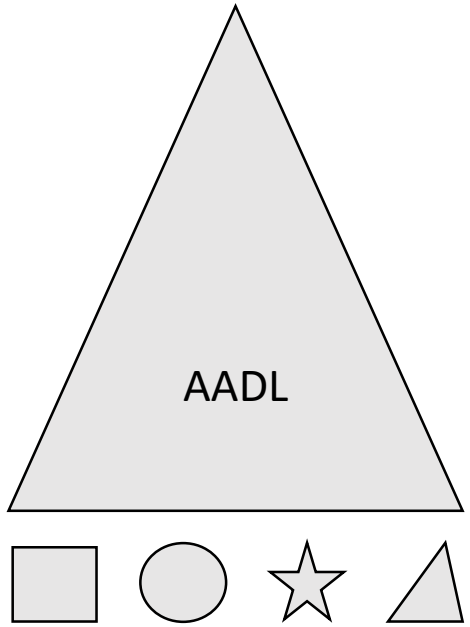
- Tool for model-based system/SW co-engineering
- Developed in a series of ESA studies (2008-2016)
  - Latest release in 2019
- Input language is a variant of AADL (called SLIM)
- Functionality: formal design, formal V&V
- Based on model checking



- Requirements specification
- Requirements analysis
- Contract-based design
- Functional verification
- Fault injection
- Safety assessment and
- dependability: FTA, FMEA
- FDIR Analysis

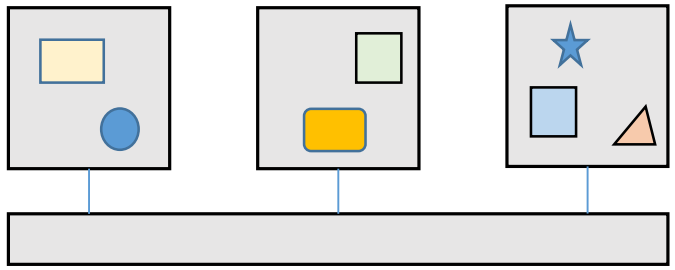
Background: the TASTE tool

- 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

Deployment Specification



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 a formal model of the system architecture, the HW components and their faults, and to validate the formal model
  - TASTE used to model the SW components, for code generation and deployment, and to test the final implementation
- Goal: foster the adoption and the industrial exploitation of the COMPASS+TASTE integrated toolchain

TASTE+COMPASS: Technical approach

- Definition of common input languages for model-based specifications
  - AADL, SDL
- Integration of COMPASS back-ends for V&V into TASTE
  - COMPASS back-ends can be called from TASTE to perform formal V&V
  - Lightweight integration into the TASTE GUI, and script-based interaction
- Encoding of AADL and SDL specifications into the language of the back-ends
- Automated formal analyses using the back-ends
- Extended editors and visualizers

TASTE + COMPASS Workflow

Development

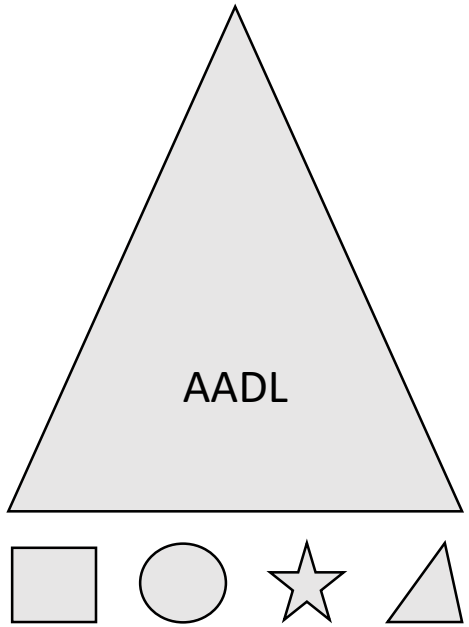
Requirements

Architectural design

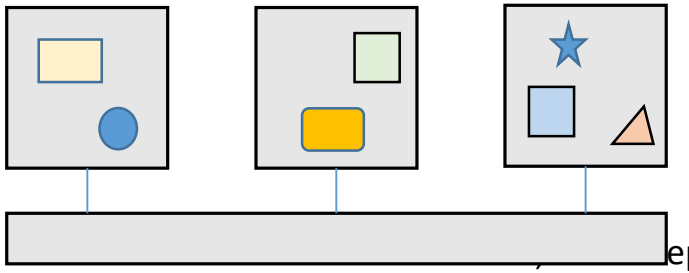
Behavior specification

Deployment specification

Running system



Deployment Specification



Early V&V

Requirements analysis

Contract-based design

Model Checking

Safety analysis

Fault injection

FTA/FMEA

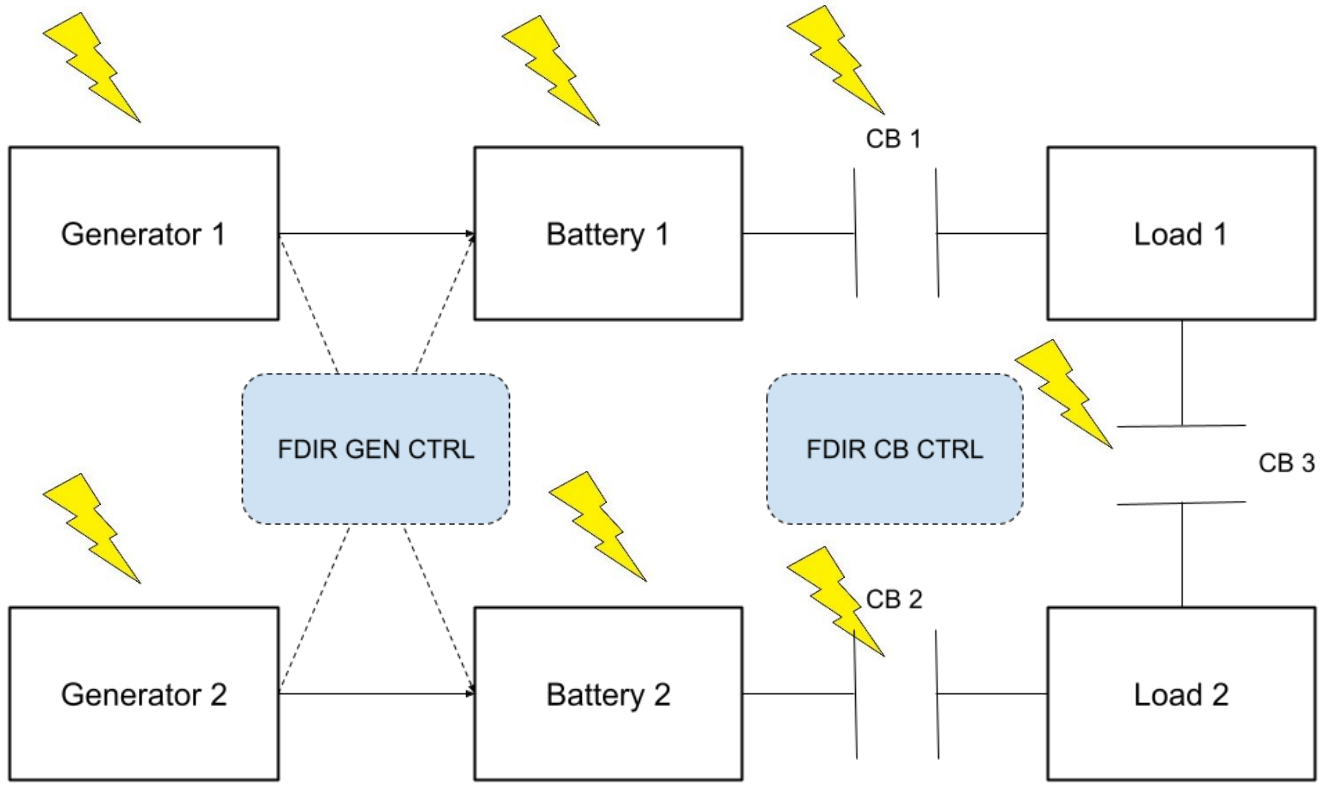
FDIR

A comparative view of COMPASS and TASTE functionality

Development phase	COMPASS functionality	TASTE functionality
Requirements specification	Specification of properties and requirements	
	Requirements analysis	
Architectural design	Contract-based design and refinement	
	Specification of system architecture (AADL)	Specification of system architecture (AADL)
Behavioral specification	Specification of the behavior of HW components (extended version of AADL)	Specification of the behavior of SW components (SDL or other language)
	Formal verification of functional behavior	
	Specification of HW faults	
	Fault injection/ Model extension	
	Formal verification of functional behavior (in presence of faults)	
	Fault Tolerance evaluation and dependability assessment	
Deployment		SW implementation
		Specification of the deployment on the target HW
		Code generation
	Trace validation for testing	Testing of the implementation

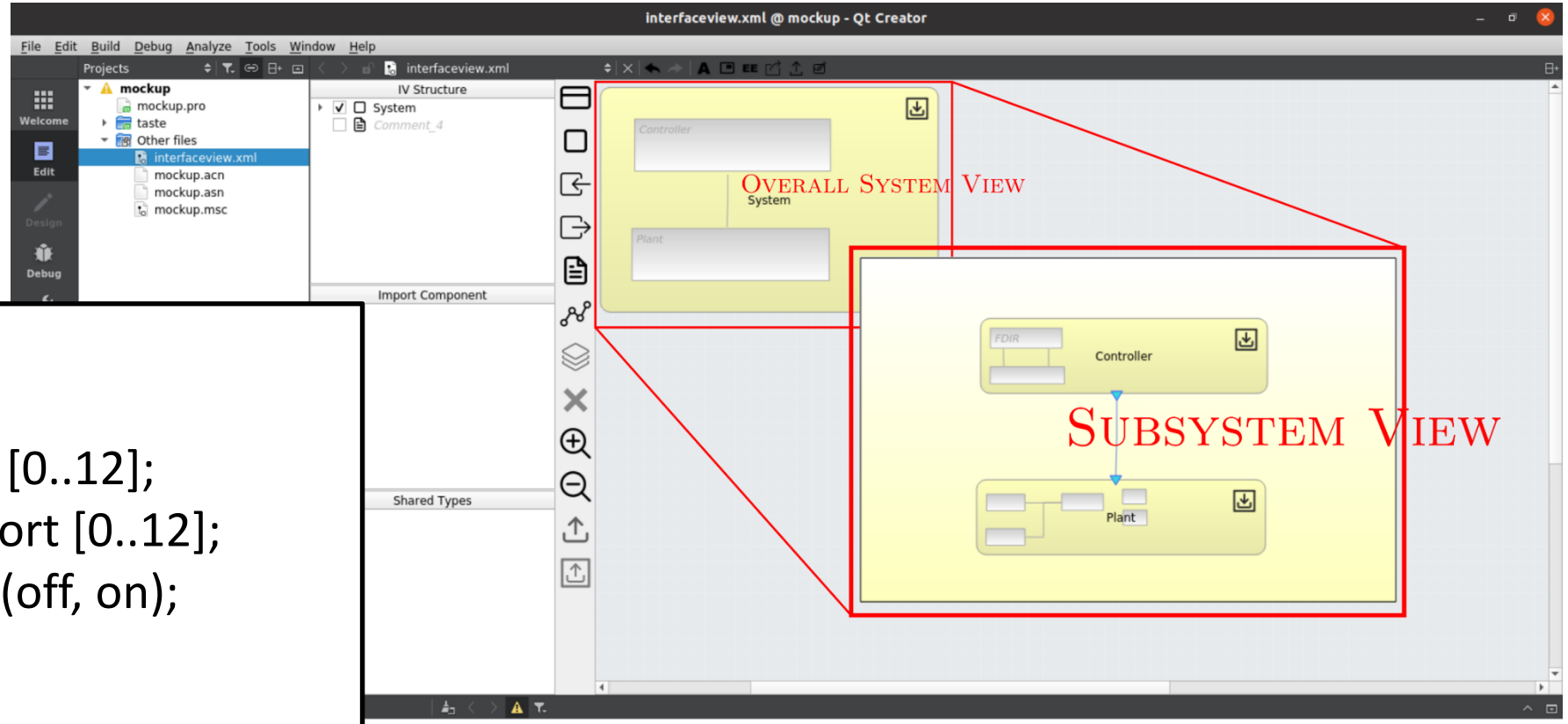
TASTE + COMPASS Workflow: An Example

- Redundant power system
  - Generators charging batteries
  - Batteries powering loads via circuit breakers
- Redundancies
  - Redundant lines between generators and batteries and between batteries and loads
- Faults
  - Generators stuck at off, batteries stuck at off, circuit breakers stuck at open/closed
- Requirements
  - All loads must be powered
  - A load must be powered by at most one battery at any given time, otherwise it gets broken
- FDIR components
  - Manage re-configurations



TASTE + COMPASS Workflow: An Example

- Modeling the system architecture (AADL)
  - Use the graphical editor available in the TASTE interface view

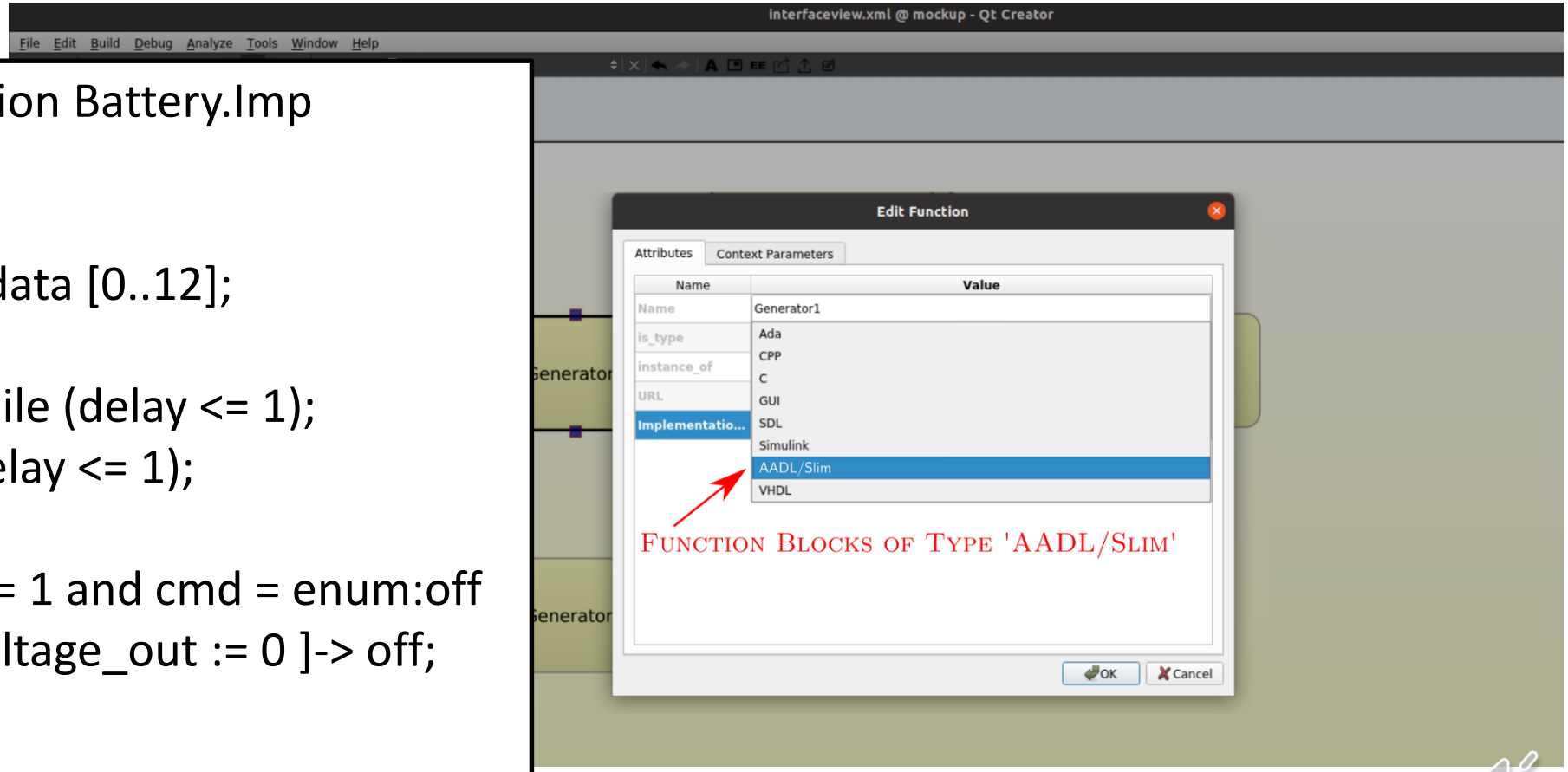



```
system Battery
 features
   voltage_in: in data port [0..12];
   voltage_out: out data port [0..12];
   cmd: in data port enum(off, on);
  …
end Battery;
```

TASTE + COMPASS Workflow: An Example

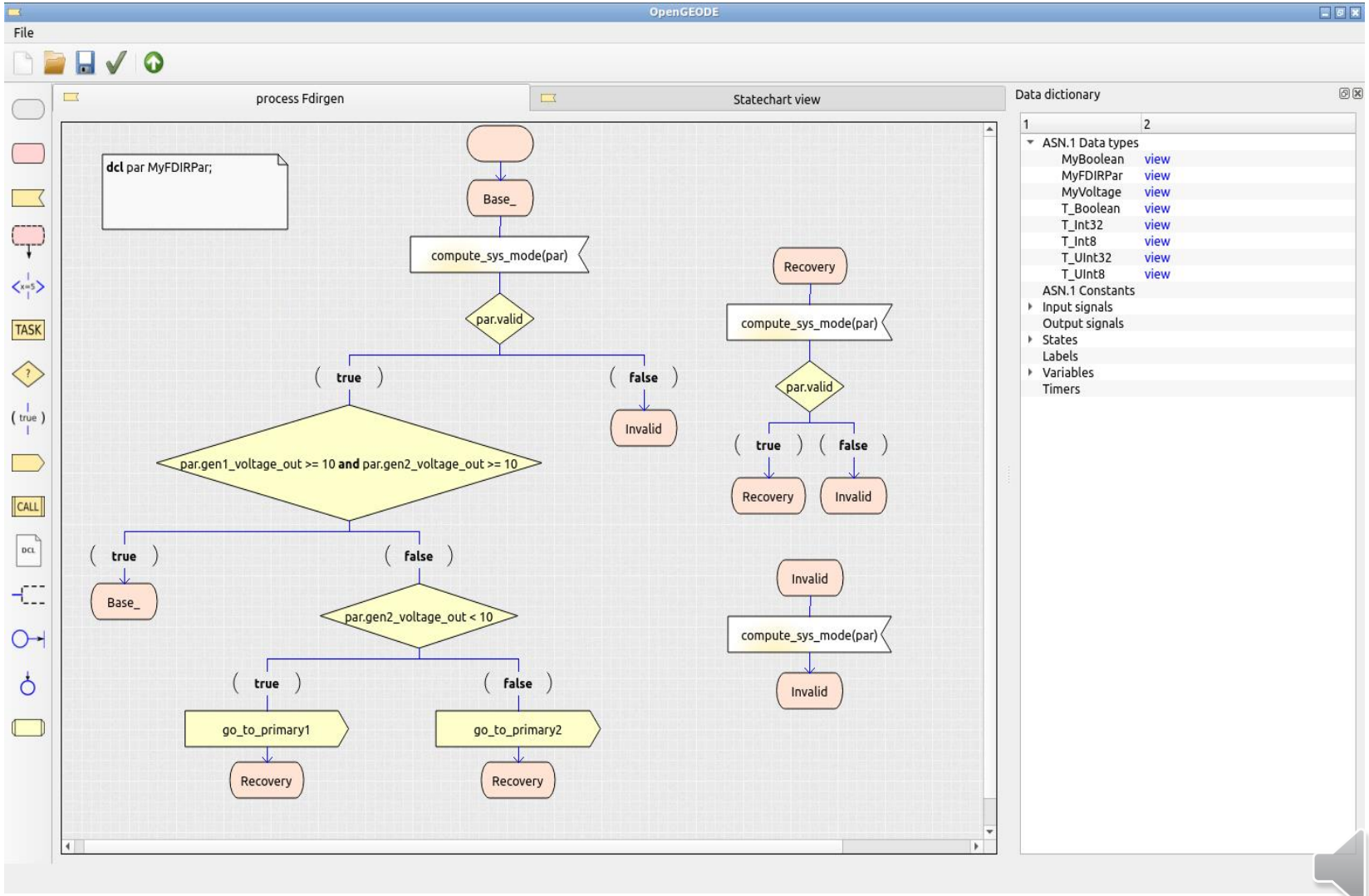
- Modeling the behavior of HW components (SLIM)
  - Declare a function block of type SLIM, and use an external textual editor (SDE)

```
system implementation Battery.Imp
 subcomponents
  delay: data clock;
  voltage_internal: data [0..12];
 states
  on: initial state while (delay <= 1);
  off: state while (delay <= 1);
 transitions
  on -[when delay >= 1 and cmd = enum:off
  then delay := 0; voltage_out := 0 ]-> off;
  …
end Battery.Imp;
```



TASTE + COMPASS Workflow: An Example

- Modeling the behavior of SW components (SDL)
  - Use the OpenGEODE graphical editor available in TASTE



TASTE + COMPASS Workflow: An Example

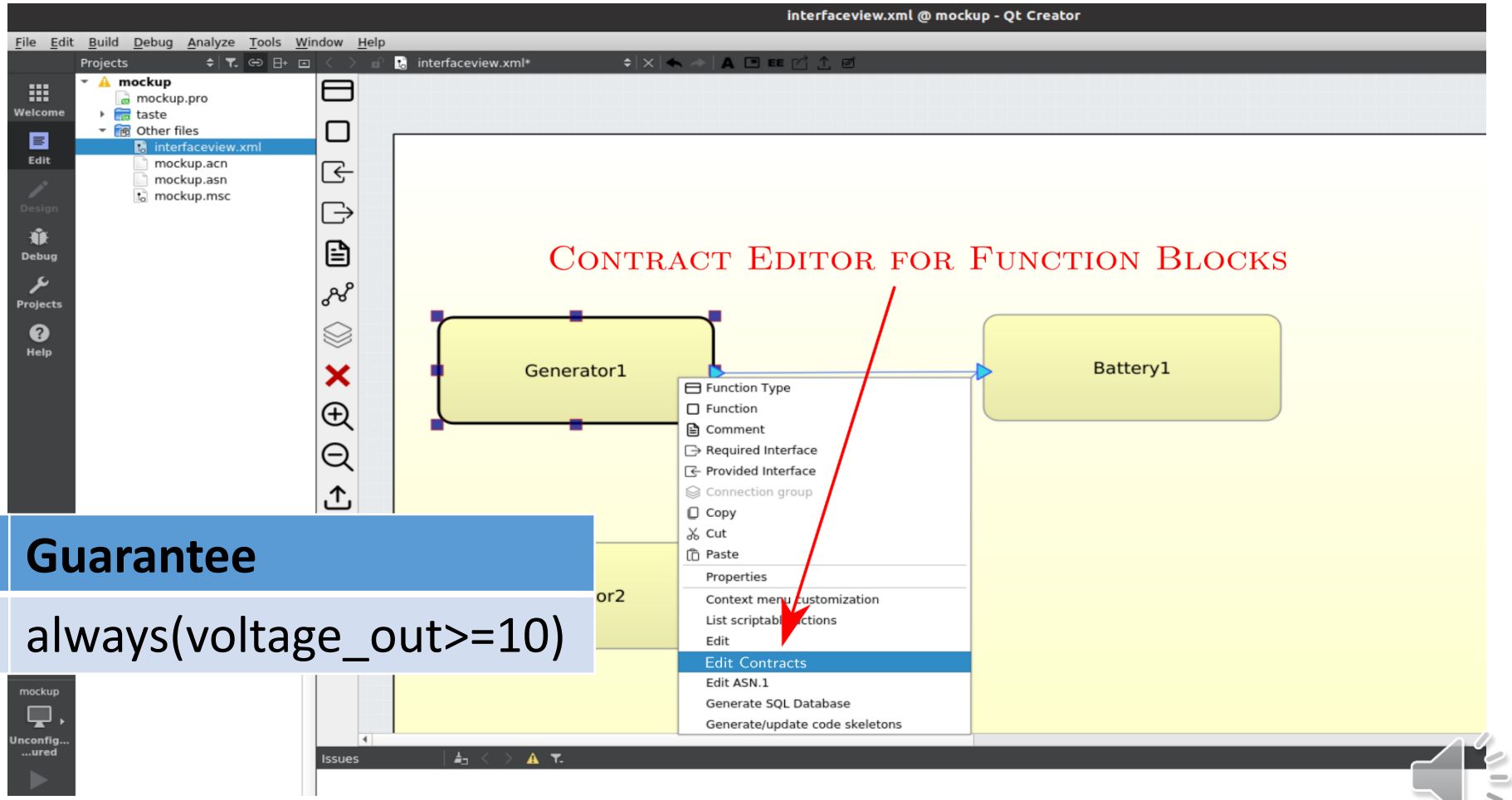
- Specification of properties and contracts

Name	Property
All loads powered	Always (ld1.is_powered and ld2.is_powered)
At least one load powered	Always (ld1.is_powered or ld2.is_powered)
No loads broken	Never (ld1.is_broken or ld2.is_broken)

Component	Name	Assumption	Guarantee
Generator	power	true	always(voltage_out >= 10)
Battery	power	always(voltage_in >= 10)	always(voltage_out >= 10)
CircuitBreaker	cmd_closed	true	always(cmd = enum:closed -> status = enum:closed)
	cmd_open	true	always(cmd = enum:open -> status = enum:open)
FDIRGen	Power	always(voltage_in_1 >= 10 or voltage_in_2 >= 10)	always(voltage_out_1 >= 10 and voltage_out_2 >= 10)
FDIRCb	cmd	always(voltage_in_1 >= 10 and voltage_in_2 >= 10)	always(cmd_bat1 = enum:on and cmd_bat2 = enum:on and cmd_cb1 = enum:closed and cmd_cb2 = enum:closed and cmd_cb3 = enum:open)

TASTE + COMPASS Workflow: An Example

- Specification of properties and contracts
  - Open an external textual editor from TASTE



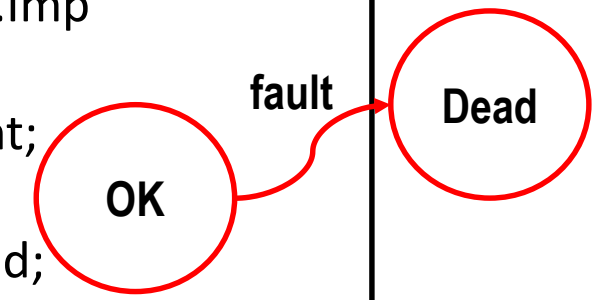
Name	Assumption	Guarantee
power	true	always(voltage_out>=10)

TASTE + COMPASS Workflow: An Example

- Fault injection
  - An example: permanent fault associated to generators

```
error model PermanentFailure
 features
   ok: activation state;
   dead: error state;
end PermanentFailure;

error model implementation
PermanentFailure.Imp
 events
   fault: error event;
 transitions
   ok -[fault]-> dead;
end PermanentFailure.Imp;
```



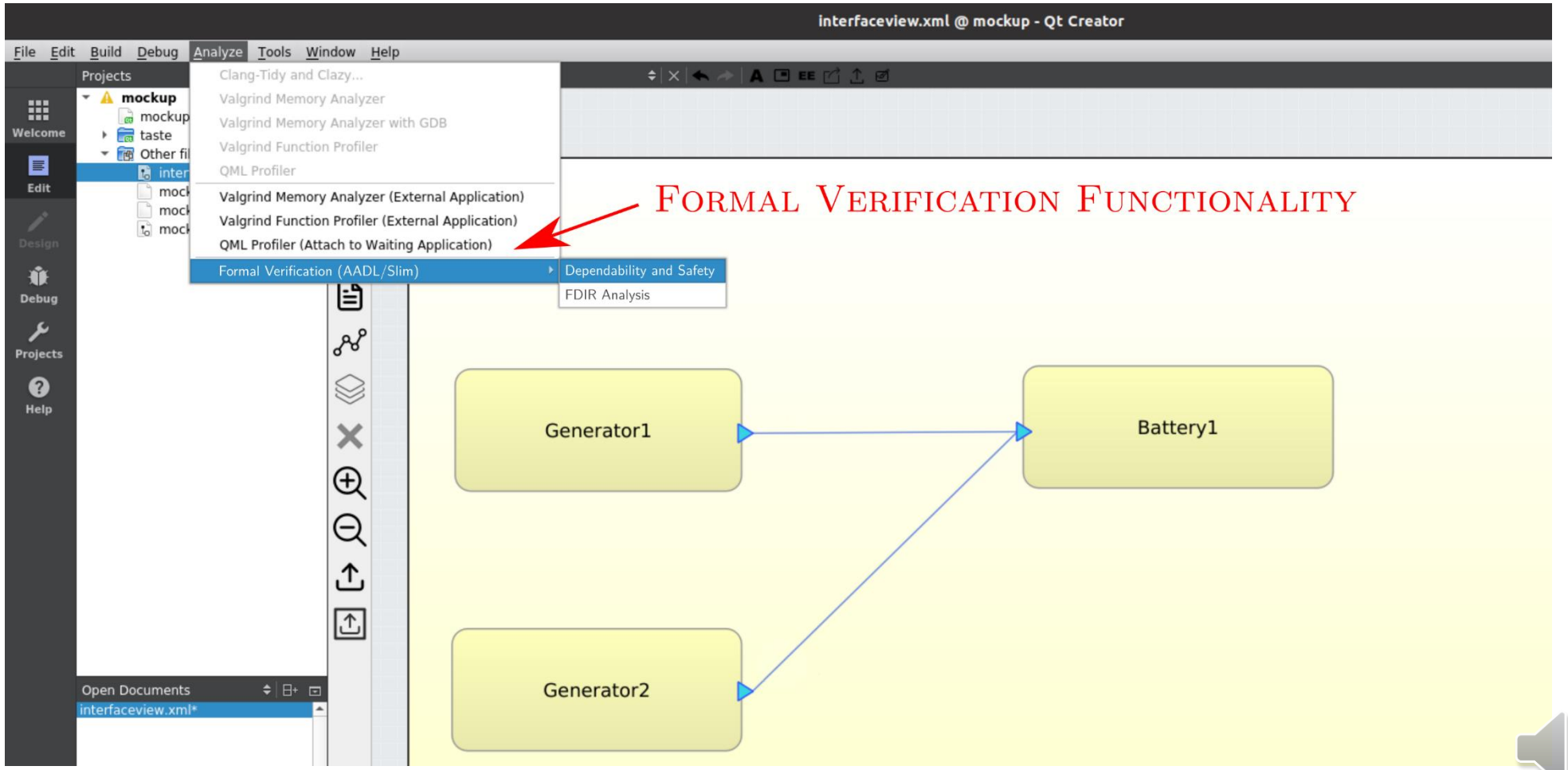

```
system Generator
 features
   voltage_out: out data port [0..12];
end Generator;

system implementation Generator.Imp
 properties
   ErrorModel => classifier(PermanentFailure.Imp);
   FaultEffects => ([State=>"dead";
    Target=>reference(voltage_out); Effect=>"0";]);
end Generator.Imp;
```

Fault injection

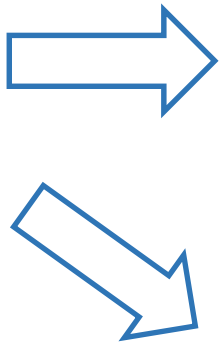
TASTE + COMPASS Workflow: An Example

- Formal verification
  - Call the formal verification functionality from TASTE



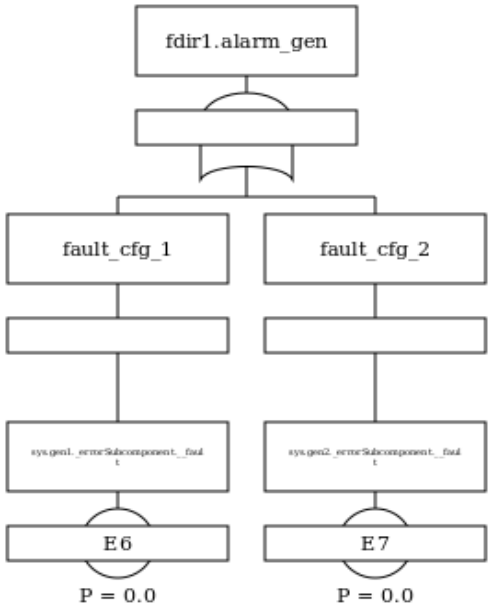
TASTE + COMPASS Workflow: An Example

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



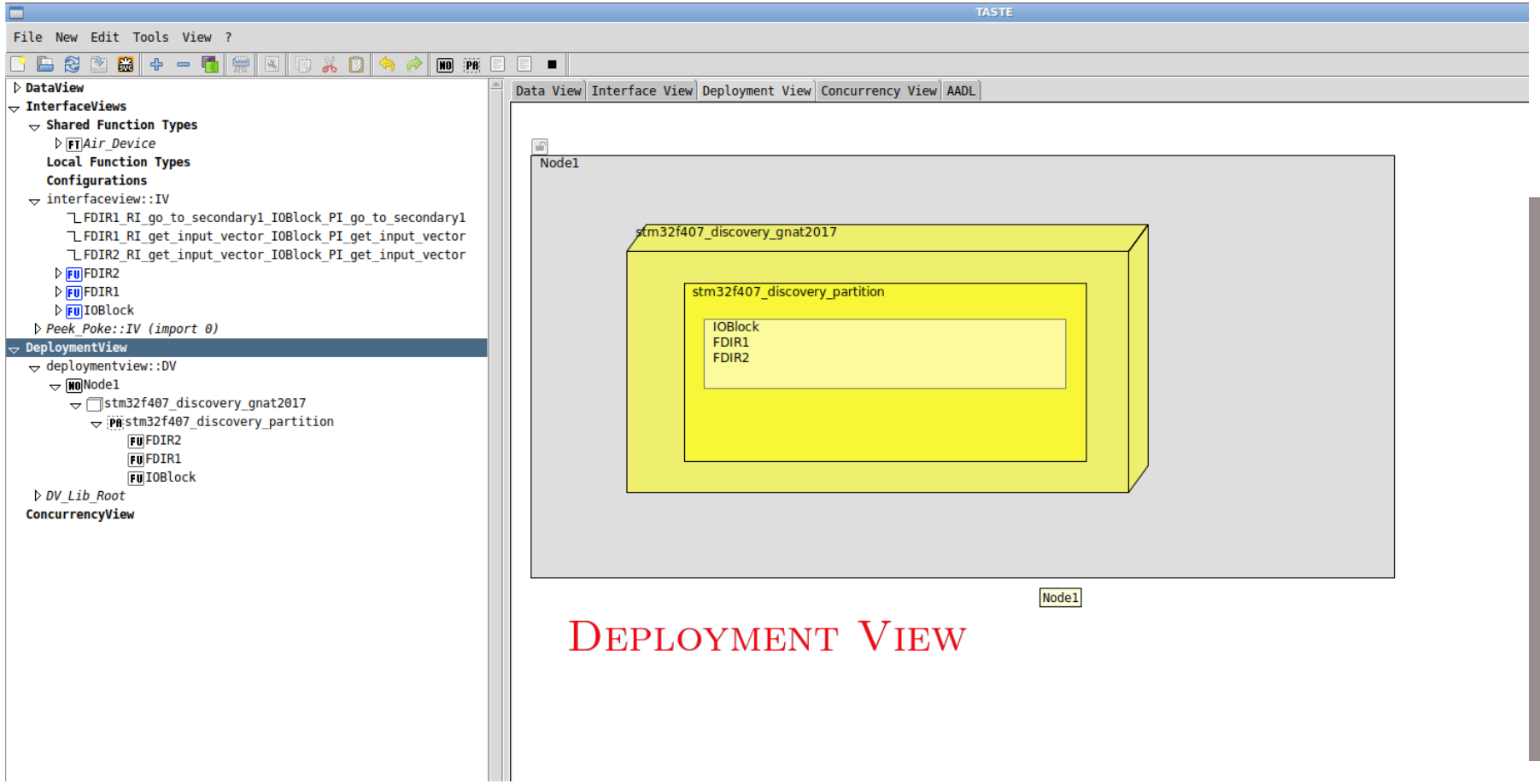
The property is false
The LTL property:Globally, it is always the case that {ld1.is_powered and ld2.is_powe

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.go_to_secondary1: FALSE		
fdir1.id	(1)	(1)	(1)	(1)
fdir1.mode	(mode_base)	(mode_base)	(mode_base)	(mode_base)
fdir1_reactivated				
fdir1.resolved_go_to_secondary1				
fdir1.resolved_go_to_secondary2				
fdir1.stutter				
fdir1_t_#delta	(1)	(0)	(0)	(0)
fdir1.timed				
fdir1_val_gen1_voltage_out	(6)	(6)	(6)	(6)
fdir1_val_gen2_voltage_out	(6)	(6)	(6)	(6)
fdir1.valid				
fdir1_val_valid				
fdir2_activated				
fdir2.active				
fdir2.cmd_bat1	(on)	(on)	(on)	(on)
fdir2.cmd_bat2	(on)	(on)	(on)	(on)
fdir2.cmd_cb1	(closed)	(closed)	(closed)	(closed)
fdir2.cmd_cb2	(closed)	(closed)	(closed)	(closed)
fdir2.cmd_cb3	(open)	(open)	(open)	(open)

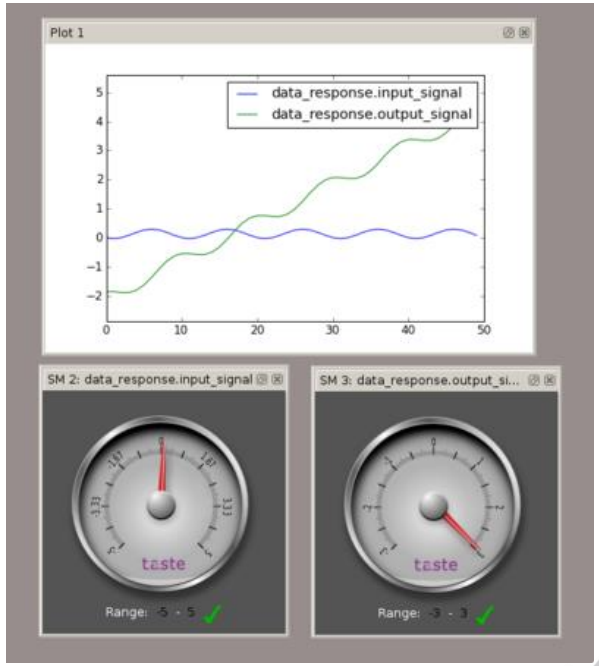


TASTE + COMPASS Workflow: An Example

- Code generation, deployment and testing
  - Use the TASTE functionality



DEPLOYMENT VIEW



TASTE + COMPASS Workflow: 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
  - Code generation
  - Deployment
  - 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 results are expected by the end of 2022