FMU Simulation Environment for Cyber-Physical Systems Co-Design & ASP Diagnose Tool

SCIENCE
PASSION
TECHNOLOGY

Introduction & Use Case

David Kaufmann

SS23 18/04/2023

David.Kaufmann@ist.tugraz.at

Overview

- Introduction
- FMU Model Generation
  - FMI Standard
  - OpenModelica FMU generation
- FMU Simulation Environment
  - Use Case – Heater model
  - Demo
- ASP Diagnose Tool
  - Use Case – Benchmark Circuits ISCAS85
- FMU Simulation Tool & ASP Diagnose Tool
  - Use Case – Two Lamps
  - Demo

Simulation & Diagnosis of Cyber Physical Systems

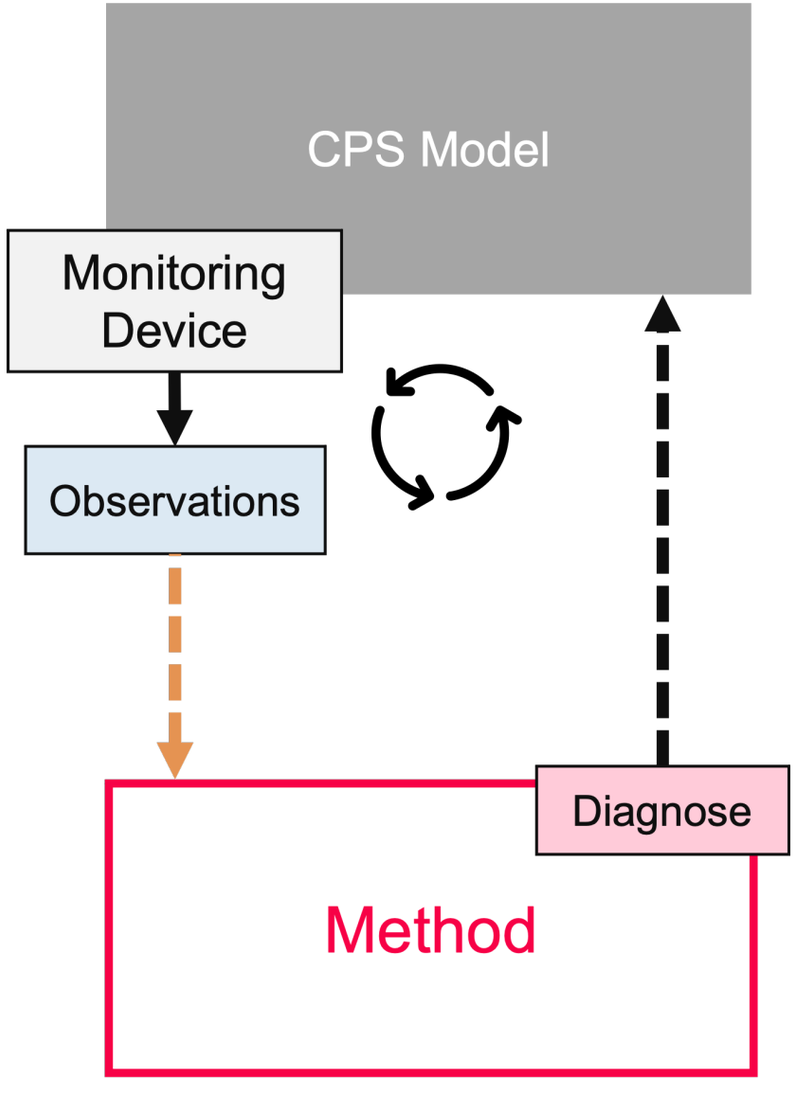
- Detection of failures during runtime
- Localization of root cause
- Initiating repair actions
- Keep the system operational under safe conditions

Requirements for **testing** and **validation** of an advanced diagnosis method:

- Co-simulation environment framework
- Standardized simulation environment
- Step-by-step simulation
- Fault injection during runtime
- Interface for information flow to and from the method under test
- Different programming environments

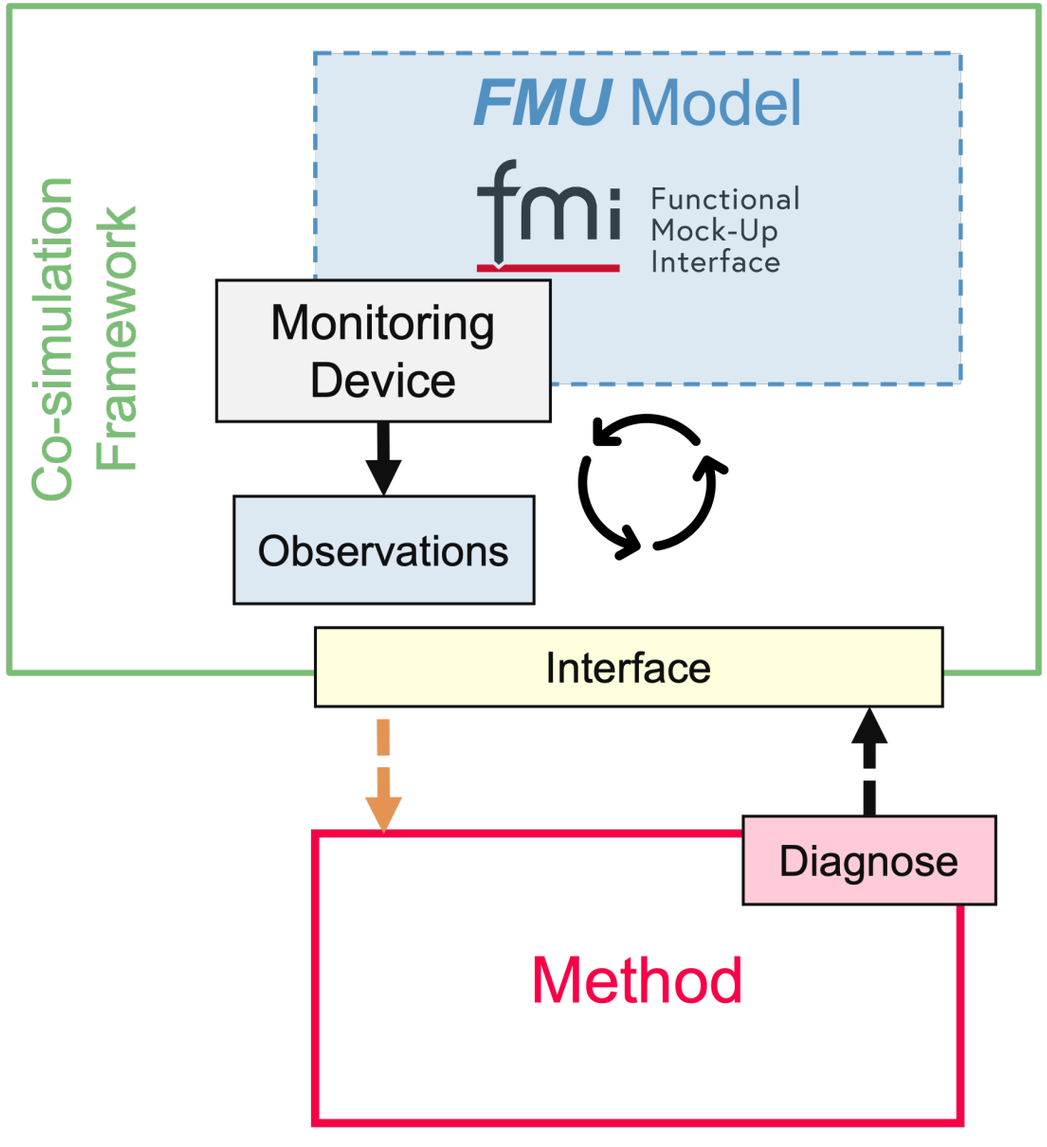
Framework Design & Method Interface

- Cyber-Physical-System (CPS) model
- Monitoring for observations
- Fault detection & localization of root cause
- Diagnose feedback for failure mitigation



Framework Design & Method Interface

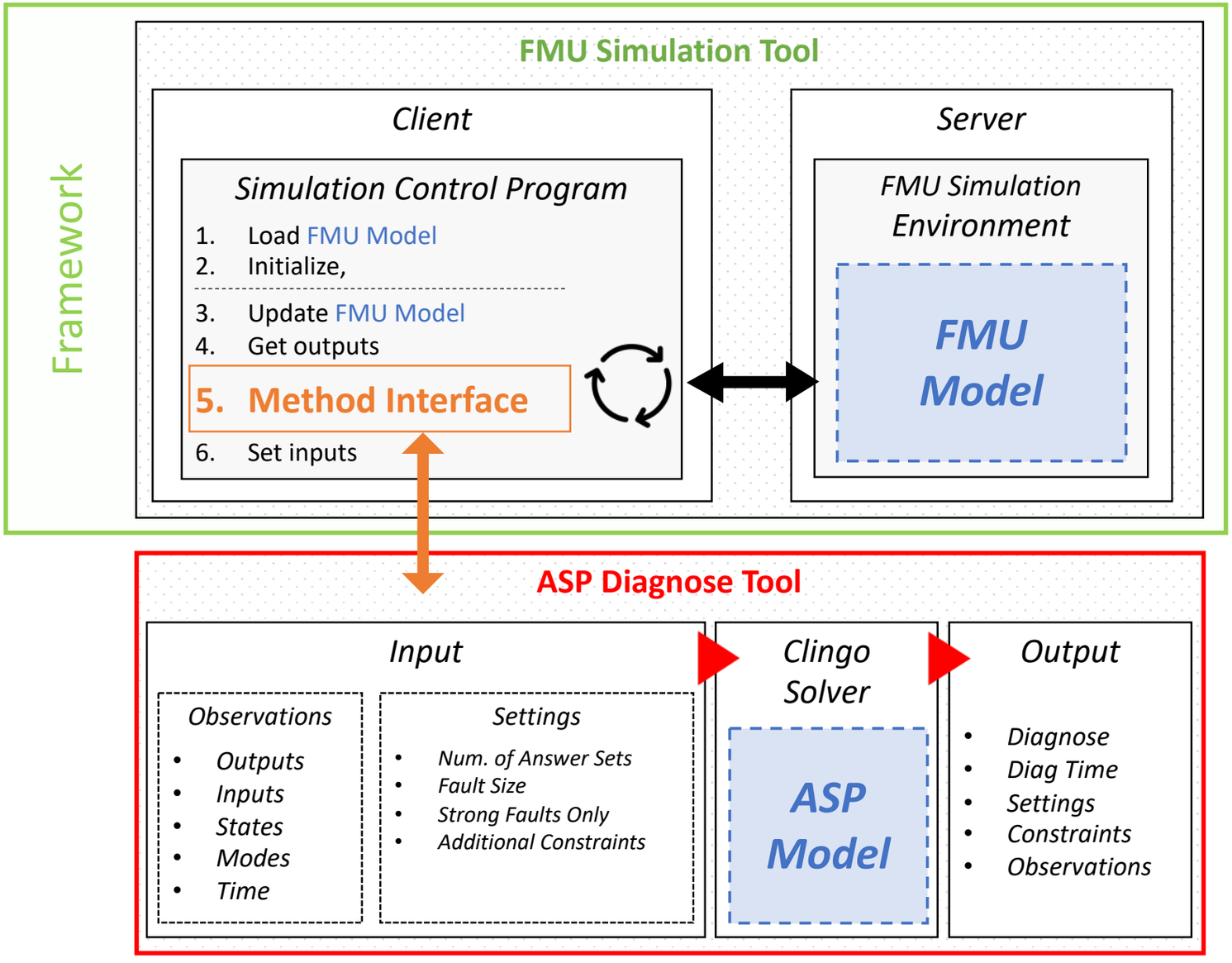
- Co-simulation framework
- Use of standardized format FMU as models
  - Solver integrated in co-simulation FMU
  - Perform step-by-step simulation
- Simple interface for linking tools
- Different programming environments
- Test & validation environment for tools



Framework Design & Method Interface

Framework:

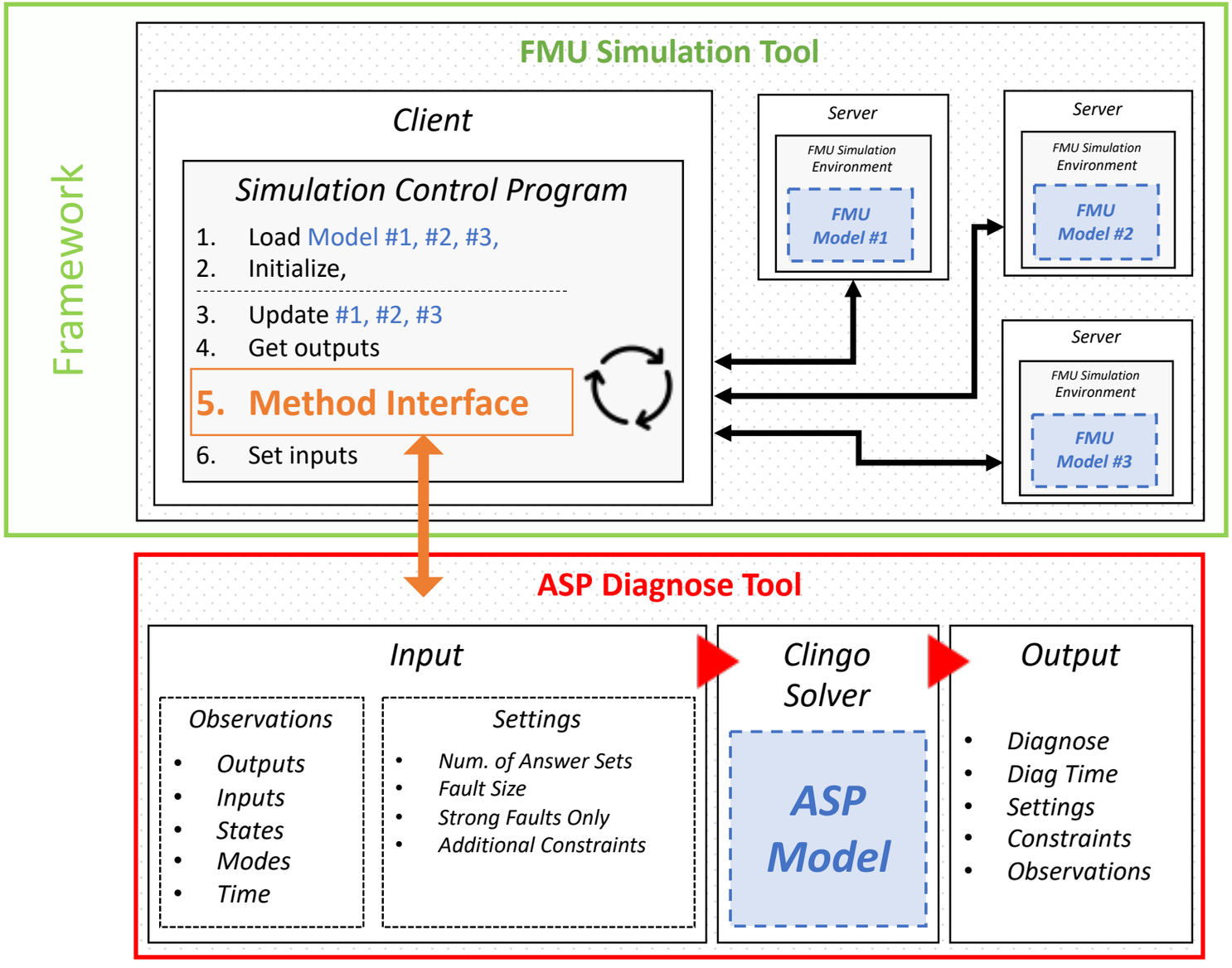
- FMU Simulation Tool
- Client-Server solution
- Dockerized simulation environment
- Multiple models in co-simulation
- PyFMI library
- REST API communication



Framework Design & Method Interface

Framework:

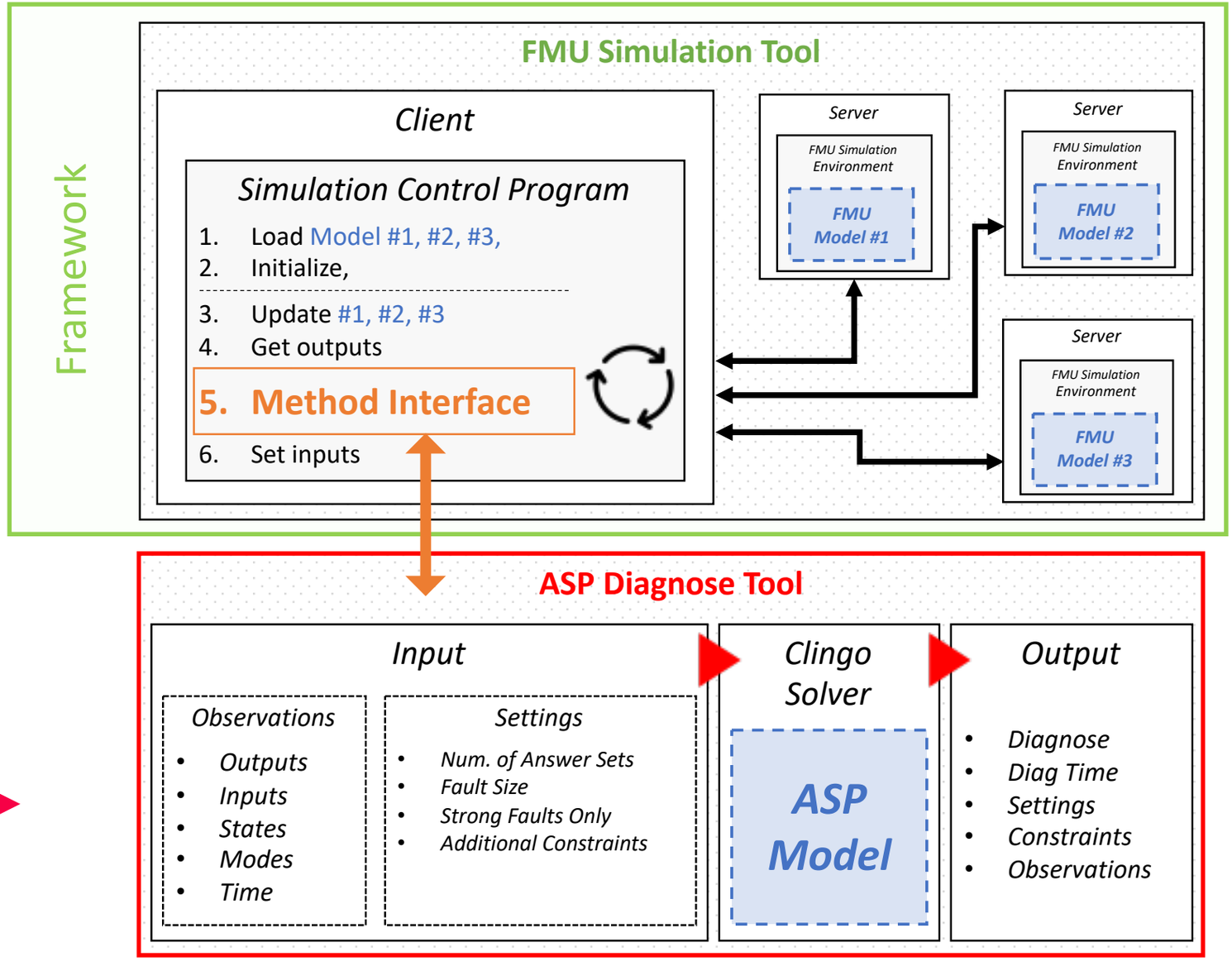
- FMU Simulation Tool
- Client-Server solution
- Dockerized simulation environment
- Multiple models in co-simulation
- PyFMI library
- REST API communication



Framework Design & Method Interface

Method:

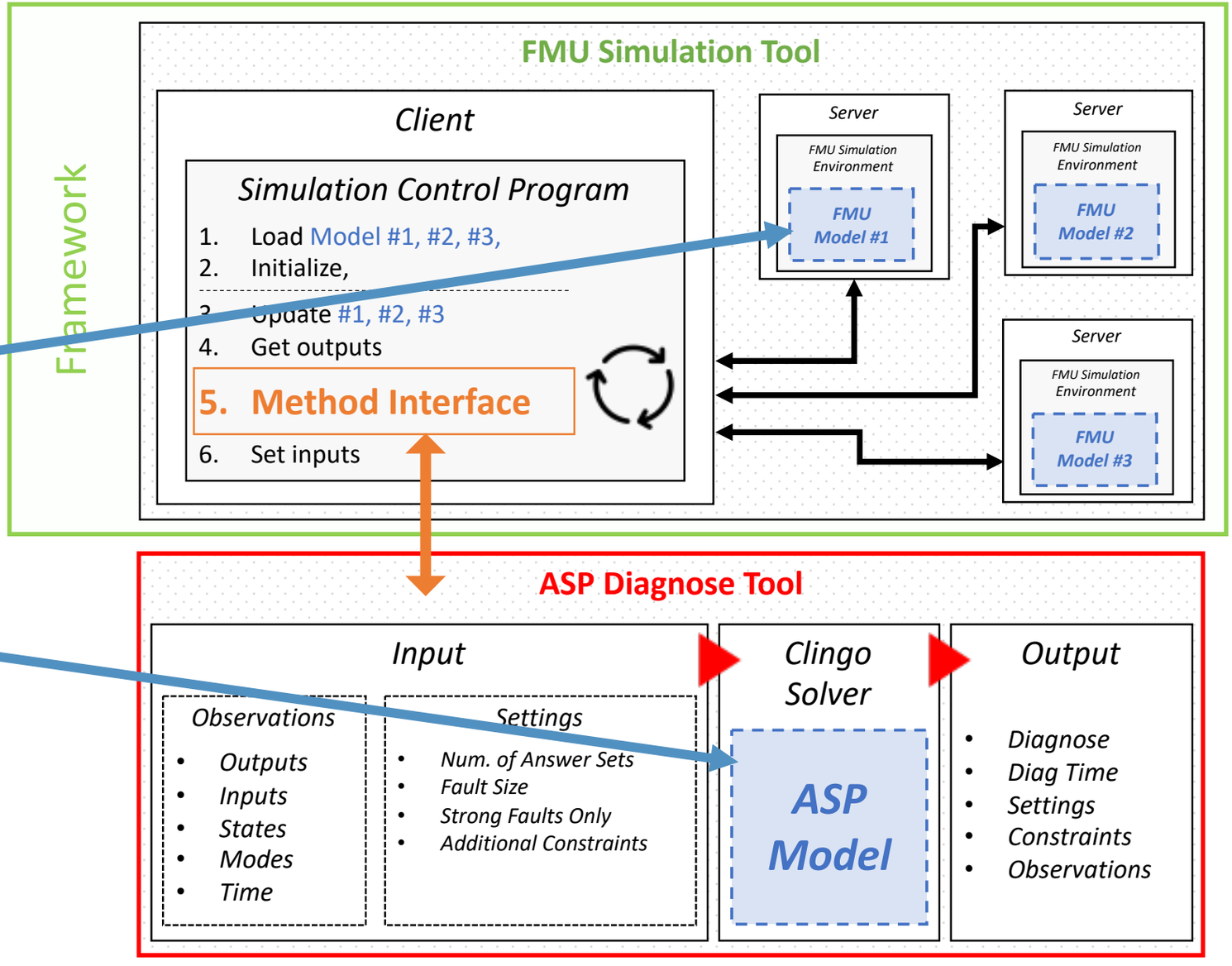
- ASP Diagnose Tool
- Theorem solver Clingo 5.4.1
- Input: observations & settings
- Output format: JSON, CSV



Framework Design & Method Interface

Models:

- FMU Model
  - generated from CPS
- ASP Model
  - abstract model of CPS



FMU Model Generation

for Cyber-Physical Systems in OpenModelica

Functional Mock-up Unit (FMU)

The Functional Mock-up Interface (or FMI) defines a standardized interface to be used in computer simulations to develop complex cyber-physical systems.

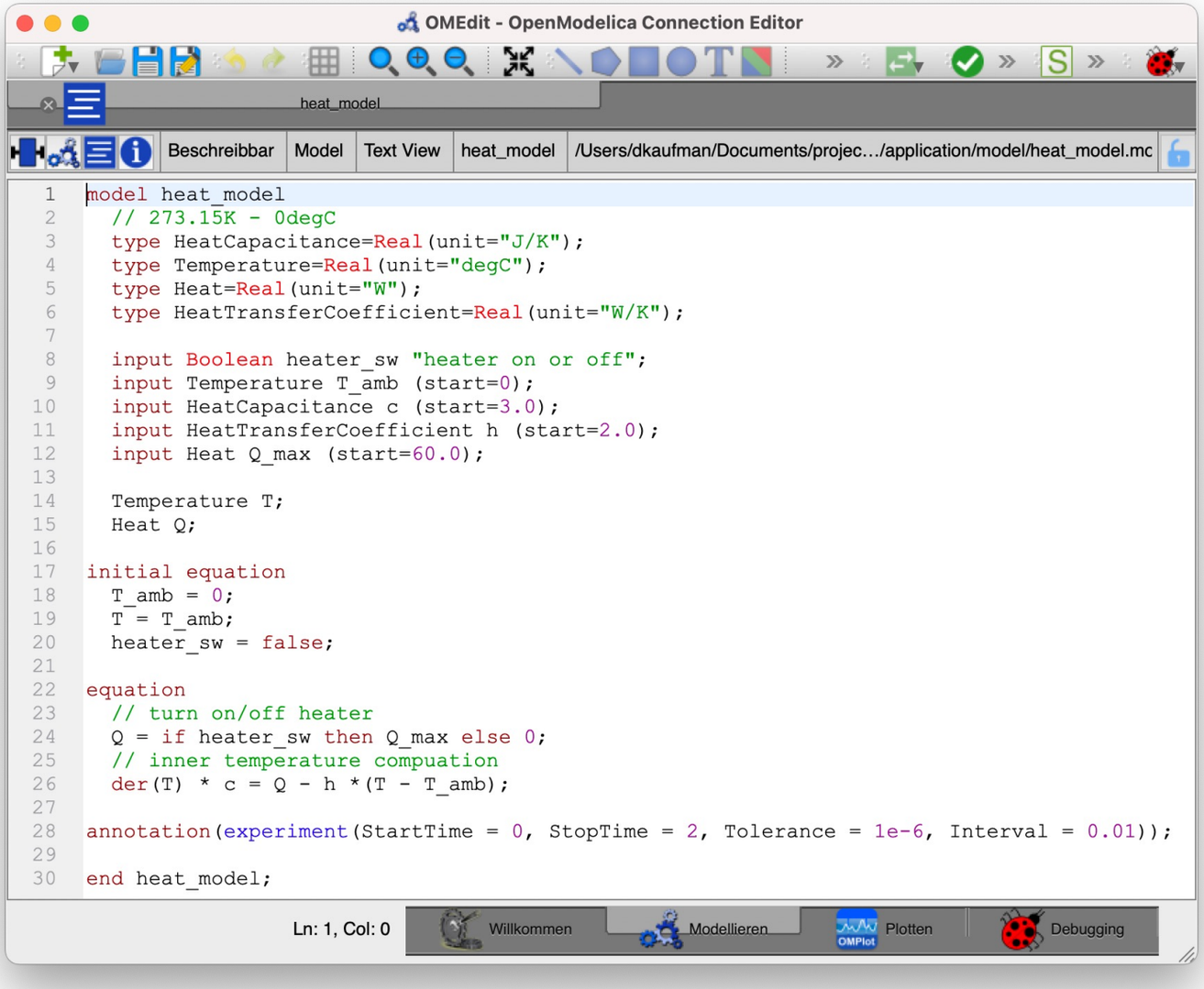
FMI defines an interface that is implemented by an executable called a Functional Mock-up Unit (FMU)

An FMU contains following:

- A model description XML file:
  - Model information.
  - Variable definitions as type, unit and description.
  - General model information as name, generation tool and FMI version.
- Model equations:
  - Differential equations, algebraic relations and discrete equations.
  - Represented in C functions and binaries.

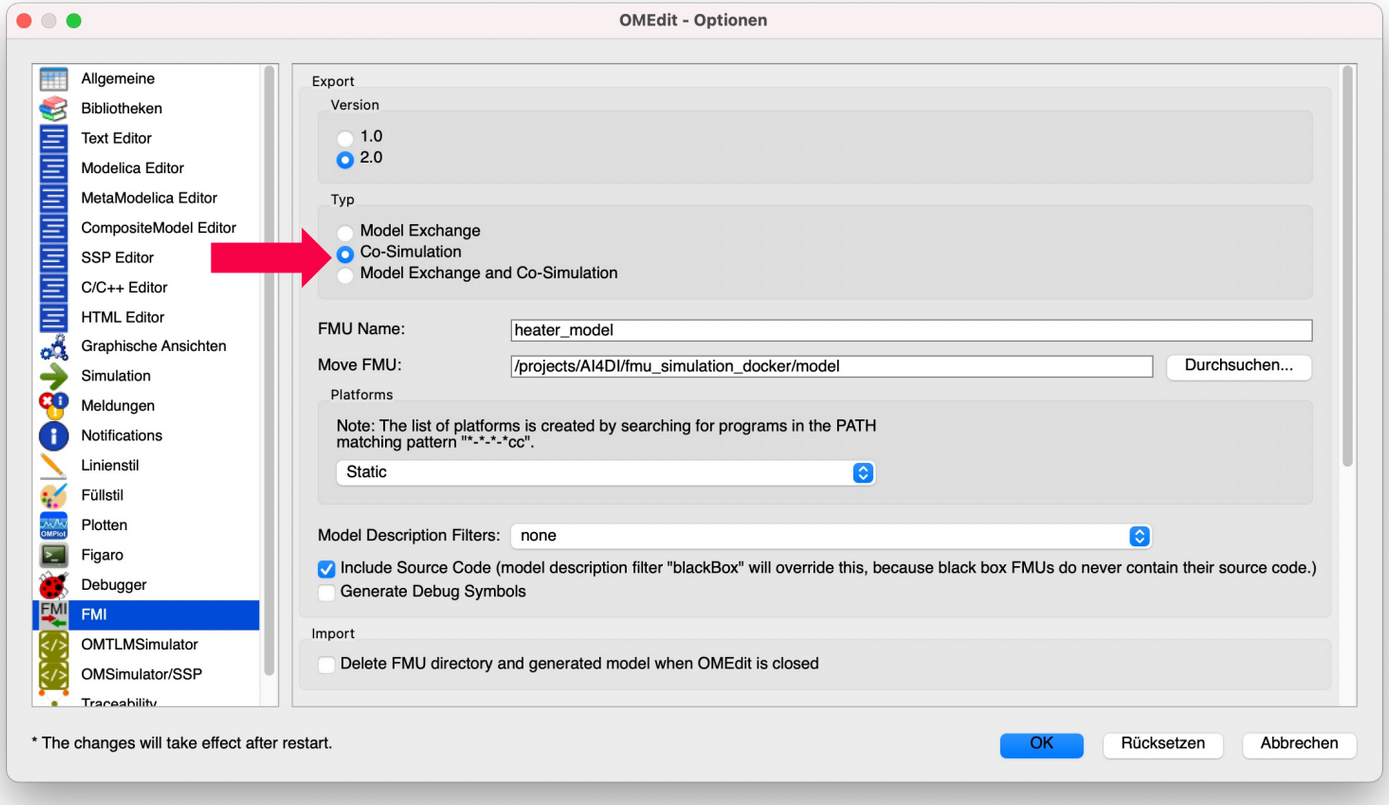
Modeling in OpenModelica

- OMEdit editor for creating models
- OS selection:
  - Linux: use pre-build OpenModelica VirtualBox (Link to Linux VB release)
  - Local OS: use installation of OpenModelica for your local system
- Model the system according to OpenModelica standard
- Input
  - Define inputs with “input” statement to have access in FMU model.
- Output:
  - All used variables can be read as output signals.

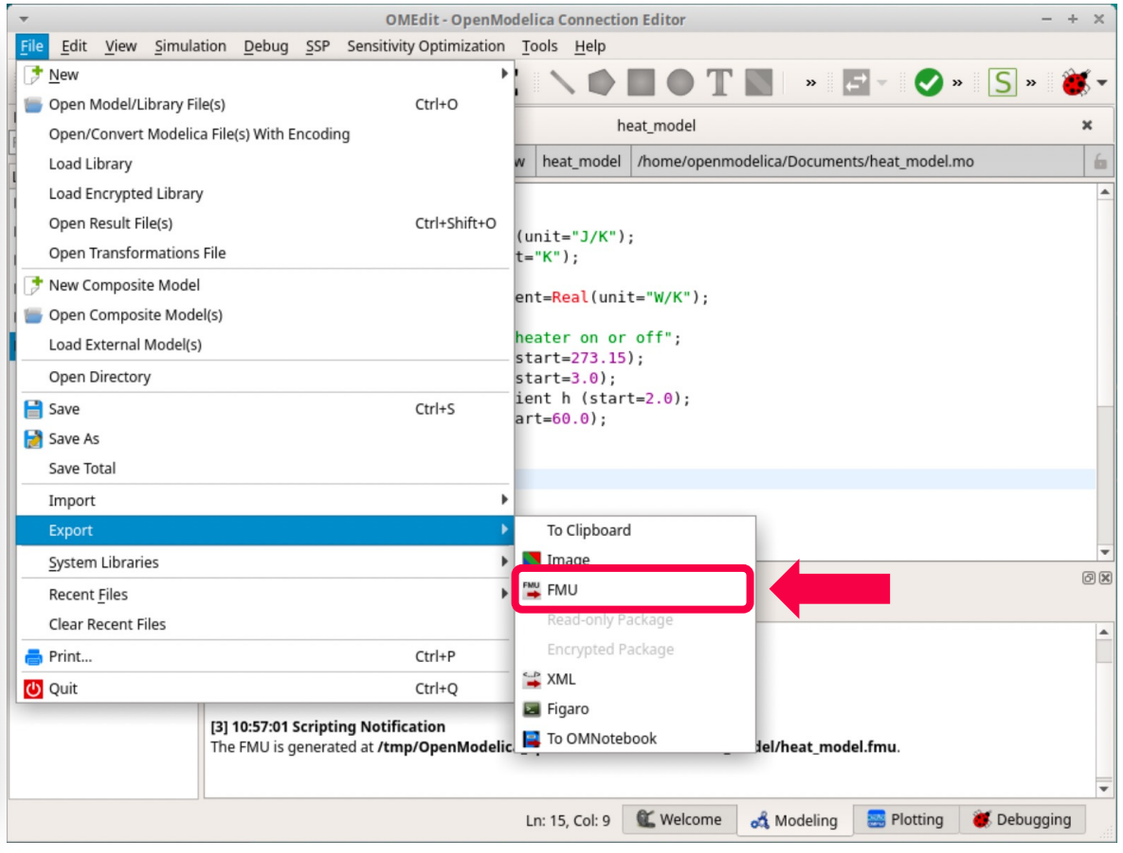


Modeling in OpenModelica

Co-Simulation:
- FMUs contain local solvers
- Enables step by step simulation



Export FMU:



FMU Simulation Environment

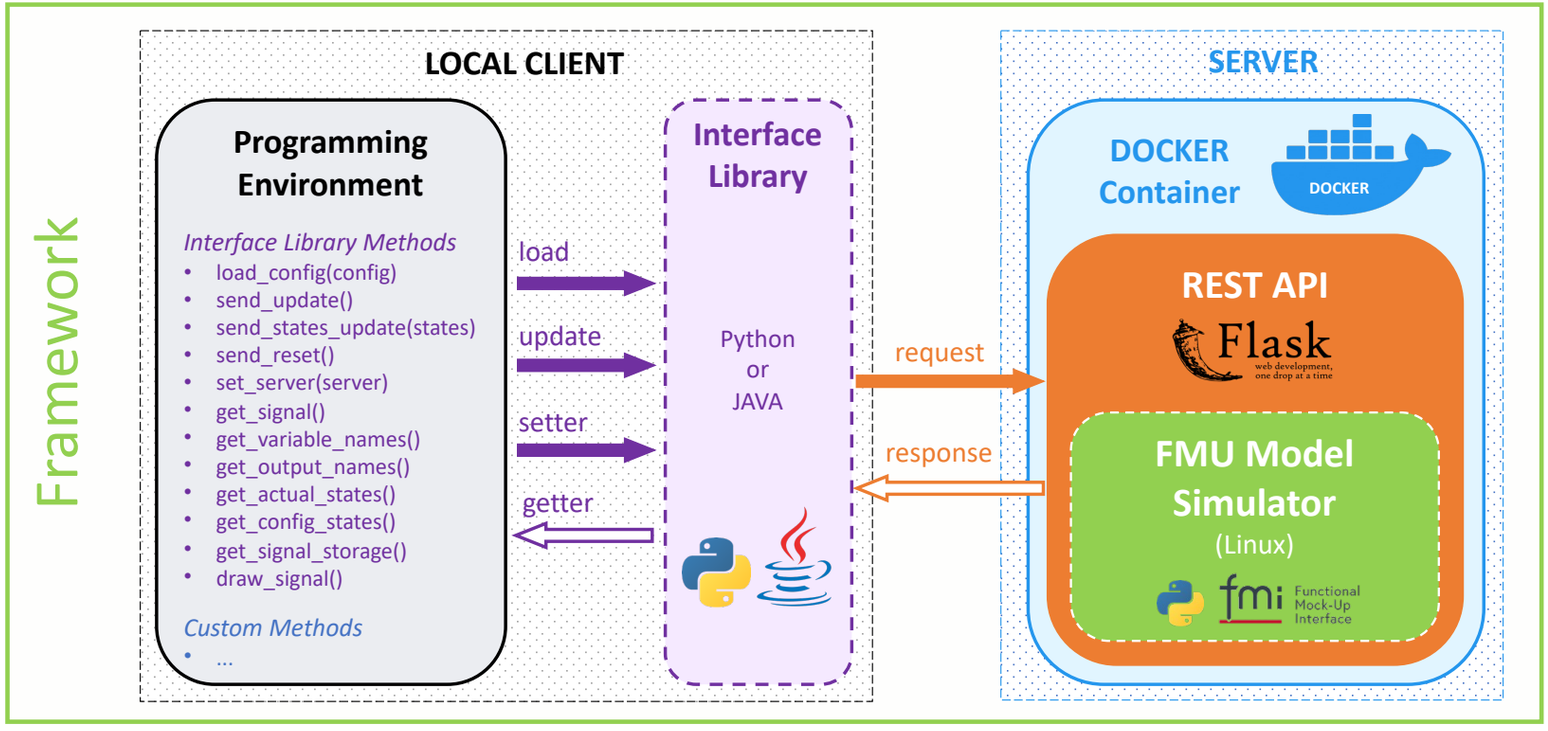
for Cyber-Physical Systems Co-Design

FMU Simulation Environment
Client Server Setup

User friendly simulation environment for **FMUs**.

Coupling of multiple simulated FMU models.

Client-server environment to access from different programming environments (Python or Java).



FMU Simulation Environment

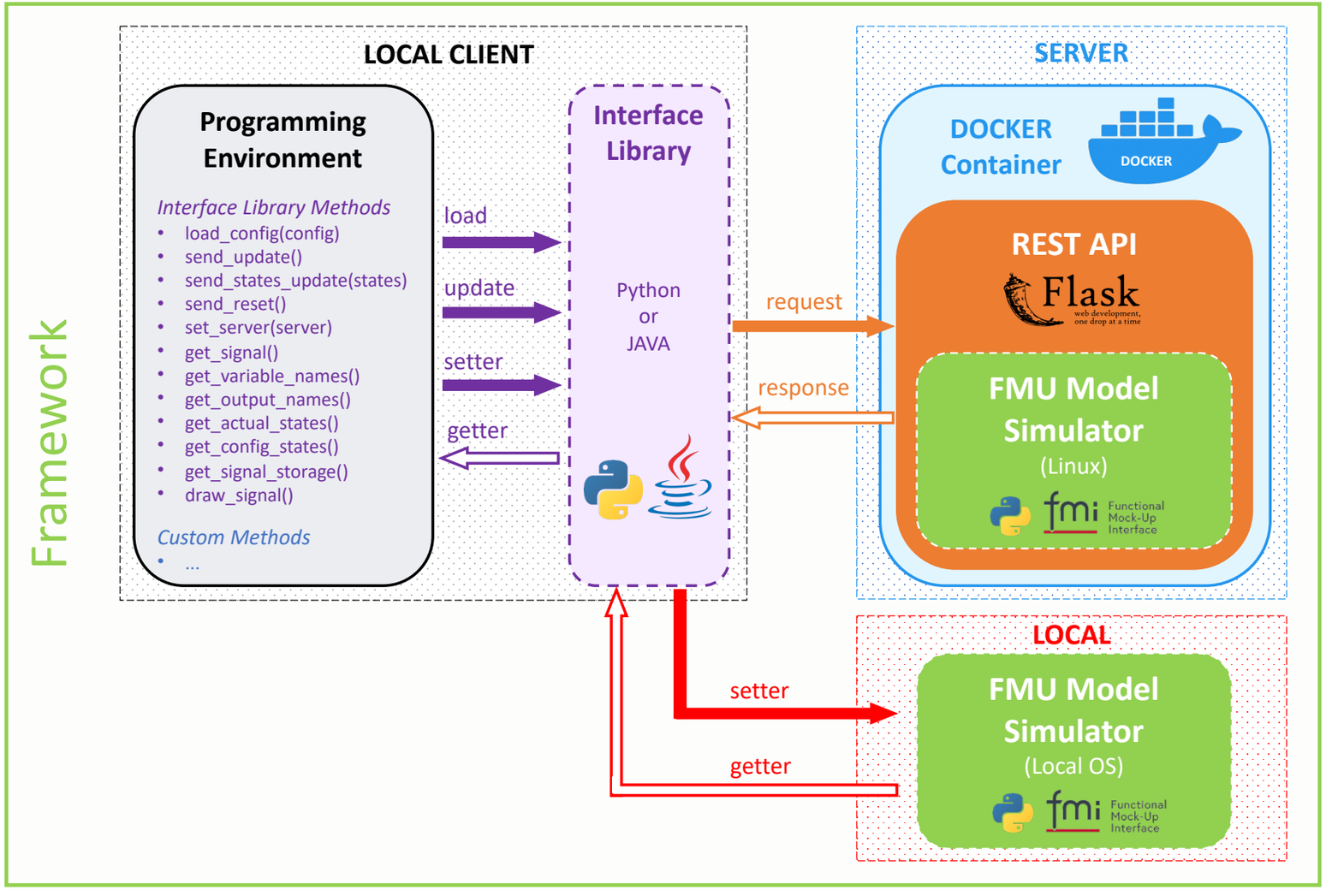
Client Server Setup & Local Setup

Dockerise simulation environment as server with a REST API for **Linux** FMU based models.

Local simulation environment for FMU models generated based on the **local OS**.

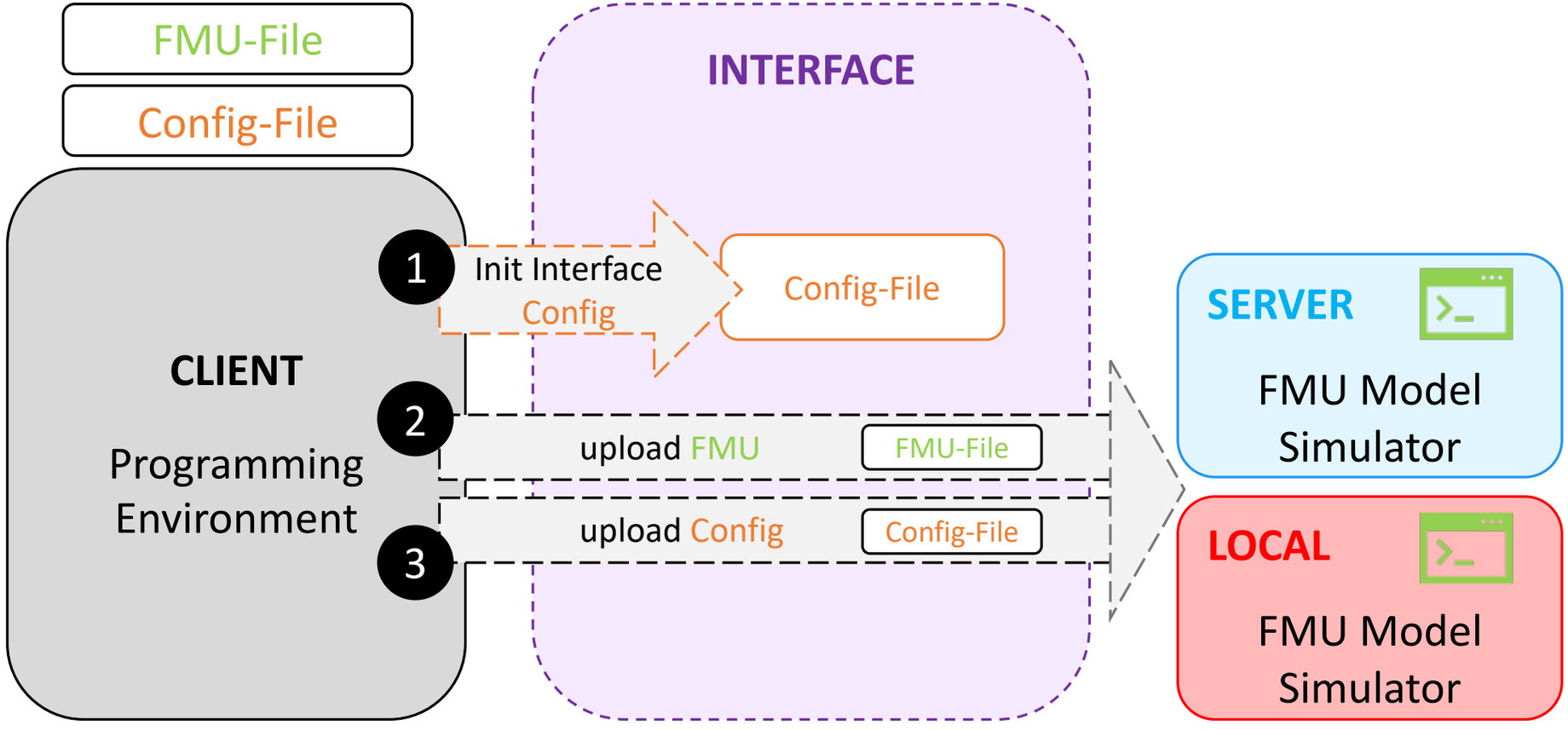
Client interface to communicate with the **server** or **local** simulation environment.

Clone Gitlab repository:
https://github.com/QAMCAS/FMU-Simulation-Environment



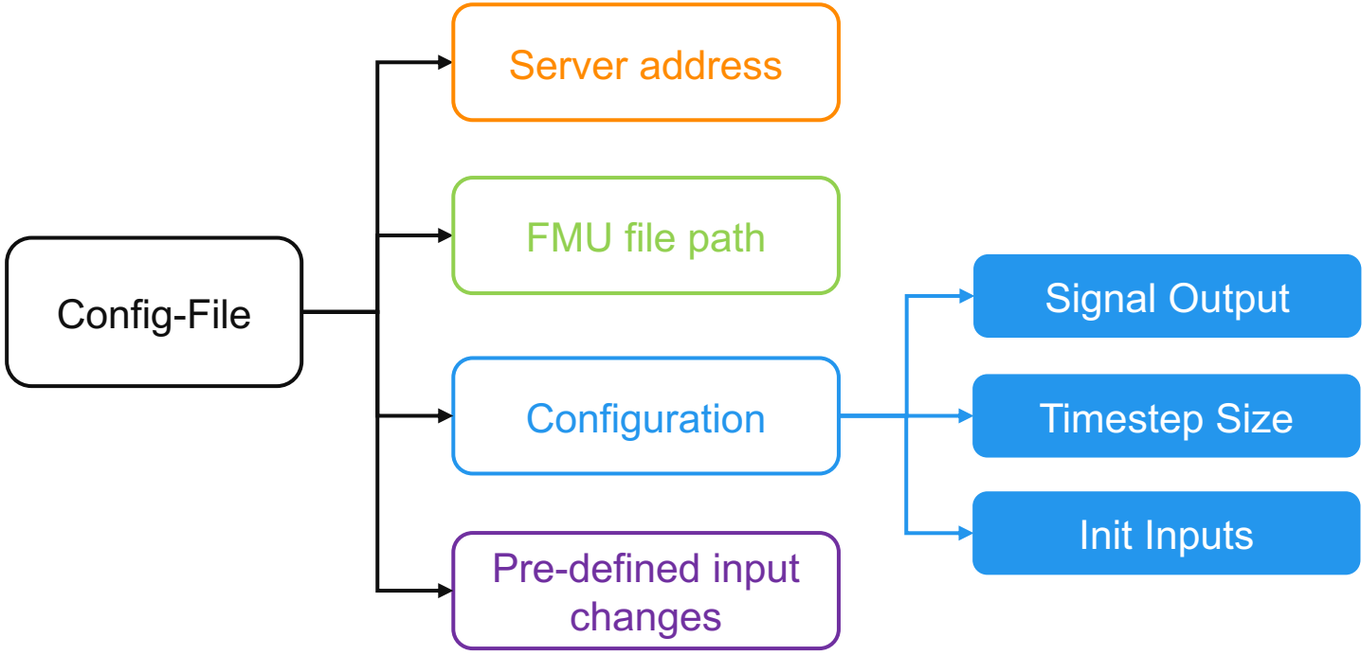
FMU Simulation Environment

Configure and Upload Simulation Model



FMU Simulation Environment

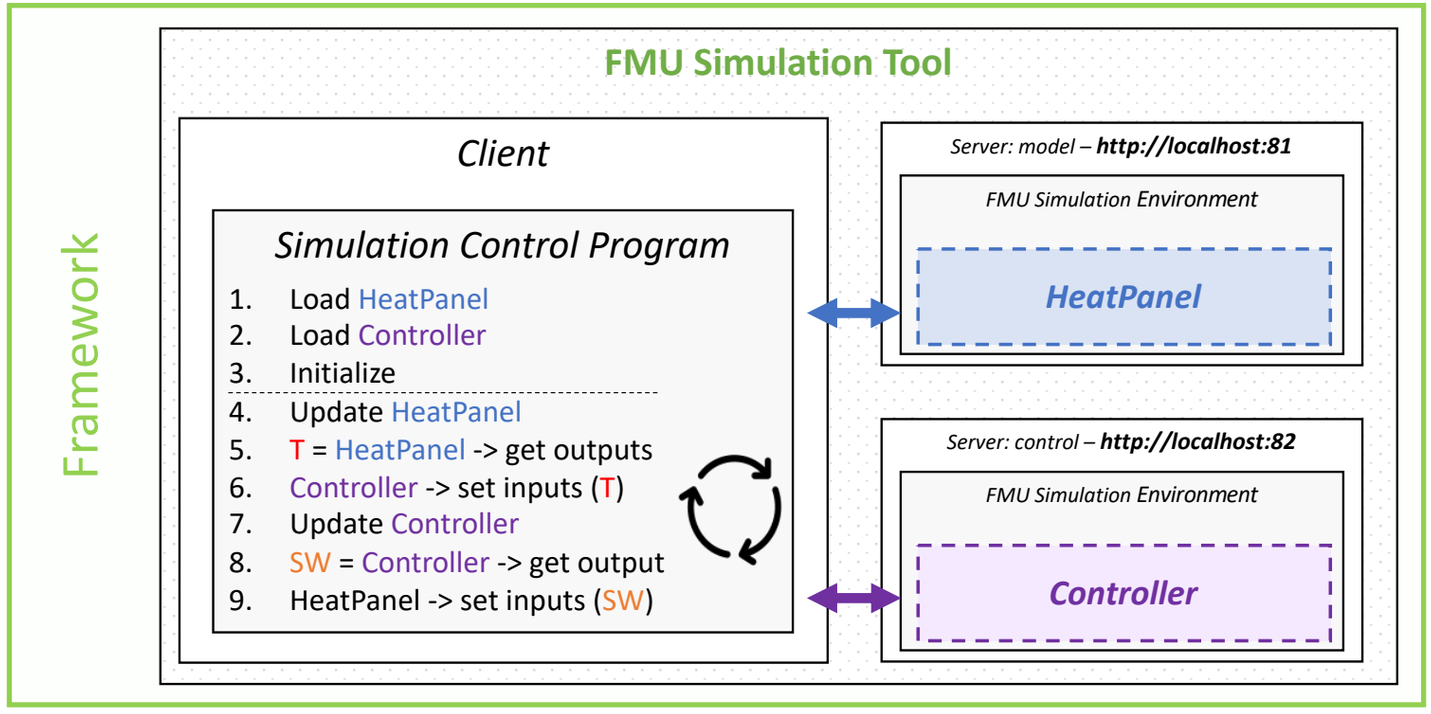
Model Configuration Structure




```
{
        "server": "http://localhost:81/",
        "fmu": "/application/model/dc_motor_model.fmu",

        "config":
        {
                "output":[
                        ”out.i",
                        ”out.v",
                        ”out.omega"
                ],
                "timestep": 0.001,

                "init_input":
                {
                        "bat_state":"ok",
                        "switch_state":"ok",
                        "switch_mode":"close",
                        "mot_state":"ok",
                        "tor_state":"ok"
                }
        },

        "input":
        [
                {
                        "time":1.0,
                        "mot_state":"ok",
                        "tor_state":“f1"
                },
                {
                        "time":2.0,
                        "mot_state":“f1",
                        "tor_state":"ok"
                }
        ]
}
```

Use Case - Heater Model

- Simple Heater Model
  - First order differential equation
  - Heater panel to heat up a medium
  - Surrounding environment with a global ambient temperature
  - Constant heat capacitance,
  - Constant heat transfer coefficient
  - Medium temperature is observed
- Temperature Control Unit
  - External placed 2-point controller



Use Case - Heater Model

Live Demo

1. Build Docker Image

2. Start Docker Container
   - Name
   - Address & port

3. Simulation Controller
   1. Initialize
      - Communication interface (server address)
      - Upload FMU model to server
      - Upload Configuration file
   2. Trigger simulation update by given timestep
   3. Call getter and setter methods
   4. Trigger next update cycle

heater_model.fmu

Heater FMU

config_heater_model.json

Heater CONFIG

heater_controller.fmu

Control FMU

config_heater_control_unit.json

Control CONFIG

Docker Container

Start HeatPanel

Name: model

Address: localhost:81

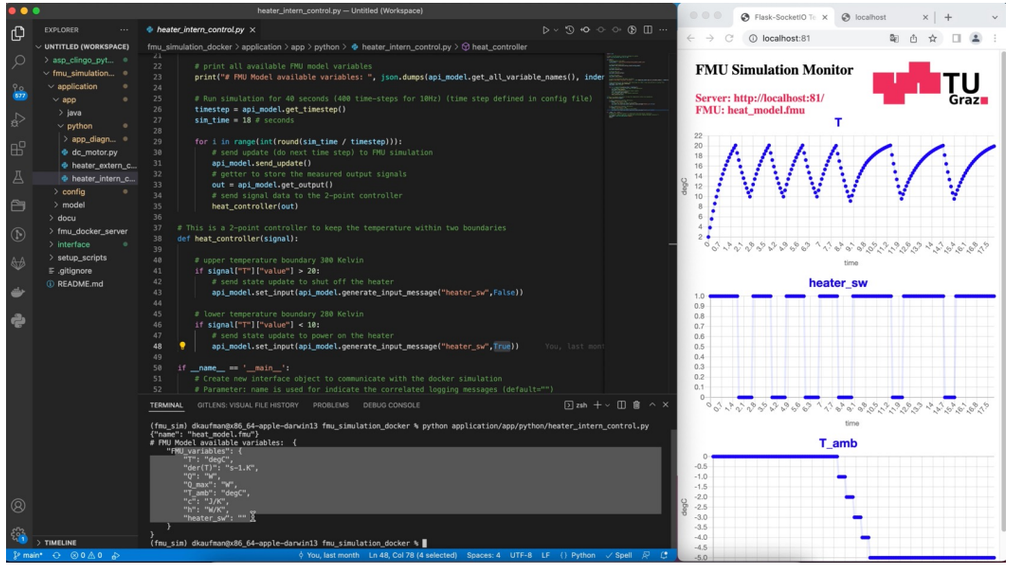
Start Control

Name: control

Address: localhost:82

heater_extern_control.py

Heater Simulation Controller



ASP Diagnose Tool

Introduction and Example

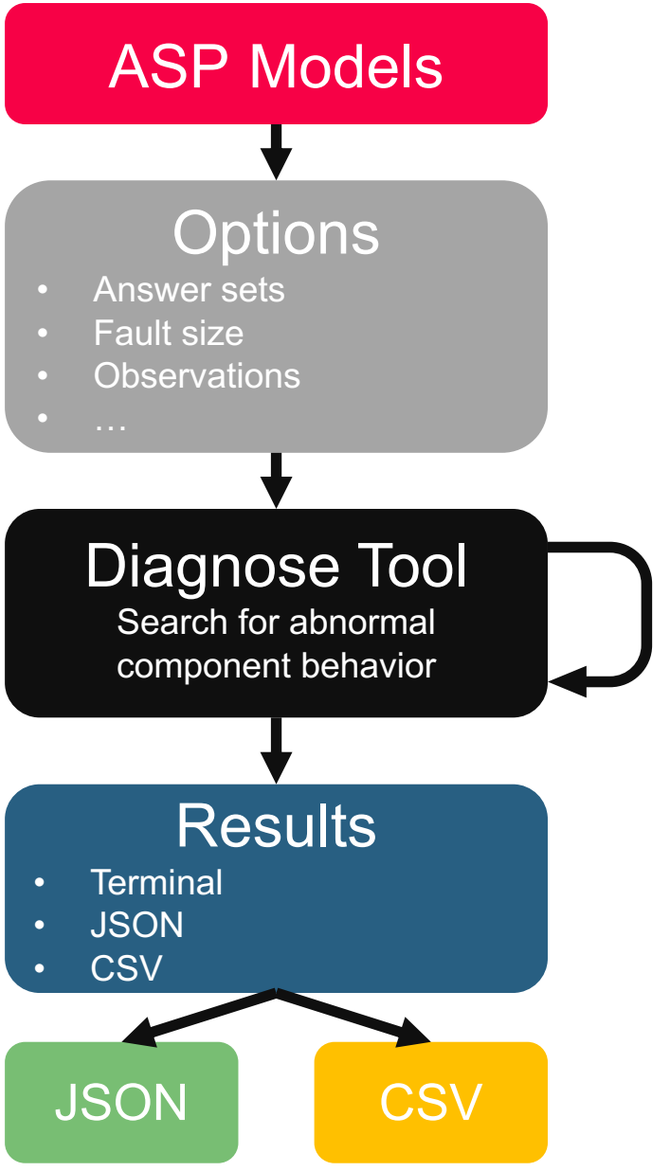
ASP Diagnose Tool

- Clone GitHub Repository: https://github.com/QAMCAS/ASP-Diagnose-Tool

- Answer Set Programming (ASP) model diagnose tool
- Theorem solver: **Clingo 5.4.1**
- Modeling cyber physical systems in ASP
- Searching for abnormal behavior of components

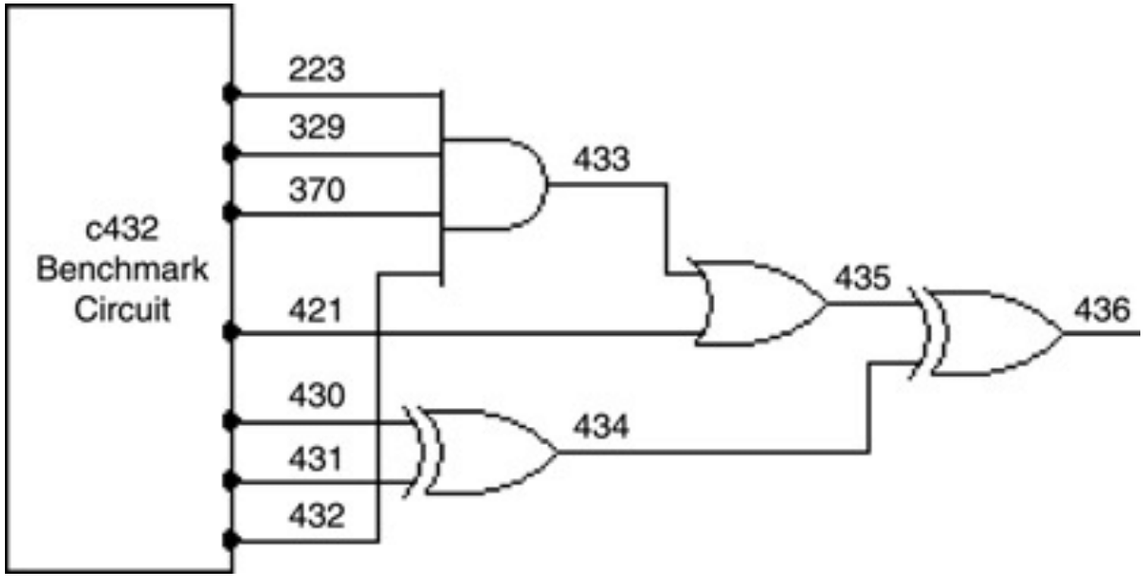
- **Setup options:**
  - ASP File or directory to perform diagnose
  - Number of required answer sets
  - Definition of fault size (e.g.: 2 -> 0 (as reference), 1, 2 searched)
  - Automatic constraint adding after each diagnose run
  - Additional observation file

- **Result options:**
  - **Terminal**
    - file, fault size, observations, diagnose, time for diagnose
  - **JSON**
    - index, file, fault size, observations, diagnose, time for diagnose
  - **CSV**
    - index, file, time for diagnose, number of diagnosis found, total time



Benchmark Circuits ISCAS85

- c432 : 27-channel interrupt controller
- c499/c1355 : 32-bit SEC circuit
- c880 : 8-bit ALU
- c1908 : 16-bit SEC/DED circuit
- c2670 : 12-bit ALU and controller
- c3540 : 8-bit ALU
- c5315 : 9-bit ALU
- c6288 : 16x16 multiplier
- c7552 : 32-bit adder/comparator



Das, Sunil & Hossain, Altaf & Biswas, Satyen & Petriu, Emil. (2008). Aliasing-free compaction revisited. Circuits, Devices & Systems, IET. 2. 166 - 178. 10.1049/iet-cds:20070119.

Benchmark Circuits ISCAS85

Results

Index	Model	Time 0	Time 1	Time 2	Time 3	NoD 0	NoD 1	NoD 2	NoD 3	ASPDiagTime
0	data/asp_test_data/c432_tc_1_1 2.pl	0,046320	0,034784	0,034225	0,056761	0	1	3	36	0,172136
0	data/asp_test_data/c499_tc_2_16.pl	0,051533	0,053972	0,047338	0,053858	0	0	0	0	0,206722
0	data/asp_test_data/c432_tc_3_6 2.pl	0,028938	0,032408	0,035814	0,089930	0	0	0	91	0,187106
0	data/asp_test_data/c432_tc_2_6.pl	0,030214	0,034715	0,029340	0,031691	0	0	0	0	0,125973
0	data/asp_test_data/c432_tc_2_1 2.pl	0,028843	0,031838	0,037406	0,084655	0	0	7	102	0,182760
0	data/asp_test_data/c499_tc_1_20.pl	0,049613	0,121508	0,055968	0,055822	0	90	0	0	0,282931


```
"data/asp_test_data/c432_tc_1_1 2.pl": [
{
        "model status": "model satisfied",
        "diag time": 0.0359950065612793,
        "fault size": 1,
        "diag found": 1,
        "diag":
        [
                [
                "ab_199gat"
                ]
        ],
        "observation": []
},
. . .
```



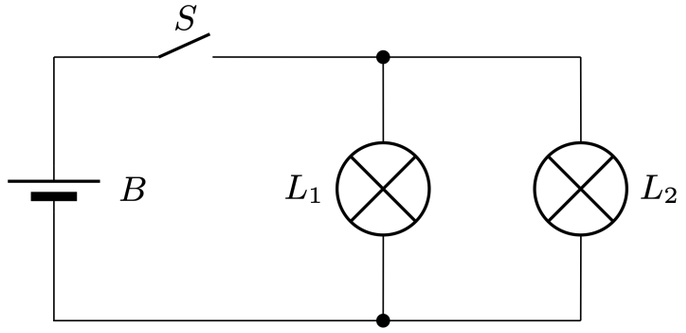
```
{
        "model status": "model satisfied",
        "diag time": 0.03448033332824707,
        "fault size": 2,
        "diag found": 3,
        "diag":
        [
                [
                "ab_223gat","ab_296gat"
                ],
                [
                "ab_203gat","ab_223gat"
                ],
                [
                "ab_223gat","ab_329gat"
                ]
        ],
        "observation": []
},
. . .
```

FMU Simulation Tool & ASP Diagnose Tool

Joint Example

Use Case – Two Lamps
FMU Model

- OpenModelica OMEdit
- Electrical circuit model
- Components:
  - Battery
  - Switch
  - Lamp 1 & 2
- Component fault types
- **FMU generation**



Component	*State*	*Description*
light bulb (`bulb`), switch (`sw`)	`ok`	ordinary behaviour
	`broken`	open connection in eletrical circuit
	`short`	short in the electrical circuit
battery (`bat`)	`ok`	ordinary behaviour
	`empty`	empty battery fault

circuit components & connections

```
model Two_Lamp_Circuit
  PhysicalFaultModeling.PFM_Bulb bulb1(r = 100.0);
  PhysicalFaultModeling.PFM_Bulb bulb2(r = 100.0);
  PhysicalFaultModeling.PFM_Switch sw;
  PhysicalFaultModeling.PFM_Ground gnd;
  PhysicalFaultModeling.PFM_Battery bat(vn = 5.0);
equation
  connect(gnd.p, bat.m);
  connect(bat.p, sw.p);
  connect(sw.m, bulb1.p);
  connect(sw.m, bulb2.p);
  connect(bulb1.m, gnd.p);
  connect(bulb2.m, gnd.p);
end Two_Lamp_Circuit;
```

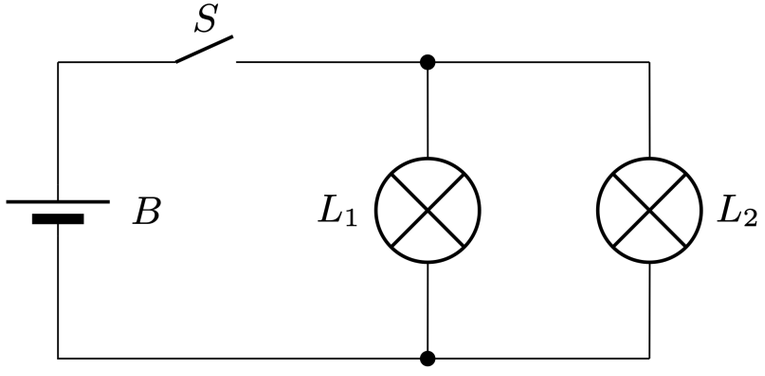
define components state input

```
model Two_Lamp_Circuit_Testbench
  PhysicalFaultModeling.Two_Lamp_Circuit sut;
  input FaultType bat_state(start=FaultType.ok);
  input OperationalMode
switch_mode(start=OperationalMode.close);
  input FaultType switch_state(start=FaultType.ok);
  input FaultType bulb1_state(start=FaultType.ok);
  input FaultType bulb2_state(start=FaultType.ok);
equation
  sut.sw.mode = switch_mode;
  sut.bat.state = bat_state;
  sut.sw.state = switch_state;
  sut.bulb1.state = bulb1_state;
  sut.bulb2.state = bulb2_state;
end Two_Lamp_Circuit_Testbench;
```

Use Case – Two Lamps

ASP Model

- Answer Set Programming
- Model behavior in first order logic representation
- Components:
  - Battery
  - Switch
  - Lamp 1 & 2
- Component connection



Lamp state (on, off) logic

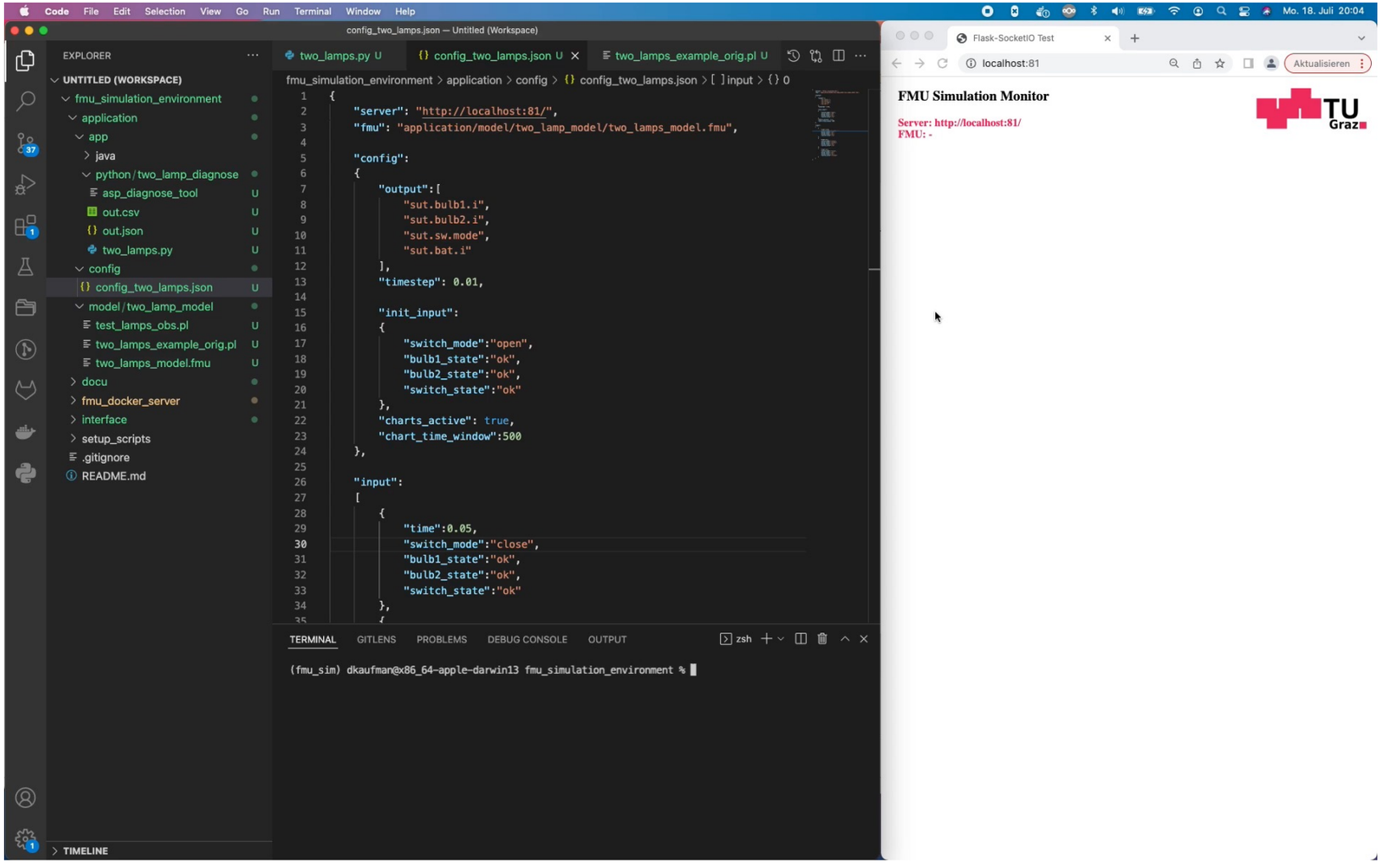
```
val(light(X),on) :- type(X,lamp), val(in_pow(X),nominal), nab(X).
val(in_pow(X),nominal) :- type(X,lamp), val(light(X),on).
val(light(X),off) :- type(X,lamp), val(in_pow(X),zero), nab(X).
```

Component connection logic

```
val(X,V) :- conn(X,Y), val(Y,V).
val(Y,V) :- conn(X,Y), val(X,V).
:- val(X,V), val(X,W), not V=W.

type(b, bat).
type(s, sw).
type(l1, lamp).
type(l2, lamp).

conn(in_pow(s), pow(b)).
conn(out_pow(s), in_pow(l1)).
conn(out_pow(s), in_pow(l2)).
```

Use Case – Two Lamps

Demo



SCIENCE
PASSION
TECHNOLOGY

Thank you for your attention!

David Kaufmann,

18/04/2023

David.Kaufmann@ist.tugraz.at