OVERTURE FMU: EXPORT VDM-RT MODELS AS TOOL-WRAPPER FMUS

AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

THE 15TH OVERTURE WORKSHOP
14 JULY 2018

CASPER THULE
PHD STUDENT

AGENDA

Me

Cyber-Physical Systems

Co-simulation

VDM-RT

INTO-CPS

FMI

EXAMPLE (Water tank)

Architecture of Overture FMU

Concluding Remarks

CASPER THULE

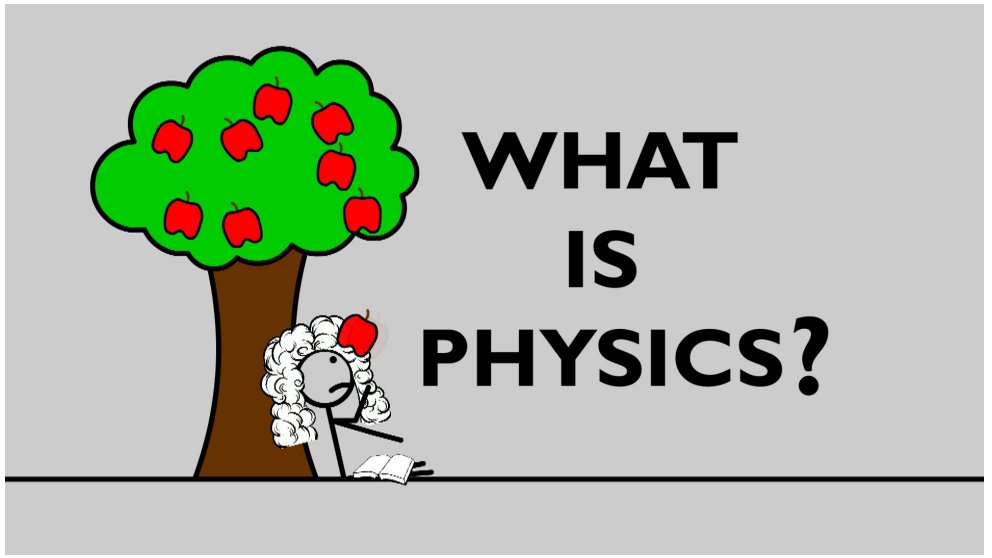
MSc. in Computer Technology, Aarhus University, 2016

PhD Student at Aarhus University - Department of Engineering

Software Engineering Research Group led by Professor Peter Gorm Larsen

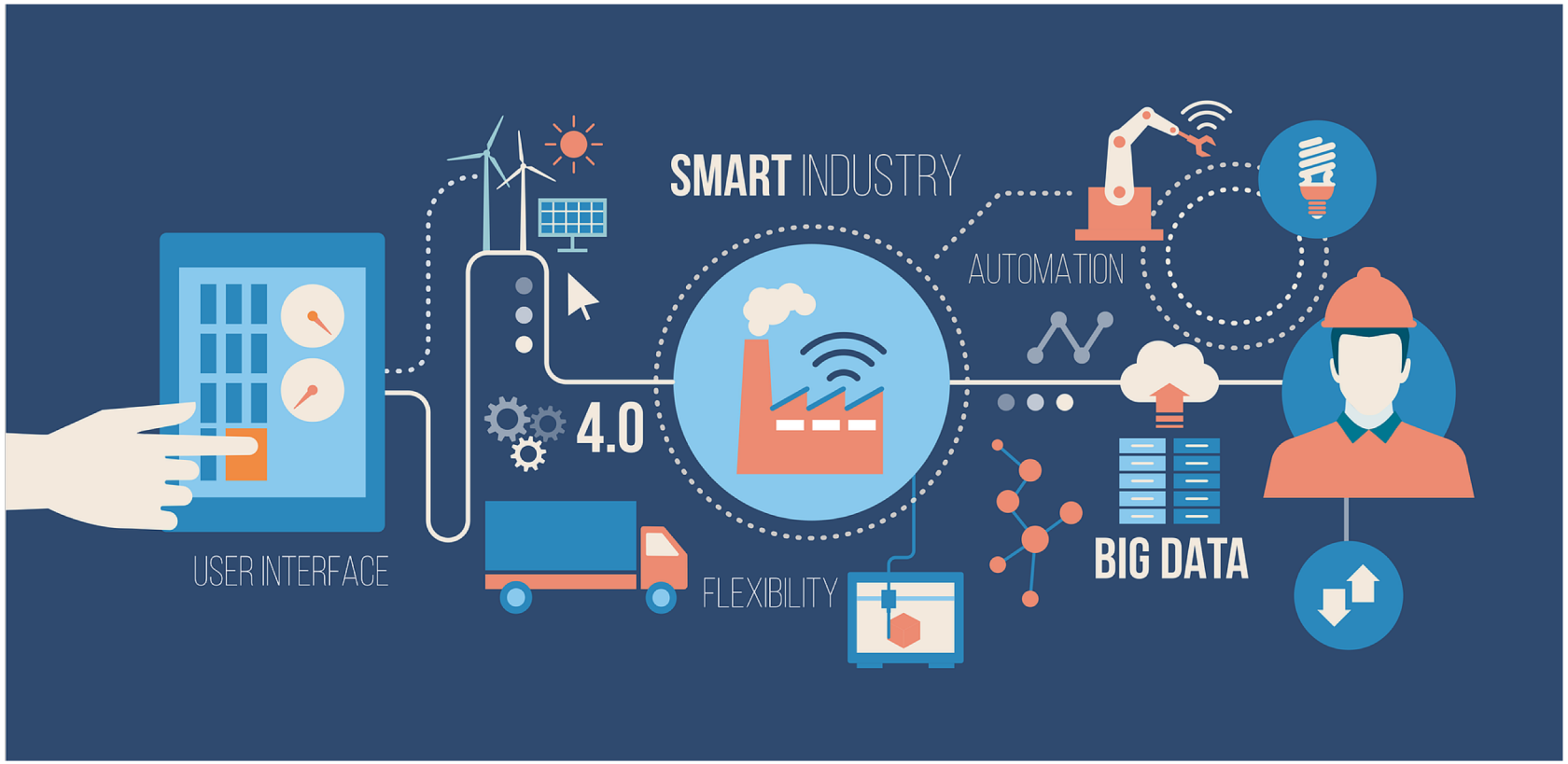
Worked on the INTO-CPS Project 2015 - 2017

CYBER-PHYSICAL SYSTEMS



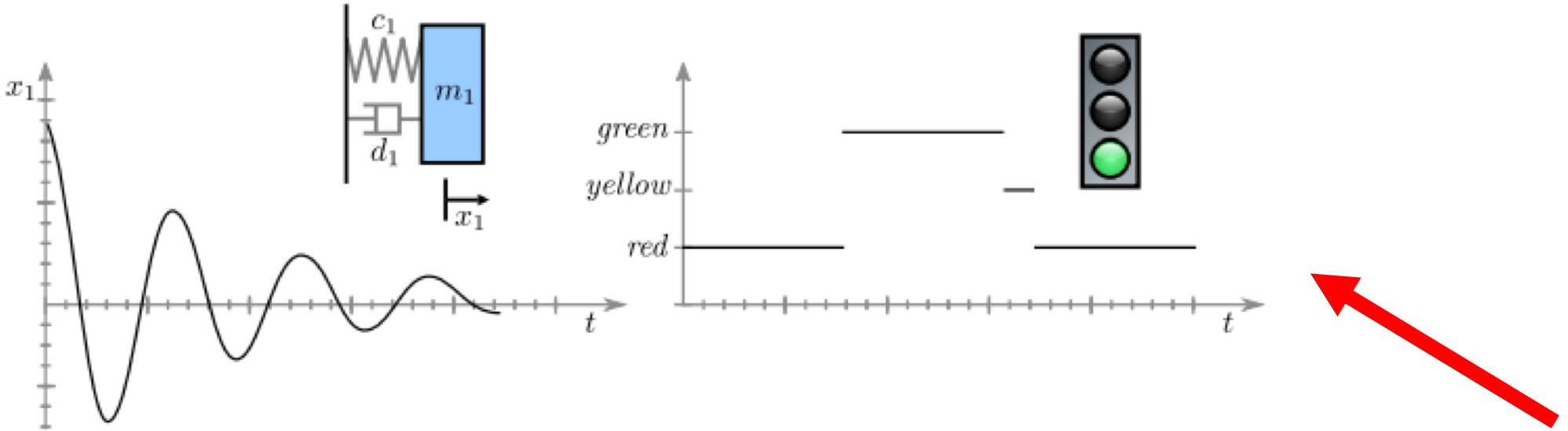
CYBER-PHYSICAL SYSTEMS COOL

Cyber Components Controlling Physical Entities



COLLABORATIVE/COUPLED SIMULATION

Hybrid Co-Simulation



Theory and Techniques for Global Simulation of a Coupled System via Composition of Simulators

VDM-RT

VDM-RT is an object-oriented VDM dialect with real-time and distribution constructs

2009: Initial work on Co-Simulation using VDM-RT and 20-sim (Marcel Verhoef PhD)

2010-2013: DESTECS project – Crescendo tool. VDM-RT, 20-Sim, XML-RPC

2015-2017: INTO-CPS. Export VDM-RT Models as simulation units (FMUs)

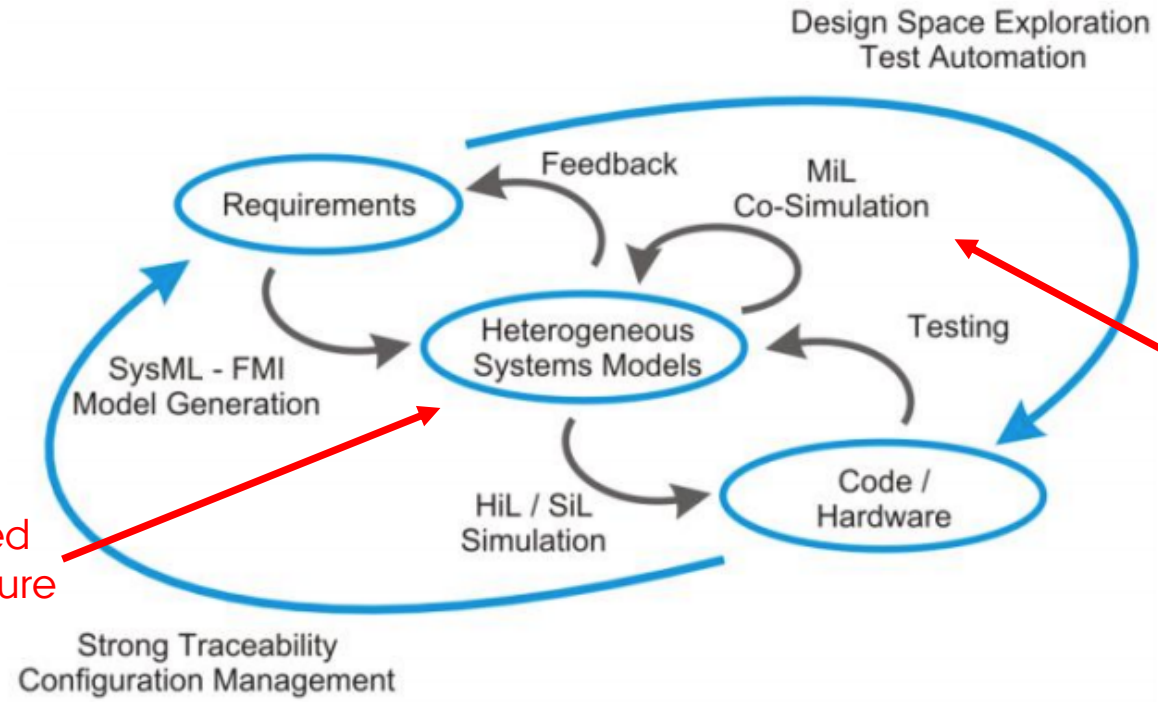
```
cpu1 : CPU := new CPU(<FP>, 200);

controller := new Controller(levelSensor, valveActuator);

cpu1.deploy(controller,"Controller");
```

```
loop()==
   cycles(2)
   let level : real = levelSensor.getLevel() in ...
```

```
thread
periodic(10E6,0,0,0)(loop);
```

INTO-CPS 2015-2017

Integrated Tool Chain For Model-based Design of Cyber-Physical Systems



Simulation unit produced with VDM-RT and Overture FMU

Co-Simulation of simulation units

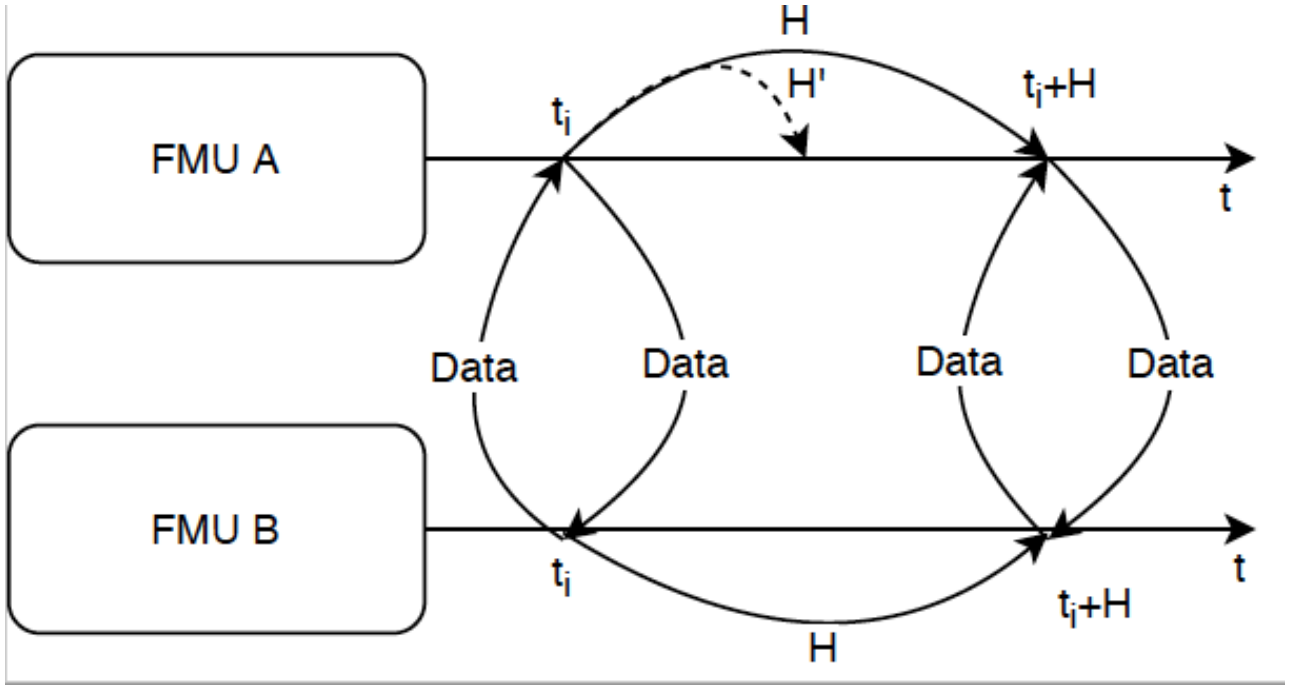
FUNCTIONAL MOCK-UP INTERFACE 2.0

C interface

- Set input / Get Output
- Do Step – Progress in time
- Set state / Get state
- Extension: GetMaxStepSize

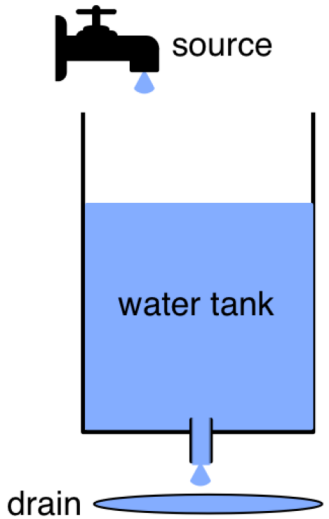
Usage constraints

Packaging

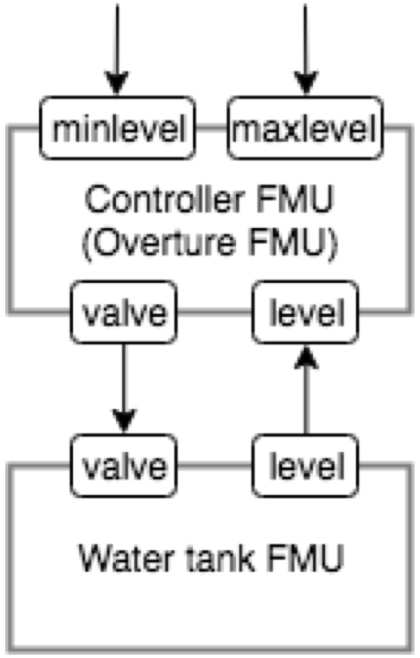


EXAMPLE

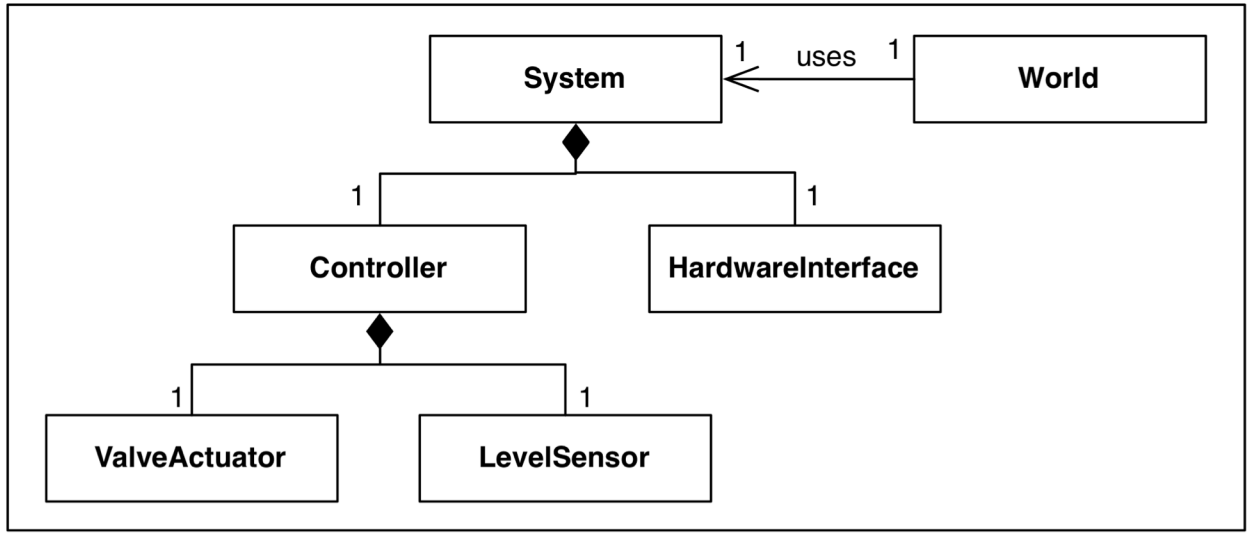
System



Co-Simulation Structure



VDM-RT Model architecture



```
1 system System
2
3 instance variables
4   -- Hardware interface variable required by FMU Import/Export
5   public static hwi: HardwareInterface := new HardwareInterface
6
7   public levelSensor : LevelSensor;
8   public valveActuator : ValveActuator;
9   public static controller : [Controller] := nil;
10
11  cpu1 : CPU := new CPU(<FP>, 200);
12 operations
13
14  public System : () ==> System
15  System () ==
16  (
17    levelSensor := new LevelSensor(hwi.level);
18    valveActuator :=  new ValveActuator(hwi.valveState);
19
20    controller := new Controller(levelSensor, valveActuator);
21
22    cpu1.deploy(controller,"Controller");
23  );
24 end System
```

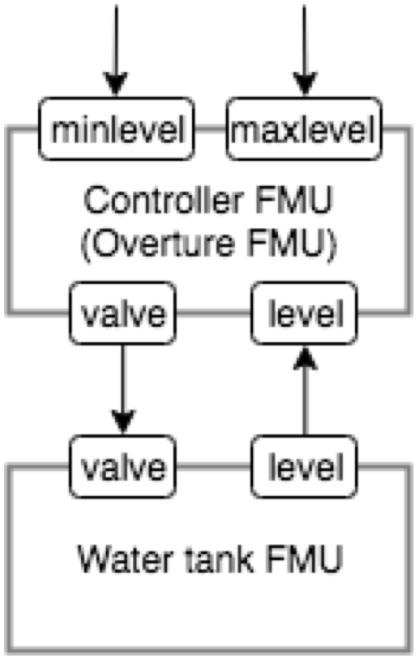
```
20 public Controller : LevelSensor * ValveActuator  ==> Controller
21 Controller(l,v)==
22 (
23   levelSensor   := l;
24   valveActuator := v;
25  );
26
27 private loop : () ==>()
28 loop()==
29   cycles(2)
30   (let level : real = levelSensor.getLevel() in (
31       if( level >= HardwareInterface`maxlevel.getValue())
32       then valveActuator.setValve(open);
33
34       if( level <= HardwareInterface`minlevel.getValue())
35       then valveActuator.setValve(close);
36     ));
37
38 thread periodic(10E6,0,0,0)(loop);
```

DANGEROUS! Prevents overlapping errors!

Two cycles = 10 milliseconds

$$\tau = cycles/freq_{CPU} = 2/200Hz = 0.01seconds$$

HARDWARE INTERFACE




```
 8 class HardwareInterface
 9
10 values
11   -- @ interface: type = parameter, name="minlevel";
12   public minlevel : RealPort = new RealPort(1.0);
13   -- @ interface: type = parameter, name="maxlevel";
14   public maxlevel : RealPort = new RealPort(2.0);
15
16 instance variables
17   -- @ interface: type = input, name="level";
18   public level : RealPort := new RealPort(0.0);
19
20 instance variables
21   -- @ interface: type = output, name="valve";
22   public valveState : BoolPort := new BoolPort(false);
23
24 end HardwareInterface
```

SYNCHRONISATION

FMI SYNCHRONISATION: When outputs are exchanged

OVERTURE FMU: Just before reading a value, just after setting a value

```
27 private loop : () ==>()
28 loop()==
29   -- SYNCHRONISATION
30   let level : real = levelSensor.getLevel() in (
31
32     if( level >= HardwareInterface`maxlevel.getValue())
33     then valveActuator.setValve(open);
34     -- SYNCHRONISATION if condition yields true
35
36     if( level <= HardwareInterface`minlevel.getValue())
37     then valveActuator.setValve(close);
38     -- SYNCHRONISATION if condition yields true
39 );
```

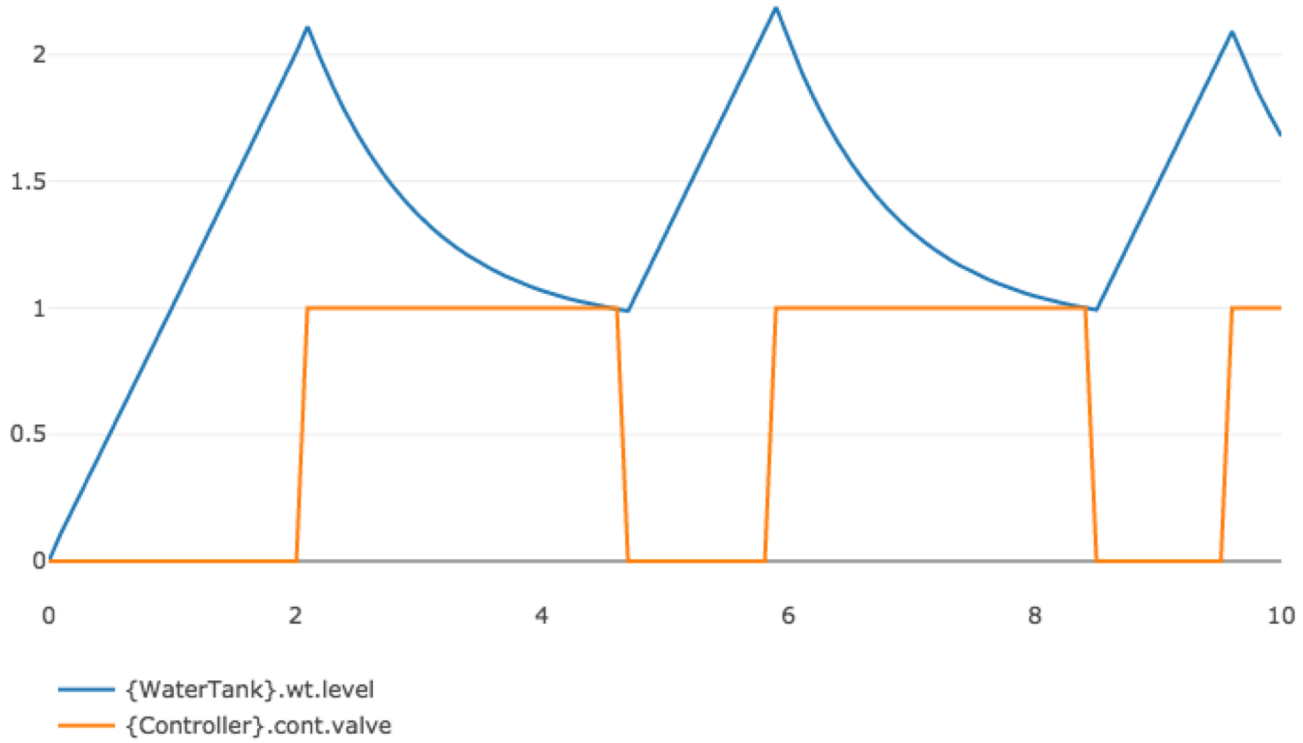
Transactions: getMaxStepSize: $min(\{\tau \mid (\Sigma, \tau) \in T\}) - \tau_{now}$

Σ is state, τ is time, (Σ, τ) is a transaction pair of state to expose at time τ

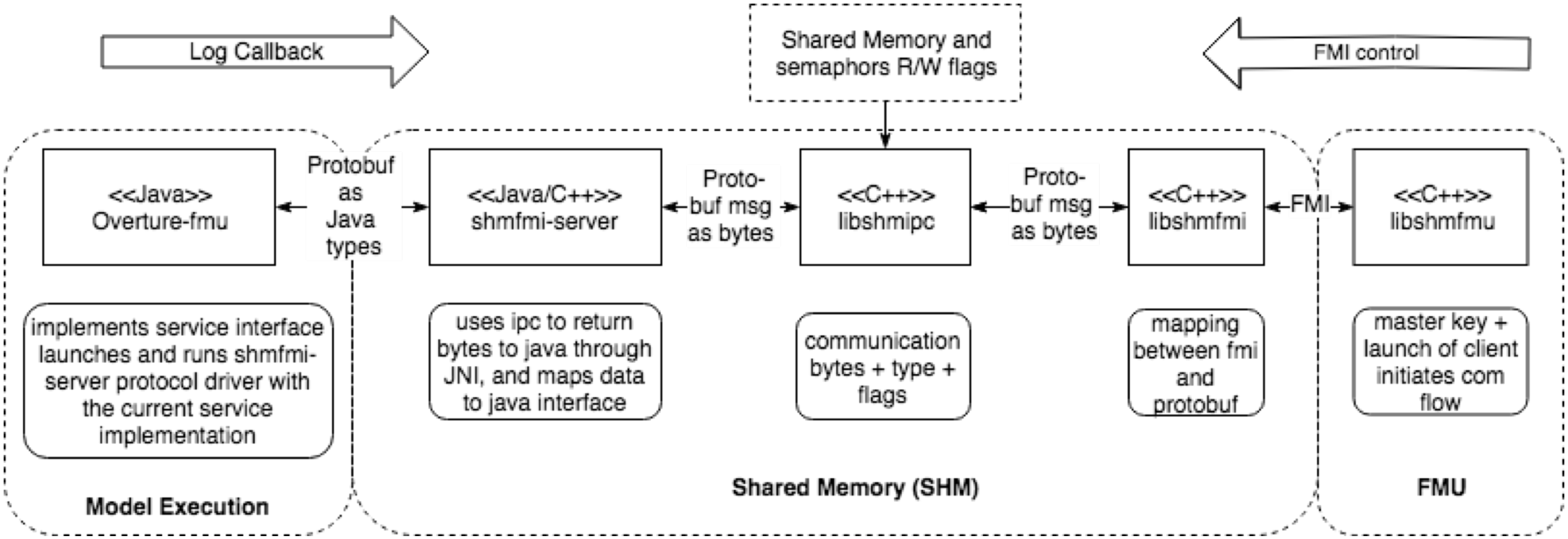
T is a set of all transactions across CPUs, and τ_{now} is the global current time of the co-simulation.

CO-SIMULATION RESULT

Delay is due to Jacobi algorithm.



HOW DOES THE MAGIC HAPPEN?



CONCLUDING REMARKS

Tool-wrapper FMU with Overture

Synchronisation

Step size calculation

Architecture

Successfully reused logic from Crescendo

Successfully added another FMU development language

Requires Java

Performance

Has been successfully applied to industry case studies in the INTO-CPS Project

AARHUS
UNIVERSITY