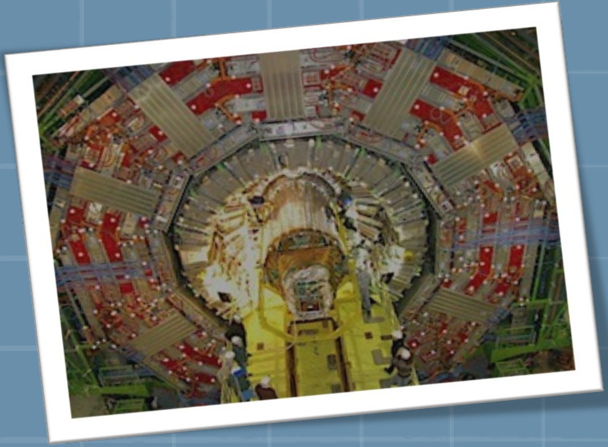
RealTime and Timing

What is a Real Time System?

- **Wikipedia**
  - **A system is said to be “real-time” if the correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed. Real-time systems are classified by the consequence of missing a deadline.**
- **Classifications:**
  - **Hard: Missing a deadline is a total system failure.**

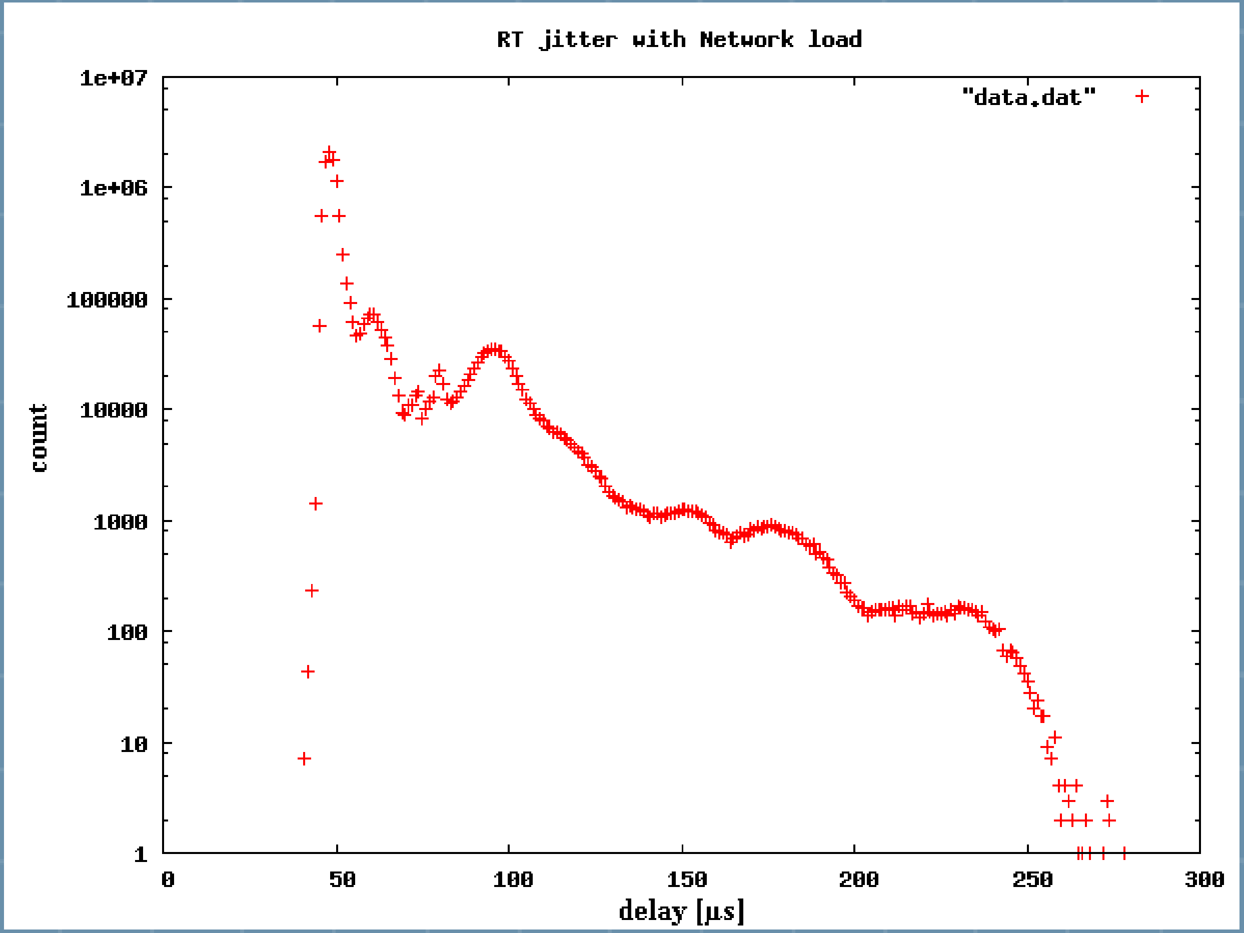
    → use hardware e.g. a FPGA, use FESA to configure the hardware
  - **Soft: The usefulness of a result degrades after its deadline, thereby degrading the system's quality of service.**

    → use FESA



Performance
FESA + RT-Linux

Time between receiving hardware-trigger and execution of a RT-Action



- 55h
- 10M measurements
- 1MB/sec network load
- 10MB/sec filesystem load
- used FESA v. 3.0-beta

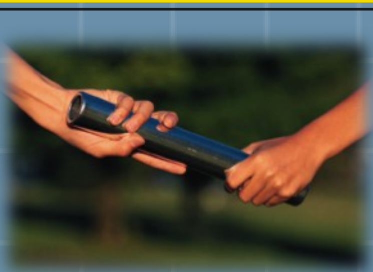
The Mission

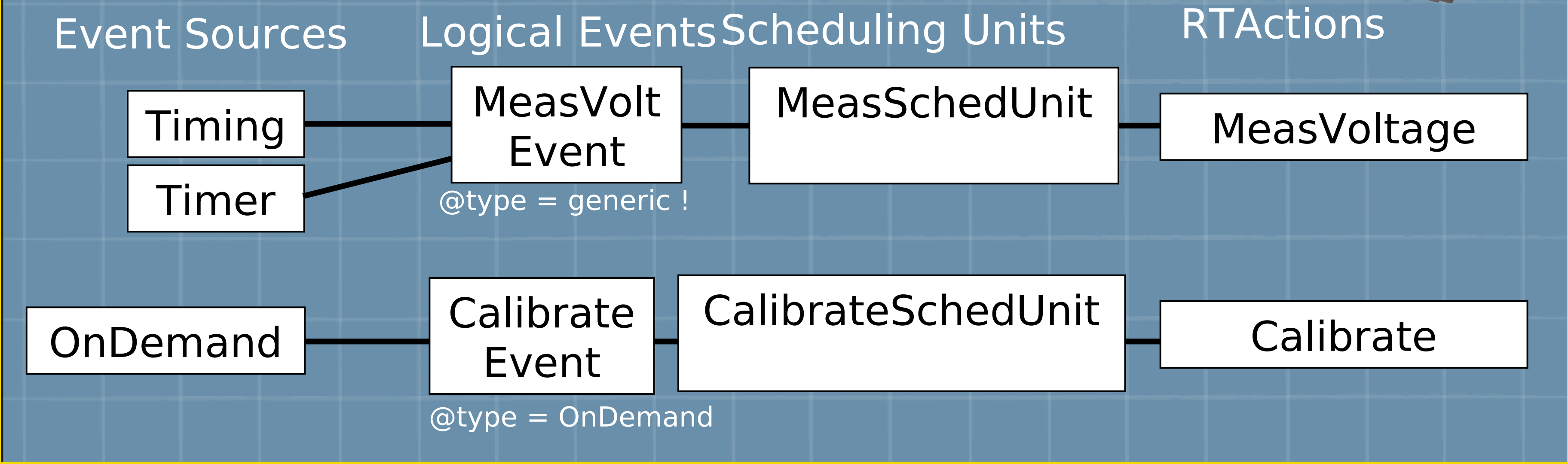
- Measure a Voltage
- Measurement of “Device1” triggered by Timing
- Measurement of “Device2” triggered by Timer
- Calibration of the device can be done by client-request

On any problems: fesa-support@gsi.de

What elements we need?

DESIGN





Event Sources

Logical Events

Scheduling Units

RTActions

Timing

Timer

MeasVolt Event

@type = generic !

MeasSchedUnit

MeasVoltage

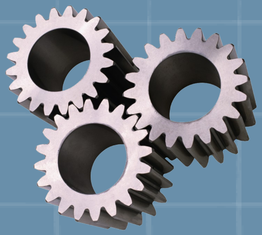
OnDemand

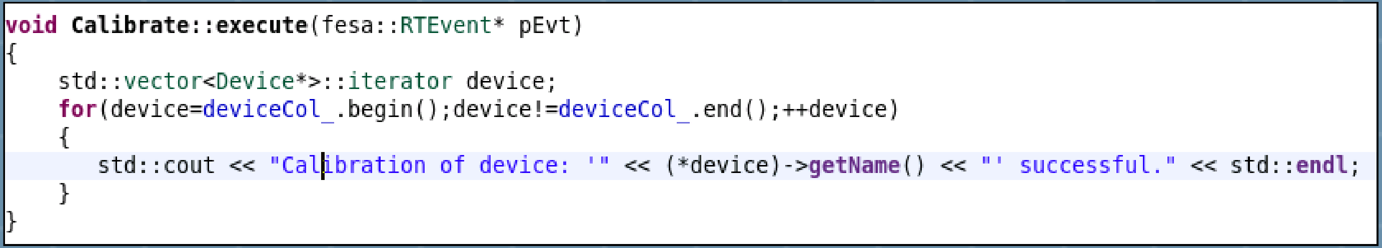
Calibrate Event

@type = OnDemand

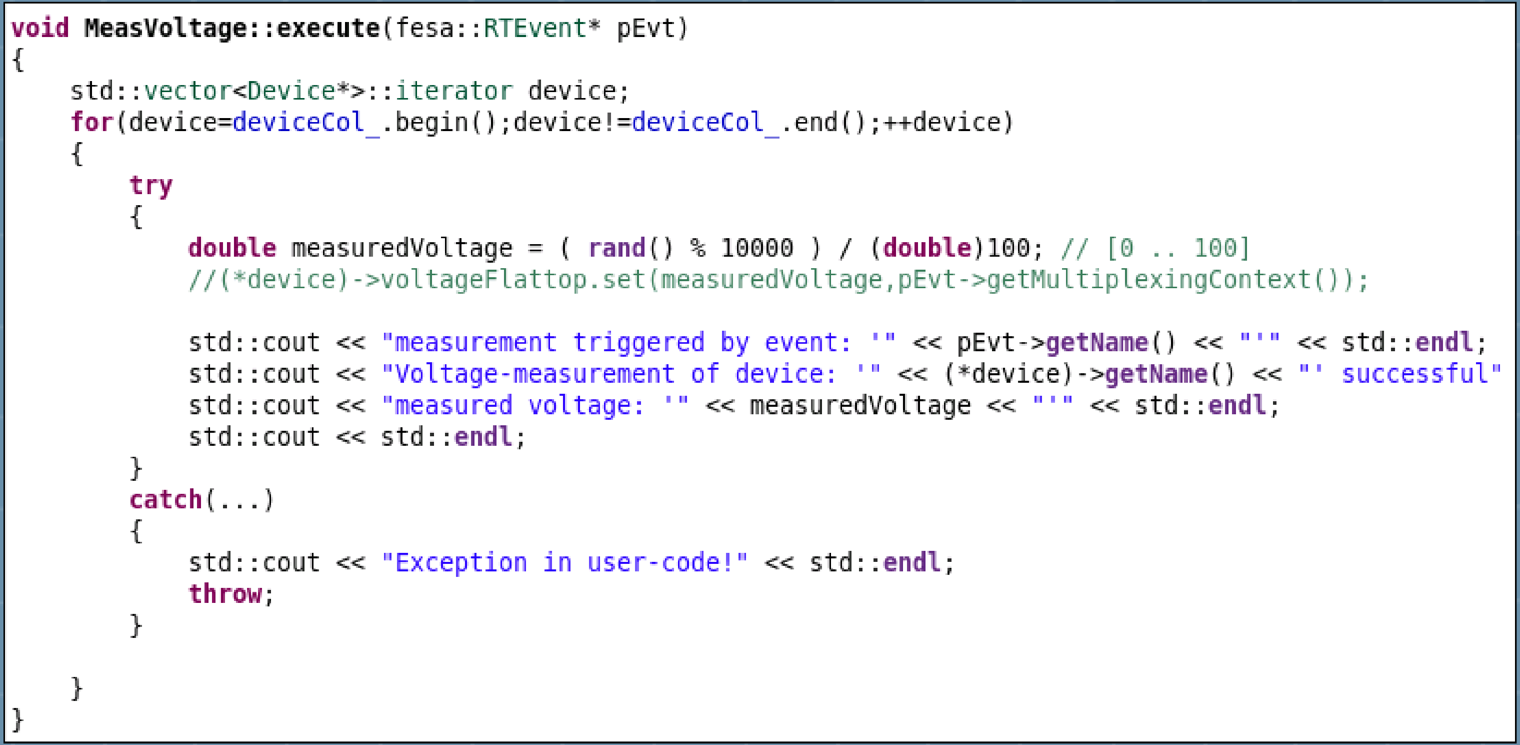
CalibrateSchedUnit

Calibrate





```
void Calibrate::execute(fesa::RTEvent* pEvt)
{
    std::vector<Device*>::iterator device;
    for(device=deviceCol_.begin();device!=deviceCol_.end();++device)
    {
        std::cout << "Calibration of device: '" << (*device)->getName() << "' successful." << std::endl;
    }
}
```



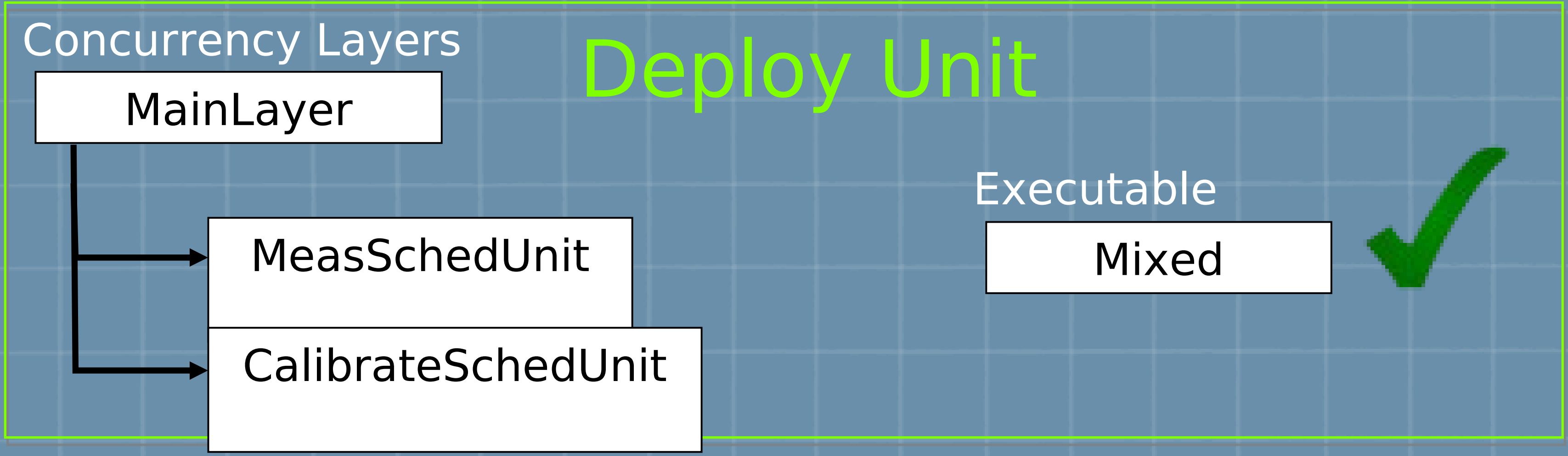
```
void MeasVoltage::execute(fesa::RTEvent* pEvt)
{
    std::vector<Device*>::iterator device;
    for(device=deviceCol_.begin();device!=deviceCol_.end();++device)
    {
        try
        {
            double measuredVoltage = ( rand() % 10000 ) / (double)100; // [0 .. 100]
            //(*device)->voltageFlattop.set(measuredVoltage,pEvt->getMultiplexingContext());

            std::cout << "measurement triggered by event: '" << pEvt->getName() << "'" << std::endl;
            std::cout << "Voltage-measurement of device: '" << (*device)->getName() << "' successful"
            std::cout << "measured voltage: '" << measuredVoltage << "'" << std::endl;
            std::cout << std::endl;
        }
        catch(...)
        {
            std::cout << "Exception in user-code!" << std::endl;
            throw;
        }

    }
}
```

TIP: Use dev + Ctrl + Space + deviceCollection = Skeleton

Exercise 1: Class Design

- Create a new class “MyVoltmeter”
- Add a Timer, Timing and an On-Demand event-source and two logical events:
  - “MeasVoltEvent” (@type = generic)
  - “CalibrationEvent” (@type = on-demand)
- Create two Real Time Actions:
  - “MeasVoltage”
  - “Calibrate”
- Create a Command-Property
  - “Calibrate”
  - add a set-server-action “TriggerCalibration”
    - add the OnDemandSource as “triggered-event-source”
- Create two Scheduling Units that links the RT actions with the logical events.
- Generate the code
- Add the code in the execute method for the RT actions
- Compile the class

What elements we need?



Concurrency Layers

Deploy Unit

MainLayer

MeasSchedUnit

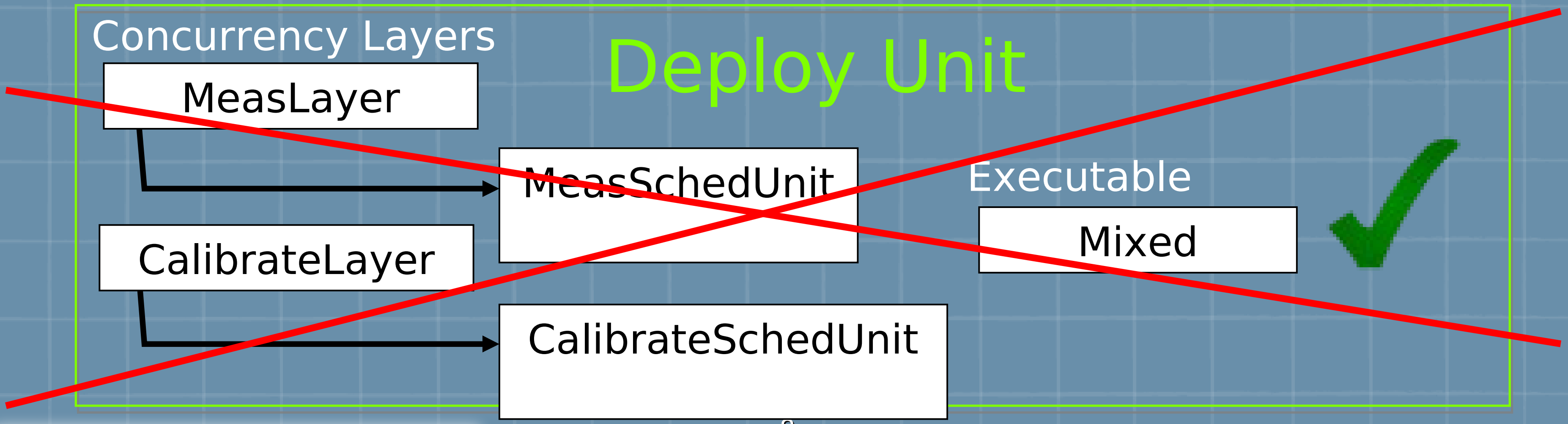
CalibrateSchedUnit

Executable

Mixed

✓

vs



Concurrency Layers

Deploy Unit

MeasLayer

MeasSchedUnit

CalibrateLayer

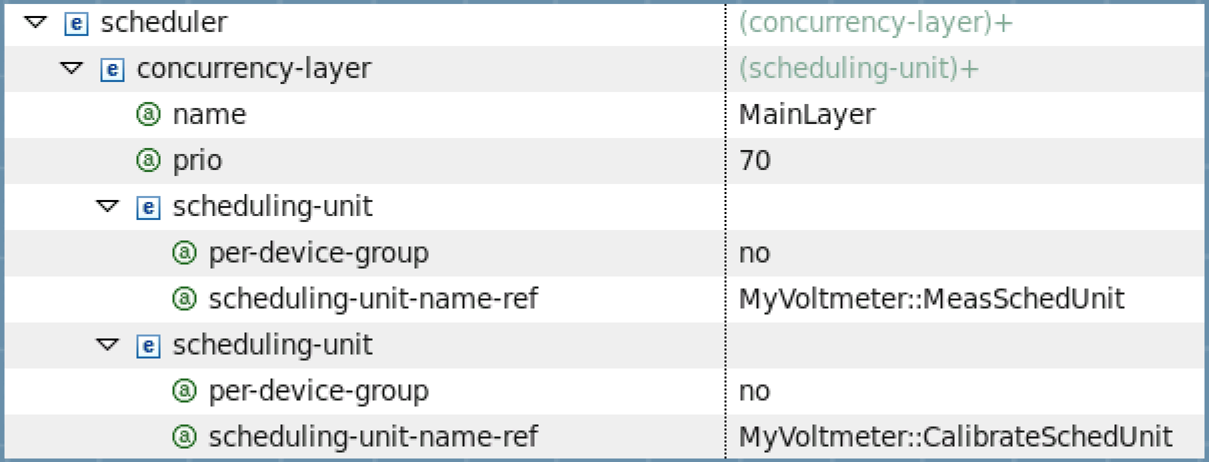
CalibrateSchedUnit

Executable

Mixed

✓

Scheduling Units & Scheduler

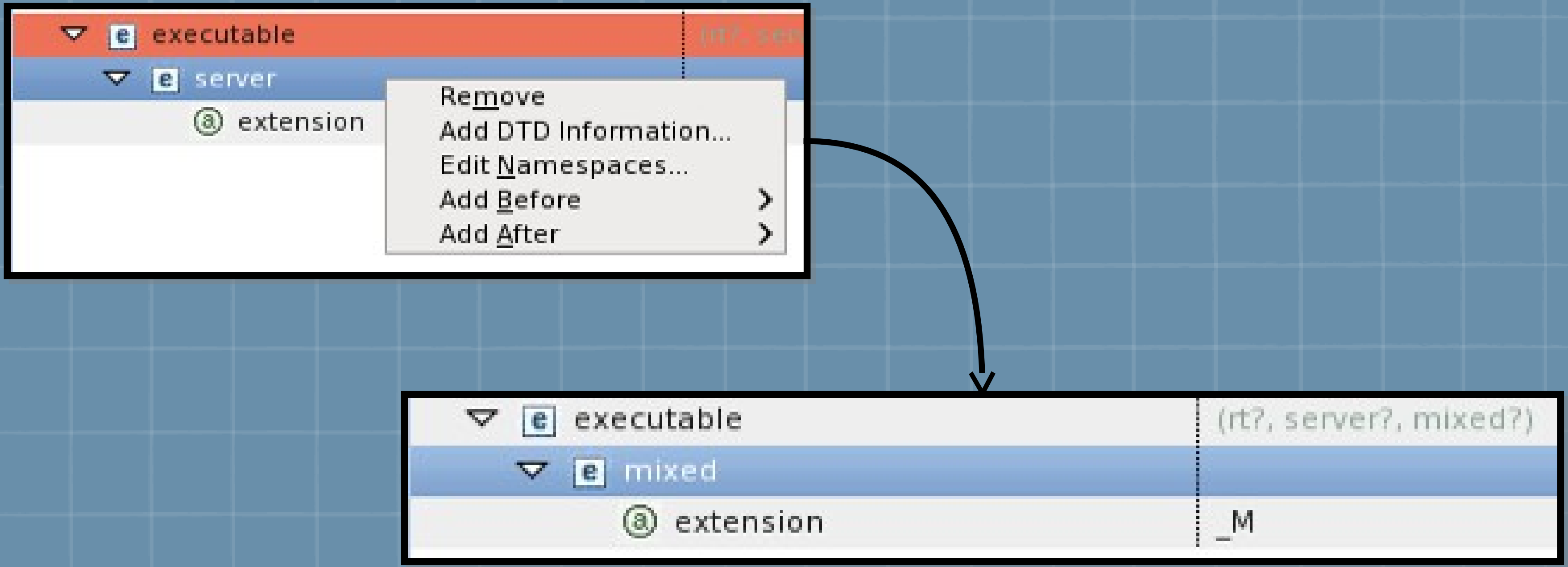


▽ e scheduler	(concurrency-layer)+
▽ e concurrency-layer	(scheduling-unit)+
ⓐ name	MainLayer
ⓐ prio	70
▽ e scheduling-unit	
ⓐ per-device-group	no
ⓐ scheduling-unit-name-ref	MyVoltmeter::MeasSchedUnit
▽ e scheduling-unit	
ⓐ per-device-group	no
ⓐ scheduling-unit-name-ref	MyVoltmeter::CalibrateSchedUnit

- **Each concurrency-layer describes one thread.**
- **per-device-group**
  - **yes = each device will get it's own RTAction-instance**
  - **no = devices which use the same concrete-event will share the same RTAction-instance**

Executable: Mixed

- Since we are working also with Real Time, the mixed executable is required instead of server-only.

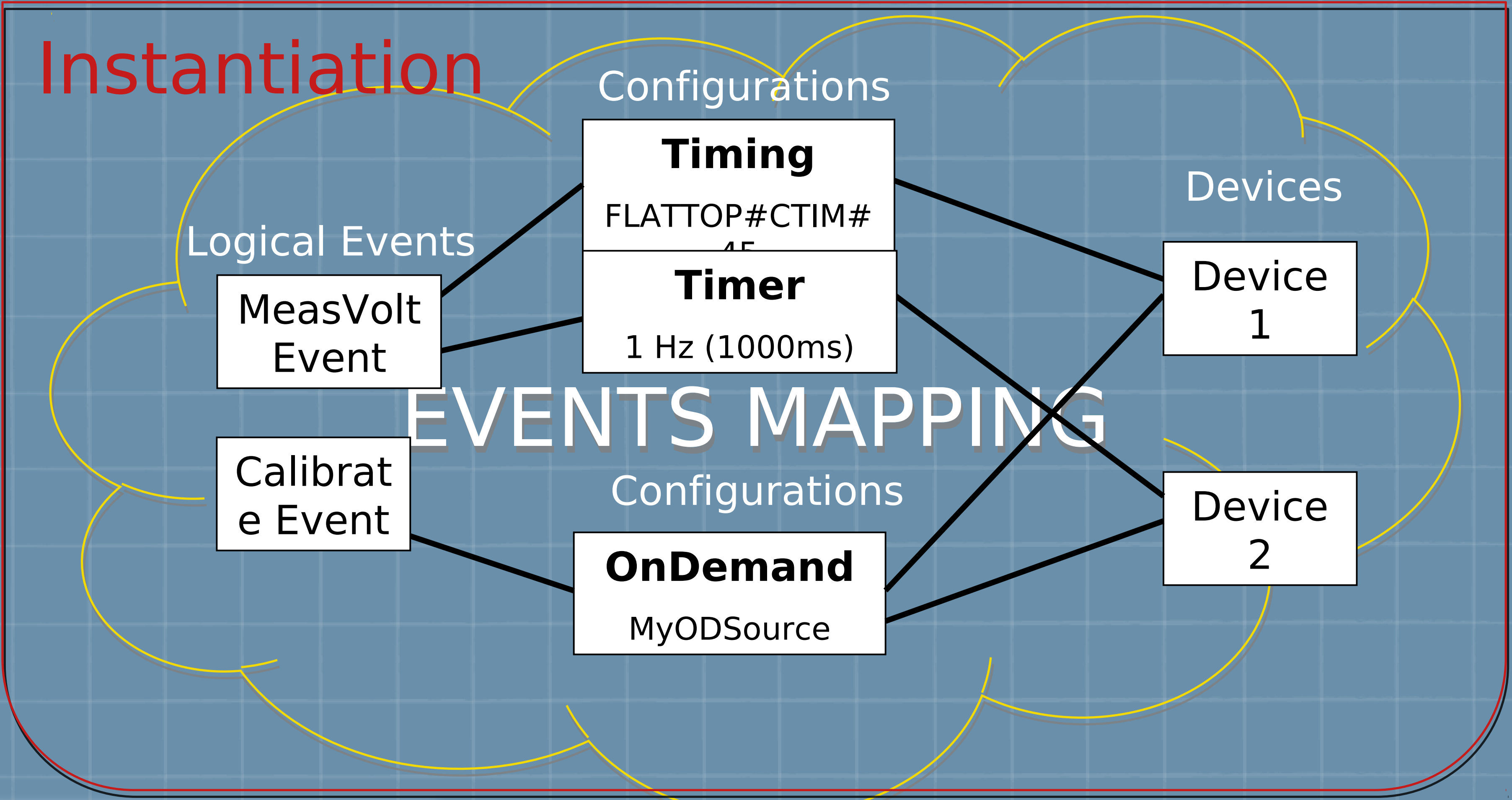


Exercise 2: Deploy Unit

- Create a Deploy-Unit named "MyVoltmeter_DU"
- Create a concurrency layer in order to schedul the two scheduling-units.
- Remove the server executable and add the mixed one.
- Generate the code & compile

On any problems: fesa-support@gsi.de

What elements we need?



Instantiation

EVENTS MAPPING

Logical Events

MeasVolt Event

Calibrate Event

Configurations

Timing

FLATTOP#CTIM#45

Timer

1 Hz (1000ms)

Configurations

OnDemand

MyODSource

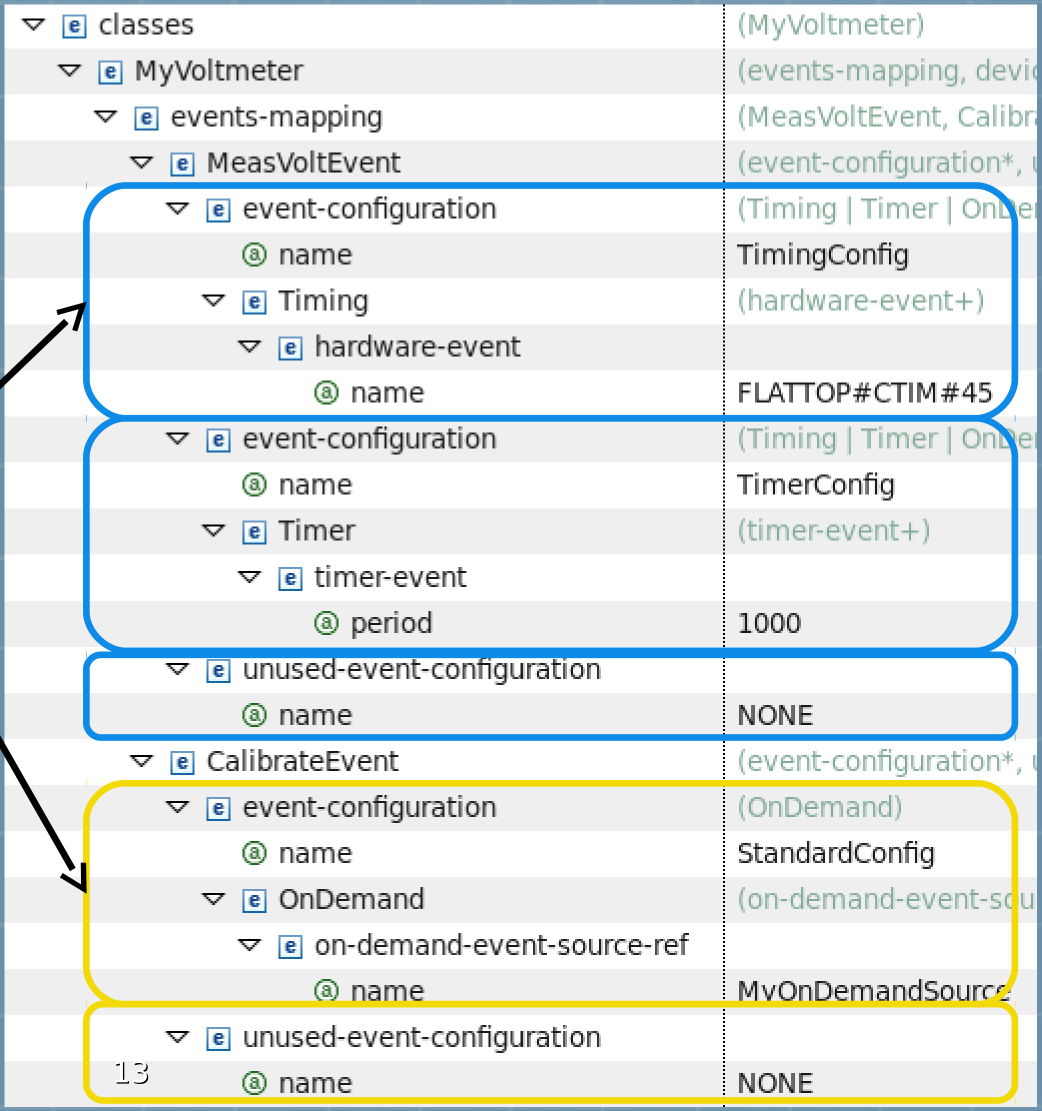
Devices

Device 1

Device 2

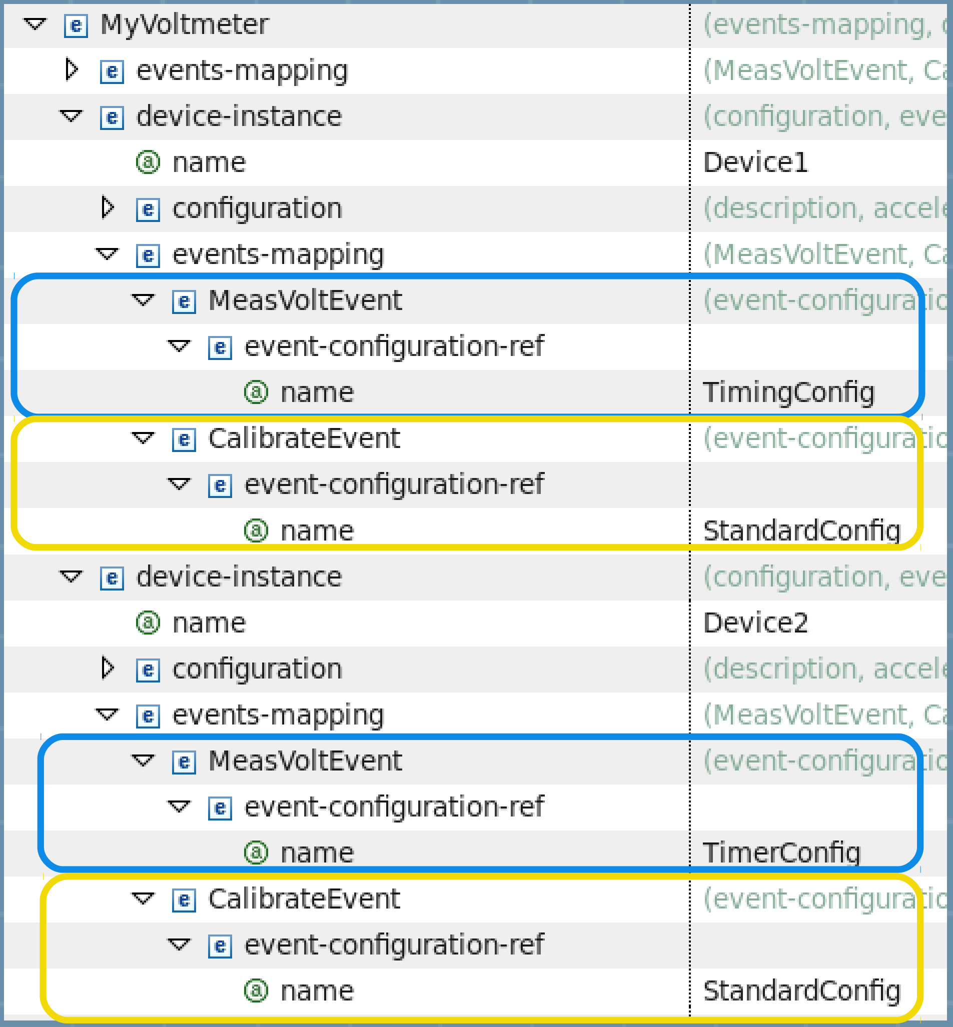
Event Mapping

Add any number of ***event-configurations*** **per logical event.**



```
▽ e classes                                 (MyVoltmeter)
  ▽ e MyVoltmeter                           (events-mapping, devic
    ▽ e events-mapping                      (MeasVoltEvent, Calibr
      ▽ e MeasVoltEvent                     (event-configuration*, u
        ▽ e event-configuration             (Timing | Timer | OnDe
            ⓐ name                          TimingConfig
          ▽ e Timing                        (hardware-event+)
            ▽ e hardware-event
                ⓐ name                      FLATTOP#CTIM#45
        ▽ e event-configuration             (Timing | Timer | OnDe
            ⓐ name                          TimerConfig
          ▽ e Timer                         (timer-event+)
            ▽ e timer-event
                ⓐ period                    1000
        ▽ e unused-event-configuration
            ⓐ name                          NONE
      ▽ e CalibrateEvent                    (event-configuration*, u
        ▽ e event-configuration             (OnDemand)
            ⓐ name                          StandardConfig
          ▽ e OnDemand                      (on-demand-event-sou
            ▽ e on-demand-event-source-ref
                ⓐ name                      MyOnDemandSource
        ▽ e unused-event-configuration
            ⓐ name                          NONE
```

Event Mapping

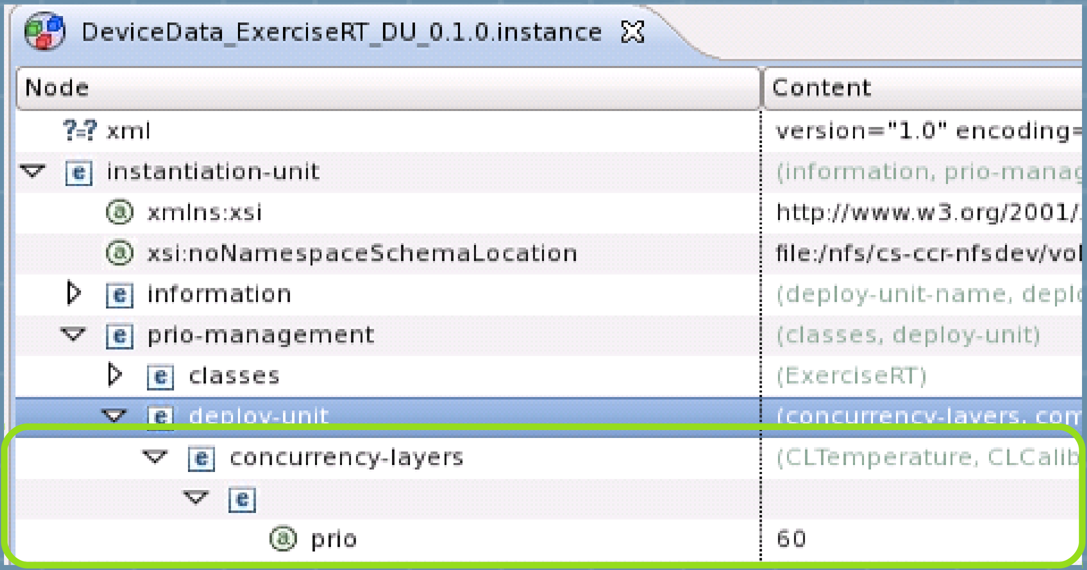


```
▽ e MyVoltmeter                               (events-mapping, d
    ▷ e events-mapping                        (MeasVoltEvent, Ca
    ▽ e device-instance                       (configuration, eve
          ⓐ name                              Device1
        ▷ e configuration                     (description, accele
        ▽ e events-mapping                    (MeasVoltEvent, Ca
            ▽ e MeasVoltEvent                 (event-configuratio
                ▽ e event-configuration-ref
                      ⓐ name                  TimingConfig
            ▽ e CalibrateEvent                (event-configuratio
                ▽ e event-configuration-ref
                      ⓐ name                  StandardConfig
    ▽ e device-instance                       (configuration, eve
          ⓐ name                              Device2
        ▷ e configuration                     (description, accele
        ▽ e events-mapping                    (MeasVoltEvent, Ca
            ▽ e MeasVoltEvent                 (event-configuratio
                ▽ e event-configuration-ref
                      ⓐ name                  TimerConfig
            ▽ e CalibrateEvent                (event-configuratio
                ▽ e event-configuration-ref
                      ⓐ name                  StandardConfig
```

Choose different event-configurations per device.

Priorities

- Priorities can be changed in the instantiation file
- Defaults can be given in the deployment-unit
- NICE-Scheduling vs. RR-Scheduling (–noRTSched)

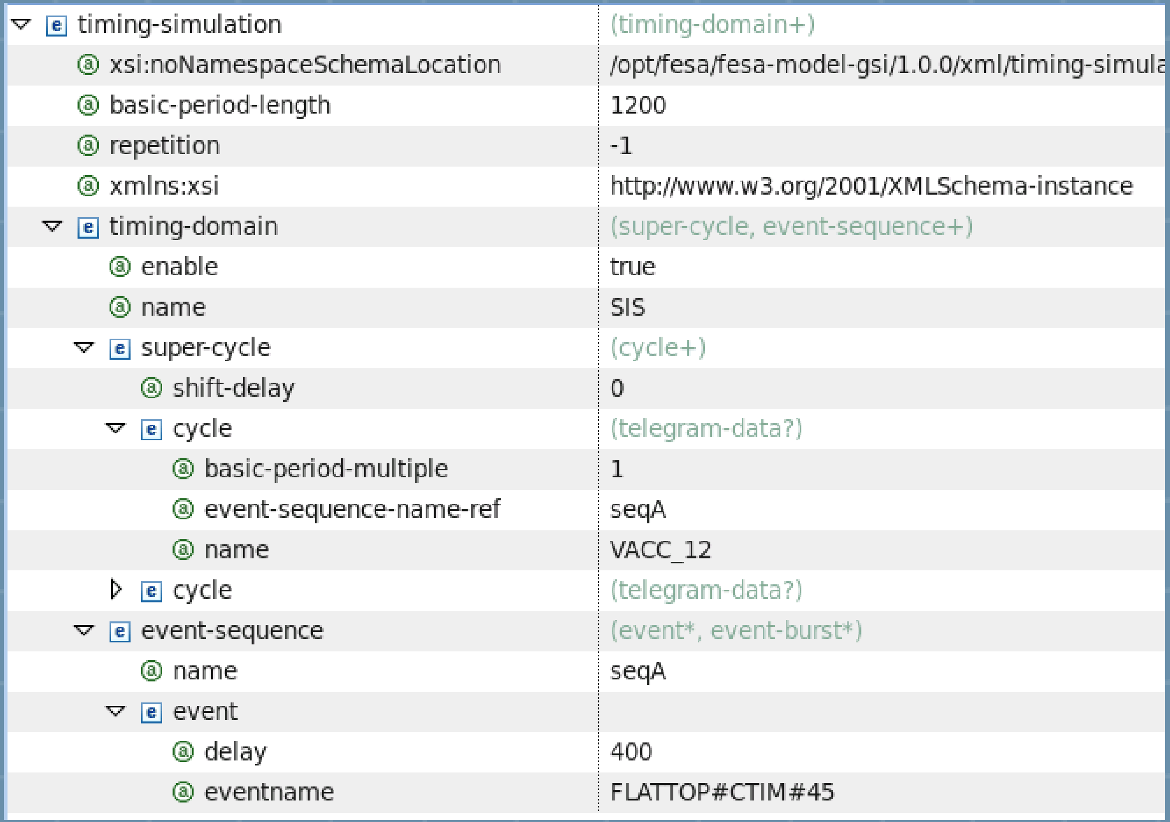


DeviceData_ExerciseRT_DU_0.1.0.instance

Node	Content
?-? xml	version="1.0" encoding=
▽ e instantiation-unit	(information, prio-manag
ⓐ xmlns:xsi	http://www.w3.org/2001/
ⓐ xsi:noNamespaceSchemaLocation	file:/nfs/cs-ccr-nfsdev/vol
▷ e information	(deploy-unit-name, deplo
▽ e prio-management	(classes, deploy-unit)
▷ e classes	(ExerciseRT)
▽ e deploy-unit	(concurrency-layers, com
▽ e concurrency-layers	(CLTemperature, CLCalib
▽ e	
ⓐ prio	60

Timing Simulation

XSI:noNamespaceSchemaLocation=“**/opt/fesa/fesa-model-gsi/1.3.1/xml/timing-simulation/TimingSimulationSchema.xsd**”



```
▽ e timing-simulation                    (timing-domain+)
      ⓐ xsi:noNamespaceSchemaLocation    /opt/fesa/fesa-model-gsi/1.0.0/xml/timing-simula
      ⓐ basic-period-length              1200
      ⓐ repetition                       -1
      ⓐ xmlns:xsi                        http://www.w3.org/2001/XMLSchema-instance
  ▽ e timing-domain                      (super-cycle, event-sequence+)
        ⓐ enable                         true
        ⓐ name                           SIS
    ▽ e super-cycle                      (cycle+)
          ⓐ shift-delay                  0
      ▽ e cycle                          (telegram-data?)
            ⓐ basic-period-multiple      1
            ⓐ event-sequence-name-ref    seqA
            ⓐ name                       VACC_12
      ▷ e cycle                          (telegram-data?)
    ▽ e event-sequence                   (event*, event-burst*)
          ⓐ name                         seqA
      ▽ e event
            ⓐ delay                      400
            ⓐ eventname                  FLATTOP#CTIM#45
```

./startScript.sh -f -timsim TimingSimulationConfig.xml -noRTSched

Exercise 3: Instantiation

- **Define two configurations for the “MeasVoltEvent”**
  - **Timing (Flattop#CTIM#45)**
  - **Timer 1Hz (1000ms)**
- **Define a configuration for the “CalibrationEvent”**
  - **OnDemand**
- **Create two devices and assign the configurations to them**
  - **One device should use the configuration Timing for the “MeasVoltEvent” the other device should use a Timer.**
  - **Both devices should use OnDemand for the “CalibrationEvent”**
- **Start the binary by using the startscript (add “-c x86_64” if needed)**
- **Use the FESA-Explorer to trigger the RTAction Calibrate (via the connected property)**

On any problems: fesa-support@gsi.de