Firmware Synthesis for Ultra-Thin IoT Devices Based on Model Integration

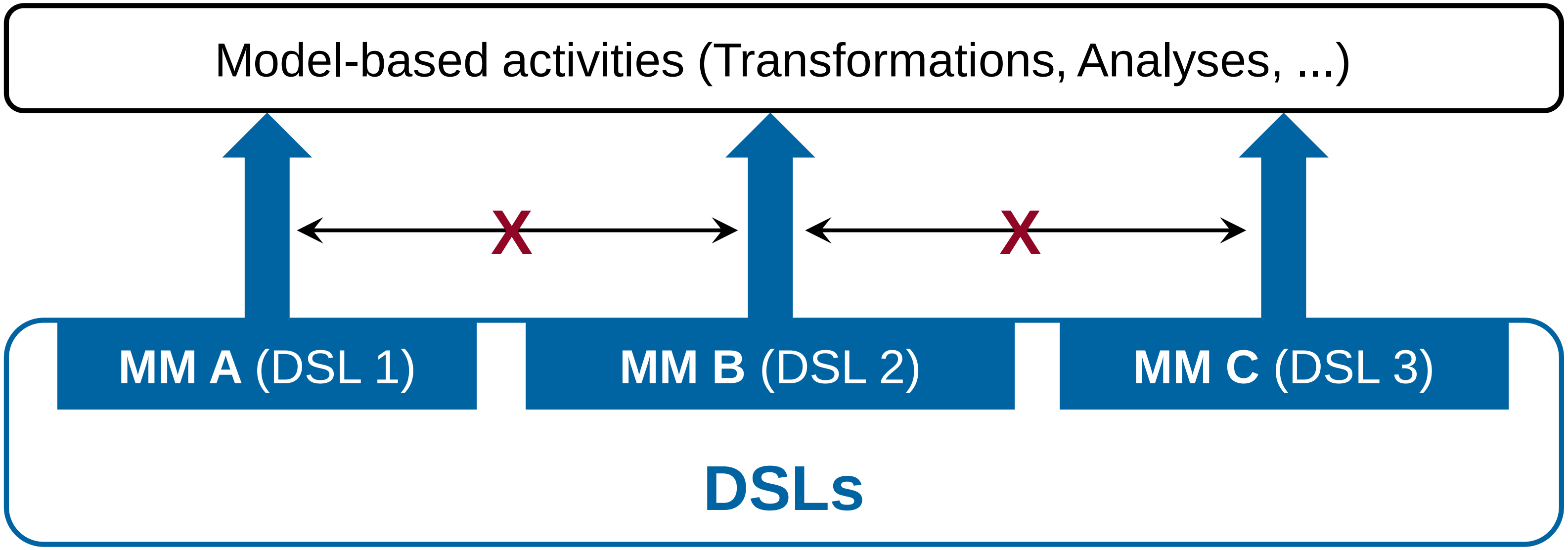
Arthur Kühlwein[1], Anton Paule[1], Leon Hielscher[1],
Wolfgang Rosenstiel[2], and Oliver Bringmann[2]

[1] FZI Research Center for Information Technology
[2] University of Tübingen

International Workshop on Modeling Language Engineering and Execution 2019

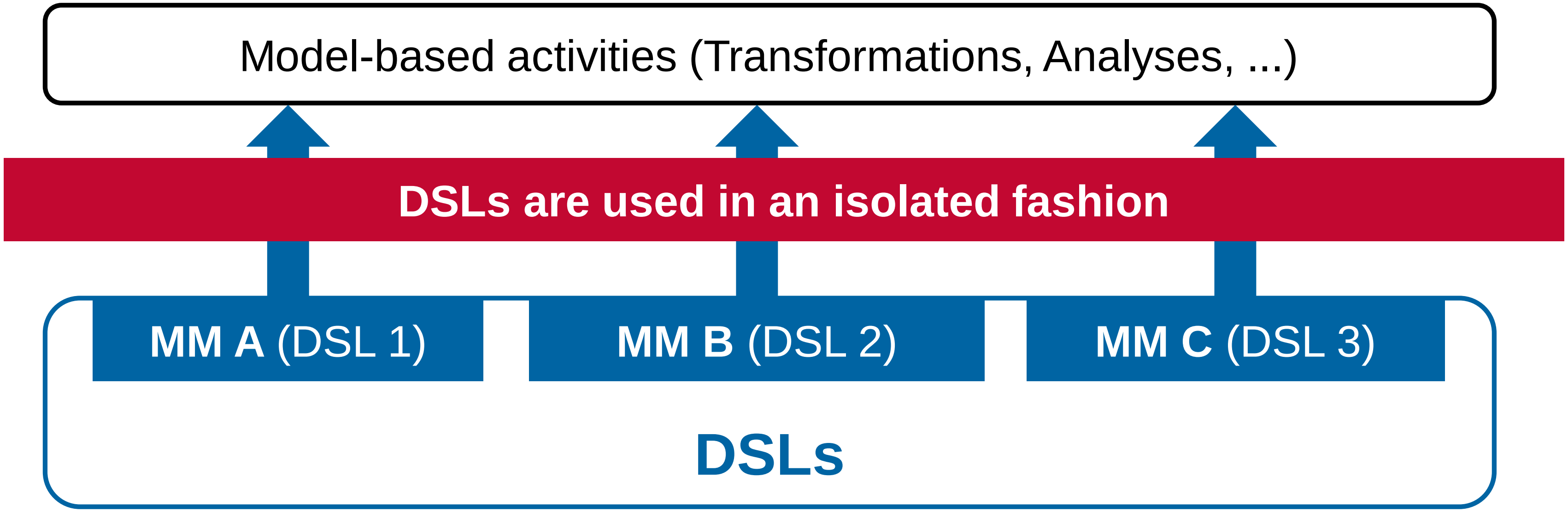
Introduction & Motivation

- FW for ultra-thin IoT devices challenging to develop
  - Resource constraints (Power, memory...)
  - Extensive FW functionalities (RT computing, security, safety, ...)
  - Market pressure (Short time-to-market)
- MD approaches can tackle some of these issues, variety of different DSLs are used



Introduction & Motivation

- FW for ultra-thin IoT devices challenging to develop
  - Resource constraints (Power, memory...)
  - Extensive FW functionalities (RT computing, security, safety, ...)
  - Market pressure (Short time-to-market)
- MD approaches can tackle some of these issues, variety of different DSLs are used


Model-based activities (Transformations, Analyses, ...)
DSLs are used in an isolated fashion
MM A (DSL 1)
MM B (DSL 2)
MM C (DSL 3)
DSLs

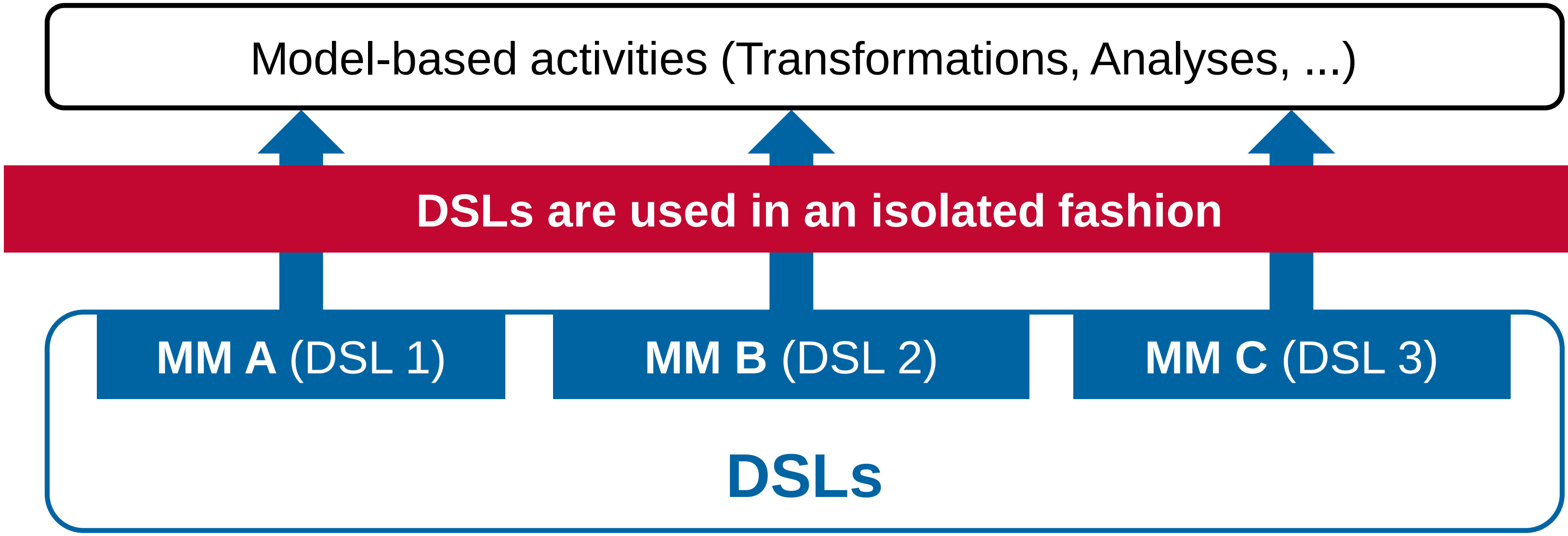
Introduction & Motivation

⟹ No common interface between MMs

- Difficult automation of FW development
- Capabilities of MD activities limited by MM

⟹ Co-design & coordination challenging

- HW/SW codesign common practice
- Prolongs FW development cycle
- Can lead to late detection of design errors



Model-based activities (Transformations, Analyses, ...)
DSLs are used in an isolated fashion
MM A (DSL 1)
MM B (DSL 2)
MM C (DSL 3)
DSLs

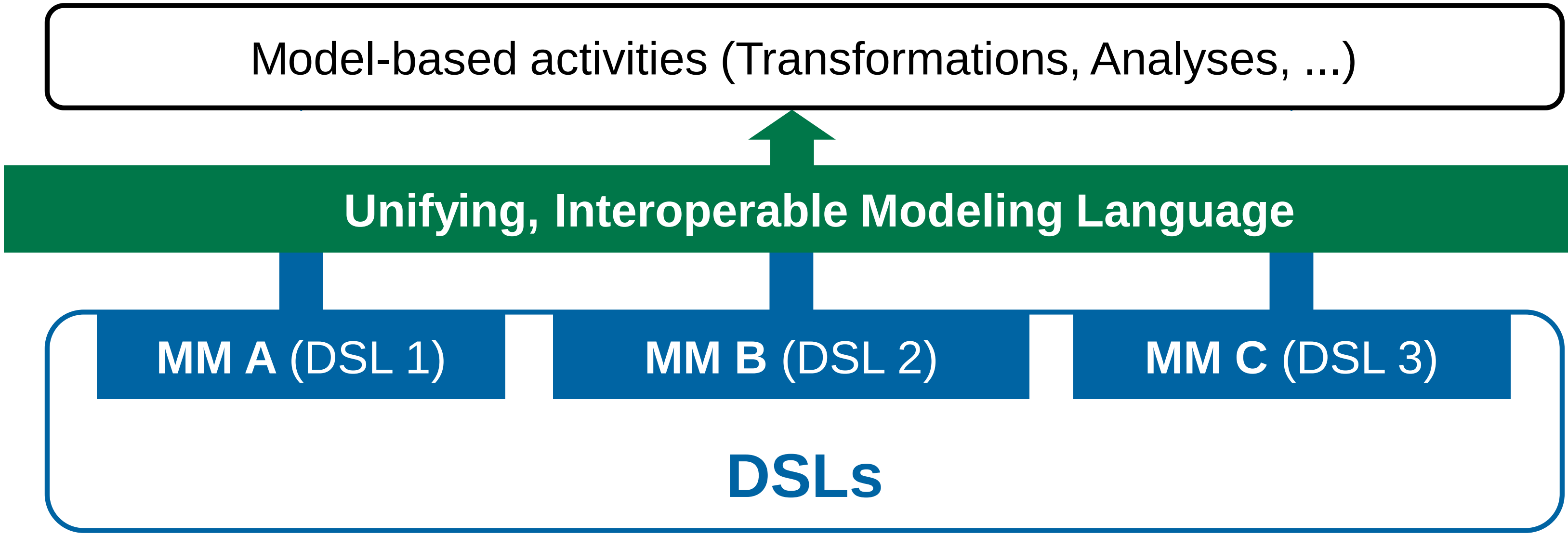
Introduction & Motivation

⟹ **No common interface between MMs**

- Difficult automation of FW development
- Capabilities of MD activities limited by MM

⟹ **Co-design & coordination challenging**

- HW/SW codesign common practice
- Prolongs FW development cycle
- Can lead to late detection of design errors


Model-based activities (Transformations, Analyses, ...)
Unifying, Interoperable Modeling Language
MM A (DSL 1)
MM B (DSL 2)
MM C (DSL 3)
DSLs

Introduction & Motivation

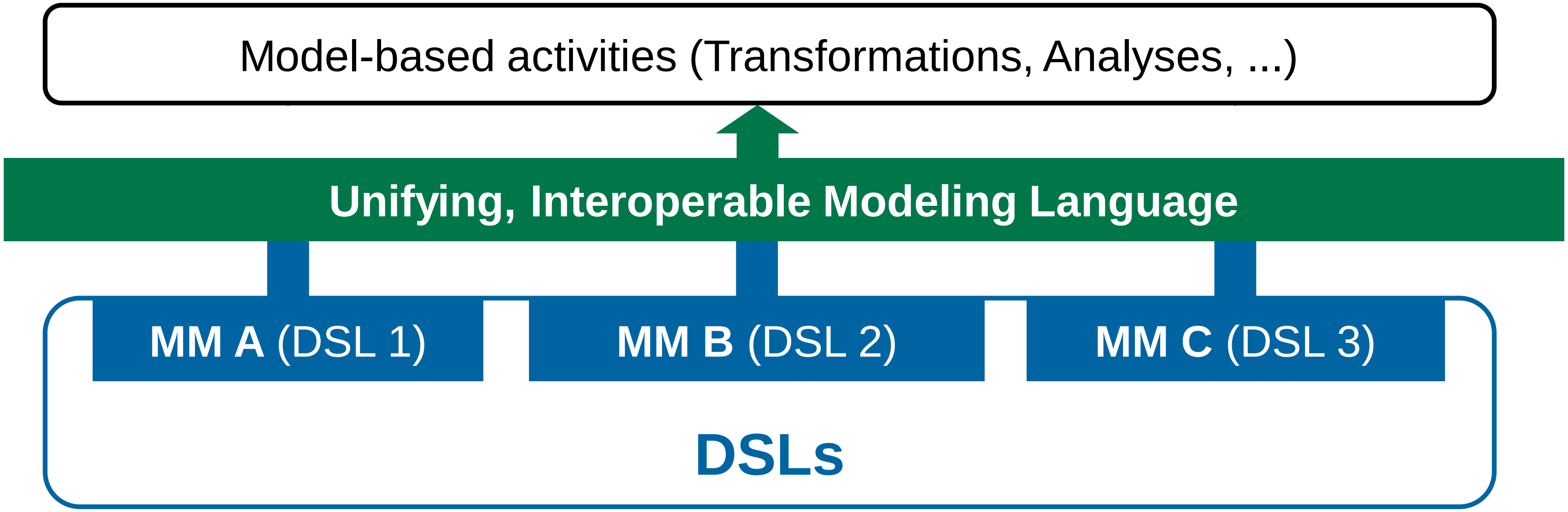
⟹ **Exploit data synergies via common interface**

- Expand capabilities of MD activities

⟹ **Easier co-design & coordination**

- Shorter FW development cycle
- Earlier detection of design errors

- **Holistic approach to the automated synthesis of FW**


Model-based activities (Transformations, Analyses, ...)
Unifying, Interoperable Modeling Language
MM A (DSL 1)
MM B (DSL 2)
MM C (DSL 3)
DSLs

Introduction & Motivation

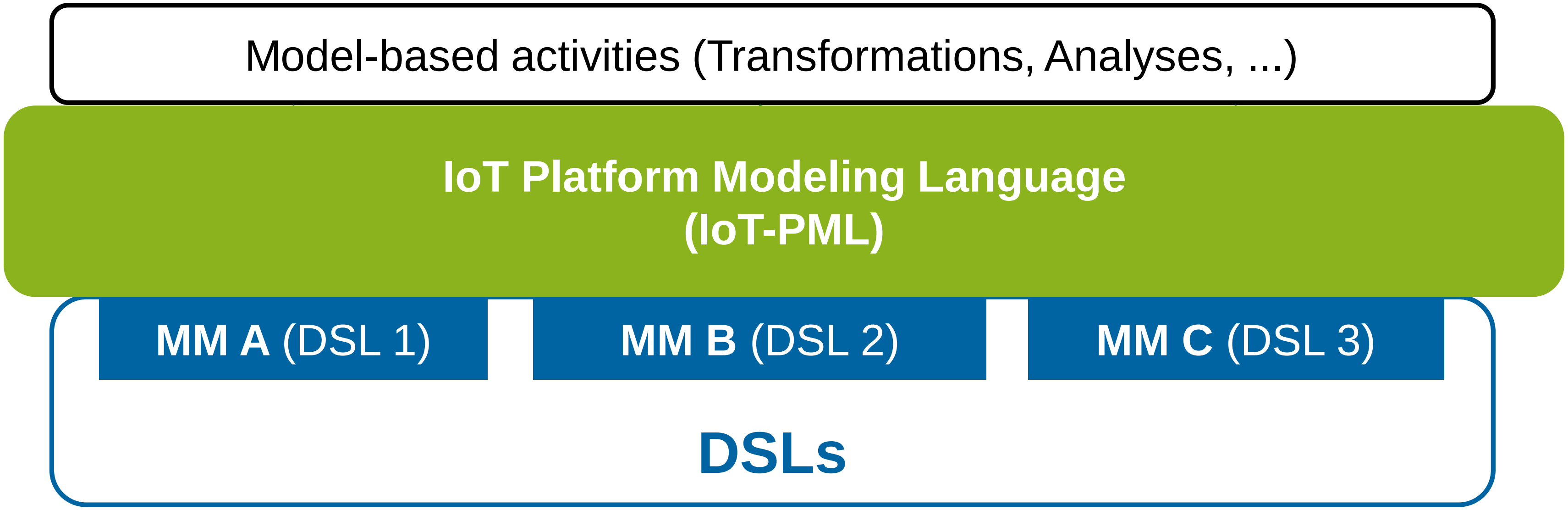
⟹ **Exploit data synergies via common interface**

- Expand capabilities of MD activities

⟹ **Easier co-design & coordination**

- Shorter FW development cycle
- Earlier detection of design errors

- **Holistic approach to the automated synthesis of FW**

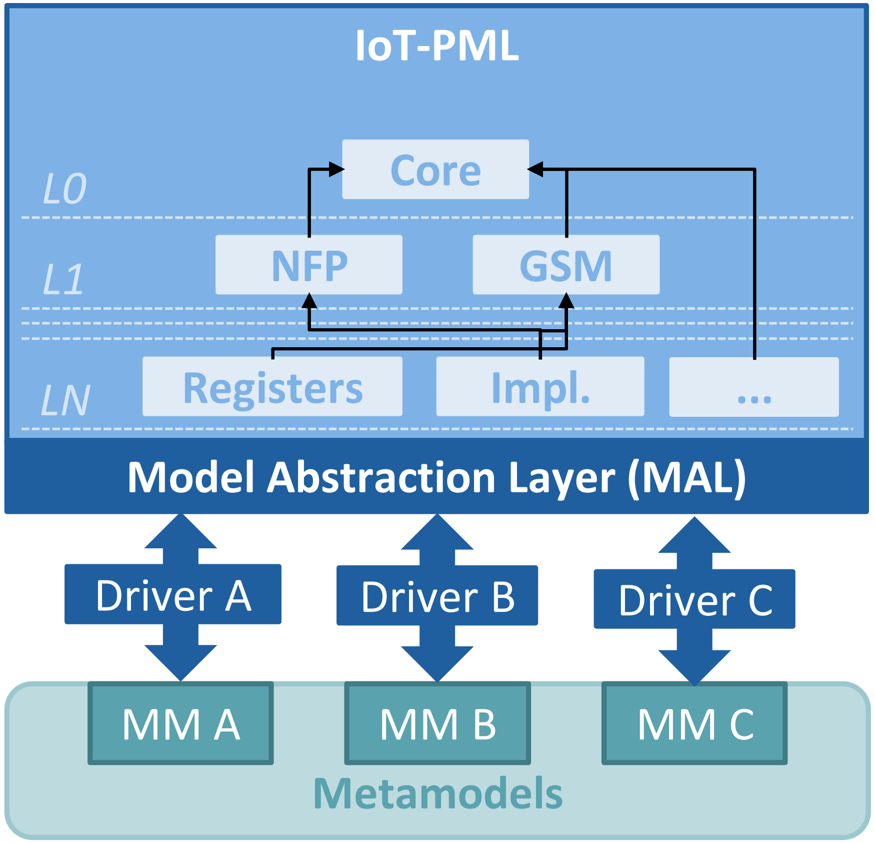

Model-based activities (Transformations, Analyses, ...)
IoT Platform Modeling Language
(IoT-PML)
MM A (DSL 1)
MM B (DSL 2)
MM C (DSL 3)
DSLs

The IoT-PML

- **Basic idea:** Capture essential concepts of related MMs
  - FRs, NFRs/NFPs of SW/HW platform
  - Device configurability
  - Usage scenarios

- Common abstractions of these concepts to enable effective integration and cooperation
  - Careful analysis necessary, as we do **not** want to create a gargantuan metamodel
  - Provide data exchange at model runtime via a model abstraction layer
- Support for top-down and bottom-up workflows
- MOF-conformant metamodel
  - Currently implemented as a UML profile

The IoT-PML – Architecture & Features



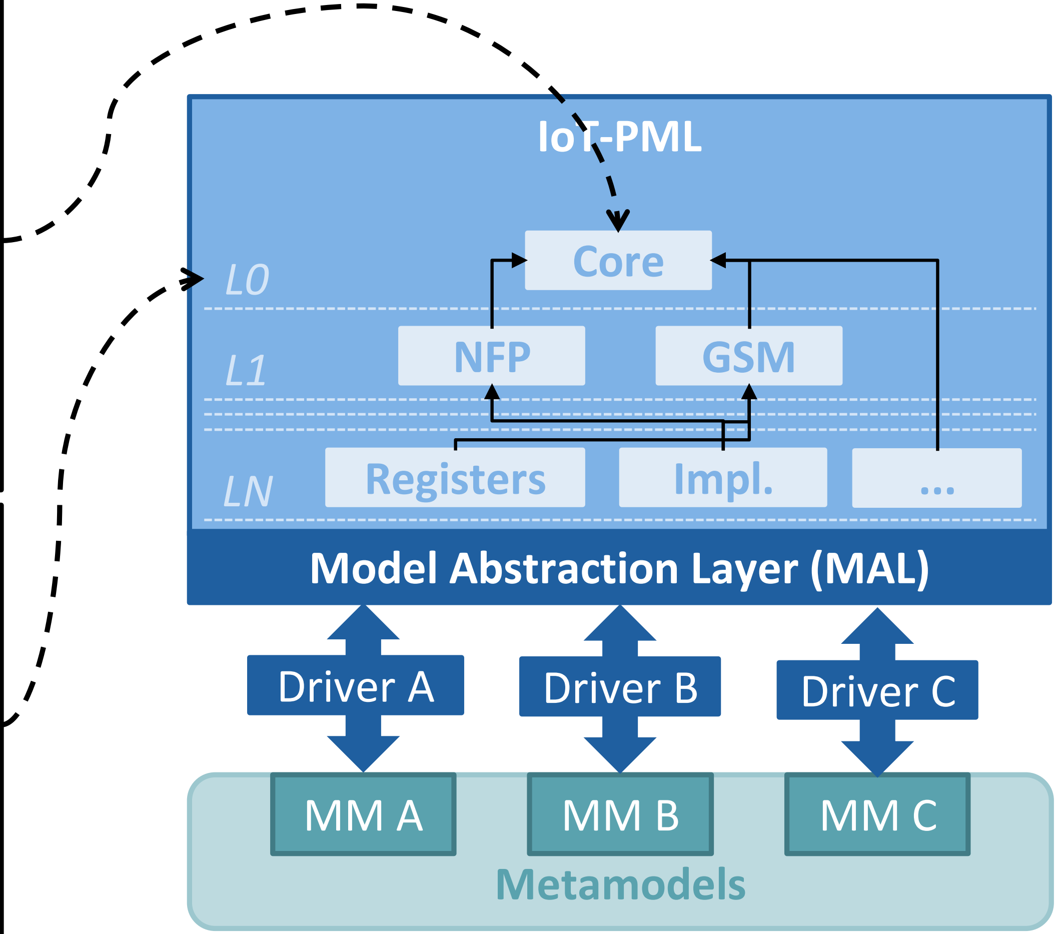
The IoT-PML – Architecture & Features

Modular

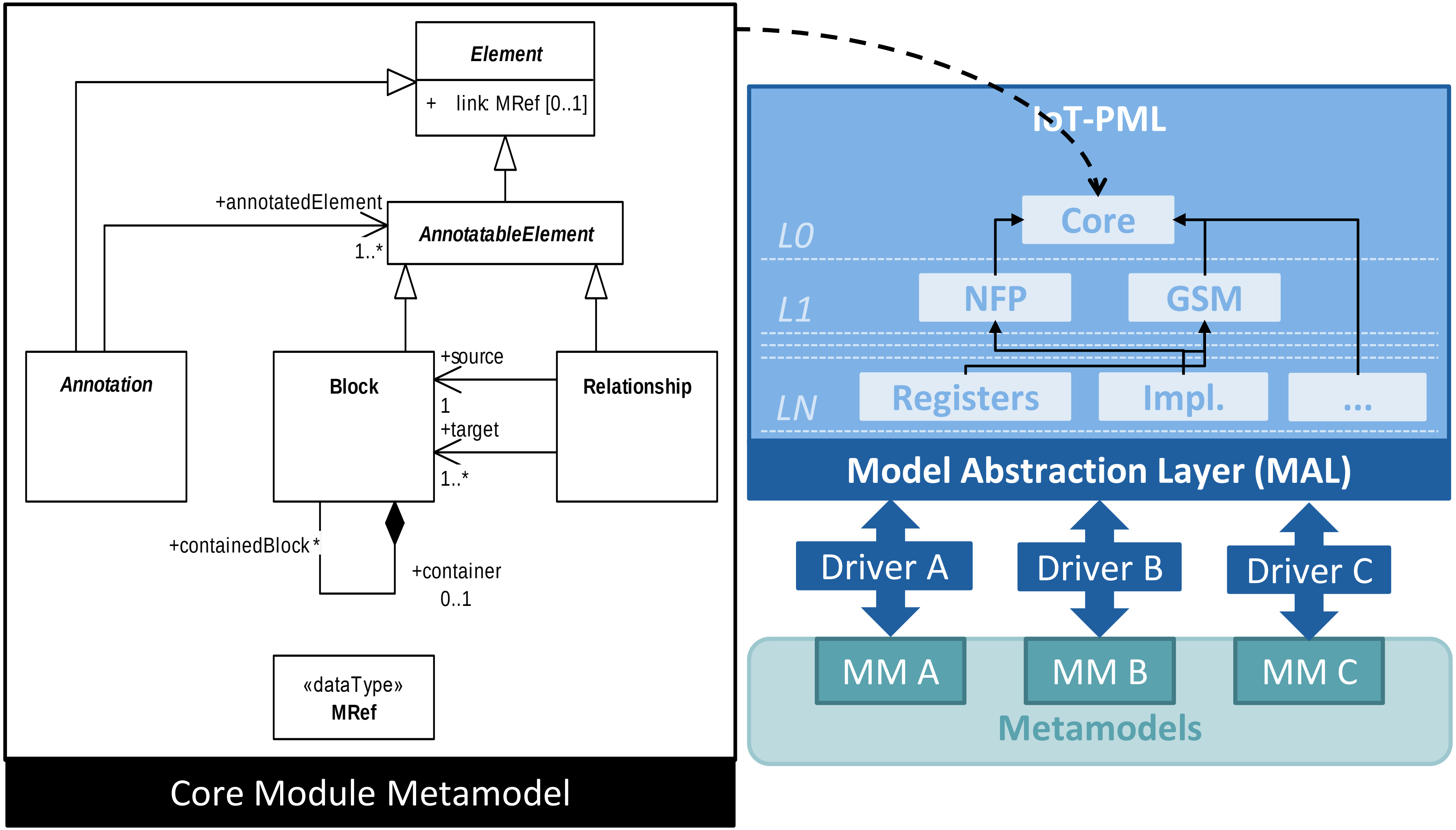
- **Module:** contains **concepts** common to a number of MMs
- **Concepts** have to specialize concepts of the Core module

Layered

- **Layer:** contains **concepts** at a particular abstraction level
- ***L0*** highest, ***LN*** lowest abstraction level



The IoT-PML – Architecture & Features



The IoT-PML – Architecture & Features

Model Linkage

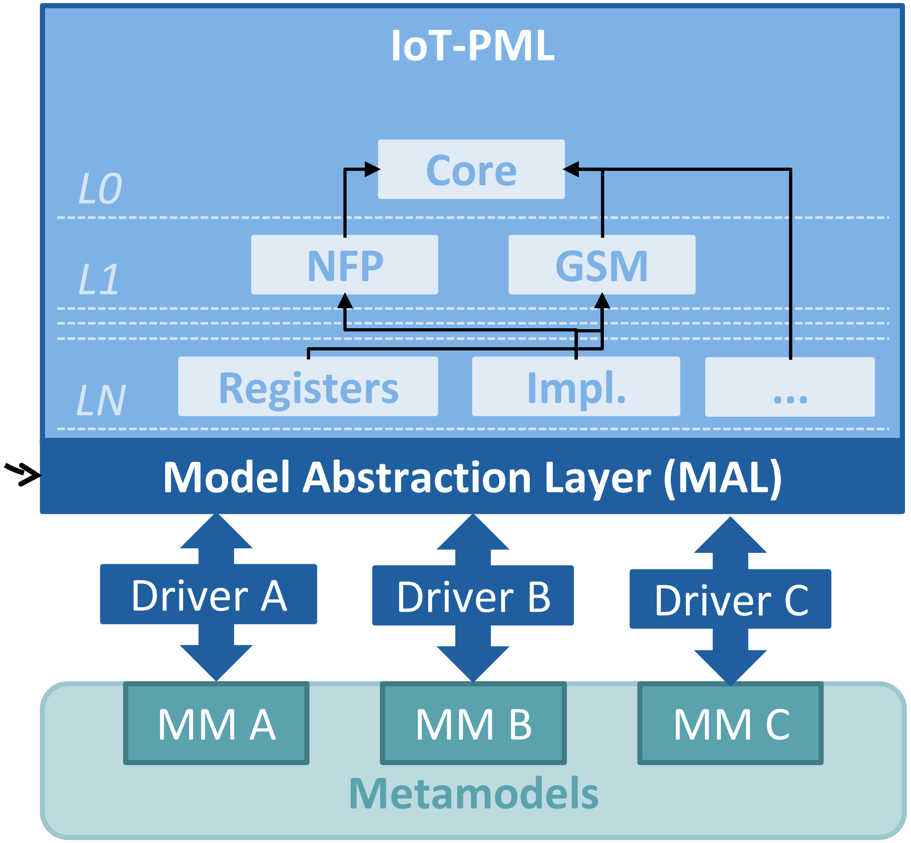
- Each IoT-PML element can link to an external model element *e* using a **model reference**

$$\mathbf{M : URI}_e$$

MM Identifier

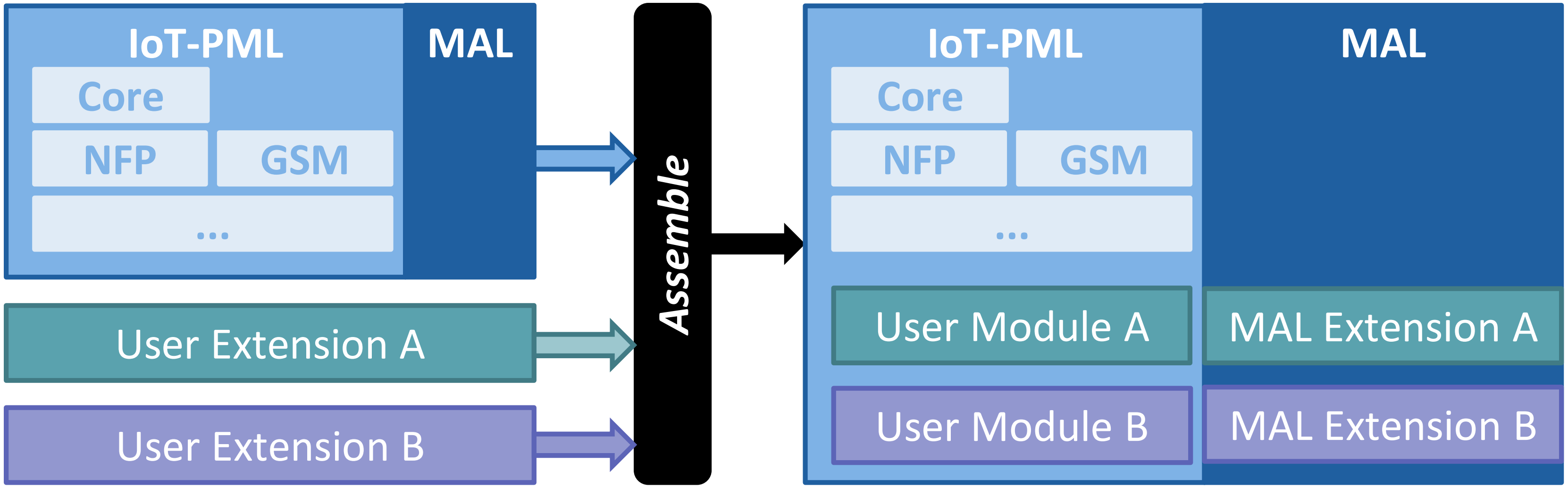
URI of *e*

- Linkage at model runtime facilitated by **Model Abstraction Layer (MAL)**
  - Module-specific interfaces, which are implemented by metamodel-specific drivers



The IoT-PML – Architecture & Features

- **Extensibility**
  - **User modules** can be added to the IoT-PML
    - New concepts, refinement of existing concepts
    - Extend MAL with corresponding interfaces
- **IoT-PML and MAL are constructed at model runtime by assembling built-in and user modules**

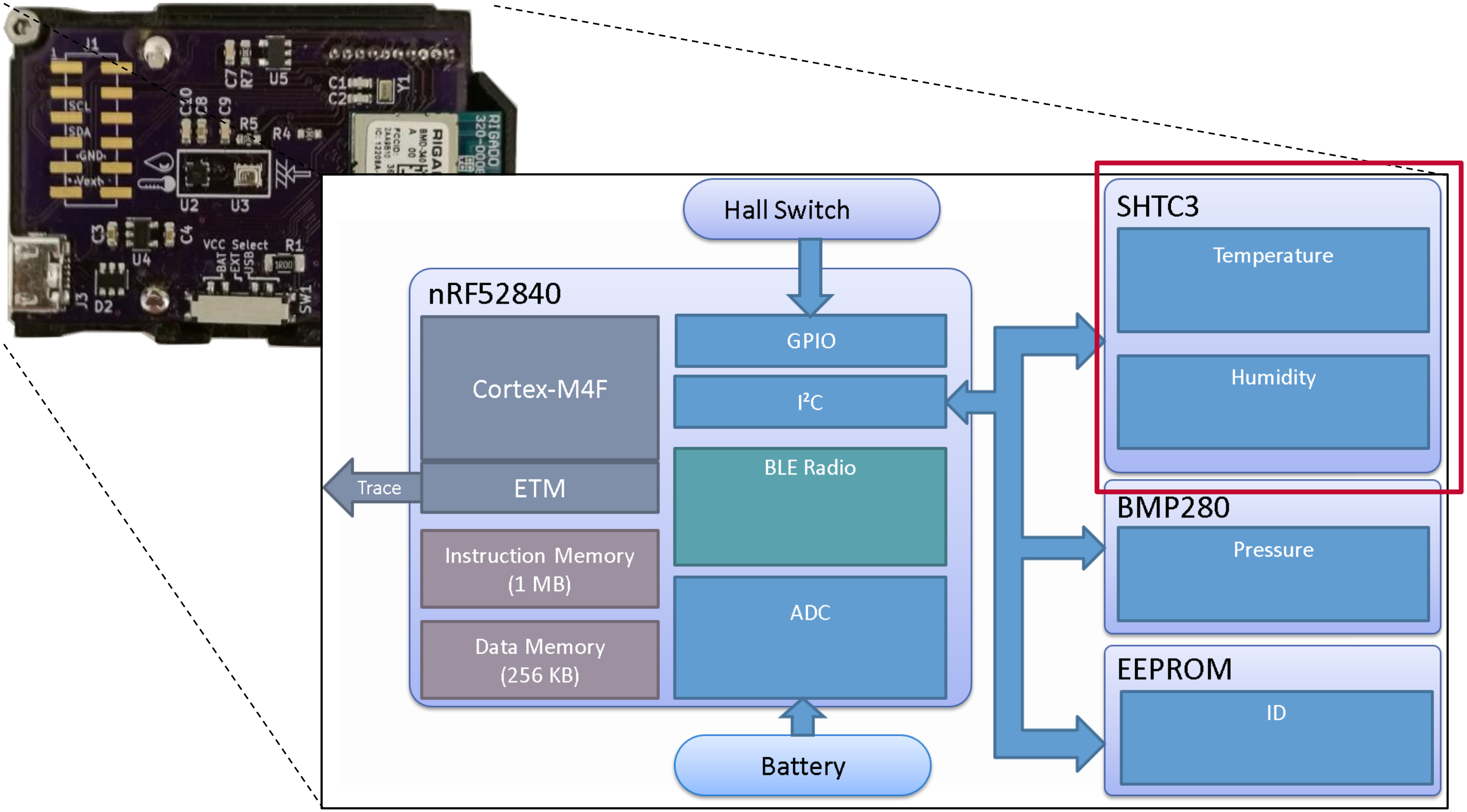


The IoT-PML – Implementation

- Currently implemented as a UML profile
  - Layer ←→ package, module ←→ (sub)profile, concept ←→ stereotype
  - Exploit UML for modeling SW aspects
  - Leverage large ecosystem of model-based technologies (M2M, M2T, ...) that evolved around OMG standards
  - Mature tooling support
- Realized using Eclipse-based frameworks and tools
  - EMF
  - Papyrus Modeling Environment

Use Case

- **Code generation (and verification) of a driver for an IoT sensor device peripheral**



Hall Switch

nRF52840

Cortex-M4F

ETM

Trace

Instruction Memory (1 MB)

Data Memory (256 KB)

GPIO

I²C

BLE Radio

ADC

Battery

SHTC3

Temperature

Humidity

BMP280

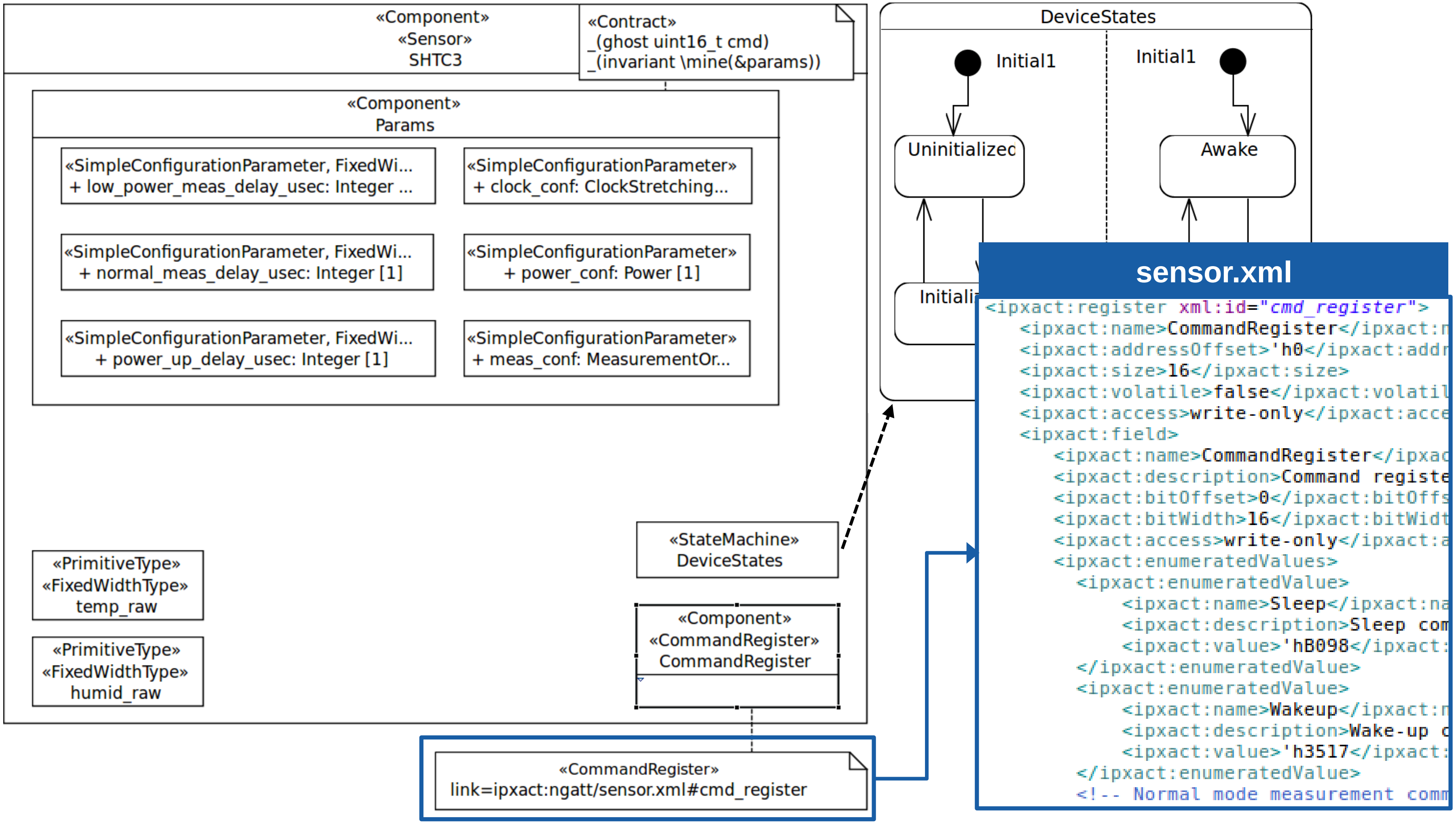
Pressure

EEPROM

ID

Use Case

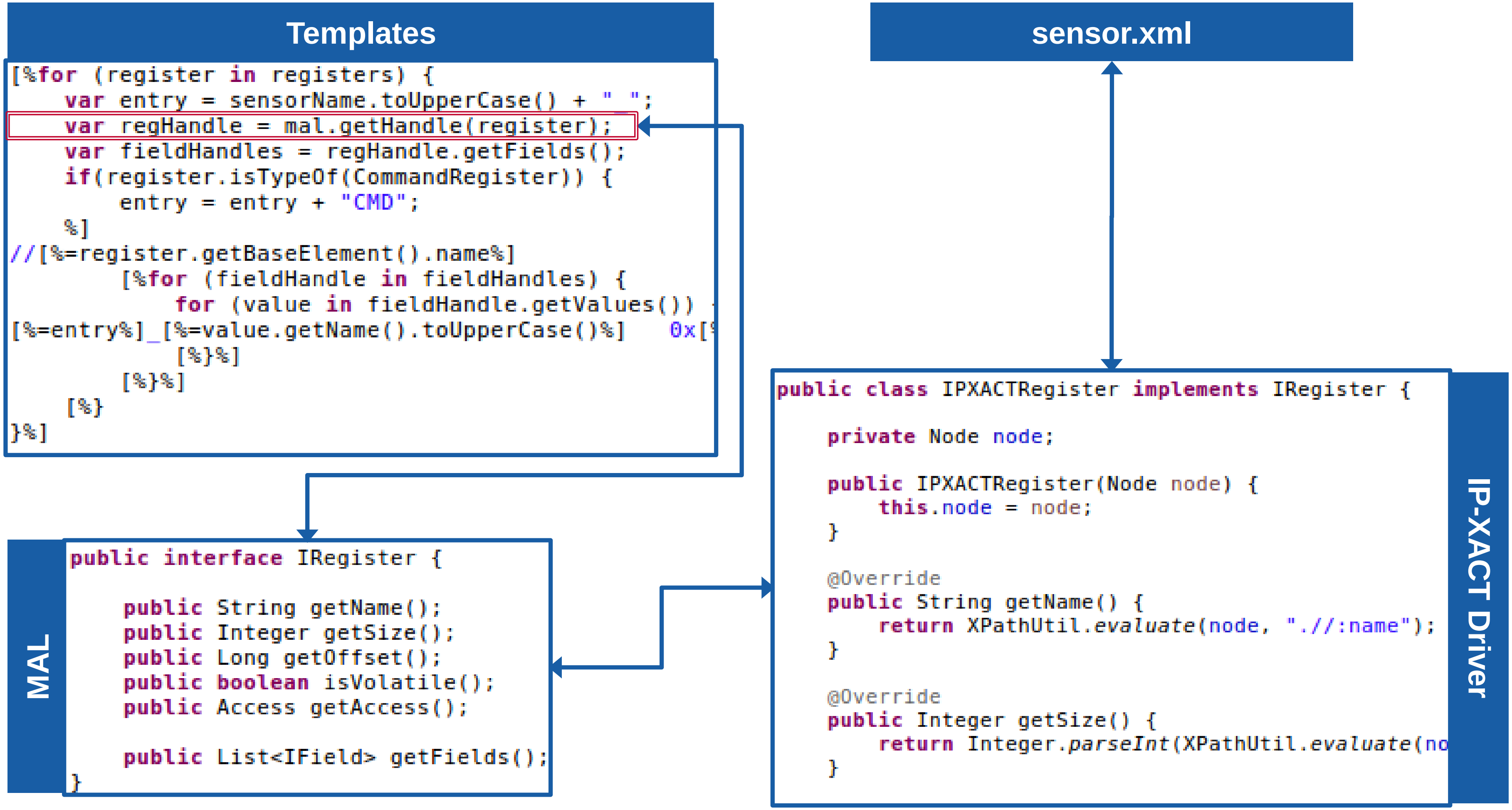
- Basic top-down workflow
  - Generate SW-centric IoT-PML model of peripheral driver


«Component»
«Sensor»
SHTC3
«Contract»
_(ghost uint16_t cmd)
_(invariant \mine(&params))
«Component»
Params
«SimpleConfigurationParameter, FixedWi...
+ low_power_meas_delay_usec: Integer ...
«SimpleConfigurationParameter»
+ clock_conf: ClockStretching...
«SimpleConfigurationParameter, FixedWi...
+ normal_meas_delay_usec: Integer [1]
«SimpleConfigurationParameter»
+ power_conf: Power [1]
«SimpleConfigurationParameter, FixedWi...
+ power_up_delay_usec: Integer [1]
«SimpleConfigurationParameter»
+ meas_conf: MeasurementOr...
«PrimitiveType»
«FixedWidthType»
temp_raw
«PrimitiveType»
«FixedWidthType»
humid_raw
«StateMachine»
DeviceStates
«Component»
«CommandRegister»
CommandRegister
«CommandRegister»
link=ipxact:ngatt/sensor.xml#cmd_register
DeviceStates
Initial1
Initial1
Uninitializec
Awake
Initiali
sensor.xml

```
<ipxact:register xml:id="cmd_register">
   <ipxact:name>CommandRegister</ipxact:n
   <ipxact:addressOffset>'h0</ipxact:addr
   <ipxact:size>16</ipxact:size>
   <ipxact:volatile>false</ipxact:volatil
   <ipxact:access>write-only</ipxact:acce
   <ipxact:field>
      <ipxact:name>CommandRegister</ipxac
      <ipxact:description>Command registe
      <ipxact:bitOffset>0</ipxact:bitOffs
      <ipxact:bitWidth>16</ipxact:bitWidt
      <ipxact:access>write-only</ipxact:a
      <ipxact:enumeratedValues>
        <ipxact:enumeratedValue>
            <ipxact:name>Sleep</ipxact:na
            <ipxact:description>Sleep com
            <ipxact:value>'hB098</ipxact:
        </ipxact:enumeratedValue>
        <ipxact:enumeratedValue>
            <ipxact:name>Wakeup</ipxact:n
            <ipxact:description>Wake-up c
            <ipxact:value>'h3517</ipxact:
        </ipxact:enumeratedValue>
        <!-- Normal mode measurement comm
```

Use Case

- Basic top-down workflow
  - Use templates to generate driver skeleton



```
[%for (register in registers) {
    var entry = sensorName.toUpperCase() + "_";
    var regHandle = mal.getHandle(register);
    var fieldHandles = regHandle.getFields();
    if(register.isTypeOf(CommandRegister)) {
        entry = entry + "CMD";
    %]
//[%=register.getBaseElement().name%]
        [%for (fieldHandle in fieldHandles) {
            for (value in fieldHandle.getValues())
[%=entry%]_[%=value.getName().toUpperCase()%]   0x[
            [%}%]
        [%}%]
    [%}
}%]
```

```
public interface IRegister {

    public String getName();
    public Integer getSize();
    public Long getOffset();
    public boolean isVolatile();
    public Access getAccess();

    public List<IField> getFields();
}
```

```
public class IPXACTRegister implements IRegister {

    private Node node;

    public IPXACTRegister(Node node) {
        this.node = node;
    }

    @Override
    public String getName() {
        return XPathUtil.evaluate(node, ".//:name");
    }

    @Override
    public Integer getSize() {
        return Integer.parseInt(XPathUtil.evaluate(no
    }
```

Conclusions

- First concept of novel unifying modeling language for ultra-thin IoT device FW
  - Linking mechanism enables data exchange with external metamodels
  - MOF-conformant, currently implemented as a UML profile
- Language development and analysis of metamodels still ongoing
  - New external metamodel: device trees
- Rudimentary tool support
  - Currently working on Papyrus integration of MAL features
  - Need to keep data consistent between IoT-PML ←→ external model
- Generic mechanism to map IoT-PML concepts to arbitrary XSD-based metamodels
  - Automatic generation of MAL drivers
  - Could be extended to arbitrary metamodels (e.g. text-based)

Thank you for your attention!

Any questions?