Modular Deployment of UML Models for V&V Activities and Embedded Execution

17th Workshop on Model-Driven Engineering, Verification and Validation co-located with MODELS 2020

20th October 2020

Valentin Besnard[1], *Frédéric Jouault[1]*, Matthias Brun[1],
Ciprian Teodorov[2], Philippe Dhaussy[2], Jérôme Delatour[1]

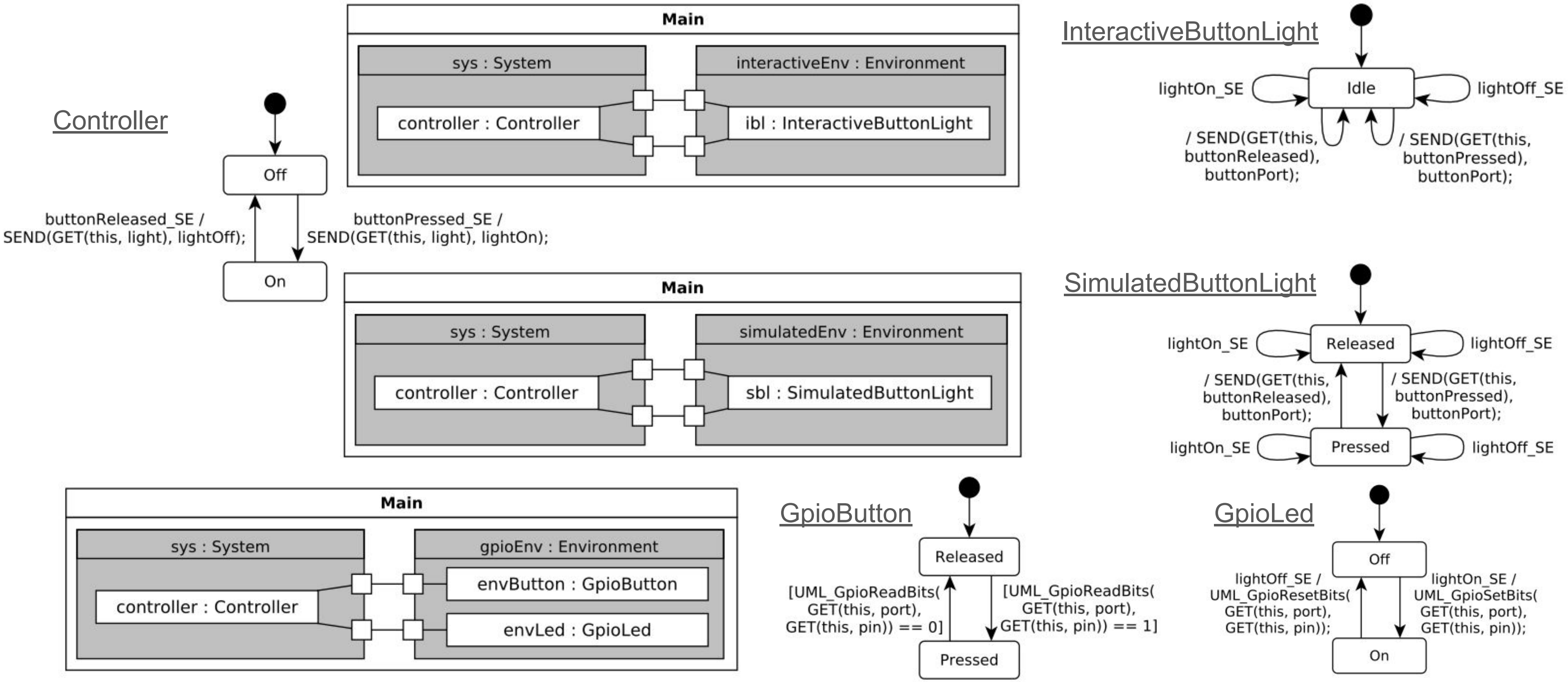
[1] ERIS, ESEO-TECH, Angers, France
[2] Lab-STICC UMR CNRS 6285, ENSTA Bretagne, Brest, France

This work has been partially funded by Davidson Consulting

Context and Problems

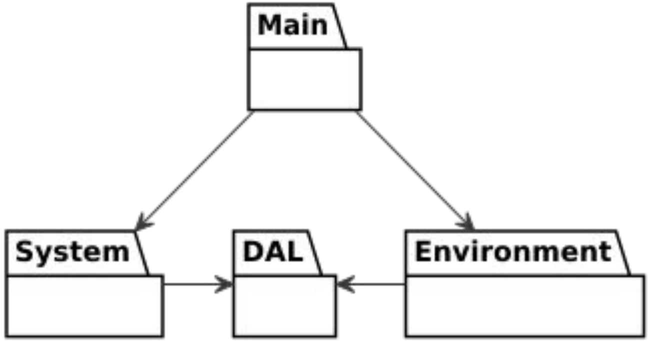
- To execute, verify and validate embedded system software, multiple models of their environment are required:
  - Abstract environment models for V&V activities
    - To close the system model execution with a superset of all possible scenarios
  - Concrete environment models for actual execution on an embedded target
    - To interact with the physical environment through sensors and actuators of the target
- Need to connect the system model to different environment models in a modular way
- Two main research challenges remain:
  - The environment model is often target-specific and tightly coupled with the system model
  - Transformations used for model deployment (e.g., code generation) are usually unproven, which makes difficult to ensure that formal properties verified during the design phase are still preserved at runtime

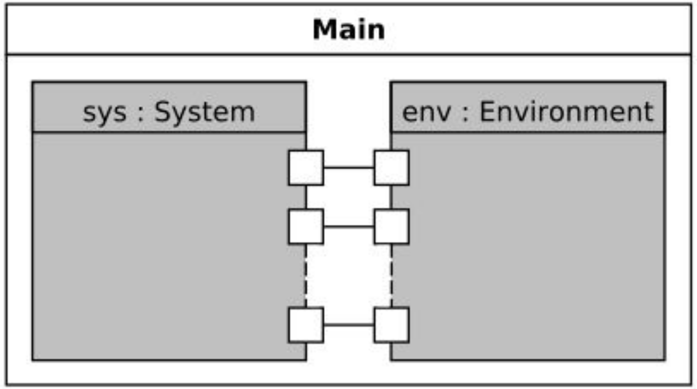
Approach Overview - the Button-Led model


Main
sys : System
interactiveEnv : Environment
controller : Controller
ibl : InteractiveButtonLight
InteractiveButtonLight
Idle
lightOn_SE
lightOff_SE
/ SEND(GET(this, buttonReleased), buttonPort);
/ SEND(GET(this, buttonPressed), buttonPort);
Controller
Off
On
buttonReleased_SE / SEND(GET(this, light), lightOff);
buttonPressed_SE / SEND(GET(this, light), lightOn);
Main
sys : System
simulatedEnv : Environment
controller : Controller
sbl : SimulatedButtonLight
SimulatedButtonLight
Released
Pressed
lightOn_SE
lightOff_SE
/ SEND(GET(this, buttonReleased), buttonPort);
/ SEND(GET(this, buttonPressed), buttonPort);
lightOn_SE
lightOff_SE
Main
sys : System
gpioEnv : Environment
controller : Controller
envButton : GpioButton
envLed : GpioLed
GpioButton
Released
Pressed
[UML_GpioReadBits(GET(this, port), GET(this, pin)) == 0]
[UML_GpioReadBits(GET(this, port), GET(this, pin)) == 1]
GpioLed
Off
On
lightOff_SE / UML_GpioResetBits(GET(this, port), GET(this, pin));
lightOn_SE / UML_GpioSetBits(GET(this, port), GET(this, pin));

Modular UML Model

- A modular UML model is divided into several files (one UML package per file):
  - **System**
    - The System component
    - All UML objects of the system
  - **Environment**
    - The Environment component
    - All UML objects of the environment
  - **DAL** (Device Abstraction Layer)
    - Interfaces and signals definition
    - → The system can be defined in a generic way
    - → The environment can be easily exchanged
  - **Main**
    - The Main composite structure



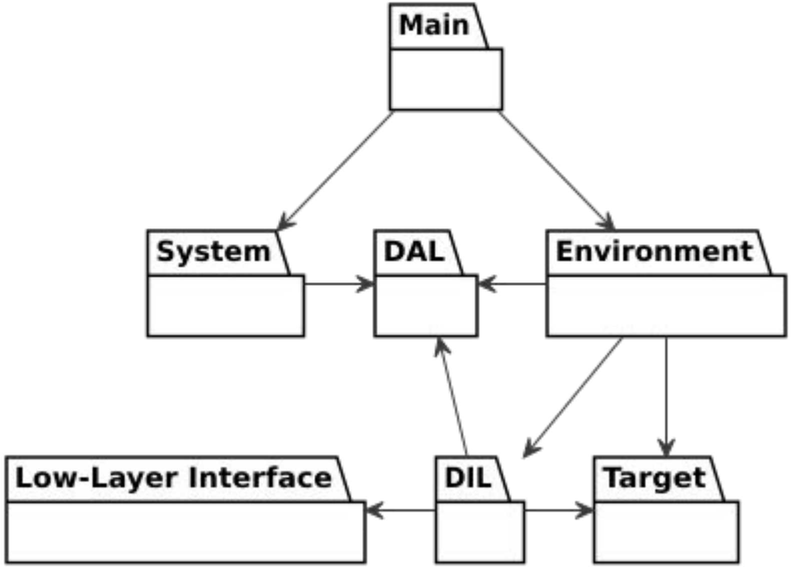


Modular UML Model - Stable XMI IDs

- Need to make references to external elements
  - Use XMI identifiers (IDs) to identify each element but two problems
    - Not human-readable (e.g., "_7wRIABydEduyofBvg4RL2w")
    - Need to be keep up-to-date between files
- Our solution - Stable XMI IDs
  - Use fully qualified names as XMI IDs (e.g., "DAL.buttonPressed")
  - Can be shared between several files → "stable"
  - Use the ElementImport mechanism of UML to import an external element in a file
  - The same qualified name in several files refers to the same element
  - OCL validation rules are used to ensure consistency
- Benefit: It becomes possible to only change the environment component to use a different one assuming that this component has the same name and the same ports

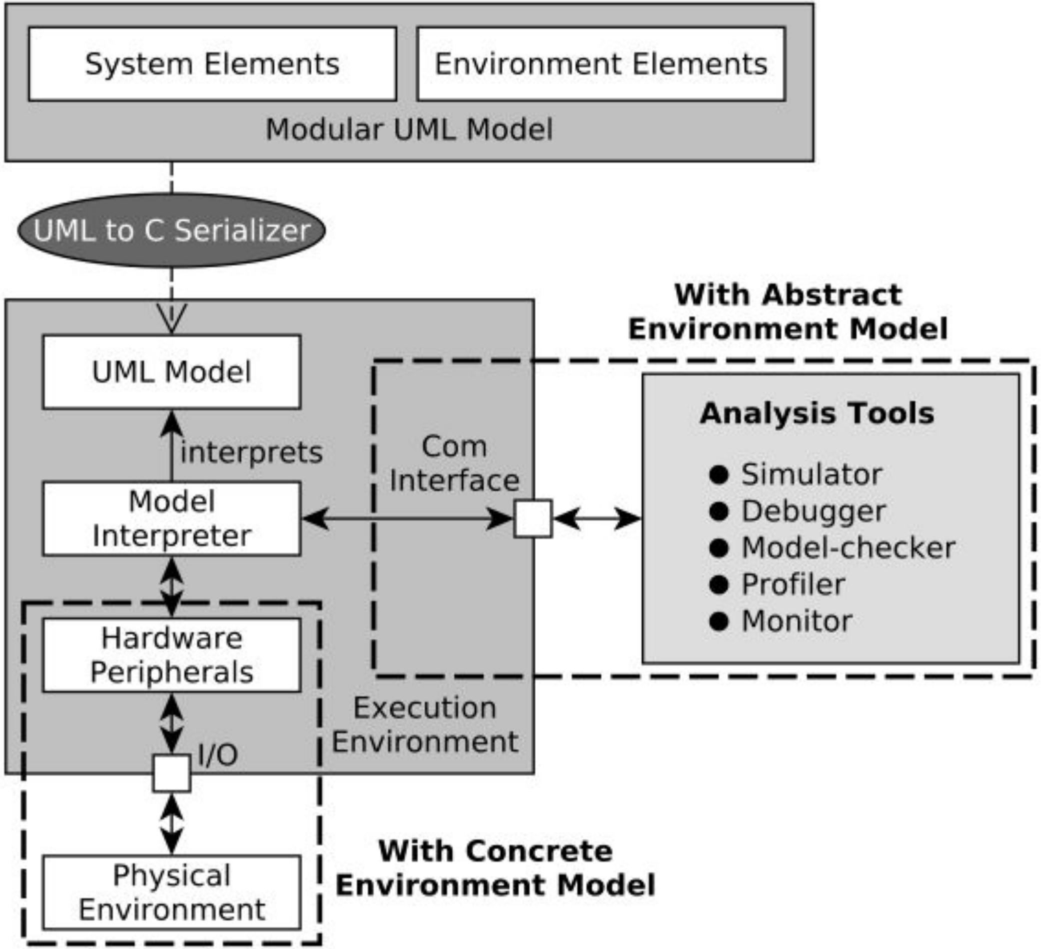
Link environment model with Hardware

- Three more UML packages (3 more files)
  - **DIL** (Device Implementation Layer)
    - UML classes that can be seen as kind of generic devices (e.g., buttons, leds, timers)
  - **Low-Layer Interface**
    - Functions (in C language) used to activate, configure and run hardware peripherals
  - **Target**
    - All available peripherals of the board
  - **Environment**
    - Instantiate DIL devices with Target parameters (actual hardware peripherals)



Deployment with EMI

- Embedded Model Interpreter
- All UML elements are serialized in C language
  - Keep relation with stable XMI IDs
  - Use EcoreUtil.resolve() method for resolving references to external elements
- Use the same pair (Model + Semantics) for V&V activities and actual execution
  - Preserve the correctness of software properties at runtime



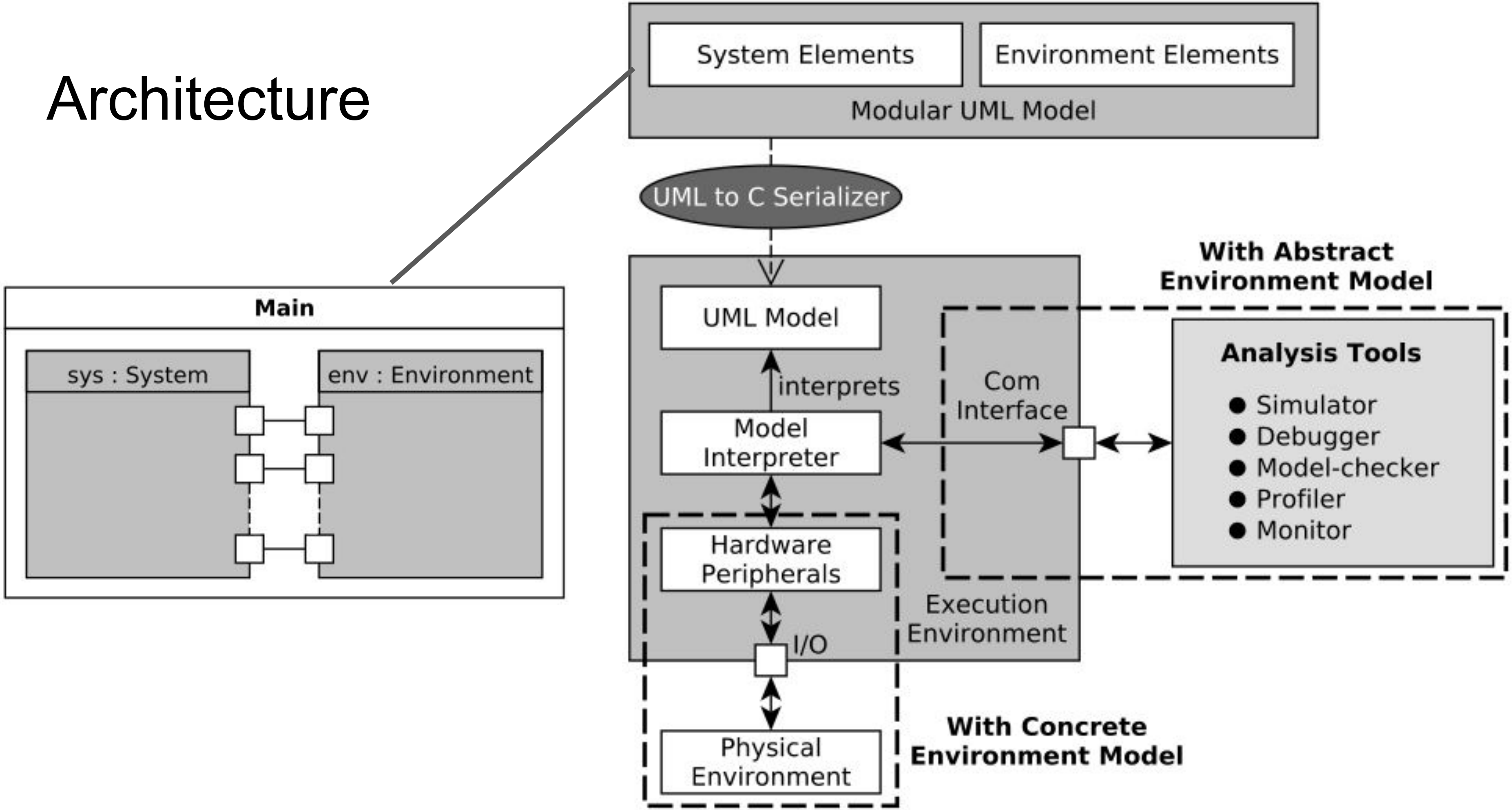
Experiments

- Applied on models of differents embedded systems
  - The controller of a level crossing system
  - The user interface of a cruise control system
- Use the OBP2 model-checker to apply V&V activities:
  - Interactive simulation
  - Deadlock detection
  - LTL model-checking
- Use an STM32 board for embedded execution
- Benefits
  - Provide modularity at model-level and avoid duplication of the system component
  - No impact on results of V&V activities
- Complexity of designing a modular UML model
  - Defining a modular UML model (or modularizing an existing UML model) requires more UML elements (ports, interfaces)

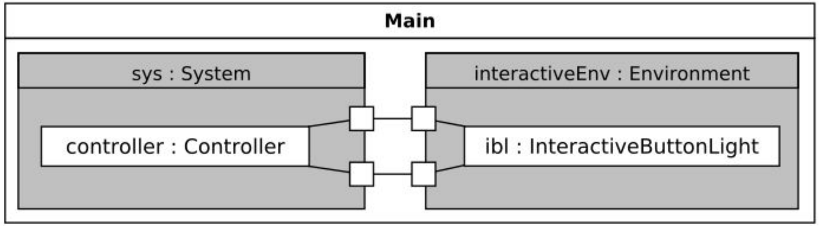
Conclusion

- System model defined in a generic way (unique system model)
  - Platform-independent
  - Decoupled from the environment
  - Deployed as it stands for model verification and runtime execution
- Different environment models can be easily linked to the system model
  - To close the system execution during V&V activities
  - To interact with the physical environment for embedded execution
- Model deployment with EMI (unique language semantic implementation) helps ensure the preservation of verified properties at runtime
- Further improvements
  - Better model the environment (e.g., to have multiple abstraction layers like in an OS)
  - Apply the approach to industrial case studies

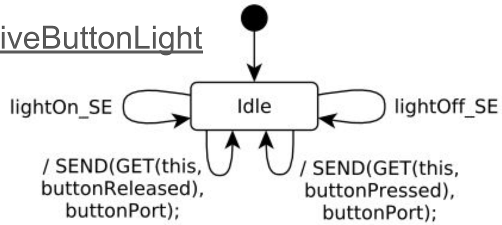
Architecture



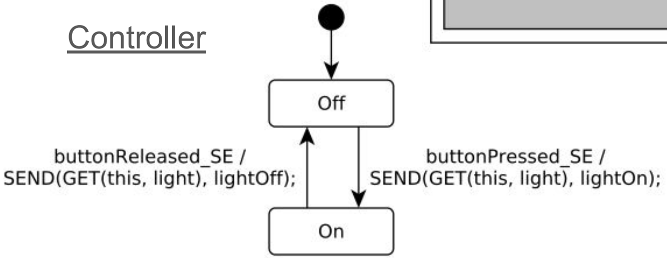
Approach Overview - the Button-Led model

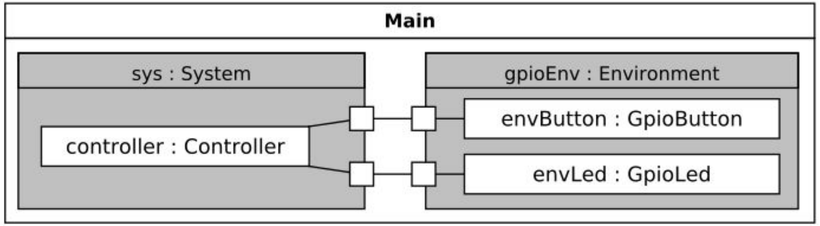


InteractiveButtonLight

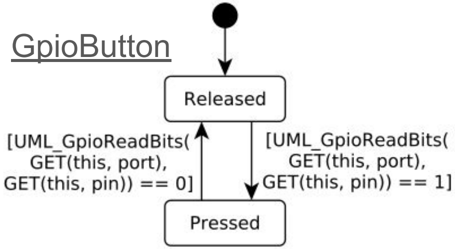

lightOn_SE
Idle
lightOff_SE
/ SEND(GET(this, buttonReleased), buttonPort);
/ SEND(GET(this, buttonPressed), buttonPort);

Controller

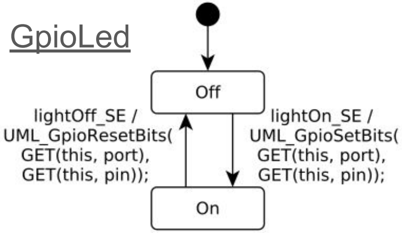

Off
buttonReleased_SE / SEND(GET(this, light), lightOff);
buttonPressed_SE / SEND(GET(this, light), lightOn);
On



GpioButton


Released
[UML_GpioReadBits(GET(this, port), GET(this, pin)) == 0]
[UML_GpioReadBits(GET(this, port), GET(this, pin)) == 1]
Pressed

GpioLed


Off
lightOff_SE / UML_GpioResetBits(GET(this, port), GET(this, pin));
lightOn_SE / UML_GpioSetBits(GET(this, port), GET(this, pin));
On