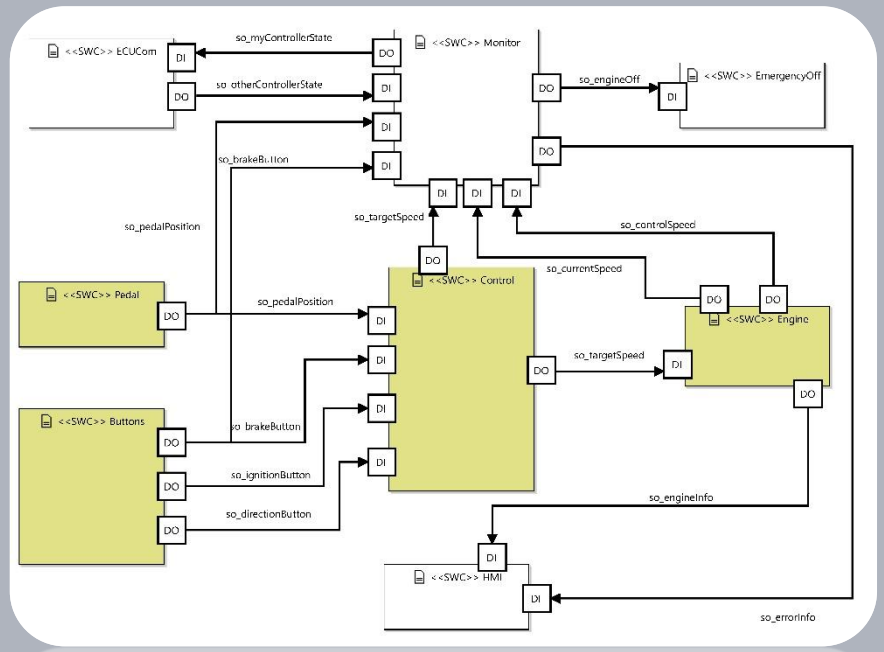
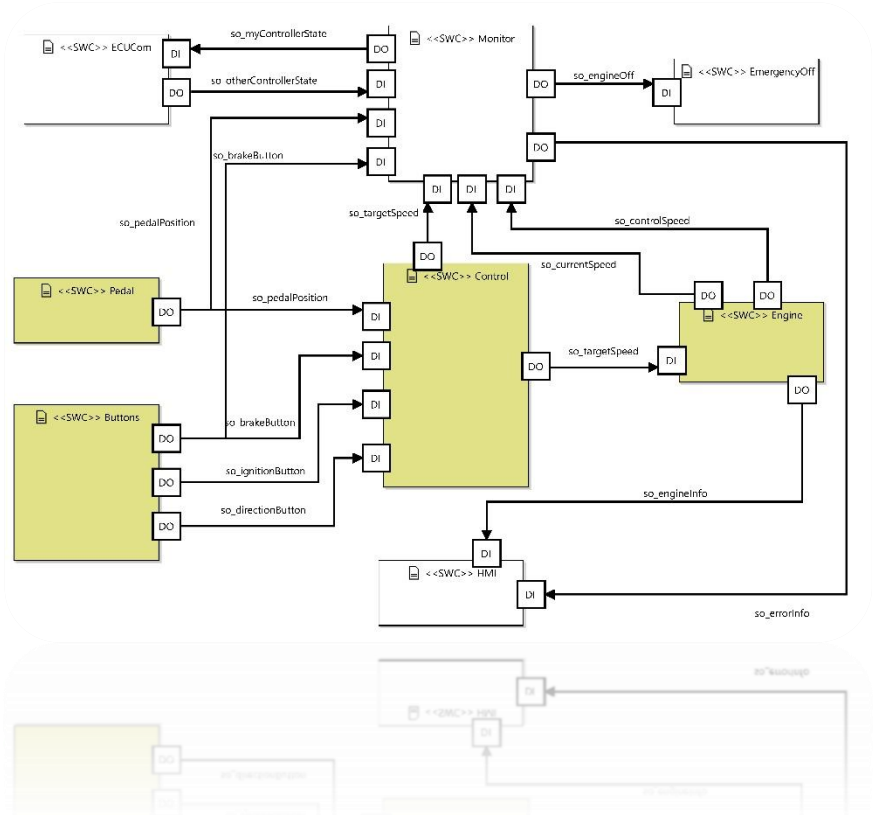
A showcase for model based code generation for multicore safety systems

Prof. Dr.-Ing. Peter Fromm



Content

- Motivation / modelling principles and goals
- An own modeling standard - the 7 Layer Model
  - Requirements and Safety Goals
  - Functional View
  - Hardware View
  - Software High Level View
  - Software Detailed View
  - Code View
  - Document View
- Behind the scenes
- Results and Outlook



Motivation

- Most safety standards require semi-formal or formal models for the architectural description of the system.

To avoid systematic faults in the software architectural design and in the subsequent development activities, the description of the software architectural design shall address the following characteristics supported by notations for software architectural comprehensibility; consistency; simplicity; verifiability; modularity; abstraction;

Table A.1 – Software safety requirements specification

(See 7.2)

	Technique/Measure *	Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1a	Semi-formal methods	Table B.7	R	R	HR	HR
1b	Formal methods	B.2.2, C.2.4	---	R	R	HR
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	R	R	HR	HR
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	R	R	HR	HR
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	R	R	HR	HR

NOTE 1 The software safety requirements specification will always require a description of the problem in natural language and any necessary mathematical notation that reflects the application.

NOTE 2 The table reflects additional requirements for specifying the software safety requirements clearly and precisely.

NOTE 3 See Table C.1.

NOTE 4 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

IEC61508

Table 2 — Notations for software architectural design

Notations		ASIL A	ASIL B	ASIL C	ASIL D
1a	Natural language[a]	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations[b]	+	+	++	++
1d	Formal notations	+	+	+	+

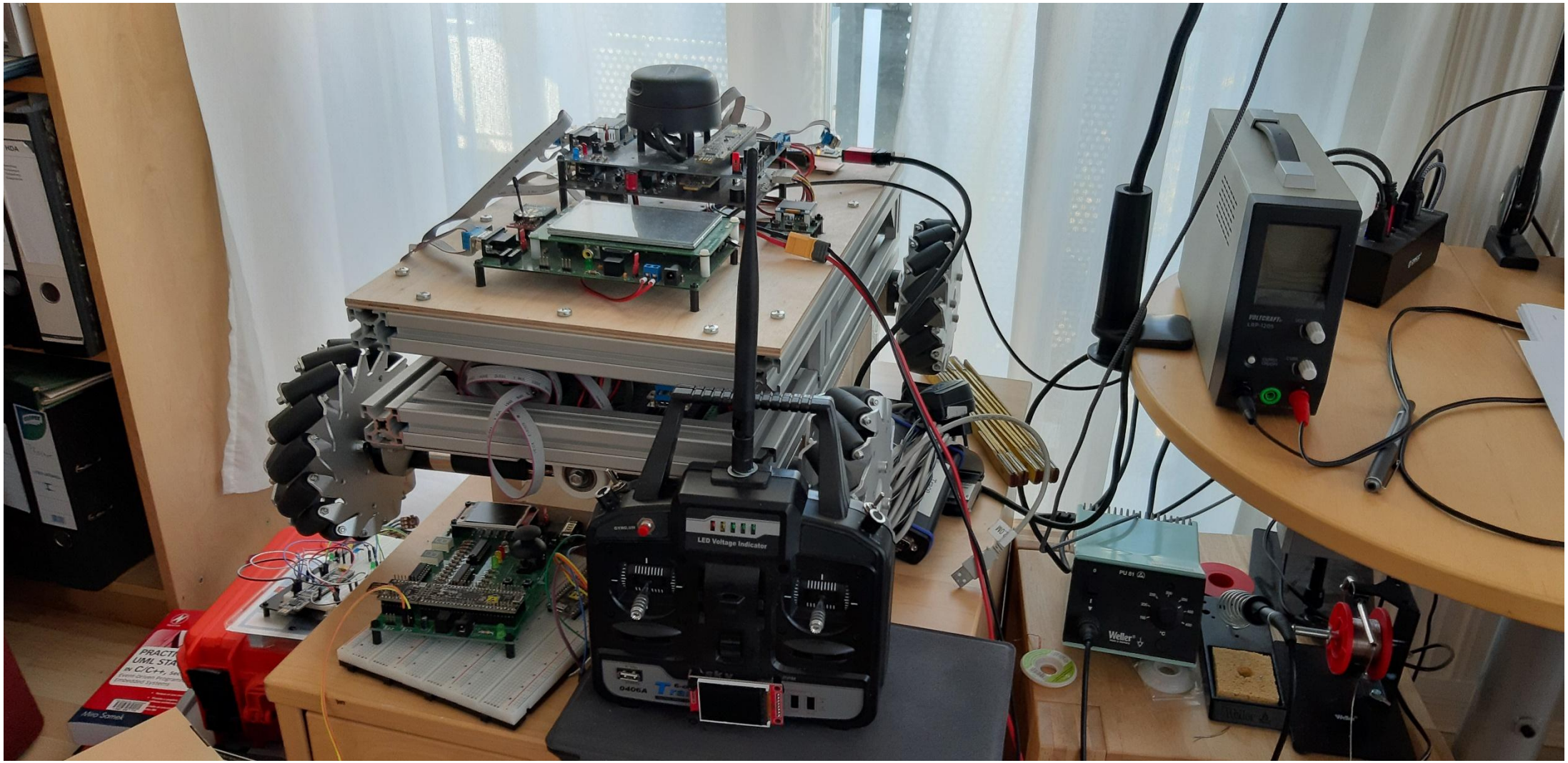
[a] Natural language can complement the use of notations for example where some topics are more readily expressed in natural language or providing explanation and rationale for decisions captured in the notation.

[b] Semi-formal notations can include pseudocode or modelling with UML®, SysML®, Simulink® or Stateflow®.

NOTE UML®, SysML®, Simulink® and Stateflow® are examples of suitable products available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of these products.

ISO26262

Example of a complex safety system – StudentCar (Corona Lab@Home)

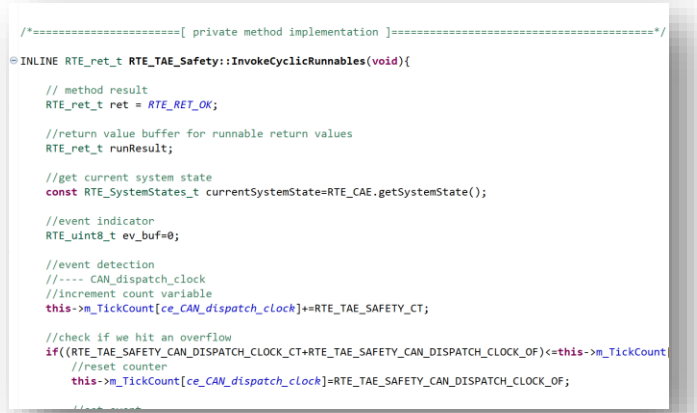


Trying to understand / maintain the architecture from code

Some metrics

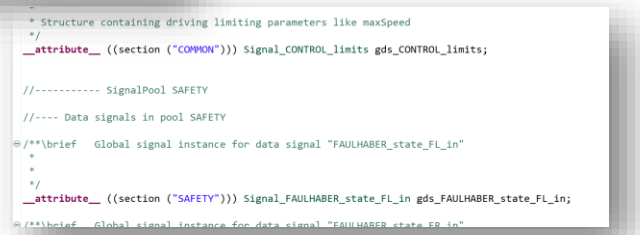
Files	318
Functions	1.947
Lines	239.369
Code Lines	53.269
Comment Lines	98.848
Executable Statements	19.839
Inactive Lines	52.064
Preprocessor Lines	43.153

Runnable execution in task context

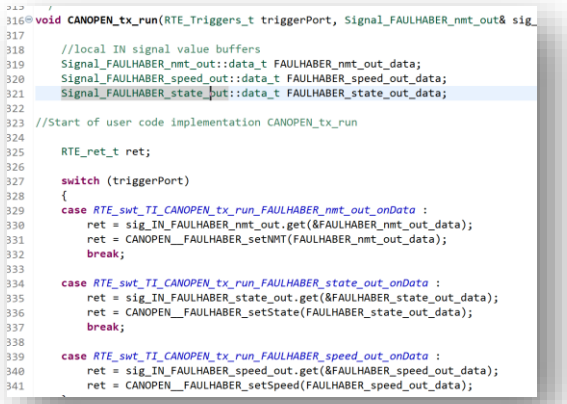


Task access right

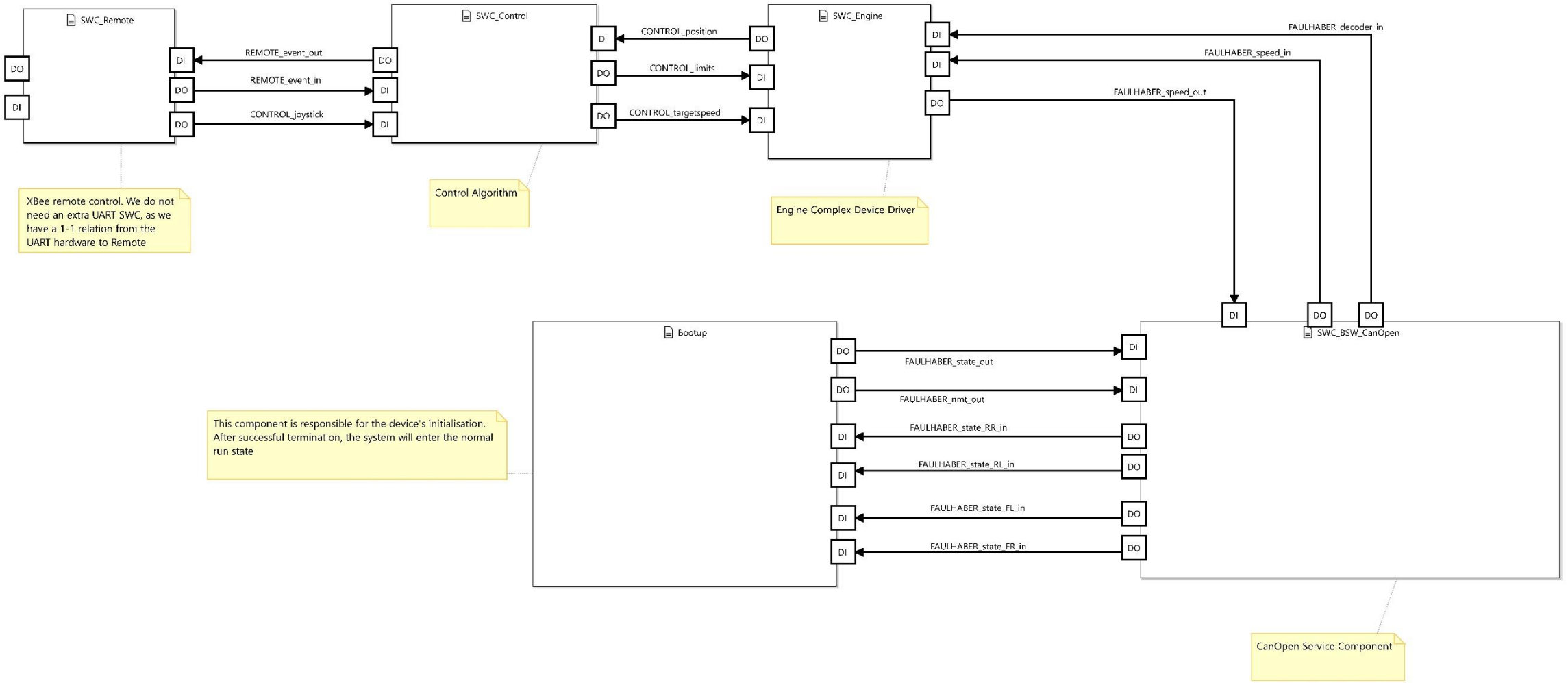
Data is mapped to different memory regions



Application is setting data



Alternative view



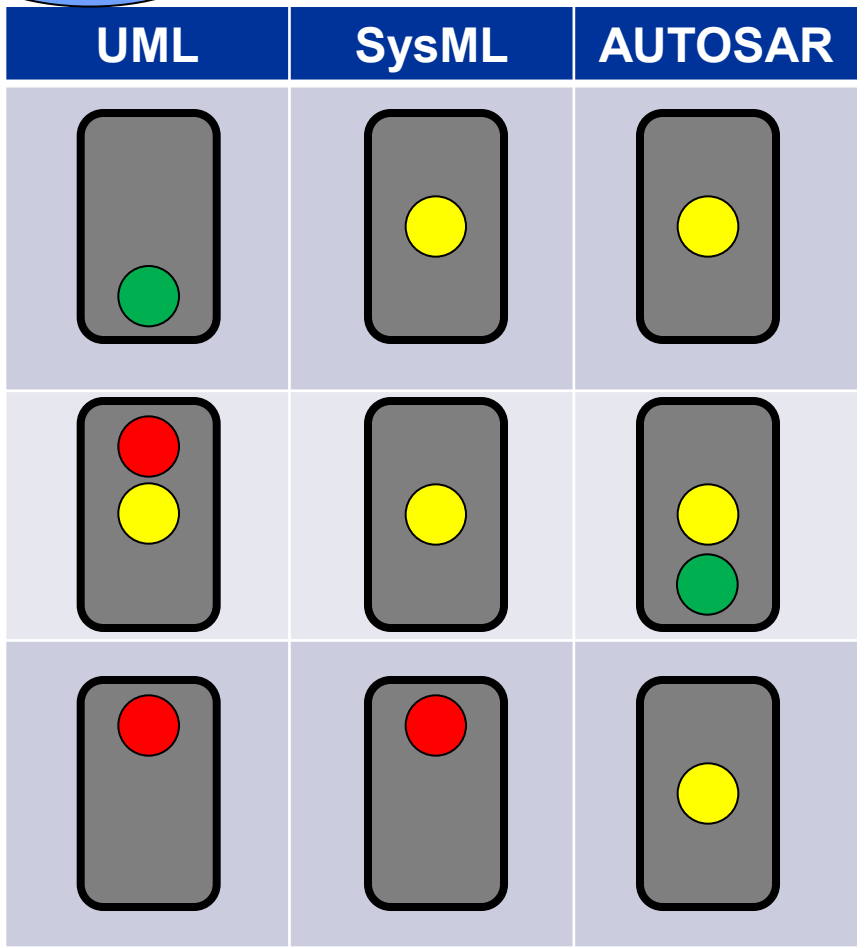
What do we expect from a good model?

How about requirements? And Traceability?

Static structure of the system
- What components are present in the system
- What are the responsibilities of the components
- What relations do I have between the components

Behaviour of the system
- Data flow between the components
- Execution events of the runnables

Some important technical details
- Execution context (core, task) of runnables
- Memory pools and memory protection
- Timing behaviour of the system



Understandability / Ease of use:

How about an own modelling language?

Goal:

- Simple modelling of the structural and runtime behaviour of single and multicore safety architectures.

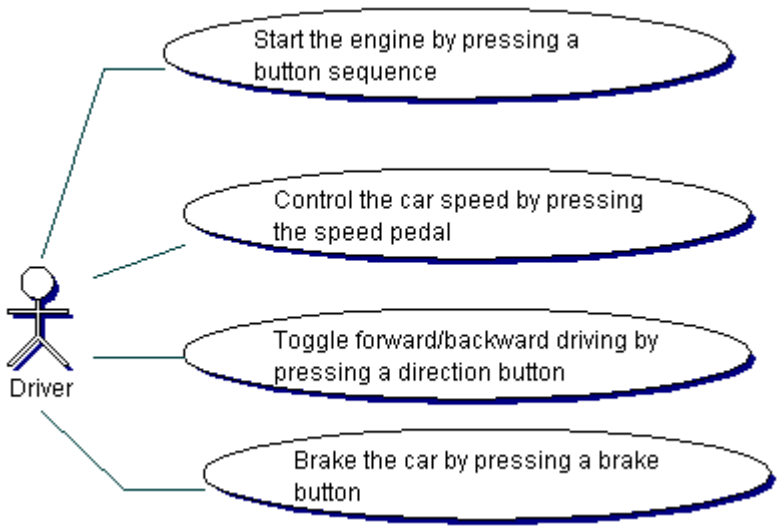
Functional Requirements:

- Describe (safety) requirements
- Create high level conceptual (functional) models
- Model the hardware (block diagram)
- Develop a (hierarchical) software model focussing on structural elements, signal flows and activation triggers
- Create C/C++ code for different controllers and RTOS
- Support safety assessments (model check, documentation)

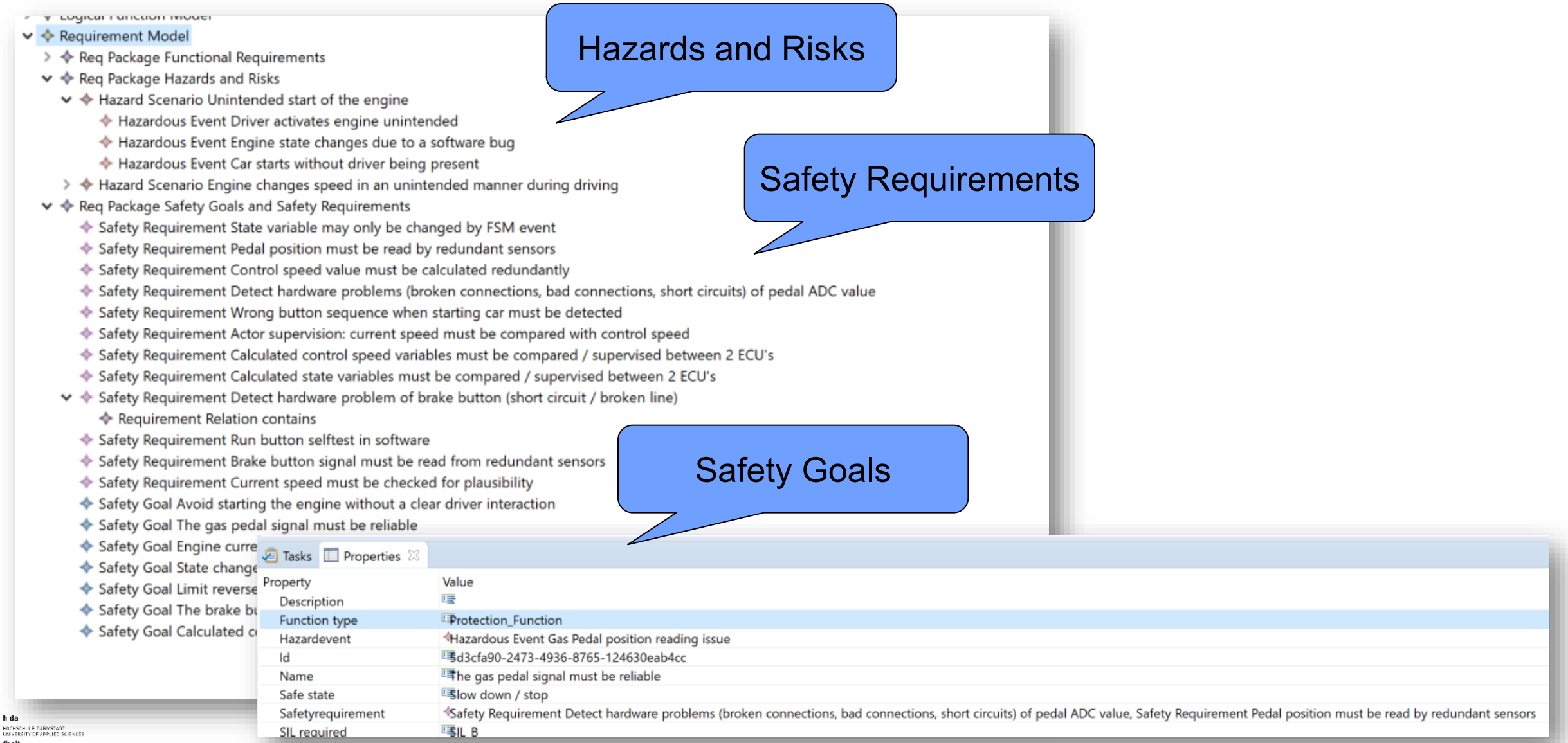
Non-Functional Requirements:

- Expressive models
- Well readable code
- Ease of use

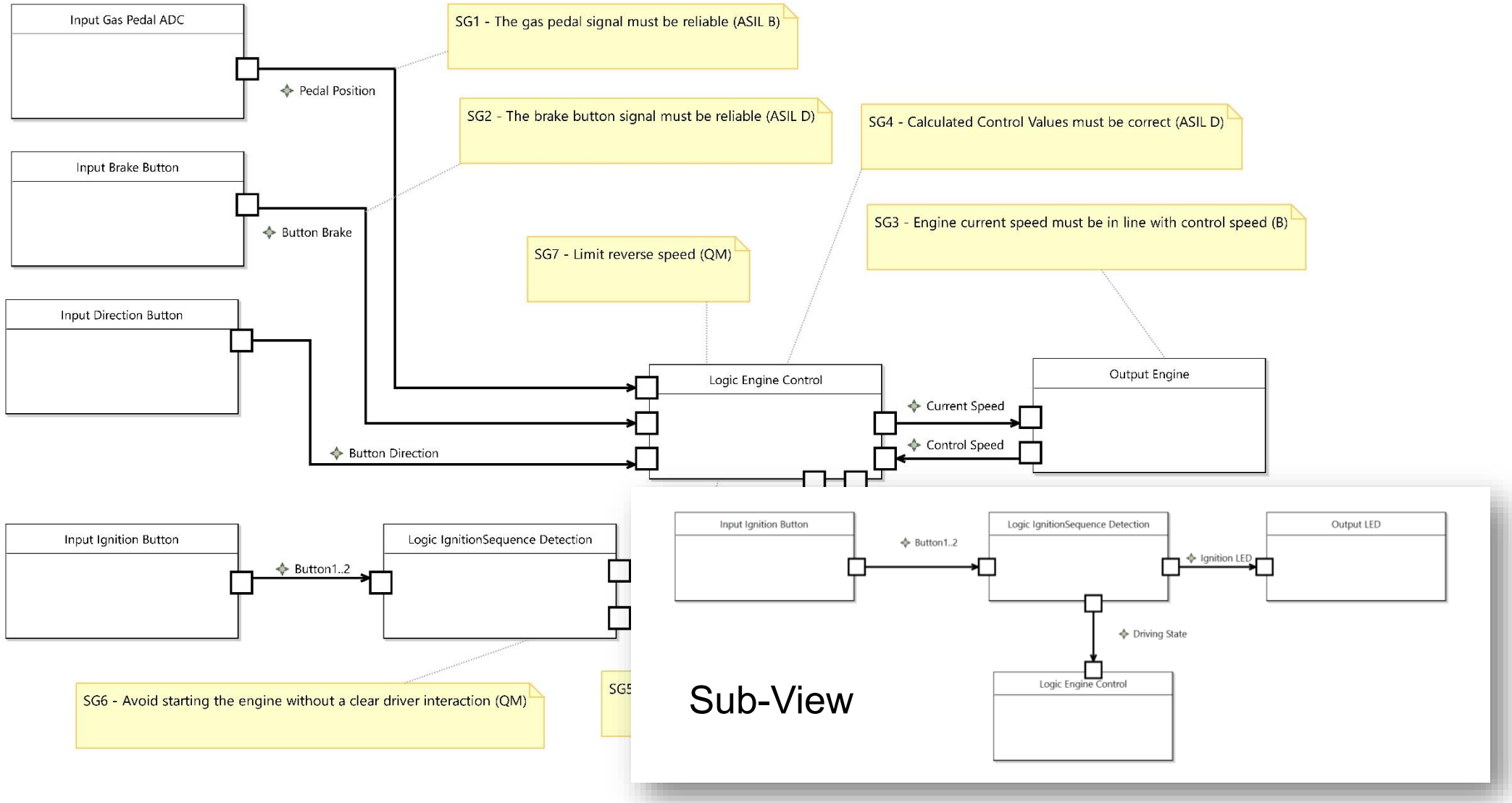
Our Use Case – Electronic Throttle Control



Layer 1 - Requirements View – Safety Focus

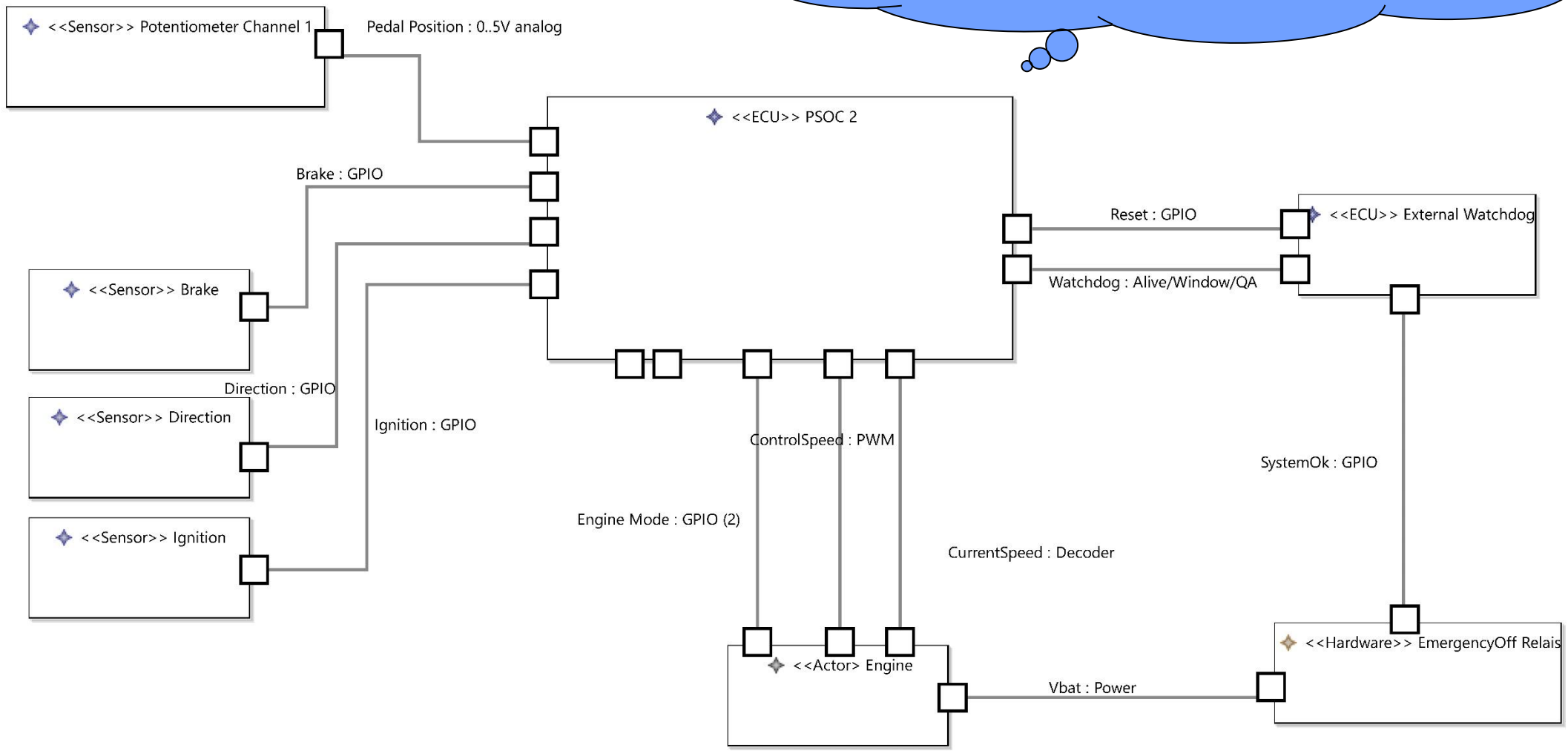


Layer 2 – Functional View



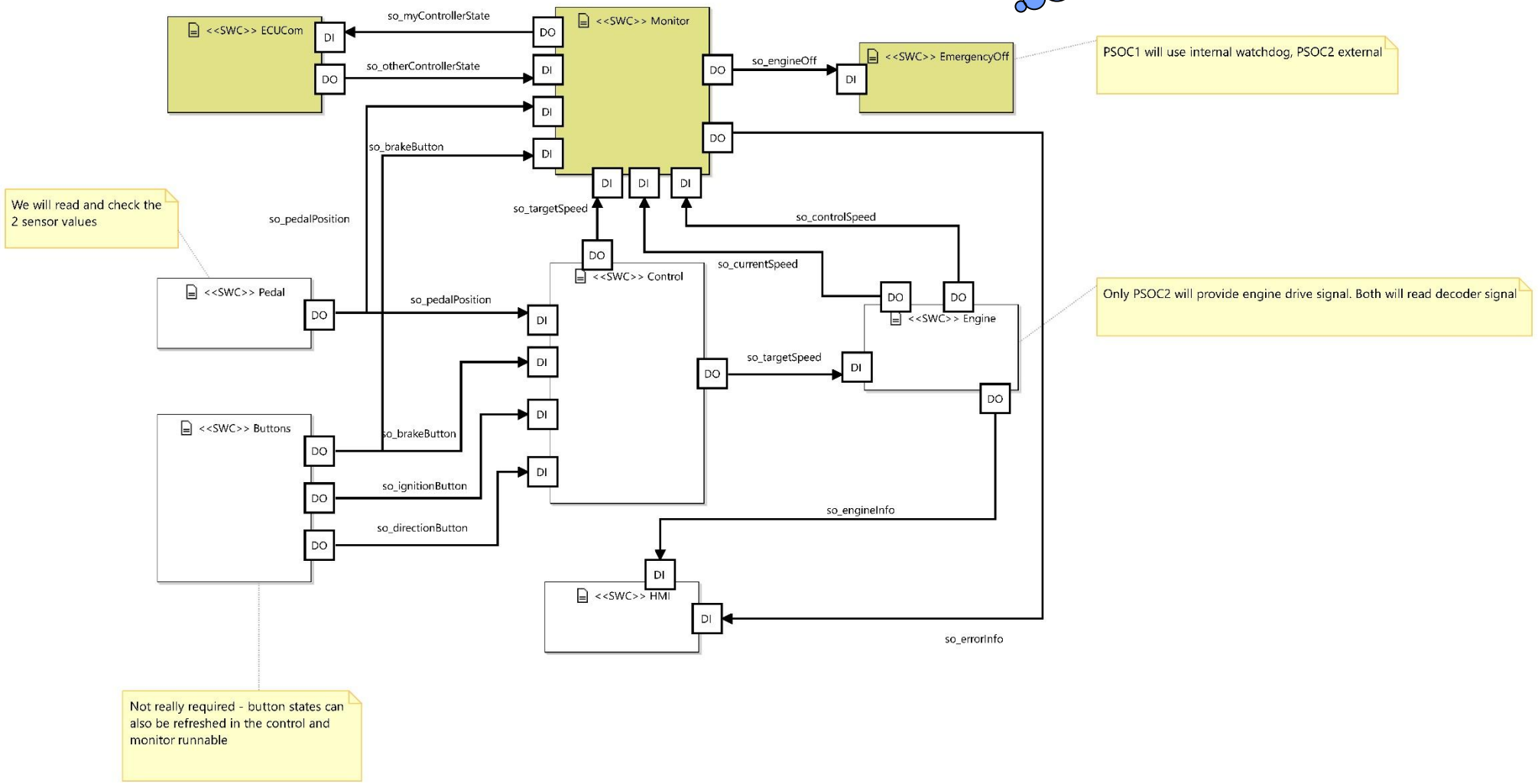
Layer 3 - Hardware View

Often close to the logical view, but more specific on the interfaces



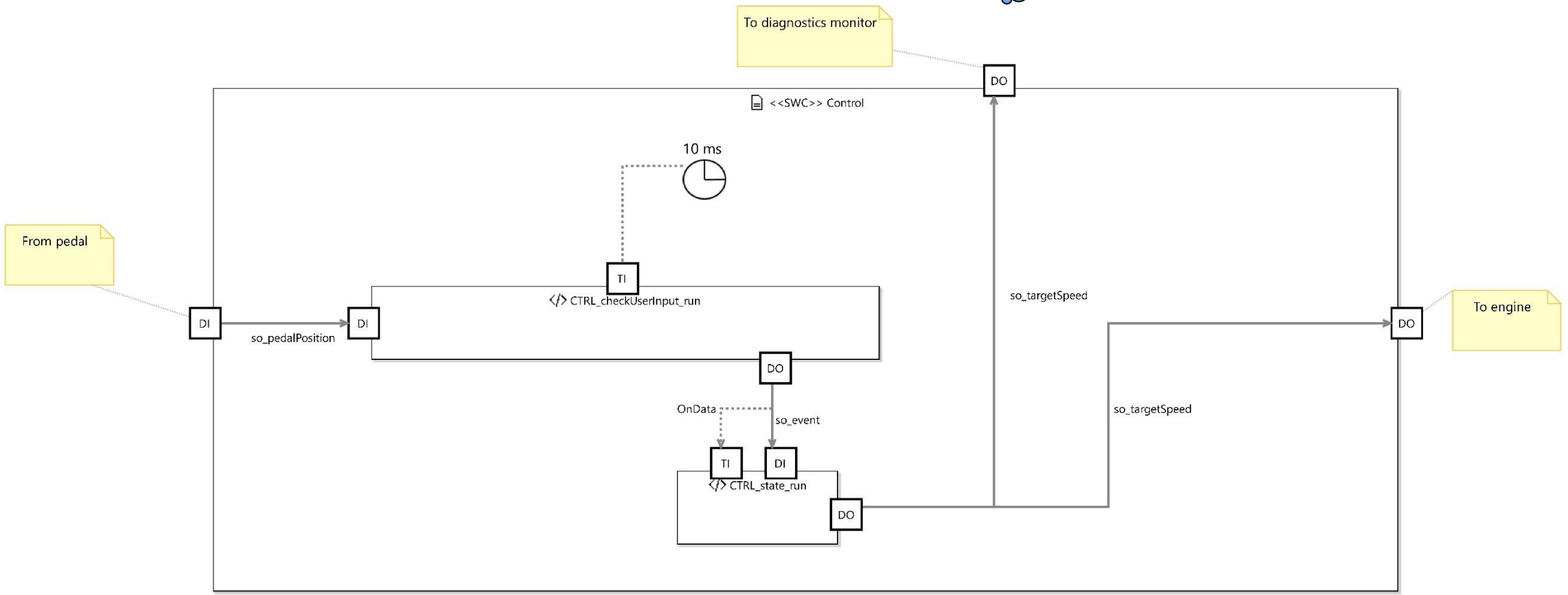
Layer 4 – Software Component View

Following the OOA/OOD philosophy. Hareware components are mapped to SWC.

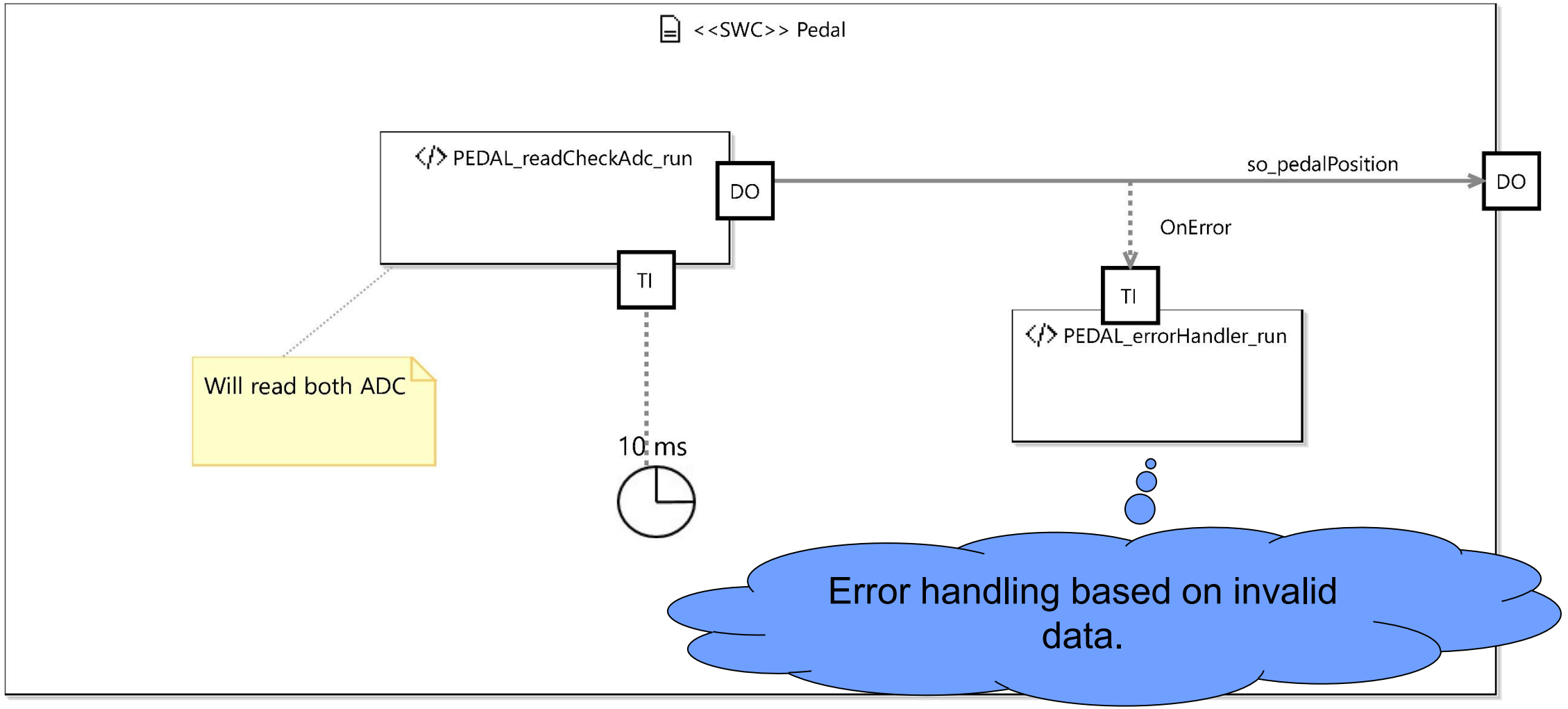


Layer 5 – Software Runnable View

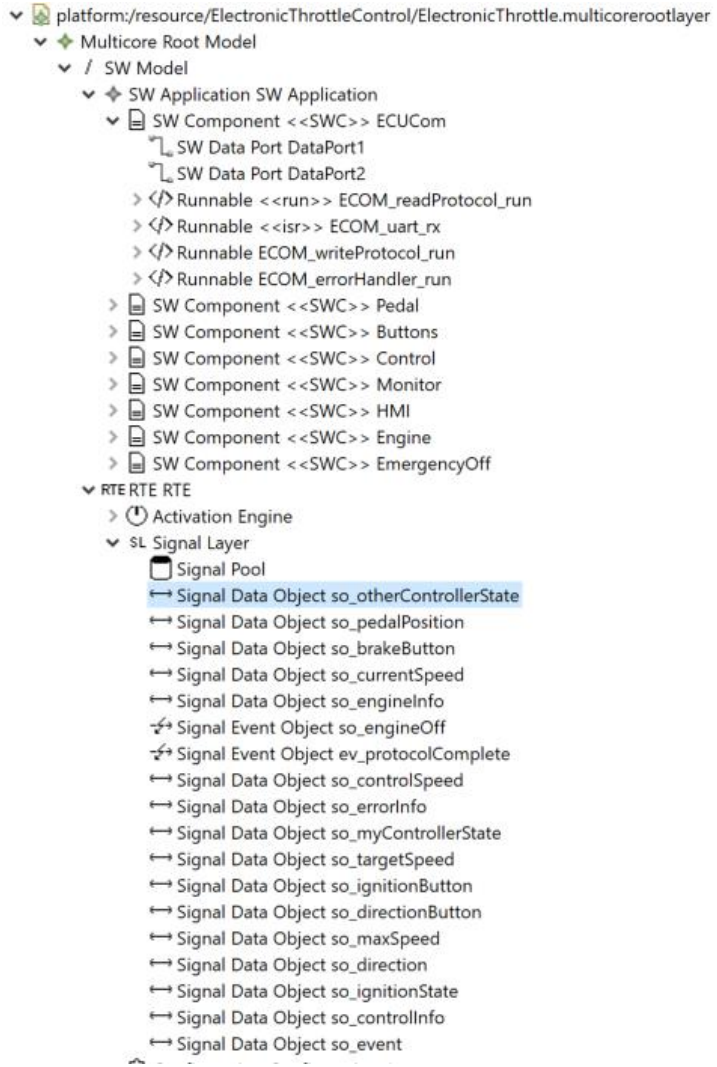
Data and activation triggers visible in a single picture!



Layer 5 – Software Runnable View



Layer 5 – Software Runnable View



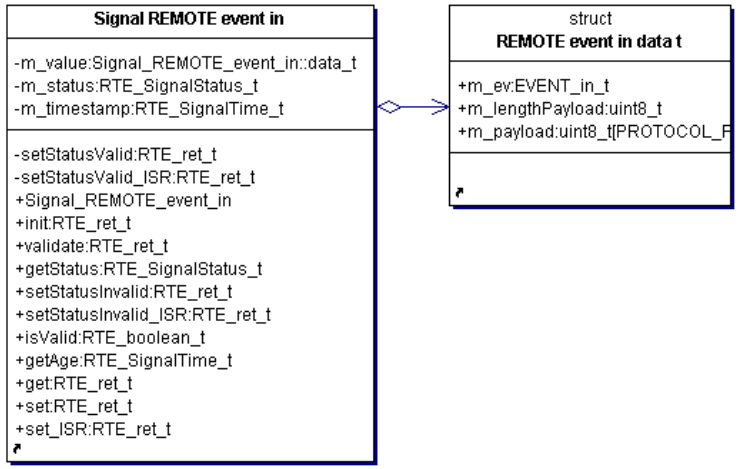
Code relevant details are added in a tree editor.

Tasks | Properties

Property	Value
Age Max	0
Age Min	0
Allow ptr access	false
Description	
Id	3378028f-d21a-46d8-99a2-a7e4af942d12
Indriver	
Init Value	
Inline	false
Is Local	false
Name	so_otherControllerState
On Data	
On Driver	
On Error	SW Trigger Port TriggerPort1
Outdriver	
Signalpool	
Type	

Layer 6 - Code

Signals with fixed API (local or global)

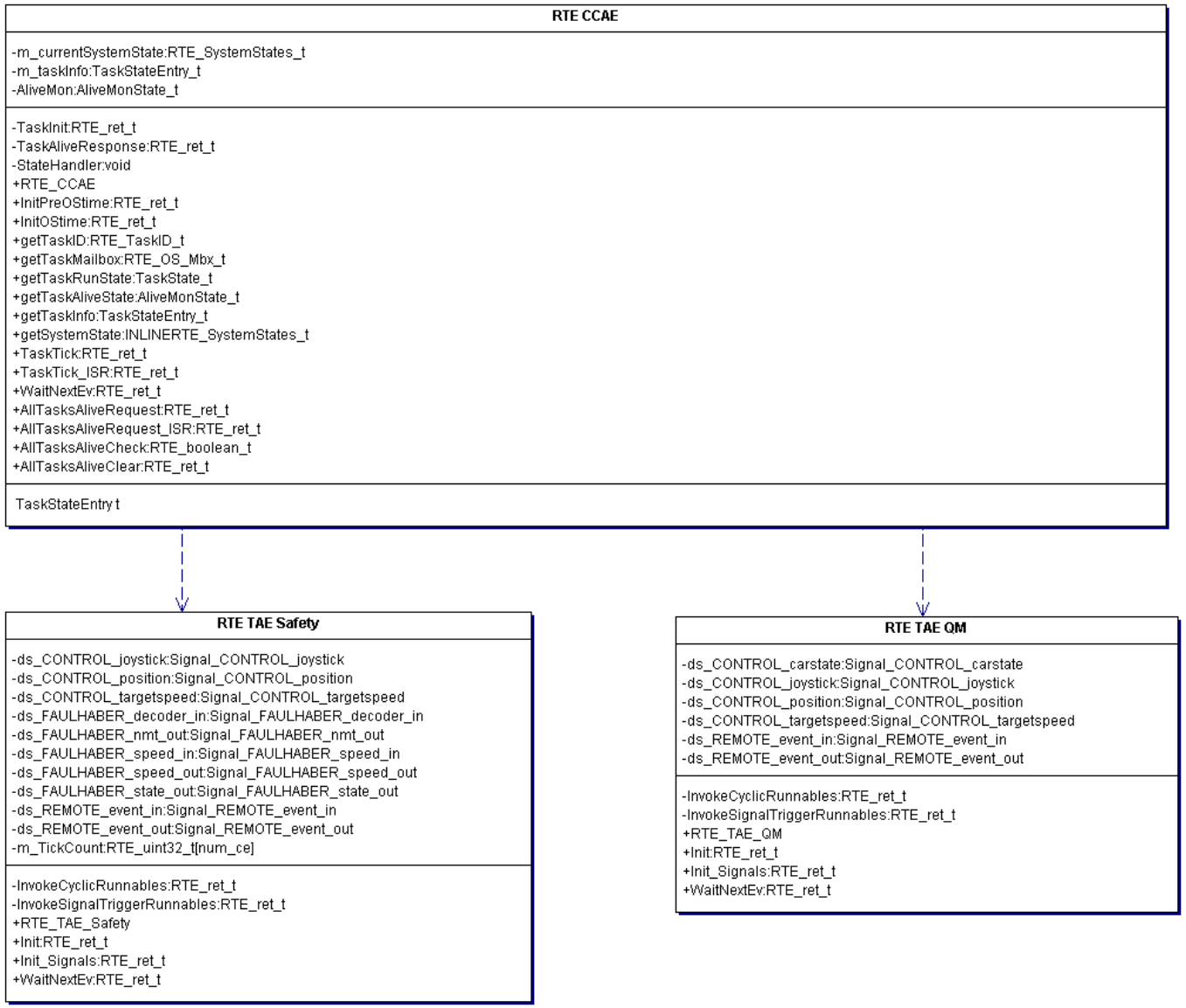


Software Runnables with references to the connected signals

```
void CONTROL_drive_run(RTE_Triggers_t triggerPort, Signal_CONTROL_carstate& sig_IN_CONTROL_carstate, Signal_CONTROL_joystick& sig_IN_CONTROL_joystick,
        Signal_CONTROL_position& sig_IN_CONTROL_position, Signal_CONTROL_limits* const p_sig_OUT_CONTROL_limits,
        Signal_CONTROL_targetspeed* const p_sig_OUT_CONTROL_targetspeed){

    //local IN signal value buffers
    Signal_CONTROL_carstate::data_t CONTROL_carstate_data;
    Signal_CONTROL_joystick::data_t CONTROL_joystick_data;
    Signal_CONTROL_position::data_t CONTROL_position_data;

//Start of user code implementation CONTROL_drive_run
    /*TODO NOTE:The carstate is ignored in this first draft !*/

    RTE_ret_t result = RTE_RET_NO_STATUS;
    RTE_ret_t valResult = RTE_RET_NO_STATUS;

    Signal_CONTROL_targetspeed::data_t  targetspeed_data = CONTROL_TARGETSPEED_INIT_DATA;
    Signal_CONTROL_limits::data_t  limits_data = CONTROL_LIMITS_INIT_DATA;

    //Get the data, ignore error cases at the moment, should be done by application (hacky)
    result = sig_IN_CONTROL_carstate.get(&CONTROL_carstate_data);
    result = sig_IN_CONTROL_joystick.get(&CONTROL_joystick_data);

    //Check for Joystick signal age

    //Todo ================================================================

    if (triggerPort == RTE_swt_TI_CONTROL_drive_run_10_0ms)
    {

        switch (CONTROL_carstate_data.m_algoIndex)
        {
        case 0 : DRIVE_0::calibrate_run(sig_IN_CONTROL_joystick, &targetspeed_data); break; //Calibration test
        case 1 : DRIVE_0::manual_run(sig_IN_CONTROL_joystick, &targetspeed_data); break; //Manual

        default : /* call error handler */;
        }

        //usbLog.printf_d(Log::DBG, "CTRL_calcSpd", "y,x,phi", 3, (double)targetspeed_data.m_vy, (double)targetspeed_data.m_vx, (double)targetspeed_data.m_vphi);
    }
```

Layer 6 - Code

Runtime Environment (Runnable Activation)



Linker Control File and Task Configuration (MPU)

```
SECTIONS{

    /* In MonoBlock linkage mode, all data will be linked to a single section
     *
     * Memory will be allocated at RAM ()
     */
    .RTE.MONOBLOCK : {

            . = ALIGN(/* Start of user code monoBlock_alignment_start */
                    _AE_Monoblock_Size_p2
                  /*End of user code */);

        PROVIDE(ADRL_RTE_CAE_RW_BEGIN = .);
        PROVIDE(ADRL_RTE_TAE_RW_BEGIN = .);

            /* CAE data */
            *(.rte.cae.base)

            /* TAE "Safety" */
            *(.rte.tae.Safety.state)
            *(.rte.tae.Safety.AliveMon)

            /* TAE "QM" */
            *(.rte.tae.QM.state)
            *(.rte.tae.QM.AliveMon)

            . = ALIGN(/* Start of user code monoBlock_alignment_CAE_RW */
                    _CAE_region_Size_p2
                  /*End of user code */);

        PROVIDE(ADRL_RTE_CAE_RW_END = .);
            /* Start of user code monoBlock_assert_CAE_RW */
            ASSERT (((ADRL_RTE_CAE_RW_END - ADRL_RTE_CAE_RW_BEGIN) == _CAE_region_Size_p2),
            /*End of user code */

            /* Runnable Monitor/Trace per TAE*/
            *(.rte.tae.Safety.RunMon)
            *(.rte.tae.QM.RunMon)
```

Layer 7 - Documentation

Descriptive In-Code Documentation

```
/* --- Details for runnable "CONTROL_drive_run" ---
 *  Description:
 *
 *  SIL required QM, SIL achieved QM. Justification:
 *  System States:
 *      normal ()
 *  Signal association
 *      InPorts:
 *          Trigger IN
 *              Trigger Name "TI_CONTROL_drive_run_10_0ms"
 *              Task: "QM"
 *              System Events
 *                  Cyclic Triggers:
 *                      "10_0" Cycle Time: 10; Offset 0
 *          Data IN
 *              DataPort Name "DI_CONTROL_carstate"
 *              - Associated Signal "CONTROL_carstate":
 *                  Description:    The overal car connection state
 *                  SignalPool:     local signal
 *                  Signal-Type:    generated, see #CONTROL_carstate_cfg.h
 *              DataPort Name "DI_CONTROL_joystick"
 *              - Associated Signal "CONTROL_joystick":
 *                  Description:    The joystick read from Zigbee protocol
 *                  Maximum Age:    300
 *                  SignalPool:     local signal
 *                  Signal-Type:    generated, see #CONTROL_joystick_cfg.h
 *              DataPort Name "DI_CONTROL_position"
 *              - Associated Signal "CONTROL_position":
 *                  Description:    Position based on dear reckoning
 *                  SignalPool:     local signal
 *                  Signal-Type:    generated, see #CONTROL_position_cfg.h
 *      OutPorts:
 *          Data OUT
 *              DataPort Name "DO_CONTROL_limit"
 *              - Associated Signal "CONTROL_limits":
 *                  Description:    Structure containing driving limiting parameters like maxSpeed
 *                  SignalPool:     gds_CONTROL_limits in "COMMON" ()
 *                  Signal-Type:    generated, see #CONTROL_limits_cfg.h
 *              DataPort Name "DO_CONTROL_targetspeed"
 *              - Associated Signal "CONTROL_targetspeed":
 *                  Description:    The car targetspeed
 *                  SignalPool:     local signal
 *                  Signal-Type:    generated, see #CONTROL_targetspeed_cfg.h
 */
void CONTROL_drive_run(RTE_Triggers_t triggerPort, Signal_CONTROL_carstate& sig_IN_CONTROL_carstate, Signal_CONTROL_joystick& sig_IN_CONTROL_joystick,
        Signal_CONTROL_position& sig_IN_CONTROL_position, Signal_CONTROL_limits* const p_sig_OUT_CONTROL_limits,
        Signal_CONTROL_targetspeed* const p_sig_OUT_CONTROL_targetspeed){
```

Markup Language System Description

Activation engine

System states

bootup

normal

error_detected

Tasks

Overview

Task Name	Cyclic Events	System Events	Cycle Time
Safety	CANopenTickClock, CAN_dispatch_clock, 50_0, 50_25, Bootup_Timer	SafetyHasBooted	1
QM	10_0		10

Activation for task “Safety”

System:

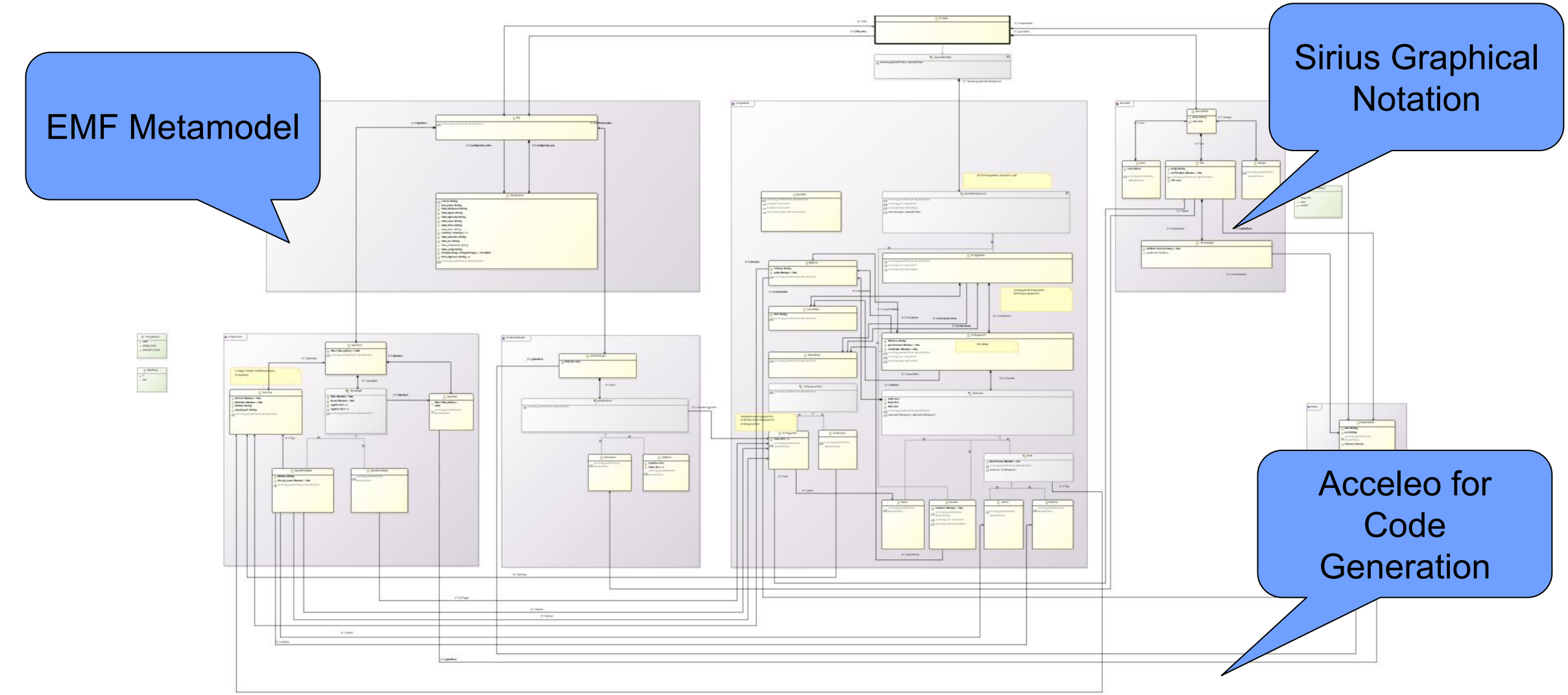
Init event “SafetyHasBooted”:

Runnable Name	Guards	SIL Level	System State	Order	has Return
BOOT_peripherallnit_run	none	bootup	0	true	

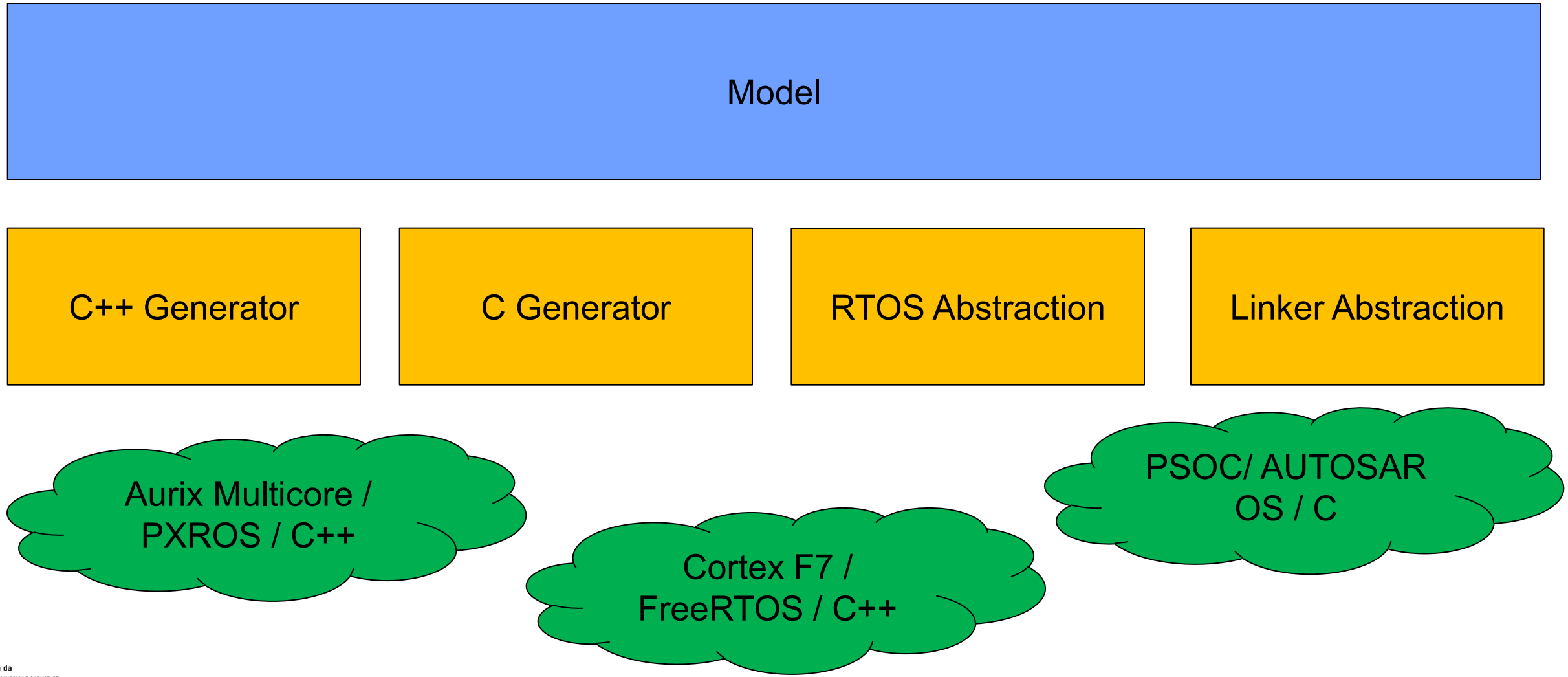
Cyclic:

The cycle time of this task is **1 ms**:

Behind the scenes



Supported Formats



Summary

- Although we have created another modelling language, the “dialect” has proven to be rather intuitive also for engineers new the topic.

- All relevant information can be found directly in the model, less relevant details can be added in a tree editor structure.
- Engineering focus moves from coding to designing.

- Generated code is well readable and testable.
- Debugging efforts are significantly reduced.

- Is Eclipse EMF the right choice?

There (stil) is no silver bullet, but we have come a bit closer.

Outlook – Moving Towards a 8 Layer Model

As we have full traceability in the model from requirements down to generated code, an additional Analytical Layer is planned:

(A)SILreq versus (A)SILach
- Compare the required (A)SIL with the reported (A)SIL

Check Freedom of Interferance of the SIL/QM functions
- Memory
- CPU
- Peripherals

Hardware/Software Validation
- Is (A)SIL code only executed on safe hardware (e.g. lockstep cores)

Contact

Prof. Dr.-Ing. Peter Fromm
M.Sc. Thomas Barth

Hochschule Darmstadt - University of Applied Sciences
FB Elektrotechnik und Informationstechnik (EIT)
Birkenweg 8
64295 Darmstadt

Email: peter.fromm@h-da.de
thomas.barth@h-da.de
Web: www.eit.h-da.de