A case study: secure information flow in AUTOSAR models

Automotive systems: Mixed-criticality safety critical systems

High criticality
Braking system, Throttle system, …

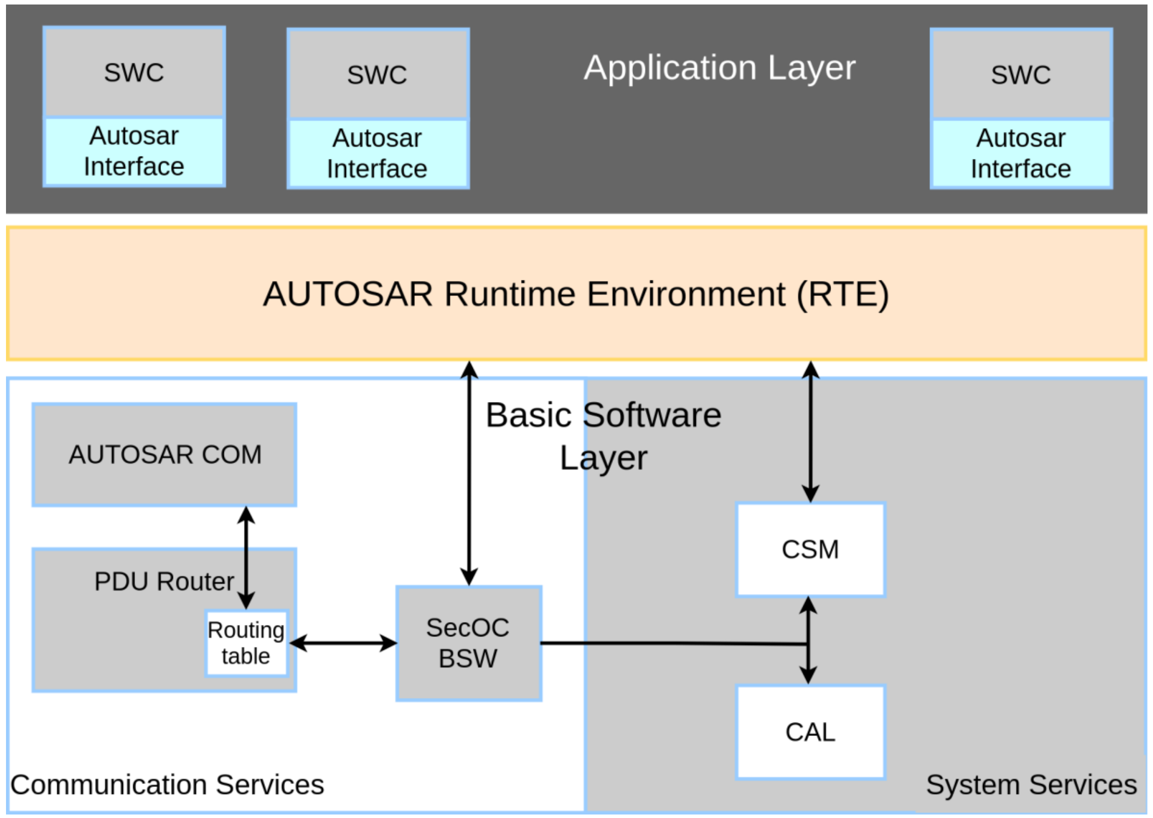
Low criticality
Infotainment system, ..

Recent research has shown that it is possible for external intruders to compromise the proper operation of safety functions getting access to the infotainment system.

Low security level data must not compromise the computation of high criticality functions

A case study: secure flow in AUTOSAR models

AUTomotive **O**pen **S**ystems **AR**chitecture: open industry standard for automotive software architectures, spanning all levels, from device drivers, to operating system, communication abstraction layers and the specification of application-level components



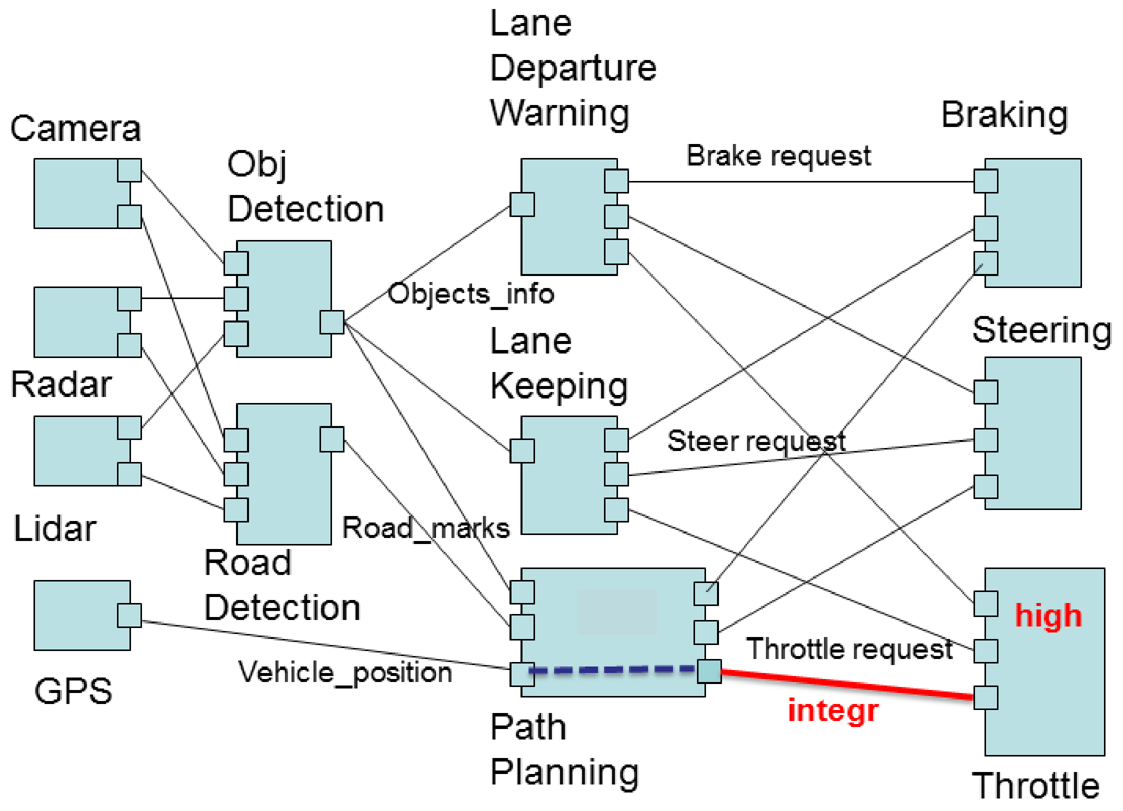
Mixed-criticality

Path Planning, Lane Keeping and Lane Departure Warning are active safety functions that receive such data and send commands to actuators (steering, throttle and brakes).

- Throttle component is assigned the high trust level;

- Throttle request link is assigned the integrity security requirement.

Autonomous driving



AUTOSAR models are extended with security annotations.

Mixed-criticality

Data received by Throttle on the link Throttle_request must satisfy integrity security requirement

The point is that:
the way in which security annotations are specified must consider the causal dependencies between data that traverse the model.

If Throttle requires integrity on its input data sent by Path Planning,
then integrity must be guaranteed also along the path from the data originator (GPS) to Path Planning (the Vehicle_position link),
otherwise, the security constraint cannot be satisfied and the set of annotations is not correct.

AUTOSAR security annotations

The simplest solution assigns integrity to all links directly or indirectly connected to Throttle_request.

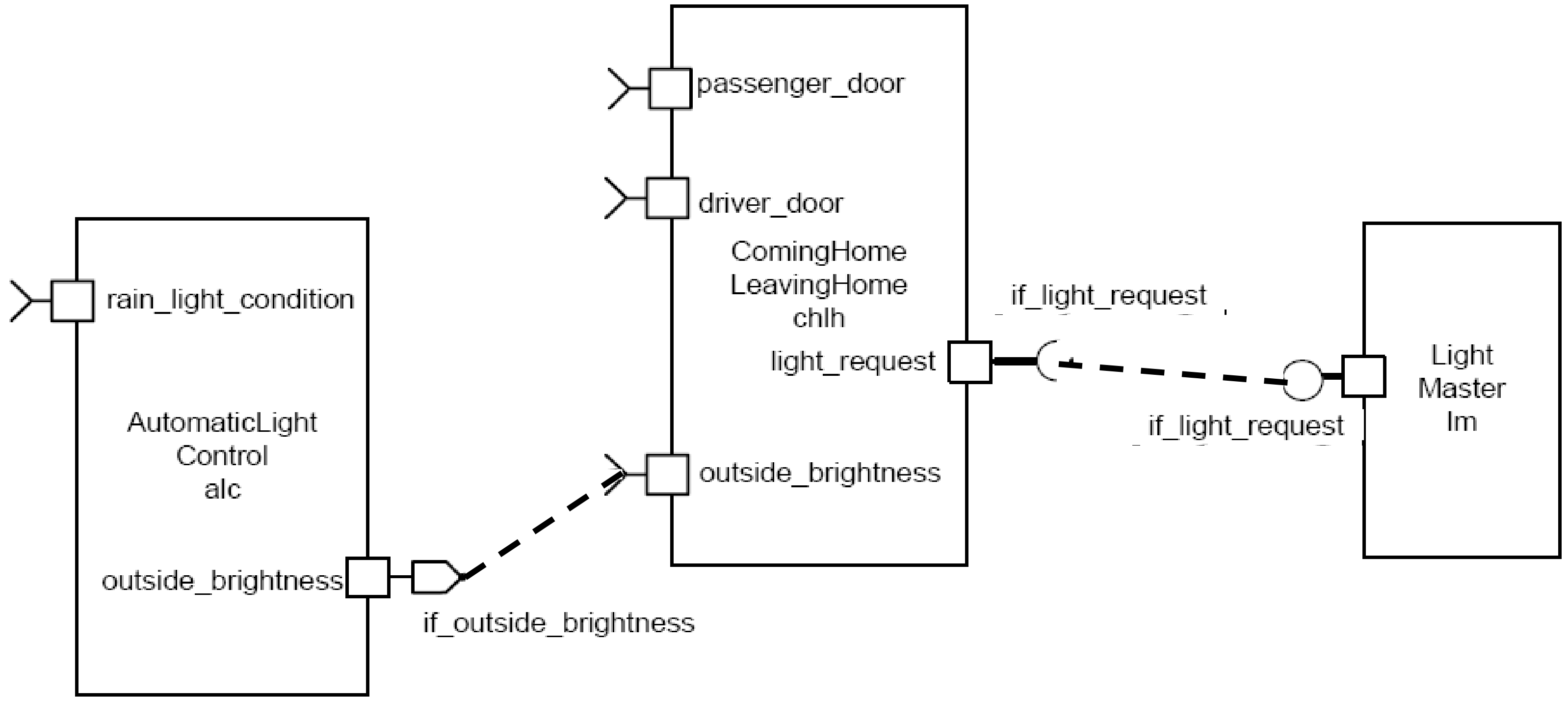
In order to obtain a more efficient solution, information flow theory can be exploited

AUTOSAR architecture

A fundamental concept of AUTOSAR is the separation between:

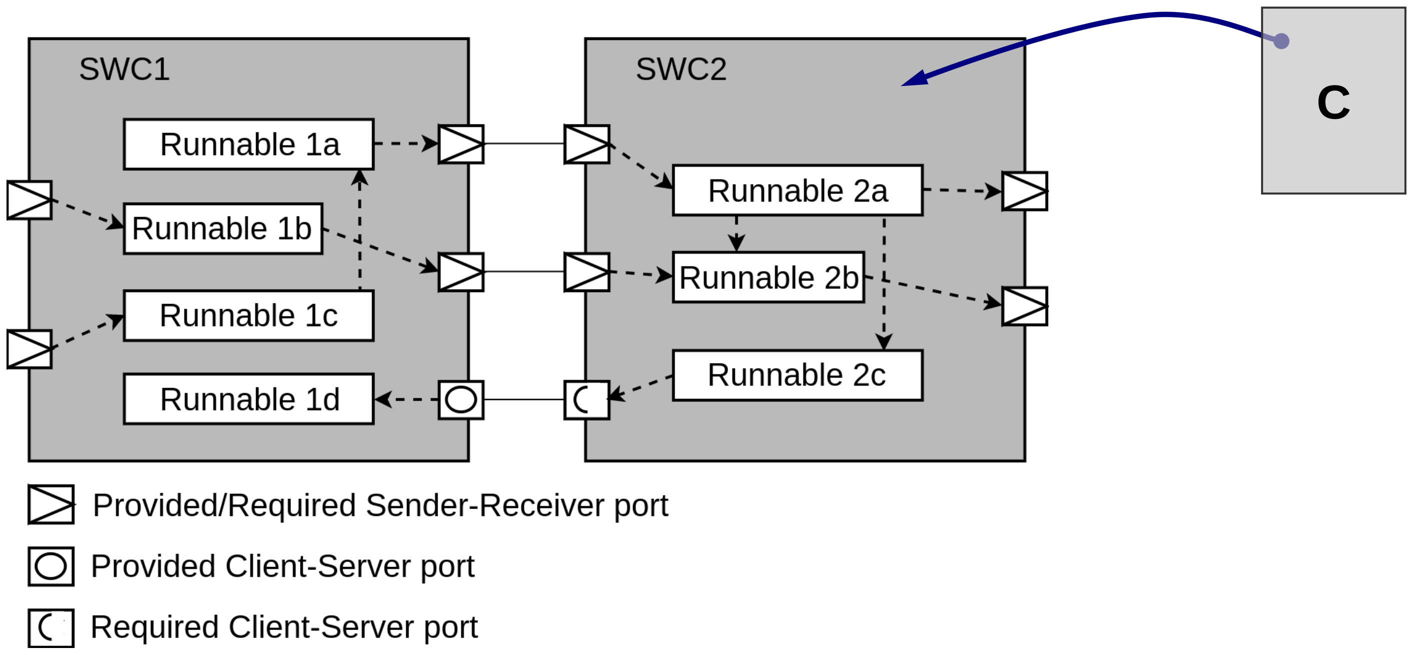
- **application** and
- **infrastructure**.

An application in AUTOSAR consists of Software Components interconnected by connectors



Runnables

- Runnables define the behavior of components
- Runnables are entry points to code-fragments and are (indirectly) a subject for scheduling by the operating system.

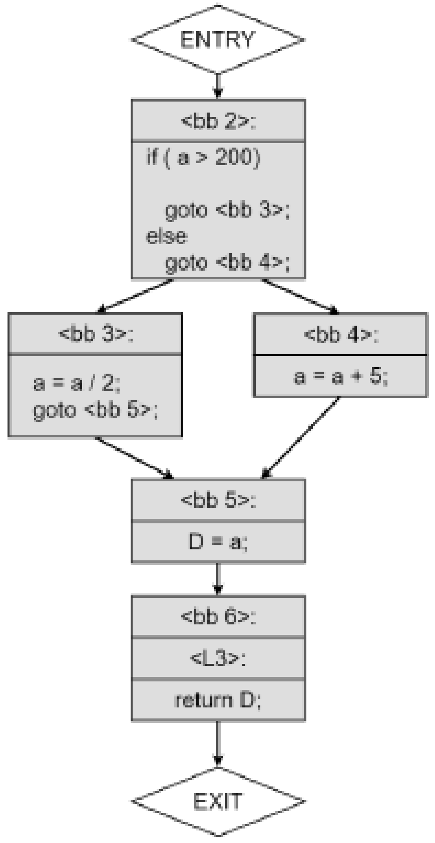


SWC1
Runnable 1a
Runnable 1b
Runnable 1c
Runnable 1d
SWC2
Runnable 2a
Runnable 2b
Runnable 2c
C
Provided/Required Sender-Receiver port
Provided Client-Server port
Required Client-Server port

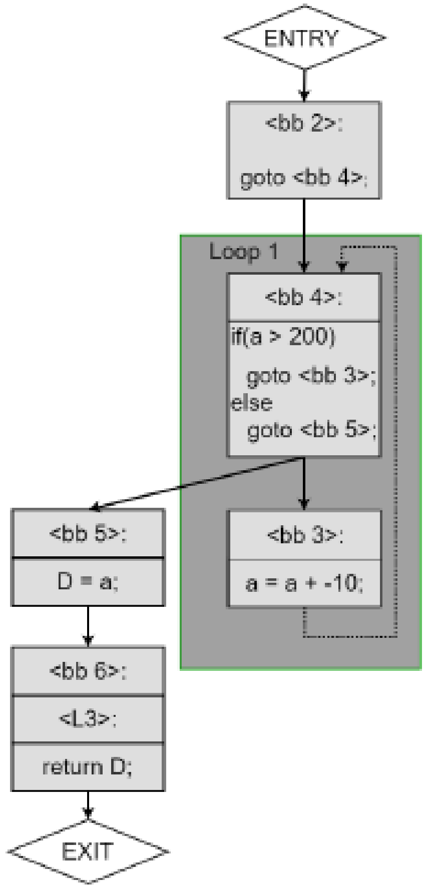
Control flow graph

```
int runnable1 (int a) {
   if (a>200)   a = a/2;
    else   a= a+5;
   return a;
}
int runnable2 (int a) {
   while (a>200)
        a = a -10;
   return a;
}
```

Visual Studio C++
gcc *-fdump-tree-cfg*



runnable1



runnable2

AUTOSAR runnable interaction

Runnable interaction

Global variables

Ports define interaction points between (runnables belonging to) different SWCs.

For interactions among runnables belonging to the same component
Inter Runnable Variables (IRVs)

The RTE provides protection mechanisms for IRVs (as opposed to global variables)

AUTOSAR Secure Flow analysis

An AUTOSAR model satisfies data secure flow if data sent on a link at run-time, always have a security requirement not lower than those specified by the security annotations.

For each link, we compute:

- the security requirement of data sent on the link

The abstract interpreter: EXEC

Each runnable is executed starting from the abstract memory and the context file, and applying the abstract rules.

All branches of conditional/iterative instructions are always executed, due to the loss of real data in the abstract semantics

Abstract semantics

A PORT is a variable.

RTE function for reading from or writing onto ports are mapped to read and write of the port variable.
For simplicity, the name of the port variable is equal to the name of the port.

RTE functions that invoke remote services trigger the runnable that implements the service. The function implementing the service is invoked

Abstract semantics

A POINTER is assumed to be simple variable, that maintains the dependencies of the pointer, plus the dependencies of the pointed data in the abstract execution.

An ARRAY is assumed to be a simple variable, that maintains the whole dependencies of each element in the array.

A STRUCTURED VARIABLE is mapped to a set of simple variables, one for each member (we use the notation, as usual). If we have a variable data that is a structure with two fields a and b, we map such variable into two simple variables, data:a and data:b, respectively.

Iterative analysis

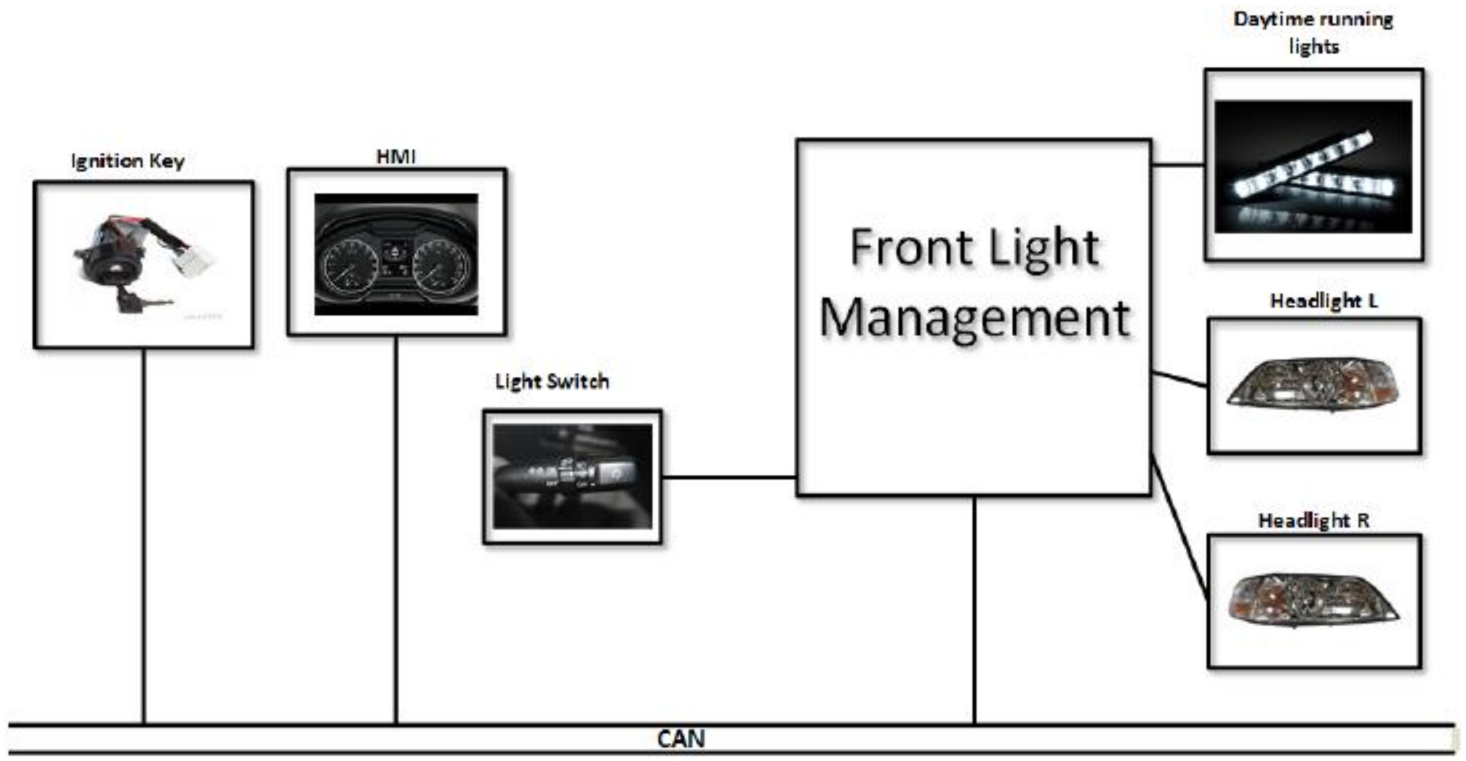
Iterative analysis until fixpoint is reached

A: security context

R: set of all runnables

```
$A := A^{\emptyset}$
$T := R$
while$(T \neq \emptyset)$
    select $r \in T$
    $T := T - \{r\}$
    $A' := EXEC(r, A)$
    if$(A' \neq A)$
        $A := A'$
        $T := R$
```

An example: Front Light Manager



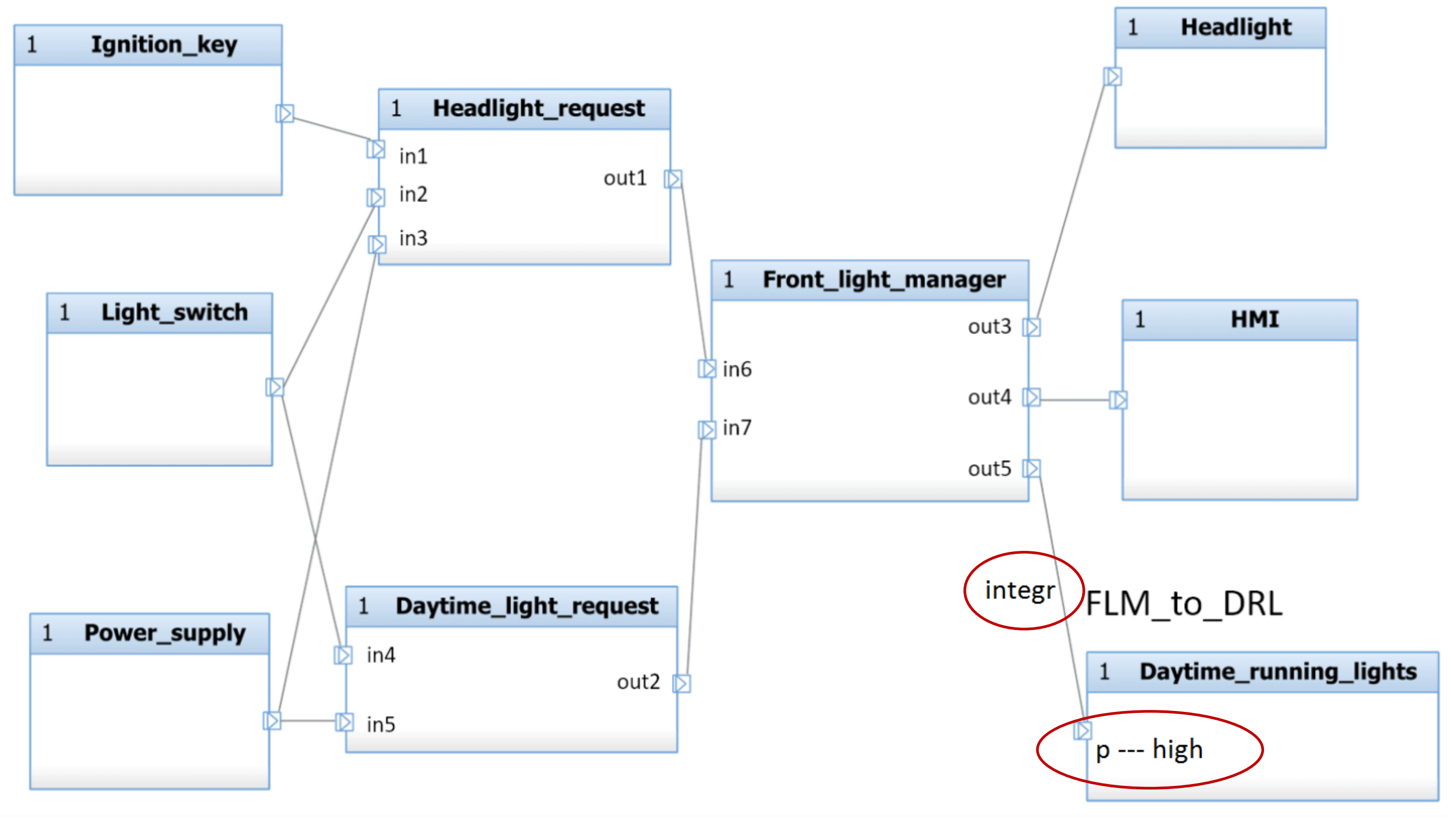
Safety Use Case Example, release 4.2.2. http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/safety-and security/auxiliary/AUTOSAR_EXP_SafetyUseCase.pdf

Front Light Manager

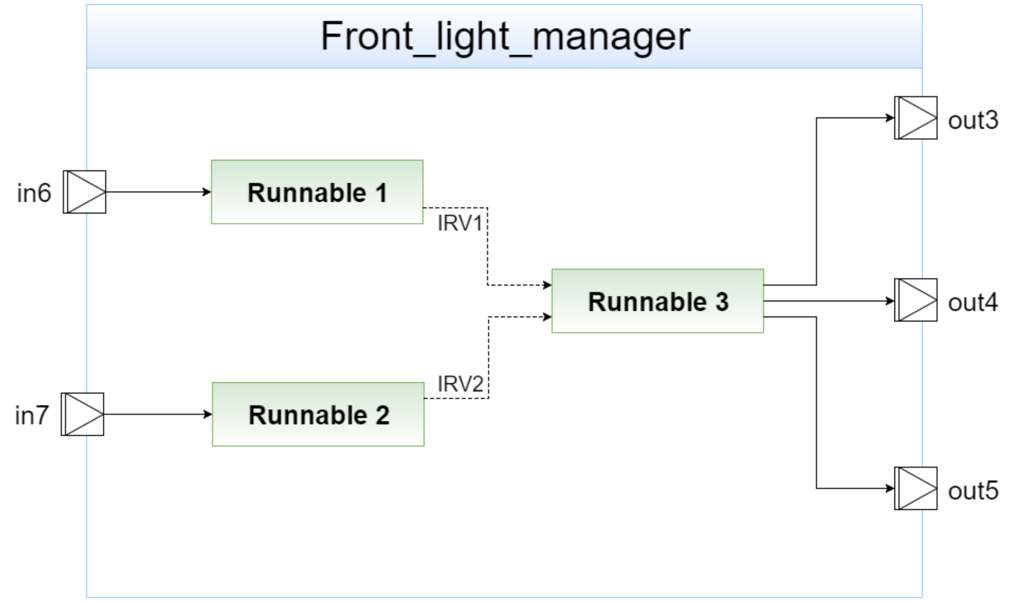
Security annotations: Daytime_running_lights : High FLM_TO_DRL : integr

Data secure flow
is not satisfied

data sent on the link FLM_TO_DRL are not protected along the path from the sources to the destination



An example of component: Front_light_manager



```
void FLM_Runnable3(void) {
    Rte_IWrite_Runnable3_PPort_out3(Rte_IrvIRead_Runnable3_IRV1());
    Rte_IWrite_Runnable3_PPort_out5((Rte_IrvIRead_Runnable3_IRV2());
    if ( (Rte_IrvIRead_Runnable3_IRV1() == REQ_HEADLIGHT_ON) ||
         (Rte_IrvIRead_Runnable3_IRV2() == REQ_DAYTIME_ON) ){
                Rte_IWrite_Runnable3_PPort_out4(LIGHTS_ON);
    }
    else{
          Rte_IWrite_Runnable3_PPort_out4(LIGHTS_OFF);
    }
}
```

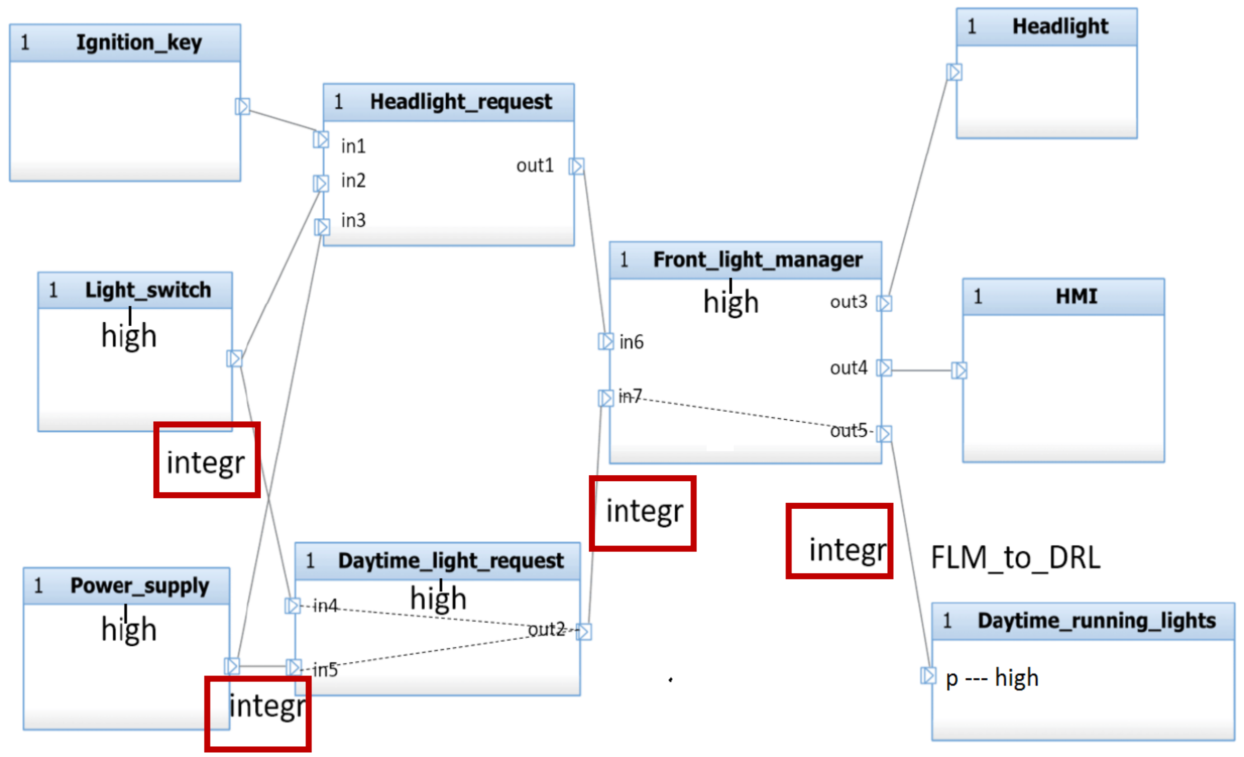
Information for generating the context

```
% global variables
int HR_voltage_threshold1;
int HR_voltage_threshold2;
int DLR_voltage_threshold1;
...
% inter runnable variables
int16_t FLM_IRV1;
int16_t FLM_IRV2;
int16_t DLR_IRV1;
...
```

```
% ports
int in1;
int in2;
...
int out1;
int out2;
...
% functions
void flm_Runnable1() 0;
void flm_Runnable2() 0;
.....
% links
out2 -> in7;
out1 -> in6;
```

Annotated model

the output port of Front light manager connected to the Daytime_running_lights (out5 in our implementation) does not depend on the input port connected to the Headlight request component (in6 in our implementation)



Model satisfies Secure flow property