# TrAF-Cloud Use Case

*Use Case: Exterior light via the cloud*

**Project** TrAF-Cloud within Vinnova/FFI/EMK (Dnr 2018-05010)

**Responsible author** Thomas Söderqvist, VGTT

**Document status** FINAL

**Document date** 2019-10-04

## Contents

## Version History

| Version | Date | Paragraphs affected | Change information |
|---|---|---|---|
| 0.1 | 2019-09-11 | All | Initial version |
| 0.2 | 2019-09-16 | Introduction | Added further details about the use case |
| 0.4 | 2019-09-27 | Plans, Sequence diagrams | |
| 1.0 | 2019-10-01 | | Final iteration 1 |
| 1.1 | 2019-10-04 | | Minor correction |

## Authors

| Name | Partner | Chapters |
|---|---|---|
| Thomas Söderqvist | VGTT | Introduction, goals and objectives |
| Mikael Thorman | VGCS | Section Cloud Technology |
| Niklas Mellegård | RISE | Section Proof of concept |
| Milin Hari Hara Krishnan | VGTT | Section Sequence diagrams |

## Terminology

| Term | Description |
|---|---|
| | |
| | |

## Introduction

The initial use case selected is the ‘Exterior Light via the cloud’. The main purpose with this use case is to establish the basic architectural components for integrating on- and off-board functionality to allow more investigation of more complex use cases in the continuation of the project.

Based on the service-oriented design principles of Adaptive AUTOSAR, the original Exterior Light has two service consumers, one via the HMI and one via sensors that detect the light condition and the light is switch on or off, which the light condition that currently exists requires. The use case Exterior Light via the cloud extends this by adding a consumer in the cloud, with arbitration of which consumer has precedence is done on-board. The provider of the service is in the truck. The UML diagram of the Exterior Light is shown in section 4.

## Goal and objectives of the project

The TrAF-Cloud project is based on the following questions:

1. How can we design a secure and safe common truck on-board and off-board architecture that enables seamless allocation and execution of functionality in the cloud?

2. How can we securely and safely allocate functions to execute either in the cloud or in the truck?

3. How can we ensure transparency during the execution of functions?

4. What are the architectural dependencies between on-board and off board architecture?

5. What principles can be used to leverage cloud-enabled truck architectures into open service platforms?

By addressing these questions, the project aims at achieving the following goals:

G1. The project will create and demonstrate a reference platform that integrates on-board and off-board functionality.

G2. The utilisation of in-vehicle computation power will decrease for a certain class of applications (to be identified in the project) while maintaining adequate responsiveness with unchanged or increased quality of service.

G3. The integration lead times will decrease with several orders of magnitude for certain functionality.

G4. The project will demonstrate and evaluate the viability of a business-oriented ecosystem for transport solution services.

## Activities planned for the use case

The initial use case activity is part of the finding the solution for goal G1. How will the truck architecture be interconnected with the cloud technology and its components?

This use case have one service consumer in the cloud and two consumers in the truck and a service provider in the truck. In the extent of the project the use case will be extended to address all goals.

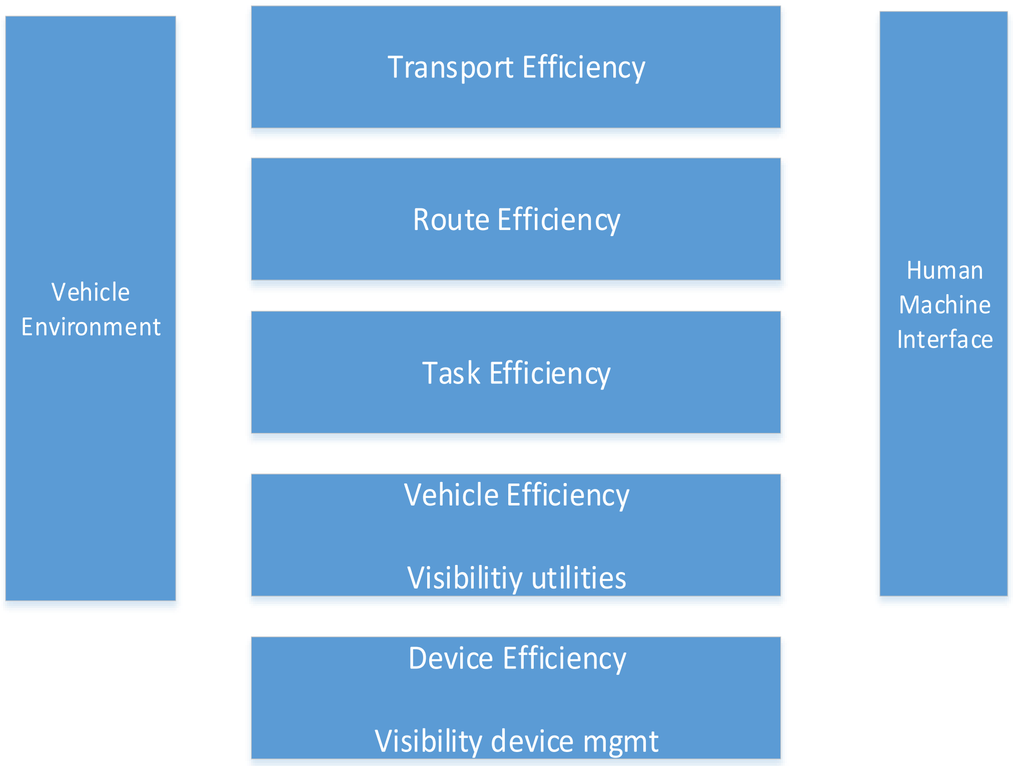
In addition there are various activities to be done to meet goal G1, see table below. The activities G2 – G4 will be described within the second iteration of this use case deliverable.

| | **Activities** | **Project goals** | **UC doc iteration** |
|---|---|---|---|
| 1 | Establishment of the first version basic architecture for integrating on- and off-board functionality including implementation and certification. This is the first use case step with a service consumer added to the cloud and the service provider in the truck. Security solutions and authorization rules set up for the first version basic architect. This includes verification. See section Proposed Proof of Concept design. | G1 | 1 |
| 2 | Establishment of the second version basic architecture with a service provider in the cloud and service consumer in the truck. In this case we need possible to change basic use case. Security solutions and authorization rules for the second version basic architecture. This includes verification. Verification methods must be modified compared to step 1 to fit this step 2. | G1 | 2 |
| 3 | Investigation of the timing characteristics in the two first steps. | G1 | 2 |
| 4 | Splitting of the TAS-FURA layers (see chapter TAS-FURA) and put some of the layers in the cloud? | G1 | 2 |
| | | | |

## TAS-FURA

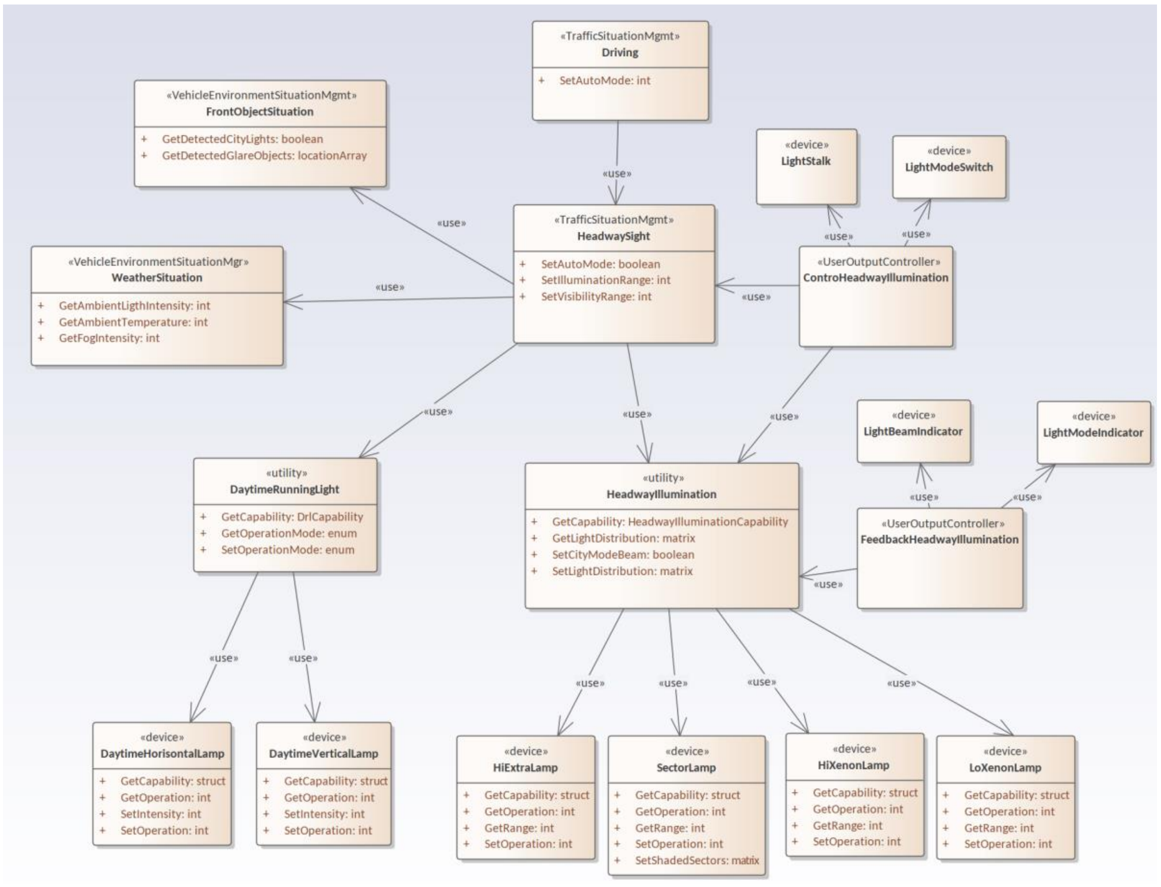
The application SW in the truck follows the TAS-FURA (Truck Application System – Functional Reference Architecture) principles. The figure below show the very simplified layers in TAS-FURA. In the two layers Device and Vehicle there are two entities specifically dedicated for the Exterior Light realization; Visibility Utilities and Visibility device management respectively. The scientific paper 'Rethink EE Architecture in Automotive to facilitate Automation, Connectivity, and Electro Mobility' [1] shows some more detail about TAS-FURA.

As a part of the project will plan to split the TAS-FURA. Which layers is possible to locate in the cloud.



## UML diagram of Exterior Light

The figure below is an overview of the UML models for this initial use case with the TAS-FURA to interface with the cloud.


«TrafficSituationMgmt» Driving
+ SetAutoMode: int
«VehicleEnvironmentSituationMgmt» FrontObjectSituation
+ GetDetectedCityLights: boolean
+ GetDetectedGlareObjects: locationArray
«device» LightStalk
«device» LightModeSwitch
«TrafficSituationMgmt» HeadwaySight
+ SetAutoMode: boolean
+ SetIlluminationRange: int
+ SetVisibilityRange: int
«VehicleEnvironmentSituationMgr» WeatherSituation
+ GetAmbientLigthIntensity: int
+ GetAmbientTemperature: int
+ GetFogIntensity: int
«UserOutputController» ControHeadwayIllumination
«device» LightBeamIndicator
«device» LightModeIndicator
«utility» DaytimeRunningLight
+ GetCapability: DrlCapability
+ GetOperationMode: enum
+ SetOperationMode: enum
«utility» HeadwayIllumination
+ GetCapability: HeadwayIlluminationCapability
+ GetLightDistribution: matrix
+ SetCityModeBeam: boolean
+ SetLightDistribution: matrix
«UserOutputController» FeedbackHeadwayIllumination
«use»
«device» DaytimeHorisontalLamp
+ GetCapability: struct
+ GetOperation: int
+ SetIntensity: int
+ SetOperation: int
«device» DaytimeVerticalLamp
+ GetCapability: struct
+ GetOperation: int
+ SetIntensity: int
+ SetOperation: int
«device» HiExtraLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int
«device» SectorLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int
+ SetShadedSectors: matrix
«device» HiXenonLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int
«device» LoXenonLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int

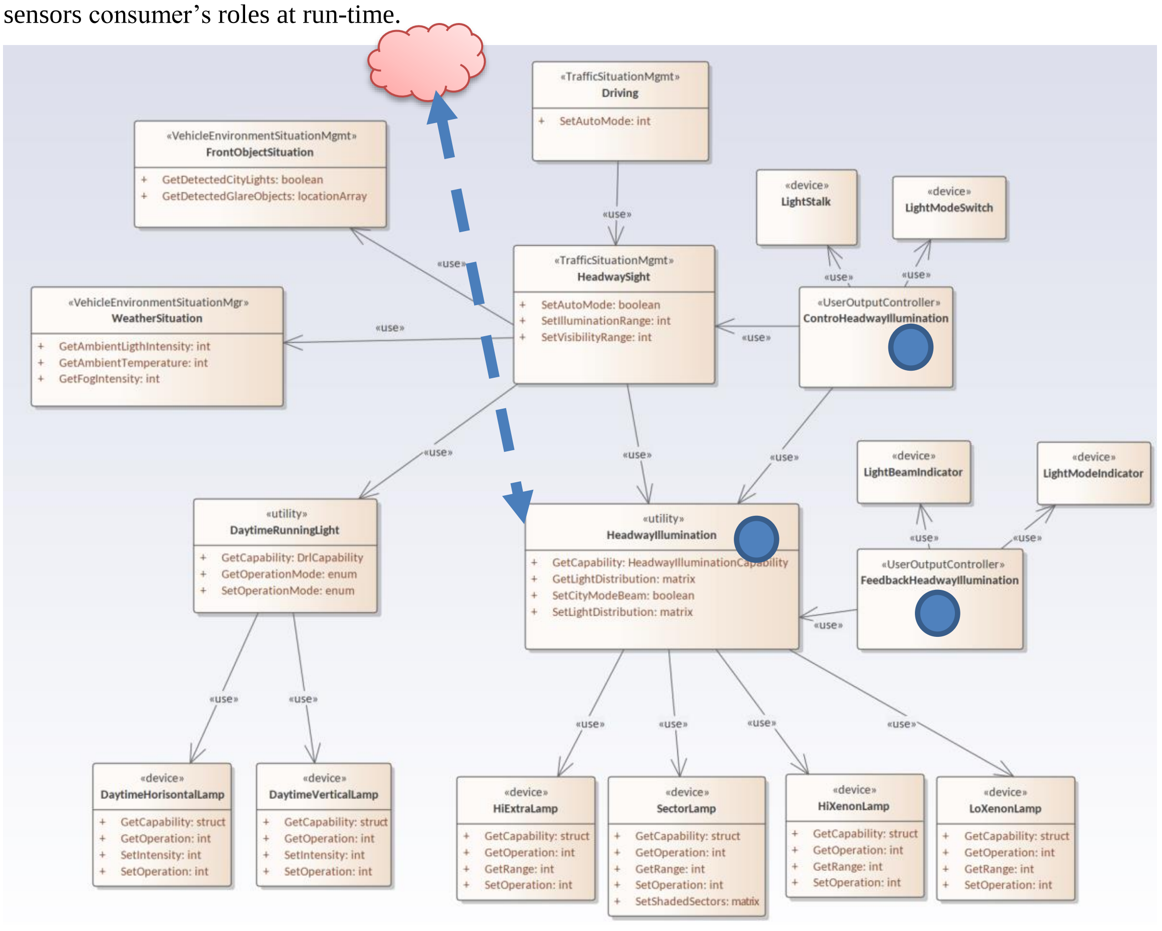
## Cyber security

The upcoming standard ISO21434 ‘Cyber Security for Vehicles’ is under development. We need to take consideration in the work with this use case. AB Volvo takes part of development of the standard. Combitech is the SiS Swedish working group for the standard.

## Description

### UML diagram

The overview diagram for the Exterior Light is shown below with the extension with a consumer in the cloud. The HeadwayIllumination element is (likely) where the decision is taken about the HMI and the sensors consumer's roles at run-time.


«TrafficSituationMgmt» Driving
+ SetAutoMode: int
«VehicleEnvironmentSituationMgmt» FrontObjectSituation
+ GetDetectedCityLights: boolean
+ GetDetectedGlareObjects: locationArray
«device» LightStalk
«device» LightModeSwitch
«TrafficSituationMgmt» HeadwaySight
+ SetAutoMode: boolean
+ SetIlluminationRange: int
+ SetVisibilityRange: int
«VehicleEnvironmentSituationMgr» WeatherSituation
+ GetAmbientLigthIntensity: int
+ GetAmbientTemperature: int
+ GetFogIntensity: int
«UserOutputController» ControHeadwayIllumination
«device» LightBeamIndicator
«device» LightModeIndicator
«utility» DaytimeRunningLight
+ GetCapability: DrlCapability
+ GetOperationMode: enum
+ SetOperationMode: enum
«utility» HeadwayIllumination
+ GetCapability: HeadwayIlluminationCapability
+ GetLightDistribution: matrix
+ SetCityModeBeam: boolean
+ SetLightDistribution: matrix
«UserOutputController» FeedbackHeadwayIllumination
«device» DaytimeHorisontalLamp
+ GetCapability: struct
+ GetOperation: int
+ SetIntensity: int
+ SetOperation: int
«device» DaytimeVerticalLamp
+ GetCapability: struct
+ GetOperation: int
+ SetIntensity: int
+ SetOperation: int
«device» HiExtraLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int
«device» SectorLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int
+ SetShadedSectors: matrix
«device» HiXenonLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int
«device» LoXenonLamp
+ GetCapability: struct
+ GetOperation: int
+ GetRange: int
+ SetOperation: int
«use»

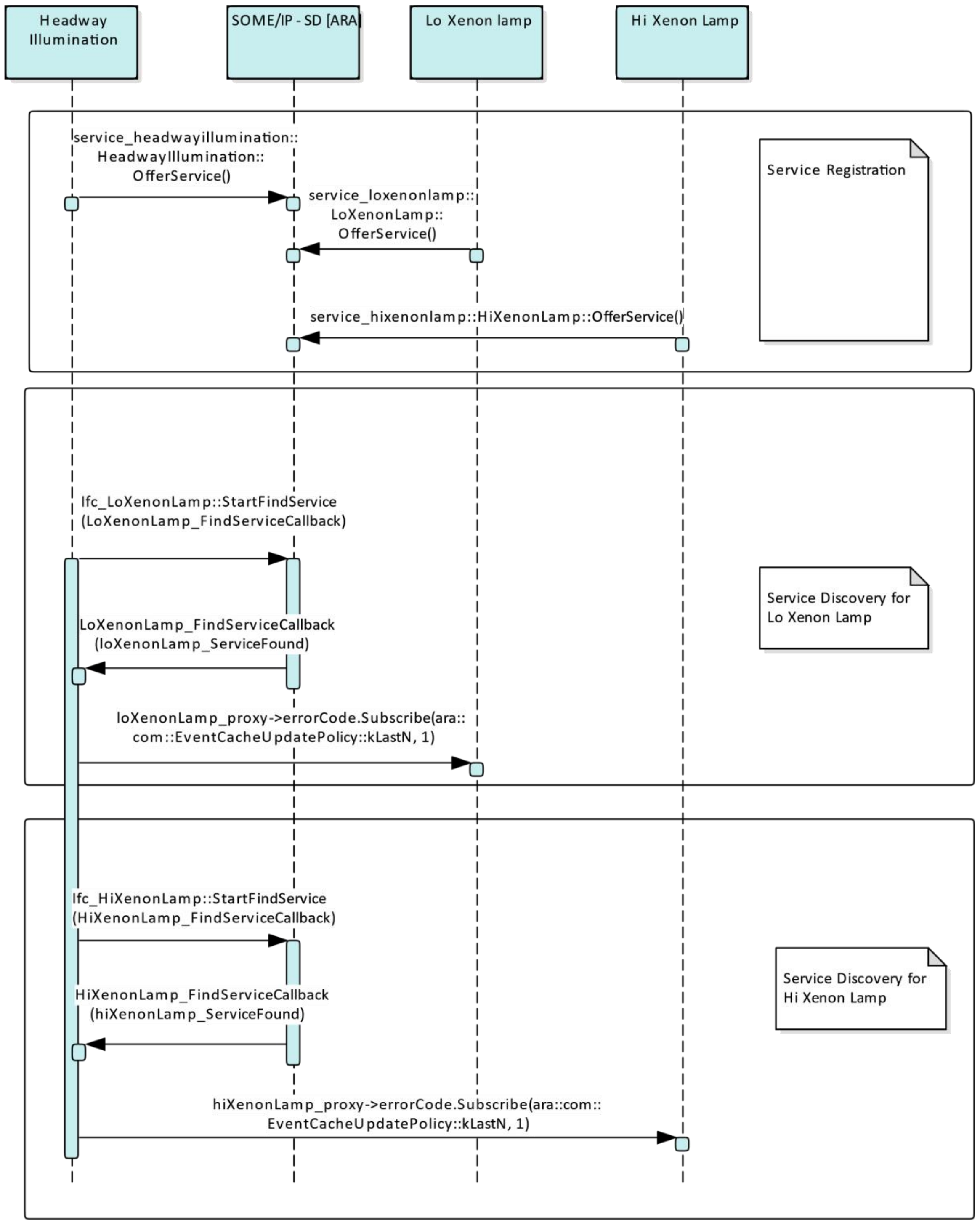
In this use case a consumer is added to the cloud and is connected to the HeadwayIllumination element which now is taken decision about the roles that the HMI, the sensor and the cloud consumers shall have in each specific situation. In some cases even if the cloud consumer try to switch off the light it is not suitable based on what the sensor decides.

The model elements marked with blue rings have some implementations but in this use case we will complement some of the other elements.

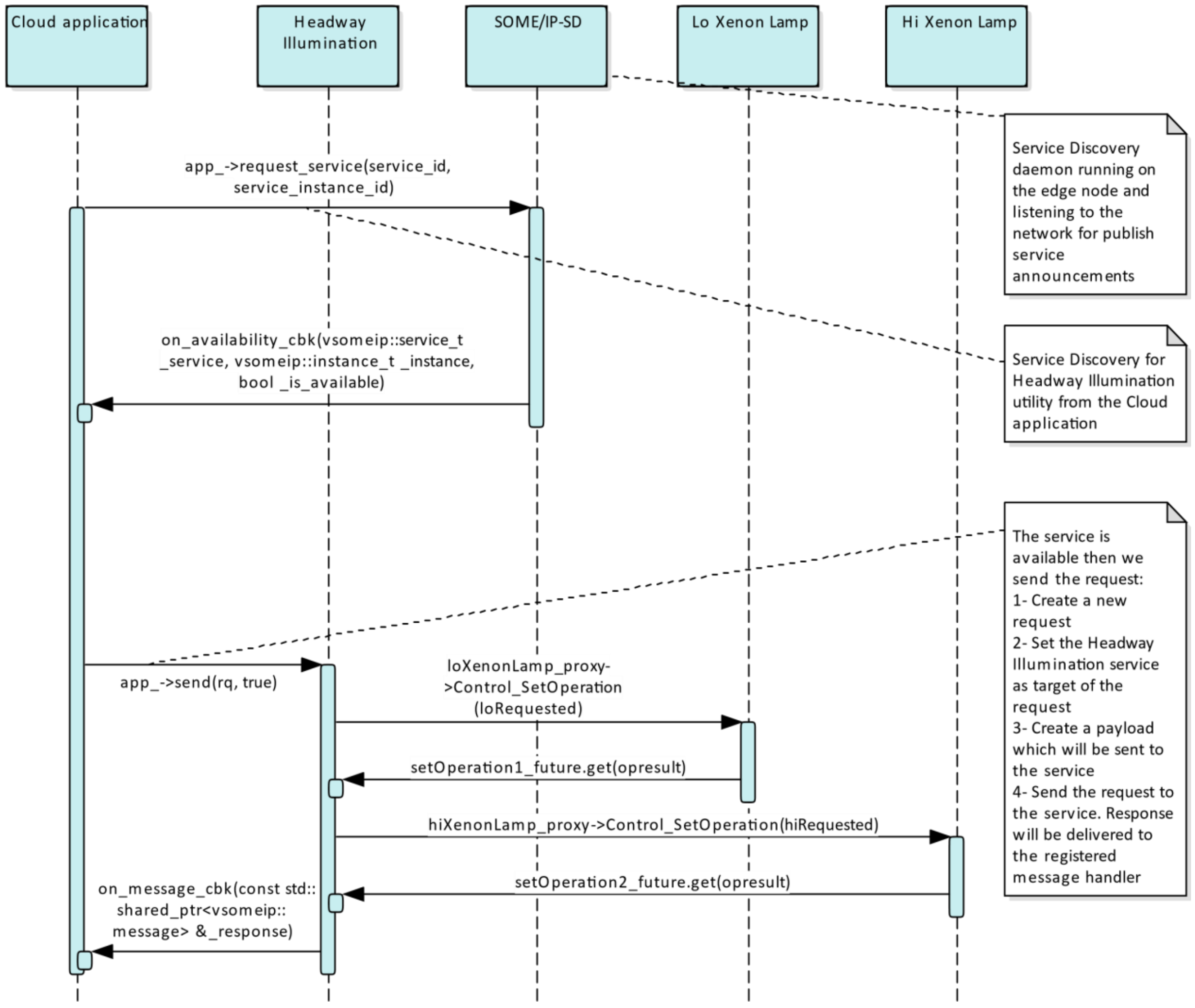
## Sequence diagrams from the truck point of view

Below are sequence diagrams seen from the truck and in the section Cloud technology there are sequence diagrams seen from the cloud point of view.

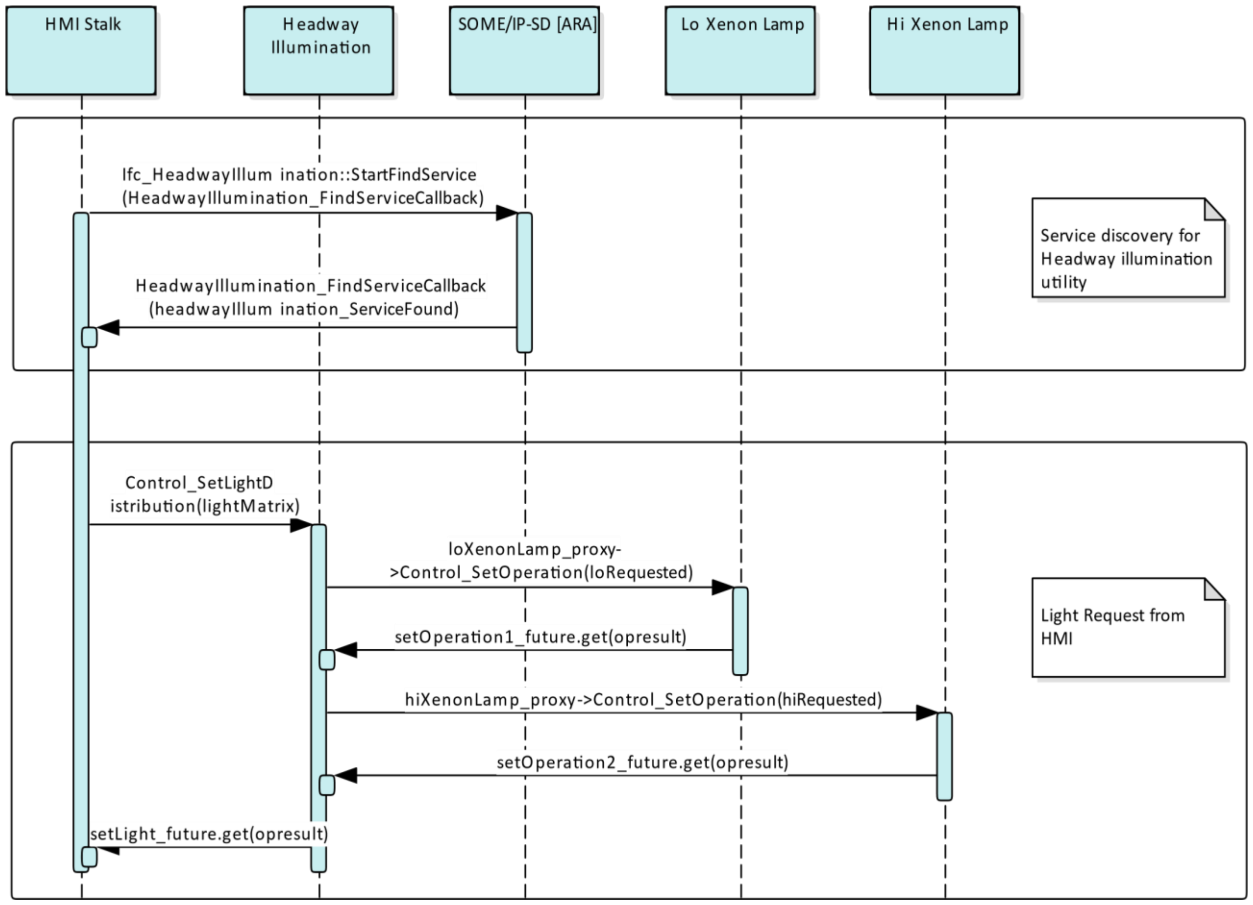
1. Service Registration of the HeadwayIllumination and Service Discovery for the two lamps Lo and Hi Xenon


Headway Illumination
SOME/IP - SD [ARA]
Lo Xenon lamp
Hi Xenon Lamp
service_headwayillumination:: HeadwayIllumination:: OfferService()
service_loxenonlamp:: LoXenonLamp:: OfferService()
service_hixenonlamp::HiXenonLamp::OfferService()
Service Registration
Ifc_LoXenonLamp::StartFindService (LoXenonLamp_FindServiceCallback)
LoXenonLamp_FindServiceCallback (loXenonLamp_ServiceFound)
loXenonLamp_proxy->errorCode.Subscribe(ara:: com::EventCacheUpdatePolicy::kLastN, 1)
Service Discovery for Lo Xenon Lamp
Ifc_HiXenonLamp::StartFindService (HiXenonLamp_FindServiceCallback)
HiXenonLamp_FindServiceCallback (hiXenonLamp_ServiceFound)
hiXenonLamp_proxy->errorCode.Subscribe(ara::com:: EventCacheUpdatePolicy::kLastN, 1)
Service Discovery for Hi Xenon Lamp

2. Service Discovery from the cloud and light request from the cloud


Cloud application
Headway Illumination
SOME/IP-SD
Lo Xenon Lamp
Hi Xenon Lamp
Service Discovery daemon running on the edge node and listening to the network for publish service announcements
app_->request_service(service_id, service_instance_id)
Service Discovery for Headway Illumination utility from the Cloud application
on_availability_cbk(vsomeip::service_t _service, vsomeip::instance_t _instance, bool _is_available)
The service is available then we send the request:
1- Create a new request
2- Set the Headway Illumination service as target of the request
3- Create a payload which will be sent to the service
4- Send the request to the service. Response will be delivered to the registered message handler
app_->send(rq, true)
loXenonLamp_proxy->Control_SetOperation (loRequested)
setOperation1_future.get(opresult)
hiXenonLamp_proxy->Control_SetOperation(hiRequested)
setOperation2_future.get(opresult)
on_message_cbk(const std::shared_ptr<vsomeip::message> &_response)

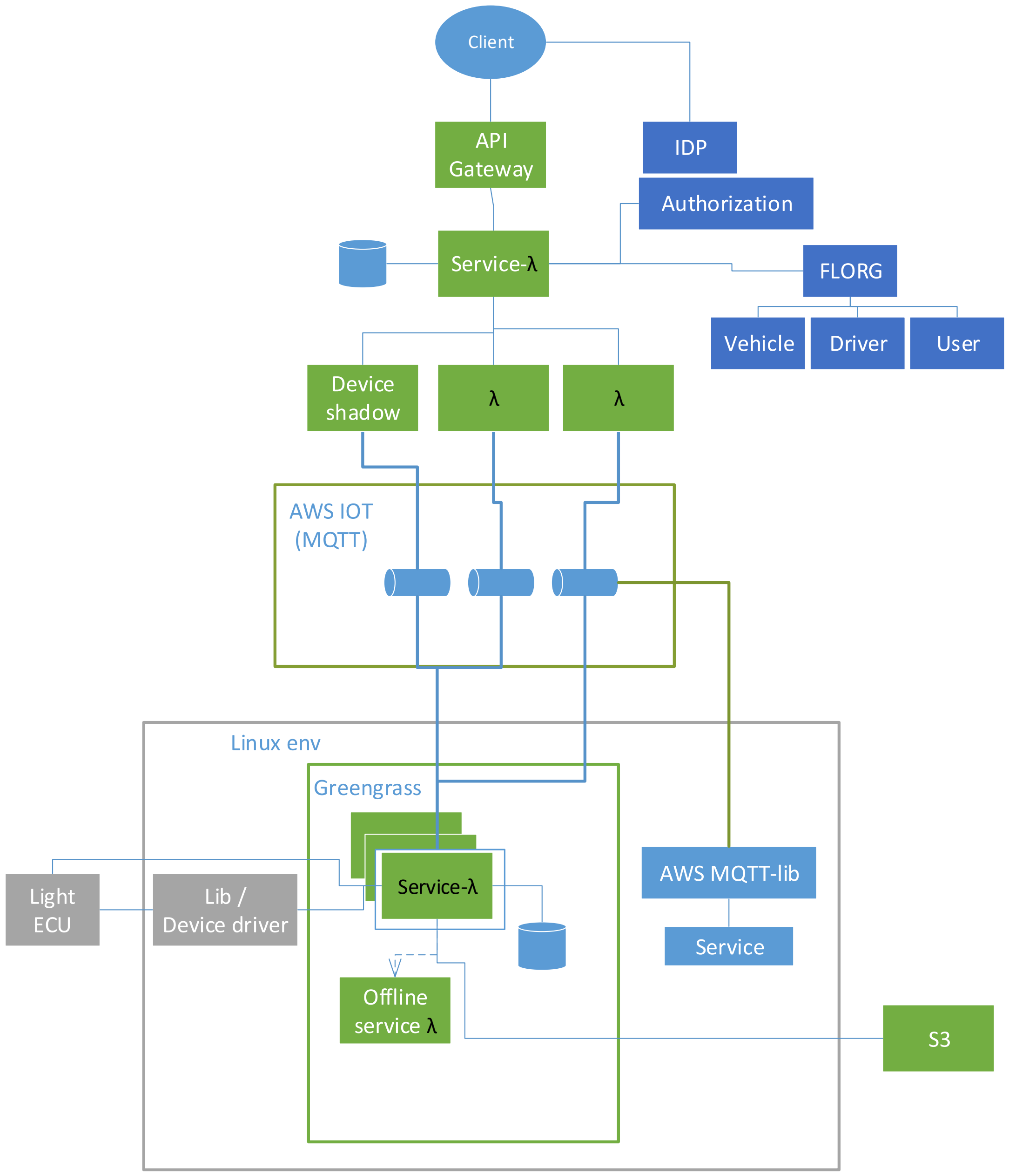
3. Service Discovery from the HMI and light request from the HMI


HMI Stalk
Headway Illumination
SOME/IP-SD [ARA]
Lo Xenon Lamp
Hi Xenon Lamp
Ifc_HeadwayIllum ination::StartFindService (HeadwayIllumination_FindServiceCallback)
HeadwayIllumination_FindServiceCallback (headwayIllum ination_ServiceFound)
Service discovery for Headway illumination utility
Control_SetLightD istribution(lightMatrix)
loXenonLamp_proxy->Control_SetOperation(loRequested)
setOperation1_future.get(opresult)
hiXenonLamp_proxy->Control_SetOperation(hiRequested)
setOperation2_future.get(opresult)
setLight_future.get(opresult)
Light Request from HMI

## Cloud technology

### Overview

This is a high level architectural image of how an AWS IOT based telematics solution could look.



### Components in the architecture

#### *Client*

This is a UI client, for example a web app, Android app, iOS app, voice app or similar.

This is responsible for rendering the user interface and populate it with data from the backend and enabling the user to modify the vehicle by calling the backend actuators.

The backend services are usually called using a REST/GraphQL API.

*IDP*

The Identity provider is responsible for managing the authentication.

AWS has its own called Cognito, but we could also use the Volvo IDP with federation.

*API Gateway*

AWS Module mapping a specific URL to a service call.

*Service lambda*

The actual service that is called from the client.

This will:

- Gather data from a database, other service or device shadow and return to client.
- Send actuator request to the vehicle for state changes or actions in the vehicle.
- Ensures that only authorized users can access services on vehicles it is authorized to use.
- Ensures that the client only can access data it is allowed to access.

*Authorization*

This component is responsible for knowing what assets a user is allowed to use and what services the user can access.

*FLORG*

A fleet organization is a VGCS specific module responsible for managing the belonging of vehicles, drivers & users to a specific fleet and the hierarchy of fleets.

It ensures that only the correct users can access its vehicles, drivers and data/services around these

*Communication service layer*

This layer abstracts away how the actual communication is done with the vehicle and what protocol that is being used.

Two examples on how this can be implemented are:

- Lambda service with (optionally) its own database
- Device shadow (not suitable for all use cases)

*AWS IOT / MQTT layer*

The actual MQTT transport layer, here the AWS variant implementing most MQTT interfaces (not level 2 though).

*Greengrass lambda service*

The lambda implementing the actual service in the vehicle.

This can be as simple as just forwarding the request/data to the correct MQTT topic in the correct transport protocol or more complex also implementing limited offline support of the service potentially with the support of other offline lambda services.

The security on which topics the lambda can access and what OS resources it can access is controlled in the AWS lambda environment enabling strong security.

The lambda can potentially also use local resources, such as databases for caching…

Software download of Lambdas is supported out of the box from AWS.

### *Non-Greengrass service*

The vehicle service doesn't have to be implemented using Greengrass, but can be implemented in any way just accessing the backoffice using MQTT.

### *Device driver / library*

Lambdas needs external drivers to be able to access external resources as for example CAN busses…

Depending on how we communicate with external services we might need to implement specific device drivers for these.

### *External ECU*

ECU's outside the Greengrass environments HW that we need to access/use services from.

## MQTT

MQTT have a very dynamic topic naming structure, we need to define how we want this to be designed.

### *Topic naming structure*

Below is an example of how we could design the topic naming structure.

***/payload-format/transport-ack/telematic-protocol/vehicle-id/service/protocol-version***

### *Example*

An example of a topic name based on the naming conventions above.

***/json/true/swap/VIN123456/light-activation/25***

### *Topic payload*

The content of each topic also needs to be defined.

Most commonly used format is JSON, Protobuf is also being implemented by AWS but any format could be used as long as both sides agree on it.

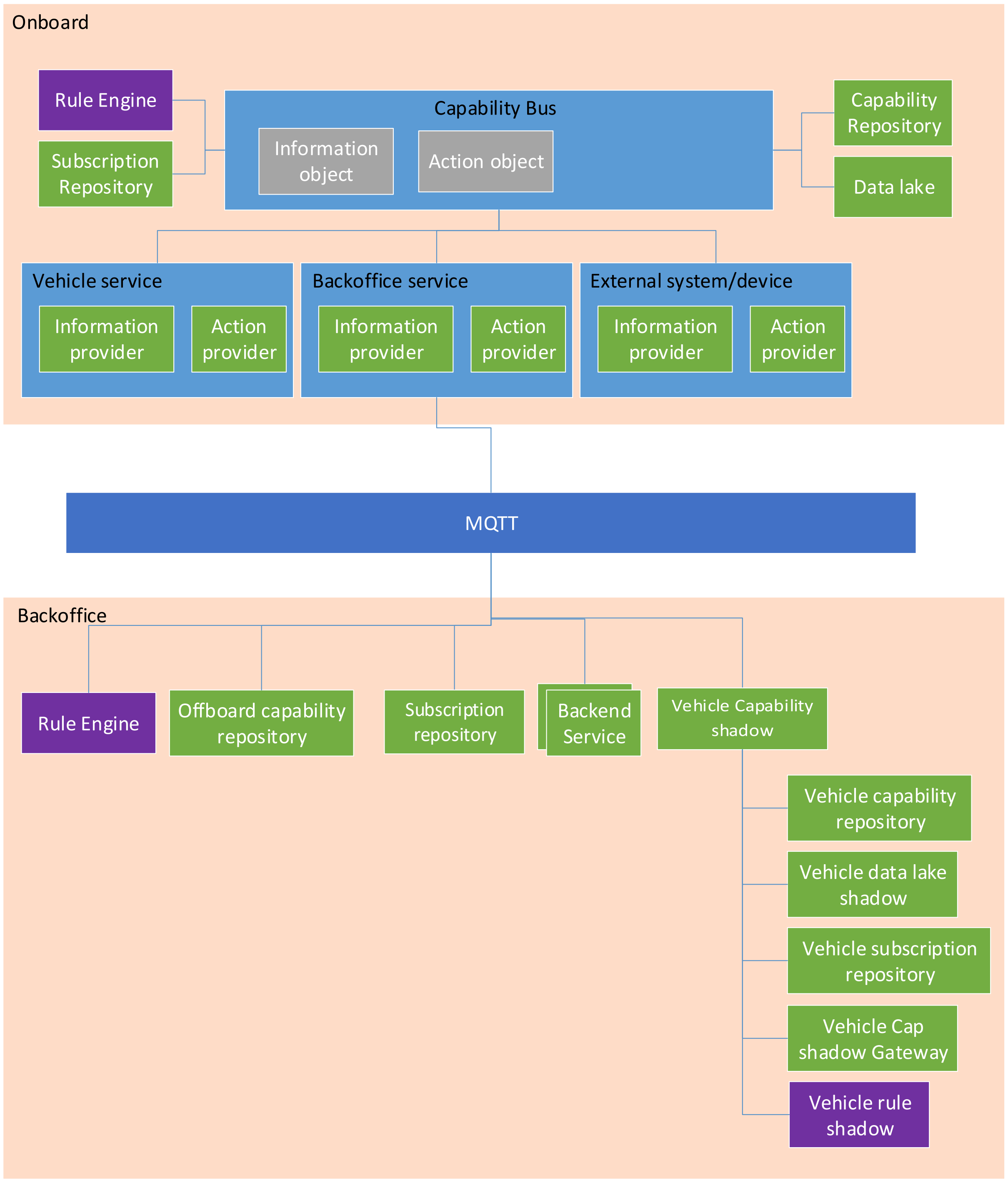
Device shadows currently only support JSON with protobuf upcoming according to AWS.

## Capability bus concept

### Purpose

- Enabling a service to access data or capabilities from another service regardless of where that service/data is located (onboard/offboard/external device)
- Enabling a service to know what capabilities that are available to it
- Enabling a service to subscribe to data regardless of source
- Make functionality configurable
  - Shorter TTM based on existing capabilities/functionality in telematics platform
  - Easier to identify/share common ”platform” components
- Enabling an end user to create specific functionality and behavior based on existing capabilites.
- Enabling a 3rd party systems and functions to utililze the vehicle capabilities to create new functions and services

## Overview



### *Vehicle services*

This chapter describes the services in the vehicle

### *Subscription repository*

A service can use this to subscribe to a data item and also define the criteria for when it wants this data.

### *Capability repository*

This is a repository with all Information providers and action providers that is connected to the capability bus.

These are the capabilities that can be used when creating new rules

### *Data lake*

This is a cache for data received from connected nodes.

The cache can subscribe to data items in the subscription repo as any other service, these data items are then stored for usage by other services.

It also exposes an action where any other service can put data into it.

### *Capability bus*

This is a communication layer through which all devices can talk to each other.

### *Connected services*

Each of the following connected services can do the following:

- Expose data items to the capability bus
- Expose actions to the capability bus
- Subscribe to data items from the capability bus
- Execute actions on the capability bus
- Create virtual data items based on calculations based on other data on the capability bus

#### ***Vehicle service***
A vehicle service can be on any ECU in the vehicle.

#### ***Backoffice service***
A gateway to/from offboard services (in the cloud), it:

- Enriches the data with information needed in offboard communication.
- Does potential protocol transformation
- Does potential transport technology mapping

#### ***External system/device***
This is a service executing on an external device (mobile phone, body builder unit…)

#### ***Rule engine***
A module running all the rules in the rule repository on the received events & data in the data cache.

This could be used to accomplish simpler tasks, connect services by configuration (not code) and generate new data items if only simple operations are needed.

This would enable us to handle our capabilities similarly to the “IFTTT”-concept.

This is an optional module.

## *Offboard services*

This chapter describes the services in the offboard system

***Vehicle capability shadow***
This is a service that manages the capabilities for each vehicle.

***Vehicle capability repository***
This is a shadow-repository with all Information providers and action providers for each vehicle

This is used when creating new rules for a specific vehicle, to know what is available.

***Vehicle data lake shadow***
A shadow of the onboard “data lake” in the vehicle to know the latest data state of data relevant to the offboard system.

***Vehicle subscription repository***
An offboard shadow of the onboard subscription repository.

***Vehicle Cap shadow gateway***
Exposes an API to manage the Vehicle capability shadow information from external systems.

## *Backend service*

This is a placeholder for all existing backend services that could operate on the vehicle capabilities and expose capabilities to the vehicle.

## *Offboard capability repository*

This is a repository with all offboard information and action providers that is connected to the capability bus.

***Subscription repository***
A service can use this to subscribe to a data item and also define the criteria for when it wants this data.

***Rule engine***
A module running all the rules in the rule repository on the received events & data in the data cache.

This could be used to accomplish simpler tasks, connect services by configuration (not code) and generate new data items if only simple operations are needed.

This would enable us to handle our capabilities similarly to the “IFTTT”-concept.

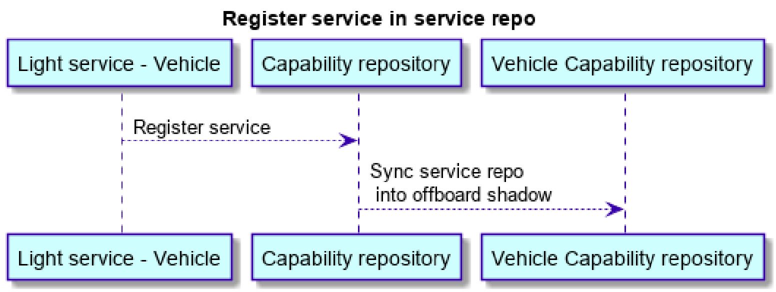
This is an optional module.

***Vehicle rule shadow***
A shadow repository of all rules in a vehicle, configuration of the vehicle is done using this shadow & it is then synced with the vehicle.

## *Example flow – Turn lights on/off*

Here we describe an example flow for the turn on lights use case

First the light service in the vehicle must inform that it has capabilities that can be used


Register service in service repo
Light service - Vehicle
Capability repository
Vehicle Capability repository
Register service
Sync service repo into offboard shadow

This is what it will supply to the Capability repository

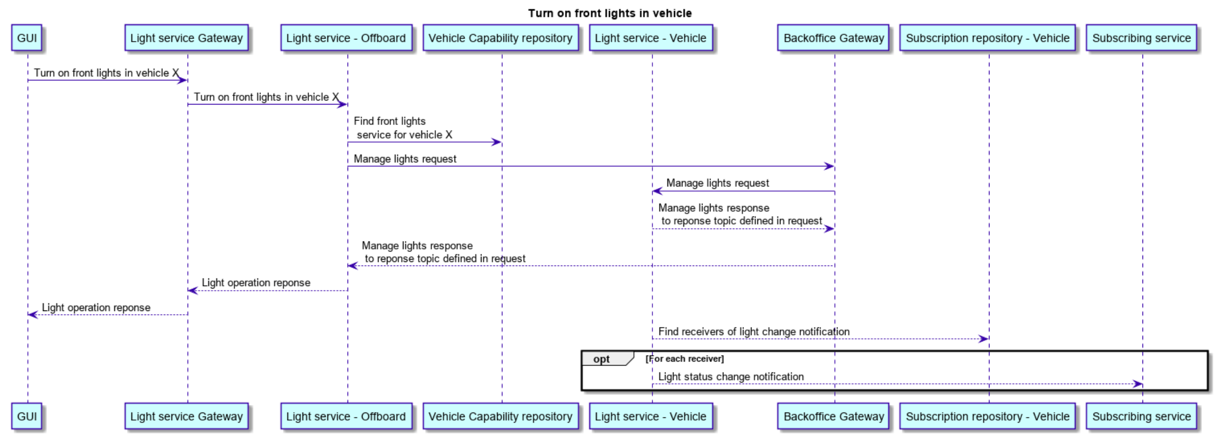
```
Capability {
apiTechnology: MQTT,
topicName: ecu-flc/ManageFrontlightsRequest,
supportedPayloadFormats: json,
minApiSupported: 1.0,
maxApiSupported: 1.1,
OperationType: ManageLights
CapabilityType: Service
}
```

This will later on be synced to the Vehicle Capability repository offboard which then will contain this info:

This is what it will supply to the Capability service

```
Capability {
apiTechnology: MQTT,
topicName: VIN1234567/ecu-flc/ManageFrontlightsRequest,
supportedPayloadFormats: json,
minApiSupported: 1.0,
maxApiSupported: 1.1,
OperationType: ManageLights
CapabilityType: Service,
VehicleId: VIN1234567
}
```

This data is then used when posting a request to the vehicle to start lights


Turn on front lights in vehicle
GUI
Light service Gateway
Light service - Offboard
Vehicle Capability repository
Light service - Vehicle
Backoffice Gateway
Subscription repository - Vehicle
Subscribing service
Turn on front lights in vehicle X
Turn on front lights in vehicle X
Find front lights service for vehicle X
Manage lights request
Manage lights request
Manage lights response to reponse topic defined in request
Manage lights response to reponse topic defined in request
Light operation reponse
Light operation reponse
Find receivers of light change notification
opt
[For each receiver]
Light status change notification

The following is sent:

```
message {
header: {
   sender: LightServiceOffboard,
    correlationId: SDFGHJK32131FGH,
    sendTime: 1234567890,
    replyTopic: /backoffice/LightService/ManageLightsResponse
},
Payload: {
    ManageLightRequest: {
        onOffOperation: on,
        requestTime: 8765435678
    }
}
}
```

And the response is:

```
message {
header: {
   sender: LightServiceVehicle,
    correlationId: SDFGHJK32131FGH,
    sendTime: 12345678932
},
Payload: {
---     ManageLightResponse: {
        requestStatus: success,
        executedTime: 8765435679
      }
    }
    }
```

This is what the “Subscribing service” will supply to the Subscription repository

```
Capability {
apiTechnology: MQTT,
topicName: ecu-oth/LightsEvent,
supportedPayloadFormats: json,
minApiSupported: 1.0,
maxApiSupported: 1.1,
subscribeToEvent: LightStatusChangeNotifications,
SubscribeFrom: FrontLightsEcu, ReaLightsEcu,
Subscriber: Subscribing service,
Criteria: []
}
```

### *Example rule*

Just to exemplify what a rule could be used for:

```
LightsOnToLongWarningRule {
IF (dw.vehicle.frontLights.status.on && currentTime – dw.vehicle.frontLights.status.on.executedTime > 1h && currentTime – dw.vehicle.lastmovedTime > 30 min) THEN sendEmailToDriver(“You frontlights have been on for “ + (currentTime – dw.vehicle.frontLights.status.on.executedTime)/60000 + “ minutes”)
}
```

That is a rule that uses data available in the warehouse and potential triggers to execute another service using data in the trigger and/or from the data warehouse.

What it does is that it sends a warning email to the driver if the frontlights have been on too long on a vehicle that hasn’t moved.

## Topic naming structure

Below is an example of how we could design the topic naming structure.

***/payload-format/transport-ack/telematic-protocol/vehicle-id/service/protocol-version***

### *Example*

An example of a topic name based on the naming conventions above.

***/json/true/swap/VIN123456/light-activation/25***

## Topic payload

The content of each topic also needs to be defined.

Most commonly used format is JSON, Protobuf is also being implemented by AWS but any format could be used as long as both sides agree on it.

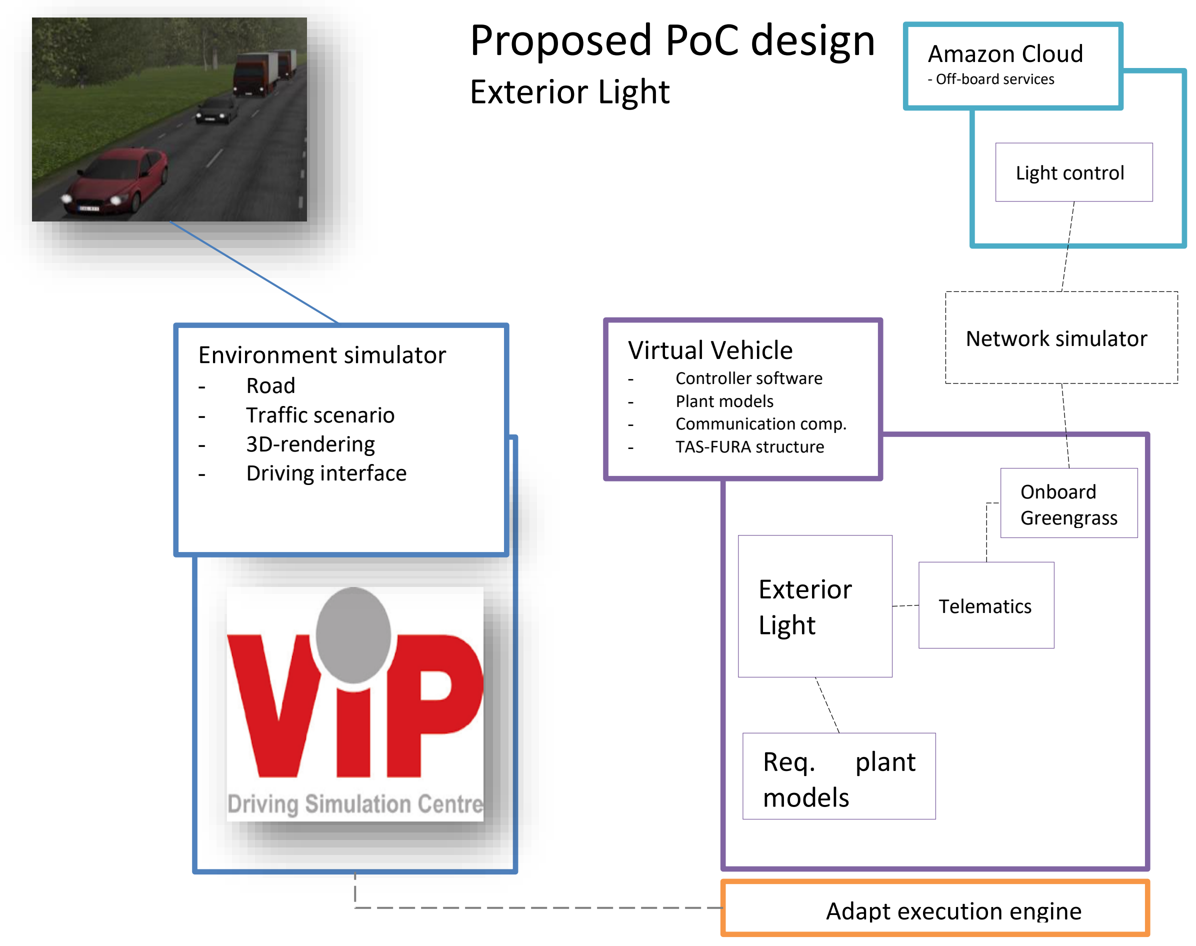
Device shadows currently only support JSON with protobuf upcoming according to AWS.

## Proposed Proof of Concept design

- Plan of action
  1. Set up basic models for the virtual vehicle
     - Existing code from ABV
     - Structure according to TAS-FURA (i.e. with a proper function architecture)
     - Document architectural models
       - For further design, as well as safety and security analyses
  2. Integrate the on-board GreenGrass component
  3. Implement a simple cloud-function to control lights
  4. Connect through a network simulator
  5. Do PoC security analyses
- Install PoC in VICTA Open Innovation Lab (at Lindholmen)
  1. More accessible than Lundby
  2. Prepare to use as basis for innovation bazaar etc

Scenarios

1. Vehicle announces availability of lighting toggle service
   a. Success criteria: cloud capability repo should show availability of service for the specific vehicle (identified with the VIN)
2. Vehicle removes the service
   a. Success criteria: cloud capability repo should **not** show availability of service for the specific vehicle (identified with the VIN)
3. Cloud application successfully turns on light on vehicle
   a. Success criteria: the light on the vehicle should somehow show that it has been turned on
4. Cloud application fails to turns on light on vehicle (e.g. vehicle shall disallow due to an overriding rule)
   a. Success criteria: Cloud application receives an error message detailing the reason for failing
5. Lights are toggled onboard,
   a. Success criteria: the cloud device shadow/digital twin shall reflect the vehicle state


Proposed PoC design
Exterior Light
Amazon Cloud
- Off-board services
Light control
Network simulator
Environment simulator
- Road
- Traffic scenario
- 3D-rendering
- Driving interface
Virtual Vehicle
- Controller software
- Plant models
- Communication comp.
- TAS-FURA structure
Onboard Greengrass
Exterior Light
Telematics
Req. plant models
ViP
Driving Simulation Centre
Adapt execution engine

## References

[1] A. Magnusson, L. Laine, and J. Lindberg, “Rethink EE Architecture in Automotive to Facilitate Automation, Connectivity, and Electro Mobility,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, New York, NY, USA, 2018, pp. 65–74.