IOT.BZH

Introduction to application framework

for AGL

Version 1.0

June 2016

Abstract

This document presents the application framework created by IoT.bzh for AGL.

Document revisions

Date	Version	Designation	Author
19 June 2016	0.1	Initial release	José Bollo

Table of contents

Legals

1. Reason of the application framework

1.1. Openness of the platform

The promise of the application framework is to allow actors to push new solutions quickly in a safe environment. The ultimate stage would make the car a platform of interest for tiers while remaining sure and under control of manufacturers.

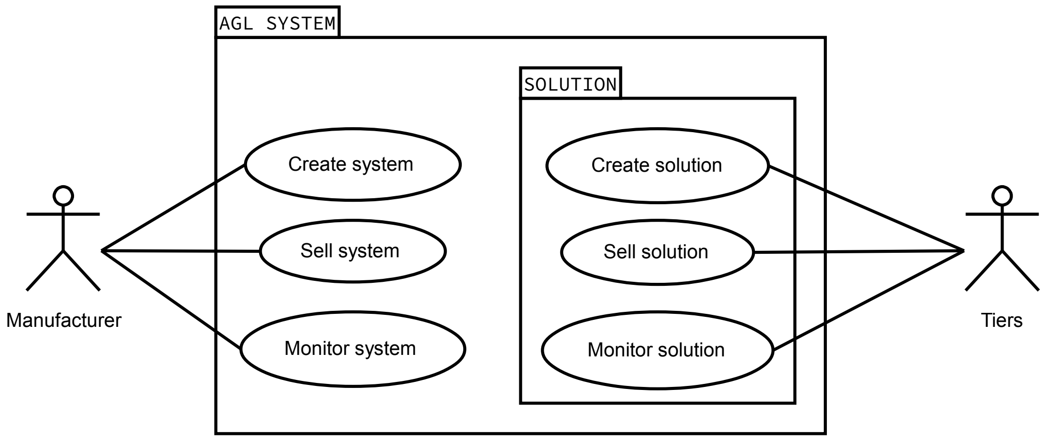

Illustration 1: openness for tiers

To reach those goals the application framework brings true openness:

- under strict security enforcement
- keeping capability for vendors to provide a fully customized product

1.2. Credentials management

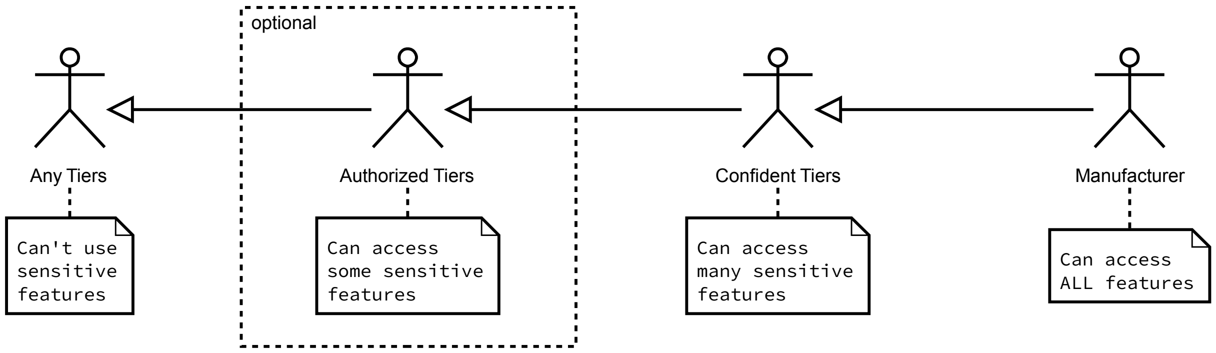
optional
Any Tiers
Authorized Tiers
Confident Tiers
Manufacturer
Can't use sensitive features
Can access some sensitive features
Can access many sensitive features
Can access ALL features

Illustration 2: credentials of tiers

Nevertheless, openness isn't the same for all tiers. Some tiers are involved to work with manufacturers on sensitive areas of the system. This leads to 2 categories, at least, of tiers depending on their credentials: any tiers and confident tiers.

1.3. Management of permissions

Credentials, permissions, privileges, capabilities, restrictions of use, granted accesses, … terminology are often equivalent to talk about system protection. Protection of a system has multiple aspects: protecting integrity, protecting privacy, ensuring safety, protecting whole services, protecting part of services, handling contextual permissions (car runs, doors opened, …).

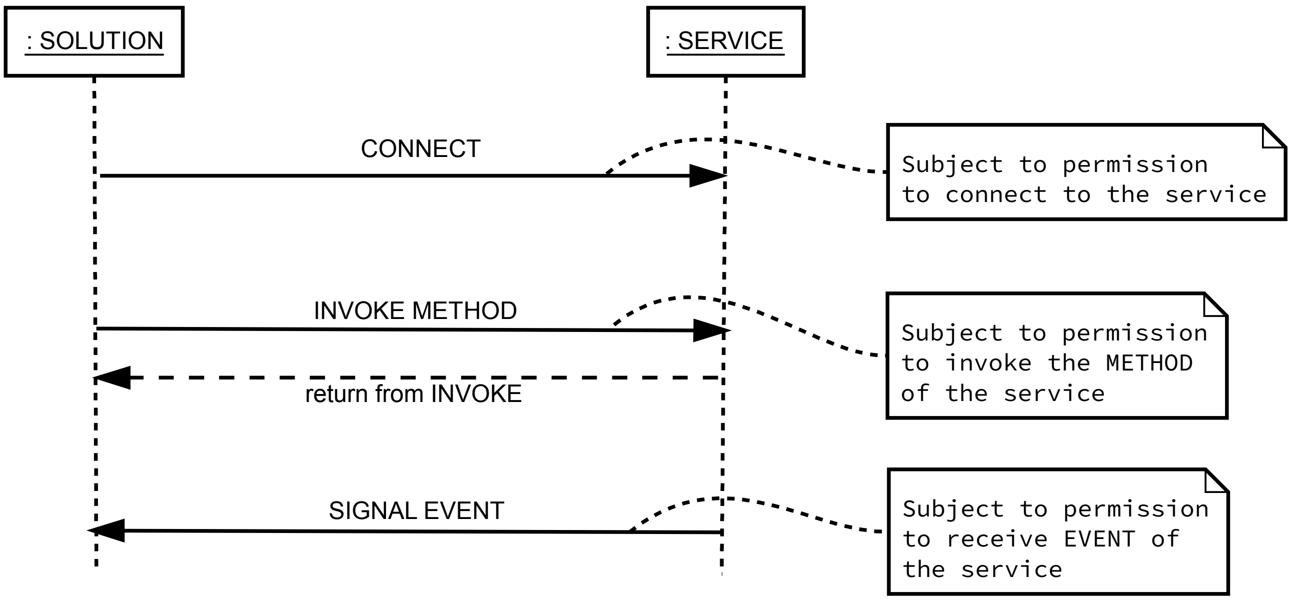
: SOLUTION
: SERVICE
CONNECT
INVOKE METHOD
return from INVOKE
SIGNAL EVENT
Subject to permission to connect to the service
Subject to permission to invoke the METHOD of the service
Subject to permission to receive EVENT of the service

Illustration 3: Aspects of enforcing permissions

To ensure openness of the platform, protection must be managed:

- it should be possible to request permission for resources exposed by tiers
- tiers should have options to expose/restrict permissions for features they provide.

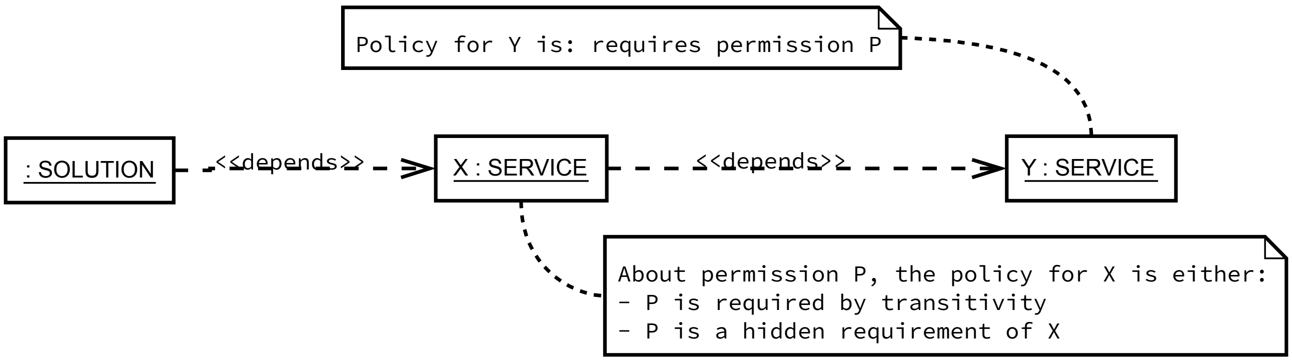
Policy for Y is: requires permission P
: SOLUTION
<<depends>>
X : SERVICE
<<depends>>
Y : SERVICE
About permission P, the policy for X is either:
- P is required by transitivity
- P is a hidden requirement of X

Illustration 4: Aspect of composition of services

1.4. Solutions might offer services

A solution is made of applications and/or services. Services are needed for:

- providing facilities to other applications (library, extension, ...)
- sharing a common state between client applications (local server, …)
- drive accesses to hardware features (user space driver API)

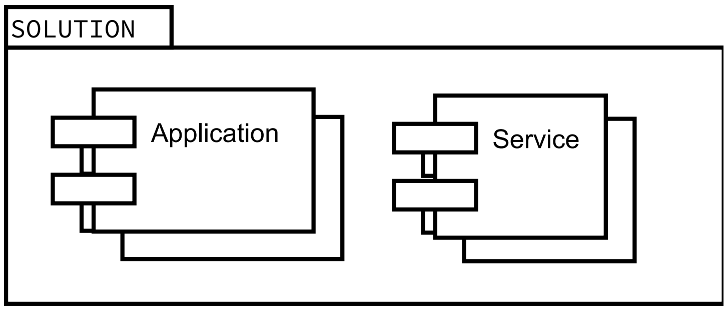

Illustration 5: Composition of solutions

Solutions can be written using many languages, examples: HTML5, C, C++, java, rust, go, vala, … The application framework provides facilities independently of chosen language.

1.5. Application framework is a critical service

Technically, the application framework is a platform service Launched at boot time by systemd.

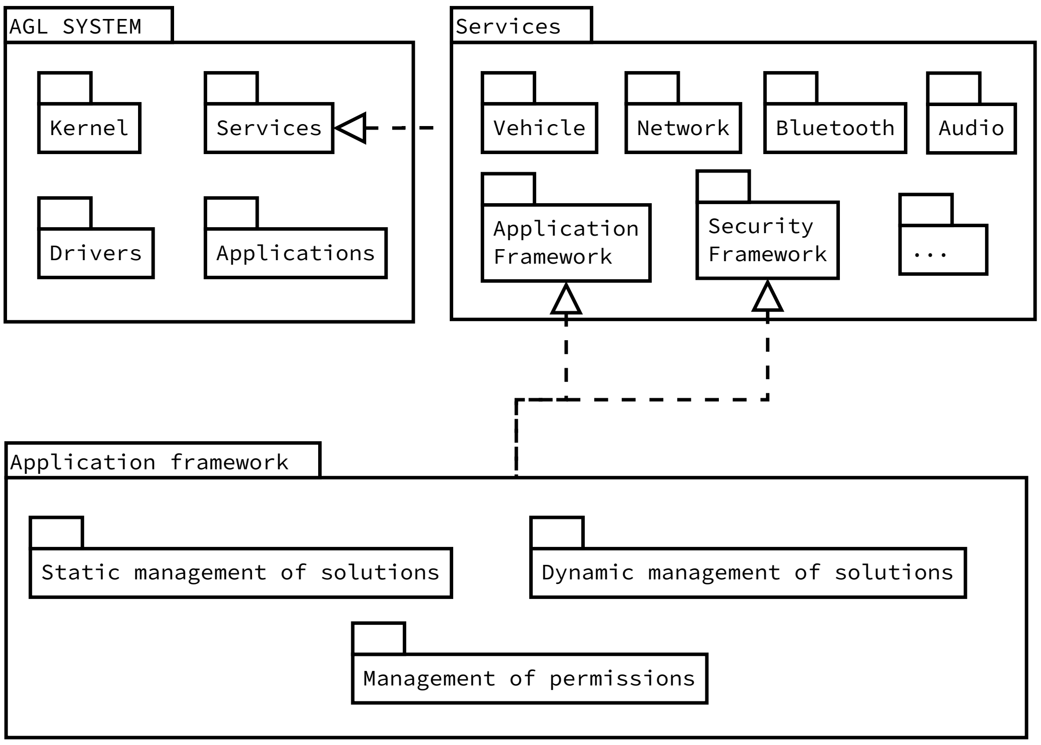

Illustration 6: Framework is a service

The framework offers 3 key services:

- static management of solutions also known as “package” management (install, remove update)
- dynamic management of solutions also known as “lifecycle” management (start, stop, pause, resume)
- management of permissions (request needed autorisation, restict usage of expose features)

It also implements parts of the security framework because of **tight imbrication due to access control provisioning and enforcing.**

2. How works the application framework

2.1. Solutions are installed in a common location

All solutions are installed in a common location and are potentially available for any execution context. Further development should provide filtering of access to applications depending on the identity or authorization of the user.

2.2. Solutions are isolated

When running, a solution owns a private label. This label is used by Smack to control any access of any process. Files created by the application will be tagged with this label. It ensures that only the application can access these files.

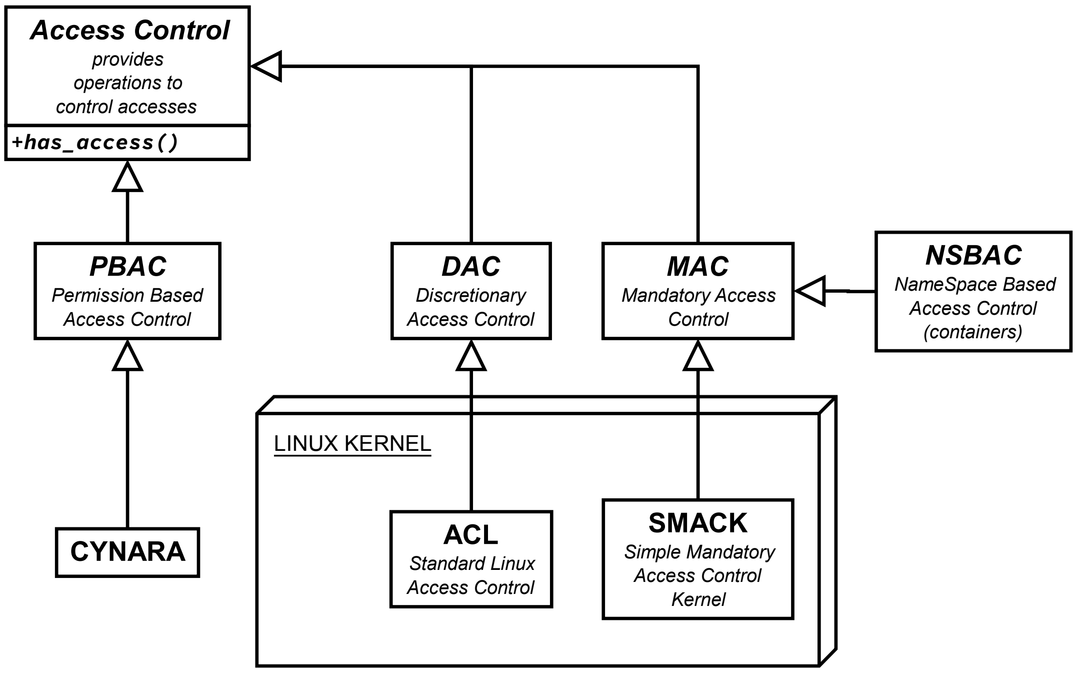
Access Control
provides operations to control accesses
+has_access()
PBAC
Permission Based Access Control
DAC
Discretionary Access Control
MAC
Mandatory Access Control
NSBAC
NameSpace Based Access Control (containers)
CYNARA
LINUX KERNEL
ACL
Standard Linux Access Control
SMACK
Simple Mandatory Access Control Kernel

Illustration 7: Access control

The application framework uses Smack and standard linux discretionary access control. Enforcing isolation with containers or name-space based access control is feasible but not implemented in current version.

2.3. Accesses to services are checked

Cynara service implements a permission based access control that allows the application framework to accesses control application usage of API with a fine grain level.

A services is only a special class of application. Services are subject to the same restrictions as any other applications.

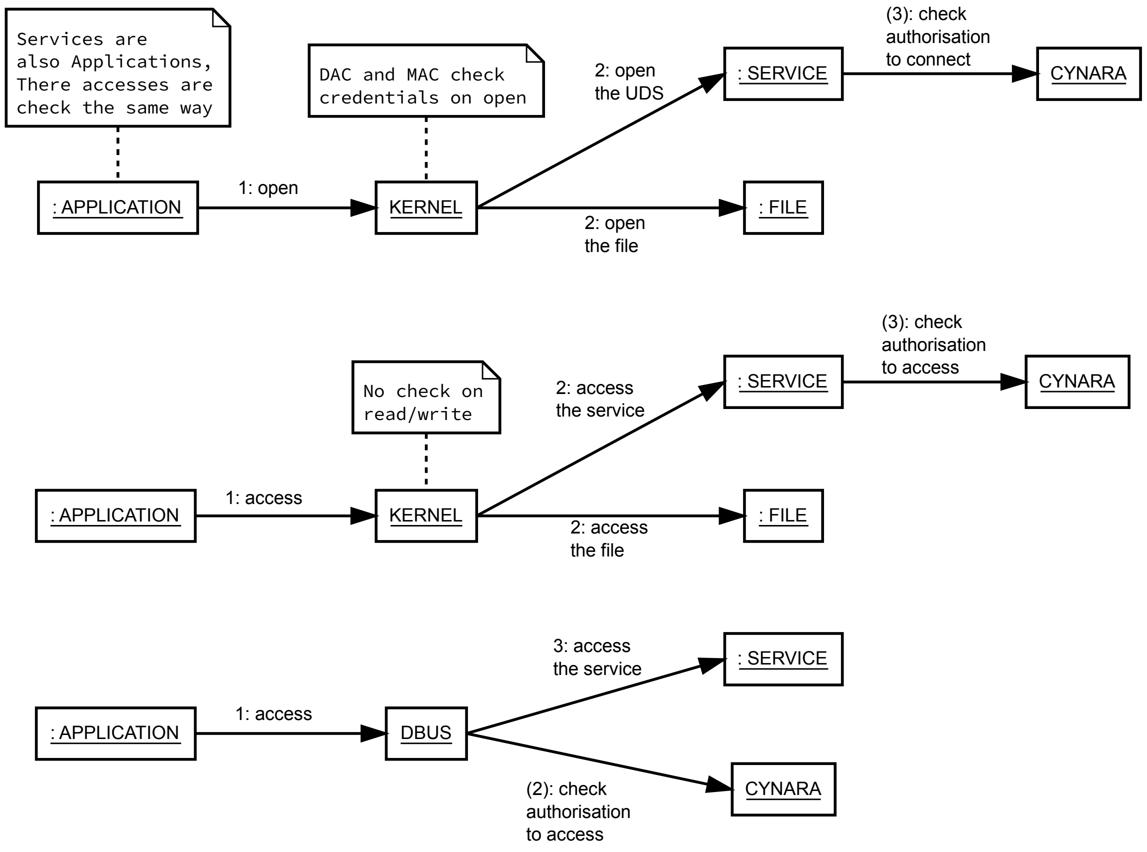

Illustration 8: Access to services and files

2.4. The binder a standard model to expose service API

A component named app-framework-binder (aka binder) provides access to a JSON API over HTTP, WebSocket or Dbus (transport options may be extended in the future).

It is intended to provide:

- a simple HTTP server for files (for HTML5 applications)
- integration of native libraries for HTML5 or other applications (ie: QT)
- high level accesses to API of services
- a framework for creating integrated services

- a framework for event and signals propagation

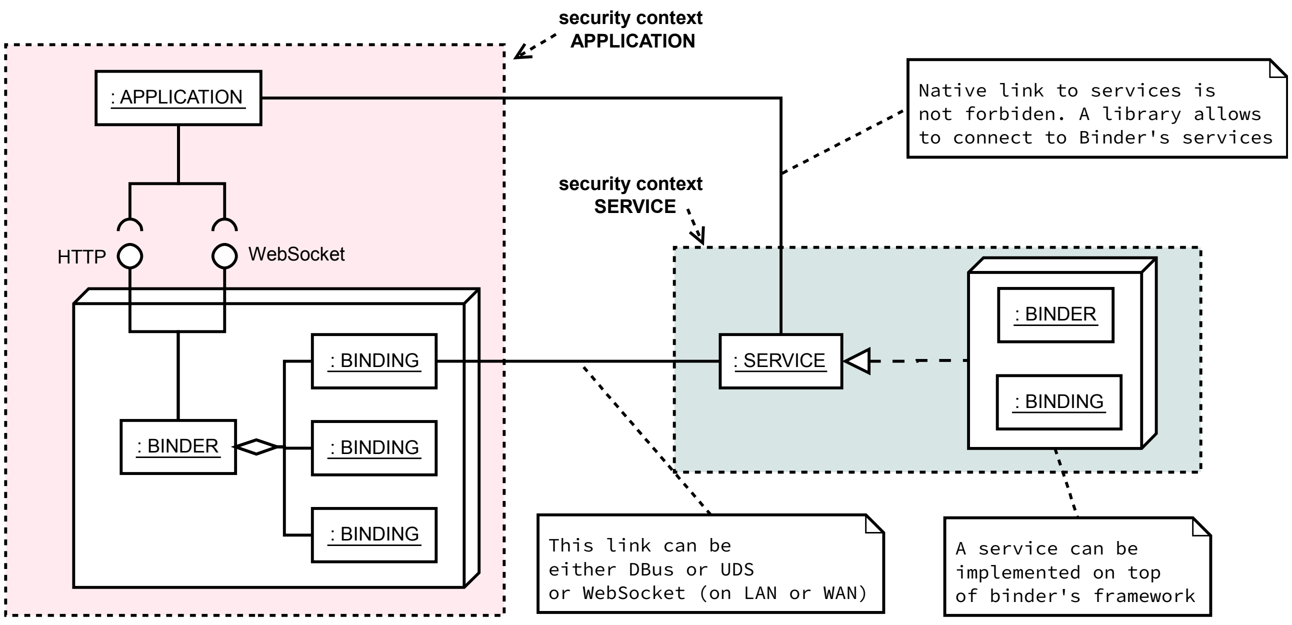
security context
APPLICATION
: APPLICATION
HTTP
WebSocket
: BINDER
: BINDING
: BINDING
: BINDING
security context
SERVICE
: SERVICE
: BINDER
: BINDING
Native link to services is not forbiden. A library allows to connect to Binder's services
This link can be either DBus or UDS or WebSocket (on LAN or WAN)
A service can be implemented on top of binder's framework

Illustration 9: binder

2.5. Binding are loosely couple with the binder

Bindings allow developers to expose API and features from a lower level service to application. Binding only care about applications or services business logic. They are independent of binder transport or security model.

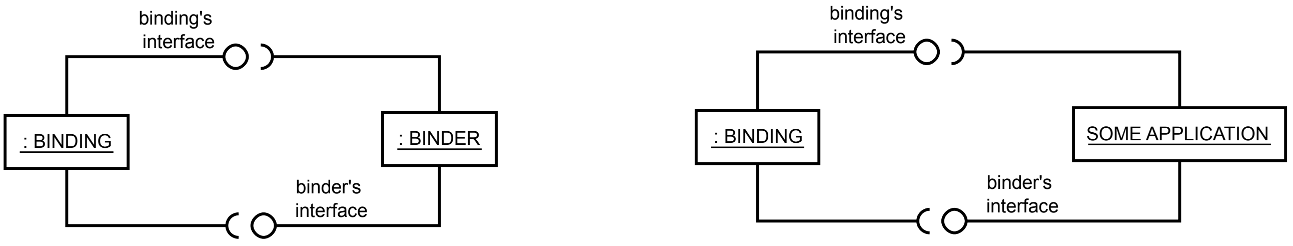

Illustration 10: bindings are true components

A binding is a standalone share library that is not linked to the application framework. Interface in between the application framework binder and its binding is negotiated at initialization time. In the future we may envision changing completely the application framework while keeping existing bindings.

This feature makes bindings true components that can be integrated in any application or in other frameworks.

2.6. Defining new API for AGL

The application framework and its binder provides a solution for defining new API. This solution emphasis the use of JSON streaming of queries and answers. The key advantage of this technique is the ability for services to evolve without breaking existing clients.

2.7. Handling legacy

The framework does not enforce legacy applications to be rewritten. That said, adaptation might be necessary in corners. For these adaptations few strategies are possibles. On lazy option is to keep legacy application isolated running as a sub-system and only enforce access control through MAC/Smack mechanism. On other option is to leverage binder/binding capabilities to either expose or reuse services with the rest of the platform.

2.8. Multi-ECU and Vehicule to Cloud model.

While today applications and services mostly run on the same system, in the near future some application may run on a remote device (e.g. phone) or some services may chose to run on the could(e.g. log, user-preferences, …).

Application framework binder provide API transparency to transport (Dbus, WebSocket, REST) as well as it allows applications/services to run either on the same ECU, multiple ECU or even in a pure Internet mode interact with services hosted on the could.

Technically binding are like a Lego brick. People may choose to agglomerate many of them in a unique binder (i.e. cost/performance) or to split them on different binders with different security profile and even to split them on multiple ECU.

3. Current state

The two following illustrations are summarizing the current state versus target concerning the static and dynamic management of applications.

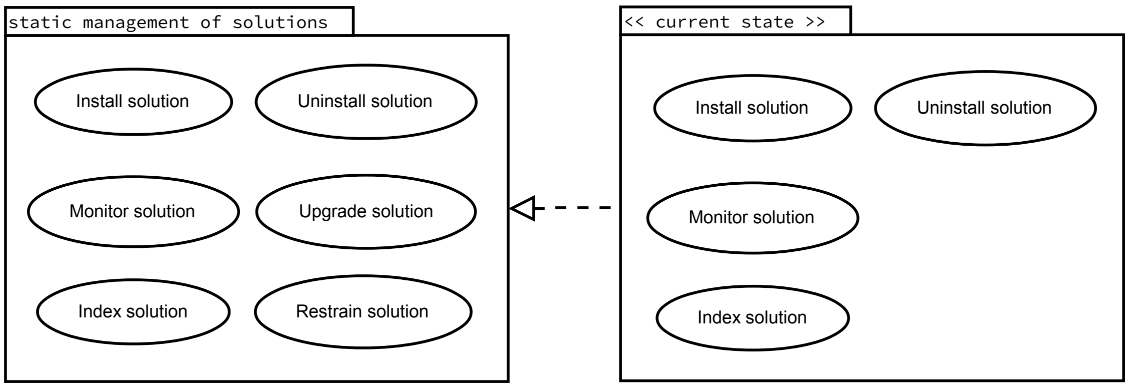

Illustration 11: current state of static management

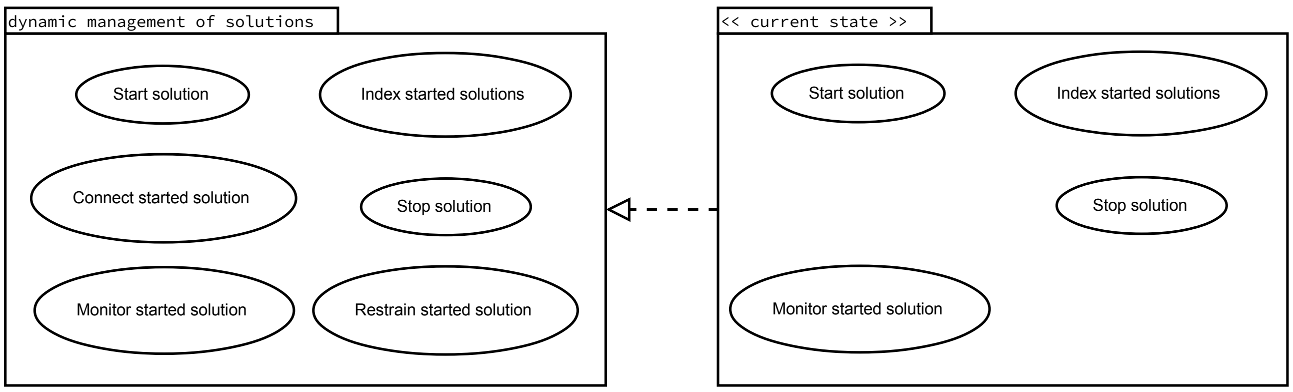

Illustration 12: current state of dynamic management