

Building Applications with AGL Framework

José Bollo



IOT  BZH



AGL Member Meeting Fall
September 7 - 8, 2016
Munich, Germany

Aren't you just hungry to bring your application inside AGL ?



Get the right tools !



your

MOTIVE
LINX
eting Fall
- 8, 2016
Germany



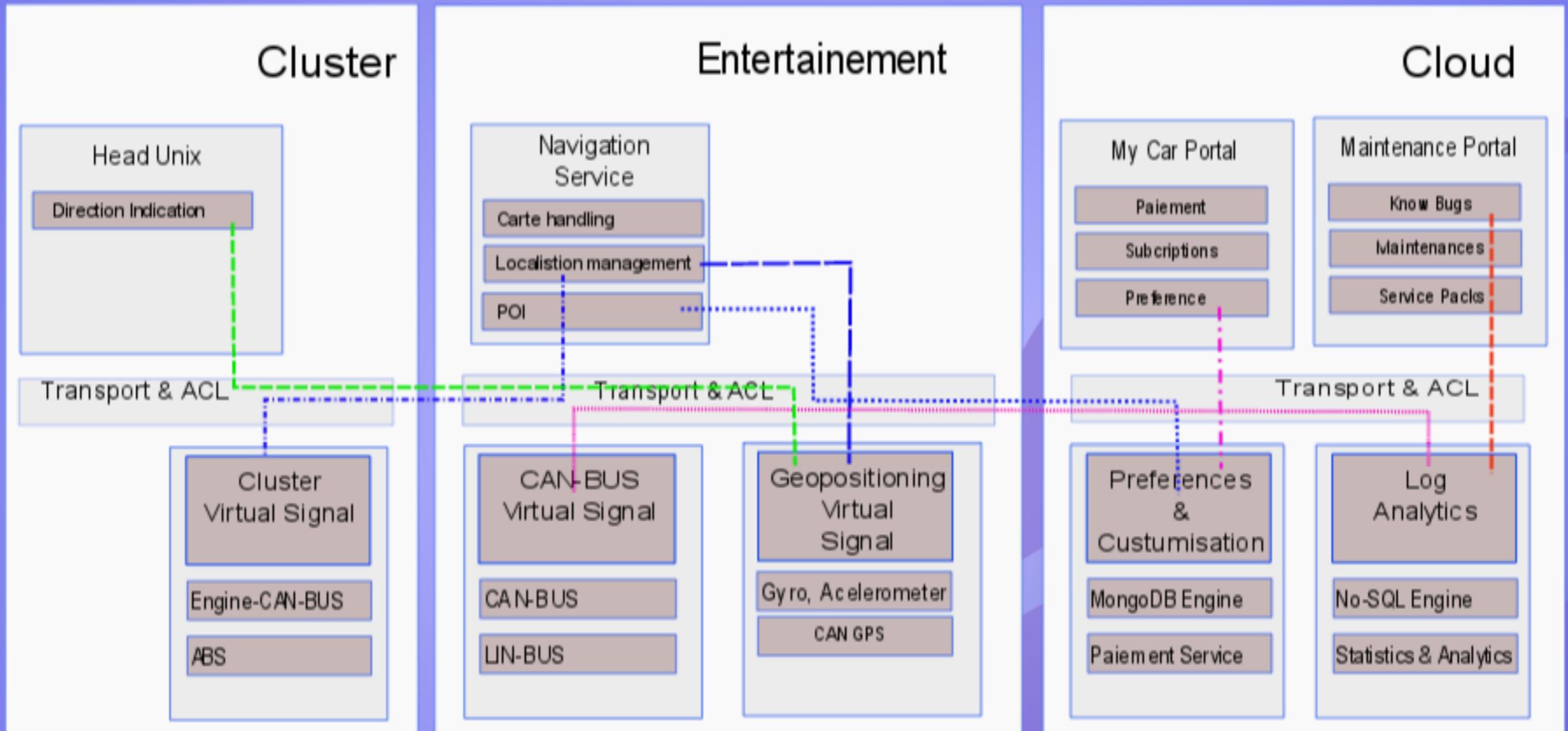
Then you should first ask
What is my application ?
an UI ? a service ?
a client of other services ?

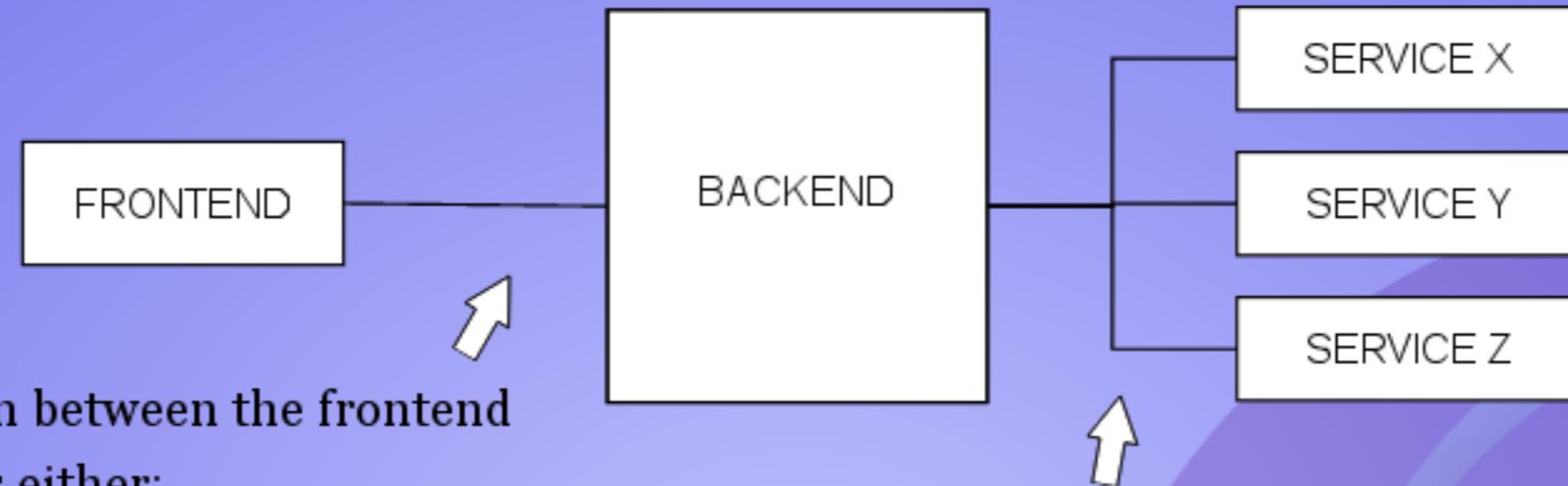
and ...

Where runs my

application ?

on car ? on a device ? on cloud ?





The connection between the frontend and backend is either:

- **null when in the same process**
- **local if in the same ECU**
- **remote if the frontend runs on a connected device**

The connection between the backend and services or between services are either:

- **null when in the same process**
- **local if in the same ECU**
- **distributed if running on different ECUs**
- **remote if services is on cloud**

- connection between the front end and back end is either:
 - null when in the same process
 - local if in the same ECU
 - remote if the frontend runs on a connected device

- either:
 - null when in the same process
 - local if in the same ECU
 - distributed if running on different ECUs
 - remote if services is on cloud

How to connect?

HTTP? HTTPS? DBUS?

WS? WSS? UDS? Denso's Service Bus? ...

Who knows?

The framework knows!

It knows:

- How to install applications
- How to launch applications
- How to connect services
- How to ensure **security** and privacy

The AGL framework (AF) is made of 2 parts:

Agl Framework - **MAIN**

af-main

Agl Framework - **BINDER**

af-binder

n environment and
ities
security contexts

af-binder: the b

- provides **afb-daem**

(with HTTP, DBUS)

af-main: the main environment and utilities

- manages security contexts
- installs and uninstalls applications
(delivered as W3C signed widgets)
 - starts and stops applications
- provides data on installed and running applications
 - command line utility **afm-util**
 - multi-user capable
 - tools for packaging

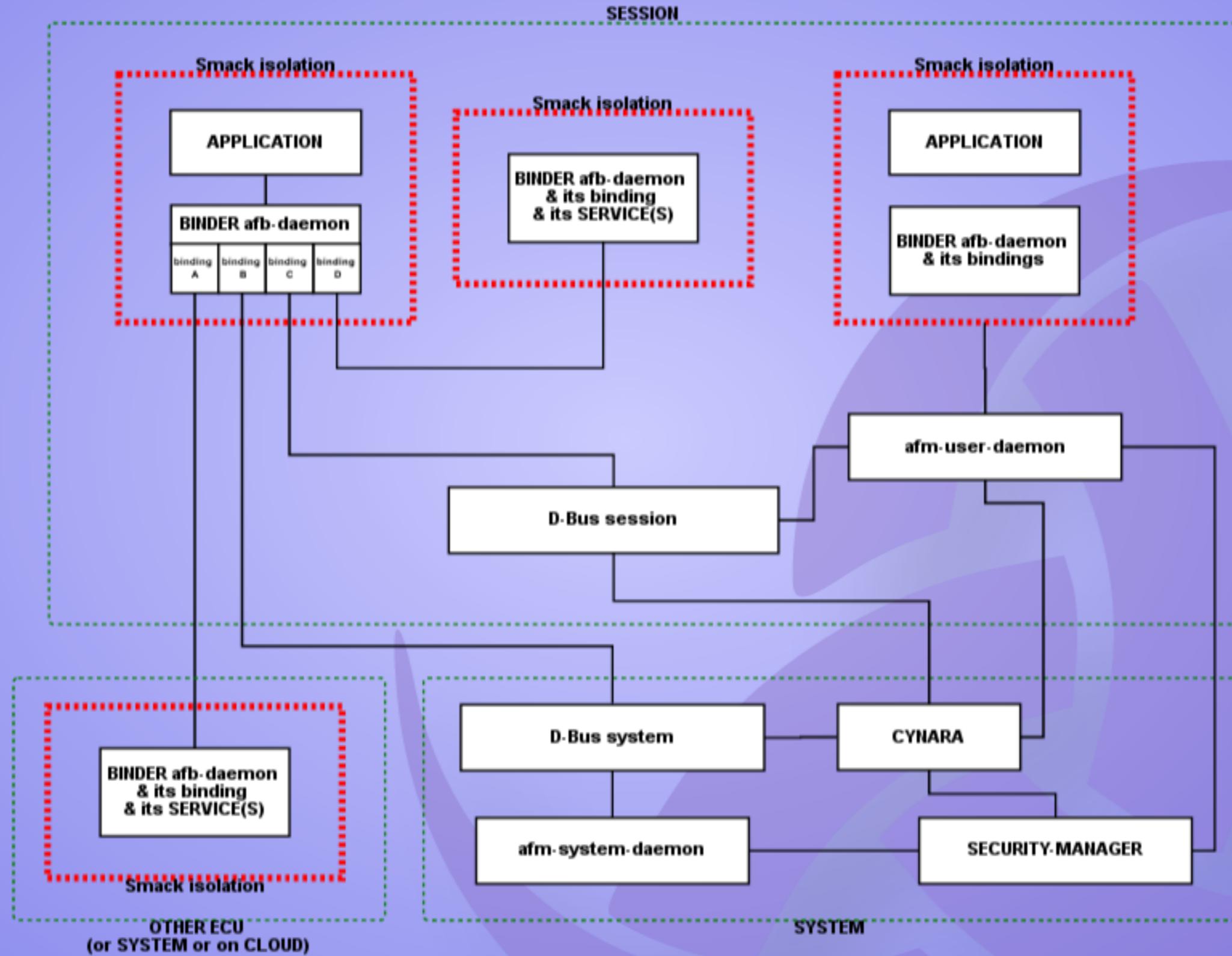
Agl Framework - BINDER

af-binder

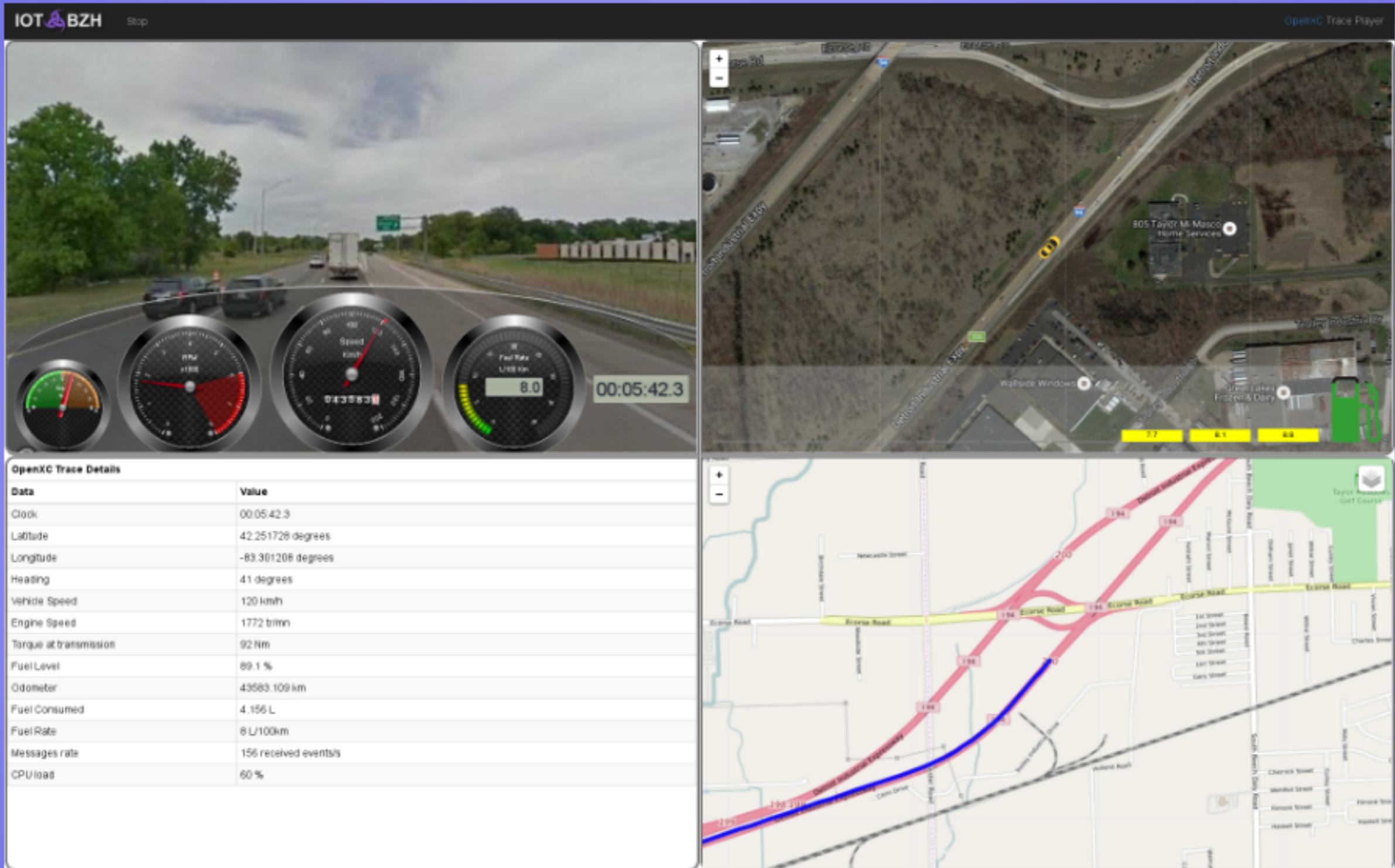
af-binder: the backend binder

- provides **afb-daemon**, the default backend
(with HTTP, DBUS, WS, ...)
- universal versatile API using agility of JSON
- establishes connection to services: DBUS and WS
- model and environment for creation of applications

illustration



The demo TXC



The demo TXC is a player of openXC traces:
the TXC binding render events of OpenXC's traces
TXC clients subscribe to expected events



Writing an application for the framework :

Write the backend binding

- Bindings for **afb-daemon** are written using C
- A binding has a **unique name**
- Bindings expose **methods** within its namespace
- Bindings may publish **events** to its subscribers
- Data are **JSON** compatible objects (json-c)
- Binding may consume other bindings

Writing an application for the framework :

Write the backend binding

- Bindings for **afb-daemon** are written using C
- A binding has a **unique name**
- Bindings expose **methods** within its namespace
- Bindings may publish **events** to its subscribers
- Data are **JSON** compatible objects (json-c)
- Binding may consume other bindings
- Event loop and D-Bus access through **libsystemd**

Write the frontend application

- Frontend can be written:

- Event loop and D-Bus access through `libsystemd`

Write the frontend application

- Frontend can be written:
 - in HTML5+CSS+JS
 - using QML
 - using any language
- Accesses to backend and service:
 - secured by key
 - using names of the **binding** and its **method**
 - communication using
 - REST over HTTP
 - JSON-RPC like protocol over WS
 - eventually JSON over D-Bus

- JSON-RPC like protocol over WS
- eventually JSON over D-Bus

Write the configuration and package the application

- Write the configuration file
 - Follows the W3C widget specification
 - Use **feature** to handle AGL specificities
- Package the application
 - Create the appropriate directories
 - Sign your delivery (ex. using **wgtpkg-sign**)
 - Pack to a widget (ex. using **wgtpkg-pack**)



The demo TXC is a player of openXC traces:
 the TXC binding render events of OpenXC's traces
 TXC clients subscribe to expected events

binding: Initialisation

```
static const struct afb_verb_desc_v1 verbs[] = {
    {"start",      AFB_SESSION_CHECK, start      , "start to play a trace"},
    {"stop",       AFB_SESSION_CHECK, stop       , "stop to play a trace"},
    {"subscribe",   AFB_SESSION_CHECK, subscribe  , "subscribes to the event of 'name'" },
    {"unsubscribe", AFB_SESSION_CHECK, unsubscribe , "unsubscribes to the event of 'name'" },
    {NULL}
};

static const struct afb_binding plugin_desc = {
    .type = AFB_BINDING_VERSION_1,
    .v1 = {
        .info = "trace openXC service",
        .prefix = "txc",
        .verbs = verbs
    }
};

const struct afb_binding *afbBindingV1Register (const struct afb_binding_interface *itf) {
    afbitf = itf;
    return &plugin_desc;
}
```

binding: Handling requests

```
static void subscribe_unsubscribe(struct afb_req request, int subscribe) {
    int ok, i, n; struct json_object *args, *a, *x;
```

```
atbitt = itt;
return &plugin_desc;
}
```

binding: Handling requests

```
static void subscribe_unsubscribe(struct afb_req request, int subscribe) {
    int ok, i, n; struct json_object *args, *a, *x;
    /* makes the subscription/unsubscription */
    args = afb_req_json(request);
    if (args == NULL || !json_object_object_get_ex(args, "event", &a)) {
        ok = subscribe_unsubscribe_all(request, subscribe);
    } else if (json_object_get_type(a) != json_type_array) {
        ok = subscribe_unsubscribe_name(request, subscribe, json_object_get_string(a));
    } else {
        ok = subscribe_unsubscribe_array(request, subscribe, a);
    }
    /* send the report */
    if (ok)
        afb_req_success(request, NULL, NULL);
    else
        afb_req_fail(request, "error", NULL);
}

static void subscribe(struct afb_req request) {
    subscribe_unsubscribe(request, 1);
}

static void unsubscribe(struct afb_req request) {
    subscribe_unsubscribe(request, 0);
}
```

binding: Subscribing to event

```

    subscribe_unsubscribe(request, 1);
}

static void unsubscribe(struct afb_req request) {
    subscribe_unsubscribe(request, 0);
}

```

binding: Subscribing to event

```

struct signal {
    const char *name;
    struct afb_event event;
};

static int subscribe_unsubscribe_sig(struct afb_req request, int subscribe, struct signal *sig) {
    if (!afb_event_is_valid(sig->event)) {
        if (!subscribe)
            return 1;
        sig->event = afb_daemon_make_event(afbif->daemon, sig->name);
        if (!afb_event_is_valid(sig->event))
            return 0;
    }
    if (subscribe)
        return afb_req_subscribe(request, sig->event) >= 0;
    else
        return afb_req_unsubscribe(request, sig->event) >= 0;
}

```

binding: Sending events

```

static void send_trace(const char *name, struct json_object *object) {
    struct signal *sig = getsig(name);

```

```
    if (!subscribe)
        return 1;
    sig->event = afb_daemon_make_event(afbitf->daemon, sig->name);
    if (!afb_event_is_valid(sig->event)) {
        return 0;
    }
}
if (subscribe)
    return afb_req_subscribe(request, sig->event) >= 0;
else
    return afb_req_unsubscribe(request, sig->event) >= 0;
}
```

binding: Sending events

```
static void send_trace(const char *name, struct json_object *object) {
    struct signal *sig = getsig(name);
    if (sig && afb_event_is_valid(sig->event))
        afb_event_push(sig->event, json_object_get(object));
}
```

client: Connecting

```
function onOpen() {
    ws.call("txc/subscribe", {event:[
        "engine_speed",
        "fuel_level",
        "fuel_consumed_since_restart",
        "longitude",
        "latitude",
        "odometer",
        "latency"]});
```

```
    struct signal *sig = getsig(name);
    if (sig && afb_event_is_valid(sig->event))
        afb_event_push(sig->event, json_object_get(object));
}
```

client: Connecting

```
function onOpen() {
    ws.call("txc/subscribe", {event:[
        "engine_speed",
        "fuel_level",
        "fuel_consumed_since_restart",
        "longitude",
        "latitude",
        "odometer",
        "vehicle_speed",
        "START",
        "STOP"]}, onSubscribed, onAbort);
    ws.call("stat/subscribe", true);
    ws.onevent("stat/stat", gotStat);
}
```

```
function doConnect() {
    document.body.className = "connecting";
    setMapsLockState(false);
    ws = new afb.ws(onOpen, onAbort);
}
```

client: Linking handlers

```
function onSubscribed() {
```

```
        ws.onevent("stat/stat", gotStat);
    }

function doConnect() {
    document.body.className = "connecting";
    setMapsLockState(false);
    ws = new afb.ws(onOpen, onAbort);
}
```

client: Linking handlers

```
function onSubscribed() {
    document.body.className = "connected";
    setMapsLockState(false);
    ws.onevent("txc/engine_speed", gotEngineSpeed);
    ws.onevent("txc/fuel_level", gotFuelLevel);
    ws.onevent("txc/fuel_consumed_since_restart", gotFuelSince);
    ws.onevent("txc/longitude", gotLongitude);
    ws.onevent("txc/latitude", gotLatitude);
    ws.onevent("txc/odometer", gotOdometer);
    ws.onevent("txc/vehicle_speed", gotVehicleSpeed);
    ws.onevent("txc/START", gotStart);
    ws.onevent("txc/STOP", gotStop);
    ws.onevent("txc", gotAny);
}
```

client: Handler

```
function gotVehicleSpeed(obj) {
    vspeed = Math.round(obj.data.value);
    wdgVsp.innerHTML = String(vspeed);
    gauges.speed.setValue(vspeed);
}
```

```
document.body.className = "connected";
setMapsLockState(false);
ws.onevent("txc/engine_speed", gotEngineSpeed);
ws.onevent("txc/fuel_level", gotFuelLevel);
ws.onevent("txc/fuel_consumed_since_restart", gotFuelSince);
ws.onevent("txc/longitude", gotLongitude);
ws.onevent("txc/latitude", gotLatitude);
ws.onevent("txc/odometer", gotOdometer);
ws.onevent("txc/vehicle_speed", gotVehicleSpeed);
ws.onevent("txc/START", gotStart);
ws.onevent("txc/STOP", gotStop);
ws.onevent("txc", gotAny);
}
```

client: Handler

```
function gotVehicleSpeed(obj) {
    vspeed = Math.round(obj.data.value);
    wdgVsp.innerHTML = String(vspeed);
    gauges.speed.setValue(vspeed);
}
```

Packaging: config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.w3.org/ns/widgets" id="txc-demo" version="0.1">
    <name>OpenXC Trace Player</name>
    <icon src="icon.png"/>
    <content src="index.html" type="application/vnd.agl.html.hybrid"/>
    <description>This is a demo using OpenXC trace</description>
    <author>Jose Bollo &lt;jobol@iot.bzh&gt; / Stephane Desneux &lt;sdx@iot.bzh&gt;</author>
    <messageAPI>0.0.1</messageAPI>
```

```
wdgVsp.innerHTML = String(vspeed);
gauges.speed.setValue(vspeed);
}
```

Packaging: config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.w3.org/ns/widgets" id="txc-demo" version="0.1">
  <name>OpenXC Trace Player</name>
  <icon src="icon.png"/>
  <content src="index.html" type="application/vnd.agl.html.hybrid"/>
  <description>This is a demo using OpenXC trace</description>
  <author>Jose Bollo &lt;jobol@iot.bzh&gt; / Stephane Desneux &lt;sdx@iot.bzh&gt;</author>
  <license>APL 2.0</license>
  <feature name="urn:AGL:provides-binding">
    <param name="name" value="txc"/>
    <param name="src" value="lib/txc-binding.so"/>
    <param name="type" value="application/vnd.agl.binding.v1"/>
    <param name="scope" value="private"/>
  </feature>
  <feature name="urn:AGL:provides-binding">
    <param name="name" value="stat"/>
    <param name="src" value="lib/stat-binding.so"/>
    <param name="type" value="application/vnd.agl.binding.v1"/>
    <param name="scope" value="private"/>
  </feature>
</widget>
```

Packaging: building the widget

Known application types

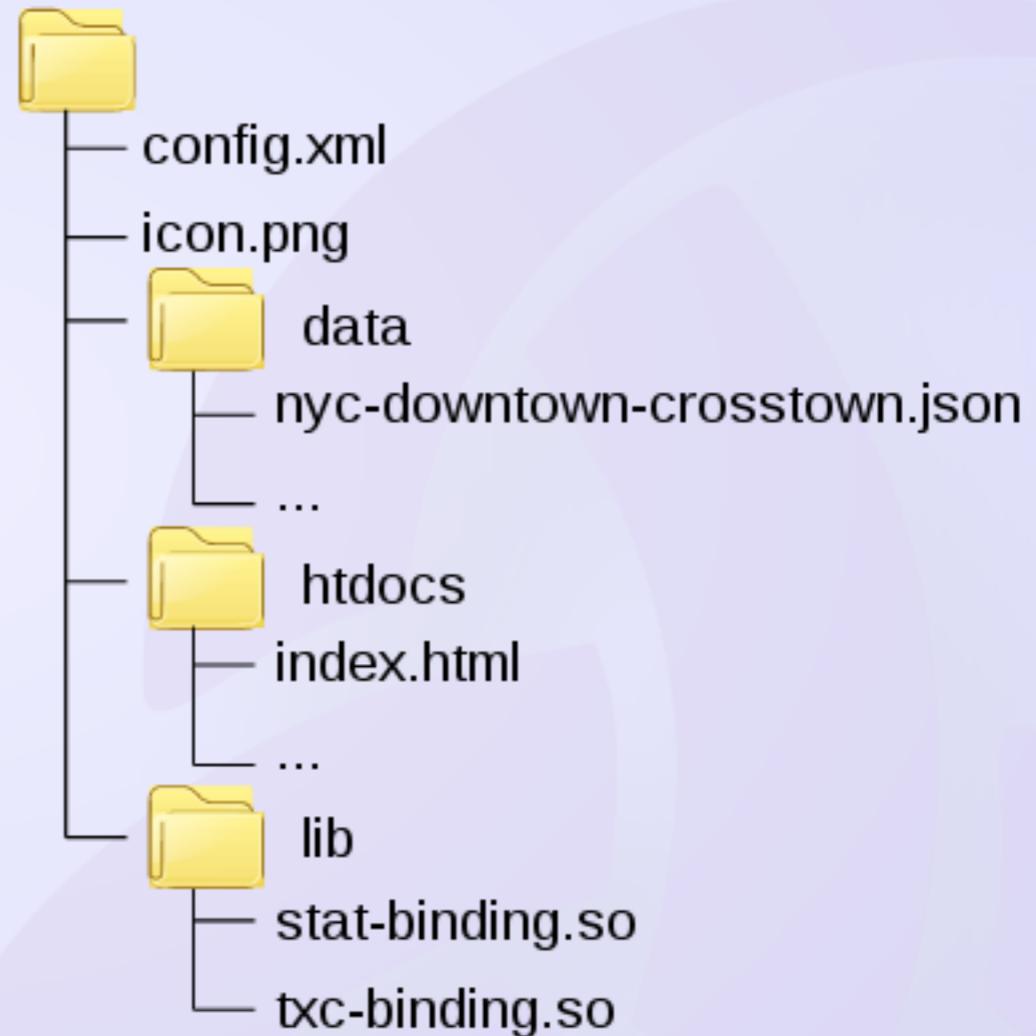
- **text/html** : Pure HTML+JS+CSS application
- **application/x-executable** : Native binary application
 - **application/vnd.agl.url** : Just a URL link
- **application/vnd.agl.service** : A public service through bindings no IHM
- **application/vnd.agl.native** : A native application with own bindings
- **text/vnd.qt.qml, application/vnd.agl.qml** : A QML application
- **application/vnd.agl.qml.hybrid** : A QML application with own bindings
- **application/vnd.agl.html.hybrid** : Html application with own bindings

Any of this types will use public bindings it declares to need

```
<param name="src" value="lib/stat-binding.so"/>
<param name="type" value="application/vnd.agl.binding.v1"/>
<param name="scope" value="private"/>
</feature>
</widget>
```

Packaging: building the widget

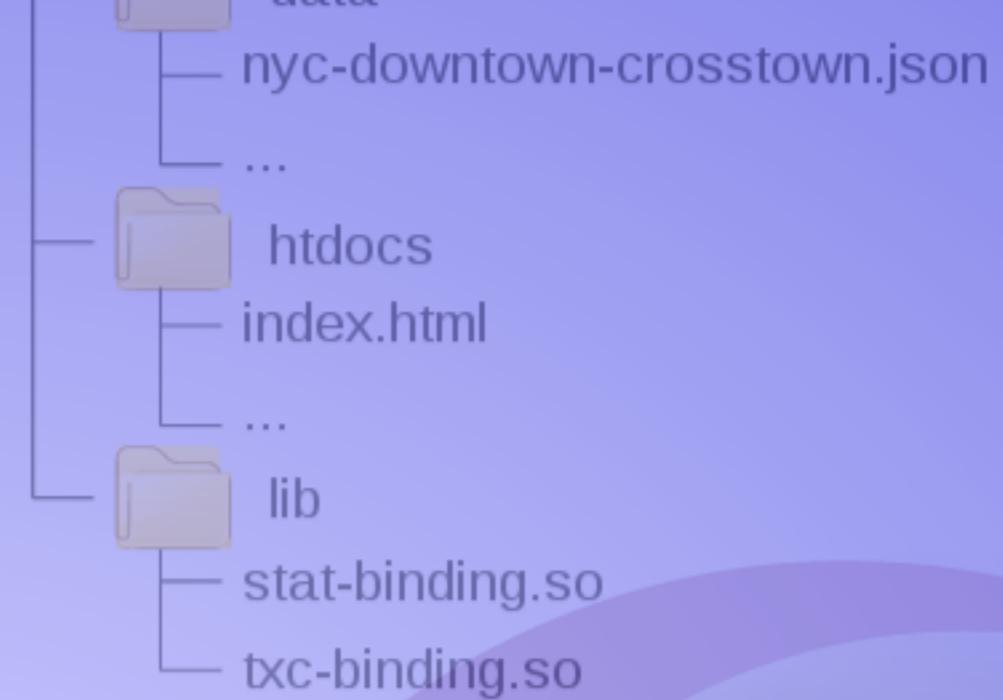
```
mkdir -p package/lib
gulp widget-config-prod
cp -a dist.prod package/htdocs
cp icon_hybrid_html5_128.png package/icon.png
mv package/htdocs/config.xml package/
cp -a data package/
cp *.so package/lib
wgtpkg-pack -f -o txc-demo.wgt package
```



Installing and running

```
afm-util install txc-demo.wgt
```

```
cp *.so package/lib  
wgtpkg-pack -f -o txc-demo.wgt package
```



Installing and running

```
afm-util install txc-demo.wgt  
afm-util start txc-demo@0.1
```

Board Porter

as an average of 424

an average of 9000

accelerated

to divided by 424)
speed

Porter

event per

sec

~ 33500

~ 25400

José Bollo from IOT&BZH

Bench using TXC-demo on board Porter

- 1)** The file **nyc-downtown-crosstown.json** has an average of 424 events per second
- 2)** A 500kbps one CAN bus can send up to an average of 9000 messages (events) per second
- 3)** **txc-demo** can play traces accelerated

then) playing the file at speed **x21** (near 9000 divided by 424) represent a CAN bus at full speed

Bench result on board Porter

listener

maximum

event per

Bench result on board Porter

listener	maximum speed	event per sec
no listener	x79	~ 33500
1 listener in same process	~ x60	~ 25400
1 listener WS	~ x35	~ 14800
1 listener DBUS	~ x15	~ 6300
3 listeners WS	~ x12	~ 5100
3 listeners DBUS	~ x6	~ 2500

Advantages of using framework

- legacy and inherited application already work in secured environment
- versatility and agility of JSON RPC provides stability for future
 - programming model of bindings is mature
 - model of services is open and secured
 - model of events is efficient and secured

14800	2500
5100	
6300	

Conclusion

The framework is integrated in AGL 2.0.

Take it, use it, try to break it

Give feedback and send request for
improvements

LINKS

- Documentation: <http://iot.bzh/download/public/2016/appfw/>
- txc-demo.wgt: <http://iot.bzh/download/public/2016/afb-demos/>
 - source code: <https://github.com/iotbzh/tzc-demo>