Beyond CAN Signaling Agent
Signaling Requirement @ AGL

• Clear Isolation
  • Low level CAN operation only depends on equipment
  • High level business logic dedicated to applications

• Security Built In
  • Navigation APP may access GeoLocation but not Telephony
  • Implement statistic/counter to monitor unexpected behaviour
  • Leverage AGL-CC framework

• API transparency for client applications
• Reuse existing technology (faster, cheaper, safer)
Clear Isolation

UI

Binder

CAN High Level Binding(s)
- Logic
- Aggregation («vehicle.doors.any.open»)
- Advanced Ops

Signals - «vehicle.doors.left.open»
(Binder Events)

CAN Low Level Binding(s)
- Decoding / Encoding
- Authentication / Crypto / Firewalling
- Transaction (set... ack...)
- Stats & Maths
- Caching (low freq. Signals, get() call)
- Debug

CAN frames 011010010

CAN BUS
Low Level Binding

• Binary to Json
  • Binary encoding decoding
  • Generate Application Friendly Signal Name & Values

• Close to Automatic Code Generation
  • OpenXC CAN vector definition in JSON
  • Other CAN vector as CANoe

• Include Basic Filtering & Statistic
  • GT/GE LT/LE
  • Timer, Cycle, Timestamps
  • Counter: last value, average, invalid ID, …
  • Can be ship to developer as binary only
High Level Binding

- Provide stable API to applications
- Provide signal composition
- Can be split/overloaded to simplify security
- Should support vehicle to cloud

- As today investigating VIWI-W3C
Security Model base of AGL AppFW
Leveraging AppFW transport & security

```c
static struct afb_auth auths[] = {
    {.type = afb_auth_Permission, .text = "urn:AGL:permission:low-can:partner:read" },
    {.type = afb_auth_Permission, .text = "urn:AGL:permission:low-can:partner:write" },
    {.type = afb_auth_And, .first = &auths[0], .next = &auths[1] }
};

static const struct afb_verb_v2 verbs_v2[] = {
    {.verb = "subscribe", .callback = f_subscribe, .auth = &auths[2], .session = AFB_SESSION_CHECK, },
    {.verb = "unsubscribe", .callback = f_unsubscribe, .auth = &auths[2], .session = AFB_SESSION_CHECK,},
    {.verb = NULL }
};

/* the integer data used by binder for the verbosity of the binder */
int afbBindingV2verbosity;

/* the structure for describing the binder */
const struct afb_binding_v2 afbBindingV2 = {
    .api = "low-can", .specification = /* the JSON description */ .verbs = verbs,
    .init = NULL , .start = start_low_can, .onevent = NULL,
};
```
OpenAPI as API definition tool

```json
{  "openapi": "3.0.0",  "$schema": "file:schema-agl-api-v2.json",  "info": {    "description": "Can Signal Low Level API", "title": "low-can", "version": "2.0"  },  "servers": [{    "url": "ws://{host}:{port}/api/low-can", "description": "The API server.",    "variables": {      "host": { "default": "localhost" },      "port": {"default": "1234"}    },    "X-afb-events": [      { "$ref": "#/components/schemas/afb-event" }    ]  }  },  "components": {    "schemas": {      "afb-reply": {        "properties": {          "jtype": {            "type": "string"          },          "request": { "$ref": "#/components/schemas/afb-request"          },          "response": {            "type": "object"          }        }      }    }  }
```
Monitoring & Debug

• AGL/Master (available from git)
  • Binder Introspection (global or by API)
  • Verbosity change (global or by API)

• AGL/DD (June 2017)
  • API trace (somehow equivalent to tcpdump)
  • Application Log trapping
  • Asynchronous event observation
  • Integration with existing debug tool (ie: wireshark)

• AGL/EE (December 2017)
  • GDB integration
  • Client session tracing through API architecture
  • Central daemon to allow binder discovery, introspection & tracing
Vehicle to Cloud
Signal integration with V2C (AGL/2018)

Local Binding

ws-client:tcp://hostname:port/MyAPI

(1) Request API

(2) Request AuthZ
(clientID@IDP, scope, ..)

(3) Forward AuthZ Request

(10) Forward AuthCode

Identity Agent

(11) Forward AuthCode

Remote Binding

ws-server:tcp://hostname:port/MyAPI

(1) Request API

(2) Request AuthZ
(clientID@IDP, scope, ..)

(3) Forward AuthZ Request

(10) Forward AuthCode

IDP (Identity Provider)
e.g. www.mycarportal.net

(12) Provide AuthCode

(13) Receive User Info

Consent/Authentication
User UI

(4) Request AuthZ on behalf Remote (clientID, scope, ..)

(5) Redirect Authentication URL for User consent

(7) Forward IDP redirect

(9) Return AuthCode

(7) User Consent/Authentication Interaction
Thank you!

Visit GENIVI at http://www.genivi.org or http://projects.genivi.org
Contact us: help@genivi.org

Further information

• Publication at http://iot.bzh/publications
• Documentation http://docs.automotivelinux.org