W3C Automotive BG: F2F in Barcelona
GENIVI Vehicle Web API

Justin(JongSeon) Park
LG Electronics
Justin (JongSeon) Park

• Chief Research Engineer, SW Platform Lab. of LG Electronics

• 10 years experience in embedded system

• Working in automotive industry for 6 years
  - Developed IVI and Telematics system

• Participating in GENIVI Alliance regarding Web Vehicle APIs
Agenda

- Introduction of Web in Automotive
- Use Cases, Characteristics of Vehicle Data
- Considerations
  - Suggested Architecture
  - Principles to define Vehicle APIs
- Introduction of GENIVI Web Vehicle APIs
  - API descriptions
  - Reference Implementation
- Conclusion
- Q&A
Web Technologies for Automotive

The first target will be obviously IVI system

- **Web Browsing in a vehicle**
  - IVI Web Browser: Big Button, Driving Regulation, etc.

- **GUI framework for HMI**
  - Portability, MVC Pattern, Abundant Dev. Pool.

- **Platform for App Store**
  - Easily adding new features even if not for App Store

- **Alternative Mirror Link**
  - Exchange data via meta data instead of transferring the whole screen

Requires Standardized Vehicle APIs
Use Cases for Web API for Vehicle Data

Categorized into three types of WebApps which access vehicle data

- **Home (Main, HMI, Dashboard)** - Installed(Build-in), OEM-provided
  - Major module that access various Vehicle Data
  - Needs almost all vehicle data for both reading/writing

---

Home UI
(Thin WebApp)

Required Well-defined APIs with documentation

HMI/App Framework
(UI Effect, Business Logic)

Middleware
Telematics App for mobile phone - Downloadable, OEM-provided

- Most Apps need to know whether vehicle is moving (regulations)
- Insurance App (Pay-as-you-drive), Any creative Apps in future
- It’s not certain that OEMs will allow Market Apps to access vehicle data
- Are we needed/able to suggest/predict all possible Apps per each data types?

Market App – Downloadable

- Mobile App Framework (Authentication, Communication)
- Mobile App (Remote Control, Diagnosis)
- Telematics Component
- Middleware

Required Well-defined APIs with documentation
We have to understand and consider characteristics of vehicle data

- **Data Characteristic**
  - So many kinds of vehicle data and data types
  - A few Persistent Data - Car Type, VIN*, Model, WMI**, etc.
  - Most data are Transient; status at a moment
  - Only the latest value is meaningful (except GPS data)

- **Vehicle Network Characteristic (usually CAN)**
  - Real data exist somewhere else not in IVI
  - Data is broadcasted rather than query

- **OEM Variations**
  - Unit, Accuracy, Frequency, etc.
  - Policy - Which data are supported, Permissions

VIN* – Vehicle Identification Number (ISO 3779)
WMI** – World Manufacturer Identifier
Considerations on set the scope of Standardization

- From Use cases
  - Market Apps: only a few types is enough
  - OEM-provided Apps: almost all data is candidates

- Two Approaches
  - Select only common data types through broad consensus
    - Hard to define the scope of common due to the variety of OEM
    - Risk to cover very small percentage of data types needed
    - Still might fail to prevent fragmentation → Only for compatibility of Market Apps?
  - Select all possible data types
    - Required much work
    - But it’s easier to subtract than to add
    - Still have an issue that only a part of data types are support depending on models
How to Overcome OEM Variations?

APIs must be very flexible to absorb variety

- Define as many data types as possible to prevent fragment
  - Need to gather OEM requirements as much as possible

- Allow OEMs much freedom to maintain their policy
  - A few mandatory data types
  - Most of data types need to be optional

- Consider flexibility of interface
  - Minimum number of common methods to support various data types
  - Less structured interfaces to absorb changes depending on OEMs
Layered architecture according to characteristics of vehicle network

- **Native Apps**
  - Web Runtime
    - Web App
    - Vehicle API
    - Vehicle Plug-in
  - IPC

- **Vehicle Network Manager**
  - Gateway to vehicle network for Apps
    - Broadcast updates of values
    - Keep the latest values
    - Message encoding/decoding

- **Vehicle Network Stack**
  - Various ways to implement it
  - IPC should cover both web and native apps

- **Vehicle Network Driver**
  - Commercial solution is usually used
    - Full tool chain – simulation, monitoring, automatic code-generation to apply the change of message database
GENIVI has full Web Vehicle API and implementation

- Collected opinions to define the types of supported data
  - GENIVI has over 168 member companies including 11 OEMs
  - To reflect the realistic requirements, OEM survey was conducted

- Total 9 groups and 129 data types are defined
  - Vehicle Information (7)
  - Running Status (26)
  - Maintenance (8)
  - Personalization (20)
  - Driving Safety (16)
  - Vision System (11)
  - Parking (4)
  - Climate/Environment (29)
  - Electric Vehicle (8)

- 9 groups are defined as 9 Interfaces
- 2 methods (get/set) are defined to access all data as the unified way
- getSupportedTypes() method is defined
- All interfaces for data exchange are defined to inherit VehicleEvent interface.
- All vehicle data belong to a type of VehicleEvent and can be accessed as an attribute of that.

```java
[NoInterfaceObject]
interface VehicleEvent : Event {
    interface RunningStatusEvent : VehicleEvent {
        ... 
        readonly attribute unsigned short speedometer;
        readonly attribute unsigned short? engineSpeed;
        ... 
    }
}
```

- `get/set/getSupportedEventTypes` can be accessible via VehicleInterface

```java
[NoInterfaceObject]
interface VehicleInterface : EventTarget {
    void get(VehicleEventType type, VehicleDataHandler handler, ErrorCallback errorCB);
    void set(VehicleEventType type, VehicleEvent data, SuccessCallback successCB, ErrorCallback errorCB);
    VehicleEventType[] getSupportedEventTypes(VehicleEventType type, boolean writable);
}
```
- Well-structured Interface
  - Some data have relations to others; these produce a type of data structure
  - Especially, a Setting method requires a set of attributes at a time
  - Usually, these are defined as a structured data types - Interfaces
  - Good for Clarity. But flexibility is inhibited

```cpp
Interface A_1 : Event {
    attribute A_1_a;
    attribute A_1_b;
    attribute A_1_c;
}

Interface A : Event {
    attribute A_1;
    attribute A_2;
}
```
Less structured interface for flexibility

- Real data: A_1_a, A_1_b, A_1_c, A_2
- Virtual type: A, A_1
- Special attribute "Type" is used as an ID to identify the intended type and the range of validity of data.

Interface A : Event {
    attribute Type;
    attribute A_1_a;
    attribute A_1_b;
    attribute A_1_c;
    attribute A_2;
}

const Type A = "A";
const Type A_1 = "A_1";
const Type A_1_a = "A_1_a";
const Type A_1_b = "A_1_b";
const Type A_1_c = "A_1_c";
const Type A_2 = "A_2";
Handling multiple data at a time (cont’d)

Example code

```javascript
function handleInterfaceA(objA) {
  if (objA.type == "A_1") {
    console.log("value A_1_a = "+objA.A_1_a); // It's valid.
    console.log("value A_1_b = "+objA.A_1_b); // It's valid.
    console.log("value A_1_c = "+objA.A_1_c); // It's valid.
    console.log("value A_2 = "+objA.A_2); // It's possible but the value is invalid in our rules.
  }
  else if (objA.type == "A_2") {
    console.log("value A_2 = "+objA.A_2); // It's valid.
  }
}
```
Tire pressure status in MaintenanceEvent interface

```csharp
interface MaintenanceEvent : VehicleEvent {
    const VehicleEventType MAINTENANCE = "maintenance";
    ......
    const VehicleEventType MAINTENANCE_TIREPRESSURESTATUS = "maintenance_tirepressurestatus";
    const VehicleEventType MAINTENANCE_TIREPRESSURESTATUS_FRONTLEFT = "maintenance_tirepressurestatus_frontleft";
    const VehicleEventType MAINTENANCE_TIREPRESSURESTATUS_FRONTRIGHT = "maintenance_tirepressurestatus_frontright";
    const VehicleEventType MAINTENANCE_TIREPRESSURESTATUS_REARLEFT = "maintenance_tirepressurestatus_rearleft";
    const VehicleEventType MAINTENANCE_TIREPRESSURESTATUS_REARRIGHT = "maintenance_tirepressurestatus_rearright";
    ......
    const unsigned short TIREPRESSURESTATUS_NORMAL = 0;
    const unsigned short TIREPRESSURESTATUS_LOW = 1;
    const unsigned short TIREPRESSURESTATUS_HIGH = 2;
    ......
    readonly attribute unsigned short? tirePressureStatusFrontLeft;
    readonly attribute unsigned short? tirePressureStatusFrontRight;
    readonly attribute unsigned short? tirePressureStatusRearLeft;
    readonly attribute unsigned short? tirePressureStatusRearRight;
    ......
};
```

Capitalization Styles
Pascal case, Attribute -> Camel Case
Getting a single vehicle data

- Let’s get the tire pressure status for the front left tire and notice the status to the driver
- Call the get function with a callback function (handleVehicleData)

```javascript
vehicle.get('maintenance_tirepressurestatus_frontleft', handleVehicleData, handleError);
function handleVehicleData(data) {
    if (data.tirePressureStatusFrontLeft == 0) {
        alert('Tire pressure status (front-left) is normal.');
    } else if (data.tirePressureStatusFrontLeft == 1) {
        alert('Tire pressure status (front-left) is low.');
    } else if (data.tirePressureStatusFrontLeft == 2) {
        alert('Tire pressure status (front-left) is high.');
    }
}
```
Getting multiple vehicle data

- Let’s get tire pressure status for all tires simultaneously
- In the previous way, you have to get the status of each tire.

```javascript
vehicle.get('maintenance_tirepressurestatus_frontleft', handleVehicleData, handleError);
vehicle.get('maintenance_tirepressurestatus_frontright', handleVehicleData, handleError);
vehicle.get('maintenance_tirepressurestatus_rearleft', handleVehicleData, handleError);
vehicle.get('maintenance_tirepressurestatus_rearright', handleVehicleData, handleError);

function handleVehicleData(data) {
    if ((data.tirePressureStatusFrontLeft != 0) || (data.tirePressureStatusFrontRight != 0) ||
        (data.tirePressureStatusRearLeft != 0) || (data.tirePressureStatusRearRight != 0)) {
        alert('Check tire pressure.
    }
}
```

- However, with the upper level type, the code becomes quite simple.

```javascript
vehicle.get('maintenance_tirepressurestatus', handleVehicleData, handleError);
```
Adding event listener(s)

- Let’s add an event listener to monitor the tire pressure status for the front left tire.

```javascript
vehicle.addEventListener('maintenance_tirepressurestatus_frontleft', handleVehicleData, false);
```

- Also, you can use the upper level type to add multiple listeners.

```javascript
vehicle.addEventListener('maintenance_tirepressurestatus', handleVehicleData, false);
```

- A callback function *(handleVehicleData)* is called whenever any of tire pressure status is changed.
Setting a single vehicle data

- Assume that driver seat position can be set in this vehicle.
- Let’s set the driver seat position for recline seatback.

```typescript
interface PersonalizationEvent : VehicleEvent {
  ……
  const VehicleEventType PERSONALIZATION_DRIVERSEATPOSITION = "personalization_driverseatposition";
  const VehicleEventType PERSONALIZATION_DRIVERSEATPOSITION_RECLINE_SEATBACK = "personalization_driverseatposition_reclineseatback";
  const VehicleEventType PERSONALIZATION_DRIVERSEATPOSITION_SLIDE = "personalization_driverseatposition_slide";
  const VehicleEventType PERSONALIZATION_DRIVERSEATPOSITION_CUSHION_HEIGHT = "personalization_driverseatposition_cushionheight";
  const VehicleEventType PERSONALIZATION_DRIVERSEATPOSITION_HEADREST = "personalization_driverseatposition_headrest";
  const VehicleEventType PERSONALIZATION_DRIVERSEATPOSITION_BACKCUSHION = "personalization_driverseatposition_backcushion";
  const VehicleEventType PERSONALIZATION_DRIVERSEATPOSITION_SIDE_CUSHION = "personalization_driverseatposition_sidecushion";
  ……
  readonly attribute unsigned short? driverSeatPositionReclineSeatback;
  readonly attribute unsigned short? driverSeatPositionSlide;
  readonly attribute unsigned short? driverSeatPositionCushionHeight;
  readonly attribute unsigned short? driverSeatPositionHeadrest;
  readonly attribute unsigned short? driverSeatPositionBackCushion;
  readonly attribute unsigned short? driverSeatPositionSideCushion;
  ……
};
```
Setting a single vehicle data

- Create an object \( \text{obj} \) and add an attribute in the \( \text{obj} \).

```
var obj = new Object();
obj.driverSeatPositionReclineSeatback = 0;
vehicle.set('personalization_driverseatposition_reclineseatback', obj, handleSuccess, handleError);
```

Setting multiple vehicle data

- Let’s set all driver seat position.
- Just add attributes to the \( \text{obj} \) and use the upper level type.

```
var obj = new Object();
obj.driverSeatPositionReclineSeatback = 0;
obj.driverSeatPositionSlide = 0;
obj.driverSeatPositionCushionHeight = 0;
obj.driverSeatPositionHeadrest = 0;
obj.driverSeatPositionBackCushion = 0;
obj.driverSeatPositionSideCushion = 0;
vehicle.set('personalization_driverseatposition', obj, handleSuccess, handleError);
```
Pros and Cons

Pros

- Various data types are supported in accordance with GENIVI members
- Seamless way of access for all data types via minimum APIs and interfaces
- Flexibility for various supported types
- Various granularity is possible
- Easily modifiable to fit OEM’s own purpose

Cons

- New way for multiple access might be unfamiliar
  - Especially, when an event handler is registered to listen a group ID, leaf node events are fired to it.
- Data is exchanged as a unified structure - tens of bytes overhead

GENIVI Web Vehicle API is still in progress

- Hope to make it better to reflect many other opinions
Composition of GENIVI Reference Implementation

- License is MPL (Mozilla Public License) v2.0
- Source code is available in the GENIVI git
- Migration to OSS is in progress
- Directory Structure

```
bin
  doc
  html
  script
  src
    VehicleDataSimulator
    VehicleNetworkAdapter
    plugin
```
How to use it?

- Download
  - Currently only available to GENIVI members
  
  ```
  git clone https://git.genivi.org/srv/git/web-api-vehicle
  ```

- Build and Install
  - Script files are provided
  
  ```
  ./script/build-all.sh
  ```

- Run
  - Need to execute 3 Apps separately
  
  ```
  ./bin/VehicleNetworkAdapter &
  ./bin/VehicleDataSimulator
  google-chrome ./html/index.html (Need to open html on browser)
  ```

VIN* – Vehicle Identification Number (ISO 3779)
WMI** – World Manufacturer Identifier

GENIVI is a registered trademark of the GENIVI Alliance in the USA and other countries
Copyright © GENIVI Alliance 2011
Screenshot from run-time

- Made as simple as possible rather than looking nice
  - To help understanding easily from the source code
  - To let developers test a certain feature
Simple Demonstration
How to standardize Web Vehicle API successfully?

- **Flexibility**
  - Vehicle API depends on rigid factors such as vehicle network protocol and OEM’s policy

- **Generality**
  - Should be fit for many OEM’s requirements
  - Limited coverage will cause additional work and fragmentation, which make it less meaningful

- **Timing**
  - Web Vehicle API needs to be standardized very soon
  - Many OEMs are now working on it in their own way
  - As time goes on, it will be harder to convince OEMs to adopt it
Any Questions?