OMA Report on Automotive Opportunity

Download current working formatted draft oma telecom and automotive 1-17[1].pdf

---

Table of Draft Inputs

1. Scope

1.1 OMAuto Goal

- Establish a venue for discussion between telecom and automotive at a technical and industry level to establish ***any network, any automobile*** connectivity.
- Adapt select current specifications to optimize for the needs of the Automotive market.
- Create a path for Auto industry to interface with the rest of IoT via standardized enablers.
- Bridge existing standards with standardization efforts in the Automotive sector.

Enable a path for automakers and Mobile Network Operators (MNOs) to encourage communications interoperability across automotive and wireless industries.

1.2 Deliverables

Medium = technical report with a set of agreed recommendations to drive future standardization work (25-50 pages)

- Get agreement across participating parties as to what work should be done and where (which SDO) it needs to be done.
- Examine landscape across mobile and automotive environment
- Identify areas of overlap across industry work groups (RVI, W3C, GSMA, 3GPP, OMA, Smart Device Link, Apple CarPlay and Android Auto, OCF, etc…)
- Produce use cases/examples including interaction between IoT and automobile
- Roadmap and proposed timeline of future work needed, for example
    - Security-related issues
    - Enabler creation or reuse to facilitate development of enhanced services
    - Collaborate with additional verticals (environmental, safety, healthcare)
- Deliverables publication to be public and available online
- Reference code-based projects already underway
- Clarify what should be standardized, open source vs. proprietary to encourage interoperability
- Other TBD

2. References

Reference	Details
\[GotAPI\]	Generic Open Terminal API Framework (GotAPI), Candidate Version 1.1 – 15 Dec 2015, Open Mobile Alliance, http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/generic-open-terminal-api-framework-1-1
\[DWAPI\]	Device WebAPI-PCH, Candidate Version 1.0 – 19 Apr 2016, Open Mobile Alliance, http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-device-webapi-v1-0
\[OCF\]	The Open Connectivity Foundation (http://openconnectivity.org)
\[IoTivity\]	The open source implementation of the OCF speciications (http://iotivity.org)
\[RVI\]	Remote Vehicle Interaction open source project (https://github.com/GENIVI/rvi_core)
\[VSS\]	Vehicle Signal Specifications (https://github.com/GENIVI/vehicle_signal_specification)

3. Terminology and Conventions

3.1 Conventions

This paper is an informative document, which is not intended to provide testable requirements to implementations.

3.2 Definitions

Term	Meaning

3.3 Abbreviations

Abbreviation	Meaning
DWAPI	Device WebAPI
GotAPI	Generic Open Terminal API
OMA	Open Mobile Alliance
OCF	Open Connectivity Foundation
RVI	Remote Vehicle Interaction
VSS	Vehicle Signal Specification

4. Introduction

We live in an increasingly automated world with complex interactions between the inventions we depend on, as well as those we merely enjoy using. The interactions (or interoperability) among devices depend largely on standardization efforts. However, to achieve interoperability between complex collections of machines such as cars, computers, appliances or even cities and homes deeper levels of agreed standards are needed. In the office setting, this has largely been resolved thanks to the widespread implementation of common software across most companies, such as Microsoft Windows and Microsoft Office. These products represent an implementation of a closed architectural design that works with itself smoothly. Apple has a similar approach where design, testing, and support all comes from a single source. However, even in this environment issues have appeared when one company or department uses one product and another company or department uses the other.

One example can be seen when using contact information and calendar reminders. At one time both Microsoft and Apple devised their own methods to store and share these small bits of information needed to synchronize between users in the enterprise. This allows people working in different countries and time zones to all see the dates in their own format and timezone, yet it is stored in a standard format for exchange. Enterprise users with mostly Microsoft Office, but a few Apple Macs, left the minority users in the cold until a standard agreed upon format was defined, developed, tested and implemented in the otherwise closed systems. The data format and API for this is now called vCard and vCal. Through its implementation, the enterprise users have the choice of device, operating system, and application. Now that tablets, phones, and even Linux PCs have been added to the enterprise, all they need to do is implement the standard. Thus BYOD or Bring Your Own Device was enabled and is now an enterprise IT expectation along with the implications of incorporating personal preferences by users to maximize productivity.

Recently the open source software model (OSS) has caught momentum as a way for hundreds of different companies to participate in the development of software design and code creation. This model not only includes a licensing philosophy that tries to remove costly barriers to participation but encourages the use of a common base of code. Beginning in 2006 or so, automotive software development began to use this method to help manage the millions of lines of code needed to create a modern vehicle. The total number of cars heading to production is divided by at least ten automakers, often with hundreds of combinations of makes, model years and trim levels resulting in relatively small numbers of validation resources for each variation. This issue is driving car development and bugs to record levels, resulting in higher prices and costly recalls after the cars are on the road. Add in the concept of self-driving cars and safety automation features and the costs can be measured in both currency and lives. The collaborative model helps with this but still requires standardization to be useful.

Interoperability standards in data formats and programming interfaces (APIs) in even more essential if non-automotive devices are attached to the "connected cars." Otherwise, consumers need to buy all their home appliances and electronic devices from the car dealer. This model ended with the "car phone" but continues in some way with the limited services available in closed telematics offers by each carmaker.

4.1 Observed Problem

Problem statement 1

- There are a series of initiatives in the automotive industry that will touch the mobile environment
- There is work going on in the mobile/IoT industry that will need to interoperate with automobiles
  - It is important for all not to waste resources by “reinventing the wheel.”
  - Use proper protocols that already exist and already deployed
- There are many one-to-one company level programs underway. There is NOT a conversation happening at the industry-to-industry level to ensure standards-based interoperability and reduce fragmentation of protocols across automotive interacting with IoT and mobile services
- The overall needs of the auto industry need to merge with mobile/IoT industry developments – harmonization of requirements and solutions between the mobile and the automobile environments. What are the problems the automotive industry are trying to solve?

Problem statement 2

- The auto industry wants to ensure openness and mobile device interoperability with the car telematics systems
- Automakers run the risk of becoming a “dumb device” with no control over user data
- MNOs and automakers are forced to the sidelines where consumer touch, data collection, and ownership are no longer future revenue sources
- Existing automotive approach of proprietary implementations is no longer working

Problem statement 3 - Vehicle Security

Vehicle Security is a very broad subject. Over the past 10 to 20 years the number of computers (ECUs) on the car and the amount of software running on these ECUs grew exponentially. Today’s cars come with tens of ECUs connected to several networks. Unfortunately, the vehicle and the network of ECUs it includes were not designed with security in mind and were not designed to be attached to the external world.

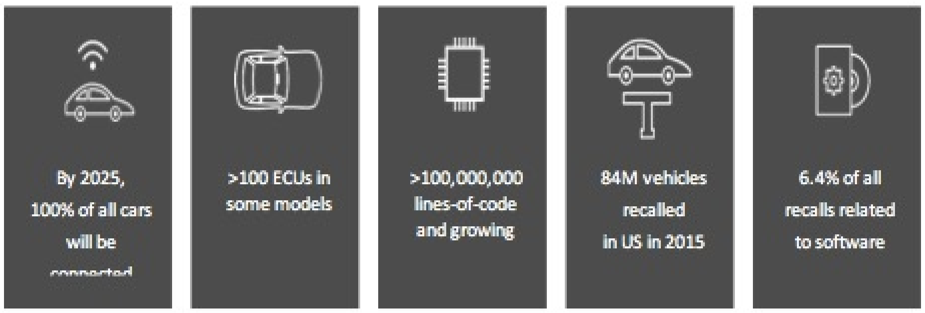

By 2025, 100% of all cars will be connected
>100 ECUs in some models
>100,000,000 lines-of-code and growing
84M vehicles recalled in US in 2015
6.4% of all recalls related to software

Figure 1- Numbers to illustrate the automotive software crisis scale

The introduction of the connected car puts the industry in a situation that is very similar to the introduction of the connected computer back in the late 90s. Connecting computers to the Internet back then required IT to build the infrastructure to protect the computers on the internal networks. Fortunately, most of this IT software could be leveraged to protect the communication of the car to the external world. However, protecting the communication between the car computers and between the hostile environment and the car computers is still a challenge.

The massive amounts of software on the car exposes many vulnerabilities that might be exploited by hostile agents to cause harm to the driver, to steal private information or to spy on the driver whereabouts. Software update Over-the-Air (OTA) is a mandatory element enabling the OEM to fix the software vulnerabilities without the need for an expensive recall. The connected car introduced a serious problem that OTA is solving taking advantage of the automotive connectivity.

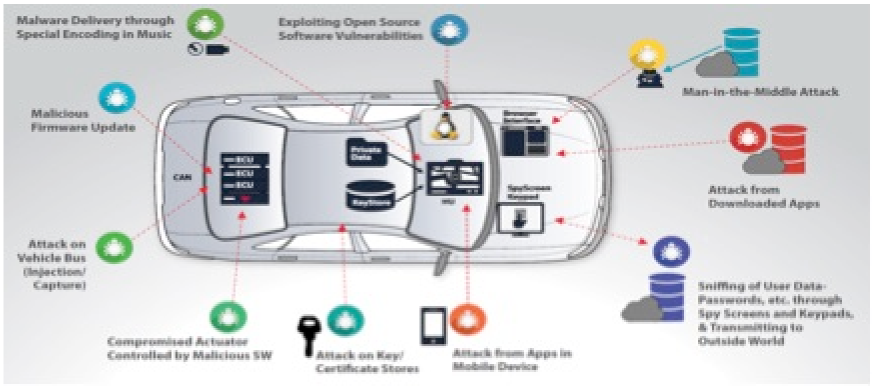

Malware Delivery through Special Encoding in Music
Exploiting Open Source Software Vulnerabilities
Man-in-the-Middle Attack
Malicious Firmware Update
CAN
ECU
ECU
ECU
Private Data
KeyStore
Attack from Downloaded Apps
Attack on Vehicle Bus (Injection/ Capture)
Sniffing of User Data-Passwords, etc. through Spy Screens and Keypads, & Transmitting to Outside World
Compromised Actuator Controlled by Malicious SW
Attack on Key/ Certificate Stores
Attack from Apps in Mobile Device

Figure 2- Some examples of the vulnerabilities of a modern car

<u>OTA Software Update Security</u>

When it comes to OTA software update the security goals of the end-to-end OTA system include the following aspects:

- Authentication
- Authorization
- Confidentiality
- Integrity

Authentication ensures which actors are allowed to work within the OTA system. It could be an admin authenticating to the OTA server, the car authenticating itself to the OTA server, or each software package being validated right before the installation. The authorization makes sure that every actor works within the policies dictated by the OTA system designer. Admin has role-based access, software for one ECU cannot install on a different ECU, etc.

Confidentiality requires that all software artifacts will be protected, so software and IP does not end up in the wrong hands and potentially exploited. Integrity verifies that every piece of software, in transit or at rest has not been maliciously or otherwise modified.

OMA DM is designed with such security design goals in mind. For the past decade, OMA DM has been successfully used on billions of mobile devices to deliver software updates over the air securely. Adopting the OMA DM protocol to provide OTA services for the automotive market is the right direction.

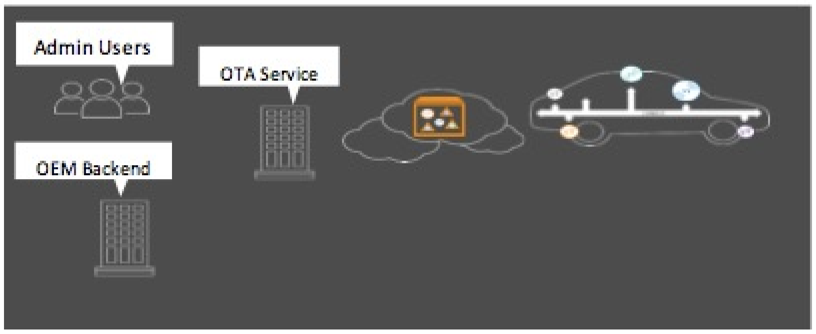


Figure 3- End to end OTA Software Update

4.2 Hot Projects in Open Automotive

- **SDL** - Smart Device Link, an open source implementation of Ford's AppLink (Ford Sync) software libraries enabling smartphone integration across all radio levels and brands of devices (Apple, Android, Windows, etc.)
- **RVI** - Remote Vehicle Interaction, an open source project by JLR Research Centre in Portland, OR. Enables universal connectivity to the vehicle bus transparently across LTE, Wi-Fi, Bluetooth, 3G, NFC and future networking technologies by embedding a standard software module in the car head unit
- **VSS** - Vehicle Data format standard emerging from work between W3C as implemented by RVI project
- **SOTA** - Software Over The Air (included in RVI above) through the entire vehicle (ASIL grade Safety and Security)
- **IOTivity** - Initially an IoT implementation of the OCF efforts, now emerging with automotive requirements influenced by RVI and VSS
- **OMA DM**, **LWM2M**, and **GotAPI** - 3 highly adopted standards for interoperability and device management developed by the wireless operators to ensure some level of compatibility and support of over 2 billion devices (phones, tablets, etc.) but not yet including cars in all cases

5. Automotive Technology Trends

5.1 Over-the-air updates

5.1.1 OMA DM (are all of Red Bend's contributions included?)

The OMA-DM protocol is deployed by leading Mobile Network Operators (MNOs) like AT&T, NTT DOCOMO, SoftBank, Sprint, T-Mobile US and Verizon Wireless. The MNOs use the OMA-DM protocol to manage mobile devices on their network. The following are some of the use cases MNOs have deployed:

- Provisioning: Initial device configuration (aka Bootstrap) and enabling and disabling device features

- Device Configuration: Modifications of various settings and parameters of the device

- Software Upgrades: Enable Over-the-air (OTA) software update including Firmware Update (FUMO) and system and applications components (SCOMO)

- Fault Management: Allow querying the device status and report device errors

The OMA-DM protocol is designed with interoperability and security in mind:

- Interoperability is achieved by the work of the OMA Interoperability (IOP) work group. The IOP work group specify and maintain the required processes, policies and test programs in addition to running regular test-fests where different vendors can attend and test their client and server software.

- Security is important because devices are exposed to software attacks. The OMA DM protocol was designed with security features like device authentication and challenges.

The OMA-DM protocol allows manufacturers and service providers to continuously increase the usefulness and value of connected devices throughout their entire lifecycle. They can optimize device performance, deliver unique insights, facilitate the rollout of innovative services and contribute to increased ARPU.

In recent years, with the introduction of the connected car, the term "mobile device" is extended to include the "large mobile device" of the automotive industry. Many car OEMs select to use the OMA-DM protocol to manage the configuration and software of the cars they are building. This enables car manufacturers to capitalize on the full business potential of connected cars by enhancing driver experience with rapid deployment of value-added in-car services, and minimizing costs through improved efficiency and a reduction in recalls.

5.2 Device APIs (move all API info to this section - Elizabeth)

5.2.1 Overview (Alan to edit)

needs to be made generic - not OMA-centric Through OMA enablers focused on device APIs, we can enable Automotive systems to offer API services to apps running in app environments provided by the Auto system. Such API services can enable apps to access vehicle information/services or other connected devices. These OMA enabler APIs are based on the Generic Open Terminal API (GotAPI) \[GotAPI\] enabler framework described in section 6.2.

5.2.2 Example Use Cases for OMA Device API utilized in the Auto Environment

OMA device APIs can enable synergy between app use cases in the Auto environment (e.g. driver safety, IoT, etc) and evolve Auto technologies. For example:

- An Auto system app can monitor driver vital signs (heartbeat, breathing rate, temperature, blood pressure, etc)
- A user can download an RVI app providing Mobile Keyless Entry to their smartphone, and use it to unlock the vehicle
- Auto system apps can also discover other external sensors, e.g. road sensors, outside air quality sensors, etc

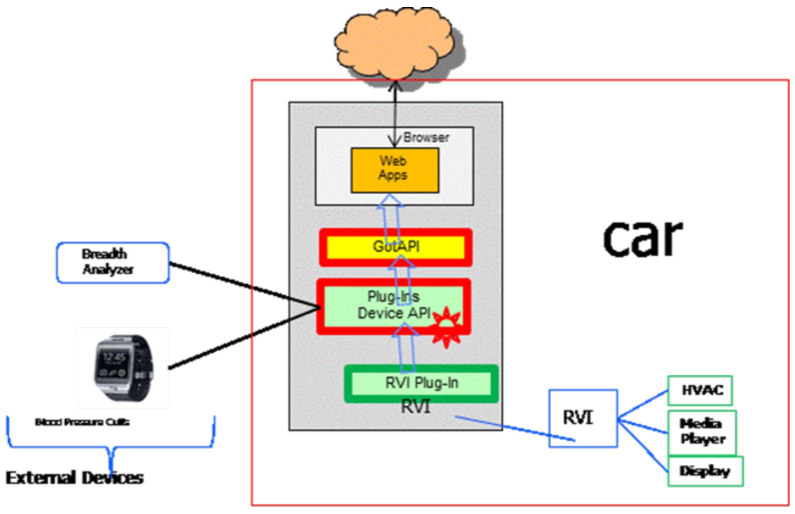
Through the two essential roles below, Auto systems (e.g. an RVI) that host a GotAPI-based API server can support a wide variety of use cases for apps running in the Auto system, e.g.

- the Auto system can offer vehicle information/services to apps running in an Auto Web runtime environment
- the Auto system can act as a GotAPI client in accessing information/services provided by connected devices

These use cases shown below:

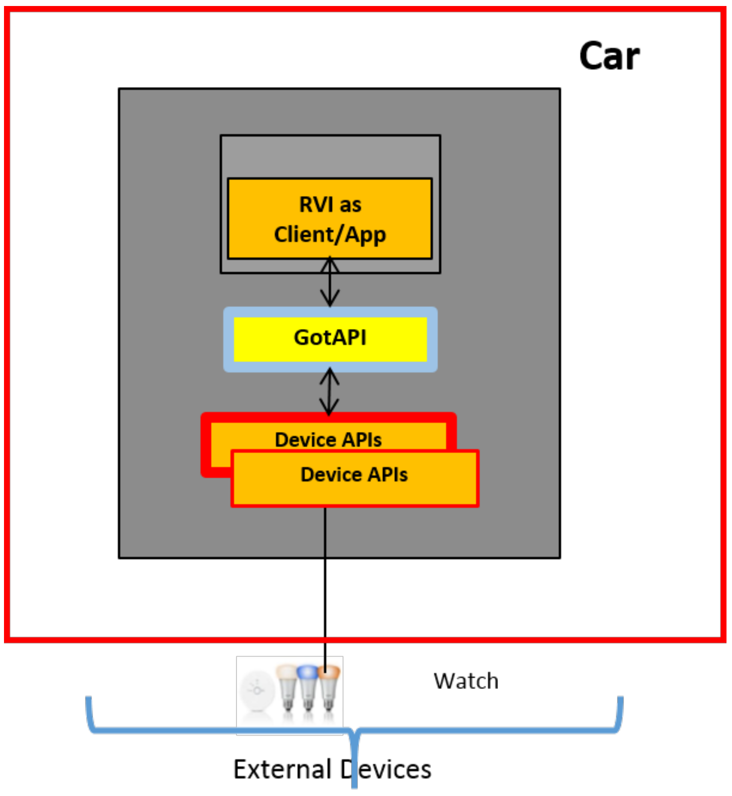
Use Case A: Web apps (e.g. an Auto Driving Lessons Coach) can access the RVI data and external device data through GotAPI/Device API. GotAPI enables the Web app to access RVI information/services without needing to implement the RVI-specific interfaces:

Use Case A



Use Case B: An RVI can access external device information/services, e.g. exposing access to a home lighting system.

Use Case B



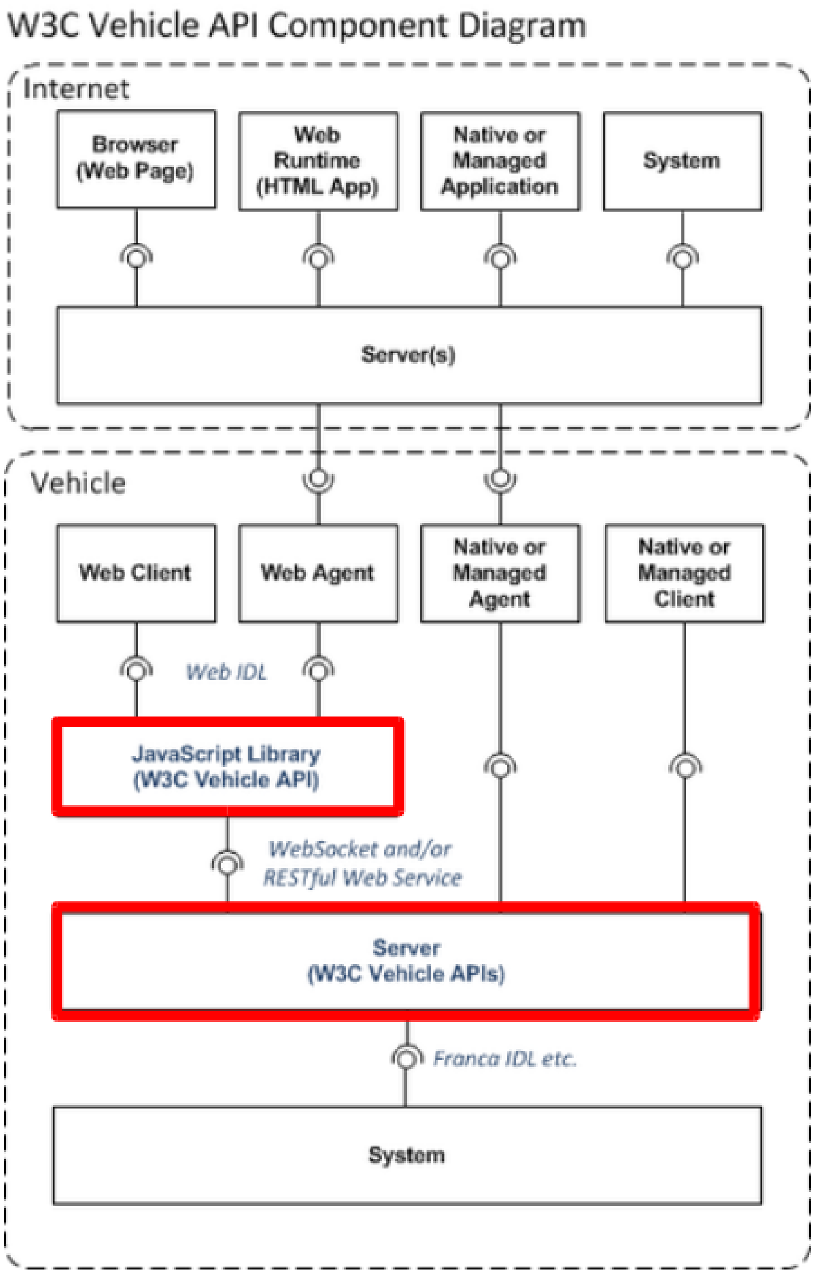
6. Automotive Industry API definition projects comparison

6.1 W3C Automotive Web API (Sanjeev-owner; Alan-editor)

6.1.1 Overview and objective of project

W3C first explored the impact of the Open Web Platform on the automotive industry at the November 2012 Web and Automotive Workshop. Participants discussed how location-based services, enhanced safety, entertainment and integration of social networking will benefit drivers and passengers. Since then, the Business and Working Groups have kept the conversation around standardization and specifications, through regular meetings with stakeholders. The Automotive Working Group develops APIs to expose vehicle data and information from automotive network bus(es) (eg. MOST and CAN). The specification consists of two parts:

1. *Vehicle Information Specification* containing durable, unchanging access methods for obtaining vehicle information;
2. *Vehicle Data Specification* to specify agreed upon standardized data elements as well as the method for extending data elements to OEM specific elements.



The W3C Automotive Specifications follows a client server architecture. The W3C automotive compliant web clients "connect" to a W3C Vehicle Server using secure websocket connection (wss://). The Vehicle Signal Server abstracts the communications with the underlying automotive networks (like CAN, MOST etc), and presents a simple set of Web API for the clients. Beyond this, the interactions between client and server follow a Request/Response model.

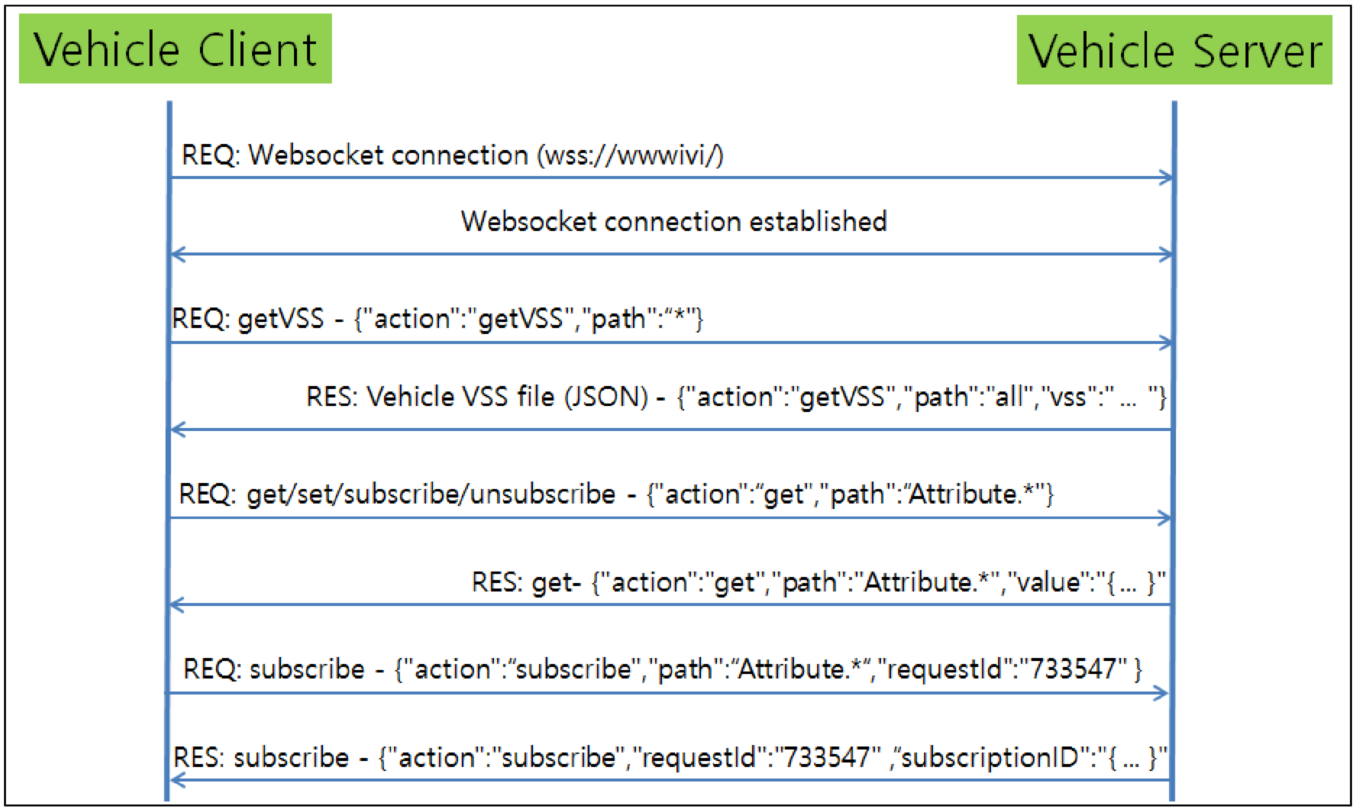
After the initial connection is established, the client is "Authorized" by the server.
W3C web clients create a vehicle object to access the API provided by the server, as follows.

```
var vehicle  =  new WebSocket("wss://wwwivi", "wvss1.0");
```

The "vehicle" object, represents an instance of the Vehicle Signal Server connection that can provide access to Vehicle Data through the W3C Information API. The supported methods are :

1. Authorize - Authorize a client to access the Vehicle API (TLS)
2. GET - Gets the value of a parameter, like body.mirrors.left
3. SET - Sets the value for a parameter
4. Subscribe - Subscribe to status changes
5. Unsubscribe
6. getVSS - get the JSON representation, to be used in GET/SET/SUBSCRIBE/UNSUBSCRIBE API.

The message format is JSON. Below is a simple flow diagram that explains the communication between W3C automotive clients and servers.



REQ: Websocket connection (wss://wwwivi/)

Websocket connection established

REQ: getVSS - {"action":"getVSS","path":"*"}

RES: Vehicle VSS file (JSON) - {"action":"getVSS","path":"all","vss":" ... "}

REQ: get/set/subscribe/unsubscribe - {"action":"get","path":"Attribute.*"}

RES: get- {"action":"get","path":"Attribute.*","value":"{ ... }"

REQ: subscribe - {"action":"subscribe","path":"Attribute.*","requestId":"733547" }

RES: subscribe - {"action":"subscribe","requestId":"733547" ,"subscriptionID":"{ ... }"

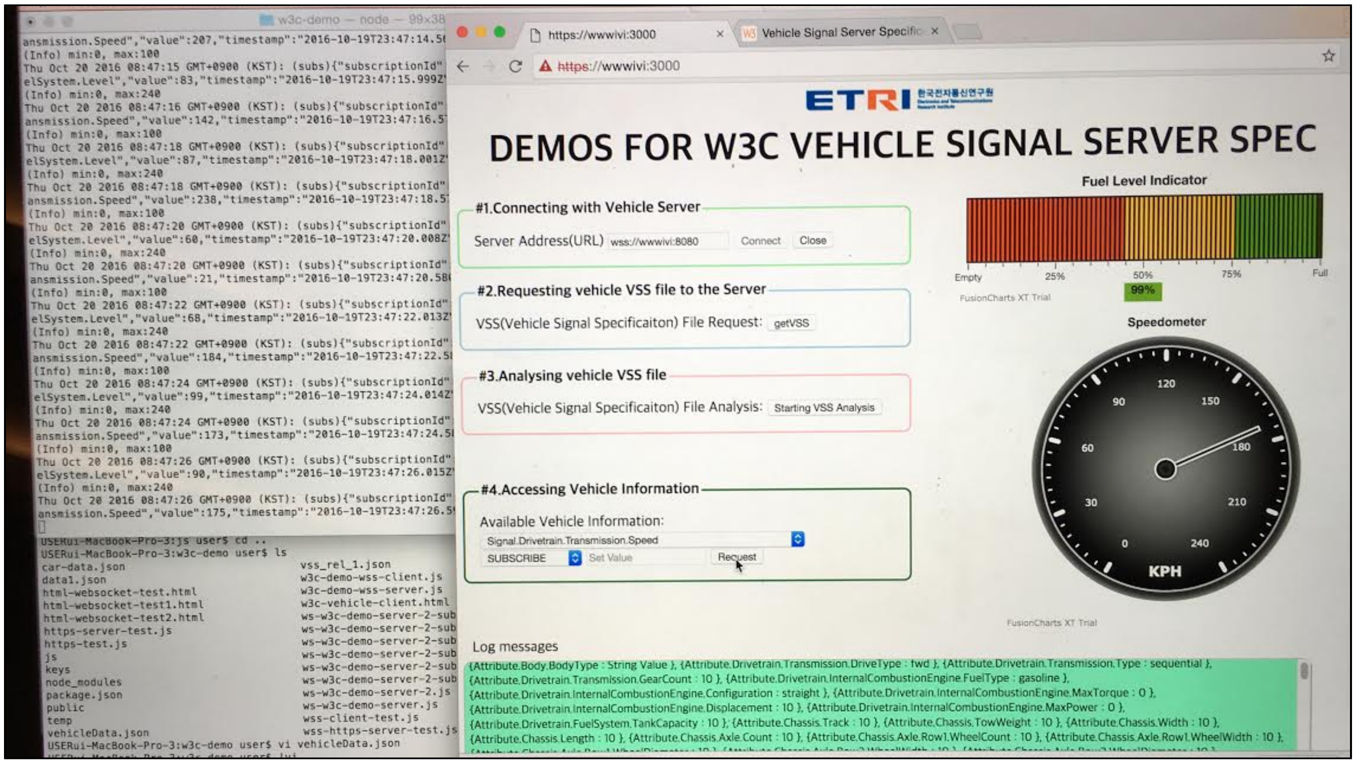
The getVSS API provides a JSON payload that represents the vehicle information accessible to clients. The authorization mechanism helps implement role based access to vehicle data.

Given below is an example of how a web client would invoke a "Subscribe" request to the vehicle server.

```
vehicle.send('{ "action": "subscribe", "path": "body.mirrors.left",
                "reqId": [some_unique_value] }');
```

This example, registers the client to be notified about changes to "body.mirrors.left" by the vehicle server. The client can process the changes via Javascript callbacks.

The W3C Automotive Business Group recently met at the Genivi AMM in San Franciso and showed their demo implementation of the above API. With less than 100 lines of Javascript, developers can easily integrate Vehicle data into their apps and services. Here's a screenshot of the demo.



```
w3c-demo — node — 99x38
ansmission.Speed","value":207,"timestamp":"2016-10-19T23:47:14.5
(Info) min:0, max:100
Thu Oct 20 2016 08:47:15 GMT+0900 (KST): (subs){"subscriptionId"
elSystem.Level","value":83,"timestamp":"2016-10-19T23:47:15.999Z"
(Info) min:0, max:240
Thu Oct 20 2016 08:47:16 GMT+0900 (KST): (subs){"subscriptionId"
ansmission.Speed","value":142,"timestamp":"2016-10-19T23:47:16.5
(Info) min:0, max:100
Thu Oct 20 2016 08:47:18 GMT+0900 (KST): (subs){"subscriptionId"
elSystem.Level","value":87,"timestamp":"2016-10-19T23:47:18.001Z"
(Info) min:0, max:240
Thu Oct 20 2016 08:47:18 GMT+0900 (KST): (subs){"subscriptionId"
ansmission.Speed","value":238,"timestamp":"2016-10-19T23:47:18.5
(Info) min:0, max:100
Thu Oct 20 2016 08:47:20 GMT+0900 (KST): (subs){"subscriptionId"
elSystem.Level","value":60,"timestamp":"2016-10-19T23:47:20.008Z"
(Info) min:0, max:240
Thu Oct 20 2016 08:47:20 GMT+0900 (KST): (subs){"subscriptionId"
ansmission.Speed","value":21,"timestamp":"2016-10-19T23:47:20.58
(Info) min:0, max:100
Thu Oct 20 2016 08:47:22 GMT+0900 (KST): (subs){"subscriptionId"
elSystem.Level","value":68,"timestamp":"2016-10-19T23:47:22.013Z"
(Info) min:0, max:240
Thu Oct 20 2016 08:47:22 GMT+0900 (KST): (subs){"subscriptionId"
ansmission.Speed","value":184,"timestamp":"2016-10-19T23:47:22.5
(Info) min:0, max:100
Thu Oct 20 2016 08:47:24 GMT+0900 (KST): (subs){"subscriptionId"
elSystem.Level","value":99,"timestamp":"2016-10-19T23:47:24.014Z"
(Info) min:0, max:240
Thu Oct 20 2016 08:47:24 GMT+0900 (KST): (subs){"subscriptionId"
ansmission.Speed","value":173,"timestamp":"2016-10-19T23:47:24.5
(Info) min:0, max:100
Thu Oct 20 2016 08:47:26 GMT+0900 (KST): (subs){"subscriptionId"
elSystem.Level","value":90,"timestamp":"2016-10-19T23:47:26.015Z"
(Info) min:0, max:240
Thu Oct 20 2016 08:47:26 GMT+0900 (KST): (subs){"subscriptionId"
ansmission.Speed","value":175,"timestamp":"2016-10-19T23:47:26.5
```

```
USERui-MacBook-Pro-3:js user$ cd ..
USERui-MacBook-Pro-3:w3c-demo user$ ls
car-data.json                          vss_rel_1.json
data1.json                             w3c-demo-wss-client.js
html-websocket-test.html               w3c-demo-wss-server.js
html-websocket-test1.html              w3c-vehicle-client.html
html-websocket-test2.html              ws-w3c-demo-server-2-sub
https-server-test.js                   ws-w3c-demo-server-2-sub
https-test.js                          ws-w3c-demo-server-2-sub
js                                     ws-w3c-demo-server-2-sub
keys                                   ws-w3c-demo-server-2-sub
node_modules                           ws-w3c-demo-server-2-sub
package.json                           ws-w3c-demo-server-2.js
public                                 ws-w3c-demo-server.js
temp                                   wss-client-test.js
vehicleData.json                       wss-https-server-test.js
USERui-MacBook-Pro-3:w3c-demo user$ vi vehicleData.json
```

https://wwwivi:3000

DEMOS FOR W3C VEHICLE SIGNAL SERVER SPEC

#1.Connecting with Vehicle Server

Server Address(URL) wss://wwwivi:8080

#2.Requesting vehicle VSS file to the Server

VSS(Vehicle Signal Specificaiton) File Request: getVSS

#3.Analysing vehicle VSS file

VSS(Vehicle Signal Specificaiton) File Analysis: Starting VSS Analysis

#4.Accessing Vehicle Information

Available Vehicle Information: Signal.Drivetrain.Transmission.Speed

SUBSCRIBE

Log messages

```
{Attribute.Body.BodyType : String Value }, {Attribute.Drivetrain.Transmission.DriveType : fwd }, {Attribute.Drivetrain.Transmission.Type : sequential },
{Attribute.Drivetrain.Transmission.GearCount : 10 }, {Attribute.Drivetrain.InternalCombustionEngine.FuelType : gasoline },
{Attribute.Drivetrain.InternalCombustionEngine.Configuration : straight }, {Attribute.Drivetrain.InternalCombustionEngine.MaxTorque : 0 },
{Attribute.Drivetrain.InternalCombustionEngine.Displacement : 10 }, {Attribute.Drivetrain.InternalCombustionEngine.MaxPower : 0 },
{Attribute.Drivetrain.FuelSystem.TankCapacity : 10 }, {Attribute.Chassis.Track : 10 }, {Attribute.Chassis.TowWeight : 10 }, {Attribute.Chassis.Width : 10 },
{Attribute.Chassis.Length : 10 }, {Attribute.Chassis.Axle.Count : 10 }, {Attribute.Chassis.Axle.Row1.WheelCount : 10 }, {Attribute.Chassis.Axle.Row1.WheelWidth : 10 },
```

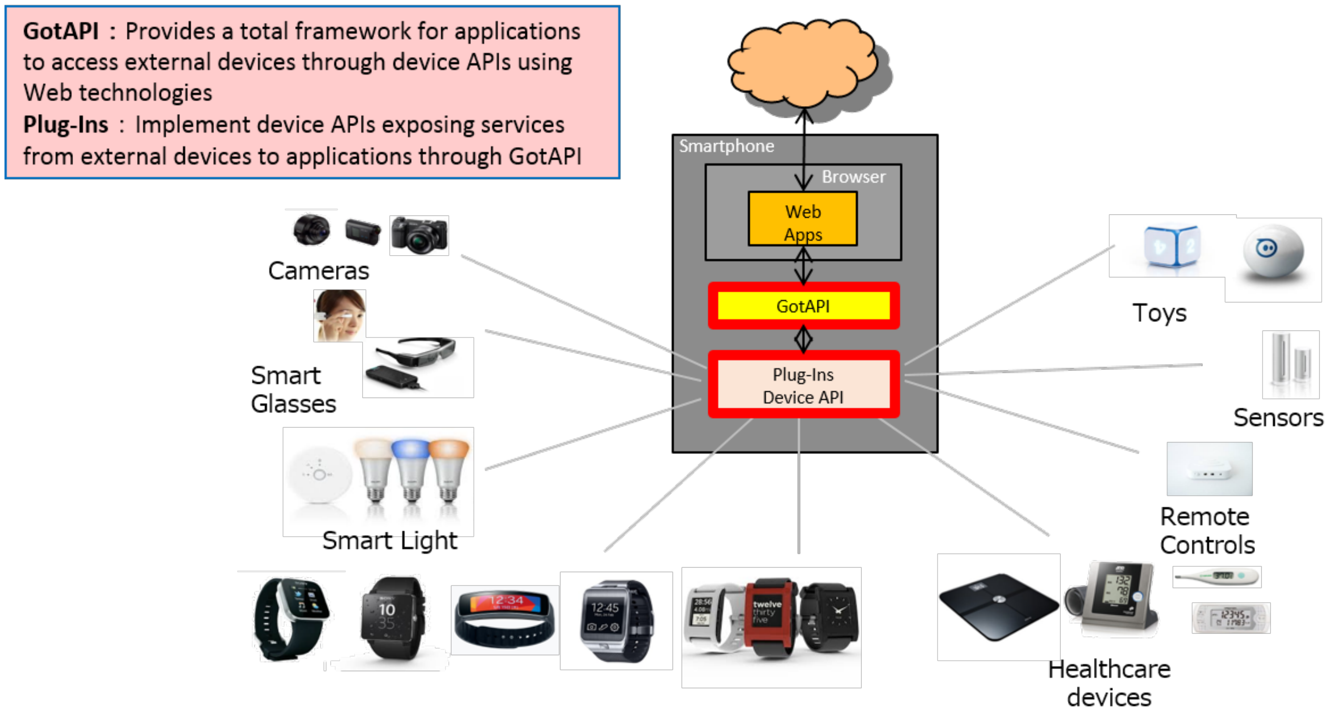
6.1.2 API's defined

6.2 OMA GotAPI and DWAPI

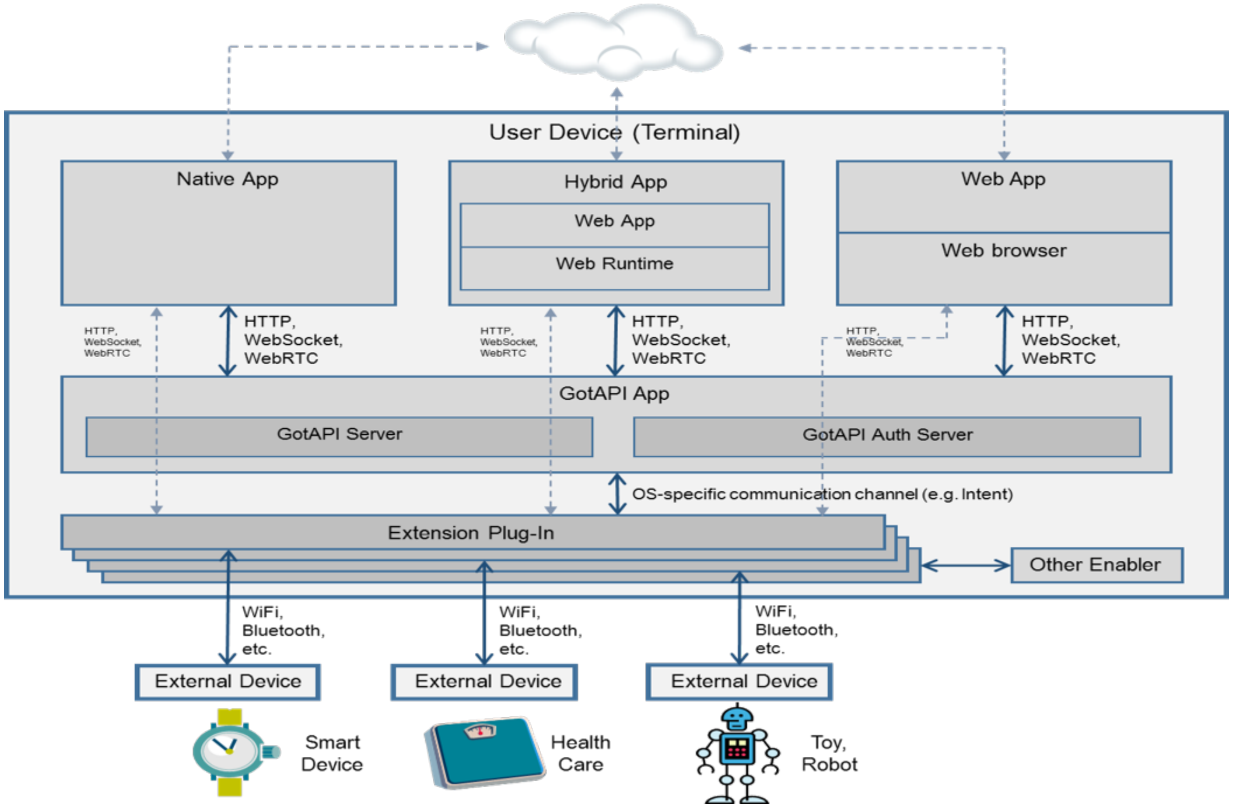
This section introduces the OMA Generic Open Terminal API Framework (GotAPI) v1.1 Enabler, and the GotAPI-based Device WebAPI (WAPI) v1.0 \[DWAPI\]. At a high level, GotAPI leverages widely supported Web technologies to enable a GotAPI server host (e.g. the Auto system) to provide an easy-to-integrate framework for apps to register and authenticate, discover available API services, access those API services, and secure the data exchanged between apps and other devices/services accessed through the GotAPI server. Without GotAPI, apps typically would have to be designed to support the specific methods through which other devices/services are accessed, e.g. a variety of APIs, protocols, and connection bearers.

DWAPI builds upon GotAPI in defining Web-based APIs used to access healthcare devices through IEEE-specified interfaces. Additional APIs such as for 3D printer access are in development at OMA.

Overview: Various External Devices Working with Apps Through GotAPI


GotAPI : Provides a total framework for applications to access external devices through device APIs using Web technologies
Plug-Ins : Implement device APIs exposing services from external devices to applications through GotAPI
Smartphone
Browser
Web Apps
GotAPI
Plug-Ins Device API
Cameras
Smart Glasses
Smart Light
Toys
Sensors
Remote Controls
Healthcare devices

GotAPI Architecture



GotAPI defines how apps and the GotAPI enabler components interact through the interfaces illustrated below:

- GotAPI-1 enables applications to make API requests and receive responses. This interface is generically specified by GotAPI, as GotAPI-based API specifications will define specific request/response transactions that can be utilized in host devices based on the available interface technologies, payload protocols, and their appropriate design patterns. These options include:
  - The interface technologies TLS 1.2, HTTP/1.1, HTTP/2, Server-Sent Events
  - The design patterns REST and JSON, such as JSON-RPC
  - A Temporary Server Feed (TSF) mechanism for binary data responses and triggering different protocols
  - The initiation of the asynchronous messaging interface, GotAPI-5, to use WebSocket

- GotAPI-2 enables applications to obtain authorization for access to GotAPI-based APIs. This interface is fully specified by GotAPI, being a standard (though optionally used) support function for all GotAPI-based APIs. GotAPI-2 supports bindings and request/response transactions that can be utilized in host devices based on the available interface technologies. These options include the interface technologies TLS 1.2, HTTP/1.1, HTTP/2, and URI scheme handling. The GotAPI-2 interface relies upon the concepts of OAuth, though with different semantics as necessary for adaptation to the available interface technologies.

- GotAPI-3 enables the remote provisioning of API access authorizations through a policy management function, which may include one or more of:
    - OMA Device Management, using a Managed Object (MO) to be defined in a future version of the GotAPI enabler
    - An implementation-specific policy management service

- GotAPI-4 enables Extension Plug-Ins for external devices and internal enablers through which they communicate with the GotAPI Server. Note that host-device-internal enablers/applications may also be connected to GotAPI servers directly in implementation specific ways without using the GotAPI-4 interface and Extension Plug-Ins.

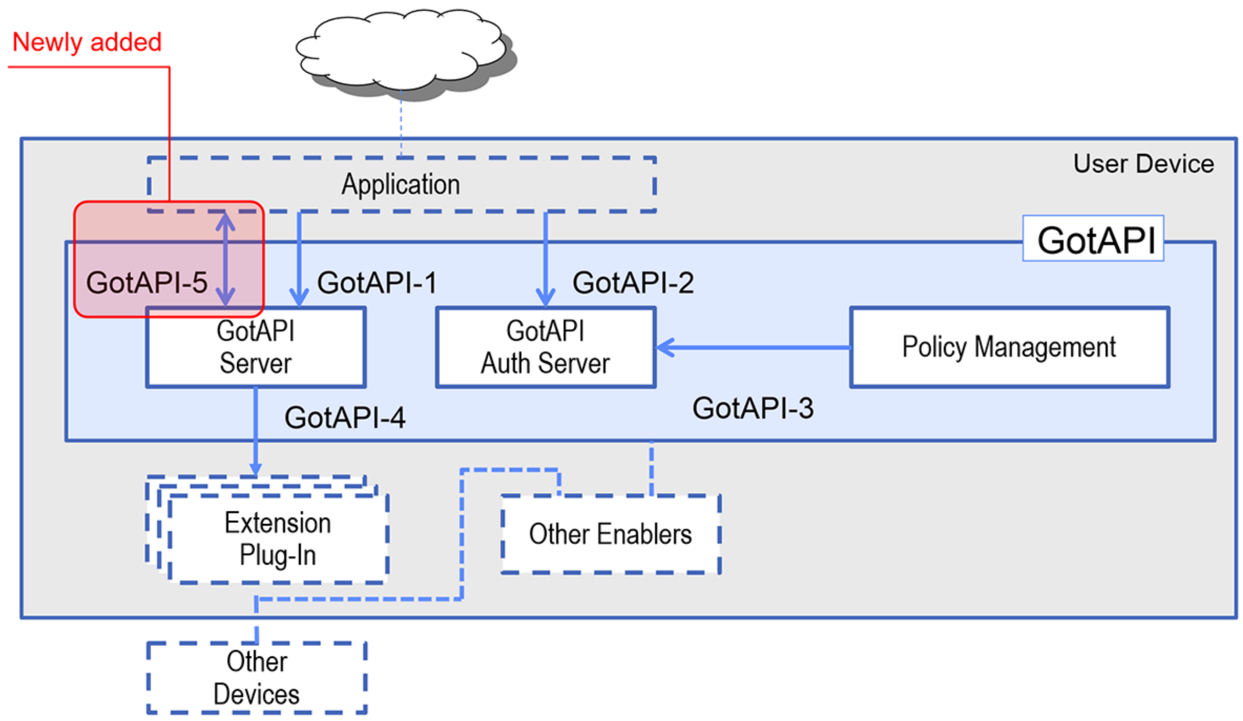
  The Extension Plug-Ins are independent applications. They are the mediators between the GotAPI Server, and external devices and internal enablers/applications. Typically, there are expected to be multiple Extension Plug-In applications installed on a device by the user or preinstalled on the device.

  The GotAPI-4 interface provides the following functions concerning Extension Plug-Ins:

    1. **Plug-In Discovery**: GotAPI-4 Plug-In Discovery enables the GotAPI Server to discover the targeted Extension Plug-In which an application wants to access and communicate.
    2. **Service Discovery**: GotAPI-4 Service Discovery enables the GotAPI Server to find all the services provided by an Extension Plug-In. In this context, the "service" means an external device or a function provided by an internal enabler through an Extension Plug-In. The Service Discovery provides not only the list of services but also the availability of each service at the time.
    3. **Approval**: GotAPI-4 Approval is the function to ensure security, especially to protect users' data and privacy from unwanted exploits, so that the users can safely use the application with external devices and enablers that connect via Extension Plug-Ins.
    4. **Data Forwarding**: GotAPI-4 Data Forwarding is the function that enables an application to communicate with the targeted Extension Plug-In through the GotAPI Server. Data Forwarding takes place after Plug-In Discovery (optional), and Approval processes have been successfully completed. GotAPI-4 Data Forwarding uses the “pass-through” mechanism so that the application can access and communicate with the APIs that (i) are implemented in the Extension Plug-In and (ii) expose the features of the external devices or internal enablers.

- GotAPI-5: The GotAPI-5 interface enables applications to listen to asynchronous messages from Extension-Plug-Ins via the GotAPI Server using WebSockets.

GotAPI Interfaces



Transport Protocols

6.3 LWM2M

6.3.1 Overview

The OMA LWM2M (Light Weight Machine To Machine) protocol provides the capability for applications to communicate and manage IoT devices. LWM2M is based on the IETF Constrained Application Protocol (CoAP) providing communication between a LWM2M Server and a LWM2M Client (where Client is located in a constrained IoT device).

6.3.2 Key features

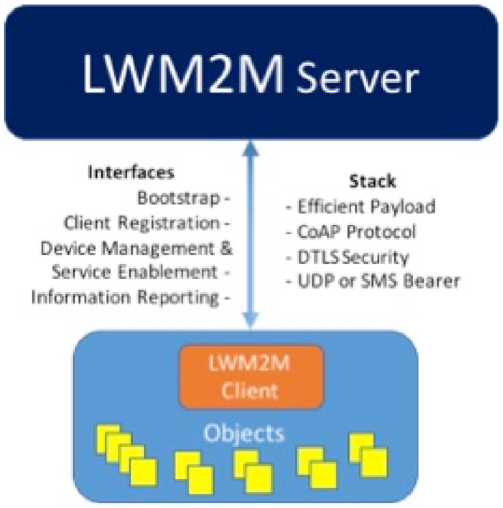
Some of the abilities which are helpful from connected car perspective from LWM2M are the following:

- Low data byte transmission with allowance for client to sleep (i.e., non-continuous communication)
- Suitable for constrained devices (hence reduced footprint for reduced CAPEX for car manufacturers)
- Several objects can be reported in one message using simple content types like TLV, JSON and single objects with simple text content type
- Ability to support a range of solutions through pre-defined objects (connected car objects/resources)
- Client (connected car) could send notifications to server based on defined trigger events e.g. periodic reporting, reporting upon a change of value, reporting based on thresholds reached. This is key as it gets past the poll and notification model
- LWM2M by default works with 3GPP technologies hence easier integration with MNOs

6.3.3 Core Interfaces

The core interfaces between the server and the client are categorized into:

- Bootstrap
- Client Registration
- Device management and service enablement
- Information Reporting



6.3.4 Functionality

OMA LWM2M is unique in the aspect that it converges both management and application control functionality within one communication session allowing for efficient handling of IoT devices. Since OMA LWM2M protocol is based on IETF CoAP, the OMA LWM2M protocol allows different transport bindings (e.g., UDP, SMS) and is secured using IETF DTLS protocol.

The device management features defined by OMA for release 1.0 of LWM2M are:

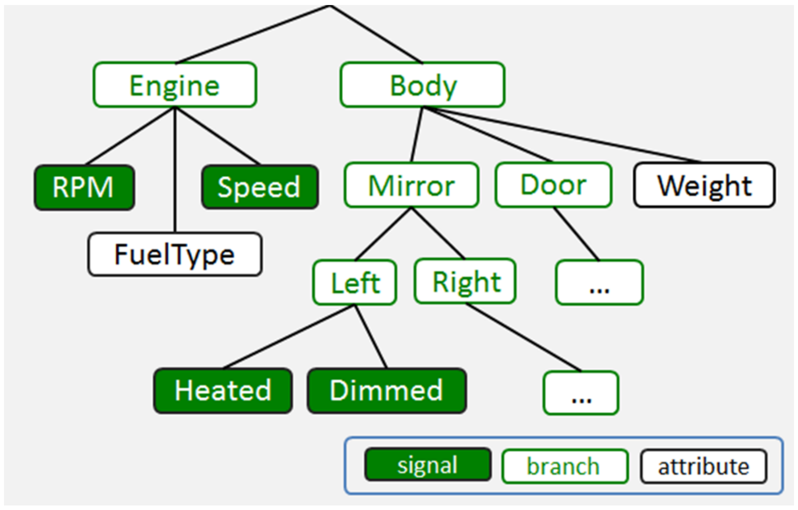
- Access control on the specific contents that could be handled remotely by different entities
- Software and firmware Management for applications inside the Client
- Lock & Wipe of the device from misuse
- Connection management for choosing various radio methods by the Client
- Device Capability Management to identify the capabilities existing in the device
- Location of the device
- Connection Statistics concerning communication characteristics over the air

6.3.5 Object Registry

The object registry provides a unique way of identifying the necessary and relevant objects. The object registry is maintained by OMNA (Open Mobile Naming Authority). It includes categories for interleaving 3rd party management objects and application objects into the OMNA system from vendors and other standards organizations (for, e.g., IPSO Alliance and oneM2M)

6.5 W3C VSS Standardization

A result from the RVI project is being developed within W3C regarding consistent naming of vehicle endpoints that could abstract the OEM specific software from third party applications remotely or locally accessing them, as illustrated below:



Initially, VSS is supported by open source projects under development in Jaguar Land-Rover's OSTC in Portland, OR and quickly being accepted by others including some telematics system suppliers.

6.6 OCF IoTivity

6.6.1 Overview and purpose of project

The Open Connectivity Foundation

The Open Connectivity Foundation (OCF), founded in 2014, is a global consortium of more than 200 member companies, focused on creating a single open specification to help assure secure interoperability between devices for consumers, business, and industry. The OCF has three key deliverables:

a. An Open Specification: URL

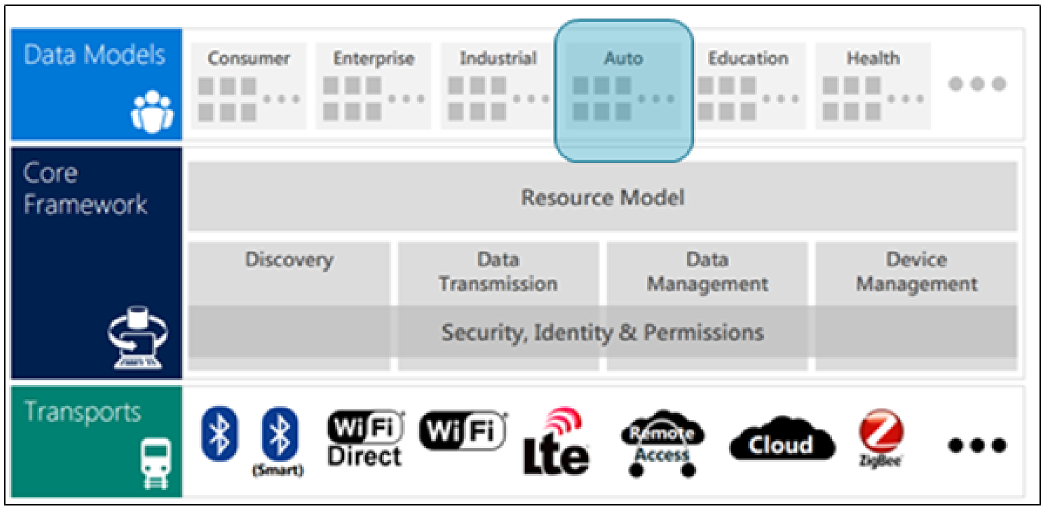
b. An Open-Source Reference Implementation of the OCF Specifications: IoTivity

c. A vendor-neutral certification program: OCF certification

6.6.2 IoTivity

The IoTivity project was created to bring together the open source community to accelerate the development of the framework and services required to connect these billions of devices. The IoTivity project is sponsored by the Open Connectivity Foundation (OCF), a group of industry leaders who will be developing a standard specification and certification program to address these challenges. IoTivity will deliver an **open source reference implementation of the OCF standard specifications**, yet will not be limited to those requirements. IoTivity is licensed under Apache 2.0 and follows a well-defined open source development model and release cycle.

IoTivity has already been ported to a multitude of OS platforms including Linux (ex. Ubuntu), Android and Tizen and it has been verified on multiple developer reference hardware devices like Arduino, Raspberry Pi, smartphones (Android and Tizen), wearable devices (like Gear S2). IoTivity supports multiple programming languages like Java, C/C++, Javascript (nodejs) etc. For resource constrained devices running on low power MCUs, IoTivity has an alternative implementation of the core OCF specifications known as “iotivity-constrained”.



IoTivity Architecture

The IoTivity architecture can be classified into three layers.

1. The Transport Abstraction layer abstracts the wired/wireless connectivity details for different hardware and software platforms and presents a unified API to enable secure Device-To-Device communications between OCF devices in a transport agnostic way.
2. The core framework implements the core OCF protocol as defined in the OCF specifications and handles essential functions like security, device discovery, data transmission and management and device provisioning.
3. The data models help represent the individual devices and their capabilities as resources, as defined in the OCF specifications, that can be securely controlled other OCF devices in the network.

An elaborate description of OCF and IoTivity is outside the context of this report. Please refer to the references section for more details.

6.6.3 The OCF Automotive Project

The automotive domain has been one of the key areas of focus for OCF since the beginning and as consumers start expecting their “connected vehicle” to be part of a seamless experience that depends on how the vehicle operates in a smart ecosystem involving a smart home, intelligent infrastructure and even with other vehicles.

In the words of Bill Ford (https://www.ted.com/talks/bill_ford_a_future_beyond_traffic_gridlock)

“We are going to build smart cars, but we also need to build smart roads, smart parking, smart public transportation systems and more... We need an integrated system that uses real-time data to optimize personal mobility on a massive scale without hassle or compromises for travelers.”

Optimizing personal mobility is no longer with the connected vehicle alone. To unlock these opportunities, OCF launched the Automotive Project in August 2016. For an automotive IoT ecosystem to flourish, the deployment architecture should be flexible enough for the key stakeholders to bring in their value proposition and make it easy for consumption by the rest of the ecosystem. Service providers hosting the infrastructure expect minimal changes to the existing investments and expect new revenue streams by extending their services into the IoT network. Customers expect new services and experiences with zero setup overhead. Application developers are looking to leverage their existing assets and generate more user engagement. The aim of the Project is to provide the technology, standards and collaboration needed to ensure secure interoperability between automotive and other verticals, starting with consumer electronics, smart home and healthcare. The initial use cases enabled by the project fall into the following broad categories.

- *Home Energy Management*
- *Security System Interaction*
- *Vehicle Location*
- *Smart Home device status*
- *Vehicle Control from Smart Device*
- *Smart Device Control from Vehicle*
- *Smart Mobility service integration*

The Project is led by members from Samsung, Honeywell, Cisco, SmartThings, ETRI, GRL and Tinnos. The project will also closely collaborate with other OCF working groups, leading automotive companies and alliances, open source projects and standards bodies. The OCF Automotive Project will define the data models. Additionally, the group will drive the certification requirements for compliant bridging implementations, which is essential for realizing valuable and commercially attractive use cases, to the automotive industry.

6.6.4 Initial Prototyping

OCF Automotive Project has been under incubation for the past six months. The incubation was carried out together with the Genivi Alliance under the Genivi Incubator projects initiative. The goals for this incubation involved realizing the use mentioned above cases using IoTivity open source project and the RVI open source implementation provided by Genivi.

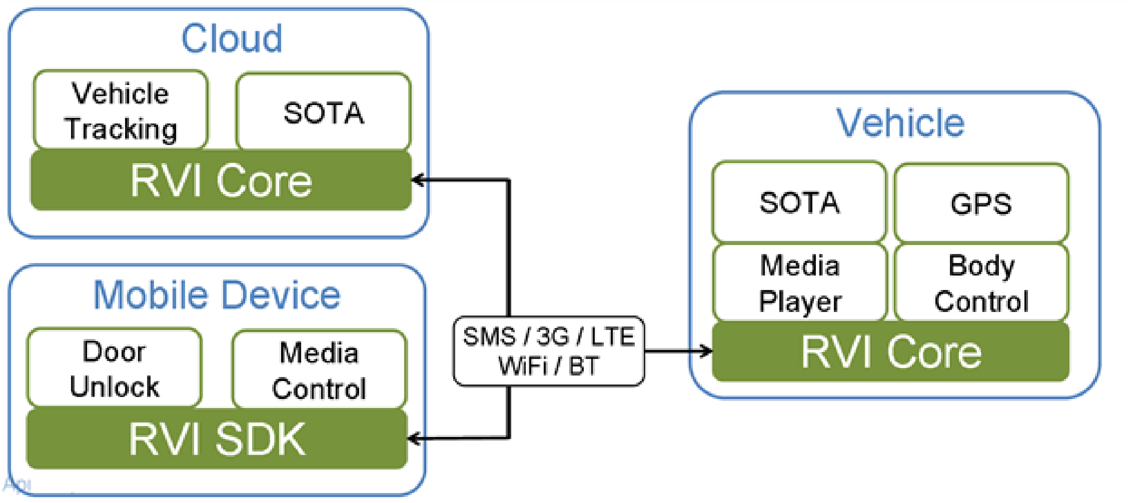
The RVI or Remote Vehicle Interaction project provides a standardized means for communications between the vehicle and its remote services over many different protocols. RVI separates the vehicle data representation from the transport. The "Vehicle Signal Specification" is a straightforward and flexible format to represent vehicle signals using a tree-like hierarchy. RVI Core stack handles the software infrastructure needed to discover other RVI endpoints and end to end delivery of messages, i.e. the transport agnostic communications infrastructure. RVI is capable of handling message delivery over Bluetooth, WiFi, LTE and SMS.

6.6.5 Remote Vehicle Interaction Overview

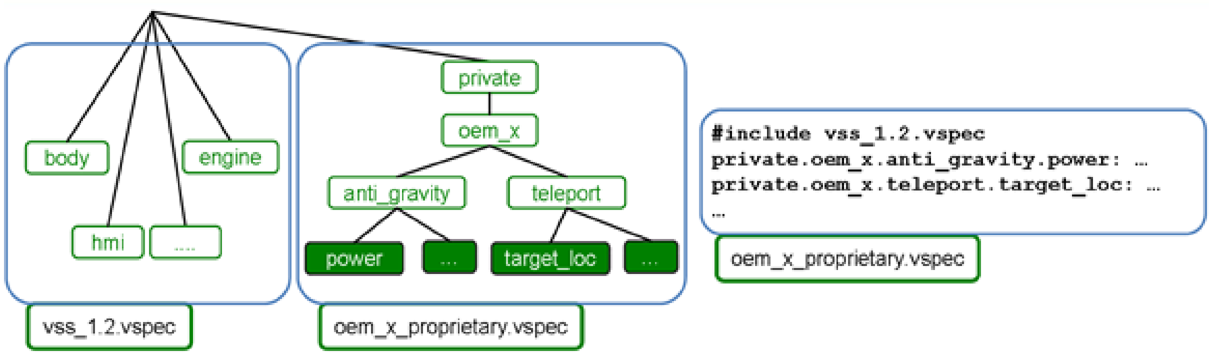
The RVI or Remote Vehicle Interaction project provides a standardized means for communications between the vehicle and its remote services over many different protocols. RVI separates the vehicle data representation from the transport. The "Vehicle Signal Specification" is a straightforward and flexible format to represent vehicle signals using a tree-like hierarchy. RVI Core stack handles the software infrastructure needed to discover other RVI endpoints and end to end delivery of messages, i.e. the transport agnostic communications infrastructure. RVI is capable of handling message delivery over Bluetooth, WiFi, LTE and SMS.

RVI project also publishes the “VEHICLE SIGNAL SPECIFICATION” via GitHub.

This repository specifies a set of vehicle signals that can be used in automotive applications to communicate the state of various vehicle systems. A standardized vehicle signal specification allows for an industry actor to use a common naming space for communicating vehicle state and, ultimately, permits the decoupling of the IVI stack from the underlying vehicle electrical architecture. The collection of signal specifications, or simply signals, is vendor independent. Vendor-specific extensions can be specified in a dedicated and uncontrolled branch of the signal specification tree.



This repository defines a set of vehicle signals that can be used in automotive applications to communicate the state of various vehicle systems. A standardized vehicle signal specification allows for an industry actor to use a common naming space for communicating vehicle state and, ultimately, permits the decoupling of the IVI stack from the underlying vehicle electrical architecture. The collection of signal specifications, or simply signals, is vendor independent. Vendor-specific extensions can be specified in a dedicated and uncontrolled branch of the signal specification tree.



```
#include vss_1.2.vspec
private.oem_x.anti_gravity.power: …
private.oem_x.teleport.target_loc: …
…
```

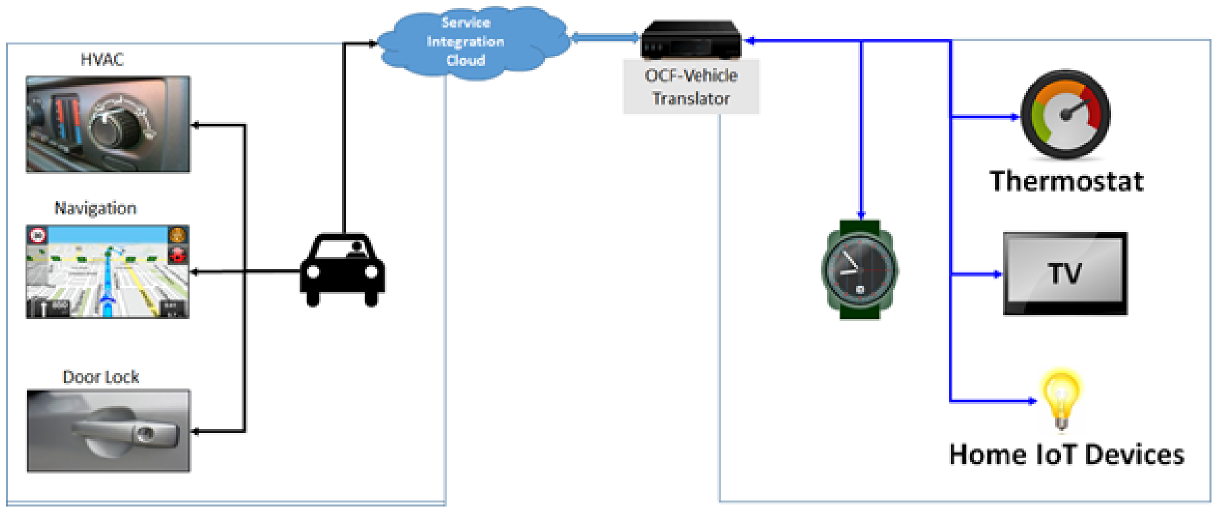
- A proprietary signal specification can use the GENIVI VSS as a starting point
- Can be used in production project to integrate with vendors
- Mature private extensions can be submitted for VSS inclusion

An open source RVI implementation combined with an open data format (VSS) to represent vehicle state, together provide the necessary software components that can provide secure connectivity to select functions of a vehicle, in a customizable manner. RVI enables a simple way to combine IoT devices and vehicle states to implement new services that leverage the capabilities of these devices.

6.6.6 Developing the OCF-Vehicle Translator

A MEAN (Mongo-Express-Angular-Node) stack based gateway framework (WSI – Web Service Interface) for bringing multiple services and IoT devices together as part of a single programmable structure, was already available in the context of the IoTivity open source project. We defined an abstract service metadata model as a way for services to be represented within an IoT network and to move the execution of the business logic from the cloud to inside the IoT network or automobile depending on the desired deployment configuration. This approach provides the flexibility to choose the nexus of device-service integration at the hands of the developers.

Thinking about the connectivity to the vehicle as a service, with the help of RVI, helped us model the use cases regarding discovering and invoking services provided by an RVI instance. This made the task of realizing the use cases to be simply a matter of service invocation rather than dealing with the protocol or messaging details. This reduces the burden for app developers to a large extent.



The RVI service node hosted in the vehicle HMI running on GDP connects via the cloud to the home gateway. The RVI-OCF translator/gateway also runs an RVI node. RVI provides many means to access services, and we decided to use WebSockets. The RVI-OCF translator/gateway connects via WebSockets to the local RVI node which then talks to the remote RVI node. Connection & remote RVI service discovery is handled purely by RVI and the RVI-OCF translator/gateway.

After connecting to the remote RVI node, the WSI Gateway just translates between the service definitions provided by Remote RVI Service Node into corresponding OCF device control commands based on application developer logic. This mapping is entirely independent of how the gateway functions and thus provides maximum flexibility to implement any rules or service logic you can come up with, depending on the scenario. In this case, the service was controlling the HVAC parameters of a vehicle. In future, we could extend the service to be whatever the situation demands. Another major benefit of this approach is that the IoT device data is never transmitted out of the home. Similarly, the vehicular information is never sent to the cloud. This provides a clear separation of the data across the domains, namely smart home and vehicle with a reduced security and privacy risk – while giving total control over the customer data to the existing services. In addition to the above demos and use cases, we also have made OCF stack called as IoTivity, as part of the GDP and AGL (Automotive Grade Linux), so that interested developers will now be able to implement use cases.

The OCF-Vehicle Translator is a software component that can be placed in

1. Cloud
2. Home Gateway
3. Vehicle IVI unit

Depending on the desired end user experience and deployment configuration, the translator enables seamless service oriented access between IoT devices, web services (apps) as well as vehicle controls.

6.6.9 Next Steps

OCF has the vision to connect the next 25 billion things

- Automakers want simple interconnection with smart devices.
- Developers want apps to work across car brands and ecosystems.
- End users want secure connectivity and services across devices.

The goal of OCF is to help create an interoperable specification for automotive IVI domain; that enables connected vehicle use cases. Over the next few months, Genivi and OCF will be working together with the **W3C Automotive Working Group**, to develop OCF data models for automotive, which can be used by OCF-Vehicle translator implementations, web service integrators or home gateway developers to build products and services that help bring to market, the connected vehicle – IoT crossover scenarios to life.

6.6.8 Related Links

A complete video demonstration is available here. https://www.youtube.com/watch?v=351m-GrRSNE

Source code is published under https://github.com/GENIVI/meta-genivi-ocf-demo/tree/master and http://git.s-osg.org/ocf-automotive-sampleapps/

The complete blog is presented here. https://blogs.s-osg.org/osg-ocf-automotive-fortnight/

7.0 Analysis and recommendations to achieve interoperability

This paper tells a story about the desire for both automotive and consumer electronics to find improved way to interact and provide the compelling experience to buyers of cars and "things" that will motivate them to invest their money in new models that work together better. Gone are the days of expecting customers to buy every component of a system as all one brand as many did when buying a stereo system for their home in the 70s.

A matched set of your phone, tablet, PC, car, home lighting, thermostat, and security system is highly unlikely. Consumers are already struggling with the subtleties of Android and Apple devices with different interactions based on closed architecture. Automakers are resisting the entry of these private consumer models into their products, yet are happy to adapt cross-device standards as shown by Toyota and Ford supporting SD-Link. An independent ecosystem of software developers is a bold trend in automotive. The pace of acquisition and respect for startups by car companies continues to increase. Open source projects such as the GENIVI IVI baseline, Linux Foundation AGL Unified Code Base, as well as the RVI and IoTivity projects provide evidence that open platforms in automotive are changing the car development process dramatically.

The need for IoT standards is clearly needed. While these standards emerge, both industries need to learn from past efforts. There is some overlap between GotAPI, accepted in the communications space, and OCF's IoTivity, which is not yet proven but could become a standard from the device makers. Similarly, OMA DM has an extensive implementation portfolio with communications providers and overlaps some with GENIVI's RVI implementation while RVI promises to incorporate OMA DM to include carrier support of LTE. Intensifying these efforts will help align and reuse the work already done.

W3C has the responsibility to further the work of VSS, regardless of the other projects and there appears to be no competition for this automotive specification, and the W3C Automotive Working Group needs to increase emphasis on this aspect of the RVI related work they are undertaking currently. Today a small handful of engineers and technical leaders are working on these projects, this investment by suppliers must increase. Automaker endorsement and intention to build product will bring more experts to the table.

In all cases these projects require more automaker support to succeed. The demonstration efforts by OCF and GENIVI are helpful, and also deserve more interaction between OMA, W3C, OCF to architect as a fully documented standard. SAE must be a part of this group effort. SAE has strong alignment with the auto suppliers and OEMs. This heritage recently surged by substantial industry excitement around Automated Driving Systems (ADS). No longer referred to as autonomous driving per SAE J3016 SEP2016, these ADS modules will not be functional as separate units but will need significant external information from other "things" both inside and outside the car to achieve higher levels of driving automation (Level 5).

Rather than constrain the need for standard interfaces, APIs and data formats to convenience features (operating your garage door) a clear standard will find its way into many interactive automotive and consumer systems. Safety related priorities must also align with the work of consortia focused on automotive production development.

Unquestionably, security is a key factor and methodologies under development for protecting and securing private and safety critical information should be the topic of study for another paper in this series.

Similar to the integration challenges experience in enterprise office systems, have been reduced by standard data formats and API's, such as vCard and vCal. Automotive leaders and communications leaders working together can deliver on the promise of open, safe, secure and smart cars that work with all your other smart "things".

8. References (Temporary)

1. **http://www.volkswagenag.com/content/vwcorp/info_center/en/news/2016/07/connected_car.html**
2. **http://fortune.com/2016/07/13/jaguar-land-rover-autonomous-cars-britain/**