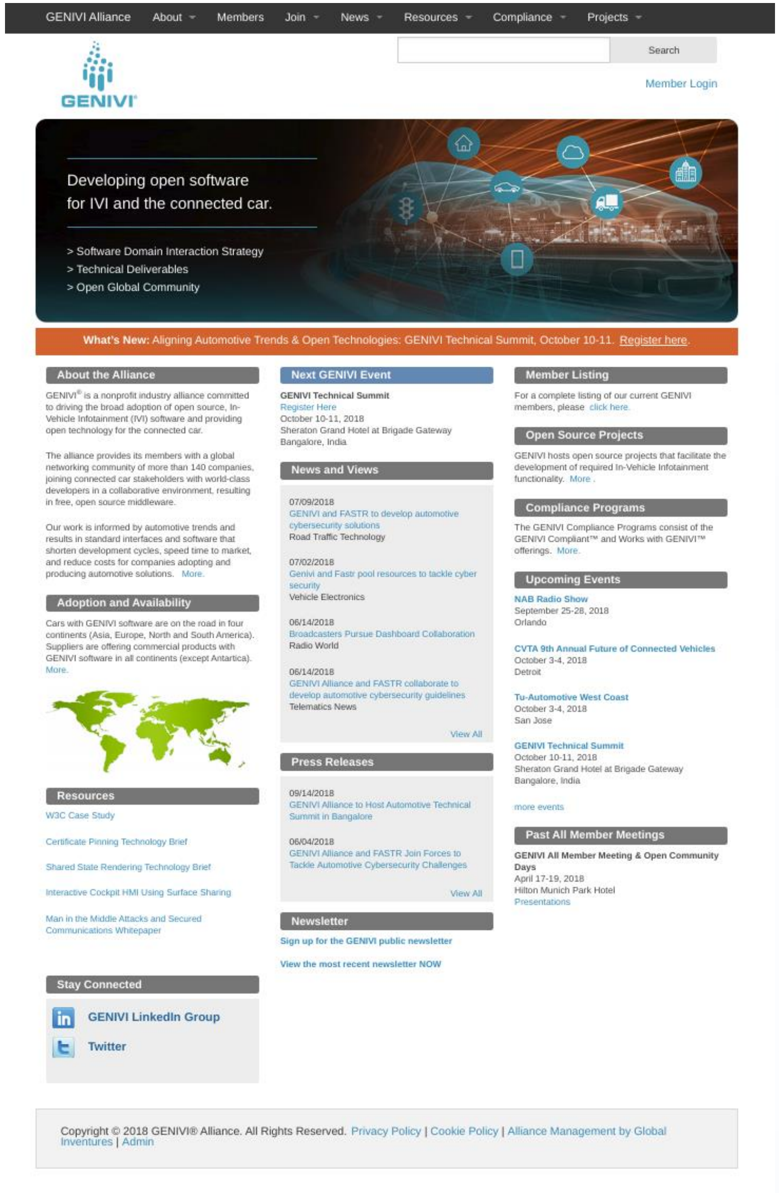
Bringing the Car to the Internet

October 10, 2018 | GENIVI and W3C – Enabling the Connected Car through Collaboration

Rudolf J Streif

Networking Expert Group Lead, GENIVI Alliance

Automotive Leadership meets Web Innovation



Developing open software
for IVI and the connected car.

> Software Domain Interaction Strategy
> Technical Deliverables
> Open Global Community

What's New: Aligning Automotive Trends & Open Technologies: GENIVI Technical Summit, October 10-11. Register here.

About the Alliance

GENIVI® is a nonprofit industry alliance committed to driving the broad adoption of open source, In-Vehicle Infotainment (IVI) software and providing open technology for the connected car.

The alliance provides its members with a global networking community of more than 140 companies, joining connected car stakeholders with world-class developers in a collaborative environment, resulting in free, open source middleware.

Our work is informed by automotive trends and results in standard interfaces and software that shorten development cycles, speed time to market, and reduce costs for companies adopting and producing automotive solutions. More.

Adoption and Availability

Cars with GENIVI software are on the road in four continents (Asia, Europe, North and South America). Suppliers are offering commercial products with GENIVI software in all continents (except Antartica). More.

Resources

W3C Case Study

Certificate Pinning Technology Brief

Shared State Rendering Technology Brief

Interactive Cockpit HMI Using Surface Sharing

Man in the Middle Attacks and Secured Communications Whitepaper

Stay Connected

GENIVI LinkedIn Group

Twitter

Next GENIVI Event

GENIVI Technical Summit
Register Here
October 10-11, 2018
Sheraton Grand Hotel at Brigade Gateway
Bangalore, India

News and Views

07/09/2018
GENIVI and FASTR to develop automotive cybersecurity solutions
Road Traffic Technology

07/02/2018
Genivi and Fastr pool resources to tackle cyber security
Vehicle Electronics

06/14/2018
Broadcasters Pursue Dashboard Collaboration
Radio World

06/14/2018
GENIVI Alliance and FASTR collaborate to develop automotive cybersecurity guidelines
Telematics News

View All

Press Releases

09/14/2018
GENIVI Alliance to Host Automotive Technical Summit in Bangalore

06/04/2018
GENIVI Alliance and FASTR Join Forces to Tackle Automotive Cybersecurity Challenges

View All

Newsletter

Sign up for the GENIVI public newsletter

View the most recent newsletter NOW

Member Listing

For a complete listing of our current GENIVI members, please click here.

Open Source Projects

GENIVI hosts open source projects that facilitate the development of required In-Vehicle Infotainment functionality. More .

Compliance Programs

The GENIVI Compliance Programs consist of the GENIVI Compliant™ and Works with GENIVI™ offerings. More.

Upcoming Events

NAB Radio Show
September 25-28, 2018
Orlando

CVTA 9th Annual Future of Connected Vehicles
October 3-4, 2018
Detroit

Tu-Automotive West Coast
October 3-4, 2018
San Jose

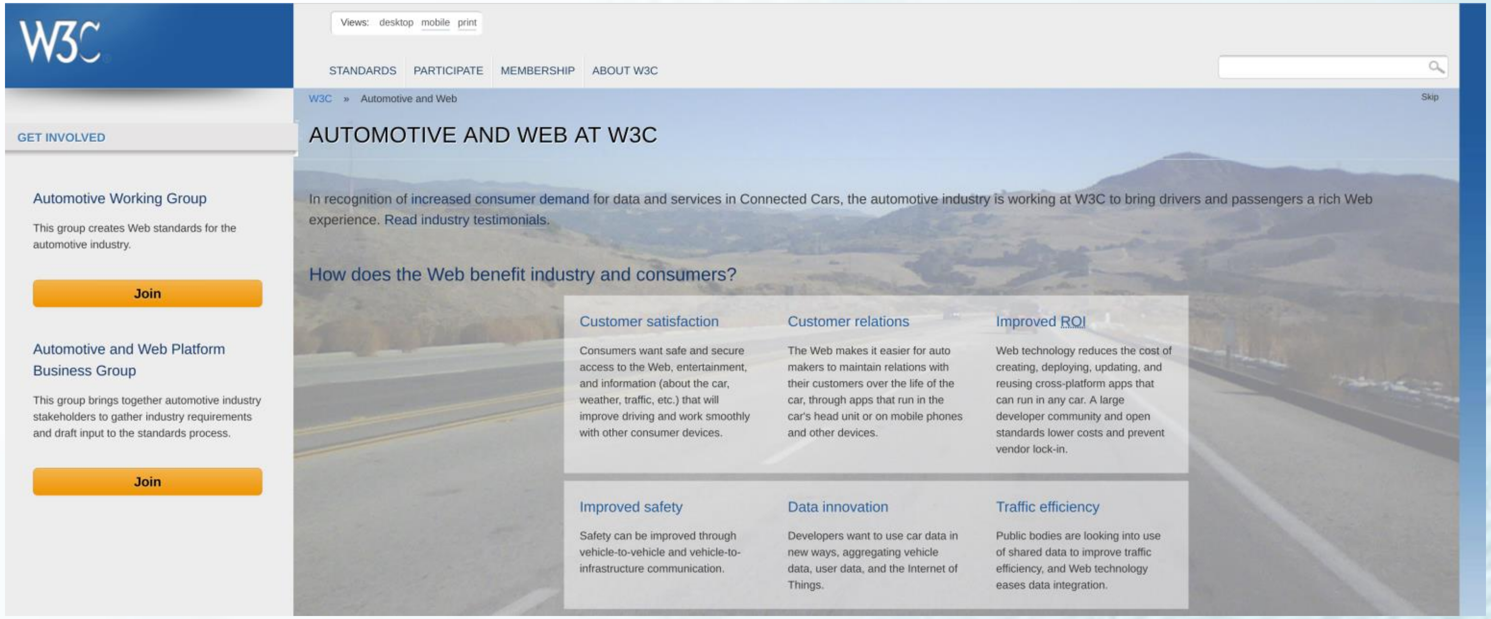
GENIVI Technical Summit
October 10-11, 2018
Sheraton Grand Hotel at Brigade Gateway
Bangalore, India

more events

Past All Member Meetings

GENIVI All Member Meeting & Open Community Days
April 17-19, 2018
Hilton Munich Park Hotel
Presentations

Copyright © 2018 GENIVI® Alliance. All Rights Reserved. Privacy Policy | Cookie Policy | Alliance Management by Global Inventures | Admin



AUTOMOTIVE AND WEB AT W3C

Automotive Working Group

This group creates Web standards for the automotive industry.

Automotive and Web Platform Business Group

This group brings together automotive industry stakeholders to gather industry requirements and draft input to the standards process.

In recognition of increased consumer demand for data and services in Connected Cars, the automotive industry is working at W3C to bring drivers and passengers a rich Web experience. Read industry testimonials.

How does the Web benefit industry and consumers?

Customer satisfaction

Consumers want safe and secure access to the Web, entertainment, and information (about the car, weather, traffic, etc.) that will improve driving and work smoothly with other consumer devices.

Customer relations

The Web makes it easier for auto makers to maintain relations with their customers over the life of the car, through apps that run in the car's head unit or on mobile phones and other devices.

Improved ROI

Web technology reduces the cost of creating, deploying, updating, and reusing cross-platform apps that can run in any car. A large developer community and open standards lower costs and prevent vendor lock-in.

Improved safety

Safety can be improved through vehicle-to-vehicle and vehicle-to-infrastructure communication.

Data innovation

Developers want to use car data in new ways, aggregating vehicle data, user data, and the Internet of Things.

Traffic efficiency

Public bodies are looking into use of shared data to improve traffic efficiency, and Web technology eases data integration.

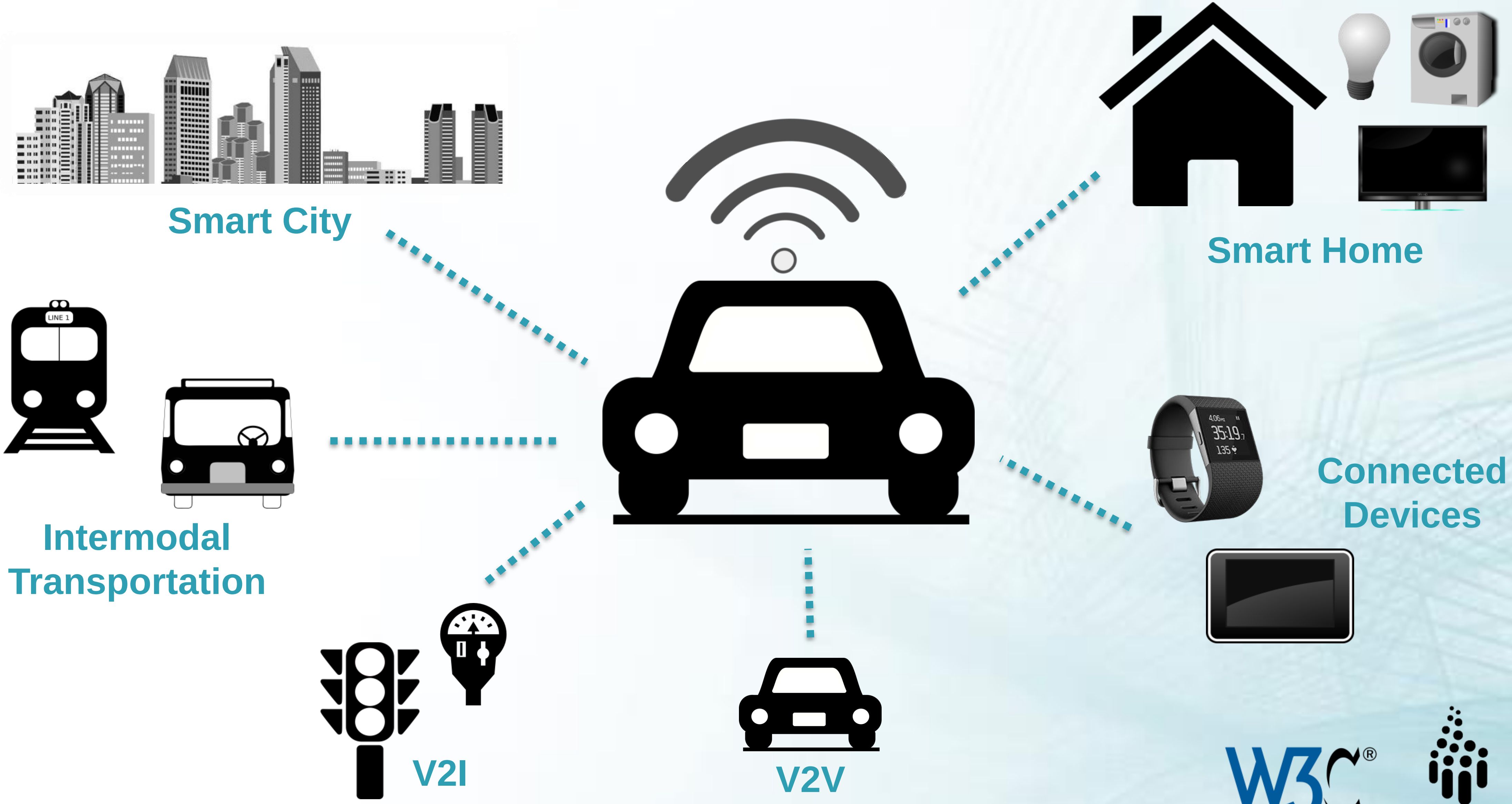
www.w3c.org/auto

www.genivi.org

W3C Automotive Working Group

- Mission – develop specifications for an open web platform for developers to access vehicle data through IVI systems and vehicle data protocols
- Participants – Access, Alibaba Group, APTOPT, Audi, Baidu, Caruso GmbH, ETRI, Fraunhofer Gesellschaft, GENIVI Alliance, Geotab, IBM, INRIX, Institut Telecom, International Forecourts Standards Forum, Jaguar Land Rover, KDDI, LG Electronics, Mitsubishi Electric, VW
- Status – Vehicle Information Service Specification (VISS) is currently in Candidate Recommendation (CR) state to become an official W3C standard.

The Problem



Smart City

Smart Home

Intermodal Transportation

Connected Devices

V2I

V2V

W3C®

GENIVI®

The Challenge

- Providing access to vehicle status information and data to cloud services, web applications, mobile devices and more.
- There is no standard convention for a vehicle data API.
- OEMs wish to be able to easily extend a standard API with signals and controls for their purposes.
- Security mechanisms are required that provide authentication and authorization to access vehicle signals and control.
- Design that decouples signal interface from the electrical architecture of the vehicle.

Conventional Approach – “Fat API”

- An API for every signal or control:

```
var vehicle = navigator.vehicle;
vehicle.vehicleSpeed.get().then(function (vehicleSpeed) {
    console.log("Vehicle speed: " + vehicleSpeed.speed);
}, function (error) {
    console.log("There was an error"); });
var vehicleSpeedSub = vehicle.vehicleSpeed.subscribe(function (vehicleSpeed) {
    console.log("Vehicle speed changed to: " + vehicleSpeed.speed);
    vehicle.vehicleSpeed.unsubscribe(vehicleSpeedSub);
});
```

- Issues with this approach:
  - Addition of new signals and controls requires change of the specification.
  - Challenges maintaining backwards compatibility.
  - Complexity in providing per-API authorization and access control.
  - Single end-point addressing.

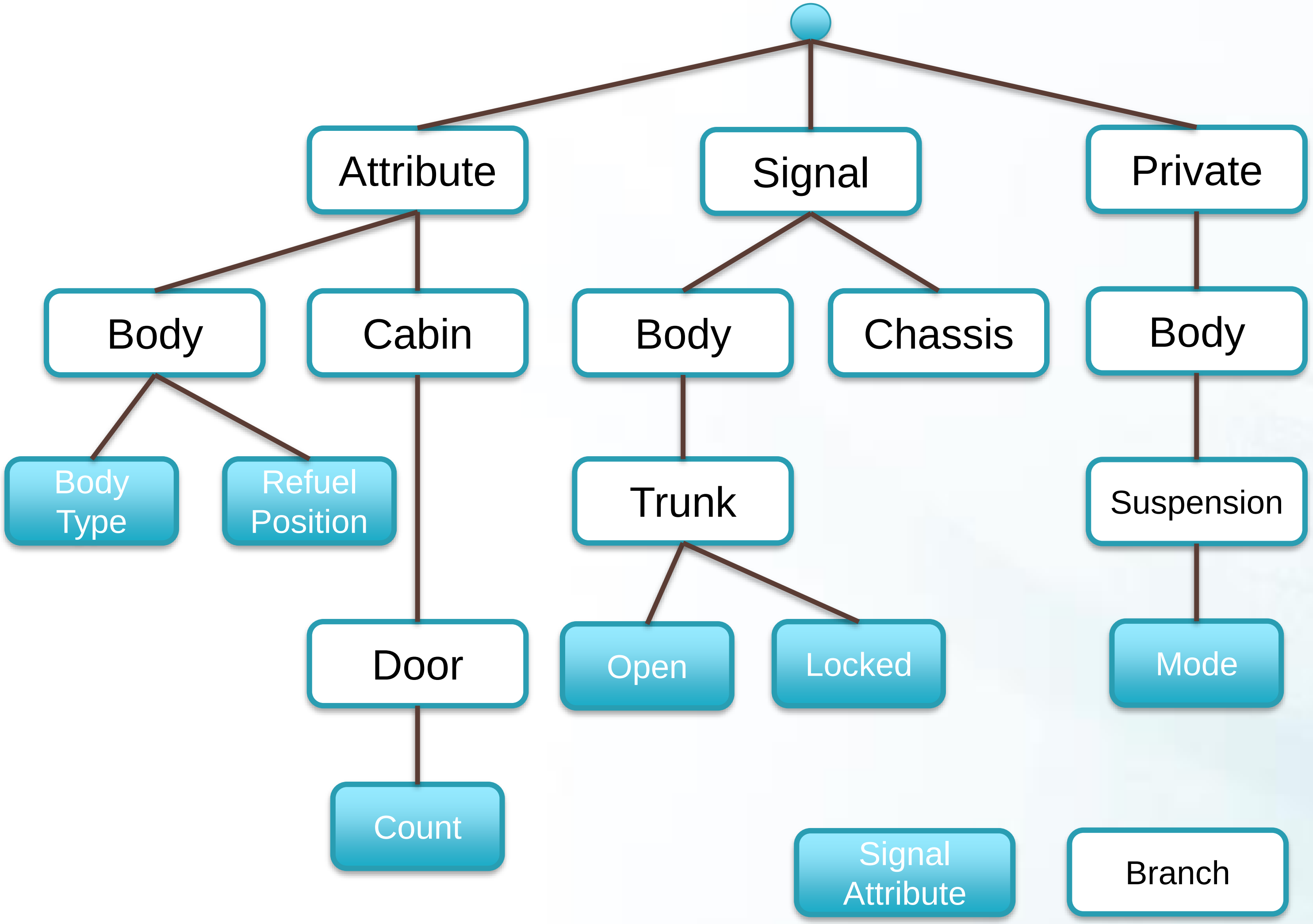
New Approach – Services with Signal Tree

- The core services *get*, *set*, *subscribe*, *unsubscribe*, *getVSS* and *authorize* are provided by a network server.
  - The services *get*, *set*, *subscribe* and *unsubscribe* provide access to vehicle signals and controls.
  - The service *getVSS* allows clients to query the server for available signals.
  - Using the *authorize* service, the client presents a security token to the server for authentication and authorization.
- Vehicle Signals and Controls are identified as nodes of a vehicle signal tree.
  - A fully qualified signal name addresses a single signal node.
  - Wildcards for branches and node names provide for addressing of signal groups.

Vehicle Signal Tree

Vehicle Signal Specification

Vehicle Signal Tree



- Tree structure provides for hierarchical access to signals and attributes.
- Branches group signals and attributes into entities that logically belong together.
- Wildcards allow access to entire sets of signals.

Addressing

```
Signal.Chassis.Brake.FluidLevel
Signal.Drivetrain.FuelSystem.Level
Attribute.Cabin.Door.Count
Attribute.Engine.Displacement
```

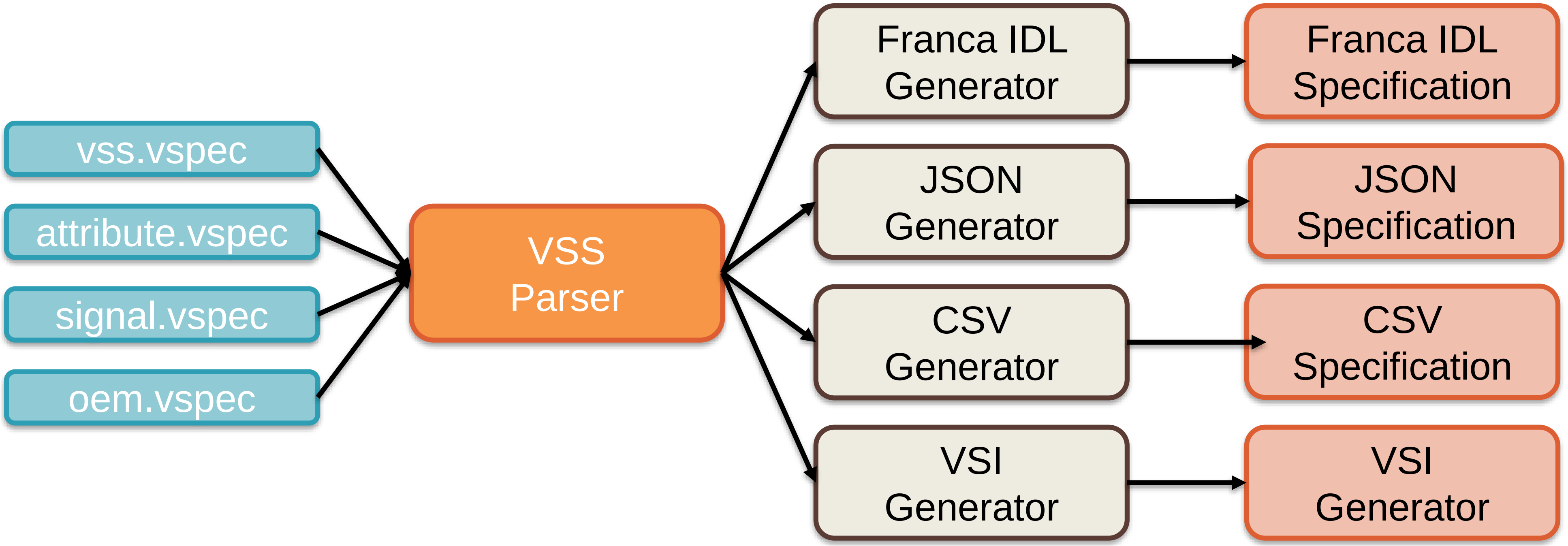
- Dot-notation for name path.
- Last path component, called node, represents the signal or attribute.
- Leading path components represent the branches.
- Wildcards can be used to address multiple signals and/or branches.

Specification Format

```
- Signal.Drivetrain.Transmission:
  type: branch
  description: Transmission-specific data
- Signal.Drivetrain.Transmission.Speed:
  type: Int32
  min: -250
  max: 250
  unit: m/s
  description: Current vehicle speed, sensed by gearbox
```

- Formatted as YAML lists
- Simple conversion into other formats such as JSON, France IDL, CSV, and more
- # denotes a comment or a directive

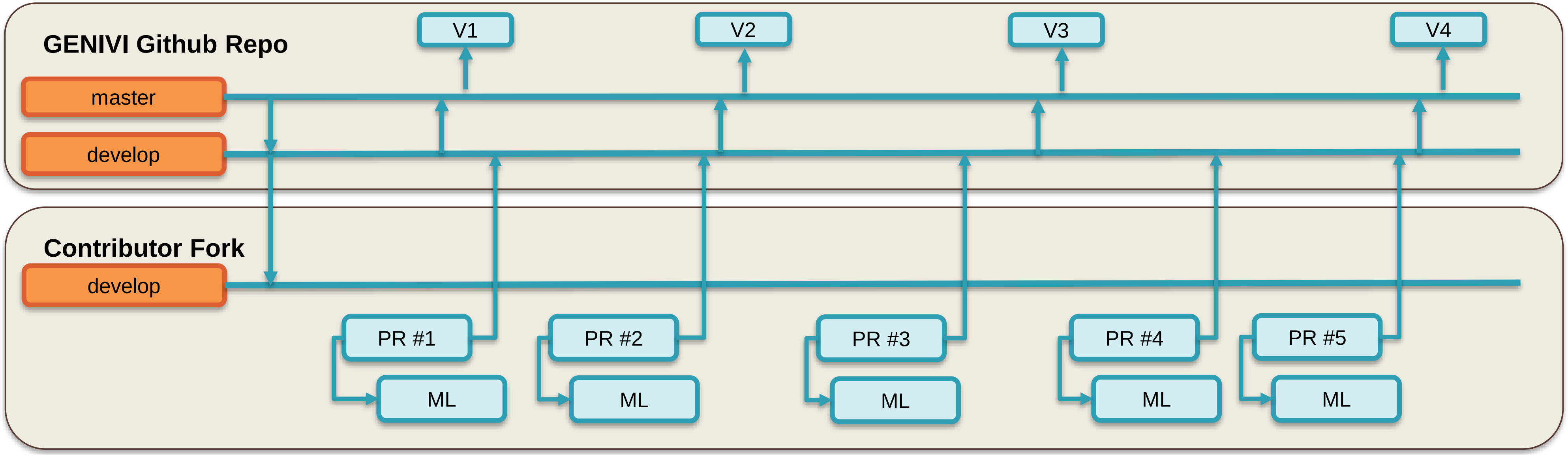
Format Transformation



- Tools written in Python transform VSS YAML (vspec) format into other formats.
- Standard Python library parses VSS YAML into a data structure.
- Output generators use the data structure to write their specific format.
- Output generators for Franca IDL, JSON, CSV and VSI are currently available. Other generators can easily be added.
- The VSI generator creates an alphabetically sorted list of the fully qualified signal and attribute names and assigns an index value to them.

Contribution and Releases

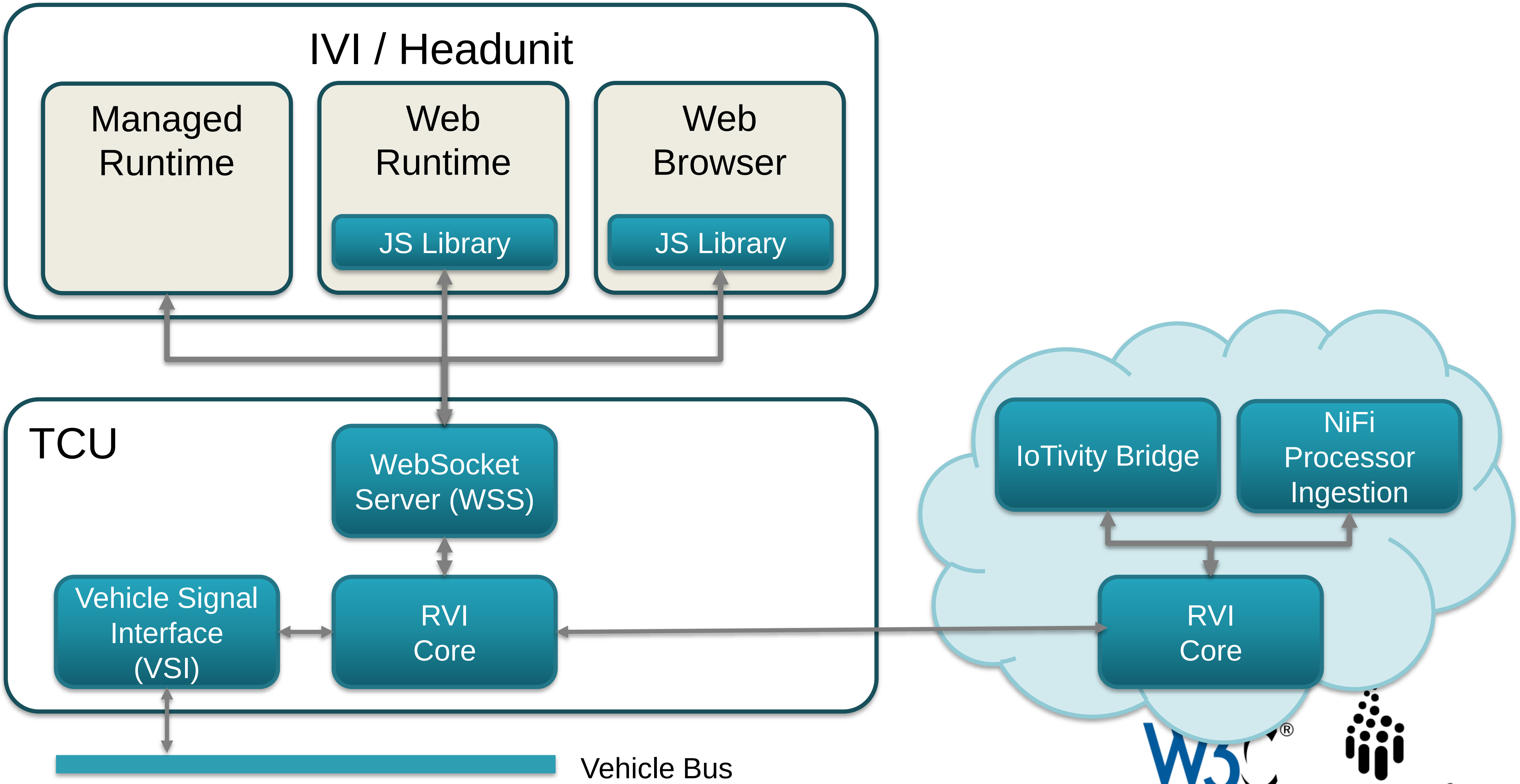
- Repository on Github under the GENIVI organization: https://github.com/GENIVI/vehicle_signal_specification



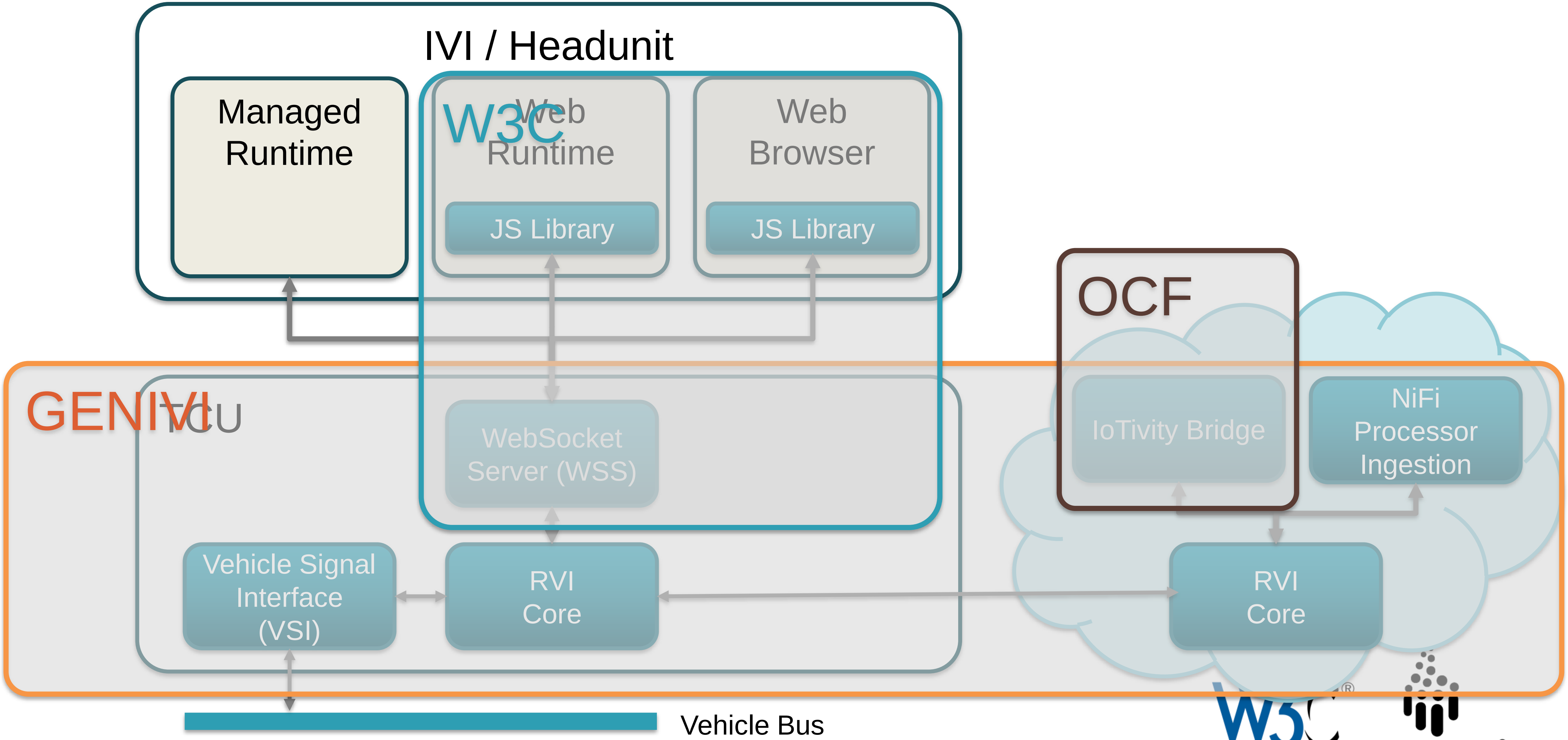
- Contributor forks GENIVI VSS repo.
- Contributor makes changes and submits pull-request against develop branch.
- Contributor e-mail genivi-projects mailing list pull-request info (hypertext link).
- Maintainer and contributors discuss and approve. Maintainer merges pull request.
- Releases are created by merging the develop branch into the master branch and tagging the master branch.

Architecture

Vehicle Data Interfaces Architecture



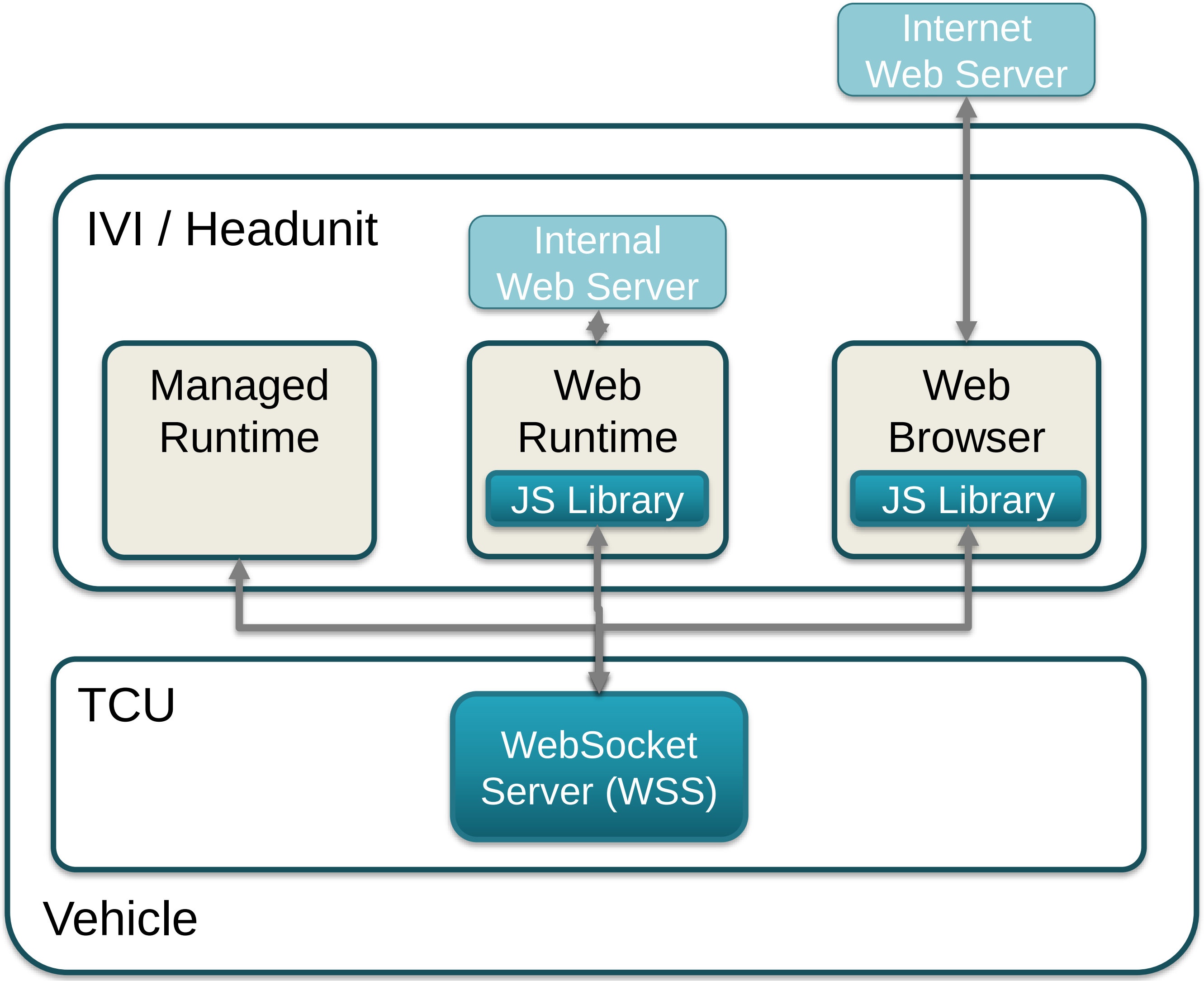
Vehicle Data Interfaces Architecture



Vehicle Information Service Specification (VISS)

Overview

- A web socket server (NodeJS etc.) provides access to vehicle signals.
- Web clients such as applications running inside a web runtime or a web browser communicate with the web socket server using a JavaScript library which implements the web socket server protocol and exposes an object API.
- Native clients can directly use the web socket server protocol.
- Clients can be agents with no UI or applications with UI.



Service Messages

Service Messages	
authorize	Enables client to pass security tokens for Security Principals to the server to support access-control.
getVSS	Allows the client to request metadata describing signals and data attributes that are potentially accessible.
get	Enables the client to get a value once.
set	Enables the client to set a value once.
subscribe	Enables the client to receive a notification containing a JSON data structure with values for one or more vehicle signals and/or data attributes. The client requests that it is notified when the signal changes on the server.
unsubscribe	Allows the client to notify the server that it should no longer receive notifications based on that subscription.
unsubsribeAll	Allows the client to notify the server that it should no longer receive notifications for any active subscription.

Repository and Specification

https://github.com/w3c/automotive

https://w3c.github.io/w3c/automotive

Thank you!

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

Visit W3C Automotive at http://www.w3c.org/auto
https://www.w3.org/auto/wg / https://www.w3.org/community/autowebplatform
Contact Ted Guild: ted@w3c.org

Backup Slides

Vehicle Signal Tree

Vehicle Signal Specification

Specification Format – Branch Description

```
- Signal.Drivetrain.Transmission:
  type: branch
  description: Transmission-specific data
```

- Fields
  - `type` – always set to branch for a branch
  - `description` – informative text describing the branch

Specification Format – Signal Description

```
- Signal.Drivetrain.Transmission.Speed:
  type: Int32
  min: -250
  max: 250
  unit: m/s
  description: Current vehicle speed, sensed by gearbox
```

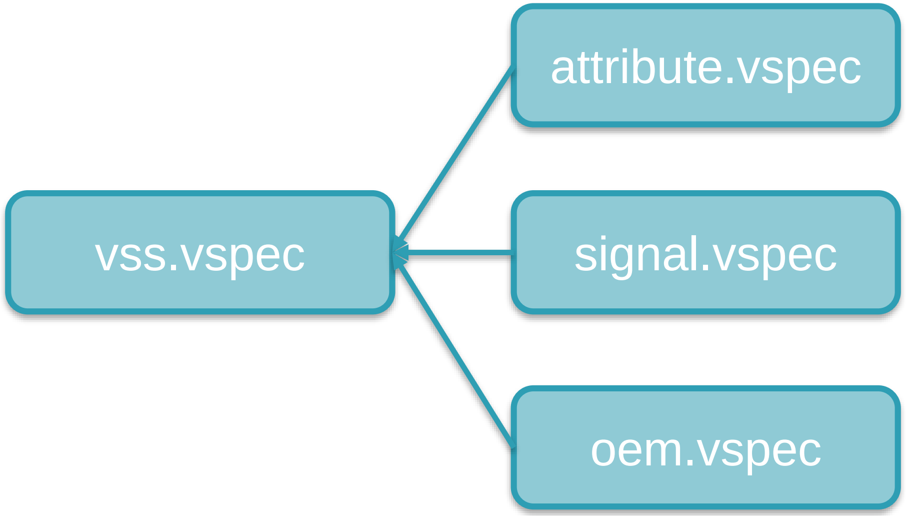
- Fields
  - `type` – data type expressed as France IDL data type
  - `unit` – SI unit unless the type is Boolean
  - `min`, `max` – unless the type is Boolean or enumeration
  - `enum` – enumeration values for enumeration
  - `description` – informative text describing the signal

Specification Format – Attribute Description

```
- Attribute.Cabin.Door.Count:
  type: Uint8
  value: 4
  description: Current vehicle speed, sensed by gearbox
```

- Fields
  - Same as signal
  - `value` – attribute setting
- Attributes are used to describe configuration data.

Aggregate File Inclusion

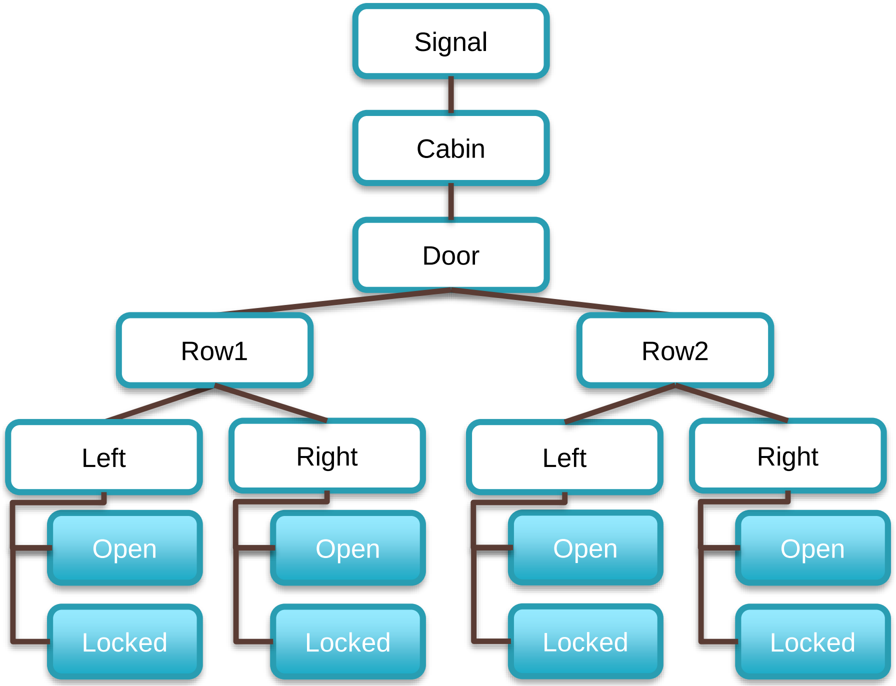


vss.vspec

```
# top level vspec
#include attribute.vspec
#include signal.vspec
#include oem.vspec
```

- Vehicle signal specification files (vspec) can include other vspec file using the `#include` directive.
- Content of the included file is inserted into the including file at the position of the `#include` directive.
- Facilitates collaboration and minimizes editorial conflicts.

Reuse File Inclusion



door.vspec

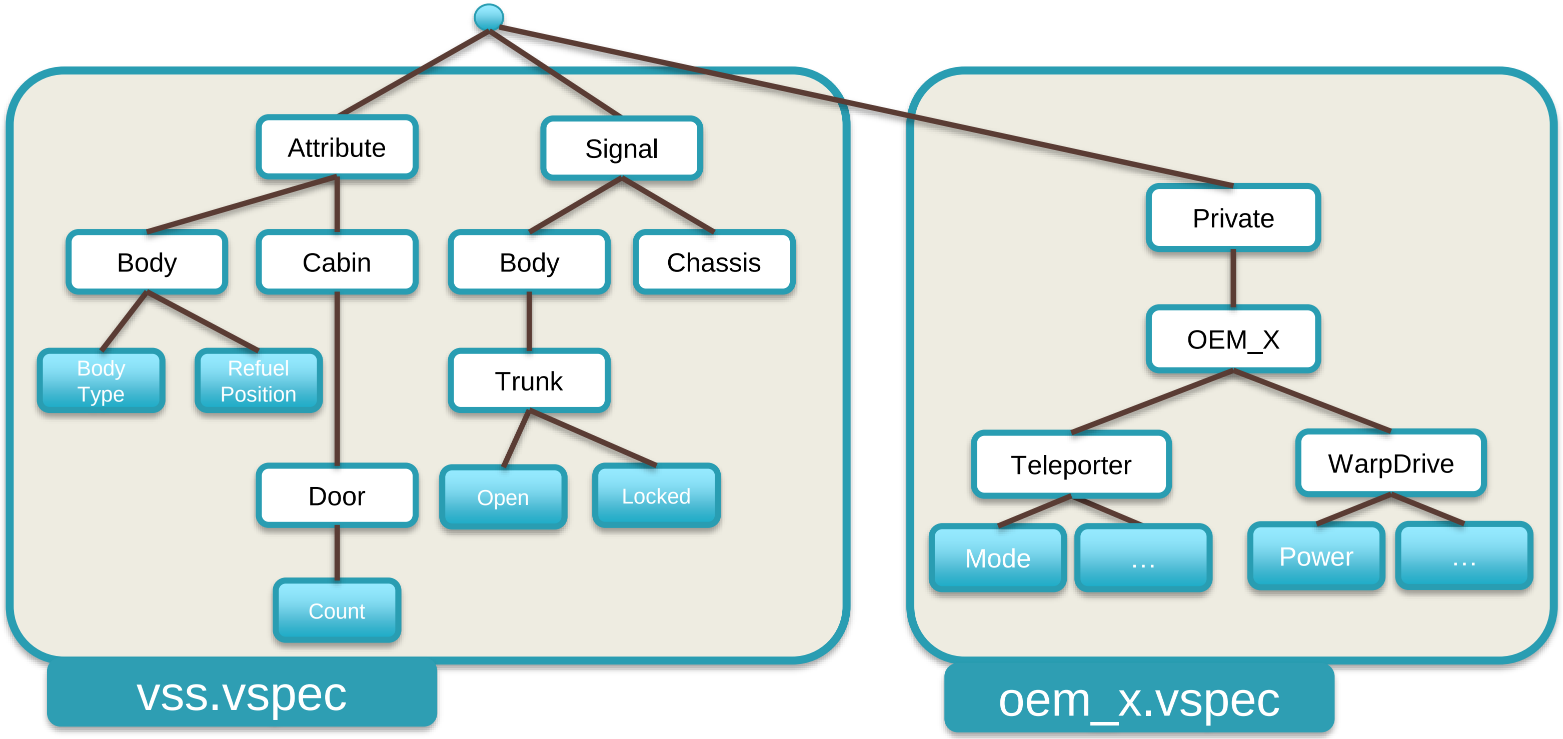
```
# door signals
-  Open:
   type: Boolean
   description: Door is open
-  Locked:
   type: Boolean
   description: Door is locked
```

cabin.vspec

```
# doors
#include door.vspec Signal.Cabin.Door.Row1.Left
#include door.vspec Signal.Cabin.Door.Row1.Right
#include door.vspec Signal.Cabin.Door.Row2.Left
#include door.vspec Signal.Cabin.Door.Row2.Right
```

- Specification fragments are included at a specific position of the signal tree.
- Specification fragments can be reused and an update is automatically reflected everywhere where the fragment is used.

Private OEM Extensions



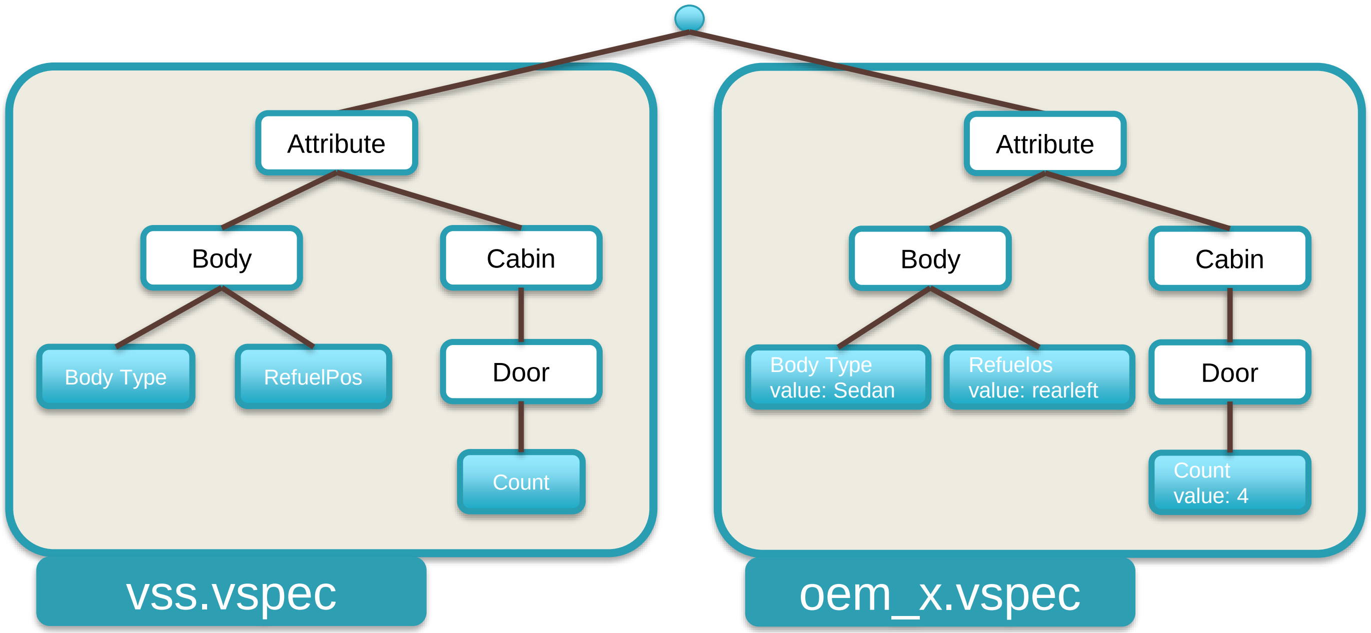
oem_x.vspec

```
# Include standard vspec
#include vss.spec

# Add proprietary signals
- Private.OEM_X.Teleporter.Mode:
  …
- Private.OEM_X.WarpDrive:
  …
```

- OEMs can use GENIVI vspec as a starting point and add proprietary signals.
- Use cases for
  - Reserved use by OEM and chosen vendors;
  - Public use by 3rd party application developers.
- Mature private extensions intended for public use can be submitted for VSS inclusion.

Attribute Declaration and Definition



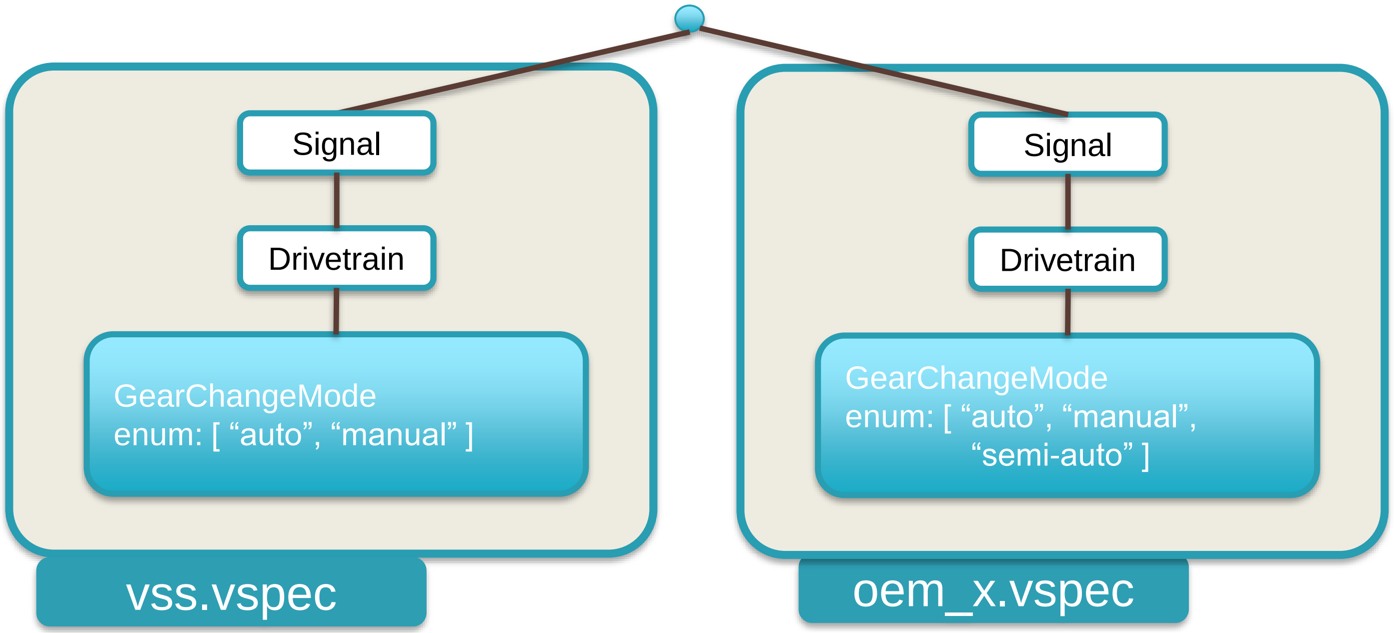
oem_x.vspec

```
# Include standard vspec
#include vss.spec

# Override/define attributes
-  Attribute.Body.BodyType:
   value: Sedan
-  Attribute.Cabin.Door.Count:
   value: 4
```

- Standard VSS either
  - Only declares an attribute or
  - Declares and attribute and assigns a default value.
- Declaration is overridden by definition in an OEM- or model-specific VSS file with the correct value.

Overriding Signal Definitions



vss.vspec

```
# Default signal definitions
- Signal.Drivetrain.GearChangeMode:
  enum: [“auto”, “manual” ]
```

oem_x.vspec

```
# Include standard vspec
#include vss.spec

# Override/define signal definitions
- Signal.Drivetrain.GearChangeMode:
  enum: [“auto”, “manual”,
              “semi-auto” ]
```

- Standard vspec lacks setting or has incorrect setting for a OEM/model etc.
- OEM/model-specific vspec can override the setting.

Vehicle Information Service Specification (VISS)

Authorization – Security Principles

Security Principal	Token Name	Description
User	Authorization	The user that the client is making requests on behalf of. This may be a person e.g. driver or passenger, it may be an organisation e.g. Emergency Services or may be any other legal entity.
Device	www-vehicle-device	The originating device that is making the request to the server. This may be an ECU in the vehicle that is hosting the WebSocket Server or may be a device that is connected to the vehicle via a WiFi hotspot or may be any other device.

Authorization – Example

```
if(userTokenOnly){
  // Pass user token only
  vehicle.send('{ "action": "authorize",
    "tokens": { "authorization": "<user_token_value>" },
    "requestId": "<some_unique_value>" }');

} else if (deviceTokenOnly) {
  // Pass vehicle/device token only
  vehicle.send('{ "action": "authorize",
    "tokens": { "www-vehicle-device": "<device_token_value>" },
    "requestId": "<some_unique_value>" }');

} else if (userAndDeviceToken) {
  // Pass tokens for user and device
  vehicle.send('{ "action": "authorize",
    "tokens": { "authorization": "<user_token_value>",
      "www-vehicle-device": "<device_token_value>" },
    "requestId": "<some_unique_value>" }');
}
```

Authorization – Security Token

Element	Description
Path	The signal path the token authorizes. The path may be a branch name or contain wildcards to authorize entire branches.
Actions	List of actions that the token authorizes for the path. The list contains at least one of the actions getVSS, get, set, subscribe and unsubscribe.
Valid From	Timestamp in UTC indicating the date and time from which on the token is valid.
Valid Until	Tmestamp in UTC indicating the date and time until which the token is valid.

Introspection – getVSS

WebIDL

```
interface vssRequest {
    attribute Action  action;
    attribute string? path;
};

interface vssSuccessResponse {
    attribute Action action;
    attribute string path;
    attribute object vss;
};

interface vssErrorResponse {
    attribute Action action;
    attribute string path;
    attribute Error  error;
};
```

Message

```
client -> {
   "action": "getVSS",
   "path": "Signal.Body"
 }
 receive <- {
   "action": "getVSS",
   "path": "Signal.Body",
   "vss": { }
 }
```

Get Signal Value – get

WebIDL

```
interface getRequest {
    attribute Action  action;
    attribute DOMString path;
};

interface getSuccessResponse {
    attribute Action action;
    attribute DOMString path;
    attribute any value;
    attribute DOMTimeStamp timestamp;
};

interface getErrorResponse {
    attribute Action action;
    attribute DOMString path;
    attribute Error  error;
 attribute DOMTimeStamp timestamp;
};
```

Message

```
client -> {
   "action": "get",
   "path": "Signal.Drivetrain.Speed",
 }
 receive <- {
   "action": "get",
   "path": "Signal.Drivetrain.Speed",
   "value": 55,
   “timestamp": <DOMTimeStamp>
 }
```

Set Signal Value – set

WebIDL

```
interface getRequest {
    attribute Action  action;
    attribute DOMString path;
    attribute any value;
};

interface setSuccessResponse {
    attribute Action action;
    attribute DOMString path;
    attribute any value;
    attribute DOMTimeStamp timestamp;
};

interface setErrorResponse {
    attribute Action action;
    attribute DOMString path;
    attribute Error  error;
    attribute DOMTimeStamp timestamp;
};
```

Message

```
client -> {
    "action": "set",
    "path": "Signal.Cabin.Door.*.IsLocked",
    "value":{ [ {"Row1.Right.IsLocked" : true },
                {"Row1.Left.IsLocked" : true },
                {"Row2.Right.IsLocked" : true },
                {"Row2.Left.IsLocked" : true } ] }
    }

  receive <- {
    "action": "set",
    "path": "Signal.Cabin.Door.*.IsLocked",
    "value":{ [
{"Signal.Cabin.Door.Row1.Right.IsLocked" : true },
{"Signal.Cabin.Door.Row1.Left.IsLocked" : true },
{"Signal.Cabin.Door.Row2.Right.IsLocked" : true },
{"Signal.Cabin.Door.Row2.Left.IsLocked" : true } ]
},
    "timestamp": <DOMTimeStamp>
  }
```

Subscription Request – subscribe

WebIDL

```
interface subscribeRequest {
    attribute Action  action;
    attribute DOMString path;
    attribute object? filters;
    attribute string requested;
};

interface subscribeSuccessResponse {
    attribute Action action;
    attribute string requestId;
    attribute string subscriptionId;
    attribute DOMTimeStamp timestamp;
};

interface subscribeErrorResponse {
    attribute DOMString path;
    attribute string requestId;
    attribute Error  error;
    attribute DOMTimeStamp timestamp;
};
```

```
interface subscriptionNotification {
    attribute string  subscriptionId;
    attribute DOMString path;
    attribute any value;
    attribute DOMTimeStamp timestamp;
};

interface subscriptionNotificationError {
    attribute string  subscriptionId;
    attribute DOMString path;
    attribute object filters;
    attribute Error error;
    attribute DOMTimeStamp timestamp;
};
```

Subscription Request – subscribe

Message

```
client -> {
    "action": “subscribe",
    "path": "Signal.Drivetrain.Transmission.TripMeter",
    “requestId": 1004 }
}

  receive <- {
    "action": “subscribe",
    "reqestId": 1004,
    “subscriptionId": 35472,
    "timestamp": <DOMTimeStamp>
  }
```

Unsubscription Request – unsubscribe

WebIDL

```
interface unsubscribeRequest {
    attribute Action  action;
    attribute string subscriptionId;
    attribute string requestId;
};

interface unsubscribeSuccessResponse {
    attribute Action action;
    attribute string? subscriptionId;
    attribute string requestId;
    attribute DOMTimeStamp timestamp;
};

interface unsubscribeErrorResponse {
    attribute Action action;
    attribute string subscriptionId;
    attribute Error  error;
    attribute string requestId;
    attribute DOMTimeStamp timestamp;
};
```

Message

```
client -> {
    "action": “unsubscribe",
    "subscriptionId": 102,
    "requestId": 5273
    }

  receive <- {
    "action": “unsubscribe",
    "subscriptionId": 102,
    "requestId": 5273
    "timestamp": <DOMTimeStamp>
  }
```

Server-side Filtering

- For signal subscriptions filters can be provided to throttle messages on the server side.
- Filters only apply to nodes of the VSS tree and not to branches.
- Filter tags include:
  - Interval
  - Range
  - Minimum Change

Message

```
// client receives data every 100ms
 { "action": "subscribe",
   "path": "<any_path>",
   "filters": { "interval": 100 },
   "requestId": "<some_unique_value>" }

 // client receives data when the value is
 // between 100 and 200 (inclusive)
 { "action": "subscribe",
   "path": "<any_path>",
   "filters": {
       "range":{ "above": 100, "below": 200 }
   },
   "requestId": "<some_unique_value>" }

 // client receives data when the value is below
 // 100 (inclusive)
 { "action": "subscribe",
   "path": "<any_path>",
   "filters": { "range": { "below": 100 } },
   "requestId": "<some_unique_value>" }
```