Towards a vehicle DATA specification

An API-first approach to model vehicle-related data.

COVESA

Accelerating the future of connected vehicles

04.2024

Daniel Alvarez-Coello
Research Engineer
BMW Group

Daniel Wilms
Product Manager Platform
SPREAD

COVESA

Accelerating the future of connected vehicles

1 **Background**

- VSS analysis and limits
- Previous attempts for more expressivity

2 **Proposal**

- VSS feature set correspondence in GraphQL
- Other features

3 **Conclusion**

- Summary
- Q&A

COVESA

Accelerating the future of connected vehicles

1 **Background**

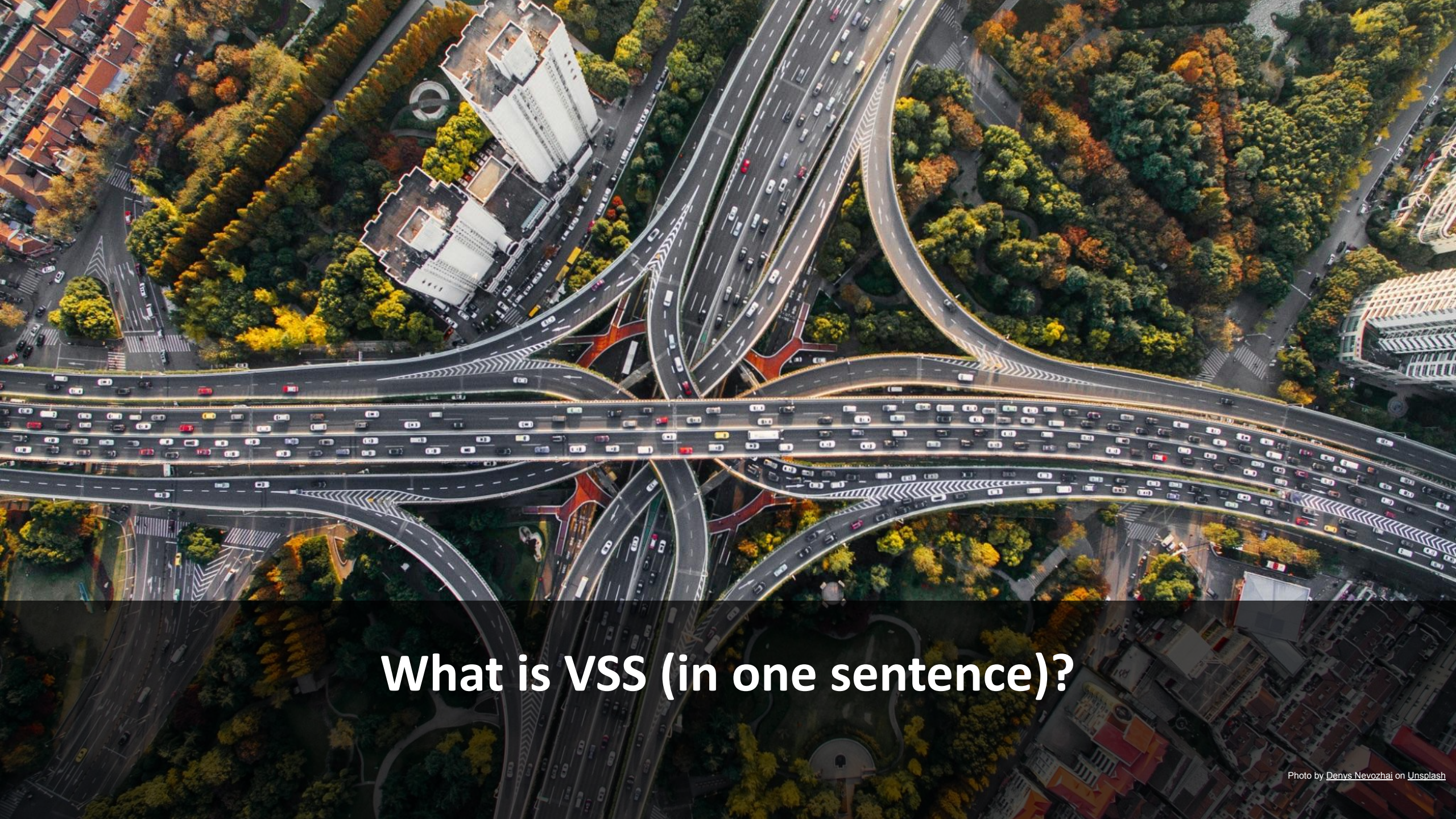
- VSS analysis and limits
- Previous attempts for more expressivity

2 **Proposal**

- VSS feature set correspondence in GraphQL
- Other features

3 **Conclusion**

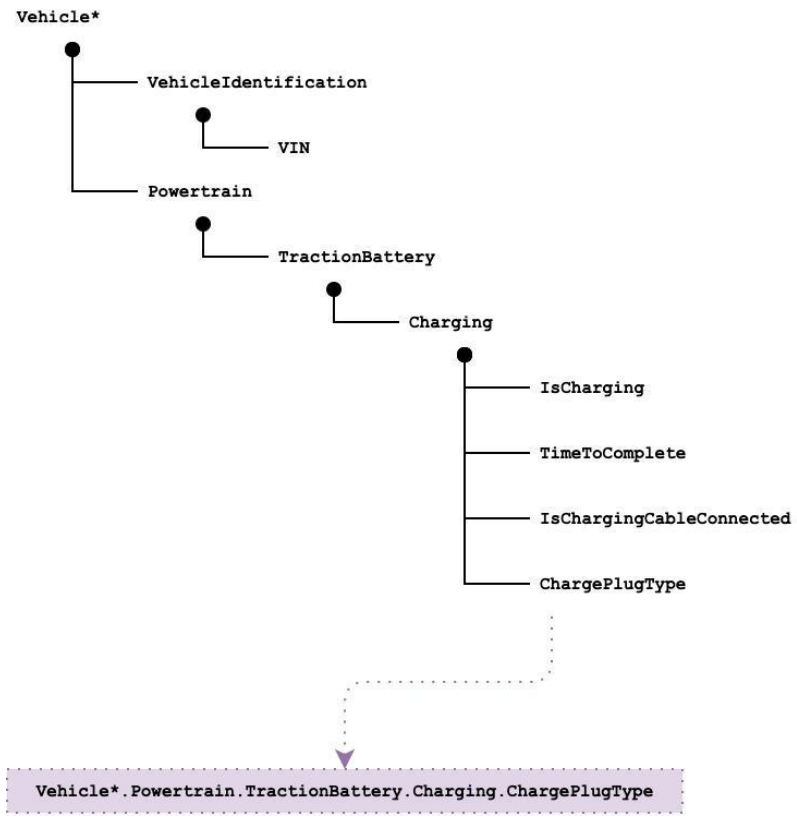
- Summary
- Q&A



What is VSS (in one sentence)?

Photo by Denys Nevozhai on Unsplash

(current) VSS is a controlled vocabulary of car properties organized in a tree hierarchy.




```
Vehicle.VehicleIdentification.VIN:
  type: attribute
  datatype: string
  ...

Vehicle.Powertrain.TractionBattery.Charging.IsCharging:
  type: sensor
  datatype: boolean
  ...

Vehicle.Powertrain.TractionBattery.Charging.TimeToComplete:
  type: sensor
  datatype: uint32
  ...

Vehicle.Powertrain.TractionBattery.Charging.IsChargingCableConnected
  type: sensor
  datatype: boolean
  ...
```

Vehicle Signal Specification (VSS)

- ✔ Easy to contribute and maintain (e.g., YAML files)
- ✔ Friendly for non-experts
- ✔ Tools to export it in different formats
- ✔ Serve as a **naming convention** for vehicle properties

https://covesa.github.io/vehicle_signal_specification/

What is the current scope of VSS?

Photo by Victor Sánchez Berruezo on Unsplash

Our understanding of the functional requirements covered by VSS.*

Functional requirement	VSS item
Model can be specified in multiple files.	Include
Human-friendly context is provided.	Concatenated path
Hierarchical concept scheme to group properties belonging to the same area of interest is supported.	Branch type Aggregate
Properties can be grouped .	Aggregate Struct
Properties whose values do not change often can be specified.	Attribute type
Properties whose value might change often can be specified as observable or actuatable.	Sensor type Actuator type
Definitions can be reused when there are multiple occurrences.	Instances
Arrays or lists are supported.	Arrays (i.e., datatype[])
Extended list of primitive datatypes is supported.	(u)intX, boolean, float, double, string
Units and quantity kind can be specified.	Unit Dimension
Default values can be specified.	Default
Allowed values can be specified.	allowed: [‘value1’, ..., ‘valueN’]
A custom redefinition of the concepts is possible.	Overlay
Min and max expected values can be specified.	min: 0 max: 100
Concepts in the model can evolve.	deprecation

* As of 02.2024. Interpretation of the official project documentation available at: https://covesa.github.io/vehicle_signal_specification/

Do these features have any limit, disadvantage or problem?

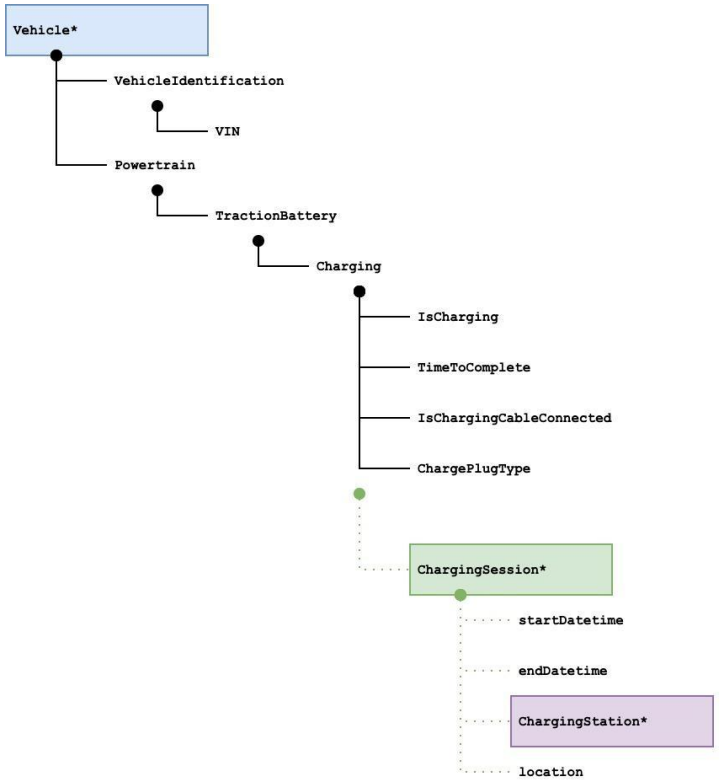
Photo by Victor Sánchez Berruezo on Unsplash

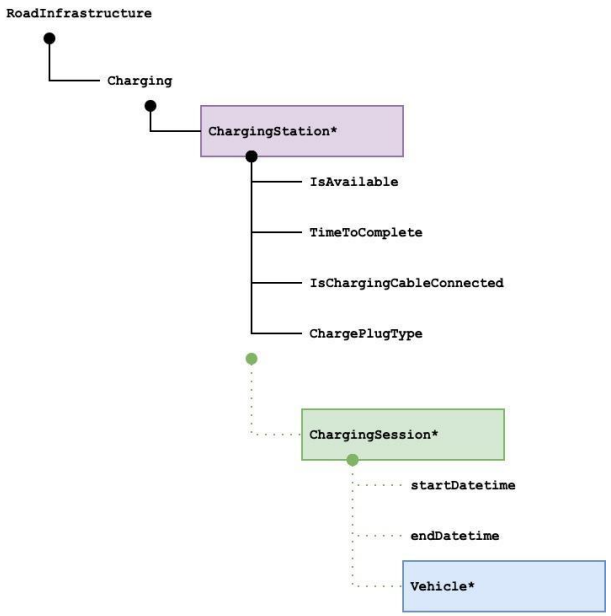
VSS limits* (1)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- No unified or controlled criteria is given for classifying the information in one tree.
- Using multiple classification criteria within the same concept scheme (i.e., tree) is a bad practice.
- Often used as the ID.**

Where would you locate concepts of mutual interest? (e.g., **`ChargingSession`**, **`SeatOccupancy`**)





* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.
** The identifiers have been recently addressed by the inclusion of a mechanism that creates a hashed UUID and it is expected to be used in the future.

VSS limits* (2)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- Branches provide context only. The primary focus is on **`SomeFeatureOfInterest.SomeProperty`**.
- The dot separator has various different implicit meanings.
- One tree can effectively cover **one domain only**.

Vehicle components or physical ***parts*** (i.e., material things).

```
Cabin.Door.Window

Cabin.(<--partOf--)Door.(<--partOf--)Window
```

Functions of a physical part (i.e., immaterial things also part of the same concept scheme).

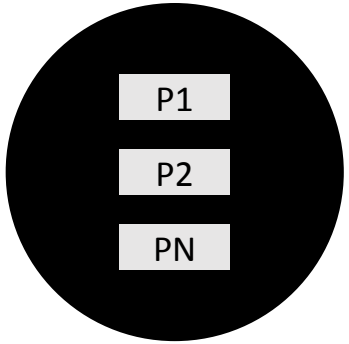
```
Battery.Charging

Battery.(<--functionOf--)Charging
```

* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.

VSS limits* (3)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- Little to no use of them in the public VSS.
- No control over how vehicle properties are used in the actual implementation.
- Mix up between the responsibilities of a data model and a data schema.



Struct → R&W collectively.
Aggregate → R&W atomically.

Enforced by whom?

* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.

COVESA
Accelerating the future of connected vehicles

VSS limits* (4)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- No timestamp associated to the actual data. It must be complemented with the time domain in the schema.
- If an attribute change its value, how is the history supposed to be handled?
- Is the history a task of the model, or the actual database that uses the model?

* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.

VSS limits* (5)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- No single sensor or actuator is being modeled (VSS models the properties and the features of interest).
- The read and write capabilities are ultimately defined in the actual implementation.

```
Vehicle.Speed:
  type: sensor
  datatype: float
  unit: km/h
  description: Vehicle speed.
```

We don’t model the sensor (e.g., `Speedometer`) but the property `Speed`.

```
Vehicle.Cabin.Seat.[instance].Position:
  type: actuator
  datatype: uint16
  unit: mm
  description: Seat position...
  min: 0
```

We don’t model the actuator (e.g., `SeatPositionMotor`) but the seat’s property `Position`.

* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.

VSS limits* (6)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- Default instances are not necessarily going to be used. A user can always overwrite.
- Instance definition should be decoupled from the actual definition of the concept that can be instantiated.
- The resulting path has the instance name embedded.

* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.

VSS limits* (7)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- Currently, no support for multiple units that belong to the same quantity kind (ongoing discussion).
- The resulting path has the instance name embedded.

```
# Datetime

unix-time:
  definition: Number of non-leap seconds which have passed since 00:00:00 UTC on Thursday, 1 January 1970
  unit: UNIX Timestamp
  quantity: datetime
  allowed-datatypes: ['uint32','uint64','int64']
iso8601:
  definition: Date and Time expressed as a string according to ISO 8601
  unit: ISO 8601
  quantity: datetime
  allowed-datatypes: ['string']
```

Is the timestamp unit?
Should it not be a datatype?

* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.

VSS limits* (8)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- Information-only (i.e., just referencial).
- Validation is not part of the model but of the implementation.

* *Disclaimer:* The opinion presented does not necessarily reflect the thinking of the entire alliance. More than a critique, it is an invitation to improve collectively.

Are there other considerations or misunderstandings?

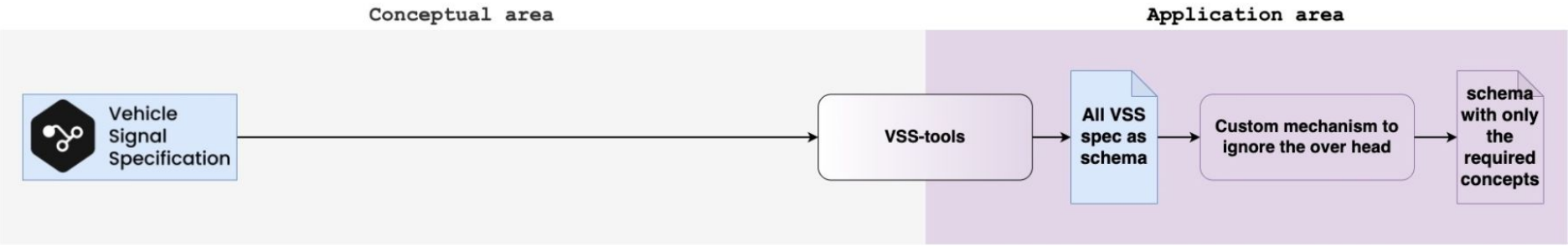
Photo by Pablo García Saldaña on Unsplash

Yes, here are a few aspects that require attention:

~~**Vehicle**~~
- So far, the focus has been entirely on cars and not vehicles!
- What about buses, motorcycles, hydrofoils, planes, etc.?

~~**Signal**~~
- In VSS, a "signal" is more like the property of a feature of interest
- VSS doesn't cover the physical wire carrying information, where multiple properties can be encoded (i.e., a Signal).

Specification of what exactly?
- The agreement on the meaning (i.e., conceptual or logical area)?
- The agreement on the data structure (i.e., application or physical area)?



https://wiki.covesa.global/display/WIK4/Defining+the+COVESA+data+modeling+strategy+and+its+associated+artifacts

Are we using the right tool for the job?

Data model? → Conceptual abstraction of a domain
Common goal? → To serve as a ***controlled vocabulary***

- Question translates to **"What do I want to express in the data model"**?

GraphQL Schema Language

Shape of data in APIs using a graph structure (i.e., multiple arbitrary trees) with types and fields.

SCHACL

Constraint language for RDF data

SKOS

Vocabularies.

OWL

Reasoning.

VSS tree

Custom classification concept scheme.

Term list

No hierarchies

- Passenger
- Car
- Person
- Vehicle
- Charging station
- ...

Taxonomy (or tree structures)

An organized list of terms

- Vehicle
  - Car
  - Train
- Person
  - Driver
  - Passenger
- ...

Thesaurus

A taxonomy extension that defines the concepts further.

- Vehicle
  - Preferred label
    - Vehicle @en
  - Term in other languages
    - Vehículo @es
  - Synonyms
    - Automobile
  - ...

Ontology (or semantic networks)

A formalization of our knowledge in one or multiple domains with the relationships and interactions between concepts

- A Person can own a Vehicle.
- A Passenger is a Person that moves inside a Vehicle which is driven by a different Person.
- A Driver cannot be a Passenger.
- A Car is a Vehicle that has four wheels.
- ...

Machine readable

Less ← **Complexity & Expressiveness** → **More**

*(Figure) Semantic spectrum, adapted from:
H. Hedden, The accidental taxonomist,
Third edition. Medford, New Jersey: Information Today, Inc, 2022.

COVESA
Accelerating the future of connected vehicles

Previous attempts for increased expressivity

Functional requirement
Approach can be applied to other domains (i.e., not-only cars).
Main modeling view is the feature of interest and the related properties.
Multiple (arbitrary) hierarchies are possible.
Schema as a contract.

VSS

- (BMW) An ontology for vehicle-related data.
- (Ford) Integrating Vehicle Signals with VSS and Metadata.
- (Ford) Transforming from a vehicle centric data model to a domain agnostic information model.
- (SPREAD) GraphQL schema as a contract.

https://wiki.covesa.global/display/WIK4/Data+Models+and+Ontologies

This presentation / proposal

AMM Porto 04.2023

AMM Gothenburg 04.2024

Vspec2ttl (contrib)

W3C member submission

W3C Discontinued Draft 02.2024

Some lessons learned

- Semantic Web → Steep learning curve for newcomers.
- Ontology modeling is a niche area (e.g., OWL).
- Often perceived as unnecessary overhead.
- Reasoning is (currently) a non-priority feature.
- Integration of domains has the highest priority.

Ontology VSSo

SSN SOSA

- B. Klotz, R. Troncy, D. Wilms and C. Bonnet. **"VSSo: A Vehicle Signal and Attribute Ontology," 2018.**
- D. Wilms, D. Alvarez-Coello and A. Bekan, **"An Evolving Ontology for Vehicle Signals," 2021.**
- https://www.w3.org/TR/vsso

COVESA

COVESA

Accelerating the future of connected vehicles

1 **Background**

VSS analysis and limits

Previous attempts for more expressivity

2 **Proposal**

VSS feature set correspondence in GraphQL

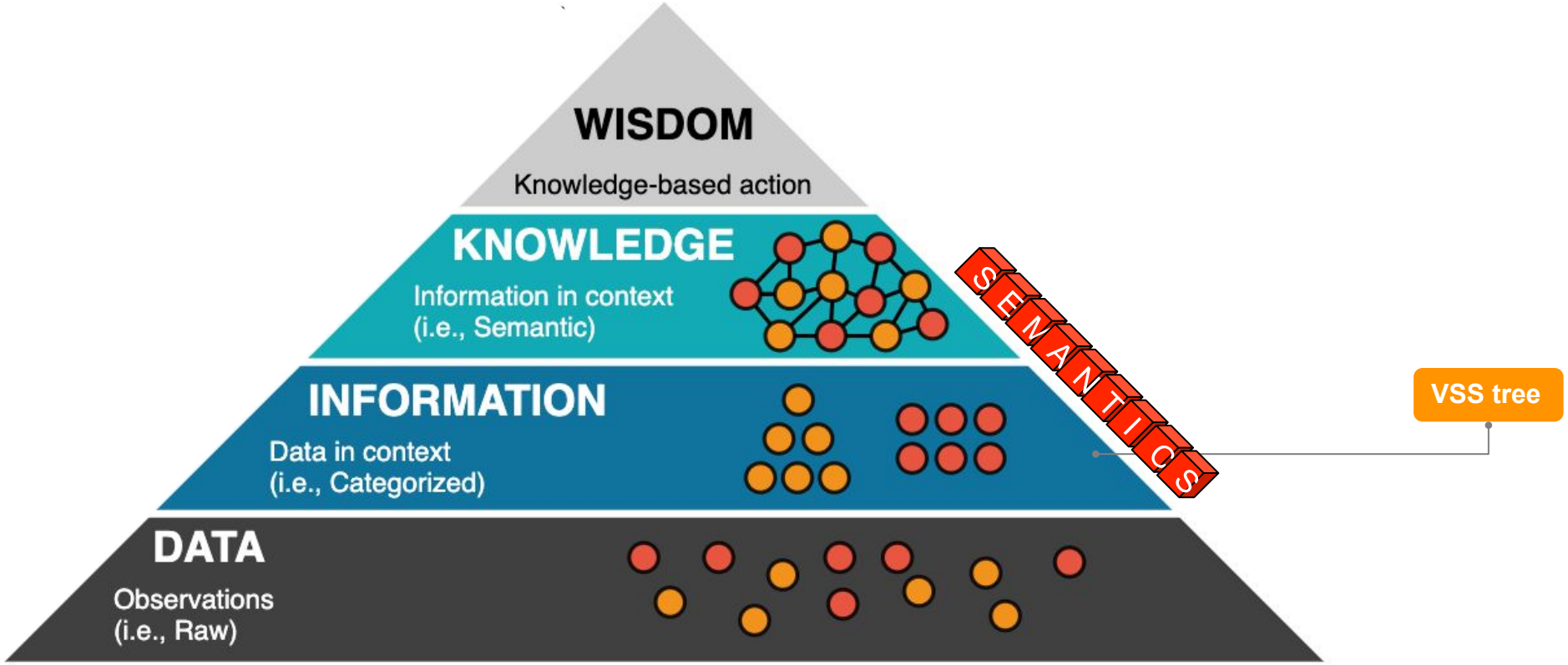
Other features

3 **Conclusion**

Summary

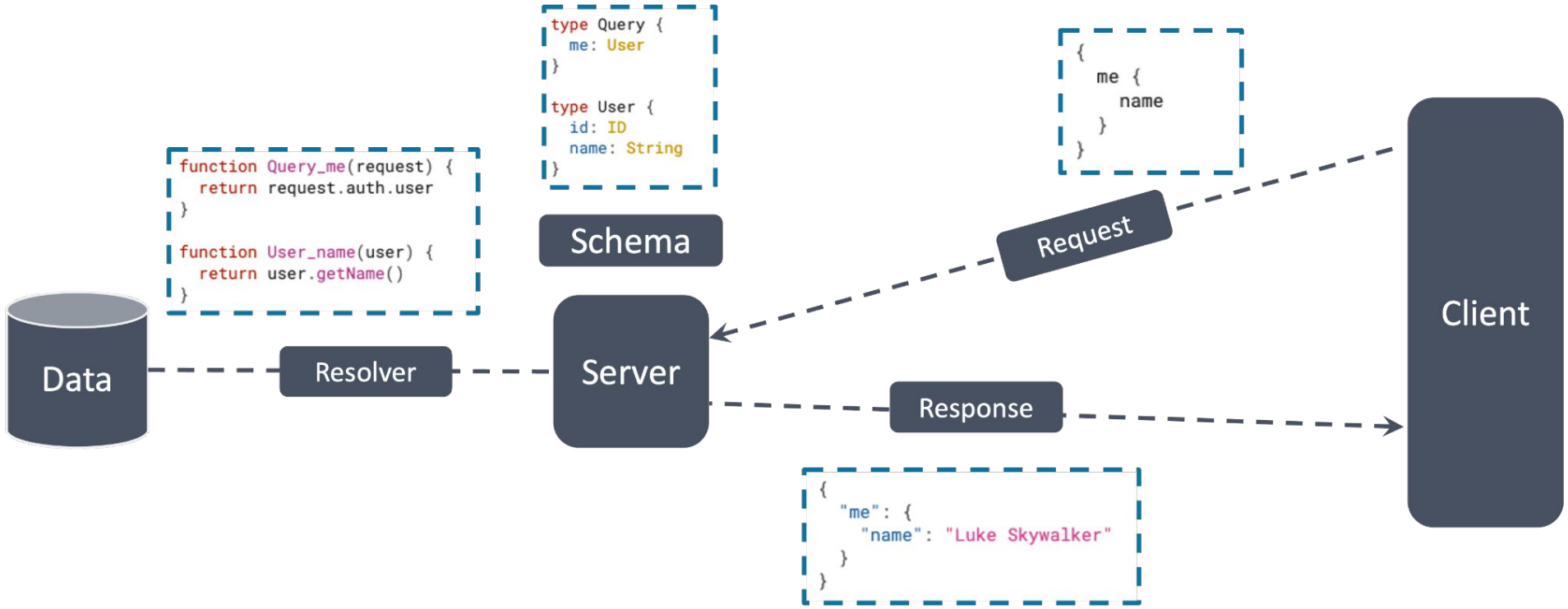
Q&A

High amount of explicit context (i.e., ontology) is key for being data-centric.



*On the Importance of semantic data models in modern architectures:
D. Alvarez-Coello, D. Wilms, A. Bekan, and J. Marx Gómez, "Towards a Data-Centric Architecture in the Automotive Industry," in Procedia Computer Science, Algarve, Portugal: Elsevier, Feb. 2021, pp. 658–663. doi: 10.1016/j.procs.2021.01.215.
**(Figure) The DIKW hierarchy, interpreted from:
J. Rowley, "The wisdom hierarchy: representations of the DIKW hierarchy," *Journal of Information Science*, vol. 33, no. 2, pp. 163–180, Apr. 2007, doi: 10.1177/0165551506070706.

GraphQL in a nutshell.



In GraphQL, a client specifies the structure of the data it needs by defining a query, which can include multiple fields and nested objects. The server then responds with the exact data requested by the client, in the same shape as the query.

VSS examples in GraphQL (1)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- Multiple files possible → Splitting by Domain or Type as needed → You model a network and not a single tree
- Modularization is possible → Smaller (more manageable) pieces
- Schema Stitching (aka., Schema Merging) → Combine everything into a bigger schema

WindowSchema.graphql

```
type Window {
  position: String!
}
```

DoorSchema.graphql

```
type Door {
  isOpen: Boolean!
  window: Window
}
```

VSS examples in GraphQL (2)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

```
"""
High-level vehicle data.
"""
type Vehicle {
  """
 Overall vehicle width.
 @original_datatype: VSSDataType.UINT16
 @unit: LenghtUnit
 """
 width: Int @deprecated(reason: "v4.1 replaced with
WidthExcludingMirrors and WidthIncludingMirrors")
}

"""
Attributes that identify a vehicle.
"""
type Vehicle_VehicleIdentification {
  """
  17-character Vehicle Identification Number (VIN) as
defined by ISO 3779.
  """
  vin: String
}
```

```
Vehicle.Width:
 datatype: uint16
 default: 0
 deprecation: v4.1 replaced with WidthExcludingMirrors
and WidthIncludingMirrors
 description: Overall vehicle width.
 type: attribute
 unit: mm

Vehicle.VehicleIdentification:
 description: Attributes that identify a vehicle.
 type: branch

Vehicle.VehicleIdentification.VIN:
 datatype: string
 type: attribute
 description: 17-character Vehicle Identification
Number (VIN) as defined by ISO 3779.
```

VSS examples in GraphQL (3)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

```
"""All Advanced Driver Assist Systems data."""
type Vehicle_ADAS {

"""
Some description here...
"""
  supportedAutonomyLevel: AutonomyLevel

enum AutonomyLevel {
  SAE_0
  SAE_1
  SAE_2
  SAE_3
  SAE_4
  SAE_5
}
```

```
Vehicle.ADAS:
  description: All Advanced Driver Assist Systems data.
  type: branch

Vehicle.ADAS.SupportedAutonomyLevel:
  datatype: string
  type: attribute
  allowed: [
    'SAE_0', # No Driving Automation
    'SAE_1', # Driver Assistance
    'SAE_2', # Partial Driving Automation
    'SAE_3', # Conditional Driving Automation
    'SAE_4', # High Driving Automation
    'SAE_5'  # Full Driving Automation
  ]
  description: Some description here...

Vehicle.ADAS.ABS:
  description: Antilock Braking System signals.
  type: branch

Vehicle.ADAS.ABS.IsEnabled:
  datatype: boolean
  description: Indicates if ABS is ...
  type: actuator

Vehicle.ADAS.ABS.IsEngaged:
  datatype: boolean
  description: ...True = Engaged. False = Not Engaged.
  type: sensor
```

VSS examples in GraphQL (4)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- In GraphQL, we specify:
  → the shape of the data
  → the operations that can be performed
- Query → Read (i.e., vss sensor type)
- Mutation → Write (i.e., vss actuator type), Update, Delete

```
"""
Antilock Braking System signals.
"""
type Vehicle_ADAS_ABS {
  """Indicates if ABS is..."""
  isEnabled: Boolean
  """... True = Engaged. False = Not Engaged."""
  isEngaged: Boolean
}
```

```
query {
  getVehicleADASABS {
    isEnabled
  }
}

mutation {
  updateVehicleADASABS(input: { isEnabled: true }) {
    isEnabled
  }
}
```

```
Vehicle.ADAS:
  description: All Advanced Driver Assist Systems data.
  type: branch

Vehicle.ADAS.SupportedAutonomyLevel:
  datatype: string
  type: attribute
  allowed: [
    'SAE_0', # No Driving Automation
    'SAE_1', # Driver Assistance
    'SAE_2', # Partial Driving Automation
    'SAE_3', # Conditional Driving Automation
    'SAE_4', # High Driving Automation
    'SAE_5'  # Full Driving Automation
  ]
  description: Some description here...

Vehicle.ADAS.ABS:
  description: Antilock Braking System signals.
  type: branch

Vehicle.ADAS.ABS.IsEnabled:
  datatype: boolean
  description: Indicates if ABS is ...
  type: actuator

Vehicle.ADAS.ABS.IsEngaged:
  datatype: boolean
  description: ...True = Engaged. False = Not Engaged.
  type: sensor
```

VSS examples in GraphQL (5)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- Things like `doorCount` can be resolved directly with a query.

```
"""All in-cabin components, including doors."""
type Vehicle_Cabin {
  """
  Number of doors in vehicle.
  @original_datatype: VSSDatatype.UINT8
  @default: 4"""
  doorCount: Int

  """
  All doors…
  instances: ['Row[1,2]', ['DriverSide', 'PassengerSide']]
  """
  door:Vehicle_Cabin_Door
}

"""All doors..."""
type Vehicle_Cabin_Door {
  """Door window status..."""
  window: Vehicle_Cabin_Door_Window
}

"""Door window status..."""
type Vehicle_Cabin_Door_Window {
  """
  Some description…
  @original_datatype: VSSDataType.UINT8
  @unit: percent
  @min: 0
  @max: 100
  """
  position: Int
}
```

```
Vehicle.Cabin.Door:
  type: branch
  instances:
    - Row[1,2]
    - ["DriverSide","PassengerSide"]
  description: All doors...

Vehicle.Cabin.DoorCount:
  datatype: uint8
  default: 4
  description: Number of doors in vehicle.
  type: attribute

Vehicle.Cabin.Door.Row1.DriverSide.Window:
  type: branch
  description: Door window status...

Vehicle.Cabin.Door.Row1.DriverSide.Window.Position:
  datatype: uint8
  type: actuator
  min: 0
  max: 100
  unit: percent
  description: Some description...
```

VSS examples in GraphQL (6)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

```
Vehicle.Cabin.Infotainment.SmartphoneProjection.SupportedMode:
  datatype: string[]
  type: attribute
  allowed: [ 'ANDROID_AUTO', 'APPLE_CARPLAY', 'MIRROR_LINK', 'OTHER' ]
  description: Supportable list for projection.
```

```
"""
All smartphone projection actions.
"""
type Vehicle_Cabin_Infotainment_SmartphoneProjection {
  """
  Supportable list for projection.
  """
  supportedMode: [ProjectionMode]
}

enum ProjectionMode {
  ANDROID_AUTO
  APPLE_CARPLAY
  MIRROR_LINK
  OTHER
}
```

VSS examples in GraphQL (7.1)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- We can specify
  → how the data is to be read with a `query`.
  → how the data is to be written with `mutation`.

```
type Vehicle_VehicleIdentification {
  vin: String
  speed: Float
}

type TractionBattery {
  netCapacity: Float
  charging: Charging
  stateOfCharge: StateOfCharge
  range: Float
}

type Charging {
  chargeLimit: Float
  isCharging: Boolean
  timeToComplete: Int
}

type StateOfCharge {
  current: Float
}
```

Vehicle.VehicleIdentification.VIN

Vehicle.Speed

Vehicle.Powertrain.TractionBattery.NetCapacity

Vehicle.Powertrain.TractionBattery.Charging.ChargeLimit

Vehicle.Powertrain.TractionBattery.StateOfCharge.Current

Vehicle.Powertrain.TractionBattery.Range

Vehicle.Powertrain.TractionBattery.Charging.IsCharging

Vehicle.Powertrain.TractionBattery.Charging.TimeToComplete

Example taken from EV Charging Event Data Aggregation:
https://wiki.covesa.global/display/WIK4/EV+Charging+Event+Data+Aggregation+Project

VSS examples in GraphQL (7.2)

VSS item
Include
Concatenated path
Branch type Aggregate
Aggregate Struct
Attribute type
Sensor type Actuator type
Instances
Arrays (i.e., datatype[])
(u)intX, boolean, float, double, string
Unit Dimension
Default
allowed: [‘value1’, ..., ‘valueN’]
Overlay
min: 0 max: 100
deprecation

- We can specify
  → how the data is to be read with a `query`.
  → how the data is to be written with `mutation`.

```
query GetEVChargingData {
  vehicle {
    identification {
      VIN
    }
    speed
    powertrain {
      tractionBattery {
        netCapacity
        charging {
          chargeLimit
          isCharging
          timeToComplete
        }
        stateOfCharge {
          current
        }
        range
      }
    }
  }
}
```

```
mutation UpdateVehicleDetails($input:
VehicleInput!) {
  updateVehicle(input: $input) {
    success
    message
    vehicle {
      identification {
        VIN
      }
      speed
      powertrain {
        tractionBattery {
          netCapacity
          charging {
            chargeLimit
            isCharging
            timeToComplete
          }
          stateOfCharge {
            current
          }
          range
        }
      }
    }
  }
}
```

Example taken from EV Charging Event Data Aggregation:
https://wiki.covesa.global/display/WIK4/EV+Charging+Event+Data+Aggregation+Project

Any other added value?

Photo by J-Photos on Unsplash

Controlling the data schema AND the standard interactions (e.g., queries and mutations).

Operation ▷ Query

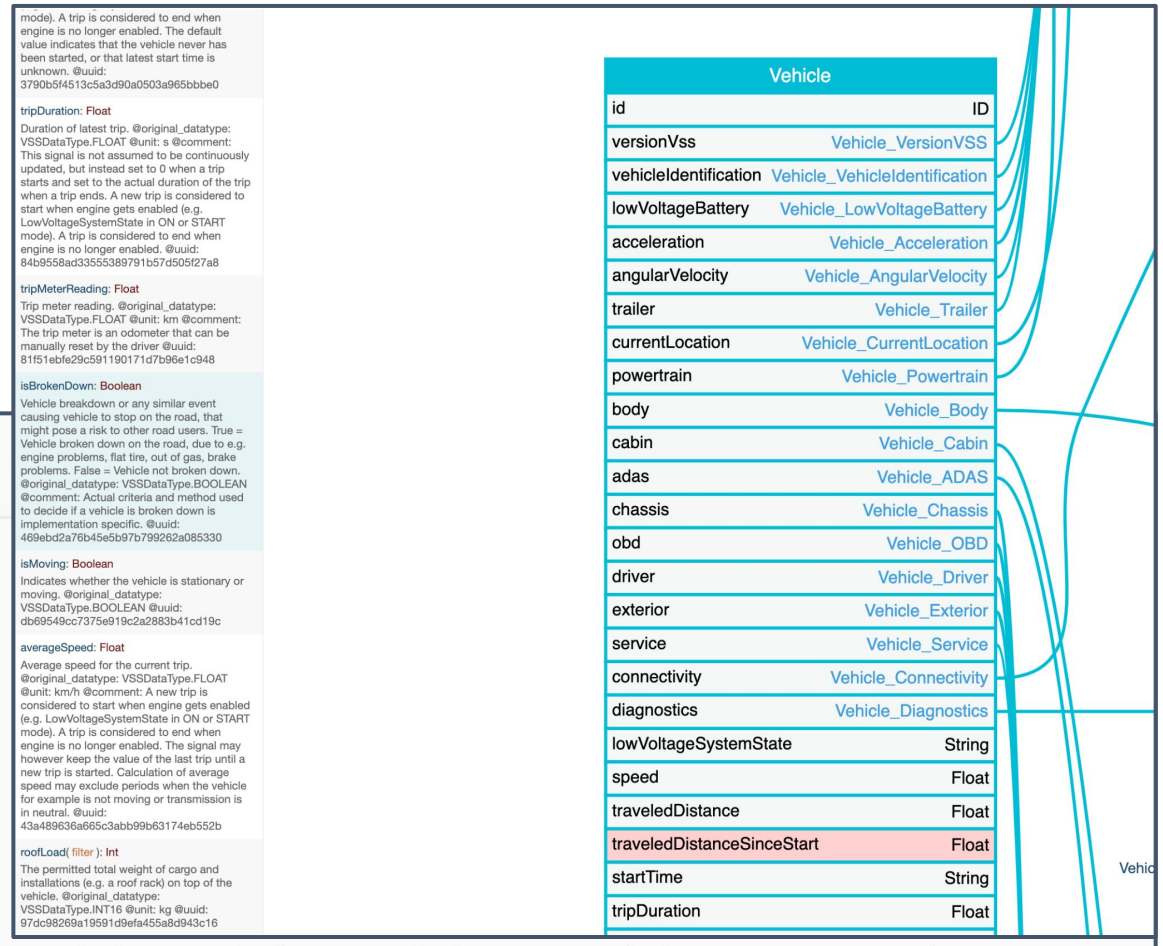
```
query Query($datasetId: ID) {
  chargingPoints(datasetId: $datasetId) {
    id
    occupancy {
      status
      vehicle {
        id
        speed
        vehicleIdentification {
          vin
          brand
        }
      }
    }
  }
}
```

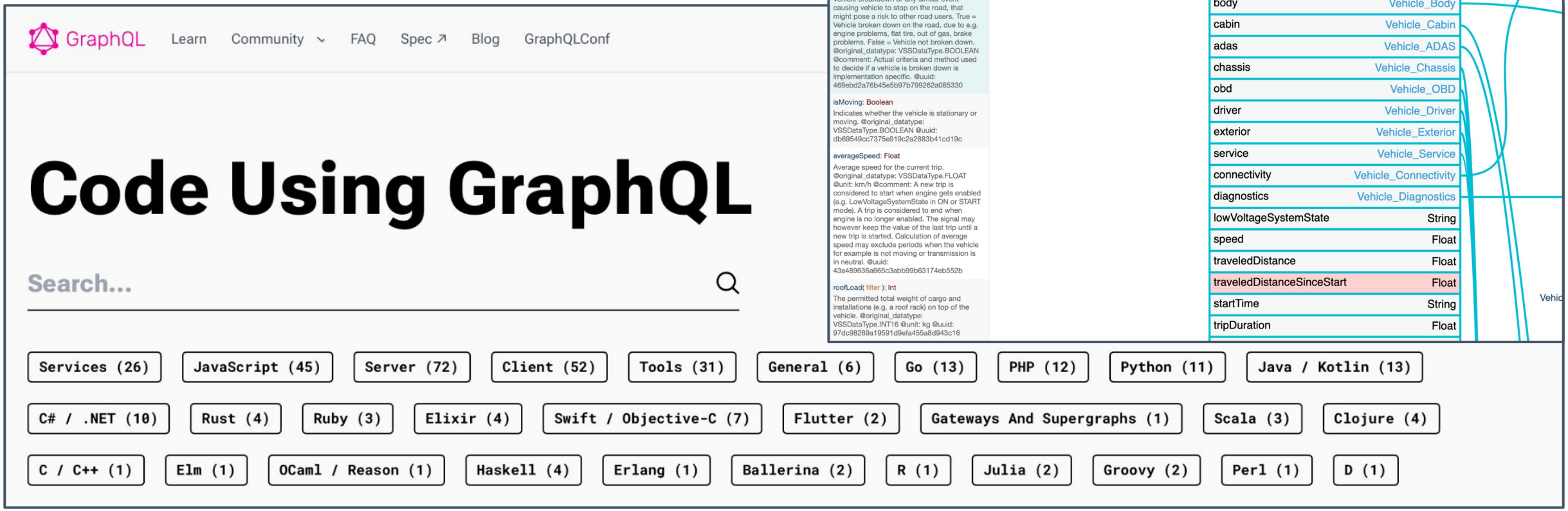
Response


```
{
  "data": {
    "chargingPoints": [
      {
        "id": "ChargingPoints/7193b760-d4b4-45bd-b466-0b6690f5e9e5",
        "occupancy": {
          "status": "OCCUPIED",
          "vehicle": {
            "id": "Vehicles/eda5e2d7-116b-4391-9e93-b43265821010",
            "speed": 0,
            "vehicleIdentification": {
              "vin": "JH4KA8172PC002873",
              "brand": "BMW"
            }
          }
        }
      }
    ]
  }
}
```


COVESA
Accelerating the future of connected vehicles

Important community and tooling available.





https://graphql.org/community/tools-and-libraries/

COVESA

Accelerating the future of connected vehicles

1 **Background**

VSS analysis and limits

Previous attempts for more expressivity

2 **Proposal**

VSS feature set correspondence in GraphQL

Other features

3 **Conclusion**

Summary

Q&A

Summary

About the current state

- VSS feature set was analyzed to uncover its actual scope.
- There are a few important limitations and misunderstandings.
- Previous attempts for expressivity thought Us some lessons.

About the proposal

- A schema language, such as the one of GraphQL, enable Us to not only cover the agreement on the meaning but also on the structure.
- Integration of domains is much more natural as a graph than as an individual tree.
- All VSS features can be realized, and many more advantages (e.g., community, tooling)

Moving from a DESCRIPTIVE (i.e., informative-only) data model…

```
Vehicle.Cabin.Door.Row1.DriverSide.Window.Position:
datatype: uint8
description: Item position. 0 = Start position 100 = End position.
max: 100
min: 0
type: actuator
unit: percent
```

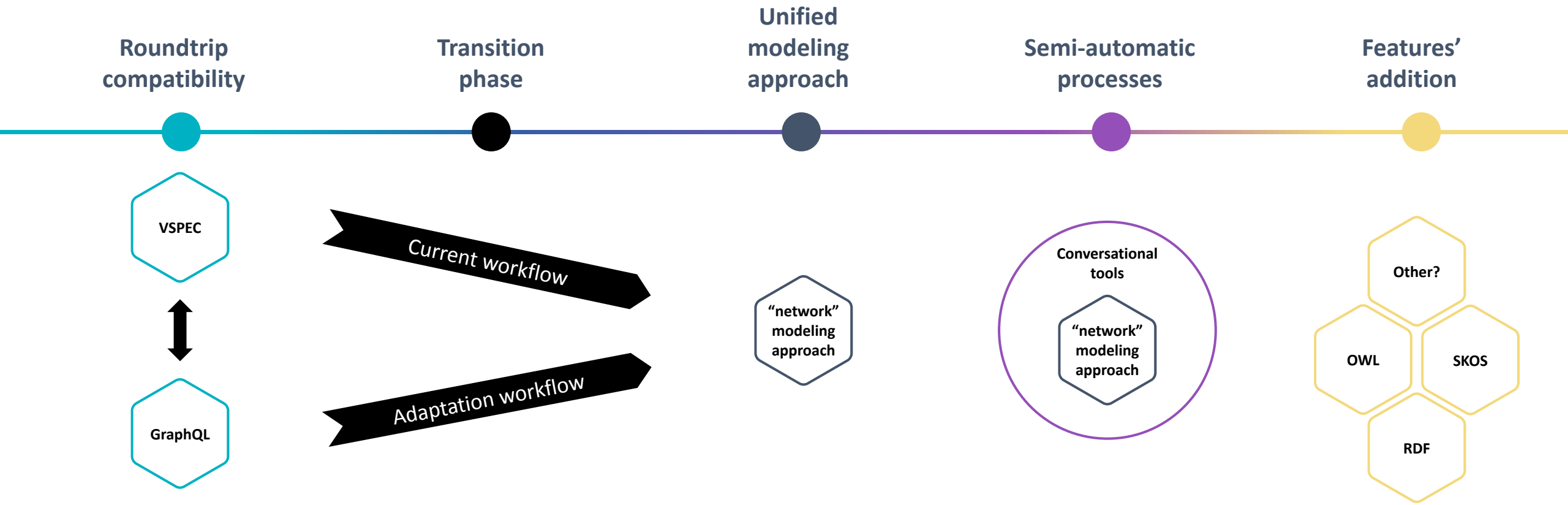
Current VSS alone has no control on how apps use the model.

…to a PRESCRIPTIVE one! (i.e., using exiting schema languages)

```
type Window {
  id: ID!
  # Represents the position of the window.
  # The position value is an integer indicating the percentage
    of closure, ranging from 0 (fully closed) to 100 (fully open).
  position: Int!
  skosConcept: String! # Pointer to unique definition
}
```

Schema representing a contract between the data producer and consumer.

What is next?


Roundtrip compatibility
Transition phase
Unified modeling approach
Semi-automatic processes
Features' addition
VSPEC
GraphQL
Current workflow
Adaptation workflow
"network" modeling approach
Conversational tools
"network" modeling approach
Other?
OWL
SKOS
RDF

COVESA

Accelerating the future of connected vehicles

Q&A