VSSo: a Vehicle Signal and Attribute Ontology for the Web of Things

Klotz Benjamin [a,b,*], Troncy Raphaël [a], Wilms Daniel [b] and Bonnet Christian [b]
[a] *EURECOM, Sophia Antipolis, France*
E-mails: benjamin.klotz@eurecom.fr, raphael.troncy@eurecom.fr, christian.bonnet@eurecom.fr
[b] *BMW Group, Munich, Germany*
E-mails: benjamin.bk.klotz@bmwgroup.com, daniel.dw.wilms@bmwgroup.com

Abstract. Application developers in the automotive domain have to deal with thousands of different signals and attributes, represented in highly heterogeneous formats, and coming from various car architectures. This situation limits the development of modern applications. We hypothesize that a formal model of car signals, in which the definition of signals are uncorrelated with the physical implementations producing them, as well as a common data layer, would improve interoperability between connected cars and their ecosystem. In this paper, we propose VSSo, a vehicle signal and attribute ontology that builds on the automotive standard VSS, and that follows the SSN/SOSA design pattern for representing observations and actuations. We also describe a more general driving context ontology supporting the description of events. VSSo is comprehensive while being extensible for OEMs, so that they can use additional private signals in an interoperable way. We developed a simulator for interacting with data modeled using VSSo is available at http://automotive.eurecom.fr/simulator/query

Keywords: Automotive, Ontology, Web of Things, SSN/SOSA, Signal, Sensor, LwM2M

1. Introduction

Automotive applications rely on the ability to manage highly heterogeneous data, coming from cars themselves or from other parties such as web services, or connected things like smart homes and smart cities. For instance, there are important opportunities in the embedded AI (Artificial Intelligence), predictive maintenance and safety-enhancing systems[1]. In this context, vehicle data needs to be interoperable in order to be handled by remote applications and services regardless of the brand, model, and internal network architecture of each connected vehicle. This is actually challenging today as a developer needs deep insights into the architecture of a vehicle[2] in order to have access and to process data coming from the vehicle sensors. In addition, information about signal metadata is needed in order to interpret the returned values. As soon as the internal architecture changes, the developer has to update the implementation and will need the same prior knowledge. This might be the case already with different models of the same brand. With an historic legacy, OEMs (Original Equipment Manufacturers) have a digitalization effort to do in order to catch up with Internet and telecommunication leaders[3].

One particularly eloquent example of such legacy is the implementation of ADAS (Advance Driver Assistance Systems) in more and more complex fashion. The early ADAS like the ABS (Antilock Braking System) and TCS (Traction Control System) existed before the 2000s to solve the problem of vehicle dynamics stabilization. The 2000s have seen developed the Adaptive Cruise Control, Lane Departure Warning and new sensors in car (radar, infrared) to include more warnings, comfort and information [1]. Future ADAS

*Corresponding author. E-mail: benjamin.klotz@eurecom.fr.
[1] http://www.visualcapitalist.com/future-automobile-innovation/
[2] http://www.ieee802.org/1/files/public/docs2013/new-tsn-diarra-osi-layers-in-automotive-networks-0313-v01.pdf
[3] https://hbr.org/2016/04/a-chart-that-shows-which-industries-are-the-most-digital-and-why

are meant to make vehicle more and more autonomous. A side effect of this development is the complexity of vehicle's electronic architecture and software system [2]. Current premium vehicles embed millions of lines of code[4], and potentially more than a hundred ECU[5] (Electronic Control Unit) while a car in 1981 could require 50,000 lines of code [3].

Finally, the vehicle's electronic architecture and software system is not only massive, it also contains thousands of parts (sensors or actuators). Wikipedia lists already about 500 of them but provides a definition for about 200[6] while all parts listed are not basic components. To produce them, OEM hire several suppliers, and the global understanding of a vehicle is therefore distributed among communities.

We notice a growing interest in using semantic technologies for addressing the challenge of defining a formal model of car signals [4]. The Internet of Things (IoT) is a novel paradigm that has been "rapidly gaining ground in the scenario of modern wireless telecommunications" [5]. Its basic idea is the "pervasive presence around us of a variety of things or objects which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals" [5]. The Web of Things (WoT) [6] is a specification from the W3C that narrows down the semantic description of Things and protocol binding so that devices from different IoT standards (OCF, OMA, Zigbee..), data formats and protocol can be operated from the same applications.

We therefore observe a gap between the need for data interoperability and the current state of the art in terms of vehicle modeling. We see a need for an ontology or an equivalent data model focusing on vehicle signals and attributes. We identify a requirement: a vehicle data model should be compliant with automotive standards such as VSS [7] (Vehicle Signal Specification) or ISO 20078 [8] and ISO 20080 [9] and follow best modeling practices from the Web of Things (WoT) in order to be used. We require such an ontology to be comprehensive enough to cover most known signals and attributes while being extensible by OEMs. This paper proposes the VSSo ontology for this purpose. In this context, we raise the following research questions:

- *How to design an ontology describing vehicle data?*
- *How to enable automotive application to interact through the Web of Things?*

Enabling automotive applications to interact through the WoT would mean to widen the range of developers that could work on it. The goal of enabling a wider range of developers means to ensure the ease of use for non automotive experts, the availability of metadata and documentation, and the reuse of well-known standards and best practices that web developers are experienced with. In order to test our proposal, we formulate the following hypothesis:

- Using the Web of Things will improve the efficiency of developing applications on vehicle data, compared to the state of the art
- Non automotive experts can use the Web of Things and the VSSo ontology more effectively than the state of the art to interact with a vehicle and build cross-industries applications.

The remainder of this paper is structured as follow. First, we extensively describe the related work in terms of automotive data access and data models, as well as the broader set of connected things using semantic web technologies 2. Next, we introduce the so-called VSSo and Driving Context ontologies, discussing their design in Section 3, how it maps to the Web of Things in Section 4. We evaluate the VSSo usage in Section 5, and we show how this ontology can be used and consumed in Section 6. Finally, we conclude and outline some future work in Section 7.

2. Related Work

2.1. Automotive data access

There are flourishing means to access vehicle data ranging from OEM-specific implementations to new standards which are arising. In this section, we extensively described the various mechanisms that have been proposed so far.

2.1.1. OEM APIs and Web services

Despite the complexity of modern vehicles, there is a trend of publishing understandable Web services and APIs by OEMs, that enable to access to specific vehicle signals and use tree structures to represent car data. For example BMW uses the platform *If This Then That* (IFTTT) to expose vehicle data in order

[4] https://futuremonger.com/100-million-lines-of-code-4-tb-data-per-day-is-that-your-next-car-a2724e9bd3fa

[5] https://www.techopedia.com/your-car-your-computer-ecus-and-the-controller-area-network/2/32218

[6] https://en.wikipedia.org/wiki/List_of_auto_parts

to connect with a wide range of Web Servicesfoot-notehttps://ifttt.com/bmwlabs. These simple tree structures can be related to VSS [7] which is further detailed in Section 2.2.

With the OEM APIs come automotive-specific data models. For instance Mercedes-Benz released four APIs[8]:

- Car Configurator API that describes vehicles options and configurations;
- Dealer API that describes car dealers;
- Remote Diagnostic Support that retrieves Diagnostic Trouble Codes (DTC);
- Vehicle Image API that provides images of components.

They also announced a Connected Vehicle API that would read different states of the car (tire, location, odometer, fuel, doors, battery). It currently interacts only with records. For the 2018 OI Competition[9], Porsche released APIs for 140 data sources for simulated sports cars including a race API (vehicle dynamics signals), charging, offroad (suspensions), chassis settings, light and weather conditions APIs. It is now being standardized under the *High Mobility* platform[10]. This API is made for a simulator.

PSA Group has realeased two APIs[11]:

- Connected Car Development, with 89 signals. Certain signals are only readable while other are also writable. The signals are clustered in categories: crash, eco-driving, environment, maintenance, place, referential, running, safety, trip, vehicle.
- Eligibility API, to check if a vehicle is eligible for remote management.

The Connected Car Development API mostly focus on signal that do not change much over time and space, as for instance the reference fuel price. As part of it Next Generation Infotainment system (NGI), General Motors released several commercial APIs[12] dealing with the navigation and infotainment system, dynamic signals as well as about 400 data points[13] to cover use cases such as hard braking or display alerts. Ford has incorporated the AppLink technology [10] in its vehicles[14] which allows the integration of mobile apps from smartphones with the Human-Machine Interface (HMI) of the connected vehicle. Its APIs concern the dashboard and steering wheel buttons as well as facial and vocal recognition. Toyota is developing a Mobility Services Platform[15] (MSPF) to make their future vehicle fit for sharing within the scope of smart cities. This platform will not only focus on the identification and payment aspects of mobility, but also fleet management and vehicle unlocking from smartphones. Volkswagen released a Car Configurator Web API[16][17] that focuses on the configuration management of a vehicle.

2.1.2. Other automotive standards

There are also several automotive standards that enable to access to vehicle data. We describe the most notable examples in the following paragraphs.

OBD-II. On-board diagnostics (OBD) are vehicleś self-diagnostic and reporting capabilities. Its main usage consist in providing a vehicle owner or repair technician access to the status of the various vehicle subsystems. Current OBD implementation uses a standardized digital communications port to provide real-time data in addition to a standardized series of diagnostic trouble codes (DTC) identifying malfunctions.

OBD-II is a set of standards[18] specifying the type of diagnostic connector and its pinout, the electrical signalling protocols available, and the messaging format. In addition, it provides a list of vehicle signals to monitor and a way to encode the data for each of them. They are referred by their Parameter ID (PID) to cross-reference between the pinouts of electronic components and their functions. There are 10 diagnostic services described in OBD-II[19]:

1. Show current data;
2. Show freeze frame data;
3. Show stored Diagnostic Trouble Codes;
4. Clear Diagnostic Trouble Codes and stored values;
5. Test results, oxygen sensor monitoring (non CAN only);

[7] https://github.com/GENIVI/vehicle_signal_specification/
[8] https://developer.mercedes-benz.com/
[9] http://www.porsche-next-oi-competition.com/
[10] https://high-mobility.com
[11] https://developer.psa-peugeot-citroen.com/inc/
[12] https://developer.gm.com/vehicle-apis
[13] https://techcrunch.com/2017/01/26/gms-new-sdk-for-in-car-infotainment-apps-offers-access-to-nearly-400-data-points/
[14] https://developer.ford.com/pages/applink
[15] https://corporatenews.pressroom.toyota.com/releases/toyota+launches+new+mobility+ecosystem+concept+vehicle+2018+ces.htm
[16] https://productdata.vwgroup.com/overview.html
[17] http://udc-configurator.volkswagen.nl/
[18] https://www.outilsobdfacile.fr/norme-communication-obd.php
[19] https://www.sae.org/standards/content/j1979_201009/

6. Test results, other component/system monitoring (test results, oxygen sensor monitoring for CAN only);
7. Show pending Diagnostic Trouble Codes (detected during current or last driving cycle);
8. Control operation of on-board component/system;
9. Request vehicle information;
10. Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs).

The main benefit of OBD-II is that the physical and digital interface is standard for vehicles regardless of their brands.

Extended vehicle standard. The Extended Vehicle is a set of ISO standards [11] that were initiated in 2014 and are still under development. The purpose of the Extended Vehicle is to abstract vehicles from its physical form and to interact with its extended interfaces [12]. These interfaces are being standardized in [8], on similar way as OBD-II. They also define a basic data model, further described in specific use cases [9] like identify ECUs, read DTC, read readiness codes, read DTC snapshot data, read diagnostic parametric data, read malfunction indicator status, clear DTCs, adjust system settings, activate actuators or activate a self-test routine[20].

The Neutral Vehicle. With the similar goal of exposing vehicle data, the Neutral Vehicle [13] platform[21] aims at being a standard combining automotive specificity and neutral servers. The Neutral Vehicle platform provides an end-to-end framework for exchanging vehicle data between physical cars and the cloud with access from third parties. The platforms aims at providing security, scalability and interoperability to enable the development of future advanced applications by third parties. Its data access reuses OBD-II, data link devices, ECU readers and other connected devices. Its core is a neutral server providing neutral data exchange between all parties.

VISS. The Vehicle Information Server Specification [14] (VISS) is a vehicle server specification, currently a candidate recommendation from the W3C. This enables a client to GET or SET vehicle signals and data; to SUBSCRIBE to receive notifications and to UNSUBSCRIBE from receiving notifications. Its data model is per default VSS. This allows, among others, to take advantage of the extension mechanism from VSS to include more signals and attributes. The VISS also describes a discovery mechanism that defines the set of signals and data that a client can access at a particular point in time. Its interface is being specified in the Vehicle Information API Specification [15] (VIAS).

2.2. *Data models in the automotive domain*

There are important similarities between most data models developed by OEM and the structure of the Vehicle Signal Specification[22] (VSS). Its tree structure was originally specified by the GENIVI Alliance and W3C. The specification states, for example, that the speed measured by the GPS can be accessed by going through a tree from the *Cabin*, the *Infotainment* and then the *GPS* branches to finally reach the *Speed* signal. Hence, car signals data is accessed within some context (the branch of the vehicle that generates it). VSS is the building block of VISS [14] and VIAS [15]. However, its structure does not solve entirely the issue of interoperability. Indeed, with a huge amount of sensors embedded in most modern cars, many of them can be still obscure to non automotive experts and rely on non standard units. Therefore, the knowledge on how to interpret values should also be represented. A single API for all vehicles, for instance, would make implementations very complex as soon as the unit system changes. In addition, the tree structure of VSS creates a lot of redundancy. For instance, about 63% of VSS signals are about seats[23] because VSS considers 25 potential seats positions.

There are also data models that include semantic metadata in the automotive domain. Many ontologies have been developed in order to model specific use cases in the automotive domain. In 2003, [16] proposed an ontology-based data access for vehicles. [17] describes the relationship between components, failures and their symptoms. [18] proposes an automotive ontology describing the user's actions and car context. More generally, several research projects proposed ontology-based representations of some vehicle context to provide advanced driver-assistance systems

[20]Current draft available: https://www.iso.org/obp/ui/#iso:std:iso:20080:dis:ed-1:v1:en

[21]https://neutralvehicle.com/

[22]https://github.com/GENIVI/vehicle_signal_specification/

[23]In https://github.com/GENIVI/vehicle_signal_specification/blob/master/vss_rel_1.0.vsi there are 707 concepts relating to seats out of 1110 concepts

(ADAS) [19, 20, 21, 22, 23], but they are not complete or extensible, nor they are automotive standards.

schema.org is also a *de facto* standard [24] for marking up web pages with structured metadata to facilitate Web search. There are extensions dedicated to the IoT[24] and the automotive[25] domain. They are still under development, notably for aligning some WoT concepts with other ontologies like SSN/SOSA [25, 26], and currently, they only define a small set of static properties describing cars. The automotive extension comes from the work of the W3C Automotive Ontology Working Group[26] which started with the goal of describing cars in e-commerce. It is based on a number of ontologies that describe cars' attributes and configuration in the e-commerce:

- Car option ontology[27] for the commercial aspects of offers for sale or rental. It contains 12 classes and 19 properties.
- Vehicle sales ontology[28] (VSO) for describing cars, boats, bikes, and other vehicles for e-commerce with 33 classes and 54 properties.
- Used cars ontology[29] for describing aspects of used cars for e-commerce with 22 classes and 46 properties.
- Volkswagen Vehicle Ontology[30] for describing Volkswagen-specific features of automobiles with 30 classes and 50 properties. Its interest is limited to the domain of the e-commerce for one brand.

However, they do not include sensors or signal data.

2.2.1. *Modeling requirements*

A suitable connected car signal data model should enable a web developer to query and extract knowledge from a set of vehicle signal data stream and static database with no deep expertise of the automotive domain. In this section, we define a set of competency questions, which we will later use as a mean of evaluating the produced data model.

Such a data model should be generic and be suitable for all automotive domains and use cases. For example, the e-commerce and configuration management is already well-known with schema.org, while the diagnosis domain is well-studied [17] but not accessible for web developers. In addition are the domain of telematics [13] and the seamless experience in regard to smart cities and smart homes for instance.

The e-commerce requires to answer questions like *What is the model of this car?* or *How old is this car?*. The diagnosis and maintenance domain would have to provide answers for questions like *What type of transmission does this car have?* or *How many different speedometers does this car contain?*. Telematics service providers would query current signals, such as *What type of fuel does this car need?* or *What is the current gear?*. For seamless experience, an application developer would ask *What are the destination coordinates?* or *What is the local temperature on the driver side?*. Such competency question can be clustered in function of the type of information requested: static attributes of vehicles, static description of car signals, and dynamic values of signals.

Description of car attributes. There is a need to define a number of static properties or attributes describing either a complete vehicle or its parts (later named branches), such as the engine, and their position.

- What are the attributes of a car and what do they express?
- How many attributes does a car have?
- What is the model of this car?
- What is the brand of this car?
- What is the VIN of this car?
- When was this car produced?
- What are the dimensions of this car?
- What type of fuel does this car need?
- What type of transmission does this car have?
- What are the characteristics of this engine?
- How many doors does this car have?
- How many seats does this car have?
- On which side is the steering wheel located?

Description of car signals. A car contains numerous sensors that produce signals. Here is a list of competency questions that should return metadata about a vehicle signal: its host branch, its sensor or actuator, its unit system or its position in the vehicle.

- Is there a signal measuring the steering wheel angle?
- Which signals are actuatable?
- Which signals are both observable and actuable?
- How many sensors does this car contain?
- How many different speedometers does this car contain?

[24] iot.schema.org
[25] auto.schema.org
[26] http://www.automotive-ontology.org/
[27] http://semanticweb.org/wiki/Car_Options_Ontology.html
[28] http://www.heppnetz.de/ontologies/vso/ns
[29] http://ontologies.makolab.com/uco/ns.html
[30] http://www.volkswagen.co.uk/vocabularies/vvo/ns

– Which part of this car produces a signal of type `vss:LongitudinalAcceleration`?
– Which signals measure a temperature and in which part are they located in the car?
– What unit type does the signal `vss:VehicleYaw` use?
– What are the characteristics of the sensor producing the signal "TravelledDistance" in the OBD branch?
– What are the maximum values allowed for all signals from a Vehicle?

Description of dynamic car states. When being used, car sensors will generate a lot of values that depend on time and space. One should be able to query the current values of the signals as well as past historical ones. This leads to additional competency questions.

– What is the current gear?
– What are the values of all signals representing the speed of this car in this moment?
– Which windows are currently open?
– What is the local temperature on the driver side now?
– What are the current values of signals defining the driver seat position?
– When was the last time the speed was over 100 km/h?
– When and where was the last time the driver's door was unlocked?
– What was the maximal speed reached by the car?

2.3. *Connected Things on the Semantic Web*

The Semantic Web would tackle the connected vehicle the same way it would do with connect things from other domains. Thus, standards and best practices are developed for domain-independent connected Things.

2.3.1. *Ontologies for connected Things*

W3C and OGC have developed standards for defining systems with their signals. The Semantic Sensor Network[31] (SSN) ontology [25] is an ontology for describing sensors and their observations, the involved procedures, the studied features of interest, the samples used to do so and the observed properties, as well as actuators and actuations. SSN follows a horizontal and vertical modularization architecture by including a lightweight but self-contained core ontology called SOSA[32] [26] (Sensor, Observation, Sample, and Actuator) for its elementary classes and properties, that was released in October 2017. Both SSN and SOSA are domain independent. There are applications built using them in different domains including satellite imagery, large-scale scientific monitoring, industrial and household infrastructures, social sensing, citizen science, observation-driven ontology engineering, and the Web of Things (WoT) [25].

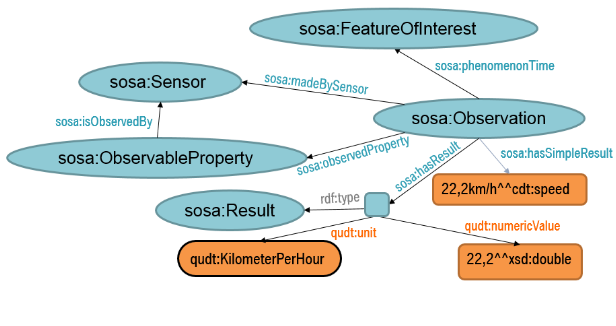

Fig. 1. The SOSA modeling pattern for sensors and observable properties

We hypothesize that the generic modeling patterns defined in the SSN/SOSA ontology [26] are adequate to describe observations and that an additional vocabulary is needed to define the specific terms in the automotive domain.

2.3.2. *The Web of Things*

For interacting with heterogeneous systems following different standards in the Internet of Things (IoT), we look at the solution proposed by the Web of Things (WoT) at the W3C [6, 27, 28]. Started in 2016, the goals of the WoT Working Group[33] are to allow the discovery, sharing, composition and reuse of connected physical devices in a web layer and, therefore, counter the fragmentation of the IoT[34].

The WoT Working Group has split its activities into four main categories:

1. WoT Things Description (TD) Specification, defining the description of metadata and interaction of Web Things,
2. WoT Scripting API Specification, defining a Web Thing API as well as discovery mechanisms,

[31] http://www.w3.org/ns/ssn/

[32] http://www.w3.org/ns/sosa/

[33] https://www.w3.org/WoT/WG/

[34] https://webofthings.org/2016/01/23/wot-vs-iot-12/

3. WoT Binding Templates Specification, providing solution to include Web Things from different standards of the IoT,
4. WoT Authentication and security.

At the heart of the specification is the WoT servient [27]: an entity consisting of a Web client, a Web server and device control capabilities. It is essentially a *virtual device* which provides access, controls and get statuses from physical IoT devices.

The W3C Web of Things WG presented a few use cases of servients including one about a connected car[35] as visible in Figure 2, with WoT-based services running in the back-end of the connected car. The collection and analysis is deployed to a fleet of cars to determine traffic patterns. In this use case, after a discovery phase of car components through a connection gateway, the WoT servient collects data pushed from car components and allows services to access car components through its WoT interface. There have been early implementation [29] of this use case.

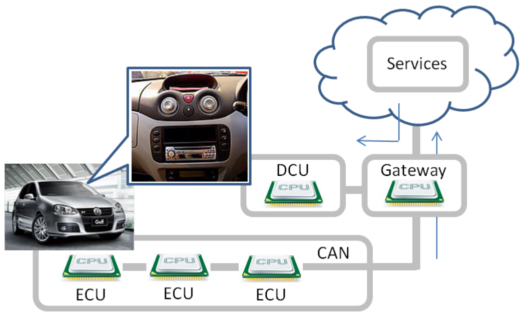

Fig. 2. W3C Web of Things use case: a connected car with a cloud server defined in [27]

This example shows the main benefit of WoT for the automotive domain: it would allow the decorrelation from automotive standard for car data and, therefore, allow developers who are not automotive experts to use WoT interaction patterns with vehicles as Web Things. It also would enable the collection and analysis of sensor data coming from vehicles of different models and brands. We are using WoT in our research to benefit from WoT interactions and be able to combine them in a common web layer.

In the Thing Descriptions (TD), annotations about Things, capacities and interactions are semantic annotations. The WoT ontology [28] defines those terms. In this ontology, a `wot:Thing` implements a `wot:Security`, defined as its security mechanism, and a number of `wot:InteractionPattern` that can be subclassed as `wot:Property`, `wot:Action` and `wot:Event`. Their instances are the interactions associated with a Thing, and are defined by a `wot:Link` and `wot:CommunicationProtocol` to access the device, and `wot:DataSchema` for their input/output. In addition to that wot:Property instances can have a property `wot:isMeasuredIn` to define a `om:Unit`[36].

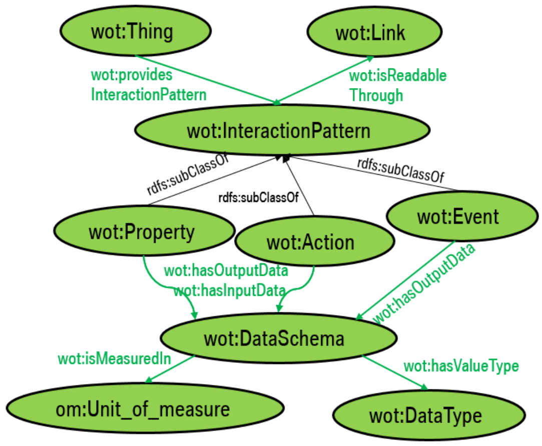

Fig. 3. A set of WoT classes, as visible in a TD

3. VSSo and Driving Context Ontology

We developed VSSo, a vehicle signal ontology based on the GENIVI and W3C standard data model VSS (Vehicle Signal Specification). This ontology and its documentation are available at http://automotive.eurecom.fr/vsso. The version v1.12 of VSSo contains 496 classes, 84 object properties and 59 datatype properties. It has a Description Logic expressivity of $\mathcal{ALUHOI}+$, and it is interlinked with the auto.schema.org properties.

3.1. VSS

The Vehicle Signal Specification defines a tree containing 452 *Branches*, 59 *Attributes* and 1062 *Signals* that aim to represent car data (Figure 4). The specification states that:

[35] https://w3c.github.io/wot-architecture/#connected-car

[36] http://www.wurvoc.org/vocabularies/om-1.8/Unit_of_measure

- *Branches* are car parts or components. They are represented as nodes in the VSS tree. Branches can contain other branches or signals and attributes. For instance the top branches in the VSS tree are *Body*, *ADAS*, *Cabin*, *Chassis*, *Drivetrain*, *OBD* and *Vehicle*.
- *Attributes* are the static information about a car that should not change over time and space. Attributes are represented as leaves in the VSS tree. For instance, the dimensions and VIN (Vehicle Identification Number) of a car are attributes of the *Vehicle* branch. Attributes are defined by a path starting with "Attribute" and defining its position in the VSS tree. For instance the VIN is `Attribute.Vehicle.VehicleIdentification.VIN`. They also have entries such as a description, a type, a unit or restrictions on values. All properties defined in http://auto.schema.org are attributes in VSS.
- *Signals* are the dynamic information about a car that is either produced by a sensor, consumed by an actuator or properties of complex embedded systems. Signals are also represented as leaves in the VSS tree. For instance, `Signal.Drivetrain.Transmission.Speed` is the car speed, measured in the *Transmission* branch. Signals, like attributes, have entries providing a description, a type, and potentially a unit and restrictions on values.

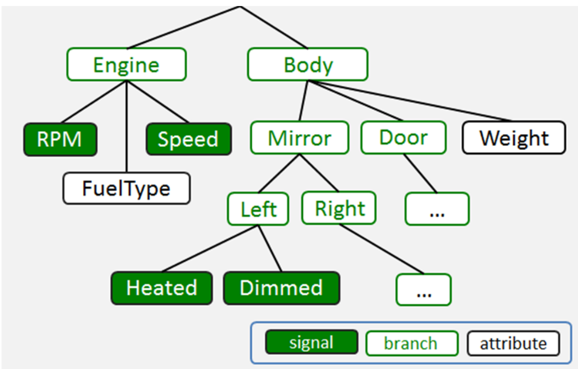

Fig. 4. The GENIVI Vehicle Signal Specification structure

In its original form, VSS did not contain information about sensors or actuators producing or consuming data. In order to describe the difference between signals measuring the same phenomenon, but sensed by different sensors, such as the car speed, we added new entries in VSS signals. We also corrected some entries to make VSS more consistent, especially in the naming convention and choice of standard units. Those corrections have been approved by GENIVI and are now part of evolution of this standard. VSS is meant to be a technology-independent specification for car data. This means that a component or signals specific to a particular brand or car model should not define a specific technology as other competing ones exist to do the same task. For instance, the traveled distance is measured by an *Odometer* regardless of the technology used.

3.2. General modeling pattern

The general idea behind the design of the VSS ontology is to take advantage of the structure of VSS. All branches are part of a complete tree, as sub-branches of bigger branches. This structure gives a more understandable meaning to signals. Therefore, we reuse it in a component-based pattern using subclasses of `vsso:Branch` linked with the transitive object property `vsso:partOf`. This means that a VSS *Branch* is used to generate a new class, and the mother or children branches are attached to it with a `vsso:partOf` property (Listing 1.1).

Listing 1: `vsso:Drivetrain` is an example of a generated class part of `vsso:Vehicle`

```
vsso:Drivetrain a rdfs:Class, owl:Class;
    rdfs:subClassOf vsso:Branch;
    rdfs:subClassOf [
        a owl:Restriction;
        owl:onProperty vsso:partOf;
        owl:allValuesFrom vsso:Vehicle
    ];
    rdfs:label "Drivetrain"@en;
    rdfs:comment "Drivetrain. All body components"@en.
```

The second interesting structural aspect of VSS is the set of entries defining VSS concepts. Indeed, attributes, branches and signals are all defined by at least a name, a type and a description. These entries allow the generation of one class per VSS concept, with a RDFS label and comment. Attributes and signals also have additional entries, such as a unit, or a set of potential values (sometimes a minimum and maximum values) and a sensor or actuator. All these entries define the specific details of an attribute or signal, and make more sense to a machine than a label or description (Listing 1.2).

Listing 2: `vsso:AmbientAirTemperature` is a signal measured by a `vsso:Thermometer` in `qudt:DegreeCelcius`

```
vsso:AmbientAirTemperature a rdfs:Class, owl:Class;
    rdfs:subClassOf vsso:ObservableSignal;
    rdfs:label "AmbientAirTemperature"@en;
    rdfs:comment "Signal.Vehicle.AmbientAirTemperature :
        Ambient air temperature"@en;
    rdfs:subClassOf [
        a owl:Restriction;
        owl:onProperty sosa:isObservedBy;
        owl:allValuesFrom vsso:Thermometer
    ];
    rdfs:subClassOf [
        a owl:Restriction;
        owl:onProperty qudt:unit;
        owl:allValuesFrom qudt:TemperatureUnit
    ].
```

We generate a datatype property for each VSS attribute which are sub-properties of a generic `vsso:attribute` datatype property. All those attributes being static, since their values do not evolve in time and space, there is no need to model them using a pattern involving dynamic observations. VSS attributes are attached to VSS branches which is materialized in the domain of those properties, while their range makes use of a custom datatype (Listing 1.3).

Listing 3: `vsso:tankCapacity` is an attribute of the `vsso:FuelSystem` branch

```
vsso:tankCapacity a owl:DatatypeProperty;
    rdfs:subPropertyOf vsso:attribute;
    rdfs:label "TankCapacity"@en;
    rdfs:comment
        "Attribute.Drivetrain.FuelSystem..Tank.Capacity:
        Capacity of the fuel tank in liters"@en;
    rdfs:domain vsso:FuelSystem;
    rdfs:range cdt:volume.
```

Signals, however, are going to be observed over time and space and there is a need for an adapted modeling pattern taking dynamics into account. In order to model it, we take advantage of the SSN/SOSA pattern for modeling sensors, actuators, observable and actuatable properties, observations and actuations. SOSA uses the triplets (*Observation, ObservableProperty, Sensor*) and (*Actuation, ActuatableProperty, Actuator*) where the first element defines the abstraction data, the second the signal and the third the appliance producing or consuming the data. Observations and Actuations contextualize the data with properties such as `sosa:FeatureOfInterest` (e.g. a car), the `sosa:Result` or `sosa:SimpleResult` depending on how units are defined, as well as `sosa:phenomenonTime` and `geo:lat`, `geo:long` for the spatiotemporal context of the observation or actuation (Figure 1).

SSN/SOSA does not define a unique unit ontology, but it is open to use multiple ones. The examples provided in the specification[37] use QUDT[38][30], OM[39][40] [31] or a custom datatype[41] [32]. The main unit ontologies have been compared in [33]: OM is the largest one with relatively few issues, while QUDT is a medium sized ontology with some inferential inconsistencies but a partial mapping with schema.org[42]. The authors of [32] have developed a custom datatype supporting the units from the UCUM (Unified Code for Units of Measure) system[43]. In order to remain open, we only set restrictions on unit systems in QUDT and let the user choose the units freely.

3.3. *Modeling problems and new VSS policies*

Several exceptions and issues prevent the trivial generation of a proper ontology from VSS. Some concepts share the same name or require clarification, signals must be compliant with the SOSA pattern and there are branches defining position concepts that are not relevant for a VSS ontology.

3.3.1. *Clarification of concept names*

Homonymy. VSS relies on a full path to define an attribute or a signal. Usually, the path contains all the context to be interpreted as generic name. For instance `Signal.Drivetrain.Engine.Speed`, is clearly the rotation speed of the engine while `Signal.Cabin.Infotainment.Navigation-.CurrentLocation.Speed` is the vehicle speed measured by the GPS. However they would both generate a class `vsso:Speed` if we would take the leaf of the tree as a basis. Therefore, VSS concepts are renamed for clarification. In the same example, they will generate `vsso:EngineSpeed` and `vsso:VehicleSpeed`.

Synonymy. Sometimes, two different path in the VSS tree actually refer to the same concept. This happens when the same phenomenon can be measured by more

[37] https://www.w3.org/TR/vocab-ssn/#examples
[38] http://www.qudt.org
[39] Version 1.8.6 from http://www.wurvoc.org/vocabularies/om-1.8/
[40] Version 2.0.6 from https://github.com/HajoRijgersberg/OM
[41] https://ci.mines-stetienne.fr/lindt/v2/custom_datatypes.html
[42] https://www.w3.org/TR/2016/NOTE-tabular-data-primer-20160225/#units-of-measure
[43] http://unitsofmeasure.org/ucum.html

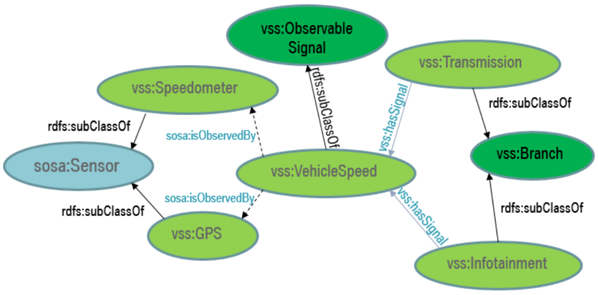

Fig. 5. Two signals representing the same concept of vehicle speed

than one sensor or in different parts of the vehicle. For example, both signals `Signal.Drivetrain.Transmission.Speed` and `Signal.Cabin.Infotainment.Navigation.CurrentLocation.Speed` measure the speed of the car, one in the gearbox using rotation speed measures and the other one using GPS coordinates. The transformed `vsso:VehicleSpeed` class is unique in VSSo. Its instances differ given the sensors that will produce the value and the branch hosting the sensor (Figure 5).

3.3.2. *Restriction required by the SOSA pattern*

In SOSA, there is no definition of specific signals but only `sosa:ObservableProperty` and `sosa:ActuatableProperty`. These classes are not mutually exclusive but simply define signals that are meant to be read or written.

Signals are observable, actuatable or both. We define two main signal classes in the VSS ontology: `vsso:ObservableSignal`, as a subclass of `sosa:ObservableProperty`, and `vsso:ActuatableSignal` subclass of `sosa:ActuatableProperty`. All signals in VSS are subclasses of at least one of them. For instance, `vsso:VehicleSpeed` is only measured and is therefore a subclass of `sosa:ObservableProperty`, while `vsso:MirrorHeating` only acts on the mirror and is a subclass of `sosa:ActuatableProperty`. Many signals are subclasses of both. The choice of making a signal observable or actuatable is based on the existence of the sensor and actuator entries of each VSS Signal. If it has a sensor, it is observable, but if it has an actuator, it is actuatable.

All signals have at least a sensor or actuator. In order to be compliant with SOSA, we must define a `sosa:Sensor` for all `sosa:ObservableProperty` and a `sosa:Actuator` for all `sosa:ActuatableProperty`. This means that the entries we added in VSS to define those devices are used to create classes, subclasses of either `sosa:Sensor` or `sosa:Actuator`. These sensors and actuators should be as technology-independent as possible, as their physical instances vary from one OEM to another. Some signals relate to complex systems such as the infotainment system where there are no physical sensors or actuators. In this case, a virtual system defines the sensor/actuator producing or consuming the data without being a physical device.

3.3.3. *Branch structure modeling choices*

The VSS tree structure contains choices that prevents an automatic generation of RDF classes for branches.

All branches are `vsso:partOf vsso:Vehicle`. The path defining attributes and signals begins with the top element of the tree, being either “Attribute” or “Signal”. The modeling choice would require the top branch to be the complete vehicle that contains all branches. There is, nevertheless, a branch among the top one called “Vehicle” containing attributes and signals about the full vehicle, such as its VIN. We take this branch as the top one containing all other branches (Figure 6).

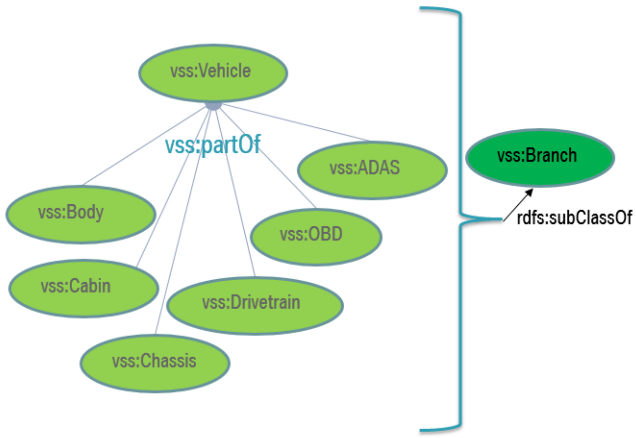

Fig. 6. The `vsso:Vehicle` branch is taken as the one containing all other branches and corresponds to the full vehicle

Position-related concepts are not branches. In VSS, the path to certain attributes and signals contains the position of certain branches. This is especially the case for elements that exist multiple times within one car, such as doors, seats and mirrors. For instance, there are signals like `Door.Left.IsLocked` and `Mirror.Right.Tilt`. It is not desired to have classes defining the concepts of “left” and “right”. A solution could be to define a class per signal in VSS, but the result would not be consistent with the goal of having generic signal classes. Instead, we decide to model the hosting branches with an object property `vsso:position`. This defines instances of such branches with the correct positions and still refer to a unique class. Using the same example, a door instance would have `vsso:position vsso:Left` and the mirror instance a `vsso:position vsso:Right`.

3.4. The driving context ontology

VSSo models signals and attributes of vehicles. However, automotive data generally refers to a much broader set of domains. One can cite, for example, the description of trajectories, points of interest, the behavior and mental state of the driver, or even the weather and the road states. Such automotive domains are depicted in Figure 7

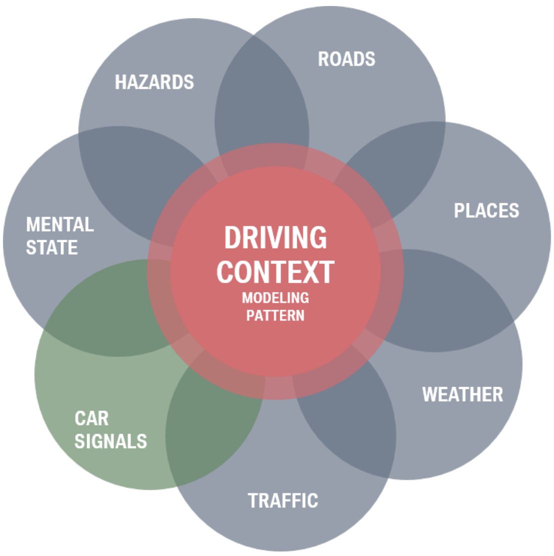

Fig. 7. The driving context model encapsulates numerous factors related to the driver, the environment and the vehicle itself

3.4.1. Driving context model

VSSo, based on the SSN/SOSA modeling pattern, describes signals as observable or actuatable properties of vehicle branches which are feature of interests. The main limitation of this model is that it describes signals as states or properties of its features of interest. In contrast, the Web of Things proposes three main interaction patterns: properties that one can read and write, actions that one can invoke, and events that one can observe. We will see in Section 4 how we can bind our automotive-specific data model with the Web of Thing, which requires an additional event-based modeling pattern.

Another motivation for more generic contextual driving data is the large number of domains interacting with vehicles. There has been research on multiple domains which resulted on new data models or ontologies [19, 34, 35, 36, 37, 38]. The following models are of interests to model a driving context:

- Car signals and attributes
- Driver/passenger emotions and mental state
- Driver/passenger behavior
- Road state and close events
- Area state and close Points of Interest (PoI)
- Weather

Based on the existing work on those different domains and the resulting models, we created a central pattern for making these domains interact one with another. Our model focuses on three central concepts: the driver, the car and the road on which it is driving. Those central features of interest can be further detailed but are always described either in a state-oriented or event-oriented pattern. Their contextual features are either their state, such as a mental state of a driver, or events they are involved in, such as an accident involving cars and their passengers.

A state is a class that refers to a certain observation of a property of a feature of interest. The state pattern is here, as for VSSo, the SSN/SOSA pattern. The different states we have identified are the car signals, the weather state, the emotional and behavioral state of the driver and passengers and the states of a spatial region, including the roads and local area.

To be compliant with current best practices in event modelling for a driving context, we use a popular event ontology [39]. We have a need to describe events as classes, with object properties linking them to their participants - mostly cars and people - as well as the potential result of the event. Furthermore, we need an event composition pattern, to create sub-events, and we want to use the same modeling of time and locations as in SSN/SOSA. With its simple pattern and its reuse of FOAF, OWL-Time and the WGS84 Geo Positioning Ontology, the Event ontology is the most adapted choice.

Based on this central pattern, we created the driving context ontology. It is composed of four main concepts:

- The vehicle and all its signals and attributes imported from the VSSo ontology
- Driver and passengers and their mental and behavioral States
- The road and surroundings state and events
- The trajectory and its sequences and points of interest

3.4.2. Driving context ontology

The driving context core ontology is centered on the `:State` class. Its subclasses are the `:WeatherState`, `:EmotionalState`, `:BehavioralState` of occupants, `:SpatialRegionState` and VSS signals.

The `:WeatherState` is aligned to the `wo:WeatherState` in the Weather ontology[44] [35] and `seas:WeatherForecast` in the SEAS weater ontology[45]. These ontologies have different approaches to describe a weather state with several subclasses or literal description.

The driver and passenger State model first describes their roles. `:Driver` and `:Passenger` are mutually exclusive roles of a `foaf:Person`, linked with a `:role` object property. A `foaf:Person` has then independently a `:MentalState` and a `:BehavioralState`. These states are mapped to the MFOEM Emotion ontology[46] [37] and SIO ontology[47] [36] that contain extensive subclasses of respectively affective and bodily processes, and emotions and interactions.

The `:SpatialRegionState` has two subclasses: `:RoadState` and `:AreaState`. `:RoadState` is aligned to `tti:RouteOfTransport` in the Toyota TTI ontology[48] [38], and `ro:RoadSystem` in the Road Ontology[49].

4. Automotive Web Things

In order to take advantage of the best practices of the WoT specification to interact with VSSo data, we first need to ensure that the models are compatible. Next, we need to bind it to a certain access and communication protocol.

4.1. VSSo-WoT modeling patterns

Web Things are defined in their Thing Descriptions, which are based on the WoT ontology [28]. They are usually serialised in JSON-LD using the WoT ontology as main context. For instance a car, defined as a `wot:Thing` could have an interaction to control a window. It is an instance of `wot:Action`, that expects an input. It has another interaction to read its speed value. In this case, it is instantiated as a `wot:Property` and expects a `wot:DataSchema` output as well as a unit.

With VSSo, we apply the following matching rules. All `vss:ObservableSignal` and `vss:ActuatableSignal` instances can be used as `wot:Property` of a car Thing, and all `vss:ObservableSignal` instances are not writable. Likewise, all `vss:ActuatableSignal` instances can be used as `wot:Action`. We do not cover the case of `wot:Event` in this research.

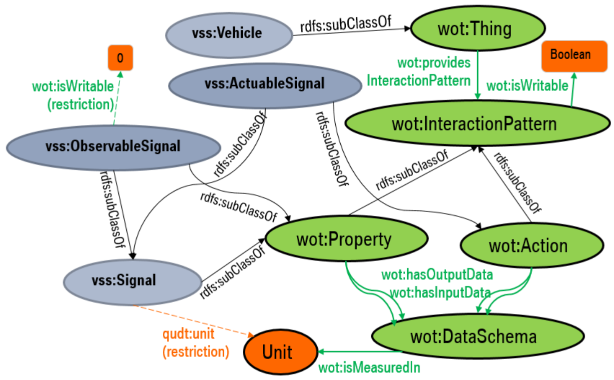

Fig. 8. Modeling pattern of VSSo with WoT (VSSo in blue, WoT in green, units and literals in orange)

Listing 4: extracts from a TD representing a car and the `wot:Property` of the `vss:VehicleSpeed`

```
"@context": [
"https://w3c.github.io/wot/w3c-wot-td-context.jsonld/",
"https://w3c.github.io/wot/w3c-wot-common-context.jsonld/",
{"om":"http://www.wurvoc.org/vocabularies/om-1.8/"},
{"auto": "https://auto.schema.org/" },
{"vsso": "http://automotive.eurecom.fr/vsso#" }]
"@type": ["Thing","vss:Vehicle", "auto:Car"],
"name" : "BMW 7 Series",
"auto:brand" : "BMW",
"interaction" : [
```

[44] https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/WeatherOntology.owl
[45] https://ci.mines-stetienne.fr/seas/WeatherOntology
[46] http://purl.obolibrary.org/obo/mfoem.owl
[47] http://semanticscience.org/ontology/sio.owl
[48] https://www.toyota-ti.ac.jp/Lab/Denshi/COIN/Ontology/TTICore-0.03/
[49] http://ci.emse.fr/opensensingcity/ns/wp-content/plugins/smartcities/survey_files/vocabs/vocabulary_81

```
{
  "@type": ["Property","vsso:VehicleSpeed"],
  "wot:isMeasuredIn":"om:Speed_Unit",
  "name": "speed",
  "outputData": { "type": "float" },
  "observable": true,
  "writable": false,
  "link":
  [{
    "href" : "property/read/speed",
    "mediaType": "application/json"
  }]}]
```

4.2. Protocol Binding

The WoT specification does not yet provide strict rules about how WoT can be bound to specific ecosystems and communication protocols. We proposed two bindings in regards to two usage of WoT in the automotive domain: connected vehicle with limited resource, and vehicle fleet abstraction in the cloud.

4.2.1. HTTP(S) binding

In recent experiments, we used HTTP and HTTPS for interacting with a car servient. Even though there are no strict standard on how HTTP is bound to a WoT thing, there are some common practices: GET to read a property, PUT or POST to write a property or invoke an action. In our approach, we can read and write properties respectively with a GET and a POST method containing the new value as payload. We can invoke actions with a POST request with an optional payload. One can observe events using HTTP long-polling [50]. In this case, the sub protocol is defined as in Listing 4.2.1.

Listing 5: Extract from a TD defining an event accessed by HTTP long-polling

```
"events":{
    "tire-pressure-warning":{
        "type": "string",
        "forms": [{
            "href": "https://myCar.example.com/tire",
            "subProtocol": "LongPoll"
        }]
    }
}
```

4.2.2. LwM2M binding

We use the device management protocol LwM2M[51] (LightweightM2M) specified at the Open Mobile Alliance[52] for exchanging data between the vehicle and our backend. LwM2M is designed for remote management of sensor networks in machine-to-machine environments. It is built on top of CoAP and features a RESTful architectural design, with an extensible resource and data model.

A READ request can give information about a sensor value at one moment. If the server *subscribes* to speed sensor value, the client will do regular READ request and the server will access it in soft real-time. A WRITE request can update a value for an actuator. In this case, the WRITE request would also contain a parameter value pushed to the vehicle.

A discovery and a Thing Description consumption phase provides the remote servient a description of properties, actions and events that can be called through LwM2M equivalent operations on mapped objects. In this case, a mapping between LwM2M and WoT operations, as well as a definition of TD as LwM2M objects will be provided. Our implementation demonstrates the usability of LwM2M as protocol for WoT, and allows its usage. This is especially relevant for applications with constrained devices.

5. Evaluation

5.1. Competency questions

In order to evaluate the coverage of the VSS ontology, we tried to write SPARQL queries for all competency questions described in the Section **??**[53]. We generate synthetic traces data using the VSSo ontology.

What are the attributes of a car and what do they express? This query retrieves the static attributes information about a car.

```
SELECT ?branch ?attribute ?value
WHERE {
  ?attribute rdfs:subPropertyOf vsso:attribute.
  ?branch ?attribute ?value. }
```

What are the attributes of the chassis? This query is interesting for focusing on only one branch of the car.

```
SELECT ?attribute ? value
WHERE {
  ?attribute rdfs:subPropertyOf vsso:attribute.
  ?branch ?attribute ?value;
           a vsso:Chassis. }
```

[50] https://realtimeapi.io/hub/http-long-polling/

[51] http://openmobilealliance.org/iot/lightweight-m2m-lwm2m

[52] http://openmobilealliance.org

[53] A more complete list is available at https://github.com/klotzbenjamin/vss-ontology

Table 1
WoT binding with LwM2M and HTTP(S)

WoT		LwM2M		HTTP(S)	
Interaction	Method	Method	Input	Method	Input
Property	Read	Read	Property object	GET	
	Write	Write	Property instance, parameter	POST	Parameter
	Observe			GET (longpoll)	
Action	Invoke	Execute	Action object	POST	Parameter
	Update/cancel task	Write on instance	Action instance, parameters		
Event	Subscribe	Observe	Event object	GET (longpoll)	
	Update/Cancel subscription	Write on instance	Event instance, parameter		

Which unit system does the signal `vsso:VehicleYaw` *use?* The ontology enables to perform queries on units.

```
SELECT ?unitsystem
WHERE {
  ?yaw  a vsso:VehicleYaw;
        qudt:unit ?unitsystem . }
```

What is the current gear? A developer should only be required to know the URI of a signal to retrieve its last value and metadata[54]. In this example, with the SSN/SOSA observations, we check that the current time is the time of the observation and retrieve the value and unit.

```
SELECT ?signal ?result ?time
WHERE {
  ?signal a vsso:CurrentGear.
  ?obs a sosa:Observation;
           sosa:observedProperty ?signal;
           sosa:hasSimpleResult ?result;
           sosa:phenomenonTime ?time.
  FILTER(?time == NOW())
}
```

Which windows are currently open? In this case, we consider the position of a car component, to make sure that one can define it in instances of car branches and signals. The window position is in percent, so if a signal is observed with a value different from 100, the branch that contains it is kept and we look at the property `vsso:position` of the remaining branches.

```
SELECT ?position
WHERE {
  ?windowPosition a vsso:WindowPosition.
  ?window vsso:hasSignal ?windowPosition.
  ?obs a sosa:Observation;
           sosa:observedProperty ?windowPosition;
           sosa:hasResult ?result;
           sosa:PhenomenonTime ?time.
  FILTER(?time == NOW())
  ?result qudt:numericValue ?value.
  FILTER(?value < 100)
  ?window vsso:position ?position.
}
```

[54] In the case of time-related query, we assume we can define the current time with a function NOW().

VSSo fits our requirements of being based on an automotive standard and semantically enriching car data. Furthermore, with more than 300 different signals and 50 attributes, VSSo defines more concepts than all ontologies, vocabularies and schemata from the state of the art, making it more complete. Finally, because VSSo is based on a specification meant to be extended, it is also easy to extend, as we will see in Section 6.2.

5.2. *Efficiency of interactions with a car*

In regard to the research question, we want our model to fit the following requirements:

- Be automotive specific,
- Contain the semantic metadata of automotive concepts, through a vocabulary, ontology or schema for instance,
- Describe most attributes of a car,
- Describe most signals of a car,
- Be generic, or as use-case independent as possible.

In the state of the art (Section **??**), most initiatives were developed for the automotive domain only. Only the Web of Things model is independent from it. Therefore, when used alone, it is missing domain-specific knowledge. The static attributes are described in several models from the state of the art. The auto.schema.org extension has 20 attributes, and the ontologies it originates from have up to 50 attributes specific to their usage. This is quite close to the number of attributes described in VSS, which already includes all auto.schema.org concepts. OBD-II provides access to

Table 2

Comparison of the different models in regard to the hypothesis: we can describe most automotive signals and attributes with semantics in generic applications

Initiative	Automotive specific	Attributes coverage	signals coverage	Semantic metadata	Application	Release status
auto.schema.org and its sources[24]	Yes	High	Very limited	Yes	E-commerce	Schema
Toyota TTI Car ontology[38]	Yes	Limited	Limited	Yes	ADAS	Ontologies
Context-aware services[20, 21, 22, 23]	Yes	very limited	Very limited	Yes	ADAS	Not public
DFKI automotive ontology[40]	Yes	Limited	Limited	Yes	ADAS	Not public
OCM ontology[41]	Yes	Very limited	Limited	Yes	ADAS	Not public
VSS	Yes	High	Very high	No	Generic	Standard available
OBD2	Yes	Limited	High	No	Diagnosis	Standard available
Extended Vehicle[8, 9, 11, 12]	Yes	Unknown	Unknown	No	Telematics, Diagnosis	Not released
Mercedes Benz Connected Vehicle API	Yes	Very limited	Limited	No	Multiple use cases	Not released
Mercedes Benz other APIs	Yes	High	Limited	No	Multiple use cases	APIs available
High Mobility APIs	Yes	High	High	No	Mutliple use cases	APIs available
PSA Connected Car API	Yes	Very limited	Limited	No	Telematics	API available
General Motors APIs	Yes	Unknown	Unknown	No	Multiple use cases	APIs available
Ford/AppLink	Yes	Very limited	Limited	No	HMI	SDK available
Toyota MSPF	Yes	Limited	Very limited	No	Carsharing	API available
WoT	No	Not relevant	Not relevant	Yes	Not relevant	Specification available
VSSo+SOSA/SSN+WoT	Yes	Yes	Yes	Yes	Generic	Available

the main identifiers of a vehicles, hence a tenth of attributes. Most ontologies developed for the automotive domain, with the exception of the e-commerce ones, have a very limited coverage of car attributes, due to the limited scope of their applications (mostly ADAS) with at most 13 attributes [40]. Proprietary OEM APIs provide access and descriptions of some attributes, but depending on the API, the number will vary from zero to a few tens in the High Mobility API and the various Mercedes APIs.

The dynamic signals are described in most models from the state of the art. In the e-commerce, the signals are barely described. In auto.schema.org, the only signal available is the speed for instance. In ADAS models, only limited sets of signals are described, usually the speed, acceleration and distance to other vehicles, while one finds up to 25 signals in [40]. Those ontologies have more classes defining contextual features not produced by cars themselves. VSS defines about 1100 signals, reduced to about 300 when the redundancy is removed. OBD-II provides about 200 signals with a strong focus on diagnosis. Most OEM APIs provide access to some signals. With the exception of High Mobility that describes hundreds of signals, those APIs define only limited subsets, usually for telematics use cases limited to about 40 signals in the PSA Connected Car API.

The auto.schema.org extension and the automotive ontologies provide formal definitions of their signals and attributes and define formally what a vehicle is. This is not the case in VSS, OBD-II, the Extended Vehicle or any OEM API. The WoT provides formal definitions of interaction patterns, but lack domain semantics. They are provided by VSSo.

Automotive ontologies are, in their vast majority, defined for two specific use cases: e-commerce and ADAS. OBD-II and the Extended Vehicle, despite some reported usage[55], focus on diagnosis. VISS and VSS are generic and do not emphasize a specific use case or set of signals. OEMs API are generally use case dependent, but with the multiplication of them, their are multiple use cases covered. The PSA Connected Car API, Ford AppLink and Toyota MSPF are exception with focuses respectively on telematics, HMI data and the car sharing use case.

Finally, a data model for the automotive domain should be as open and standard as possible. This is the case of auto.schema.org and the TTI ontologies, but

[55]https://www.postscapes.com/connected-car-devices/

not for most ADAS ontologies that were not made publicly available. Both VSS and OBD-II are well-known standards, while the Extended Vehicle is not yet released. The Mercedes Benz Connected Vehicle API is not yet released, while its other APIs are available. Most OEMs APIs and SDKs are available with a possibility to developers to become a partner and test them. The WoT specification is available, yet not a standard.

We presented our work to the WoT communities during multiple plugfests. Such meetings consist in gathering researchers from multiple domains to try to interact with Web Things, and learn as much as possible in a short period of time. This test consisted in checking if non-experts could manage to interact with a car without a training. In 2017, we presented a car directly connected to the WoT and proved that anyone with the right access could interact with a set of signals from a simple Web browser. In 2018, we presented simulations of cars running in the cloud, and managed to have them interact with other Web Things. In a first case a single car[56] was parsed, understood and used by multiple non-experts. In a second time, we presented a fleet of three vehicles[57] that was accessed safely, parsed, and used by other non-experts. From those experience, we get the empirical validation of our hypothesis: non-experts can interact with our cars in the WoT efficiently and easily.

6. Applications

6.1. Use cases benefiting from VSSo

We used VSSo in various use cases to highlight its benefits. The most general use case for VSSo is the creation and query of triples about observations. Such triples are created following the SOSA pattern. For instance, an observation of a temperature in degrees Celsius would be written as in Listing 6.1, with description of the geolocation with `geo:lat` and `geo:long`.

Listing 6: An Observation of a temperature

```
:AmbientAirTemperature/observation171 a sosa:Observation ;
    geo:lat "48.151099"^^xsd:long ;
    geo:long "11.540354"^^xsd:long ;
    sosa:hasFeatureOfInterest :MyCar ;
    sosa:hasResult [ a qudt-1-1:QuantityValue ;
        qudt-1-1:numericValue "-0.5" ;
        qudt-1-1:unit qudt-unit-1-1:DegreeCelcius ] ;
    sosa:madeBySensor :MyThermometer ;
    sosa:observedProperty :MyAmbientAirTemperature ;
    sosa:phenomenonTime "2018-01-22T08:17:15.67Z"
        ^^xsd:dateTime .
:MyThermometer a vsso:Thermometer .
:MyAmbientAirTemperature a vsso:AmbientAirTemperature .
:MyCar a vsso:Vehicle .
```

We developed a simulator for car data, producing RDF triples following the SSN/SOSA and VSSo ontologies. These observations are available for trying the queries presented in Section 5.1. A public sparql endpoint is available at automotive.eurecom.fr/simulator/query.

A current challenge in trajectory pattern mining [42] is the production and analysis of car data. Among the benefits of such an analysis is the knowledge about patterns but also the understanding of trajectories for drivers [43], outlier detection [44], and predictions [45]. The current trend is to use smartphones and a limited set of signals, mostly time and location. There has been some research on the case of the automotive domain [43, 44], but it is mostly limited to open datasets of fleets of taxis or from one unique vehicle. VSSo makes it easier to combine data from a heterogenous fleet [46]. We recorded data from a BMW car and developed a interfacing server[58] to interact with these traces using the VSSo model. In this demonstration, we create a static graph describing the car's attributes, then fill it with `sosa:Observation` and use a simple reasoner to label trajectories segments between consecutive observations.

6.2. VSSo Extensions

Just like VSS is meant to be extended with private signals and branches, VSSo can import new concepts defined in other namespaces. In order to do so, a developer can directly use VSSo and its patterns to manually create new attributes, branches and signals. Another solution consists in writing the VSS extension in vspec format, and generate a new ontology. However this second solution requires a step of validation afterwards. We extended the generator with a health check script[59] in order to reduce the effort of manual validation. For instance, an OEM can define a private sig-

[56] https://github.com/w3c/wot/blob/master/plugfest/2018-prague/result.md

[57] https://github.com/w3c/wot/blob/master/plugfest/2018-sept-online/result-fujitsu.md, https://github.com/w3c/wot/blob/master/plugfest/2018-sept-online/result-panasonic.md

[58] automotive.eurecom.fr/trajectory

[59] https://github.com/klotzbenjamin/vss-ontology/tree/master/rdf-generation

nal for a new embedded camera. In order to use it, a developer will define this camera as part of the VSSo extension in a new namespace.

6.3. Automotive Semantic Web Thing Prototypes

In our prototyping, we have three challenges in regard to vehicle data-based applications:

- Find the right degree of abstraction from of a data model for vehicle data and services;
- Transport the data to the cloud reliably, securely and efficiently;
- Ease the access to both internal and external applications.

6.3.1. Connected car with LwM2M binding

This first demonstration establishes the benefits of using VSSo and a binding with LwM2M to interact directly with a vehicle and was presented at the W3C WoT F2F meeting (Düsseldorf, 2017). In this prototype, we demonstrate the feasibility of implementation of a car as a WoT servient. The prototype highlights the potential use of properties, actions and events on a motionless vehicle based on doors/windows sensors and actuators.

Architecture. As depicted in Figure 9, the general architecture of the prototype contains six main parts:

1. Car data access with the computing device, through the OBD (On Board Diagnosis) interface;
2. Implementation of a LwM2M client on the computing device and server in the cloud exchanging messages over CoAP;
3. Protocol Binding: implementation of a mapping between LwM2M and WoT (Table 1);
4. Thing Description: retrieval and parsing of metadata;
5. Scripting API: WoT endpoint;
6. WoT client in a browser exchanging over HTTP with the WoT server.

The vehicle is connected to a computing device through its OBD dongle, which is then connected to the cloud via a LTE connection. A CoAP[60] (Constrained Application Protocol) server is running on the latter, that can notice a client running on the computing device and do GET/SET/SUBSCRIBE/EXECUTE calls. When a sensor value is required, the client sends an OBD job on the vehicle to retrieve the raw information, then enrich it with semantic annotations based on its TD, and sends the enriched data to the WoT server.

One possible implementation of LwM2M in java is the open source project Leshan[61]. Through Leshan and an additional implementation running in the vehicle, it is possible to have read and write access to selected and published data streams of the vehicle. To facilitate an easy integration of other components and domains a separate high-level, but proprietary API is implemented. An important aspect is to work on the same data model throughout the stack. Table 1 presents the WoT interactions patterns mapped between HTTP in the OEM cloud and LwM2M to reach the vehicle.

Implementation notes. In our implementation, we used a Rasperry Pi as a computing device, connected to the OBD interface. In this demonstration, we implement the following interactions:

- Properties speed, passenger door lock,
- Actions passenger doors lock and unlock, honk,
- Event speed value: built as a subscription to a property speed.

6.3.2. Simulated cars in the cloud with a HTTP(S) binding

In order to focus on the data model and semantic interoperability challenge, we also created a servient for an abstracted vehicle in the cloud. This makes it easier to test WoT principles on a virtual fleet with simulated data, or records.

Architecture. The general architecture of these servients is composed of 6 main parts, as depicted in Figure 10:

1. Car data mockup
2. Thing description
3. Device scripting API
4. HTTP protocol binding and access control
5. Application scripting API
6. Application script

The Car data mockup uses either a simulation of signals, or real historical data. In the case of a written property or an action updating a property value, we create an intermediate dictionary that will check, for a value to read, if it has already been overwritten. If this is the case, it will take the value from the intermediate dictionary. The device scripting API consumes the Thing Description, and creates the interac-

[60] http://coap.technology/

[61] https://www.eclipse.org/leshan/

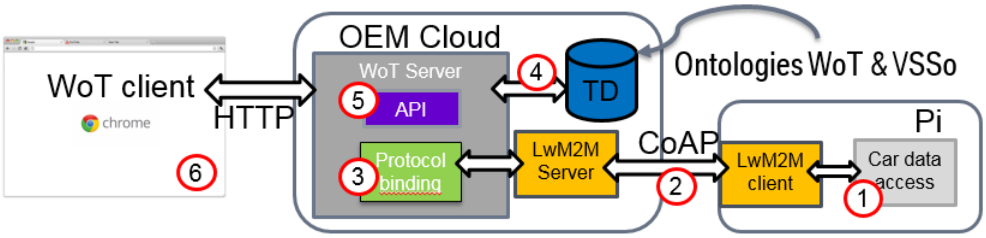

Fig. 9. Prototype architecture: a WoT servient runs in the OEM clouds and interacts with a WoT client in a browser. Box colors match the color scheme of the WoT servient architecture.

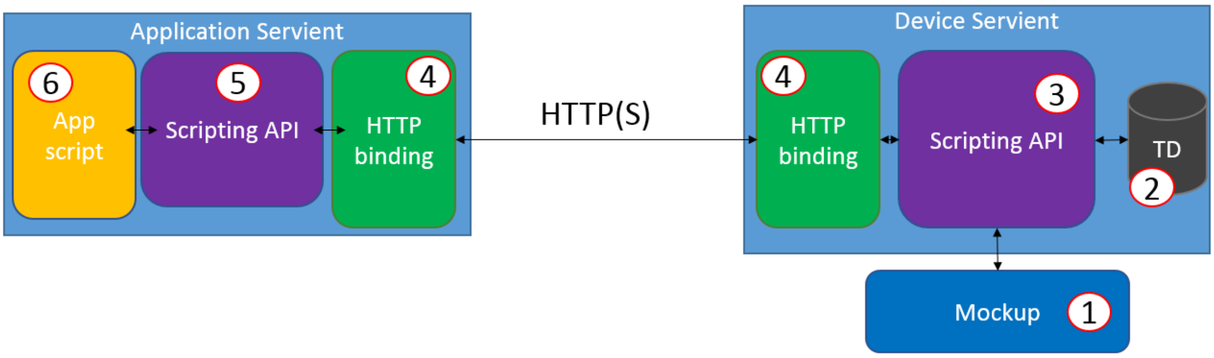

Fig. 10. Prototype architecture: multiple device servients in the cloud and and an application servient connected to them via HTTP(S).

tions mapped to the mockup. The HTTP(S) binding is based on the proposal presented in Section 4.2.1. In these implementations, both the device servient and the application servient use HTTP or HTTPS. The device servient uses also OAuth 2.0[62] to grant access to allowed users. Interactions are therefore protected, but the Thing Description is always available. The Application scripting API discovers the Thing Description of connected device servients, and retrieves their interactions and metadata. The application script then allows a developer to use those interactions and write applications.

We developed several applications during the plugfest. For example, we connected multiple times our vehicle servient to a smart home [63] or a set of devices from a smart home [64]. We also monitored a fleet, including three vehicles that we had created, and a car [65] and a truck [66] from Oracle. In this case, we retrieved all interactions and metadata for all vehicles, and due to the lack of semantic annotations of certain Thing Descriptions, we hard-coded interfaces and monitored the location and speed of all vehicles. Finally, we had applications only interacting with one vehicle servient. In this case, we applied simple rules to check if the doors of the vehicle were closed while it was moving, and turned on the DSC (Dynamic Stability Control) if the temperature was below 0 degree Celcius.

7. Conclusion and Future Work

In this paper, we identified a gap in formal definition of car signals and sensors. We used some best practices both from the Semantic Web community and the automotive standards to propose VSSo, an ontology developed on top of the SSN/SOSA W3C recommendation. This new formal representation of car signals and attributes allows semantic queries and annotation of au-

[62]https://oauth.net/2/
[63]https://youtu.be/zkL8Cdgy8PE
[64]https://youtu.be/pjgTLPlAsKQ
[65]https://github.com/w3c/wot/blob/master/plugfest/2018-sept-online/TDs/Oracle/Connected_Car_Shared.jsonld
[66]https://github.com/w3c/wot/blob/master/plugfest/2018-sept-online/TDs/Oracle/Truck_Shared.jsonld

tomotive Web Things. The Web of Things enables to expose vehicle data and servers through multiple standard protocols and ecosystems, while keeping the data model unique and standard. The experiments carried out confirm that this approach makes interactions and application development more accessible to a wider range of developers.

In the future, we will work on VSS to make it more complete and consistent, and update VSSo in order to cover even more signals and attributes. VSSo will be the basis of the development of a data model in the W3C automotive Working Group and we will provide several examples of Thing Description to use as modules describing vehicles in various use cases.

References

[1] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller and H. Winner, Three Decades of Driver Assistance Systems: Review and Future Perspectives, *IEEE Intelligent Transportation Systems Magazine* **6**(4) (2014), 6–22.

[2] M. Broy, I.H. Kruger, A. Pretschner and C. Salzmann, Engineering Automotive Software, *Proceedings of the IEEE* **95**(2) (2007), 356–373.

[3] J. Bereisa, Applications of Microcomputers in Automotive Electronics, *IEEE Transactions on Industrial Electronics* **IE-30**(2) (1983), 87–96.

[4] P. Barnaghi, W. Wang, C. Henson and K. Taylor, Semantics for the Internet of Things: early progress and back to the future, *International Journal on Semantic Web and Information Systems (IJSWIS)* **8**(1) (2012), 1–21.

[5] L. Atzori, A. Iera and G. Morabito, The internet of things: A survey, *Computer networks* **54**(15) (2010), 2787–2805.

[6] D. Raggett, The Web of Things: Challenges and Opportunities, *Computer* **48**(5) (2015), 26–32.

[7] V. Charpenay, Build a OWL ontology from VSS, 2016, unpublished.

[8] I. 20078, Road vehicles – Extended vehicle (ExVe) web services, Standard, ISO, 2018.

[9] I. 20078, Road vehicles – Information for remote diagnostic support – General requirements, definitions and use cases, Standard, ISO, 2018.

[10] S. Murphy, A. Nafaa and J. Serafinski, Advanced service delivery to the Connected Car, in: *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2013, pp. 147–153.

[11] François Croc, Extended Vehicle: The answer for a Safe & Secure connected vehicle, 2018.

[12] I. 20077, Road Vehicles – Extended vehicle (ExVe) methodology, Standard, ISO, 2018.

[13] T.N.V.W. Group, Neutral Vehicle - Technical Concept, 2017.

[14] K. Gavigan, A. Crofts, P. Kinney and W. Lee, Vehicle Information Service Specification, Candidate Recommendation, W3C, 2018.

[15] M. Aro, S. Urata and P. Kinney, Vehicle Information API Specification, Working Draft, W3C, 2017.

[16] A. Maier, H.-P. Schnurr and Y. Sure, Ontology-Based Information Integration in the Automotive Industry, in: *2nd International Semantic Web Conference (ISWC)*, 2003, pp. 897–912.

[17] A. Reymonet, J. Thomas and N. Aussenac-gilles, Ontology Based Information Retrieval: an application to automotive diagnosis, in: *20th International Workshop on Principles of Diagnosis (DX)*, Stockholm, Sweden, 2009, pp. 9–14.

[18] M. Feld and C. Müller, The Automotive Ontology: Managing Knowledge Inside the Vehicle and Sharing it Between Cars, in: *3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, Salzburg, Austria, 2011, pp. 79–86.

[19] L. Zhao, R. Ichise, S. Mita and Y. Sasaki, Core Ontologies for Safe Autonomous Driving, in: *14th International Semantic Web Conference, Posters and Demos Track (ISWC)*, 2015.

[20] A. Armand, D. Filliat and J. Ibañez-Guzman, Ontology-based context awareness for driving assistance systems, in: *IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 227–233.

[21] M. Madkour and A. Maach, Ontology-based context modeling for vehicle context-aware services, *Journal of Theoretical and Applied Information Technology* **34**(2) (2011).

[22] Z. Xiong, V.V. Dixit and S.T. Waller, The development of an Ontology for driving Context Modelling and reasoning, in: *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 13–18.

[23] S. Kannan, A. Thangavelu and R. Kalivaradhan, An Intelligent Driver Assistance System (I-DAS) for Vehicle Safety Modelling using Ontology Approach, *International Journal Of UbiComp (IJU)* **1**(3) (2010).

[24] R.V. Guha, D. Brickley and S. Macbeth, Schema. org: evolution of structured data on the web, *Communications of the ACM* **59**(2) (2016), 44–51.

[25] K. Janowicz, S. Cox, K. Taylor, D.L. Phuoc, M. Lefrançois and A. Haller, Semantic Sensor Network Ontology, Recommendation, W3C, 2017.

[26] A. Haller, K. Janowicz, S. Cox, M. Lefrançois, K. Taylor, D.L. Phuoc, J. Lieberman, R. García-Castro, R. Atkinson and C. Stadler, The Modular SSN Ontology: A Joint W3C and OGC Standard Specifying the Semantics of Sensors, Observations, Sampling, and Actuation, 2018, under review.

[27] K. Kajimoto, R. Matsukura, J. Hund, M. Kovatsch and K. Nimura, Web of Things (WoT) Architecture, W3C Unofficial Draft, W3C, 2018, https://w3c.github.io/wotwg/architecture/wot-architecture.html.

[28] V. Charpenay, S. Käbisch and H. Kosch, Introducing Thing Descriptions and Interactions: An Ontology for the Web of Things..

[29] S.K. Datta, R.P. Ferreira Da Costa, C. Bonnet and J. Härri, Web of things for connected vehicles, in: *WWW 2016, 25th International World Wide Web Conference, W3C Track, April 11-15, 2016, Montreal, Canada*, Montreal, CANADA, 2016.

[30] R. Hodgson, P.J. Keller, J. Hodges and J. Spivak, QUDT-quantities, units, dimensions and data types ontologies, 2014.

[31] H. Rijgersberg, M. Van Assem and J. Top, Ontology of Units of Measure and Related Concepts, *Semantic Web journal* **4**(1) (2013), 3–13.

[32] M. Lefrançois and A. Zimmermann, Supporting arbitrary custom datatypes in RDF and SPARQL, in: *International Semantic Web Conference (ISWC)*, 2016, pp. 371–386.

[33] J.M. Keil and S. Schindler, Comparison and Evaluation of Ontologies for Units of Measurement, *Semantic Web journal* (2018).

[34] M. Lefrançois, J. Kalaoja, T. Ghariani and A. Zimmermann, The SEAS Knowledge Model, PhD thesis, ITEA2 12004 Smart Energy Aware Systems, 2017.

[35] M.J. Kofler, C. Reinisch and W. Kastner, An ontological weather representation for improving energy-efficiency in interconnected smart home systems (2012).

[36] M. Dumontier, C. Baker, J. Baran, A. Callahan, L. Chepelev, J. Cruz-Toledo, N. R Del Rio, G. Duck, L.I. Furlong, N. Keath, D. Klassen, J. McCusker, N. Queralt-Rosinach, M. Samwald, N. Villanueva-Rosales, M. Wilkinson and R. Hoehndorf, The Semanticscience Integrated Ontology (SIO) for biomedical research and knowledge discovery, *Journal of Biomedical Semantics* **5**(1) (2014).

[37] J. Hastings, W. Ceusters, K. Mulligan and B. Smith, Annotating affective neuroscience data with the emotion ontology (2012).

[38] L. Zhao, R. Ichise, S. Mita and Y. Sasaki, Core Ontologies for Safe Autonomous Driving, in: *14th International Semantic Web Conference, Posters and Demos Track (ISWC)*, 2015.

[39] Y. Raimond, S.A. Abdallah, M.B. Sandler and F. Giasson, The Music Ontology., in: *ISMIR*, 2007.

[40] M. Feld and C. Müller, The Automotive Ontology: Managing Knowledge Inside the Vehicle and Sharing it Between Cars, in: *3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, Salzburg, Austria, 2011, pp. 79–86.

[41] Z. Xiong, V.V. Dixit and S.T. Waller, The development of an Ontology for driving Context Modelling and reasoning, in: *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 13–18.

[42] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra and K. Aberer, Semantic Trajectories: Mobility Data Computation and Annotation, *ACM Transactions on Intelligent Systems and Technology (TIST)* **4**(3) (2013), 49–14938.

[43] H. Su, K. Zheng, K. Zeng, J. Huang and X. Zhou, STMaker: A System to Make Sense of Trajectory Data, *40th International Conference on Very Large Data Bases (VLDB)* **7**(13) (2014), 1701–1704.

[44] A.R. de Aquino, L.O. Alvares, C. Renso and V. Bogorny, Towards Semantic Trajectory Outlier Detection., in: *14th GeoInfo Conference (GEOINFO)*, 2013, pp. 115–126.

[45] A. Monreale, F. Pinelli, R. Trasarti and F. Giannotti, WhereNext: A Location Predictor on Trajectory Pattern Mining, in: *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Paris, France, 2009, pp. 637–646.

[46] B. Klotz, R. Troncy, D. Wilms and C. Bonnet, Generating Semantic Trajectories Using a Car Signal Ontology, in: *The Web Conference (WWW), Demo Track*, Lyon, France, 2018.