S2DM: a Simplified Semantic Data Modeling approach

Daniel Alvarez-Coello[1,*]

[1]BMW Group. New Technologies, Digital Vehicle Infrastructure. Munich, Germany.

Abstract

The Simplified Semantic Data Modeling (S2DM) approach addresses challenges in managing controlled vocabularies by combining semantic expressiveness with usability for non-modelers. Originally motivated by the automotive domain within the COVESA alliance, S2DM is domain-independent and supports the development, extension, and governance of structured data models. S2DM consists of a modeling guideline and supporting tooling, leveraging established languages such as GraphQL SDL and SKOS to enable modularity, cross-domain references, and semantic clarity. It aims to minimize modeling effort while maximizing clarity and reuse. The approach is positioned relative to alternative frameworks, and future work includes extending tooling, adopting complementary standards, and exploring semi-automated modeling techniques.

Keywords

Data modeling, semantic data models, controlled vocabularies, GraphQL SDL, SKOS, data-centric architecture

1. Introduction

The importance of data is acknowledged worldwide. However, today, it is no longer sufficient to base decisions solely on data (i.e., to be data-driven), which has become the new norm. It is also essential to have an appropriate approach for the development and evolution of the vocabularies that are used in the physical layer (e.g., in a concrete database or application). When the scope of the domain model grows significantly, controlling those vocabularies and enhancing them with additional features can be non-trivial.

Such hurdles have been experienced in the *Connected Vehicle Systems Alliance* (*COVESA*).[1] Its community has been maintaining for almost a decade a vocabulary of vehicle properties called *Vehicle Signal Specification* (*VSS*).[2] *VSS* uses a YAML-based modeling language with custom syntax and constructs. As the automotive ecosystem evolves, the need for semantic clarity increases. Limitations of the modeling language behind *VSS* have become apparent.[3] What began as an initiative to enhance the modeling approach in the *VSS* project revealed a broader opportunity: to create a modeling approach that is itself a reusable artifact. Hence, this paper proposes the *Simplified Semantic Data Modeling* (*S2DM*) approach, whose design principles are introduced next.[4]

1.1. S2DM design Principles

The *S2DM* approach is grounded in four components according to the Design Science Research methodology []. Namely: problem, requirements, goals, and artifacts.

Problem Disparate (vehicle) data models without proper semantics. Large organizations often suffer the effects caused by application-centric architectures due to the lack of controlled vocabularies [].

Requirements *S2DM* aims to satisfy the following criteria listed in Table 1.

Woodstock'22: Symposium on the irreproducible science, June 07–11, 2022, Woodstock, NY

*Corresponding author.

daniel.alvarez-coello@bmwgroup.com (D. Alvarez-Coello)

0000-0001-5663-7543 (D. Alvarez-Coello)

[1]https://covesa.global

[2]https://covesa.github.io/vehicle_signal_specification/

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

[4]Some elements are presented from the perspective of the automotive industry, but they might be applicable to other areas.

Criteria	Requirement
Simplicity	Understandable and usable with a low entry barrier for a *Subject Matter Expert* (*SME*).
Technology Agnosticism	Applicable in any environment and independent of specific platforms or vendors.
Modularity	Smaller independent parts can be modeled and combined into a coherent whole.
Scalability & Maintainability	Supports growth and maintenance as domains expand and requirements evolve.
Metadata Resource Uniqueness	Clear explicit identification and versioning of elements to avoid confusion or ambiguity.
Multi hierarchies	Enhances find-ability by supporting multiple ways to classify and organize concepts.
Cross-Domain References	Enables linking and reuse of data across different domains and application areas.
Possible operations	Specifies data structures along with their associated capabilities.
Community & Tools	Compatible with open tools and broadly adopted industry standards for wide support.
Automated Reasoning[5]	Supports reasoning features like detecting inconsistencies or inferring facts from data.

Table 1
Design criteria and their corresponding requirements that were considered for the design of *S2DM*.

Goal(s) To minimize the effort required by a *Subject Matter Expert* (*SME*) to develop, extend, and maintain (vehicle-related) semantic data models while maximizing clarity, consistency, and reuse.

Artifact(s) The S2DM consists of: A *data modeling guideline* that explains how to formalize the data of a domain with the S2DM approach, and *CLI tools* that support its proper usage.

2. S2DM approach

At its core, *S2DM* relies on the established standards *Schema Definition Language* (*SDL*)[6] and the *Simple Knowledge Organization System* (*SKOS*).[7] *SDL* is used to specify data structures and the possible operations on that data. It is the official language in the *Graph Query Language* (*GraphQL*) ecosystem. Note that the focus is on the *SDL* language itself, so *S2DM* can be applied also without a *GraphQL* API. As a complement, *SKOS* is added to enable the classification of the concepts in multiple arbitrary hierarchies. The main ideas behind *S2DM* are summarized in six parts, which are illustrated in Figure 1.

Objects & Properties A domain model built with *S2DM* is envisioned to evolve by business value need only. So, a collection of objects of interest and its associated properties are maintained. For example, the property *position* of the *Window* object represented as an integer value.

Concept URIs A valid schema written in *GraphQL SDL* must comply with some naming rules. For instance, there cannot be multiple objects the same name within a particular schema. Such uniqueness is considered to build a *Uniform Resource Identifier* (*URI*)[8] for each concept, which enables future-proof identifiers usable in the RDF[9] world.

Multi hierarchies The *URIs* are used with *SKOS* enabling arbitrary classification concept schemes. For instance, a *Window* might be classified by its *physical membership* as *Vehicle.Cabin.Door.**Window***, or by its *movement type* as *VehiclePart.Movable.Unidimensional.**Window***.

[5]Reasoning is seen (in the context of *VSS* project) as a nice-to have feature but of secondary priority. Thus, it is not a must.
[6]https://spec.graphql.org
[7]https://www.w3.org/2004/02/skos/
[8]https://datatracker.ietf.org/doc/html/rfc3986
[9]https://www.w3.org/TR/rdf11-concepts/

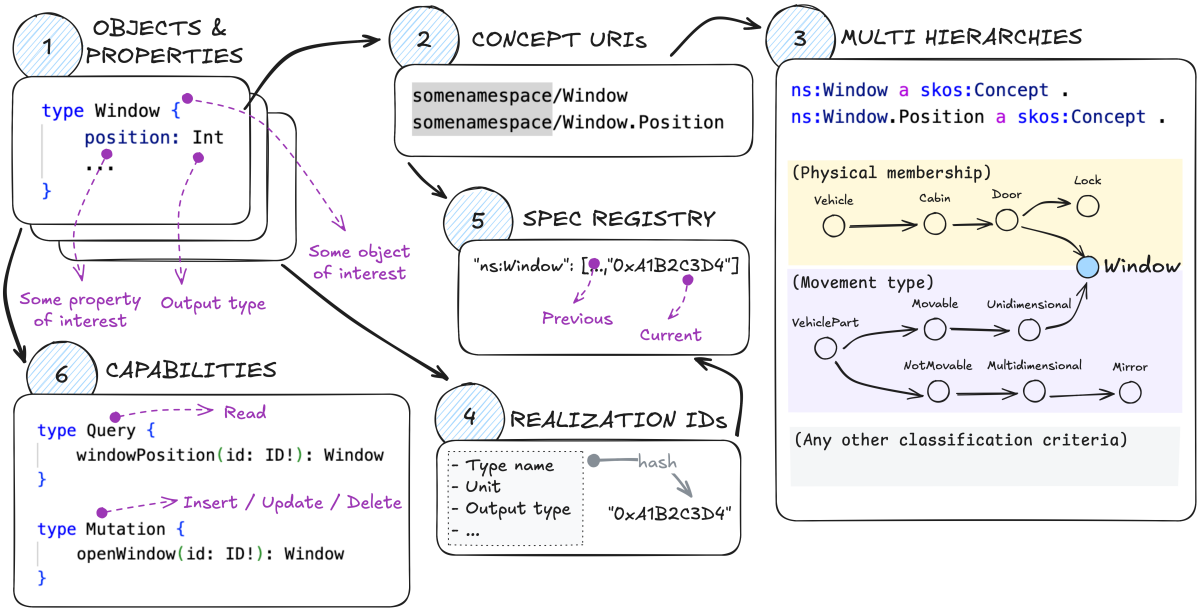

Figure 1: Overview of the *S2DM* approach. (1) Sets of objects and their properties are maintained. (2) Concepts are assigned future-proof URIs. (3) Concept URIs are re used to build classification schemes. (4) Based on the metadata in the specification, a unique hashed ID is created. (5) URIs and hashed IDs are combined into an specification registry. (6) Possible operations to the data structures are specified.

Realization IDs They refer to the unique combination of the metadata that incarnates the concept of interest. For example, the position of the window (*Window.position*) might have been initially specified with *Float* output, but then as *Integer*. Such changes in the metadata can compromise the artifact that is using the concept in the physical layer (e.g., an application). Hence, whenever a breaking change occurs, a concept is associated with a unique realization ID, a *32-bit FNV-1a hash* generated from the metadata. For example, "0xA1B2C3D4".

Spec registry Concept *URIs* and realization IDs are linked and persisted explicitly in a *JSON-LD*[10] file. One concept might be associated with multiple realization IDs, where the latest one is the one valid for the current version of the domain model. The others are kept for data governance purposes.

Capabilities As part of *SDL*, one can specify the operations that are possible on a particular data structure. For example, reading the window position (i.e., *Query*), and opening a window (i.e., *Mutation*). In this way, capabilities would be formalized but their actual implementation are beyond the scope of *S2DM*.

2.1. S2DM Modeling guideline

First of all, the specification of a domain model based on *S2DM* must comply with the syntax of its underlying languages (i.e., *GraphQL SDL* and *SKOS*). In this sense, the modeling guideline focuses only on the intended use of the custom elements that are introduced. The principal considerations are explained next. For a more complete and up-to-date description, please visit the official documentation.[11]

2.1.1. Pre-defined elements

Some elements are made available in the *S2DM* repository. They are parsed automatically when using the *S2DM* tools. As a result, they are embedded in the specification files of the organization that applies

[10] https://json-ld.org
[11] https://covesa.github.io/s2dm/

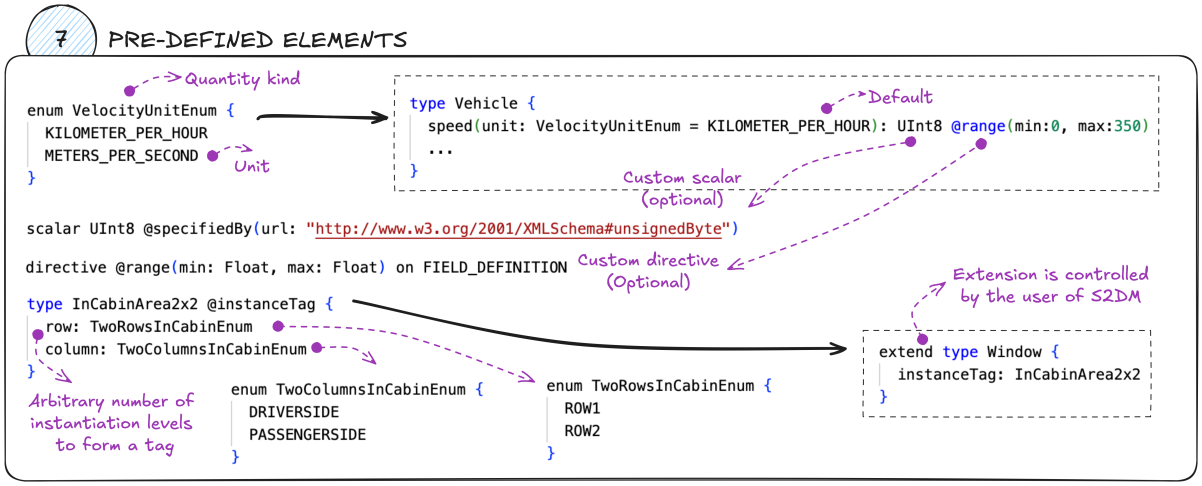

Figure 2: Example of the definition and use of the pre-defined elements in *S2DM*.

S2DM. An example is illustrated in Figure 2. Among the elements, we find units, custom scalars, custom directives, and common enumeration sets.

Units They are modeled as enumeration values, where the *enum* itself represents the quantity kind (e.g., *Velocity*). Units are used as field arguments, whose values come from a controlled *enum*.

Custom scalars Scalars are the data types to which a particular field in *GraphQL* resolves. *SDL* provides the built-in scalars *Int*, *Float*, *String*, *Boolean*, *ID*. As some use cases might need particular data types beyond those options, *SDL* supports the definition of custom ones. *S2DM* currently adds the custom scalars that correspond to the data types used in *VSS*.[12] Bear in mind that the resolving functions for custom scalars are implementation-specific and out of the scope of *S2DM*.

Custom directives They extend what one can express in the modeling language. Currently:

`@range` To specify the boundaries for the values assigned to a field.
`@cardinality` To rule how many items are possible for a particular field.[13]
`@instanceTag` To construct types that identify in a human-friendly manner instances of objects.
`@metadata` To add arbitrary information needed by specific function in the tools.

Common enumeration sets They aim to reduce the repetition of specification of things that are of common interest. For example, certain objects might have multiple instances (e.g., *Door*, *Window*, and *Seat*.). The tags that distinguish them within the same *Vehicle* is common. Namely: *ROW1.DRIVERSIDE*, *ROW1.PASSENGERSIDE*, *ROW2.DRIVERSIDE*, *ROW2.PASSENGERSIDE*. Hence, types defined with the `@instanceTag` directive enable them. Likewise, other concepts, such as *GeographicLocation* are commonly used and might be introduced in the future as part of the *S2DM*.

2.2. S2DM CLI tools

The *S2DM* CLI a tool-set to assist in the modeling, validation, evolution, and governance of domain models with *S2DM*. Figure 3 illustrates the conceptual role of the `s2dm` tools within the specification workflow. The domain model is created and maintained in the conceptual or logical layer. The *S2DM* tools provide a suite of commands that bridge this logical model with its physical usage in downstream systems, applications, or databases.

[12] https://covesa.github.io/vehicle_signal_specification/rule_set/data_entry/data_types/
[13] Only applicable in conjunction to the *GraphQL* type List modifier

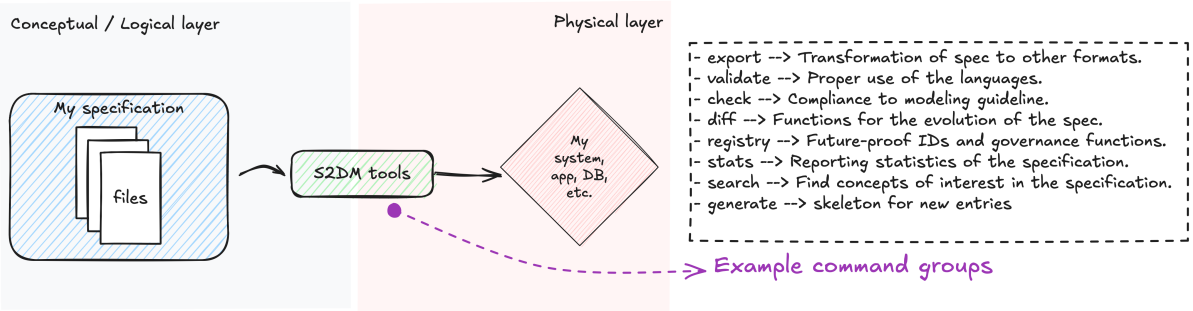

Figure 3: The role of *S2DM* tools. Domain model is specified in the conceptual layer, whereas the downstream use of the model happens in the physical layer. The tools serve as the bridge between both.

The command groups shown in the figure cover key functions such as:[14]

- **export** – Transformation of the specification into other formats, such as JSON Schema or TTL.
- **validate** – Verification of proper use of the modeling languages (e.g., syntax, naming conventions, and completeness).
- **check** – Enforcement of compliance with domain-specific modeling guidelines and governance policies.
- **diff** – Support for tracking the evolution of the specification by detecting and classifying changes (e.g., breaking, dangerous, or safe).
- **registry** – Management of future-proof identifiers (e.g., realization hashes) and related governance functions.
- **stats** – Generation of statistics and summaries about the specification's structure and completeness.
- **search** – Discovery of concepts of interest within the specification, based on keywords or similarity metrics.
- **generate** – Creation of skeleton definitions for new entries to ensure consistent modeling.

3. Related Work

A variety of modeling frameworks have been considered in the context of designing *S2DM*, each with distinct strengths and limitations when measured against the design criteria introduced in Table 1. The modeling language *VSPEC*, used in the *VSS* project[15], served as the main motivation for this work, offering simplicity and technology agnosticism but lacking native support for cross-domain references, classification schemes, and reasoning. *UML (XMI)*[16] and *SHACL (RDF)*[17] provide stronger modularity and semantic expressiveness, but introduce complexity that can deter adoption by non-specialists. *OpenAPI*[18] and *JSON Schema*[19] align well with technology agnosticism and capability specification, yet offer limited semantic enrichment or reasoning support. Semantic web standards like *SKOS (RDF)*[20] and *OWL (RDF)*[21] excel at supporting semantics, classification, and reasoning, but come with added complexity. *GraphQL*[22], with or without SKOS augmentation, presents a balanced middle ground that

[14] CLI is being developed and will be available in a first stable release soon. For an up-to-date description of the tools, consult the repository https://github.com/covesa/s2dm.
[15] https://covesa.global/vehicle-signal-specification/
[16] https://www.omg.org/spec/XMI/
[17] https://www.w3.org/TR/shacl/
[18] https://swagger.io/specification/
[19] https://json-schema.org/
[20] https://www.w3.org/TR/skos-reference/
[21] https://www.w3.org/TR/owl-features/
[22] https://graphql.org/

Criteria	VSPEC	UML (XMI)	SHACL (RDF)	OpenAPI	JSON Schema	SKOS (RDF)	OWL (RDF)	GraphQL	GraphQL + SKOS
Simplicity	✅ Yes	❌ No	❌ No	✅ Yes	⚠️ Verbose	⚠️ Partly	❌ No	✅ Yes	✅ Yes
Technology Agnosticism	✅ Yes	⚠️ Partly	⚠️ Partly	✅ Yes	✅ Yes	⚠️ Partly	❌ No	✅ Yes	✅ Yes
Modularity	⚠️ Partly	✅ Yes	✅ Yes	⚠️ Partly	✅ Yes	✅ Yes	✅ Yes	✅ Yes	✅ Yes
Scalability & Maintainability	⚠️ Partly	❌ No	⚠️ Partly	⚠️ Partly	⚠️ Partly	✅ Yes	❌ No	✅ Yes	✅ Yes
Metadata Resource Uniqueness	⚠️ Partly	❌ No	✅ Yes	⚠️ Partly	❌ No	✅ Yes	✅ Yes	⚠️ Partly	✅ Yes
Support for Multiple Classification Schemes	❌ No	⚠️ Partly	⚠️ Partly	❌ No	❌ No	✅ Yes	✅ Yes	⚠️ Partly	✅ Yes
Support for Cross Domain References	❌ No	⚠️ Partly	✅ Yes	❌ No	✅ Yes	✅ Yes	✅ Yes	✅ Yes	✅ Yes
Support for the Specification of Capabilities	❌ No	⚠️ Partly	❌ No	✅ Yes	❌ No	❌ No	❌ No	✅ Yes	✅ Yes
Community and Tools	❌ No	⚠️ Partly	❌ No	✅ Yes	✅ Yes	⚠️ Partly	⚠️ Partly	✅ Yes	✅ Yes
Support for Automated Reasoning	❌ No	❌ No	❌ No	❌ No	❌ No	⚠️ Partly	✅ Yes	❌ No	⚠️ Partly

Figure 4: Alternatives for the requirement satisfaction of the design criteria (non-comprehensive).

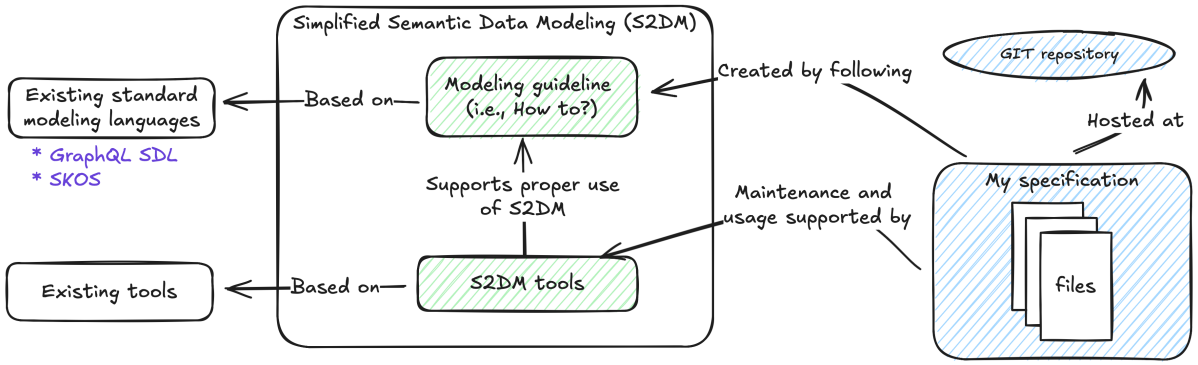

Figure 5: The role of *S2DM* and its artifacts.

combines simplicity, modularity, and strong tool support with options for semantic enrichment. A comparison of these approaches is provided in Figure 4, highlighting key aspects that informed the design of *S2DM* without aiming for a comprehensive evaluation.

4. Conclusion and Outlook

The *S2DM* approach offers a balanced modeling method that combines semantic expressiveness with usability for non-modelers. It consists of two main artifacts: a modeling guideline and supporting tooling for authoring, validation, and maintenance. Originally motivated by the needs of the COVESA alliance, S2DM is domain-independent and applicable wherever structured, semantically enriched models are needed. While S2DM does not require a GraphQL API, GraphQL can provide advantages for architectures that adopt it. The role of S2DM and its artifacts is illustrated in Figure 5.

Future work includes extending S2DM tooling, adopting standards such as QUDT, and exploring semi-automated modeling with large language models (LLMs). The use of established custom data types in GraphQL can further promote consistency and interoperability. A key goal is to establish S2DM as the main modeling approach within COVESA, while encouraging adoption in other domains. Further analysis of approaches like the Unified Data Architecture by Netflix[23] and NGSI-LD[24] may provide valuable insights for S2DM's evolution.

[23] https://netflixtechblog.com/uda-unified-data-architecture-6a6aee261d8d
[24] https://ngsild.org/