Towards Interoperable Data Products: Scenarios

Scenario 1: Creating a New Data Product based on Ontology

In the first scenario, we consider the creation of a fictional data product and its corresponding metadata. The quality control department (data consumer) is interested in collecting information about the lifespan of different parts, particularly how long after creation, a particular part needs to be replaced by a repair centre. To obtain this information, they approach the after-sales department (data provider), which can collect this information from repair reports. After consulting with the data consumer, the data provider creates the table shown in fig. 1 that contains all relevant requested information.

Rather than simply sending the table to the data consumer, the data provider decides to provide access through an API and create a data product on the internal data exchange, allowing other parties to discover and use the potentially valuable data.

The very first step for the data provider is to go to the semantic web platform and try to find the entity corresponding to their table, in this case: a repair report entity. Creating metadata is then simply the process of linking each of the columns of the data product to the corresponding properties in the repair report entity. However, for this example, we presume no such entity exists (fig. 2), and the data provider will have to explore other entities to create their metadata, as illustrated below.

Instead, the data provider finds that their domain ontology contains information about vehicles, which has most of the relevant properties captured in the table.

	A	B	C	D	E	F
1	**Car**	**Purchase Date**	**Mileage**	**Repair Date**	**Affected Parts**	**Automec**
2	Julius 4X-Drive	01/11/2018	16.000	22/02/2022	P223, P123, ...	Lewis Ha
3	August Comfort Automat	23/01/2017	17.659	22/02/2022	P980, P836	Max Vers
4	Sulla Station Sport Editior	04/02/2013	35.687	22/02/2022	P414, P186, ...	Fernandc
5	Sulla Station Sport Editior	25/05/2015	33.894	22/02/2022	P414, P218	Charles L
6	Julius 4X-Drive	20/02/2021	3.487	21/02/2022	P986, P694	Lewis Ha
7	...	...	...	...	...	...

Figure 1 **An anonymised table showing all the information requested by the quality control department. This is the data asset underlying the newly created data product.**

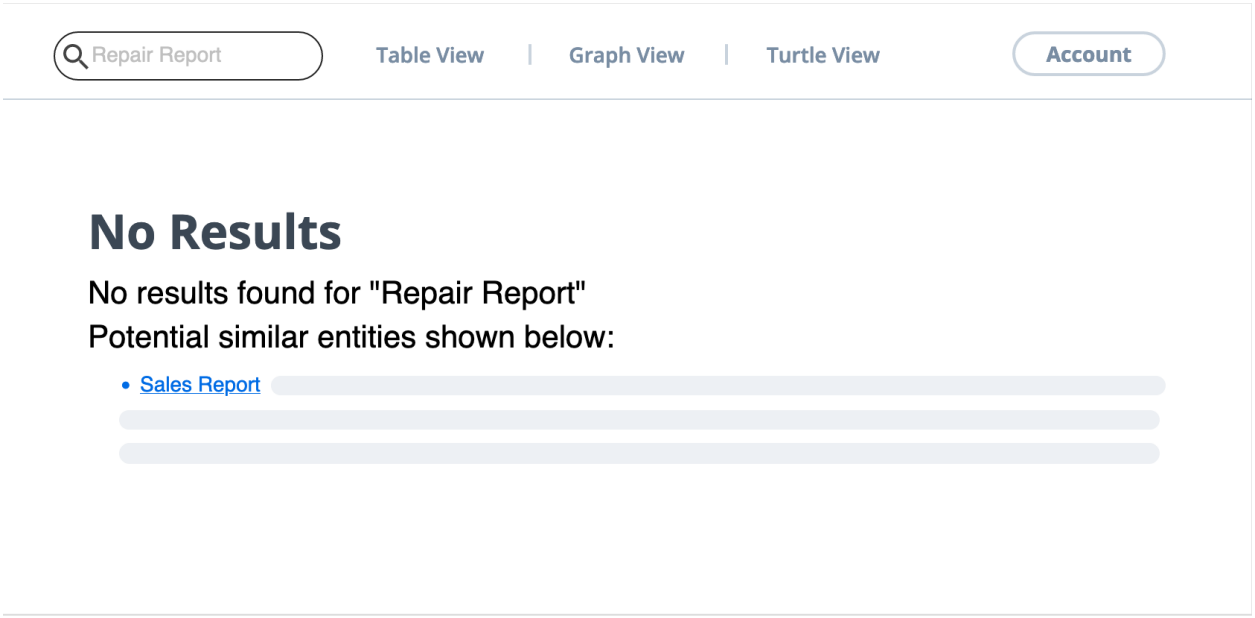

Figure 2 **For this scenario, we presume no "Repair Report" entity exists and the data provider has to look at other entities to create appropriate metadata.**

Figure 3 shows the asdm:vehicle class which is the representation of vehicle in the after-sales data model. The asdm:vehicle entity might be a subclass of another, more general vehicle entity and *inherit* some properties and limitations *vertically*. In this case, one might imagine the existence of a vehicle *class* defined in a global ontology that specifies that vehicles across all domains are either cars or trucks and can be associated with manufacturing plants. At the same time, the asdm:vehicle class is identifiably different from vehicle entities in other departments. For example, in the after-sales domain, vehicles have an owner (represented as asdm:owner) or a mileage (represented as asdm:mileage). These properties will not be present in other departments that deal with cars that have not been sold yet. At the same time, each vehicle entity is unique: asdm:vehicle is the only representation of vehicle in the after-sales department

Namespace Abbreviations

The entities shown in figures 3, through 9 indicate the ontology and domain that maintain them through abbreviations written before the colon. We provide a short overview of all example ontologies used below:

asdm:	After Sales Data Model
dbo:	DBpedia Ontology
dbr:	DBpedia Resource
ddm:	Data Exchange Data Model
gdm:	Global Data Model
pdm:	Production Data Model
qcdm:	Quality Control Data Model
rdf:	Resource Description Framework
rdfs:	RDF Schema
sdm:	Sales Data Model
xsd:	XML Schema

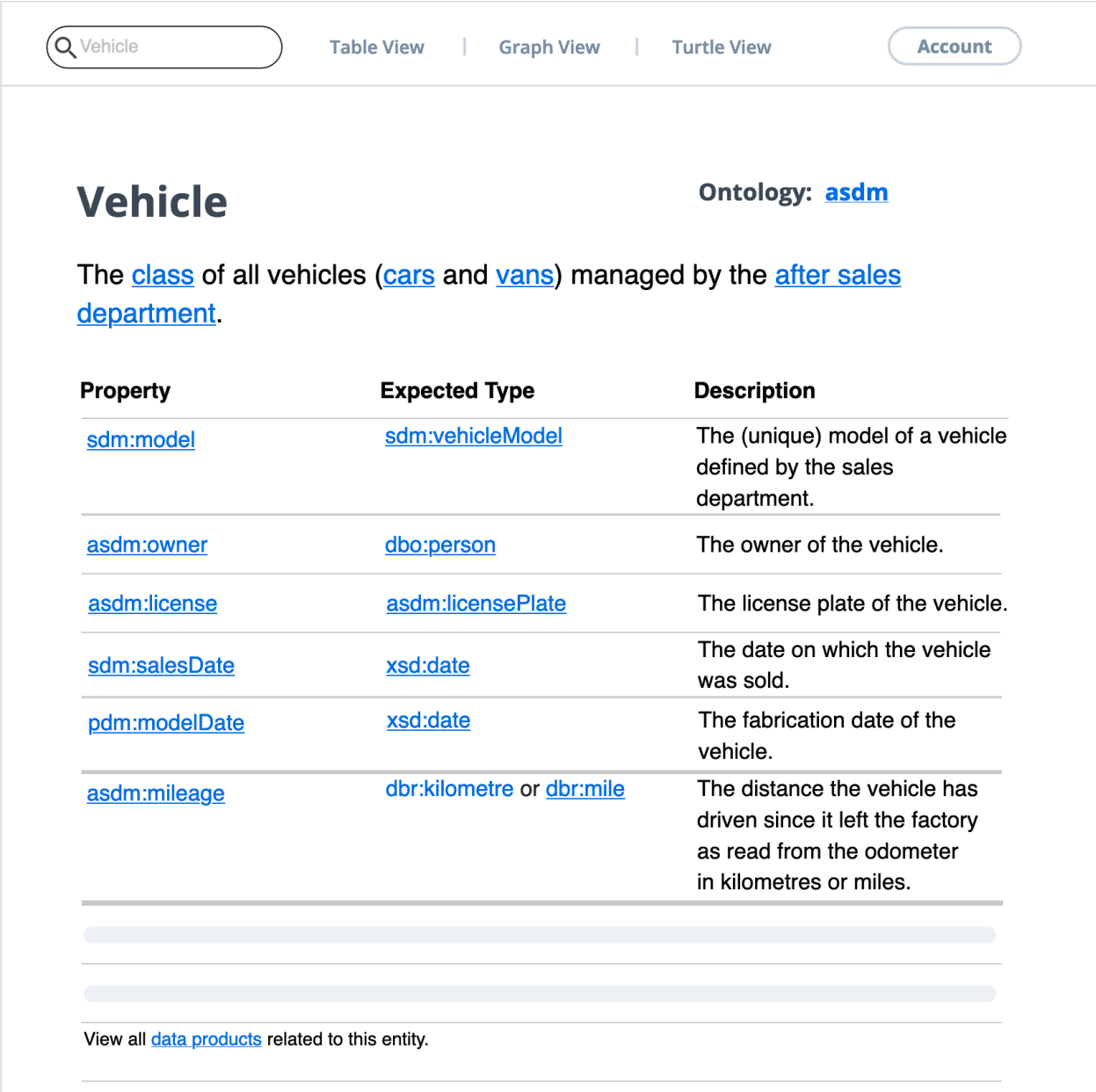

Figure 3 **The vehicle class represented in the after-sales domain model (asdm). Note that this is a class, not a data product. The class contains properties (edges in the knowledge graph) with expected types (nodes) and descriptions. The properties and types each have unique resource identifier (URI) that link them to a corresponding page. To distinguish between different entities, each URI starts with the URI of the ontology that defines that entity: asdm is for the after-sales data model, pdm is for the production data model, etc.**

data model and sdm:vehicle is the only representation of vehicle in the sales department data model. References to one of these are easily distinguished from references to the other.

In this scenario, the data provider finds three properties in asdm:vehicle that seemingly correspond to columns in their table: sdm:Model corresponds to "Car Model", sdm:salesDate corresponds to "Purchase Date" and asdm:mileage corresponds to "Mileage"[1]. The data provider can click each of these entities and their expected type to ensure these correctly describe their columns.

To illustrate this process, fig. 4 shows the entry for the sdm:vehicleModel, which is a curated set of permitted values that can be downloaded in the vehicleModels.txt file. Including such a set of permitted values in the knowledge graph promotes interoperability and reuse. This is reflected by the fact that any data provider who refers to this curated set as part of the metadata for their data product commits to using only the allowed values. Moreover, the sales department created the sdm:vehicleModel entity, not the after-sales department (as indicated by the sdm: predicate).

In order to create their data product, the data provider downloads the permitted values for all their columns, checks if their own values fall within the permitted range and changes them if necessary. Semantic web technology also makes it possible to create tools that automatically check compliance with required values, but we leave the explanation of such tools to future research efforts.

Using Entities from other ontologies.

As data products and metadata models evolve over time, disparities might appear between the data managed by one department and the metadata managed by another. Although this is not the main focus of this document, we give a brief outline here of how to deal with these disparities.

First of all, it is important to recognise that the entities in the metadata are described in terms of *business objects*, which are not expected to change frequently. Even though the data might be improved (e.g. higher granularity or more columns), the entities that describe it are expected to remain stable.

Suppose, however, that the data does change so that it no longer fits existing metadata or that the metadata changes and the data provider is informed that their metadata might no longer be up to date. In both cases, the data provider is encouraged to create their entities in their own domain ontologies that describe them and relate them to other entities. This effectively aligns the responsibilities for maintaining metadata with the same domain as the data provider.

After browsing the knowledge graph and determining relevant entities for all columns in their table, the data provider can create an entity to represent their data product as shown in fig. 5. The repair parts data product entity has standardised, reused metadata on each of the columns from the table: a column name, the corresponding entity in the knowledge graph and a description.

[1] The column (affected parts) will be discussed later on.

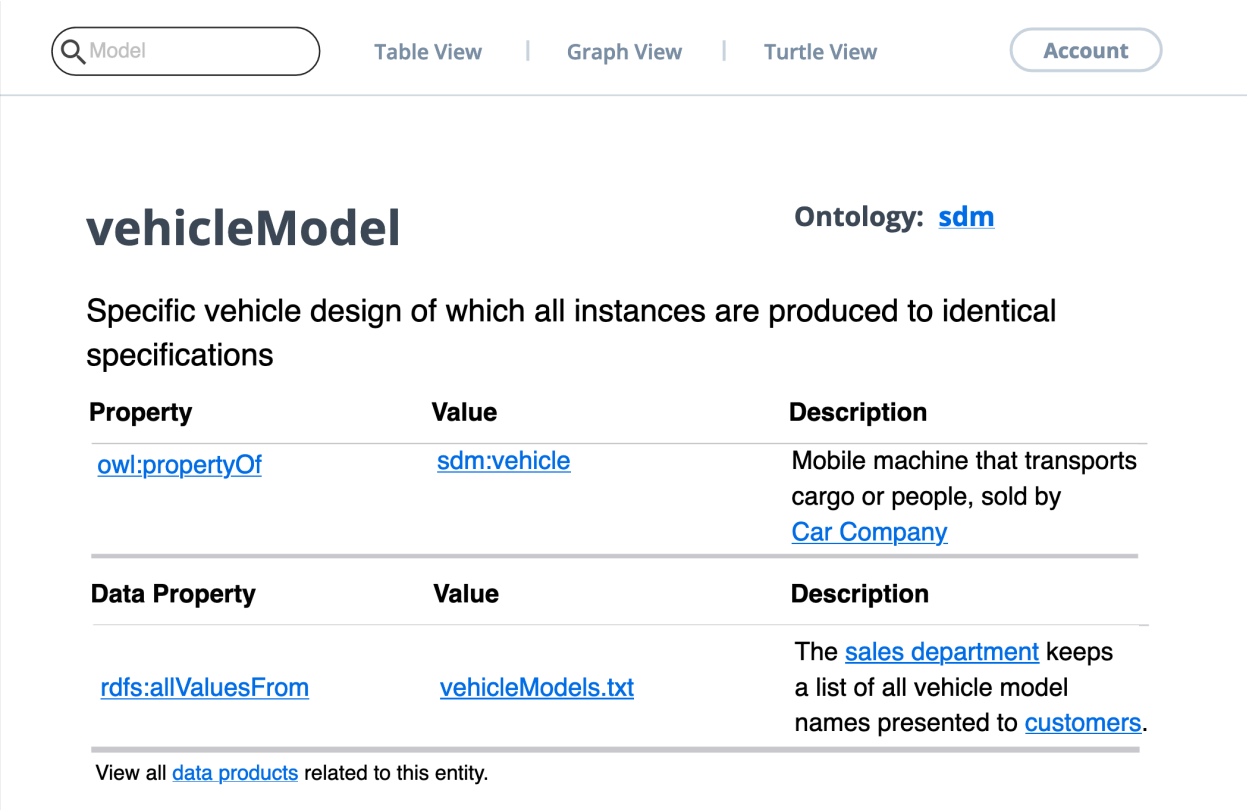

Figure 4 **The vehicleModel entity provides a curated set of permitted values. Any data provider who refers to this curated set as part of the metadata for their data product commits to using only these values.**

Note that the "Affected Parts" column has a corresponding entity from the production data model (pdm:entityId), because the data provider can reuse an entity from another domain ontology.

Additional metadata about the data product has also been added, such as the policies for access and consumption, the product owner, the data format, etc. Finally, links can be provided to the output ports of the data product, which allows data consumers to consume it easily.

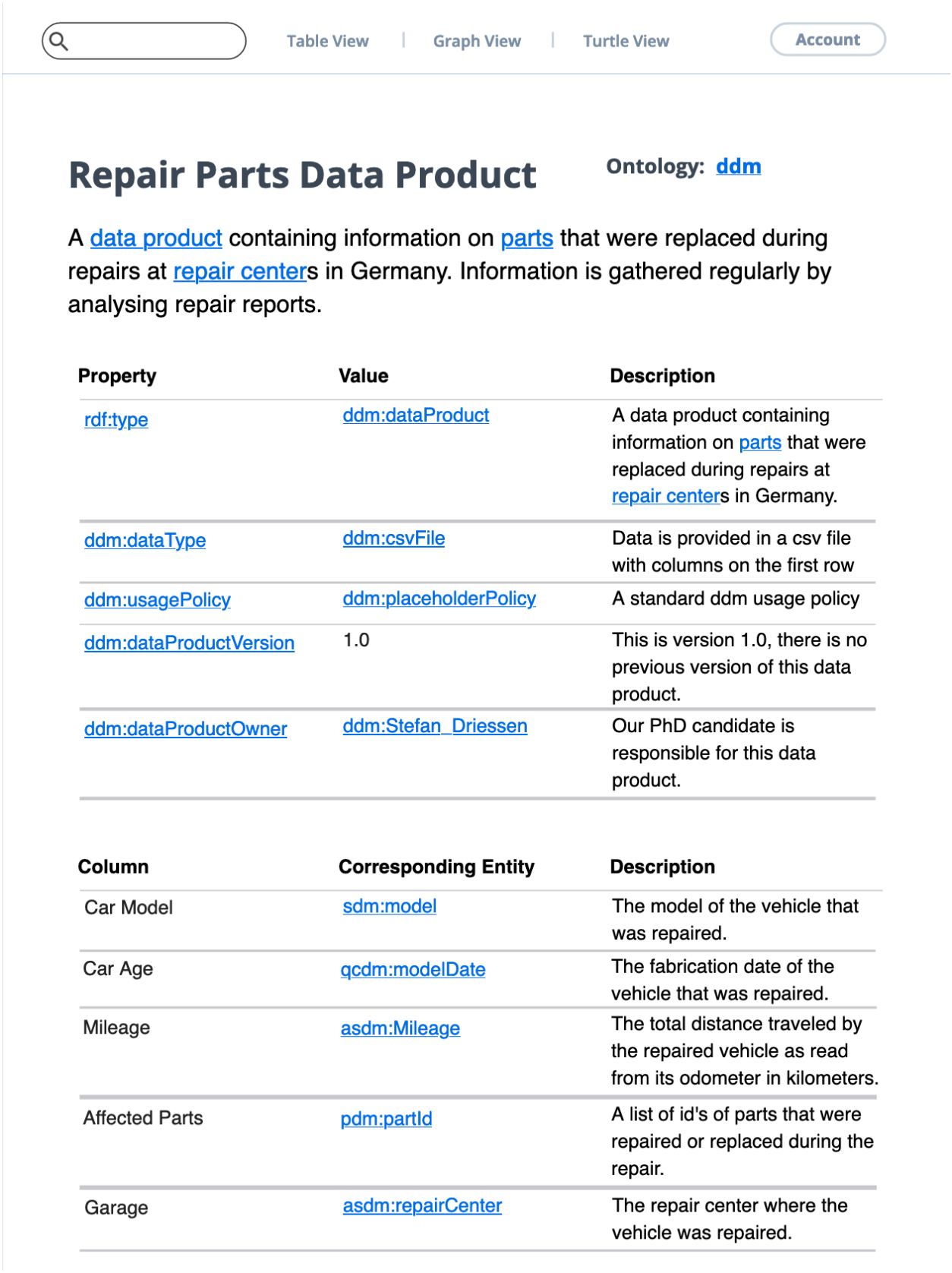

Figure 5 **The repair parts data product is part of the data exchange data model (ddm) and describes both properties of the data product such as their owner and access- and usage policies, as well as the content of underlying data asset.**

Scenario 2: Consuming an Existing Data Product

Another team in the after-sales department wants to optimise which parts to keep in supply in different warehouses. The goal of this new data consumer is to figure out which parts are generally held in store by repair centres and which parts are ordered as needed. If this were known, it would allow the automotive company's supply centres to prioritise parts that are ordered because they

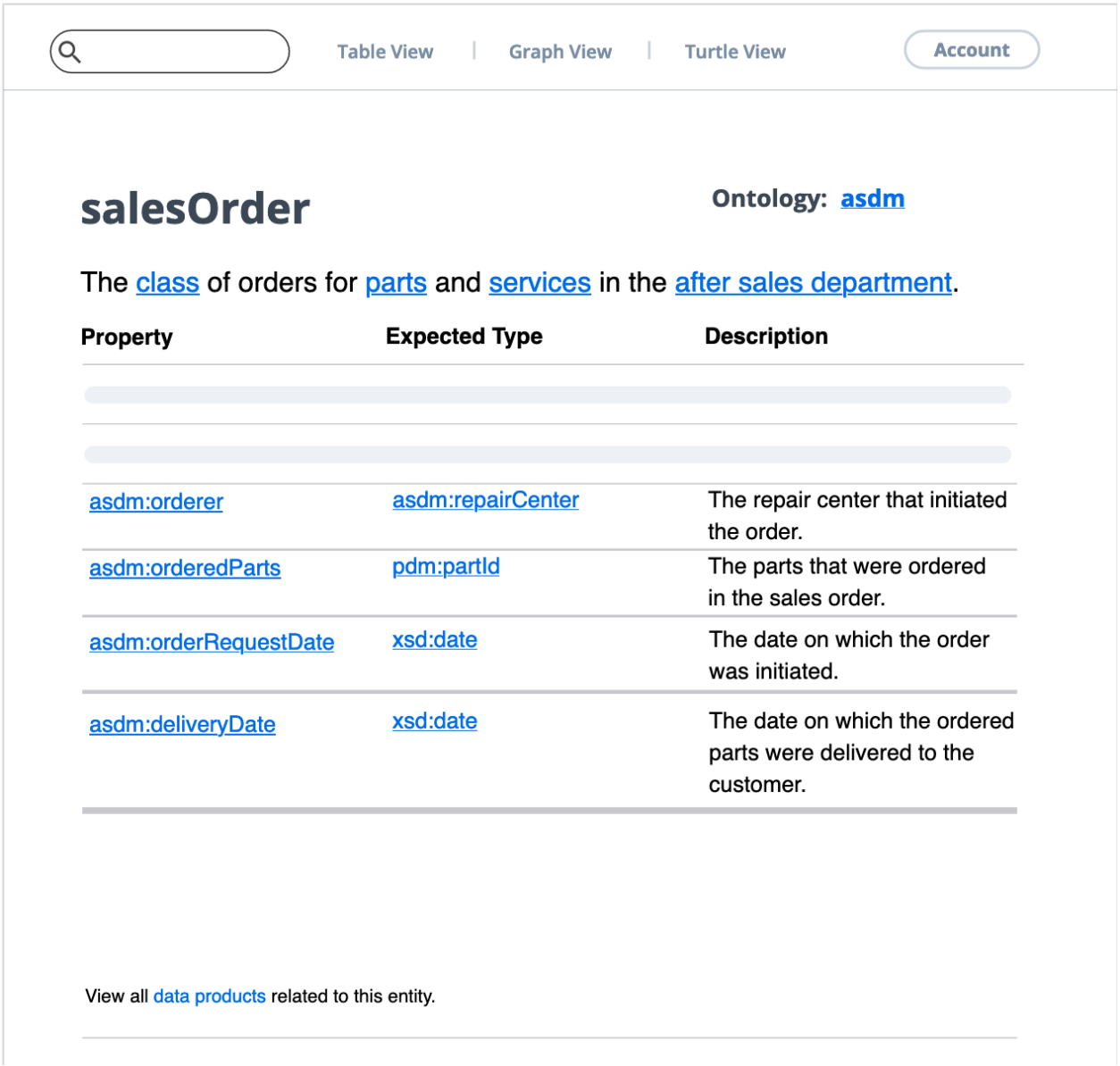

Figure 6 **The salesOrder entity as presented in the after sales data model (asdm). The entity represents a business object, specified as a rdfs:class entity. Business objects can be used as a sort of blueprint for creating data products. In this scenario, the data consumer knows that they have access to several useful properties such as orderer, orderedParts and orderRequestDate.**

are necessary for a specific repair over parts that are ordered to top up a repair centre's current supply.

In order to achieve this, the data consumer wants to compare information on when parts are ordered with information on when parts are used. The data consumer already has information from the after-sales department on their sales orders as shown in fig. 6. In particular, they have information on what parts were ordered, by whom, at what time and when they were delivered. Ideally, they want to combine this with information on when the repair centre needed these parts, specifically how long after needing the parts did the repair centre order new parts?

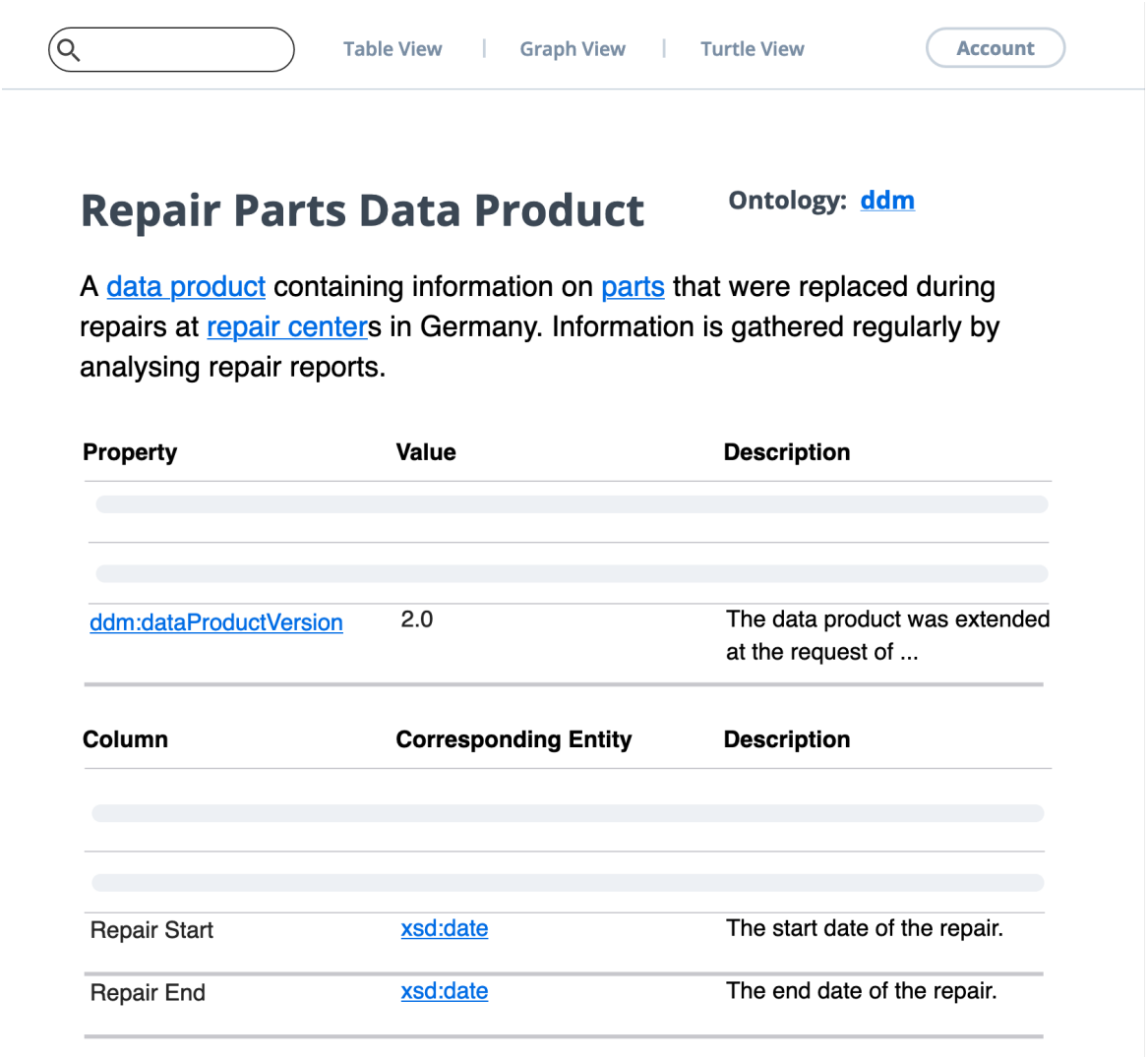

Figure 7 **As requirements change, so does the repair parts data product. The metadata is easily expanded: two new columns are created from scratch and the version of the data product is updated, with a description of the changes.**

After looking around on the internal data exchange, the data consumer concludes that this information is not yet presented as a data product. However, the data consumer does find the repairs parts data product introduced above, which contains something very similar: namely, parts that were replaced during repairs. Therefore, they decide to contact the data product owner, who is mentioned in the data product metadata and ask them if it is possible to add information on *when* the repair happened in which the parts were needed. After consultation, the original data provider decides to update the data product as shown in fig. 7.

After updating the data product, the data provider adds two new columns with information from the repair reports: the repair start date and the repair end date. It is important to note that, even though there were no existing corresponding entities in the domain ontologies to link these new columns to, the data consumer can still create metadata by linking them to the expected data type date (xsd:date). In addition, other metadata was also updated, such as a new version number (with description) and potentially a new output ports or API.

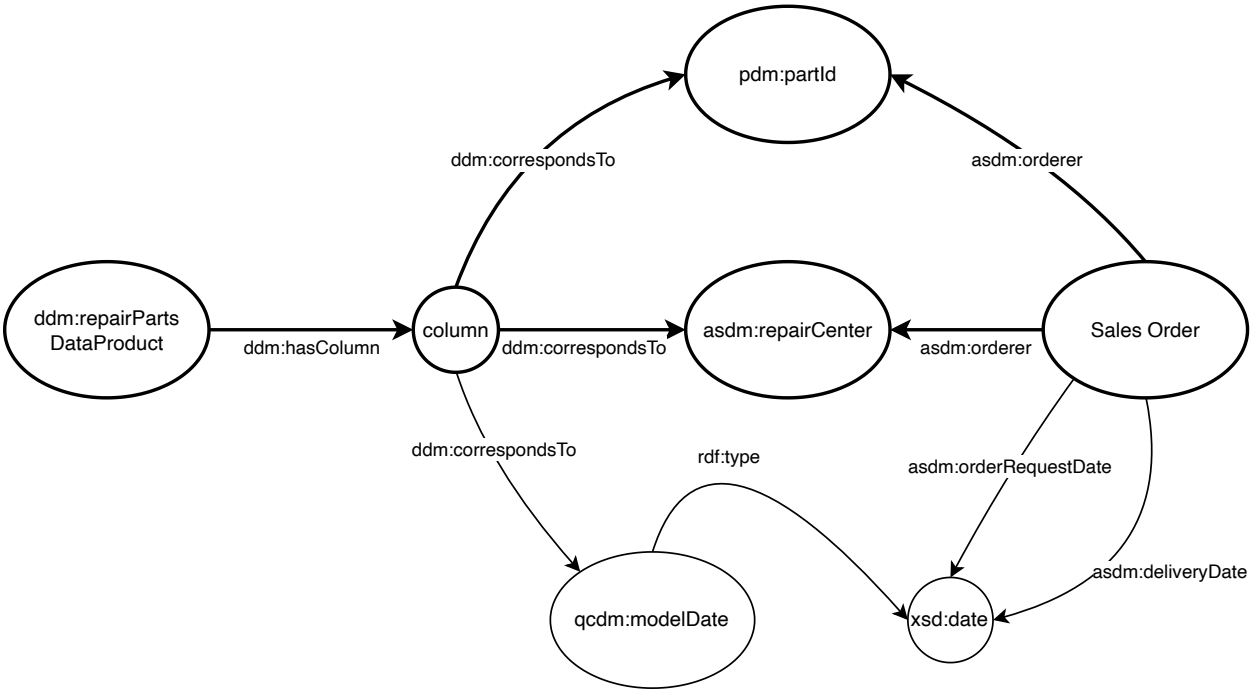

Figure 8 **Semantic web technology can use tools to automatically find connections between any two entities. Indeed, in this figure, we can see representations of data products, data types, classes and relations. In this scenario, the tool finds the connections between on the one hand a class (i.e. sales order) and on the other hand a data product (the asdm:repairPartsDataProduct). The thicker lines indicate the shortest paths between these two entities, these are relevant when trying to integrate data from the two sources.**

Even after the data product is updated, the data consumer still needs to figure out how to combine the different data sources. One of the benefits of semantic web technology is that it allows for the development of tools to assist in (partially) automated data integration. Figure 8 shows a mockup of how such a tool could help by showing the shortest path between two entities or data products that need to be linked. In this case, the automated tool recognises that the repair parts data product has several properties in common with the sales order entity. It suggests asdm:repairCenter and asdm:partId as the closest connection between the two. Both columns supposedly have curated set entities[2], and can thus act as effective keys for linking the two data products.

After combining the data, the data consumer decides to present their results as a data provider with a new data product on the internal data exchange. They create a corresponding data product entity in the knowledge graph, which is shown in fig. 9. Since this is a composite data product, the data provider makes sure to clearly include the sources in the description, effectively using other metadata to describe their data. In fact, most of the entities that the columns are linked to are the exact same as the entities shown above.

Composite Data Product Responsibility

Just like using entities from other ontologies brings the risk of disparities between metadata and data, so also does creating a data product that relies on data products from different teams across different domains. In a data exchange, the team that creates the composite data product is responsible for it. A good data exchange goes beyond designing proper systems for metadata and allows different teams to formalise agreements on data product use that allow for the necessary quality guarantees.

[2] Just like vehicleModel in fig. 4

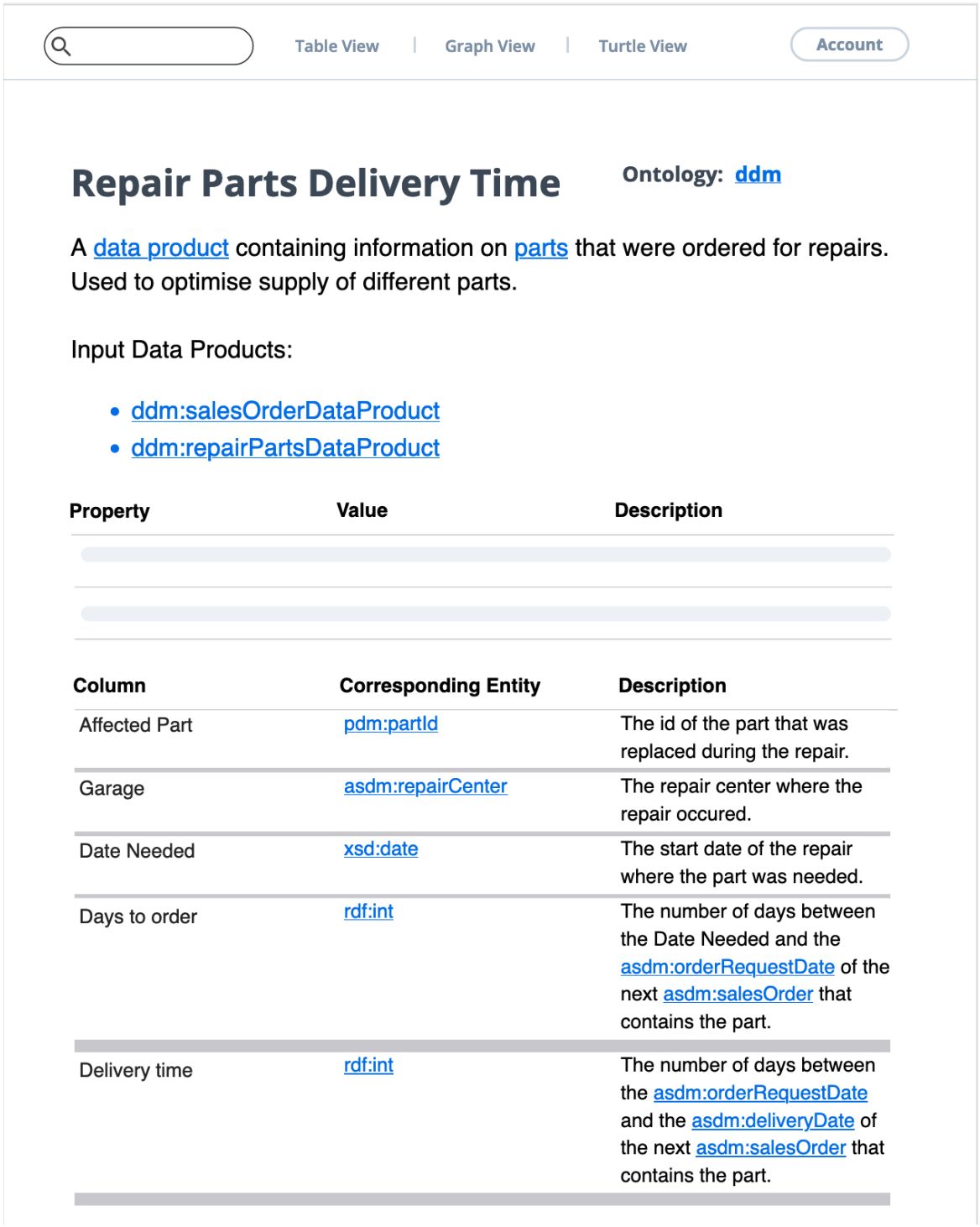

Figure 9 **The new data product that is created by integrating data from different sources. Both origins are indicated and the columns that come from other sources refer to the same entities as in the original data product.**

Scenario 3: Expanding Ontologies

In the previous scenario, we ended by noting that most properties and columns reuse the same data entities of the prior data products, leading to high interoperability. An exception to this reuse is the "Date Needed" column, which actually refers to the same entity as the repair start column in the repair parts data product in fig. 7. However, since there is no entity for repair start, there

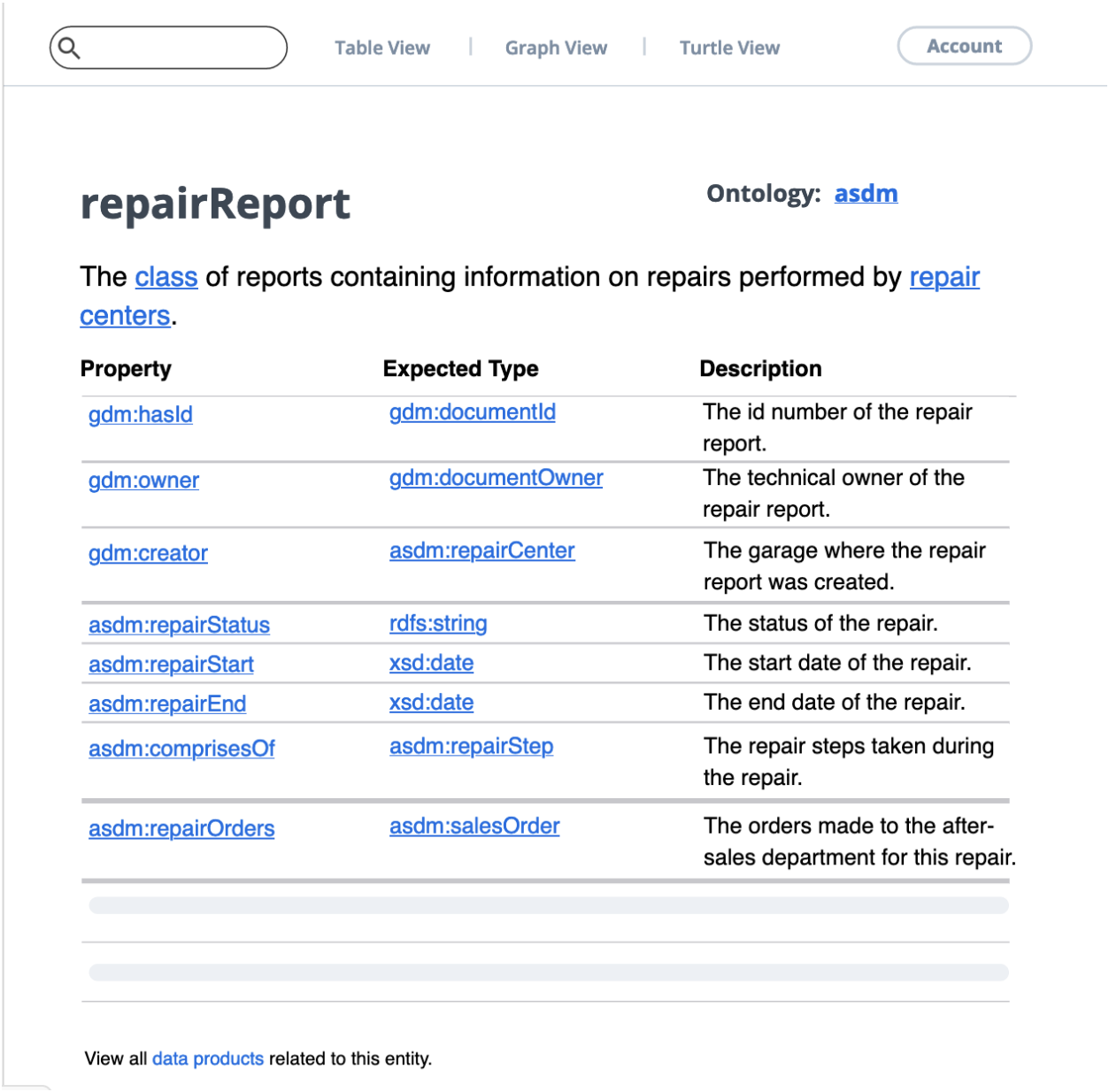

Figure 10 **Creating a class to represent the repair report business objects can be time consuming. Nevertheless, it can also lead to a number of benefits for data providers and consumers and reduce their future efforts.**

is no way to know this without demonstrating extensive knowledge of the different data products. This problem could be solved by extending the domain ontology to include an entity that describes the repair report as shown in fig. 10.

In general, the initiative to add new information to the knowledge graph can come from many directions. For example, it can come when new business processes are discovered, when a domain ontology expert tries to capture the domain logic or when a global ontology expert synthesises information from multiple domain ontologies. However, it can also come from data product creators who notice that their data products are about business objects that are not in the knowledge graph yet.

An example of the latter approach is the repair report shown in fig. 10, which provides information on the repair report entity that is useful for the data product discussed above. This entity could have been valuable for the above data products or for the creation of any completely new future data products. In fact, by providing a standardised entity, creating data products becomes more straightforward to include the repair centres themselves in the data product creation.

Additionally, adding this to the knowledge graph makes it possible for future data consumers to discover the potential of new data products. For example, the data consumer from the second scenario could have realised the existence of asdm:repairEnd and asdm:repairStart or other useful properties much more straightforwardly.

Expanding the domain ontology can also lead to the creation of new standard values such as such as a specification of the different "repair Steps". Moreover, it can help with future integration efforts by creating new and shorter paths between entities (e.g. a direct connection between "Date Needed" and "Repair Start"). Finally, relating the columns to entities in the knowledge graph allows for compliance checking with logic that can be imposed on semantic web technology. For example, we can specify in the repairReport class that repairEnd should always be after repairStart and automatically check future data products for compliance.

Reparaturbericht

Mechaniker	Lewis Hamilton	
Fahrzeug	Modell	Julius 4X-Drive
	Kaufdatum	01/11/2018
	Kilometerstand	16.000
	Marke	Car Company

Date	Problem/Diagnose	Fortschrittsaufzeichungen
22/02/2022	Scheinwerfer rechts kaputt	Aufnahme Fahrzeug
22/02/2022	Reparatur	Ersetzung Teil N100 012 34 50

Figure 11 **An anonymised repair rapport as a fillable form.**

Scenario 4: Creating a New, Similar Data Product

In the final scenario, we consider the case where the quality control department decides to scale their solution, by incorporating more data sources. For this purpose they decide to contact other garages and ask them to provide the same data in data products as before. Because the different garages are external, they will usually not have the same legacy system and, consequently, their data will be structured differently. We consider another garage, who creates repair rapports in fillable forms, such as the one shown in fig. 11.

It is clear that the data in this garage is structured in a significantly different manner than it was in the previous garage from fig. 1. Some of the more obvious differences are:

- Different formats: the first garage saves data as a table, whereas the second uses fillable forms (e.g. pdf).
- Different languages: the first garage uses English fields, whereas the second denotes everything in German.
- Different values: The second garage and the first use a different numbering system for the affected parts. In fact, in the second garage, the part id's can only be found inside a string that describes the repair step.

There are two ways that this data can be presented in a data product. In the ideal case, the quality control department can request from the data provider that they provide a data product with the exact same structure as the first data product from fig. 5. This approach fits best with the data mesh philosophy, whereby the data provider takes on all the responsibility for creating

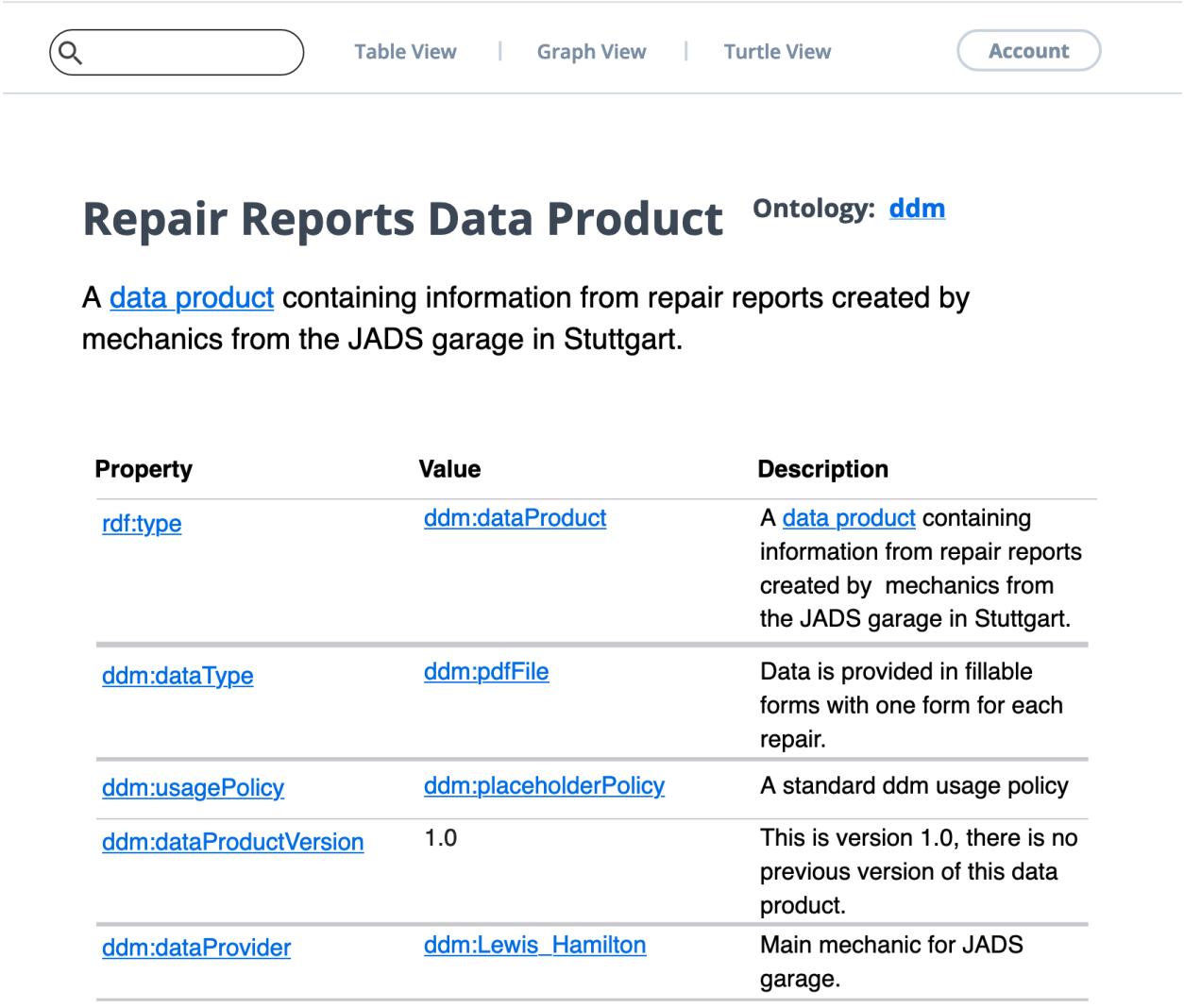

Figure 12 **Semantic metadata of the second data product. The metadata is modelled after fig. 5**

and transforming a data product. It might not always be feasible however, to ask this of the data provider. In particular, the second garage might lack the data expertise to create the transformation pipelines necessary to go from German fillable forms to English tables. In that case, they can still provide their data as-is. The second garage can use the metadata of the first data product as a template, and only make changes when necessary to describe how their domain differs from that of the first garage.

Figures 12 and 13 show the metadata for the new data product. It is clear to see that the metadata here follows the same structure as that of fig. 5. It is easy in this case for the second garage to change the metadata for the dataType to a pdfFile, which is described as a fillable file.

Field	Corresponding Entity	Description
Modell	sdm:vehicleModel	The model of the vehicle that was repaired.
Kaufdatum	xsd:date	The date of when the vehicle was first purchased.
Kilometerstand	asdm:Mileage	The total distance traveled by the repaired vehicle as read from its odometer in kilometers.
Date	xsd:date	The date when the vehicle was brought in for repair.
Fortschrittsaufzeichungen	xsd:string	The steps that were taken to repair the vehicle. This includes different parts that were replaced.

Figure 13 **Technical metadata of the second data product. The metadata is modelled after fig. 5**

Similarly, since the garage knows the data consumer is interested in the used parts, it provides the field where this can be found and describes explicitly that it can be found there.

This second approach to providing data is less optimal than the first, because it creates more work for the data consumer. Nevertheless, we believe that in some scenarios, such as when working with external garages without data engineering expertise, it is a more feasible approach.