FeriaDB: Implementing Knowledge Graphs for a Funfair Information System

Javier Carballal Morgade, Álvaro Adrada Martínez-Flórez

March 2025

Contents

1 Introduction

The domain of this project focuses on the structured representation of an amusement fair, which consists of multiple interconnected entities, including vendors, stalls, attractions, suppliers, and products. This concept is inspired by a project from a previous course at a Spanish university (Ingeniería de Software, Universidad Complutense de Madrid [1]) and has been further refined to fit better with the subject.

This database serves as the foundation for the construction of a knowledge graph that enables more advanced data retrieval, integration, and reasoning over stored information.

2 Building an Information System

The first steps are to design and implement a traditional relational database that accurately models the various components and relationships within the amusement fair.

2.1 The Domain

The information system facilitates coordination among various roles and components belonging to fair operations:

- **Fair Vendors**: People who operate within the fair, working at attractions or attending stalls.
- **Attractions**: Entertainment features designed for attendees. Attractions target different demographics, including children's attractions with height restrictions and general attractions suitable for all ages, varying in intensity.
- **Stalls**: Physical locations within the fair that sell various products, offering a range of merchandise and food items to fair visitors.
- **Products**: Items available for purchase at stalls, including both merchandise (such as branded souvenirs) and food products.
- **Suppliers**: External entities responsible for providing the products sold at the fair. They manage orders to ensure stalls remain stocked and operational.
- **Orders**: Requests for products made by stalls or fair management, managed by suppliers to facilitate the timely delivery and restocking of goods.

2.2 Entity-Relation Diagram

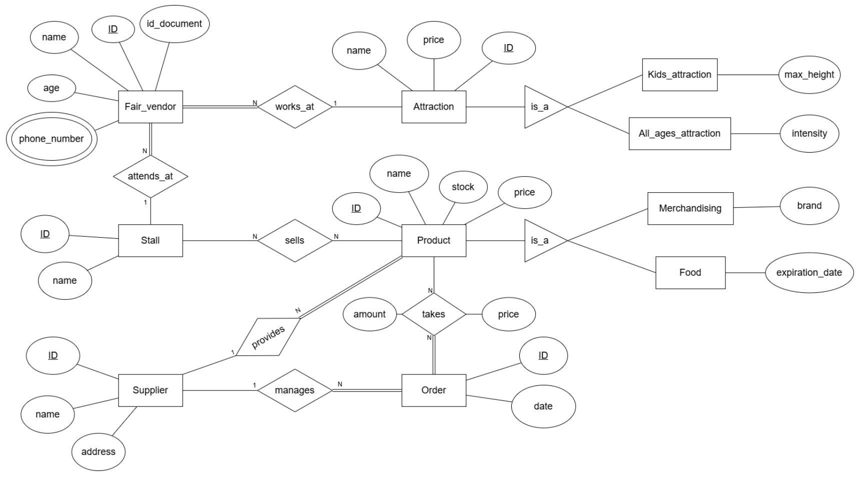

Figure 1: Entity-Relationship Diagram

This model is an accurate representation of the domain, having entities and relations corresponding to each of the elements mentioned.

2.2.1 Model Constraints

The following constraints are not explicitly enforced by the Entity-Relationship Diagram (ERD) and require additional validation through database constraints or application logic:

- A fair vendor can only work at either an attraction or a stall, but not at both simultaneously. Furthermore, they cannot work at more than one attraction or stall at the same time.
- The total price of an order must be calculated based on the sum of the prices of all products included in it.
- A fair vendor must have a valid phone number, ensuring effective communication.
- The price of an attraction must always be a positive value.
- The stock of a product must be a non-negative integer to prevent negative inventory.
- Orders must be associated with a specific date, and the date must be valid (i.e., no future orders beyond reasonable limits).

2.3 SQL Database

The SQL dump files included contain the code for the implementation of the model in SQL, as well as a handful of examples to get started.

To make sure our database is complete and realistic, we have created a dataset with a variety of examples. We based many of them on popular Spanish fairs, using real names for stalls, attractions, and vendors. This helps make the dataset more authentic and relevant to a real-world fair environment.

The dataset includes different types of entities, such as suppliers, products, stalls, attractions, and orders. For example, we added well-known stalls like "Churrería San Ginés" and "Tómbola de la Suerte", along with typical fair attractions like bumper cars and Ferris wheels. The products cover both food, such as churros and candy apples, and merchandising items such as hats and stuffed animals.

We also made sure that all foreign keys match correctly between tables. Every product has a supplier, every stall sells valid products, and every order includes real items from the database. This avoids errors and ensures that all the relationships in the database are properly maintained.

This dataset provides a good foundation for testing queries and validating constraints, making sure our database is well-structured and ready for further use.

3 Defining an Ontology

Our ontology design focused on capturing the key relationships and constraints of the domain while ensuring clarity and support for reasoning. Below, we explain specific modeling decisions made to improve consistency and usability.

3.1 OWL 2 Profile

The choice of the OWL 2 profile defines the reasoning capabilities and limitations of our ontology. Initially, we chose **OWL 2 DL**, outside of any specific profile, because of the expressive power that it offers. For instance, cardinality restrictions are not available in any subprofile of OWL 2 DL.

However, we realized that we could still enforce our needs for cardinality with *disjoint* properties, which would allow us to stay within **OWL 2 RL**. We think OWL 2 RL is the right choice for us because it guarantees decidability like OWL 2 DL, but its restrictions make it much more efficient and scalable, even over larger datasets. Since our domain model does not require features excluded from RL, adopting a more powerful profile would only increase computational cost without adding modeling benefits. It is true that for an application like this where the ontology is based on an existing relational database, OWL 2 QL is usually the best choice. However, it is not suitable for this case because it doesn't allow for expressions that we need such as *unionOf* and *disjoint*.

3.2 Defining and Organizing Concepts

The following points summarise the main principles that guided the naming strategy and structure of our ontology:

3.2.1 Semantics and Naming

- We chose the **namespace** `<http://example.org/fair#>`. We use fragments (`#`) for naming all T-Box elements. This strategy is motivated by the fact that in this way all terms can be defined within a single resource and retrieved in one request. This is good because it allows for better reasoning (engines have available the whole ontology when they make a request) and prevents the need to do multiple requests. Since the ontology is relatively compact and stable, this approach is manageable.
- **Class names** were chosen to directly correspond with entities from our domain, such as `Vendor`, `Stall`, `Attraction`, and `Product`.
- More **abstract concepts**, like `Marketplace`, were introduced to group concepts that shared some semantics. It is further refined through specialization: for instance, `ProductMarketplace` and `ServiceMarketplace` distinguish between spaces where goods or services are offered.
- We name relationships with **verb phrases** (e.g., `worksAt`, `hasOrderItem`, `belongsToOrder`), which allows to clearly define directionality and better readability.
- To avoid ambiguity, we defined **subclass relationships**, **inverse properties** (such as `hasOrderItem` / `belongsToOrder`) and used `owl:disjointWith` axioms where necessary - particularly in **mutually exclusive** categories like `Food` and `Merchandising`, or `KidsAttraction` and `AllAgesAttraction`.

The following snippet illustrates this structure in Turtle syntax:

```
:Marketplace rdf:type owl:Class ;
            rdfs:label "Marketplace" ;
            rdfs:comment "A general marketplace where transactions occur.
                          It can be a product or service marketplace." .

:ProductMarketplace rdf:type owl:Class ;
```

```
                    rdfs:subClassOf :Marketplace ;
                    rdfs:label "Product Marketplace" ;
                    rdfs:comment "A marketplace where products are sold
                                  (e.g., Stalls)." .

:ServiceMarketplace rdf:type owl:Class ;
                    rdfs:subClassOf :Marketplace ;
                    rdfs:label "Service Marketplace" ;
                    rdfs:comment "A marketplace where services are provided
                                  (e.g., Attractions)." .
```

3.2.2 Style and Annotation

- For **classes** we adopted an UpperCamelCase naming pattern (`Marketplace`, `Vendor`, `Stall`).
- For **properties** we use lowerCamelCase.
- All properties include domain and range declarations.
- Each resource is accompanied by a `rdfs:label` (for human-friendly naming) and an explanatory `rdfs:comment` (providing a gloss or detailed definition). These annotations act as documentation and make it easier for other users to integrate our ontology.

3.3 Challenges

Here are the main challenges we faced converting our domain to an ontology:

3.3.1 Orders

In our original model, an order was linked directly to products using an object property (`:takes`), while quantity and price were represented as separate data properties (`:takesAmount`, `:takesPrice`) attached to the order itself. This structure was problematic because it didn't specify which product each quantity and price referred to, making it impossible to represent multiple products within a single order unambiguously.

To address this, we restructured the model by introducing the `OrderItem` class to represent individual product entries within an order. This allowed us to explicitly associate each product with its specific quantity and unit price.

3.3.2 Price

Originally, we used a single property `:price` to represent the cost of both products and attractions, and had a separate one for orders. This led to semantic confusion, as the `:price` of a product (its default retail value) is conceptually different from the unit price at which it was purchased in an order, but they still have a semantic connection that we wanted to represent in some way.

To resolve this, we introduced a general property `:hasPrice` and modeled as subproperties `:price` and `:orderedUnitPrice` , each with clearly defined domains. In this way, we can define different price types that are all grouped in a superclass. This approach made the price semantics explicit, improved consistency across contexts, and allowed us to support reasoning over all price-related values when needed.

The following snippet shows how we modelled it:

```
:hasPrice rdf:type owl:DatatypeProperty ;
          rdfs:domain owl:Thing ;
          rdfs:range xsd:decimal ;
          rdfs:label "Has Price" ;
          rdfs:comment "Generic price property applying to
                        multiple contexts." .

:price rdf:type owl:DatatypeProperty ;
       rdfs:subPropertyOf :hasPrice ;
       rdfs:domain [ rdf:type owl:Class ;
                    owl:unionOf ( :Product :Attraction ) ] ;
       rdfs:range xsd:decimal ;
       rdfs:label "Price" ;
       rdfs:comment "Base or standard price for a product or
                     attraction." .

:orderedUnitPrice rdf:type owl:DatatypeProperty ;
                  rdfs:subPropertyOf :hasPrice ;
                  rdfs:domain :OrderItem ;
                  rdfs:range xsd:decimal ;
                  rdfs:label "Ordered Unit Price" ;
                  rdfs:comment "Price at which the product was
                                ordered (may differ from standard)." .
```

4 Creating the Mappings

In this step of the project, we aim to create mappings that define the process of populating our ontology with instances of data from our traditional MySQL database using the R2RML language. The result is a set of RDF triples that can be used with the ontology by reasoning tools to perform advanced queries and knowledge inferences.

4.1 Mapping Organization and Naming Strategies

To ensure maintainability and semantic clarity, we followed these organization and naming conventions:

- For the A-Box we use IRIs with slashes, like `http://example.org/fair/{type}/{ID}`. This means that each instance has its own full web address. We chose this format for several reasons. First, it makes the data easier to organise and scale: when every resource has its own IRI, we can store them separately instead of having everything in one file. Second, it allows us to prepare the data for web access in the future. For example, if we publish this project online, each IRI could show useful information about that specific resource. This would not be possible if we used fragment identifiers, because those all point to the same document, and we couldn't separate or serve them individually.

- Each mapping is encapsulated in a **Triples Map** that clearly represents one conceptual entity from the ontology (e.g., `Vendor`, `Product`, `Supplier`, etc.).
- **Triples Map names** follow the format `EntityNameTriplesMap`.
- **Predicates** used for property mappings were aligned with ontology terms (e.g., `:vendorName`, `:provides`, `:expirationDate`).
- **Join Conditions** were explicitly added when foreign key references existed between entities (e.g., between `OrderItem` and `Product`).

4.2 Mapping Examples

To illustrate the structure and different features of our mappings, we provide some examples below:

1. Basic Entity Mapping: Stall

```
<#StallTriplesMap>
  rr:logicalTable [ rr:tableName "STALL" ];
  rr:subjectMap [
    rr:template "http://example.org/fair/stall/{ID}" ;
    rr:class :Stall
  ];
  rr:predicateObjectMap [
    rr:predicate :stallName ;
    rr:objectMap [ rr:column "NAME" ]
  ].
```

2. Relationship between Two Classes: Supplier and Product This example demonstrates how a relationship between two classes can be modeled using a join. The supplier *provides* one or more products.

```
<#SupplierTriplesMap>

  [...]
  rr:predicateObjectMap [
    rr:predicate :provides ;
    rr:objectMap [
      rr:parentTriplesMap <#ProductTriplesMap> ;
      rr:joinCondition [
        rr:child "ID" ;
        rr:parent "SUPPLIER_ID"
      ]
    ]
  ].
```

3. Class Inheritance: Food and Merchandising The ontology includes subclasses of `:Product`, such as `:Food` and `:Merchandising`. We used separate mappings with the same subject template to express this subclassing, effectively creating multiple types for a single entity. This aligns with OWL's open-world assumption because it allows additional class assertions to be made at any time without contradicting existing information and allows entities like `Product` to be further described as specialized instances without duplication.

```
<#FoodTriplesMap>
  rr:logicalTable [ rr:tableName "FOOD" ];
  rr:subjectMap [
    rr:template "http://example.org/fair/product/{ID}" ;
    rr:class :Food
  ];
  rr:predicateObjectMap [
    rr:predicate :expirationDate ;
    rr:objectMap [ rr:column "EXPIRATION_DATE" ; rr:datatype xsd:date ]
  ].

<#MerchandisingTriplesMap>
  rr:logicalTable [ rr:tableName "MERCHANDISING" ];
  rr:subjectMap [
    rr:template "http://example.org/fair/product/{ID}" ;
    rr:class :Merchandising
  ];
  rr:predicateObjectMap [
    rr:predicate :brand ;
    rr:objectMap [ rr:column "BRAND" ]
  ].
```

4.3 Tooling and Technical Resources

To develop our mappings and generate the final knowledge graph, we used several tools that facilitated the transformation of our relational database into RDF triples and the visualization of the output.

- **R2RML-F Mapping Tool** [2]: We used this tool to define and execute our mappings from the relational database to RDF triples, following the R2RML standard.
- **MySQL Dump to CSV** [3]: This tool allowed us to convert our SQL database dump into multiple CSV files, which were then used as the input for the mapping process.
- **RDF Playground** [4]: We used this online tool to visualize the RDF triples generated from our mappings. It provided a graphical representation of the knowledge graph, which helped us verify the correctness of the data and the structure.
- **GRAPE** [5]: Initially, we started developing the mapping in Grape, but we decided to move to a classical IDE for more flexibility. Additionally, the R2RML-F tool [2] , being the one used in class, was a good fit for this assignment.

4.4 RDF Validation

To ensure the accuracy of our mappings and the consistency between our database and the generated knowledge graph, we performed a validation process. Specifically, we verified that the number of RDF instances generated for each entity type matches the expected number of records in the corresponding CSV source files.

Table 1 summarizes this comparison, showing that for every class in our ontology, the number of triples generated correctly aligns with the original dataset. This validation step gives us confidence in the correctness of our mappings.

Entity Type	**Source File**	**Expected Count**	**RDF Instances**
Supplier	`SUPPLIER`	10	10
Stall	`STALL`	10	10
Product	`PRODUCT`	10	10
Food	`FOOD`	4	4
Merchandising	`MERCHANDISING`	4	4
Vendor	`FAIR_VENDOR`	10	10
Attraction	`ATTRACTION`	10	10
Kids Attraction	`KIDS_ATTRACTION`	5	5
All Ages Attraction	`ALL_AGES_ATTRACTION`	5	5
Order	`ORDER_TABLE`	5	5
OrderItem	`ORDER_PRODUCT`	5	5

Table 1: Validation of RDF instance counts against CSV data sources

To visualize the output of our mappings and validate the generated RDF data, we used the RDF Playground [4] tool. More on that will be addressed later in the non-trivial demonstrator.

4.5 Challenges and Decisions

4.5.1 CSV Files Formatting

One of the main challenges we faced was ensuring that our dataset files were correctly recognized by the R2RML-F tool [2]. Initially, the tool did not parse our CSV files properly due to format inconsistencies. While there were probably multiple possible solutions, we decided to enforce a strict naming convention to avoid further issues.

Specifically, we renamed all CSV files using uppercase letters and applied the same rule to the column names within each file. Additionally, we made sure that both the file names and the attribute names matched exactly the ones used in the mapping definitions. This way, we guaranteed that the tool could correctly associate the data with the mappings and generate the corresponding RDF triples without errors.

4.5.2 Inconsistency between Dataset and Ontology

Another challenge we faced was the inconsistency between the initial dataset and the ontology structure we designed. Specifically, due to the way we modelled orders in the ontology, each product within an order required an associated price (`:orderedUnitPrice`).

However, the original dataset did not provide this information, since the `order_product` table only included the quantity of each product. To solve this issue, we manually added a price column to the `order_product` table in the dataset, assigning a specific price to each product in every order. This ensured full consistency between the dataset and the ontology, allowing us to correctly generate the corresponding RDF triples.

Although generally it is better to change the mappings rather than the original database, we feel like in this case changing the database is the best solution. This is because the domain actually requires that data (it figures in the ER diagram1) and can **not** be inferred from the retail price (as it may have changed since then). That's why we would say that it was an error of the database, and not of the mapping.

4.5.3 Incorrect Mapping of IDs

We had an issue with the IDs of some entities. For some of them, two separate IRIs were generated: one with the ID as an integer and another with a decimal point (e.g. `stall/2` and `stall/2.0`). We used RDF Playground [4] for verification, as shown in Figure 2, where it can be appreciated that there are two disconnected components, suggesting that both IRIs were incorrectly interpreted as distinct entities.

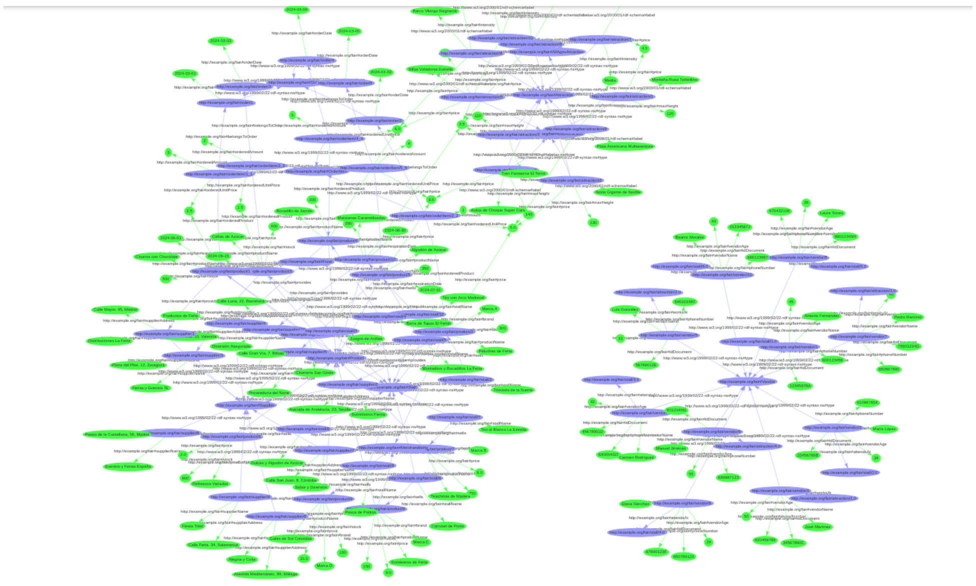

Figure 2: RDF graph generated from the incorrect mapping

In order to fix it, we faced many problems, because despite our fixes, R2RML-F [2] constantly interpreted the ID attribute as a double. To solve it, we used a subquery for the logicalTable that explicitly removed the lingering decimal point, as seen in the following code:

```
<#FairVendorTriplesMap>
  rr:logicalTable [
    rr:sqlQuery """
      SELECT ID, NAME, ID_DOCUMENT, AGE, PHONE_NUMBER,
             REPLACE(STALL_ID, '.0', '') AS STALL_ID,
             REPLACE(ATTRACTION_ID, '.0', '') AS ATTRACTION_ID
      FROM FAIR_VENDOR
    """
  ];
  rr:subjectMap [
    rr:template "http://example.org/fair/vendor/{ID}" ;
    rr:class :Vendor
  ];
```

5 Using the Ontology and Knowledge Graph

5.1 Deployment

To facilitate accesibility of our knowledge graph, we created a GitHub repository containing all key artifacts of the project: the ontology, instance data, R2RML mappings, documentation, and deployment tools.

We opted for a simple deployment strategy using Apache Jena Fuseki[6], a lightweight SPARQL server. We manually downloaded Fuseki[6] and bundled it inside the repository, along with scripts to load both the ontology and the data automatically at runtime.

To run the knowledge graph:

- On **Linux/macOS**, run `start-fuseki.sh`;
- On **Windows**, run `start-fuseki.bat`;

Fuseki then serves the graph at http://localhost:3030/fair, enabling SPARQL queries and updates. No installation or configuration is required beyond a working Java runtime.

5.1.1 Challenges and Decisions

Initially, we explored deploying Fuseki[6] through **Docker** to ensure environment consistency across different users. However, we encountered multiple problems:

- Official Docker images were difficult to use and customize, and lacked the web UI.
- Community images often had the same problems and we encountered problems setting them up.
- Some containers failed to launch due to invalid startup arguments or missing binaries.

Given time constraints and the educational focus of the project, we decided to drop Docker in favor of a **manual deployment** using the official Fuseki[6] download.

5.2 Validation with SPARQL Queries

We performed **ontology validation** by checking that the ontology meets the intended domain needs and supports meaningful queries over the data. Specifically, we designed seven non-trivial SPARQL queries to confirm that key assumptions and constraints from the domain are respected in the knowledge graph. This corresponds to answering the question "was the *right* ontology built?" rather than whether it was built *correctly* (ontology verification).

Our queries also relate to several quality dimensions. For example, checking vendors without phone numbers contributes to **completeness** (is the data sufficient to support operations?), while aggregations and subclass reasoning (e.g., vendor count by attraction type) help test the **clarity** and **consistency** of class definitions and subclass hierarchies.

Query 1: Vendors without a phone number

This query retrieves all vendors who do not have a phone number assigned. It is useful to detect missing or incomplete data, and it demonstrates the use of negation through `FILTER NOT EXISTS`.

Query 2: Product count per stall

This query counts how many products are sold at each stall. It validates the structure of relationships between stalls and products, and uses aggregation with `COUNT` and `GROUP BY`.

Query 3: Average age of vendors by assignment

This query calculates the average age of vendors depending on whether they work at a stall or at an attraction. It shows how conditional grouping and subqueries can be used to analyse data patterns using `AVG` and `COALESCE` logic.

Query 4: Vendors linked to products via stalls

This query finds vendors who are indirectly related to products through the stalls they attend. It tests the use of property paths to traverse multi-hop relationships in the graph.

Query 5: Vendors working in expensive attractions

This query returns vendors who work at attractions with a price higher than 4 euros. It combines filtering on numeric data properties with object relationships, and demonstrates conditional retrieval using `FILTER`.

Query 6: Vendor count by type of attraction

This query counts the number of vendors assigned to each type of attraction (kids or all-ages). It checks subclass usage in combination with aggregation and filtering by class type.

5.3 Visualization of the Knowledge Graph

As our chosen non-trivial demonstrator, we focused on **visualization** to make the structure and contents of our knowledge graph more accessible and interpretable. Rather than relying solely on textual or programmatic outputs, we wanted a way to present our ontology and data in a form that could be intuitively understood by both technical and non-technical users. It also provides a quick and straightforward way to make simple validations in the knowledge graph.

In order to make the resulting visualizations easier to understand and more manageable, we designed a smaller dataset with a few representative entries of the different classes.

5.3.1 WebVOWL

We first used **WebVOWL**[7], an interactive visualizer for ontologies. By loading our ontology into WebVOWL (https://service.tib.eu/webvowl/) [7], we obtained a dynamic graph representation of our schema. This allowed us to verify the class hierarchy, property domains and ranges, and subclass relationships in a visually meaningful way, making it easier to validate our design choices and communicate them clearly. An image of the interactive graph can be seen in Figure 3.

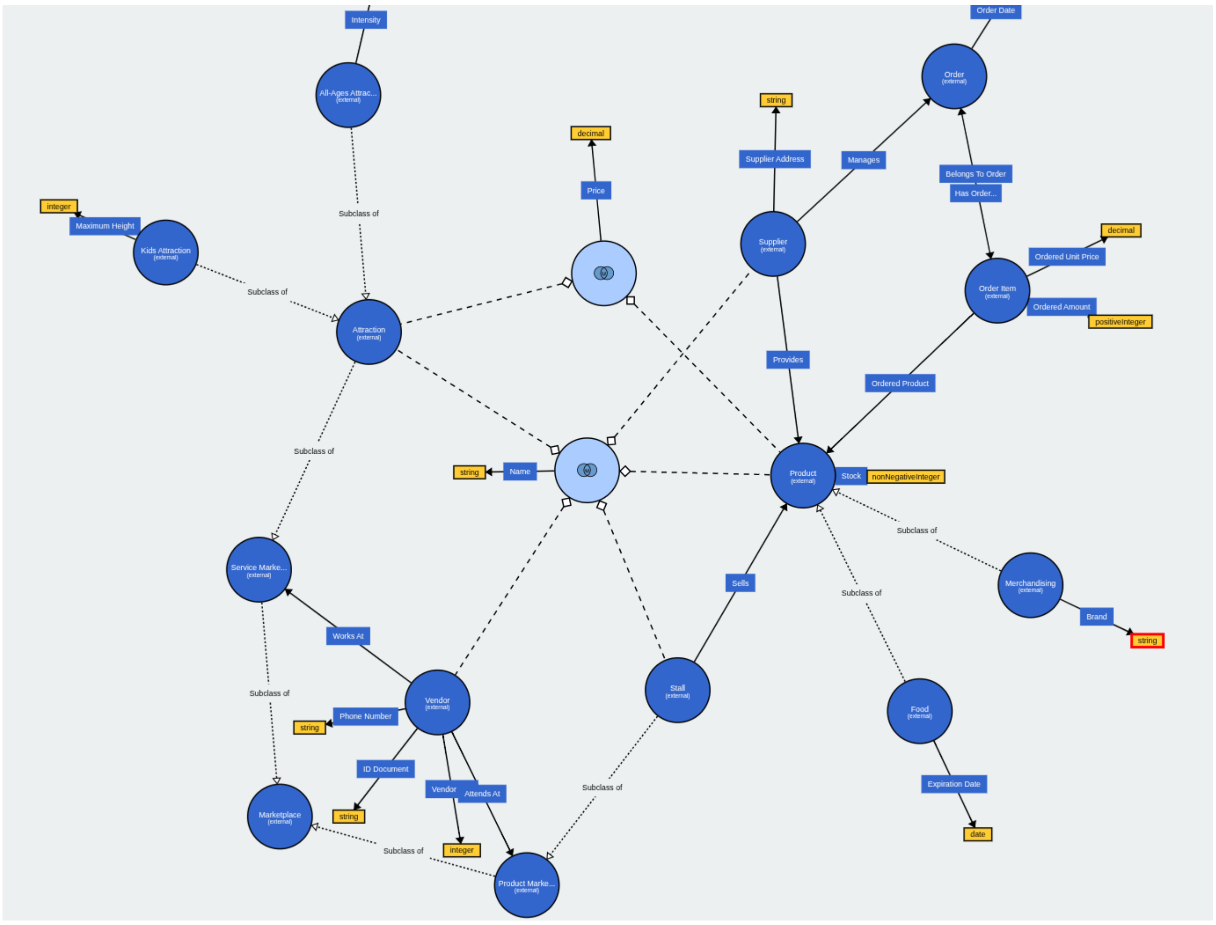

Figure 3: Ontology visualization generated with WebVOWL, showing the classes, properties, and constraints of the fair domain.

5.3.2 RDF Playground

To complement the structural visualization, we also employed the RDF Visualization Technology **RDF Playground**(https://rdfplayground.dcc.uchile.cl/)[4]. In addition to using WebVOWL[7] to visualize the ontology, this tool also allows to look at the data instances and perform SPARQL queries, at the tradeoff of being more complex and thus harder to understand as the number of items grows. With this setup, users could interactively explore the knowledge graph, test queries, and observe real-time responses, offering a concrete understanding of how our mappings materialized into RDF triples. The resulting graph can be seen in Figure 4.

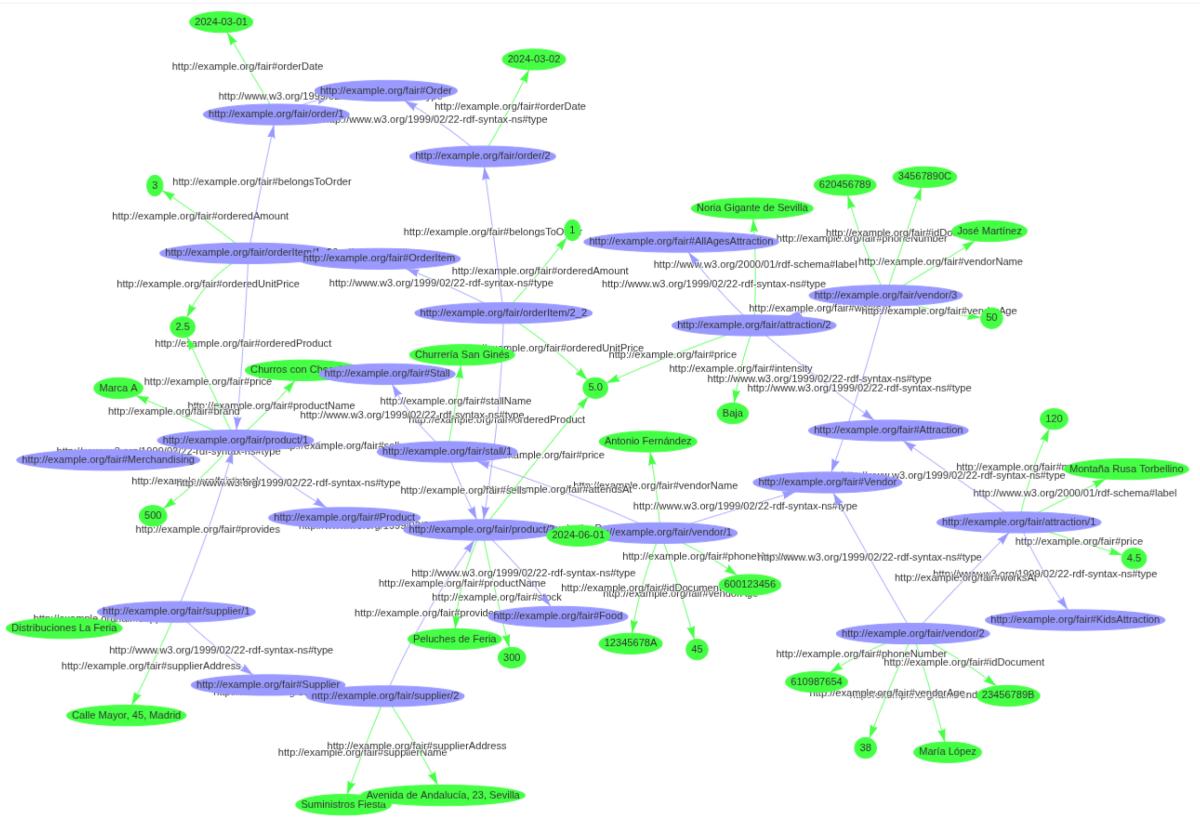
2024-03-01
http://example.org/fair#orderDate
http://example.org/fair#Order
http://example.org/fair/order/1
2024-03-02
http://example.org/fair#orderDate
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://example.org/fair/order/2
3
http://example.org/fair#belongsToOrder
http://example.org/fair#orderedAmount
http://example.org/fair/orderItem/1
http://example.org/fair#OrderItem
http://example.org/fair#orderedUnitPrice
1
http://example.org/fair#AllAgesAttraction
http://example.org/fair/orderItem/2_2
2.5
http://example.org/fair#orderedProduct
Churrería San Ginés
http://example.org/fair#Stall
5.0
Marca A
http://example.org/fair#brand
http://example.org/fair#price
http://example.org/fair#productName
http://example.org/fair#stallName
http://example.org/fair/product/1
http://example.org/fair#Merchandising
http://example.org/fair/stall/1
Antonio Fernández
500
http://example.org/fair#Product
http://example.org/fair/product/2
2024-06-01
http://example.org/fair/vendor/1
http://example.org/fair#provides
http://example.org/fair#sells
http://example.org/fair#attendsAt
http://example.org/fair#vendorName
http://example.org/fair#productName
http://example.org/fair#stock
http://example.org/fair#Food
http://example.org/fair#phoneNumber
http://example.org/fair#idDocument
600123456
Peluches de Feria
300
12345678A
45
http://example.org/fair/supplier/1
Distribuciones La Feria
http://example.org/fair#supplierAddress
http://example.org/fair#Supplier
http://example.org/fair/supplier/2
Calle Mayor, 45, Madrid
http://example.org/fair#supplierName
Suministros Fiesta
Avenida de Andalucía, 23, Sevilla
Noria Gigante de Sevilla
620456789
34567890C
José Martínez
http://www.w3.org/2000/01/rdf-schema#label
http://example.org/fair#vendorName
http://example.org/fair/vendor/3
http://example.org/fair/attraction/2
http://example.org/fair#vendorAge
50
http://example.org/fair#intensity
Baja
http://example.org/fair#Attraction
120
http://example.org/fair#Vendor
Montaña Rusa Torbellino
http://example.org/fair/attraction/1
http://example.org/fair#worksAt
4.5
http://example.org/fair#KidsAttraction
http://example.org/fair/vendor/2
610987654
23456789B
38
María López

Figure 4: Visualization of the RDF triples generated from a subset of the dataset using RDF Playground. The graph shows relationships between vendors, products, stalls, and other entities.

6 Conclusions

6.1 Summary

Over the course of this project, we gradually built a complete knowledge graph, starting from the design of an OWL ontology all the way to its deployment and demonstration. The chosen domain—a local amusement fair—provided a concrete and intuitive context in which to model entities such as vendors, stalls, attractions, and products, as well as their relationships.

This project gave us hands-on experience with semantic web technologies and deepened our understanding of knowledge representation and reasoning. By the end, we had constructed a functional and reusable knowledge graph that demonstrates both semantic modeling and data integration using established standards and tools.

We also paid attention to usability and accessibility: scripts were developed for both Linux and Windows environments to facilitate the deployment of the graph using Apache Jena Fuseki [6], and comprehensive documentation was provided in a GitHub repository. This made it easy to explore and query the data via SPARQL endpoints.

6.2 Lessons Learned

A first important lesson we learned was that translating an initial conceptual model—such as an entity-relationship diagram or a relational database schema—into a functional OWL ontology is more challenging than it initially appears. Many elements that work in a relational model, like table structures or foreign keys, do not translate directly or meaningfully into semantic representations. This forced us to rethink and adapt much of the original structure. Relationships had to be formalized, classes redefined, and the overall model restructured to reflect semantics rather than database logic.

Trying to use Docker when it was outside the scope of the project was not a very wise idea, because it consumed much time and effort without providing any significant advantage. Instead, we should have made a previous assesment to determine whether it was worth it to direct our efforts in that direction.

6.3 Limitations

One limitation of our implementation concerns the way the relational database was populated. Although we did use a relational database as required, the data was loaded from `.csv` files instead of being imported from a proper SQL dump, which was the expected approach according to the project guidelines. This did not cause any problems with the final RDF output as the mapping worked correctly, but it still differs from what was originally asked. In the future, it would be worth exploring the full capabilities of R2RML-F by connecting directly to a relational database populated via SQL, as intended.

Second, the non-trivial demonstrator we selected—ontology visualization—while useful for internal validation and understanding, does not provide a strong real-life application value. In practical terms, visualizations like those from WebVOWL are exploratory tools intended for developers or ontologists rather than end users. As such, the demonstrator lacks practical utility beyond the development and debugging phases of ontology engineering.

Finally, while the SPARQL queries we implemented are functional, they are relatively simple. This is partly because we chose to model a wide range of concepts across different aspects of the domain—vendors, stalls, products, attractions—rather than going deep into one specific area. In addition, the implementation of some queries are tailored to assumptions in our dataset, such as attractions being only for children or for all ages, and products being either food or merchandising. This limits the flexibility of the queries if used with external datasets that use a similar domain but follow different categorizations or have a more complex structure.

7 Our Team

7.1 Milestone 1: Database

The database used for the project is based on a design originally created by Álvaro Adrada in a previous year. All three team members thoroughly analyzed and understood the database structure, and collectively adapted it to the context of this subject. The report for Milestone 1 reflects this work, including additional requirements and constraints aligned with the specific objectives of the course.

7.2 Milestone 2: Ontology Engineering

For Milestone 2, Quentin developed an initial version of the ontology, capturing the core classes and relationships based on the project's domain in addition to some annotations for the report. Javier and Álvaro then revised and extended this foundation. They also produced the final version of the technical report, ensuring consistency between the ontology and its documentation.

7.3 Milestone 3: Annotating your IS (i.e., creating mappings)

Álvaro and Javier were responsible for developing the mappings required to annotate the database according to the ontology. They carefully analyzed the structure of the database and applied the corresponding mapping strategies to align it with the conceptual model defined in the ontology. Additionally, they documented the whole process in detail, including the decisions made, the challenges encountered, and the naming strategies for both mappings and RDFs generated.

7.4 Milestone 4: Using the Ontology and Knowledge Graph

Álvaro and Javier were responsible for all the work related to this milestone. They researched different options for faceted browsing and visualization, and eventually chose to focus on knowledge graph visualization as the most suitable non-trivial demonstrator for the project.

They prepared the content of the report, including the deployment section, and designed the advanced SPARQL queries to demonstrate techniques such as negation, aggregation, paths, and subqueries. They also set up the visualization demonstrator and created the accompanying video to showcase the result.

References

[1] Universidad Complutense de Madrid, "Ingeniería de Software - Ficha Docente 2024." [Online]. Available: https://web.fdi.ucm.es/UCMFiles/pdf/FICHAS_DOCENTES/2024/8732.pdf

[2] C. Debruyne, "R2rml mapping tool." [Online]. Available: https://github.com/chrdebru/r2rml

[3] J. Mishra, "Mysql dump to csv," 2016. [Online]. Available: https://github.com/jamesmishra/mysql-dump-to-csv

[4] Department of Computer Science, University of Chile, "RDF Playground." [Online]. Available: https://rdfplayground.dcc.uchile.cl/

[5] J. Duchateau, "Grape: Graph-based rdf mapping tool." [Online]. Available: https://jakubduchateau.gitlabpages.uliege.be/grape/

[6] Apache Software Foundation, "Apache Jena Fuseki - SPARQL Server." [Online]. Available: https://jena.apache.org/documentation/fuseki2/

[7] S. Lohmann, S. Negru, F. Haag, and T. Ertl, "Webvowl: Web-based visualization of ontologies," https://vowl.visualdataweb.org/webvowl.html, 2014, accessed: 2025-05-10.