Master Degree in Computer Science

KEBI Course

Intelligent Meal Suggestion System for Restaurants Using Knowledge-Based Solutions and Ontology-Driven Meta-Modelling

Students
Michele Martini

Supervisor
Prof. Knut Hinkelmann
Prof. Emanuele Laurenzi

A.A. 2024/2025

Contents

List of Figures

Listings

1. Introduction

This document outlines the process followed for the development of the Knowledge Engineering and Business Intelligence project. This first chapter introduces the context and goals of the project; Section 1.1 describes the system to be developed, while Section 1.2 defines the set of tasks to be accomplished. Chapter 2 explores the foundations of Knowledge-Based Solutions, detailing the reasoning approach adopted and the design of decision models. Chapter 3 presents the implementation of the proposed solution, focusing on the integration of Agile methodologies and Ontology-based Meta-Modelling within the system architecture. Finally, Chapter 4 offers personal opinions and perspectives on the solutions provided.
The two main tasks of the project are documented separately and can be accessed through the following links:
Task 1: https://github.com/MartiniMichele/KEBI-personalized-menu
Task 2: https://github.com/MartiniMichele/Ontology4ModelingEnvironment.

1.1 Project Description

Many restaurants have their menus digitized. Guests can scan a QR code and have the menu presented on their smartphones. A disadvantage is that the screen is very small and it is difficult to get an overview, in particular, if the menu is large. However, some guests can not or do not want every meal, e.g. vegetarians or guests with an allergy. Instead of showing all the meals that are offered, it would be preferable to show only those meals the guest prefers. The objective of the project is to represent the knowledge about meals and guest preferences and create a system that allows to select those meals that fit the guest preferences. The knowledge base shall contain information about typical meals of an Italian restaurant, e.g. pizza, pasta, and main dishes. Meals consist of ingredients. There are different types of ingredients like meat, vegetables, fruits, or dairy. For each ingredient, there is information about the calories. Guests can be carnivores, vegetarians, calorie-conscious, or suffer from allergies, e.g. lactose or gluten intolerance.

1.2 Project Tasks

The project is structured around two main tasks:

- **Task 1:** Create different knowledge-based solutions for recommending food depending on the profile of a guest (carnivores, vegetarians, calorie-conscious, suffering from allergies etc.) using the following representation languages: – Decision Tables: including DRD with sub-decisions and corresponding decision tables. – Prolog: including facts and rules. – Knowledge graph/Ontology: including rules in SWRL, queries in SPARQL and SHACL shapes.

- **Task 2:** Agile and ontology-based meta-modelling: adapt BPMN 2.0 to suggest the meals for a given customer. For this, you can re-use or extend the knowledge graph/ontology created in the previous task. One option that you have is to specify the class BPMN Task with a new class and add additional properties, similar to what we have done in class with the Business Process as a Service case. Think of a new graphical notation for the new modelling element, which could be easy to understand for the restaurant manager. Use the triple store interface (Jena Fuseki) to fire the query result.

2. Knowledge Based Solutions

In this chapter is provided an overview of knowledge based solutions and the methods by which they were implemented to ensure that the appropriate meals are recommended to the correct types of guests. Firstly, by using Decision Tables (Section 2.1) to match user preferences and filter out undesired meal. With Prolog (Section 2.2), we can extend the logic provided by the Decision Tables by applying the functionalities provided by a programming language. This allows the manipulation of data and perform even more actions. Finally, using Knowledge Graphs (Section 2.3), we can store our data in a convenient format and query them as if they were stored in a database.

2.1 Decision Model

In the realm of business analysis, the Decision Model and Notation (DMN) stands out as a recognized standard. This methodology finds application in delineating and structuring recurring decisions facilitating a format where decision models can be seamlessly shared across entities. By adhering to this standard, companies can establish a unified modeling notation, empowering them to make effective decision making.

2.1.1 DMN Elements

The DMN standard consists of four essential components:

- **Decision Requirements Diagrams (DRD):** These diagrams clarify the relationships between various decision-making elements, creating a network of dependencies.
- **Decision Tables:** These tables provide a clear visual representation for defining actions based on specified conditions.
- **Business Context:** This refers to the contextual factors that affect decision-making within the organization.
- **Friendly Enough Expression Language (FEEL):** FEEL3 is a Low-Code language used for evaluating expressions found within decision tables.

For this project, the web-app Trisotech Decision Modeler has been used. It is part of the Trisotech Digital Enterprise Suite, where you can draw and model business decision models.

2.1.2 Decision Tables

In this section, is provided a description of each decision table with reference to image 2.1. There are two main decision tables:

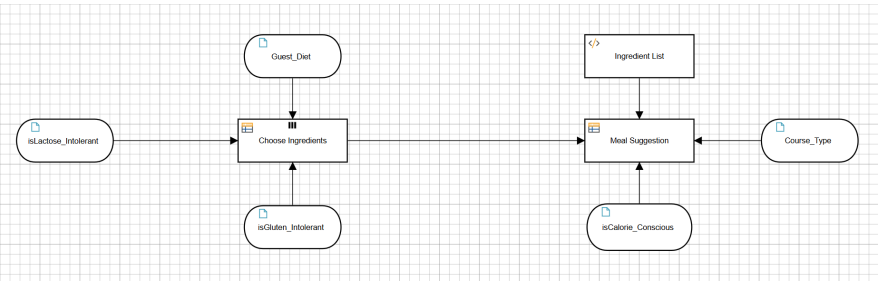

Figure 2.1: Decision model in Trisotech

- **Choose Ingredients:** This decision table (Figure 2.2) processes multiple input parameters to identify suitable ingredients for a user based on their dietary restrictions and preferences. The main input is the Guest Diet, which defines the general diet of the user:
  - **Carnivorous**: users who consume meat but avoid vegetables.
  - **Vegetarian**: users who exclude meat from their diet.
  - **Omnivore**: users who consume all types of food.

  This structure has been designed for extensibility, it can be easily expanded with more categories in the future, supporting more dietary types. In addition to the Guest Diet, the table considers two boolean intolerance parameters: is Lactose Intolerant and is Gluten Intolerant. Each of these boolean inputs specifies whether the user should avoid ingredients containing the corresponding allergen. In the decision table, a hyphen (-) indicates that the ingredient is free of the allergen and is therefore safe for all users. As you can see in Figure 2.3, a false value means that the ingredient does contain the allergen and thus will be excluded from the output if the user has that specific intolerance. The output of the table is a list of ingredients that match the user's profile.

	inputs			outputs	annotations
C	isLactose_Intolerant	isGluten_Intolerant	Guest_Diet	Choose Ingredients	Description
	Boolean	*Boolean*	*Text* *"carnivorous", "vegetarian", "omnivore"*	*list*	
1	-	-	"vegetarian","omnivore"	"Courgette"	
2	-	-	"vegetarian","omnivore"	"Eggplant"	
3	-	-	"vegetarian","omnivore"	"Lettuce"	
4	-	-	"carnivorous","omnivore"	"Shrimp"	
5	-	-	"carnivorous","omnivore"	"Pork Jowl"	
6	-	-	"carnivorous","omnivore"	"Beef Steak"	
7	-	-	"carnivorous","vegetarian","omnivore"	"Butter"	
8	-	-	"carnivorous","vegetarian","omnivore"	"Sage"	
9	-	-	"carnivorous","vegetarian","omnivore"	"Rice"	

Figure 2.2: Choose Ingredients Decision Table

9	isLactose_Intolerant	isGluten_Intolerant	Guest_Diet	Choose Ingredients	Description
10	-	-	"carnivorous","vegetarian","omnivore"	"Egg"	
11	-	-	"carnivorous","vegetarian","omnivore"	"Tomato Puree"	
12	-	-	"carnivorous","vegetarian","omnivore"	"Potato"	
13	-	-	"carnivorous","vegetarian","omnivore"	"Peach"	
14	-	-	"carnivorous","vegetarian","omnivore"	"Apricot"	
15	-	-	"carnivorous","vegetarian","omnivore"	"Watermelon"	
16	-	false	"carnivorous","vegetarian","omnivore"	"Pasta"	
17	-	false	"carnivorous","vegetarian","omnivore"	"Breadcrumbs"	
18	false	-	"carnivorous","vegetarian","omnivore"	"Pecorino"	
19	false	-	"carnivorous","vegetarian","omnivore"	"Mozzarella"	
20	false	-	"carnivorous","vegetarian","omnivore"	"Parmigiano"	

Figure 2.3: Intolerances in Choose Ingredients Decision Table

- **Meal Suggestion:** This decision table (Figure 2.4) is responsible for generating personalized meal suggestions, each accompanied by its total calorie count, based on the user's dietary profile, preferences, and meal context. It operates by filtering and combining the results produced by the preceding Ingredients decision table, which already considers the user's diet category (carnivorous, omnivore, vegetarian), along with intolerance filters (lactose and gluten). The table takes one indirect input from the previous ingredient decision table, ensuring that only suitable ingredients for the specific guest are considered in meal composition. A Literal Expression called "Ingredient List" is used to encapsulates the ingredients and their corresponding calorie values in the form of a Map (Figure 2.5). In addition, this decision table requires two input parameters to refine the suggestions:

  - **Calorie Conscious Level:** a custom integer attribute ranging from 0 to 1 that reflects the user's attention to caloric intake:
    - 0: the guest has no concern for calories and is open to all kinds of food. TotalCalories >250
    - 1: the guest is concerned about calories and is strict when choosing food. TotalCalories <= 250

    this attribute has also been designed with extensibility in mind, in the future will be easy to add more calorie conscious level.
  - **Course Type:** a string indicating the desired type of meal course, such as "First Dish", "Second Dish", "Dessert". This allows the table to select meals appropriate to the given meal phase or type.

  The output of this decision table is a list of meal suggestions, each one composed of meal name and the total calories for the full meal. To determine whether a meal can be suggested for a given user and course type, the decision table uses FEEL expressions in its output columns. Specifically, it applies the list contains() function and the logical "and" operators to verify that all the ingredients required for a given meal are included in the list of available ingredients. This

ensures that only those meals that can be safely prepared with accessible and tolerated ingredients are considered valid outputs. If all of these conditions are true, and the calorie threshold is respected, the meal will be included in the list of recommendations, otherwise, the meal will be null and total calories will be equal 0.

	inputs		outputs		annotations
C	isCalorie_Conscious	Course_Type	Meal Suggestion		Description
			Meal Suggestion	Total Calories	
	Calorie_conscious_Level (0..1)	*Text "First-Dish", "Second-Dish", "Side-Dish", "Dessert"*	*Text*	*Number*	
1	0,1	"First-Dish"	if list contains(Choose Ingredients, "Rice") and list contains(Choose Ingredients, "Shrimp") and list contains(Choose Ingredients, "Courgette") then "Riso gamberetti e zucchine" else null	if list contains(Choose Ingredients, "Rice") and list contains(Choose Ingredients, "Shrimp") and list contains(Choose Ingredients, "Courgette") then sum(for i in Ingredient List return if i.name in ["Rice", "Shrimp", "Courgette"] then i.calories else 0) else 0	
2	0,1	"Second-Dish"	if list contains(Choose Ingredients, "Beef Steak") then "Fiorentina" else null	if list contains(Choose Ingredients, "Beef Steak") then sum(for i in Ingredient List return if i.name in ["Beef Steak"] then i.calories else 0) else 0	
3	0,1	"Second-Dish"	if list contains(Choose Ingredients, "Courgette") and list contains(Choose Ingredients, "Egg") and list contains(Choose Ingredients, "Breadcrumbs") then "Polpette di zucchine" else null	if list contains(Choose Ingredients, "Courgette") and list contains(Choose Ingredients, "Egg") and list contains(Choose Ingredients, "Breadcrumbs") then sum(for i in Ingredient List return if i.name in ["Egg", "Breadcrumbs", "Courgette"] then i.calories else 0) else 0	
4	0,1	"Side-Dish"	if list contains(Choose Ingredients, "Lettuce") then "Insalata" else null	if list contains(Choose Ingredients, "Lettuce") then sum(for i in Ingredient List return if i.name in ["Lettuce"] then i.calories else 0) else 0	

Figure 2.4: Meal Suggestion Decision Table

Ingredient List
Any

```
[
  {"name": "Pasta", "calories": 180},
  {"name": "Butter", "calories": 220},
  {"name": "Sage", "calories": 0},
  {"name": "Rice", "calories": 160},
  {"name": "Shrimp", "calories": 45},
  {"name": "Courgette", "calories": 10},
  {"name": "Egg", "calories": 60},
  {"name": "Pecorino", "calories": 190},
  {"name": "Pork Jowl", "calories": 330},
  {"name": "Beef Steak", "calories": 120},
  {"name": "Breadcrumbs", "calories": 160},
  {"name": "Eggplant", "calories": 10},
  {"name": "Mozzarella", "calories": 130},
  {"name": "Parmigiano", "calories": 190},
  {"name": "Tomato puree", "calories": 10},
  {"name": "Lettuce", "calories": 10},
  {"name": "Potato", "calories": 40},
  {"name": "Peach", "calories": 10},
  {"name": "Apricot", "calories": 10},
  {"name": "Watermelon", "calories": 10}
]
```

Figure 2.5: Ingredient List Literal Expression

2.1.3 Input Configuration and Result Output

Finally, an example of meal suggestions from the model. For this first simulation, the guest parameters are set as follows: category omnivore, no intolerances, calorie-conscious level to 0, course type First Dish. Simulation is shown in Figure 2.6.

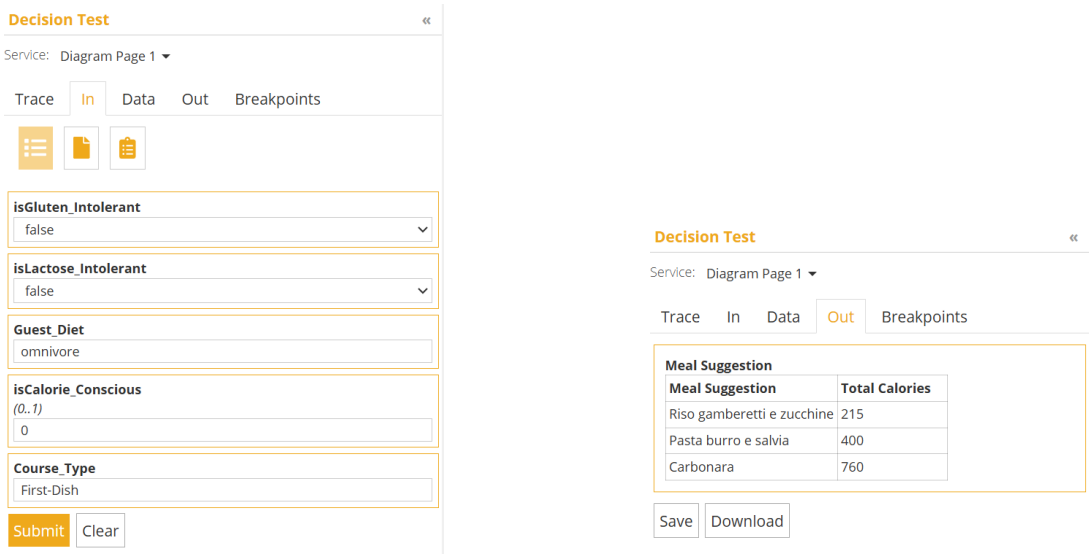

Figure 2.6: First simulation

If both food allergies are set to true, the results will change as shown in Figure 2.7.

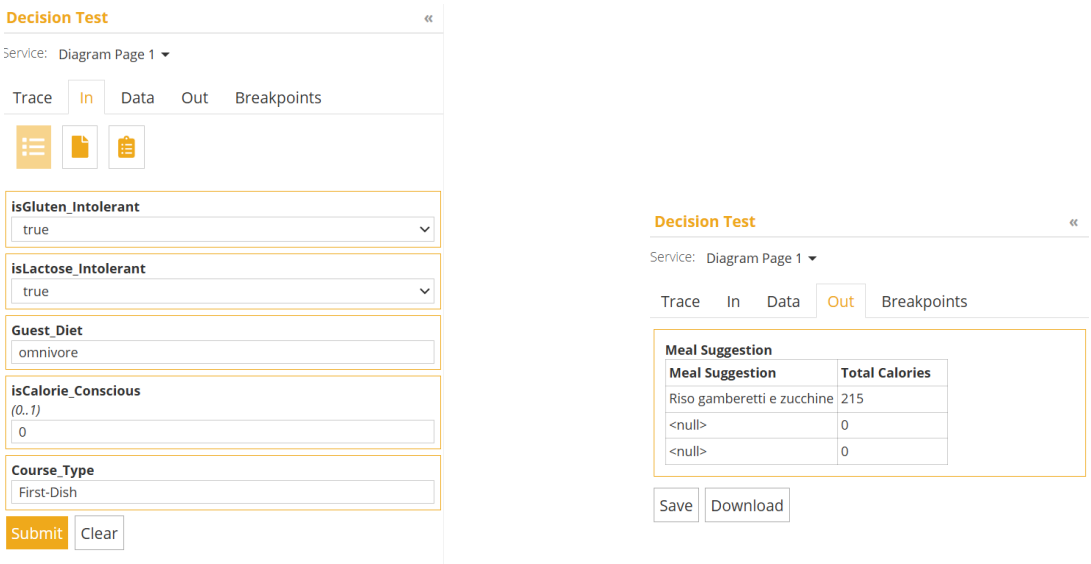

Figure 2.7: Second simulation

2.2 Prolog

Prolog is a logic programming language that has its origins in artificial intelligence, automated theorem proving, and computational linguistics. The main idea is to describe facts and rules about a problem domain so Prolog can figure out how to solve it. Prolog syntax provides:

- **Symbols: ,** (and) **;** (or) **:-** (if) **not** (not);
- **Variables:** a string of letters, digits, and underscores (_) beginning either with a capital letter or with an underscore;
- **Facts:** a predicate expression that makes a declarative statement about the problem domain;
- **Rules:** a rule is a predicate expression that uses logical implication (:-) to describe a relationship among facts;
- **Queries:** the Prolog interpreter answers queries on the facts and rules represented in its database. By asking a question, Prolog is asked whether it can prove that the question is true. If the answer is "yes", Prolog answers "yes" and displays all the links between the variables made to obtain the answer. If it cannot prove that the query is true, it answers "no".

In Prolog is also used the technique of backtracking, a core search mechanism that Prolog uses to find solutions to queries, matching it against facts and rules in the knowledge base. If one path doesn't work, Prolog automatically tries other possibilities. Additionally, there are several built-in predicates called native rules provided by Prolog, such as "length(?List, ?Length)" which allows obtaining the length of a list.

2.2.1 Prolog Implementation

The model defined in the DMN file has been implemented using Prolog on the site SWISH Prolog. As with Task 1.1, also in Prolog the restrictions are first applied at the ingredient level and then at the meal level. Consequently, if an ingredient is categorized as vegetarian and a meal contains that ingredient (unless the meal also includes a carnivorous ingredient), the meal itself is classified as vegetarian. The same logic applies for carnivorous ingredients. However, if a meal contains both a carnivorous and a vegetarian ingredient, it is classified as neither carnivorous nor vegetarian, and is therefore considered omnivorous. Meals that contain neither carnivorous nor vegetarian ingredients are still visible to both carnivorous and vegetarian users. The Prolog code is described below.

Listing 2.1 shows the definition of the list of ingredients. Each ingredient is declared with the ingredient predicate, which effectively lists all the components available for meal preparation. It facilitates the subsequent steps of the program, where these ingredients will be associated with their caloric values and categorization.

```
% Define the list of ingredients used in meals
ingredient(pasta).
ingredient(butter).
ingredient(sage).
ingredient(rice).
ingredient(shrimp).
ingredient(courgette).
ingredient(egg).
ingredient(pecorino).
ingredient(pork_jowl).
ingredient(beef_steak).
ingredient(breadcrumbs).
ingredient(eggplant).
ingredient(mozzarella).
```

```
ingredient(parmigiano).
ingredient(tomato_puree).
ingredient(lettuce).
ingredient(potato).
ingredient(peach).
ingredient(apricot).
ingredient(watermelon).
```

Listing 2.1: Facts for Ingredients

Listing 2.2 shows the caloric content for each ingredient using the kcal ingredient predicate. It maps each ingredient to its caloric value.

```
% Define the caloric content of each ingredient
kcal_ingredient(pasta, 180).
kcal_ingredient(butter, 220).
kcal_ingredient(sage, 0).
kcal_ingredient(rice, 160).
kcal_ingredient(shrimp, 45).
kcal_ingredient(courgette, 10).
kcal_ingredient(egg, 60).
kcal_ingredient(pecorino, 190).
kcal_ingredient(pork_jowl, 330).
kcal_ingredient(beef_steak, 120).
kcal_ingredient(breadcrumbs, 160).
kcal_ingredient(eggplant, 10).
kcal_ingredient(mozzarella, 130).
kcal_ingredient(parmigiano, 190).
kcal_ingredient(tomato_puree, 10).
kcal_ingredient(lettuce, 10).
kcal_ingredient(potato, 40).
kcal_ingredient(peach, 10).
kcal_ingredient(apricot, 10).
kcal_ingredient(watermelon, 10).
```

Listing 2.2: Ingredients and Kcal association

Listing 2.3 shows ingredients that belong to a specific diet.

```
% Define which ingredients are carnivorous
ingredient_carnivore(shrimp).
ingredient_carnivore(pork_jowl).
ingredient_carnivore(beef_steak).

% Define which ingredients are vegetarian
ingredient_vegetarian(courgette).
ingredient_vegetarian(eggplant).
ingredient_vegetarian(lettuce).
```

Listing 2.3: Carnivorous and Vegetarian Ingredients

Listing 2.4 shows all the ingredients that contain allergens (gluten or lactose).

```
% Define which ingredients cause lactose intolerance
ingredient_with_lactose_intolerance(pecorino).
ingredient_with_lactose_intolerance(mozzarella).
ingredient_with_lactose_intolerance(parmigiano).

% Define which ingredients contain gluten
ingredient_with_gluten_intolerance(pasta).
ingredient_with_gluten_intolerance(breadcrumbs).
```

Listing 2.4: Ingredients that contains an intolerance

Listing 2.5 shows Meals definition with the meal predicate, specifying the meal name, the course and a list of ingredients. The structured meal definitions provide a foundation for querying and analyzing meal options based on their composition.

```
% Define various meals with their ingredients
meal(pasta_burro_salvia, first_dish, [pasta, butter, sage]).
meal(risotto_gamberetti_zucchine, first_dish, [rice, shrimp, courgette]).
meal(carbonara, first_dish, [pasta, egg, pecorino, pork_jowl]).
meal(fiorentina, second_dish, [beef_steak]).
meal(polpette_zucchine, second_dish, [courgette, egg, breadcrumbs]).
meal(parmigiana, second_dish, [eggplant, mozzarella, parmigiano,
    tomato_puree]).
meal(insalata, side_dish, [lettuce]).
meal(patate_al_forno, side_dish, [potato]).
meal(macedonia, dessert, [peach, apricot, watermelon]).
```

Listing 2.5: Facts for Meals

Listing 2.6 shows the dietary classification of a meal: vegetarian, carnivorous or omnivore. These predicates enable the categorization of meals according to dietary preferences, assisting users in selecting suitable meal options based on their dietary requirements.

```
% Determine if a meal is vegetarian
vegetarian_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    forall(member(Ingredient, Ingredients),
           (ingredient_vegetarian(Ingredient); \+ ingredient_carnivore(
    Ingredient))).

% Determine if a meal is carnivorous
carnivorous_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    % Ensure all ingredients are either carnivorous or not vegetarian
    forall(member(Ingredient, Ingredients),
           (ingredient_carnivore(Ingredient); \+ ingredient_vegetarian(
    Ingredient))).

% Define meals that contain both carnivorous and vegetarian ingredients
omnivore_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    forall(member(Ingredient, Ingredients), ingredient(Ingredient)).
```

Listing 2.6: Rules for Meals diet

Listing 2.7 shows the identification of meals that contain ingredients causing gluten or lactose intolerance. Findall is used to collect relevant meal-course pairs and then filter out duplicates. These checks are critical for ensuring that meals meet specific dietary restrictions.

```
% Find meals with gluten intolerance
meal_with_gluten_intolerance(Meal, Course) :-
    findall(Meal-Course,
            (meal(Meal, Course, Ingredients),
             member(Ingredient, Ingredients),
             ingredient_with_gluten_intolerance(Ingredient)),
            MealsWithGlutenIntolerance),
    list_to_set(MealsWithGlutenIntolerance, UniqueMeals),
    member(Meal-Course, UniqueMeals).

% Find meals with lactose intolerance
meal_with_lactose_intolerance(Meal, Course) :-
```

```
    findall(Meal-Course,
            (meal(Meal, Course, Ingredients),
             member(Ingredient, Ingredients),
             ingredient_with_lactose_intolerance(Ingredient)),
            MealsWithLactoseIntolerance),
    list_to_set(MealsWithLactoseIntolerance, UniqueMeals),
    member(Meal-Course, UniqueMeals).
```

Listing 2.7: Rules for Meals intolerances

Listing 2.8 shows the definition of the calorie conscious levels and computes the total caloric content of a meal by summing the calories of its ingredients allowing for meal categorization based on their total caloric content.

```
% Determine calorie-conscious levels based on total calories
calorie_conscious_levels(Meal, Course, Levels) :-
    meal_calories(Meal, Course, TotalCalories),
    % Determine the highest applicable level based on total calories
    ( TotalCalories > 250 -> HighestLevel = 0;
    TotalCalories =< 250 -> HighestLevel = 1
    ),
    % Generate all levels up to the highest applicable level
    findall(Level, (between(0, HighestLevel, Level)), Levels).
```

Listing 2.8: Calorie Level definition and Calories Calculation for meals

Listing 2.9 shows the guest preferences predicate that filters meals based on category (carnivorous, vegetarian, omnivore), calorie level, and allergies. It combines multiple conditions to refine meal recommendations.

```
% Filter meals based on guest preferences, including category, calorie level
    , and allergies
guest_preferences(Category, CalorieLevel, Allergies, Meal, Course) :-
    % Filtered meals by Category (carnivorous, vegetarian, omnivore)
    ( Category = carnivorous -> carnivorous_meal(Meal, Course)
    ; Category = vegetarian -> vegetarian_meal(Meal, Course)
    ; Category = omnivore -> omnivore_meal(Meal, Course)
    ),
    % Filtered meals by calorie_conscious_level
    calorie_conscious_levels(Meal, Course, Levels),
    member(CalorieLevel, Levels),
    % Filtered meals by allergies
    ( Allergies = none -> true
    ; ( Allergies = lactose -> not(meal_with_lactose_intolerance(Meal,
    Course))
      ; Allergies = gluten -> not(meal_with_gluten_intolerance(Meal, Course))
    )
    )
    ).
```

Listing 2.9: Rules for meal suggestion

2.2.2 Prolog Testing

Below are some tests to see if the query of guest preferences works well. In the first scenario (Figure 2.8), the guest is omnivore, doesn't care about calorie awareness and has no allergies, the system returns all the dishes and the relative courses as it should. In the second case (Figure 2.9), the guest is carnivorous, has calorie awareness, and has no allergies, the following meals are correctly returned: 'fiorentina', 'patate al forno', 'macedonia'.

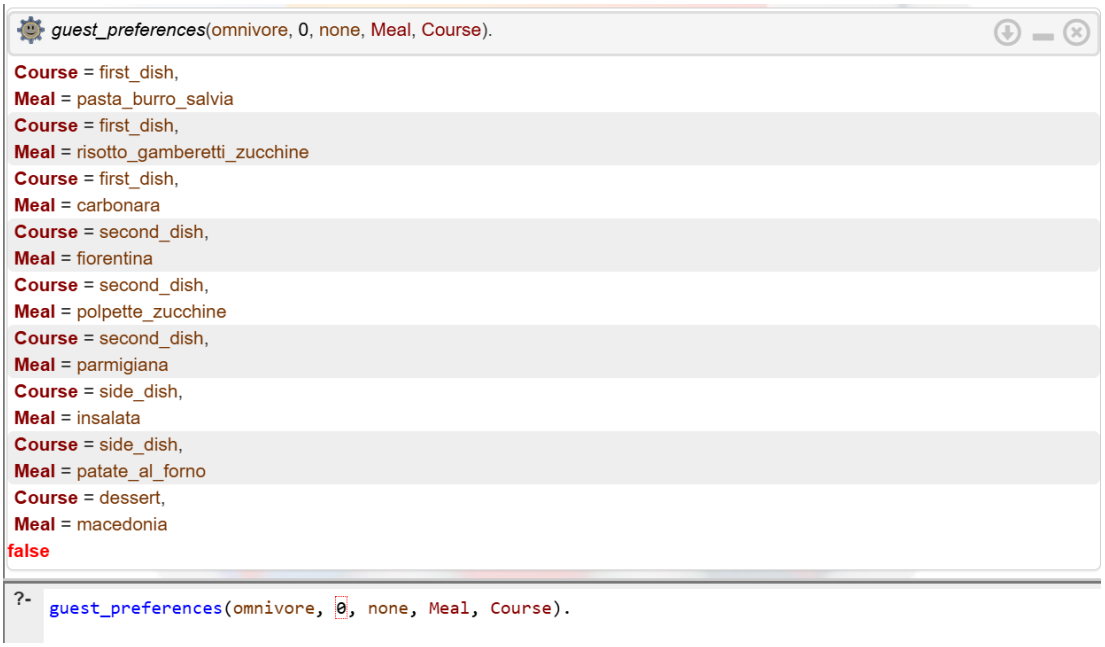

Figure 2.8: Prolog Testing Query 1

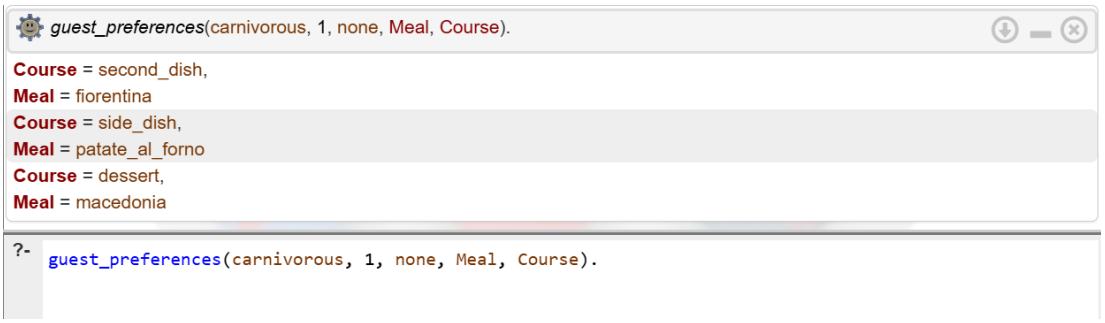

Figure 2.9: Prolog Testing Query 2

2.3 Ontology Engineering

In Computer Science, ontology engineering is a discipline focused on methodologies for constructing ontologies. This process involves formally representing and defining categories, properties, and relationships among concepts, data, and entities. Ontologies can be structured using various data models such as RDF, OWL, Neo4J, and others. For this project's ontology construction, the choice was RDF.

2.3.1 RDF

The Resource Description Framework (RDF) is a standardized approach utilized for describing and exchanging graph data in a general manner. RDF enables the description of resources that are utilized in constructing knowledge graphs. A Knowledge Graph depicts a network of real-world entities such as objects, events, situations, or concepts, highlighting the relationships that exist between them.

2.3.2 Protégé

Protégé is a freely available, open-source ontology editor and knowledge management system. It enables the definition of Semantic Web Rule Language (SWRL) to establish inference rules that generate new data from existing ontologies. Additionally, Protégé supports the Sparql Protocol And Rdf Query Language (SPARQL), a semantic query language designed for databases capable of retrieving and manipulating data stored in RDF format.

2.3.3 Menu Ontology

Starting from Prolog, we developed our ontology creating classes, properties, individuals and then we created the rules and constraints of our ontology.

Classes

We established the classes of our ontology with a hierarchy to facilitate the future expansions of the project (Figure 2.10, Figure 2.11).

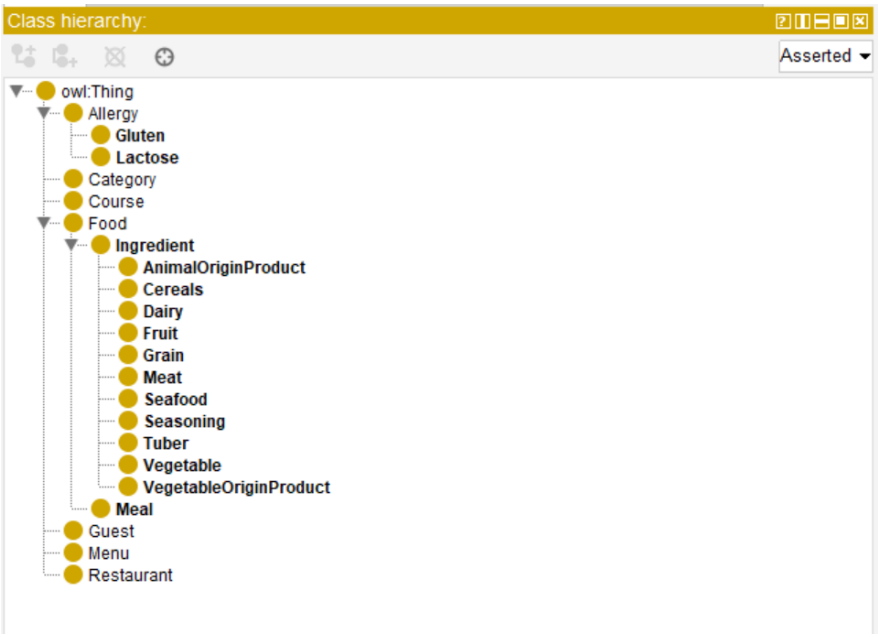

Figure 2.10: Ontology tree class hierarchy

Object Properties

Object properties are used to represent the relationships between classes (Figure 2.12). The object properties defined for the classes are the following:

- **restaurant hasMenu:** to associate a menu with a restaurant;
- **restaurant hasGuest:** to associate guest with a restaurant;
- **menu containsMeal:** to associate a meal with a menu;
- **guest hasAllergies:** it represents a guest that has an allergy;

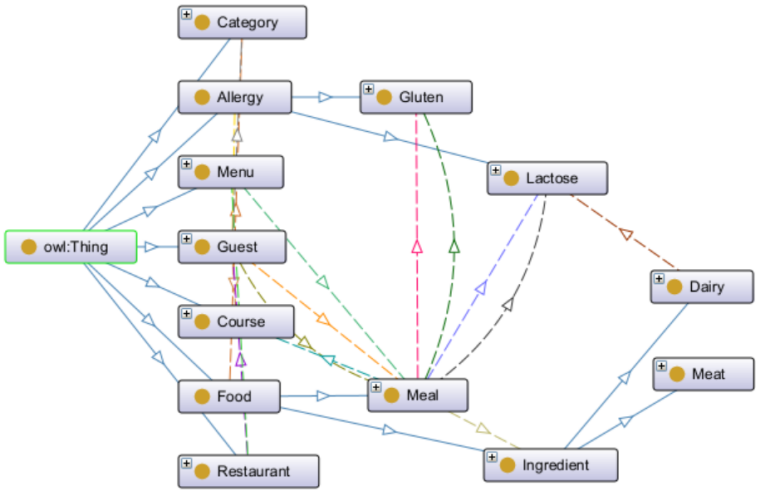

Figure 2.11: Ontology graph class heirarchy

- **guest hasCategory:** to associate a category (Carnivorous, Vegetarian, Omnivore) with a guest.
- **guest hasPreferenceCourse:** to associate a specific course with a guest;
- **guest isAtRiskForFood:** this is a SWRL rule inferred by the reasoner, which is useful for identifying potential meal-related risks due to allergies;
- **food hasCategory:** to associate a category with the food (ingredients or meals);
- **meal containsGlutenIntolerance:** this is an SWRL rule inferred by the reasoner, which identifies if the meal contains ingredients with gluten (Grain);
- **meal containsLactoseIntolerance:** this is an SWRL rule inferred by the reasoner, which identifies if the meal contains ingredients with lactose (Diary);
- **meal notContainsGlutenIntolerance:** added to check if a meal is gluten-free;
- **meal notContainsLactoseIntolerance:** added to check if a meal is lactosefree;
- **meal hasCourse:** to associate a course with the meal;
- **meal hasIngredient:** to associate an ingredient with a meal;
- **grainProduct containsGlutenIntolerance:** to associate the gluten intolerance to class Grain;
- **dairy containsLactoseIntolerance:** to associate the lactose intolerance to class Dairy

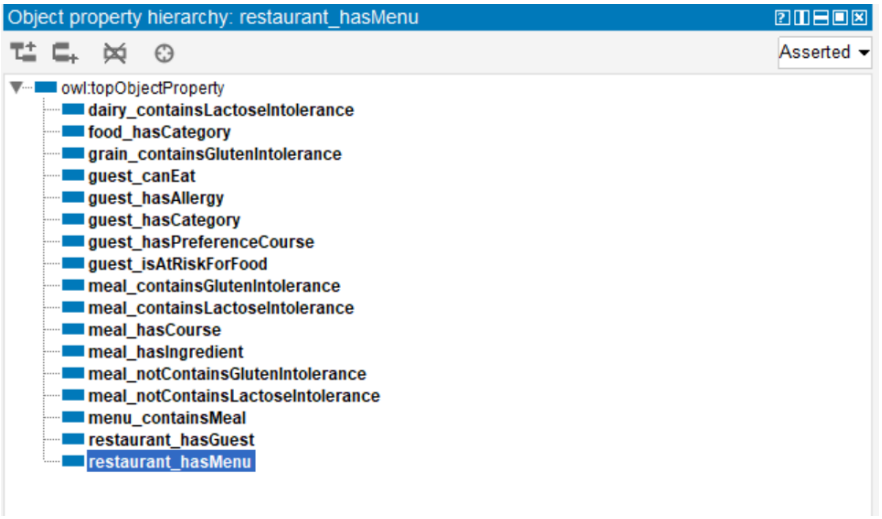

Figure 2.12: Ontology Object Property

Data Properties

Data Properties represent the attributes of the classes (Figure 2.13). The following are all the Data Properties:

- **thing hasName:** this is an attribute that all subclasses of the class Thing possess. It is a string used to identify the name of the object;
- food hasKcal: this is an attribute that all subclasses of the class Food (Ingredient and Meal) posses. It is an integer used to identify the kcal of the food;
- **guest hasLevelOfCalorieConscious:** this is an attribute for the class Guest that represents the level of calorie-consciousness of a guest. It is an integer ranging between 0 and 1;
- **meal hasLevelOfCalorieConscious:** this is an attribute for the class Meal that represents the level of calorie-consciousness of a meal. It is an integer ranging between 0 and 1;

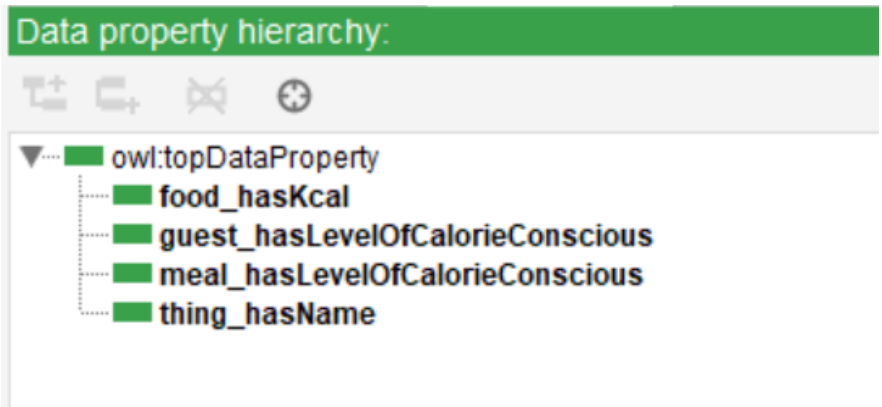

Figure 2.13: Ontology Data Property

Individuals

Given the large number of individuals (Figure 2.14) , only a few examples will be showed:

- **Guest:** Figure 2.15;
- **Menu:** Figure 2.16,
- **Carbonara Meal:** Figure 2.17;
- **Restaurant:** Figure 2.18;

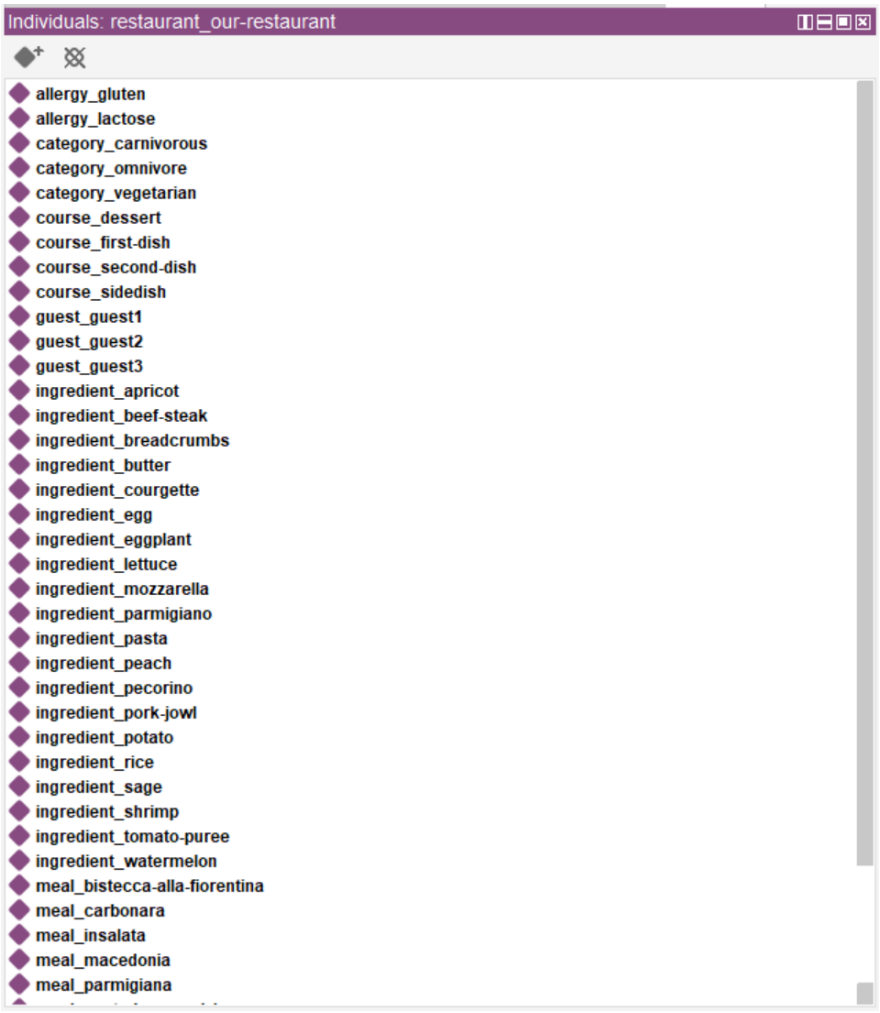

Figure 2.14: Ontology Individuals

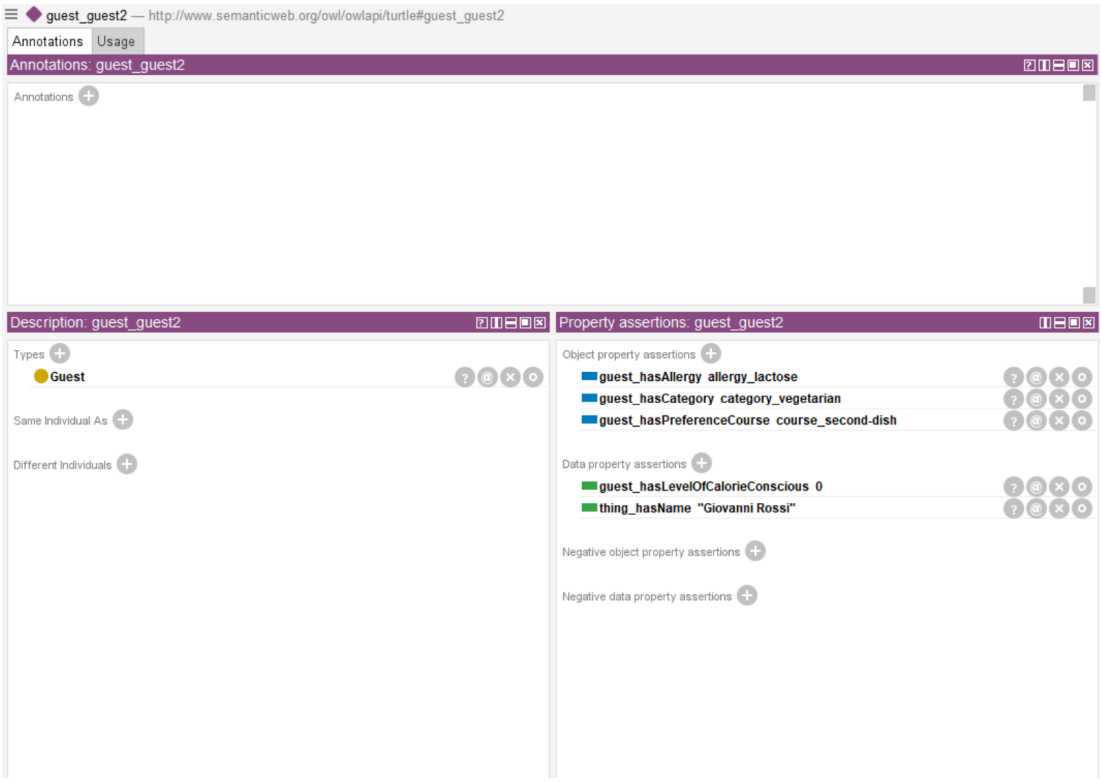

Figure 2.15: Guest2 Individual

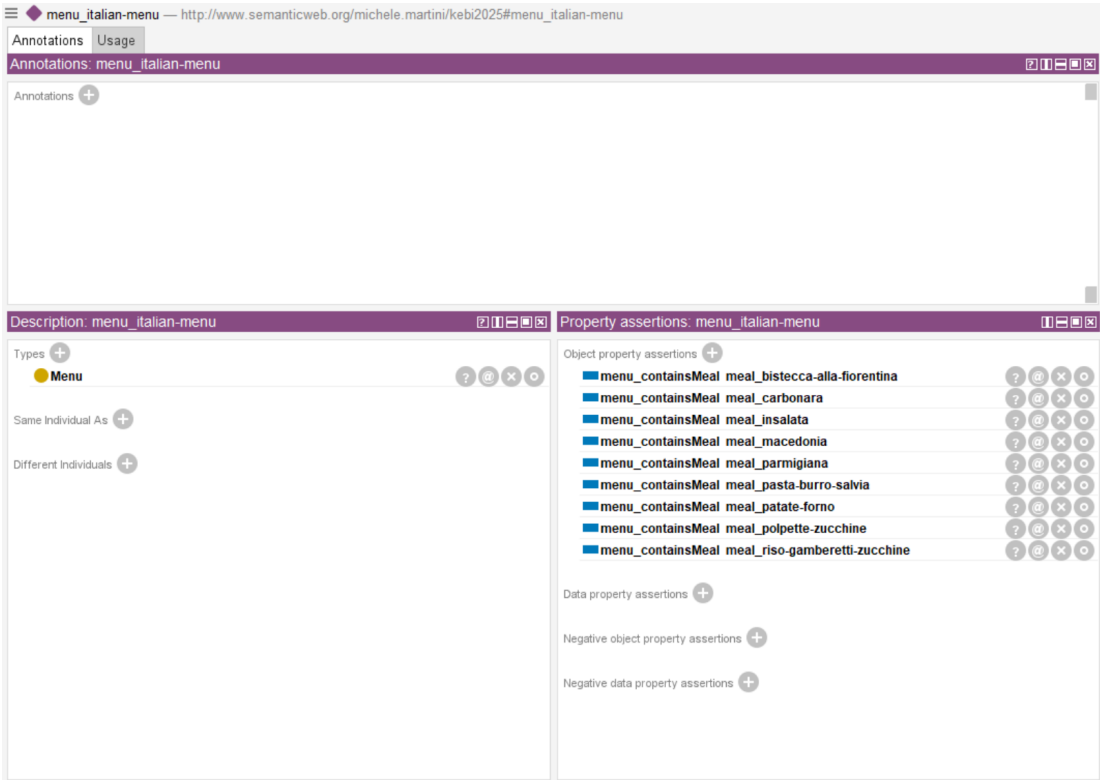

Figure 2.16: Menu Individual

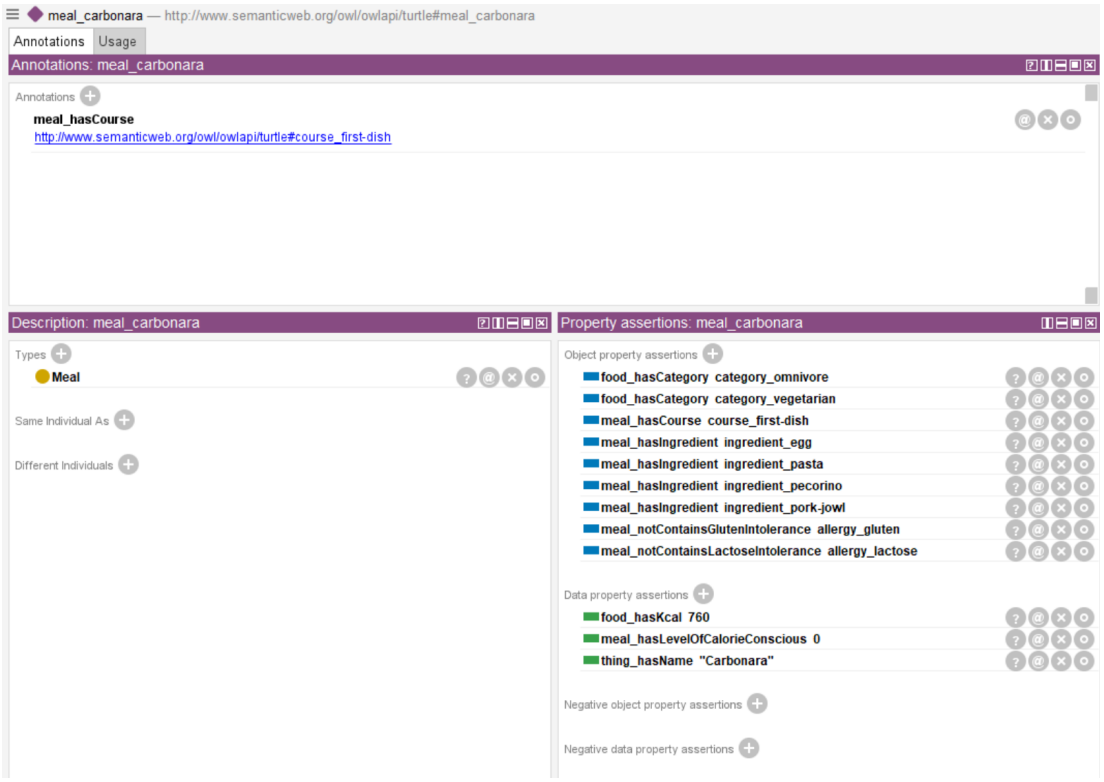

Figure 2.17: Carbonara Meal Individual

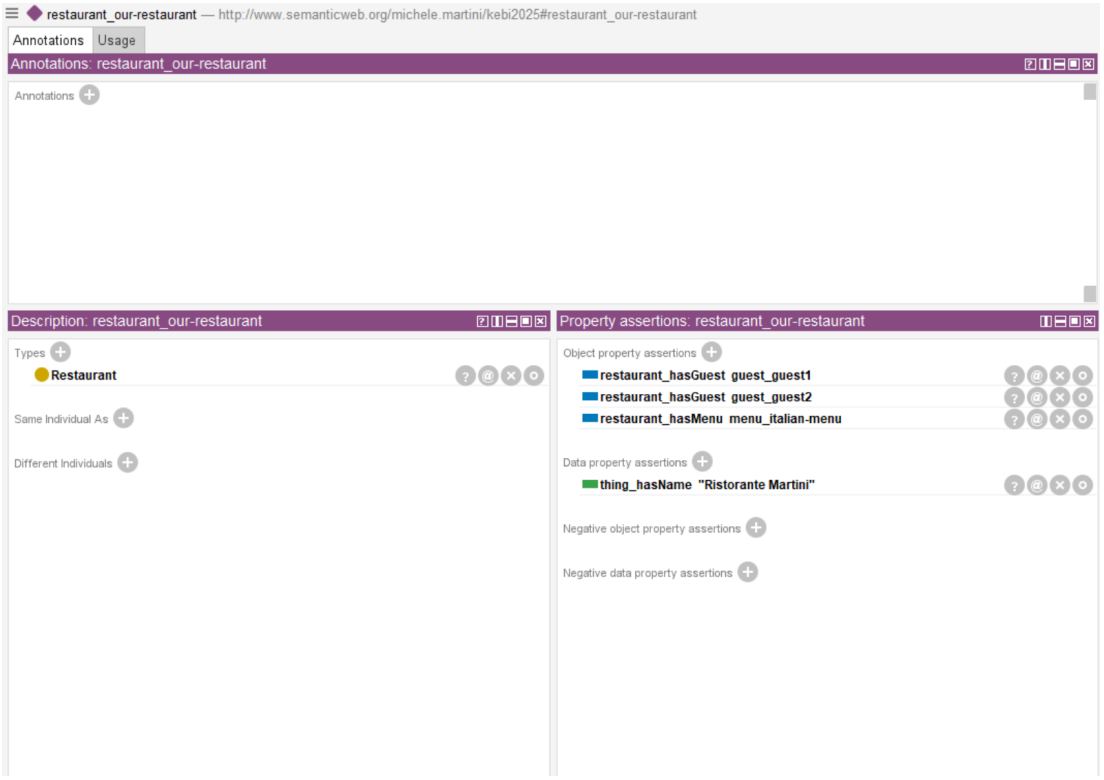

Figure 2.18: Restaurant Individual

2.3.4 SWRL Rules

With all the classes, object properties, and data properties created, the next phase was to write the rules and constraints of our ontology.

- **GuestIsAtRiskForFoodGluten** this rule infers meals that pose a risk to the user based on whether the guest has gluten intolerance or not, and to display them accordingly.

```
kebi2025:Guest(?guest) ^
kebi2025:guest_hasAllergy(?guest, ?allergy) ^
kebi2025:Meal(?meal) ^
kebi2025:meal_containsGlutenIntolerance(?meal, ?allergy)
-> kebi2025:guest_isAtRiskForFood(?guest, ?meal)
```

- **GuestIsAtRiskForFoodLactose** this rule infers meals that pose a risk to the user based on whether the guest has lactose intolerance or not, and to display them accordingly.

```
kebi2025:Guest(?guest) ^
kebi2025:guest_hasAllergy(?guest, ?allergy) ^
ebi2025:Meal(?meal) ^
kebi2025:meal_containsLactoseIntolerance(?meal, ?allergy)
-> kebi2025:guest_isAtRiskForFood(?guest, ?meal)
```

- **MealContainsGlutenIntolerance** this rule infers meals that contain ingredients with gluten intolerance by adding gluten intolerance to the meal.

```
kebi2025:Meal(?meal) ^
kebi2025:meal_hasIngredient(?meal, ?ingredient) ^
kebi2025:Ingredient(?ingredient) ^
kebi2025:grain_containsGlutenIntolerance(?ingredient, ?gluten)
-> kebi2025:meal_containsGlutenIntolerance(?meal, ?gluten)
```

- **MealContainsLactoseIntolerance** this rule infers meals that contain ingredients with lactose intolerance by adding lactose intolerance to the meal.

```
kebi2025:Meal(?meal) ^
kebi2025:meal_hasIngredient(?meal, ?ingredient) ^
kebi2025:Ingredient(?ingredient) ^
kebi2025:dairy_containsLactoseIntolerance(?ingredient, ?dairy)
-> kebi2025:meal_containsLactoseIntolerance(?meal, ?dairy)
```

- **Rule for inferring meals to eat - gluten intolerance** this rule infers the types of meals a guest can eat based on their preferences, like calories and favourite course, and whether they have a gluten intolerance.

```
kebi2025:Guest(?guest) ^
kebi2025:guest_hasCategory(?guest, ?category) ^
kebi2025:guest_hasAllergy(?guest, ?allergy) ^
kebi2025:guest_hasLevelOfCalorieConscious(?guest, ?level) ^
kebi2025:guest_hasPreferenceCourse(?guest, ?course) ^
kebi2025:Category(?category) ^
kebi2025:Allergy(?allergy) ^
kebi2025:Meal(?meal) ^
kebi2025:food_hasCategory(?meal, ?category) ^
kebi2025:meal_notContainsGlutenIntolerance(?meal, ?allergy) ^
kebi2025:meal_hasLevelOfCalorieConscious(?meal, ?level) ^
kebi2025:meal_hasCourse(?meal, ?course)
-> kebi2025:guest_canEat(?guest, ?meal)
```

- **Rule for inferring meals to eat - lactose intolerance** this rule infers the types of meals a guest can eat based on their preferences, like calories and favourite course, and whether they have a lactose intolerance.

```
kebi2025:Guest(?guest) ^
kebi2025:guest_hasCategory(?guest, ?category) ^
kebi2025:guest_hasAllergy(?guest, ?allergy) ^
kebi2025:guest_hasLevelOfCalorieConscious(?guest, ?level) ^
kebi2025:guest_hasPreferenceCourse(?guest, ?course) ^
kebi2025:Category(?category) ^ kebi2025:Allergy(?allergy) ^
kebi2025:Meal(?meal) ^ kebi2025:food_hasCategory(?meal, ?category) ^
kebi2025:meal_notContainsLactoseIntolerance(?meal, ?allergy) ^
kebi2025:meal_hasLevelOfCalorieConscious(?meal, ?level) ^
kebi2025:meal_hasCourse(?meal, ?course)
-> kebi2025:guest_canEat(?guest, ?meal)
```

2.3.5 SWRL Testing

The testing was performed running the reasoner HermiT 1.4.3.456, to process the SWRL rules. The reasoner applies logical reasoning to check the consistency of ontology, infer new facts based on rules and the data you already have, and classify individuals and classes by adding new property assertions or class memberships. Below are a couple of examples to demonstrate the SWRL rules:

- **Test 1:** Figure 2.19 shows the high-risk dishes for the user are immediately inferred, and the food that guest1 can safely consume ("riso gamberetti e zucchine") is correctly identified.

- **Test 2:** Figure 2.20 In this case, the inference is based on whether the meal contains an allergenic ingredient or not.

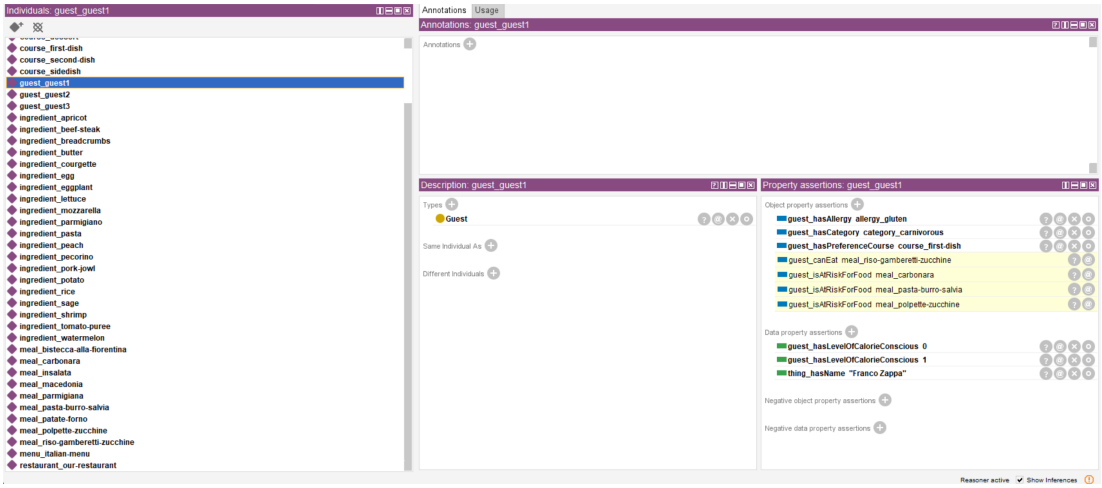

Figure 2.19: SWRL Test 1 - Guest 1: Meals Suggestion

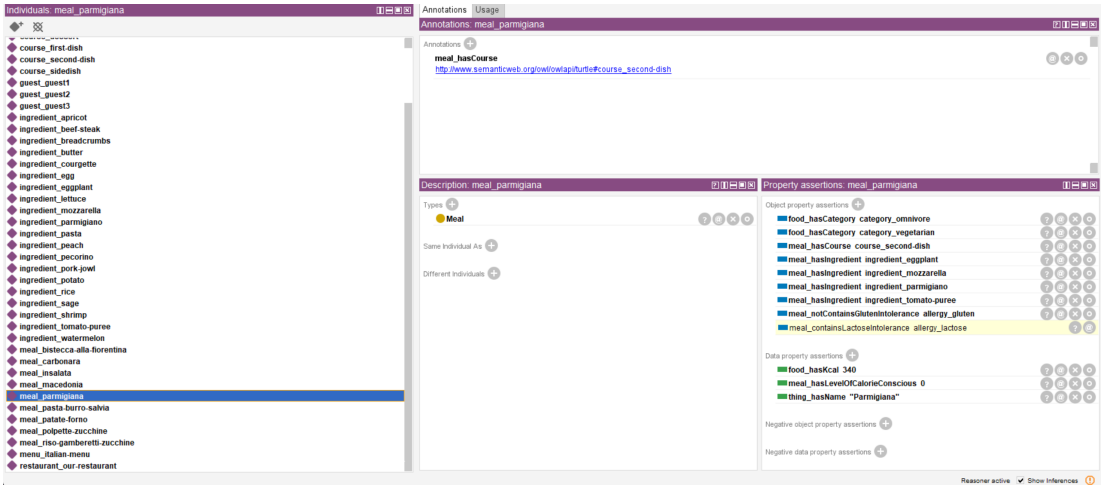

Figure 2.20: SWRL Test 2 - Parmigiana: Meals Intolerance

2.3.6 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a powerful query language and protocol used for accessing and manipulating data stored in RDF format. It allows users to write queries to extract information from RDF graphs, perform complex searches, and integrate data from diverse sources. SPARQL queries enable precise data retrieval, making it an essential tool for working with semantic web data and linked data applications. In the following subsection, some of the queries will be described. The queries have been tested using Protégé.

The following prefix has been used for all queries:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/michele.martini/kebi2025#>
```

Listing 2.10: Prefix for Queries

- **Query for meal calorie calculation:** This query is designed to retrieve the calories of every ingredient on a meal and calculate the calorie sum of the entire meal. Result in Figure 2.21.

```
SELECT ?meal ?calories
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:food_hasKcal ?calories .
}
```

Listing 2.11: Query for meal calorie calculation

- **Query for guest allergies:** This query is designed to retrieve all allergies of each guest. Result in Figure 2.22.

```
SELECT ?guest ?allergy
WHERE {
  ?guest a kebi:Guest ;
         kebi:guest_hasAllergy ?allergy .
```

```
5 }
```

Listing 2.12: Query for guest allergies

- **Query for carbonara ingredients retrieval** This query is used to see all the ingredients in a specific meal, in this case "carbonara". Result in Figure 2.23.

```
1 SELECT ?ingredient
2 WHERE {
3   kebi:meal_carbonara kebi:meal_hasIngredient ?ingredient .
4 }
```

Listing 2.13: Query for carbonara ingredients retrieval

SPARQL query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/michele.martini/kebi2025#>

SELECT ?meal ?calories
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:food_hasKcal ?calories .
}
```

meal	calories
meal_insalata	"10"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_polpette-zucchine	"230"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_bistecca-alla-fiorentina	"120"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_pasta-burro-salvia	"400"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_patate-forno	"40"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_riso-gamberetti-zucchine	"215"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_carbonara	"760"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_macedonia	"30"^^<http://www.w3.org/2001/XMLSchema#integer>
meal_parmigiana	"340"^^<http://www.w3.org/2001/XMLSchema#integer>

Figure 2.21: Result of Query for meal calorie calculation

SPARQL query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/michele.martini/kebi2025#>

SELECT ?guest ?allergy
WHERE {
  ?guest a kebi:Guest ;
      kebi:guest_hasAllergy ?allergy .
}
```

guest	allergy
guest_guest1	allergy_gluten
guest_guest3	allergy_gluten
guest_guest2	allergy_lactose

Figure 2.22: Result of Query for guest allergies

SPARQL query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/michele.martini/kebi2025#>

SELECT ?ingredient
WHERE {
  kebi:meal_carbonara kebi:meal_hasIngredient ?ingredient .
}
```

ingredient
ingredient_egg
ingredient_pork-jowl
ingredient_pecorino
ingredient_pasta

Figure 2.23: Result of Query for carbonara ingredients retrieval

Remaining SPARQL Queries

The remaining queries will be listed here without results due to their large number.

- **Query for all Carnivorous meals**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:food_hasCategory kebi:category_carnivorous .
}
```

- **Query for all Vegetarian meals**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:food_hasCategory kebi:category_vegetarian .
}
```

- **Query for view the kcal of a specific meal (carbonara)**

```
SELECT ?kcal
WHERE {
  kebi:meal_carbonara kebi:food_hasKcal ?kcal.
}
```

- **Query for view the kcal of all ingredients**

```
SELECT ?ingredient ?kcal
WHERE {
  ?ingredient a kebi:Ingredient .
  ?ingredient kebi:food_hasKcal ?kcal .
}
```

- **Query for view the level 1 of Calorie Conscious of all meals**

```
SELECT DISTINCT ?meal ?calorie
WHERE {
  ?meal rdf:type ?type .
  ?meal kebi:meal_hasLevelOfCalorieConscious ?calorie .
  FILTER(?calorie = 1)
}
```

- **Query for view the kcal of a specific ingredient (pasta)**

```
SELECT ?kcal
WHERE {
  kebi:ingredient_pasta kebi:food_hasKcal ?kcal .
}
```

- **Query for view the ingredients which has lactose intolerance**

```
SELECT ?ingredient
WHERE {
  ?ingredient rdf:type kebi:Ingredient .
  ?ingredient kebi:dairy_containsLactoseIntolerance kebi:
    allergy_lactose .
}
```

- **Query for view the ingredients which has gluten intolerance**

```
SELECT ?ingredient
WHERE {
  ?ingredient rdf:type kebi:Ingredient .
  ?ingredient kebi:grain_containsGlutenIntolerance kebi:allergy_gluten
    .
}
```

- **Query for view the meals which has gluten intolerance**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_containsGlutenIntolerance kebi:allergy_gluten .
}
```

- **Query for view the meals which has lactose intolerance**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_containsLactoseIntolerance kebi:allergy_lactose .
}
```

- **Query for view the meals that are classified as First Dish**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_hasCourse kebi:course_firs-dish .
}
```

- **Query for view the meals that are classified as Second Dish**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_hasCourse kebi:course_secod-dish .
}
```

- **Query for view the ingredient that are classified as Seafood**

```
SELECT ?ingredient
WHERE {
  ?ingredient rdf:type kebi:Seafood.
}
```

- **Query for view the meals that has more than 300 kcal**

```
SELECT ?meal ?kcal
WHERE {
  ?meal kebi:food_hasKcal ?kcal.
  FILTER(xsd:decimal(?kcal) > 300).
}
```

- **Advanced query for see the meals in base of the preference of the guest (guest1)**

```
SELECT DISTINCT ?meal ?mealName
WHERE{
  kebi:guest_guest1 a kebi:Guest ;
  kebi:guest_hasAllergy ?allergy ;
  kebi:guest_hasCategory ?category;
  kebi:guest_hasPreferenceCourse ?course ;
  kebi:guest_hasLevelOfCalorieConscious ?calorieLevel .

  ?meal a kebi:Meal;
  kebi:food_hasCategory ?category ;
  kebi:meal_hasLevelOfCalorieConscious ?calorieLevel ;
  kebi:meal_hasCourse ?course ;
  kebi:thing_hasName ?mealName .

  MINUS {?meal kebi:meal_containsGlutenIntolerance ?allergy }
  MINUS {?meal kebi:meal_containsLactoseIntolerance ?allergy }

}
```

2.3.7 SHACL

SHACL (Shapes Constraint Language) is a language used for validating RDF data against a set of conditions, known as shapes. These shapes define the expected structure and content of RDF graphs, ensuring data consistency and integrity. These are the shapes created for the project:

- **RestaurantShape** Ensures that each Restaurant instance has at least one Menu. A maxCount of 1 has not been set, to allow the restaurant manager more flexibility and add additional types of menus in the future. Additionally, it optionally allows associations with guests.

```
kebi2025:RestaurantShape a sh:NodeShape ;
    sh:targetClass kebi2025:Restaurant ;
    sh:property [
        sh:path kebi2025:restaurant_hasMenu ;
        sh:node kebi2025:MenuShape ;
        sh:minCount 1 ;
        sh:message "It is impossible that a Restaurant does not have a
    menu!!" ;
    ] ;
    sh:property [
        sh:path kebi2025:restaurant_hasGuest ;
        sh:node kebi2025:GuestShape ;
        sh:minCount 0;
    ] ;
.
```

- **MenuShape** Defines constraints for instances of Menu, ensuring that it contains at least one Meal.

```
kebi2025:MenuShape a sh:NodeShape ;
    sh:targetClass kebi2025:Menu ;
    sh:property [
        sh:path kebi2025:menu_containsMeal ;
        sh:node kebi2025:MealShape ;
        sh:minCount 1 ;
    ]
.
```

- **GuestShape** Validates instances of Guest, requiring mandatory information such as dietary category and calorie-consciousness level, while specifying optional details like allergies. The calorie-consciousness level ranges from 1 to 2. Can be linked to meals with intolerance risks.

```
kebi2025:GuestShape a sh:NodeShape ;
    sh:targetClass kebi2025:Guest ;
    sh:property [
        sh:path kebi2025:guest_hasAllergy ;
        sh:node kebi2025:AllergyShape ;
        sh:minCount 0 ;
    ] ;
    sh:property [
        sh:path kebi2025:guest_hasCategory ;
        sh:node kebi2025:CategoryShape ;
        sh:minCount 1;
    ] ;

    sh:property [
        sh:path kebi2025:guest_isAtRiskForFood ;
        sh:node kebi2025:MealShape ;
    ] ;
    sh:property [
        sh:path kebi2025:guest_hasLevelOfCalorieConscious ;
        sh:datatype xsd:integer ;
        sh:minCount 1 ;
        sh:maxCount 2 ;
    ] ;
.
```

- **MealShape** Specifies constraints for instances of Meal, including optional intolerance to lactose and gluten, categories, courses, and ingredients. A Meal can contain lactose or gluten. No maxCount has been set to keep future expansion simple. Lastly, to make a meal, there must be at least one ingredient.

```
kebi2025:MealShape a sh:NodeShape ;
    sh:targetClass kebi2025:Meal ;
    sh:property [
        sh:path kebi2025:meal_containsLactoseIntolerance ;
        sh:node kebi2025:LactoseShape ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi2025:meal_containsGlutenIntolerance ;
        sh:node kebi2025:GlutenShape ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi2025:meal_hasCategory ;
        sh:node kebi2025:CategoryShape ;
        sh:minCount 0 ;
    ] ;
    sh:property [
        sh:path kebi2025:meal_hasCourse ;
        sh:node kebi2025:CourseShape ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
```

```
    sh:property [
        sh:path kebi2025:meal_hasIngredient ;
        sh:node kebi2025:IngredientShape ;
        sh:minCount 1 ;
    ] ;
.
```

The validation was made by checking the shapes using the SHACL editor inside Protégé; the editor did not find any violations, therefore the validation is positive (Figure 2.24).

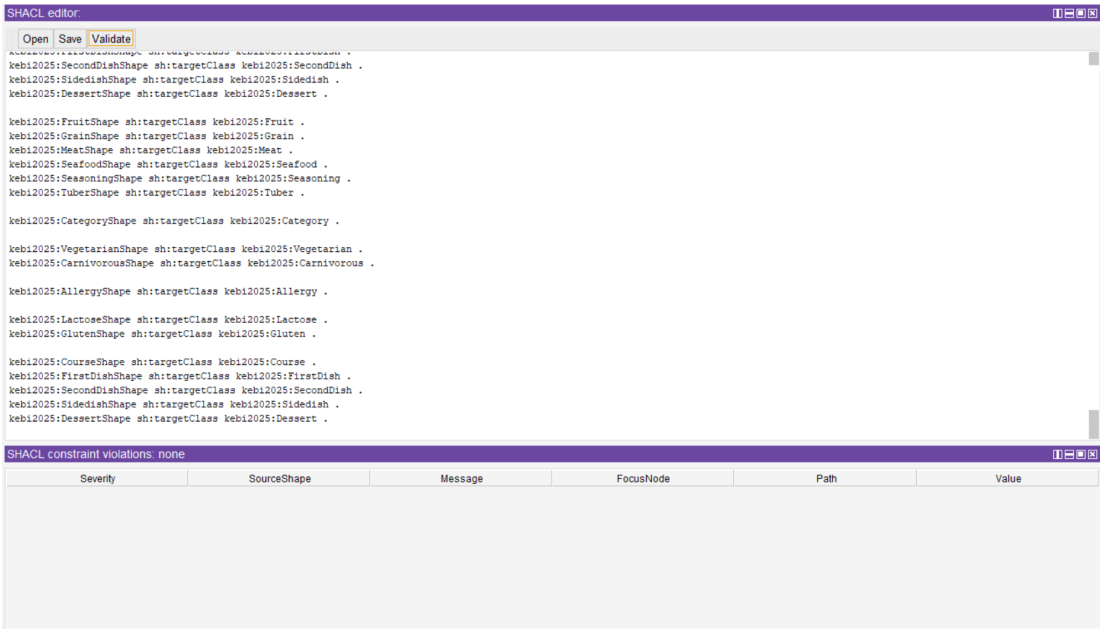

Figure 2.24: SHACL Validation

3. Agile and Ontology-based Meta-Modelling

In this chapter will be explained the implementation of agile modeling principles and ontology-based metamodelling in our project. Section 3.1 introduces AOAME, a tool that integrates domain ontologies into BPMN 2.0 diagrams. Section 3.2 focuses on BPMN 2.0, describing its key elements and how we used it to model the core process of our system.

3.1 AOAME

AOAME is a prototypical tool designed to implement an agile, ontology-based approach to meta-modeling. This approach aims to design and maintain schemas for Enterprise Knowledge Graphs (EKGs), which structure the knowledge of a specific application domain, enabling analysis, reasoning, and integration of information extracted from various data sources. The primary challenge in developing and maintaining an EKG schema is the requirement for expertise in both ontology engineering and the application domain. AOAME proposes an approach that extends traditional meta-modeling with agile principles, allowing for domain-specific adaptations of modeling languages and real-time testing. This approach facilitates the involvement of domain experts in the engineering cycle. AOAME is developed following the Design Science Research methodology and serves as a prototype to evaluate the utility of this approach. It provides capabilities to extend, modify, and remove modeling constructs, and updating the tool palette accordingly. These actions are supported by meta-modeling operators that generate SPARQL statements and update the triplestore, ensuring consistency between human-interpretable and machine-interpretable knowledge. The software architecture of AOAME leverages Java libraries to communicate with **Apache Jena Fuseki** that retrieve ontologies from the triplestore and display them in a GUI, as well as implementing the meta-modeling operators. Apache Jena Fuseki is a robust and scalable SPARQL server designed to handle RDF data efficiently. It provides a RESTful interface to perform various operations on RDF datasets, including: executing SPARQL queries; performing SPARQL updates; load, store, and manage RDF datasets. AOAME has been validated through the implementation of real-world scenarios and significant case studies, demonstrating its operability and generality across various application domains. AOAME facilitates close cooperation between domain experts and language engineers during the engineering of domain-specific modeling languages (DSMLs), thus enhancing the adaptability and effectiveness of the meta-modeling process.

3.2 BPMN 2.0

BPMN stands for Business Process Model and Notation. It is a graphical representation used to specify business processes in a workflow. BPMN provides a standardized way to visualize the sequence of business activities and the flow of information between them. It is designed to be understandable by all business figures, including stakeholders, business analysts and technical developers. The key elements of BPMN 2.0 are:

- **Events:** Represent occurrences that affect the flow of the process (e.g., start/end events, intermediate events).
- **Activities:** Tasks or work that needs to be performed (e.g., user tasks, service tasks, manual tasks).
- **Gateways:** Decision points that control the divergence and convergence of the process flow (e.g., exclusive gateways, parallel gateways, inclusive gateways).
- **Pools and Lanes:** Represent major participants in the process, with pools representing different organizations or entities, and lanes representing subdivisions within those entities.
- **Artifacts:** Supplementary elements such as data objects, groups, and annotations that provide additional information about the process.

In this project, BPMN is used to assist the restaurant manager in understanding the customer ordering process. Specifically, the BPMN diagram will consider both allergies and dietary preferences. Customers will be required to declare any allergies and indicate if they are vegetarian, carnivorous or omnivore. Subsequently, they can decide on the types of meals they would like to order. This approach ensures that the ordering process is tailored to individual needs, enhancing customer satisfaction and safety.

3.2.1 BPMN 2.0 Implementation

The Figure 3.1 presents the BPMN model developed to represent the Meal Suggestion System within a hypothetical restaurant scenario. The model is structured as a pool divided into two lanes: Customer and Menu. The process is initiated by the customer by scanning the QR code to see the menu, then he must insert the parameters like allergies, food category, and calorie-conscious level, finally the tool provides the suggestions about meals. Once the customer input is received, the restaurant lane processes the information by invoking a custom task called Suggest Meal that represents the core logic of the system. This task determines which meals are suitable for the guest based on the provided parameters. The reasoning considers dietary restrictions, allergen avoidance, nutritional preferences and course preference to generate personalized meal suggestions. Following this, the restaurant sends the suggested meals back to the guest. Upon delivery of the suggestions, the business process concludes, having fulfilled its goal of offering a tailored menu.

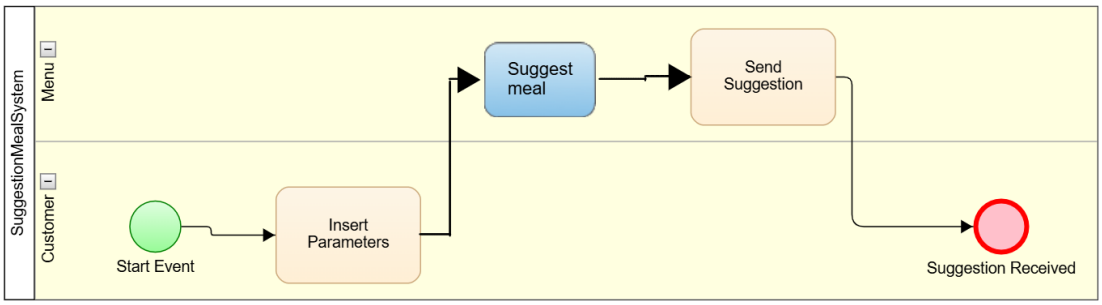

Figure 3.1: BPMN 2.0 model built in AOAME

After building the BPMN 2.0 Jena Fuseki has been used to execute some queries to examine the graph, two queries have been created:

- **Query 1:** used for see the property and their relative value of the extended task "Suggest Meal". Result visible in Figure 3.2

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
PREFIX lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/michele.martini/kebi2025#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?property ?value
WHERE {
  mod:SuggestMeal_e3453738-7fcf-48f8-a0f6-cdcfc2c24d5d ?property ?value
     .
}
```

- **Query 2:** used for analyze the extended task and return the meals which satisfied all data property described.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
PREFIX lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/michele.martini/kebi2025#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  mod:SuggestMeal_e3453738-7fcf-48f8-a0f6-cdcfc2c24d5d lo:
    guest_hasCategory ?category .
  mod:SuggestMeal_e3453738-7fcf-48f8-a0f6-cdcfc2c24d5d lo:
    guest_hasCalorieConscious ?levelCC .
  mod:SuggestMeal_e3453738-7fcf-48f8-a0f6-cdcfc2c24d5d lo:
    guest_hasAllergy ?allergy .
  mod:SuggestMeal_e3453738-7fcf-48f8-a0f6-cdcfc2c24d5d lo:
    guest_hasCoursePreference ?course .

  BIND(IF(?category = "Vegetarian", kebi:category_vegetarian,
           IF(?category = "Carnivorous", kebi:category_carnivorous,
             IF(?category = "Omnivore", kebi:category_omnivore, ?
    category))) AS ?finalCategory) .

  BIND(IF(?course = "First Dish", kebi:course_first-dish,
                IF(?course = "Second Dish", kebi:course_second-dish,
```

```
                    IF(?course = "Side Dish", kebi:course_sidedish,
                       IF(?course = "Dessert", kebi:course_dessert, ?
  course)))) AS ?finalCourse) .

BIND(IF(?allergy = "Gluten", kebi:grain_containsGlutenIntolerance,
            IF(?allergy = "Lactose", kebi:
  dairy_containsLactoseIntolerance,
                IF(?allergy = "none", "none", ?allergy))) AS ?
  finalAllergy) .

?meal a kebi:Meal .
?meal kebi:meal_hasLevelOfCalorieConscious ?kcal .
FILTER(?kcal = ?levelCC)

?meal kebi:food_hasCategory ?finalCategory .
?meal kebi:meal_hasCourse ?finalCourse .

OPTIONAL {
  ?meal kebi:meal_hasIngredient ?ingredient .
  ?ingredient rdf:type ?ingredientType .
  FILTER ((?finalAllergy = "none") ||
          (?finalAllergy = kebi:grain_containsGlutenIntolerance && ?
  ingredientType = kebi:Grain) ||
          (?finalAllergy = kebi:dairy_containsLactoseIntolerance && ?
  ingredientType = kebi:Dairy))
}
FILTER (!BOUND(?ingredient))
}
```

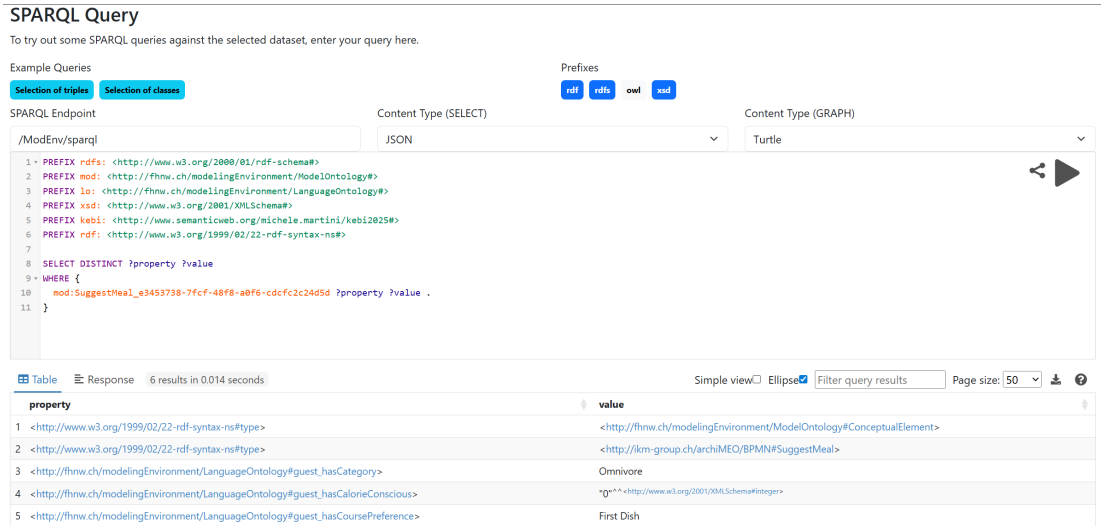

Figure 3.2: Query 1 Result

For the second query a couple of examples have been tested:

- **Test 1:** As we can see in Figure 3.3, we set up the attributes of our BPMN class and then fire the query in Jena Fuseki that gives back the result (Figure 3.4).

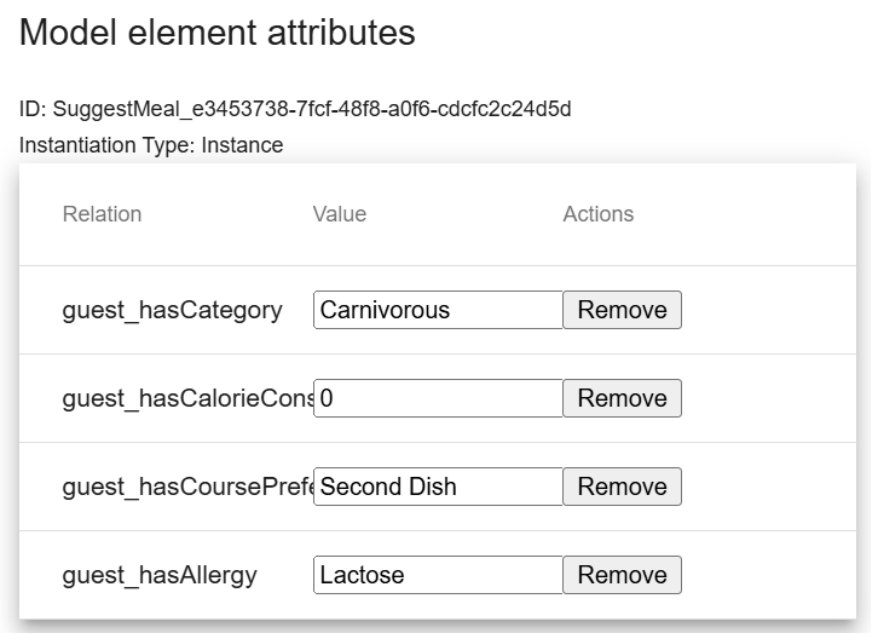

Figure 3.3: Test 1 input

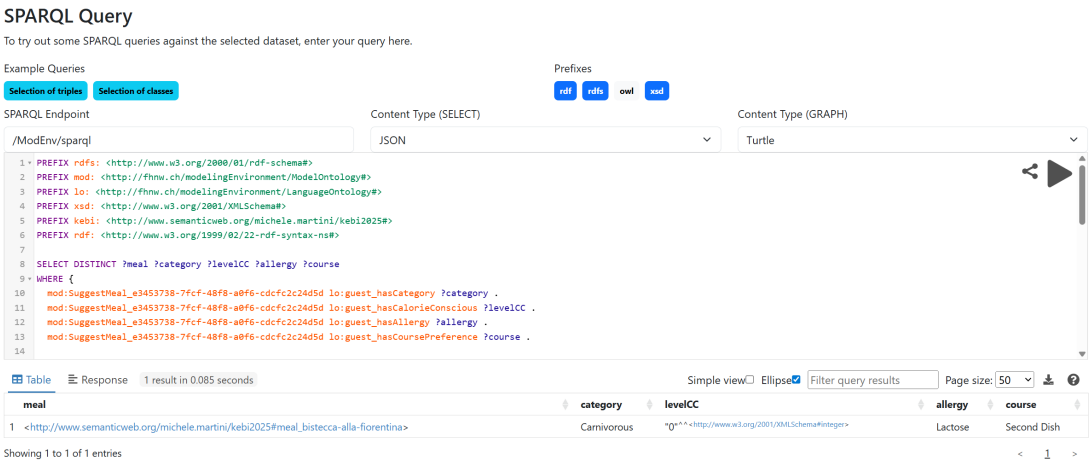

Figure 3.4: Test 1 Result

- **Test 2:** Just like the first test the attributes are set up and visible in Figure 3.3, then the query is fired in Jena Fuseki that gives back the result (Figure 3.4).

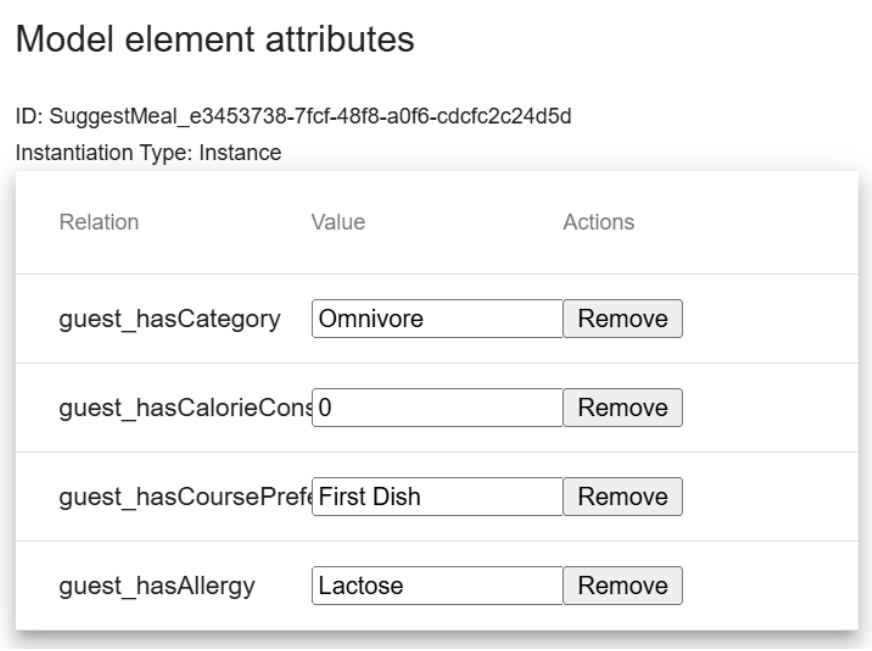

Figure 3.5: Test 2 Input

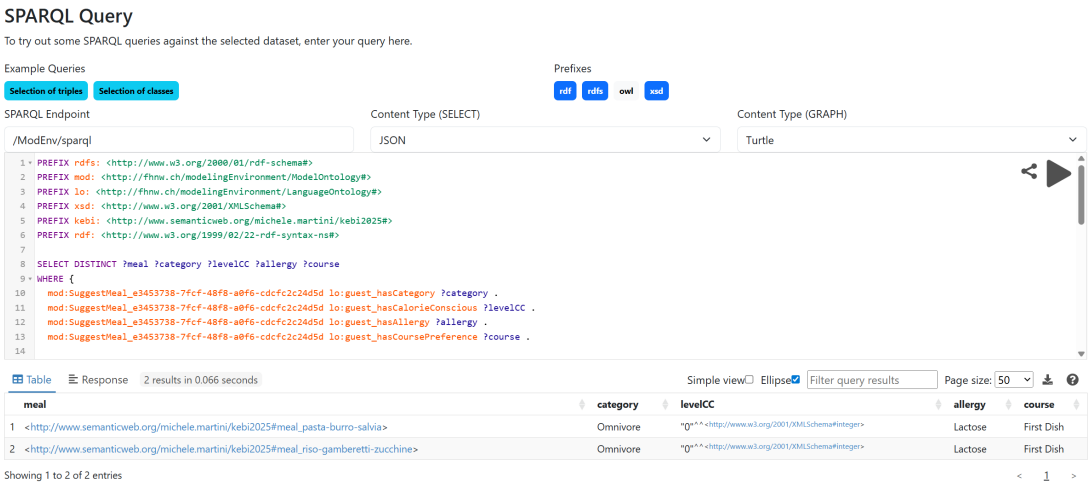

Figure 3.6: Test 2 Result

4. Conclusions

The project aimed to improve the dining experience by creating a system that customizes restaurant menus based on customer preferences and dietary restrictions. The main objectives of the project were to give the ability to navigate large digital menus on small screens and ensure that guests are presented only the meals they prefer to consume. Multiple knowledge-based solutions have been developed to recommend meals based on guest profiles using various representation languages.

During this project, I discovered the world of knowledge engineering, exploring various knowledge-based solutions, each with its own set of advantages and disadvantages. For me the most important part of my academic journey is to discover new things, acquire new knowledge and learn new subjects, which is something that i deeply enjoy. This project allowed me to do just that. Now a small overview of my impressions on each solution will be provided.

4.1 Decision Tables

Decision tables operate in a very straightforward manner, taking inputs and generating outputs based on predefined rules organized in rows. The outputs are selected via "Hit Policy", which dictates how results are selected and grouped, if necessary. In the project was used the "Collect Policy", enabling concurrent evaluation of all rules. Output results are aggregated into sets for respective output columns. This operational clarity makes it very user-friendly and easy to implement. However, in complex contexts with numerous variables and intricate conditions, the applicability of decision tables can become impractical or exceedingly intricate. Manipulating data using decision tables can become tedious and time-consuming, in addition to difficulties in debugging, made more difficult by limited support from available online tools. In conclusion, while decision tables provide a easy to understand and easy to manage rule-based logic, their effectiveness diminishes as complexity grows. They are best suited for relatively simple decision-making processes, where their structured approach can be fully utilized without being overwhelmed by more complex conditions and data sets.

4.2 Prolog

I really enjoyed working with Prolog during the project. It provided me with an opportunity to explore a new programming language that, although not excessively challenging to understand, presented me with the possibility to learn a new way of reasoning with code which I was unaware of. Prolog greatly emphasizes recursion, lacks conventional loop structures, and straightforward arithmetic operations, however, it excels in scenarios where there is need for logical reasoning and querying knowledge bases,

where its declarative nature makes data management and retrieval simple and powerful. Despite the enjoyable learning experience, Prolog has some drawbacks, for me the main one was the clarity of the syntax and logic that sometimes could be ambiguous, which led to confusion during development. Additionally, while Prolog is optimized for logical operations, performance can take a hit when handling large knowledge bases, resulting in extended computation times for obtaining results. In conclusion i really liked discovering and understanding Prolog, which worked good for this project, but for larger project that require fast response it may not be the best solution.

4.3 Protégé

I viewed this approach as similar to Object-Oriented programming languages, therefore it was easier to understand and use with respect to the other solutions. In this framework is possible to define objects along with their relationships and attributes, on top of that there is the possibility to infer knowledge from these object instances. Protégé uses SPARQL to handle queries, this querying method closely resemble standard SQL, which simplified the creation of queries by my part. The SHACL validator is very useful for testing whether a property is correctly defined or not. SWRL rules enable type inference allowing you to assign properties to individuals, an example in this project was the inference to determine whether a meal contained allergenic based on its ingredients. The advantages of this approach include enhanced reasoning capabilities, thanks to SWRL, and strong data integrity and lack of errors, thanks to SHACL. The disadvantages of this approach lies precisely in the use of SHACL and SWRL. The main downside of SHACL is that it requires a careful definition of shapes, which can become time-consuming and complex, especially as the ontology grows in size. For SWRL the main problem is that querying tools may lack the built-in reasoner for type inference, making the creation of queries more difficult since they need to be more precise.

4.4 AOAME

I found AOAME to be a valuable tool due to its ability to adapt BPMN 2.0 with any ontology. This allows the creation of a diagram that is much simpler and easier to read compared to many others that do not utilize this tool. In this project, by creating the class "Suggest Meal", a subclass of "Task", I was able to develop a new and intuitive graphical notation for the restaurant manager. Additionally, four new types of properties (course, allergy, calorie-conscious, and category) were introduced and integrated them into our ontology. By using Apache Jena Fuseki, the ontology was integrated into a triplestore. This integration allowed the execution of SPARQL queries, dynamically generating menu suggestions personalized for each customer unique needs. The process of creating queries was not difficult since it was the same tool used in Protégé. However, AOAME faces challenges such as significant bugs that impact user experience and system stability. Despite these cons, AOAME remains a very powerful and highly useful tool to intuitively improve and streamline the decision-making process for business managers.

4.5 Conclusion Summary

In conclusion, each knowledge-based solution has its strengths and weaknesses. Decision tables are straightforward and user-friendly but can become difficult to handle in complex scenarios. Prolog offers intuitive data management but struggles with large knowledge bases. Protégé provides a powerful hybrid approach for object-oriented knowledge modeling but requires careful rule management. AOAME excels in integrating meta-modeling and ontologies for enhanced data integration and reasoning but suffers from stability issues. Choosing the appropriate solution depends on the specific needs of the project, the complexity of the knowledge base, and the desired level of scalability.