University of Camerino

School of Science and Technology

Master degree in Computer Science (LM-18)
Knowledge Engineering and Business Intelligence

Personalized Menu System

Authors:
Micarelli Simone
simone.micarelli@studenti.unicam.it

Nappo Davide
davide.nappo@studenti.unicam.it

Supervisor
Prof. Knut Hinkelmann

Prof. Emanuele Laurenzi

A.Y. 2024/2025

Contents

Listings

List of Figures

1. Introduction

This document aims to outline the process undertaken to complete the Knowledge Engineering and Business Intelligence project. Section 1.1 will provide a description of the system to be developed, while Section 1.2 will define the tasks to be accomplished. Following this, we will examine the System Design phase (Chapter 2), the implemented Knowledge-Based Solutions (Chapter 3), and the description of our implementation of Agile and Ontology-based Meta-Modelling. In the Conclusion chapter (Chapter 5), we will offer our personal perspectives on the solutions presented. The solution developed can be found at this link: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project

Nel Capitolo 1 illustreremo prima le motivazioni che ci hanno spinto a perseguire l'obiettivo descritto e quindi la struttura della tesi.

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 following are the tasks to tackle for completing the Knowledge Engineering and Business Intelligence projects:

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)

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. System Design

In this chapter, we outline our ideas regarding the data inputs and outputs that the system will process.

2.1 Input format

We have created an input form using Google Forms which will act as the basis for the remainder of the project. An example of this input form is displayed in Figure 2.1. The form is intended for completion by the customer in order to generate a tailored list of the most suitable meals according to their preferences.

In the form shown in Figure 2.1, the user's first step is to choose between "carnivorous," "vegetarian," or "omnivore" (indicated by radio buttons, meaning only one choice is possible). This selection is mandatory.

The user must then specify any food intolerances (such as lactose or gluten).

Next, they indicate their preference for calorie intake by selecting one of four options: high-calorie foods (level 0), medium-high calorie foods (level 1), medium-low calorie foods (level 2), or low-calorie foods (level 3). This is also a mandatory choice, with level 0 preselected by default.

Lastly, the user can fill out the relevant section to specify the desired course category, choosing from "Appetizer," "First Course," "Main Course," "Pizza," "Drink," or "Dessert."

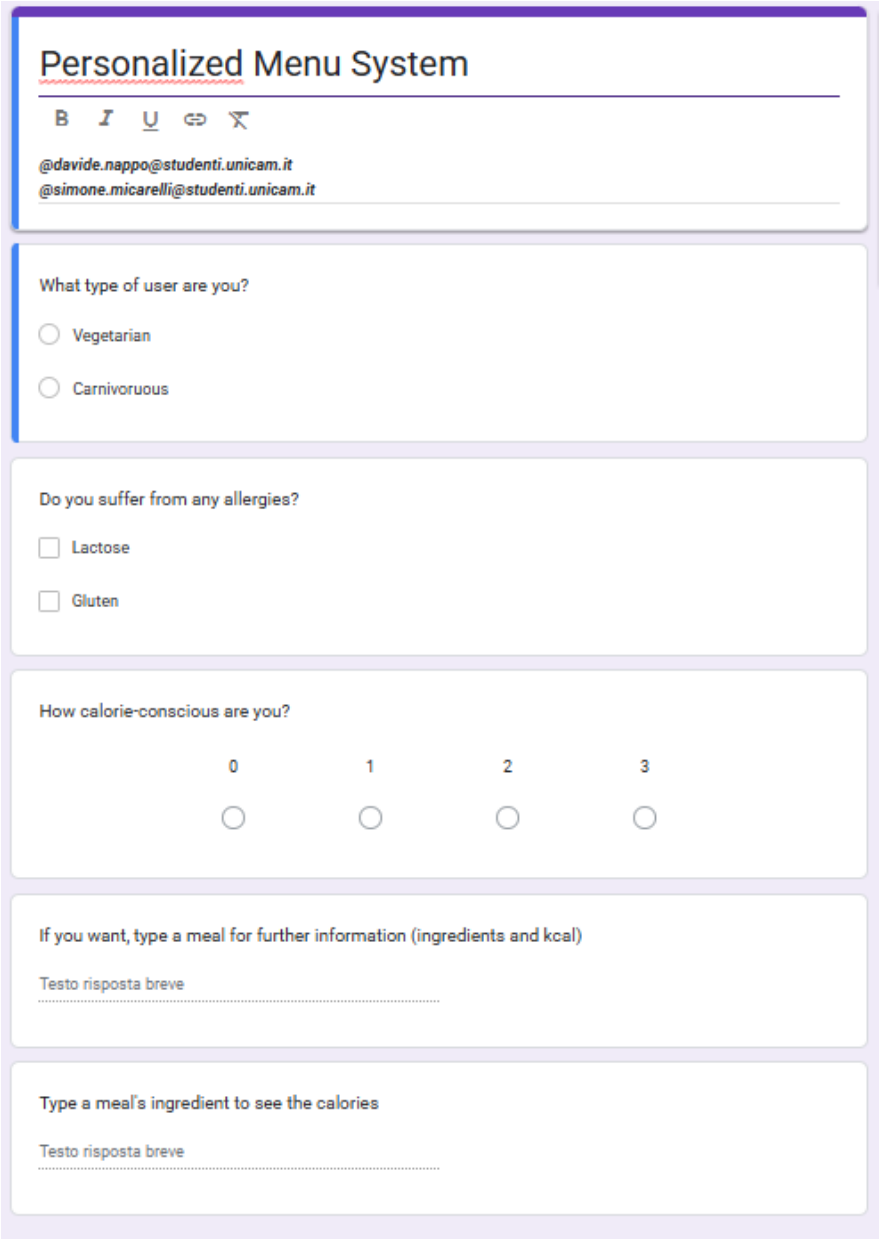


Figure 2.1: Sample Input form for the customer

2.2 Output format

In the Figure 2.2 we can see an example of how the form might be completed by the user.

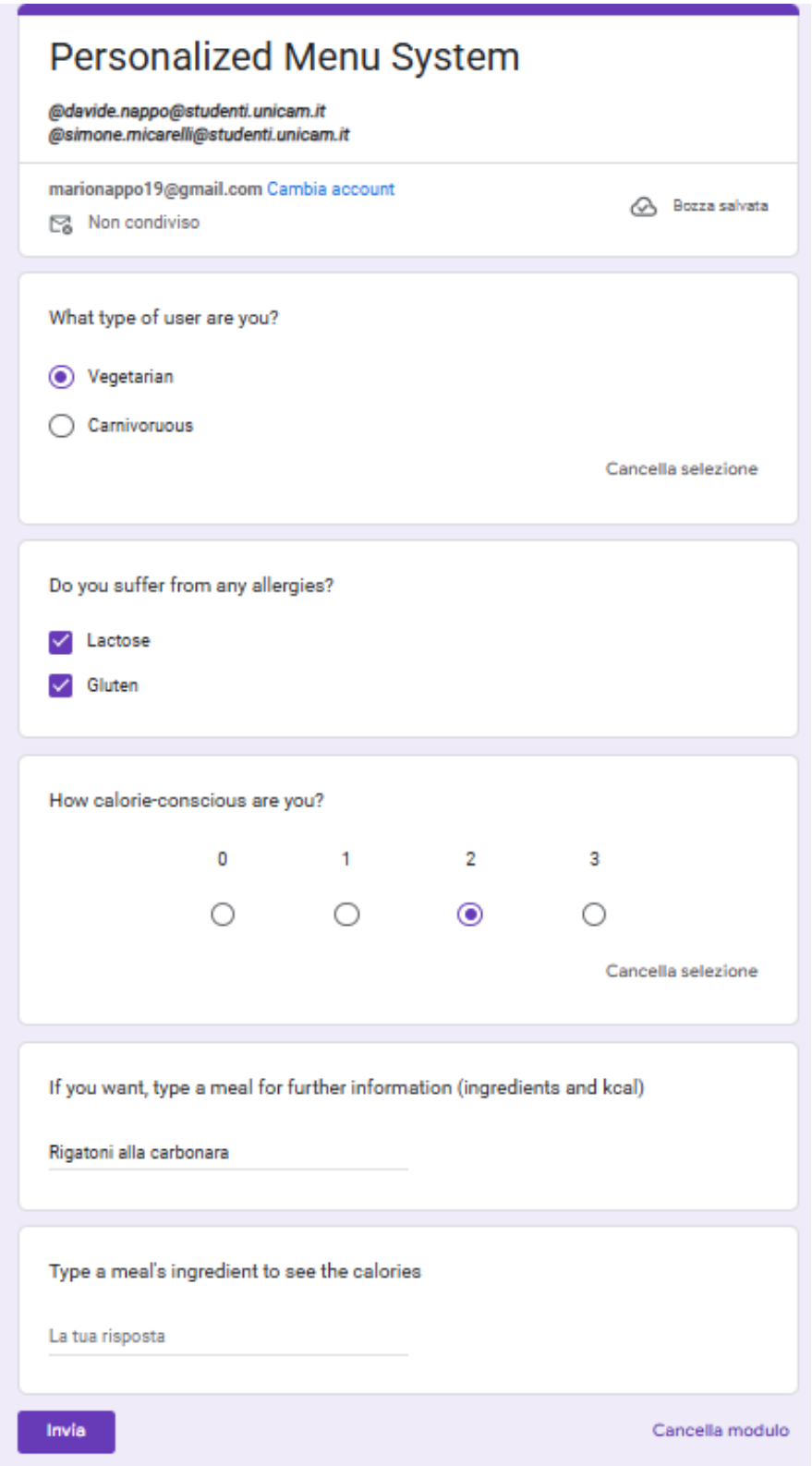


Figure 2.2: Sample filled out input form

3. Knowledge Based Solution

Building on the previously defined input form and format, our knowledge base can be queried in three distinct ways. The first approach uses Decision Tables (Section 3.1) to align with user preferences and filter out unwanted meal options. The second method employs Prolog (Section 3.2), enhancing the logic of the Decision Tables by leveraging the capabilities of a programming language. This enables users to manipulate data and perform additional operations. Finally, Knowledge Graphs (Section 3.3) allow us to store information in a structured format and query it in a manner similar to a database. In the following sections, we will examine each of these three methods, outlining their main strengths and limitations.

3.1 Decision Model

In the field of business analysis, the Decision Model and Notation (DMN), defined by the Object Management Group (OMG), is recognized as an established standard. This framework is used to describe and organize recurring decision-making processes within organizations, enabling decision models to be easily shared among different stakeholders. By following this standard, organizations can adopt a consistent modeling language, supporting decision-making practices that align with effective management strategies and ensure compliance with business rules.

DMN Elements

The DMN standard consists of four main components:

- Decision Requirements Diagrams: Illustrate the relationships among the various decision-making elements, creating a network of dependencies.
- Decision tables[1]: Present a clear and concise visual format for defining actions based on specific conditions.
- Business context: Refers to the set of contextual factors that influence the decision-making process within an organization.
- Friendly Enough Expression Language (FEEL): A dedicated language used to evaluate expressions within decision tables.

[1] Wikipedia. *Decision Table.* URL: https://en.wikipedia.org/wiki/Decision_table

3.1.1 Camunda

Camunda[2]is an open source platform developed in Java, designed to support workflow and process automation for organizations of any size. Its capabilities include creating decision models as well as, more broadly, BPMN process diagrams and DMN decision tables. In addition, Camunda provides REST APIs[3], enabling developers who do not work with Java to access its functionalities. In our project, we used Camunda to build the knowledge base for meal management and to define decision-making processes and rules consistent with our organizational objectives.

3.1.2 Our solution

As shown in Figure 3.1, our model comprises two Decision Tables and a single Literal Expression. Unlike the initial submission, we first applied constraints to the ingredients and then to the meals. The following section provides a detailed description of each table.

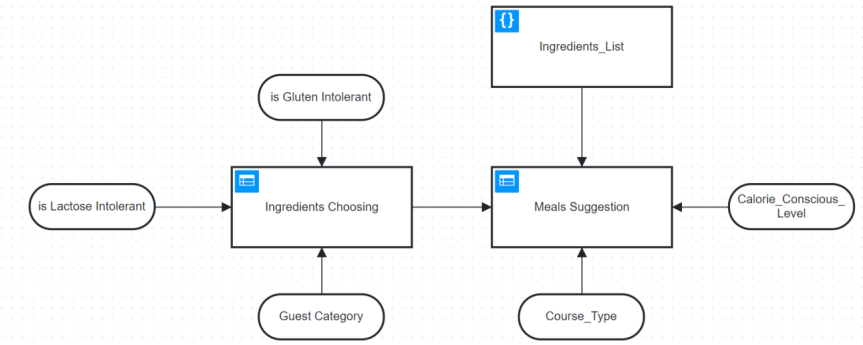


Figure 3.1: Our decision model in Camunda

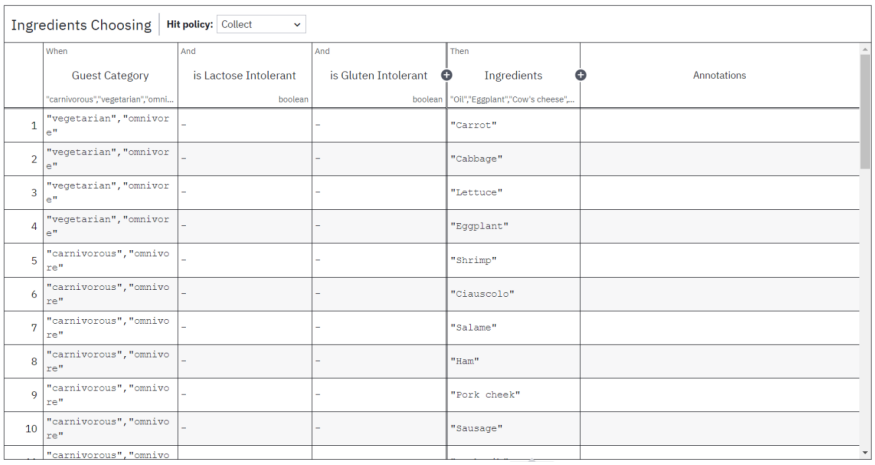
3.1.3 Decision Tables

Referring to Figure 3.1, we describe each Decision Table below, along with its respective inputs and outputs.

- Ingredient Selection: This table receives multiple input parameters, such as the user's dietary category—omnivore, carnivore, or vegetarian. It also considers two boolean values indicating whether the user is lactose intolerant and whether they are gluten intolerant. Based on these parameters, the table returns a list of meals filtered according to the user's preferences. As shown in Figures 3.2 and 3.3, the Guest Category input includes three options: Carnivorous, for users who follow a meat-based diet without consuming vegetables; Vegetarian, for users who avoid meat; and Omnivore, for users who eat both meat and vegetables. This structure was designed with scalability in mind, allowing, for example, the future addition of a Vegan category to identify vegan-friendly ingredients. For the boolean inputs, a hyphen ("-") in the Decision Table indicates that the ingredient is free from allergens. Conversely, a value of "false" means the ingredient contains the allergen.

[2] *Camunda.* URL: https://camunda.com/
[3] *Camunda Documentation.* URL: https://docs.camunda.org/manual/7.19/reference/dmn/feel/

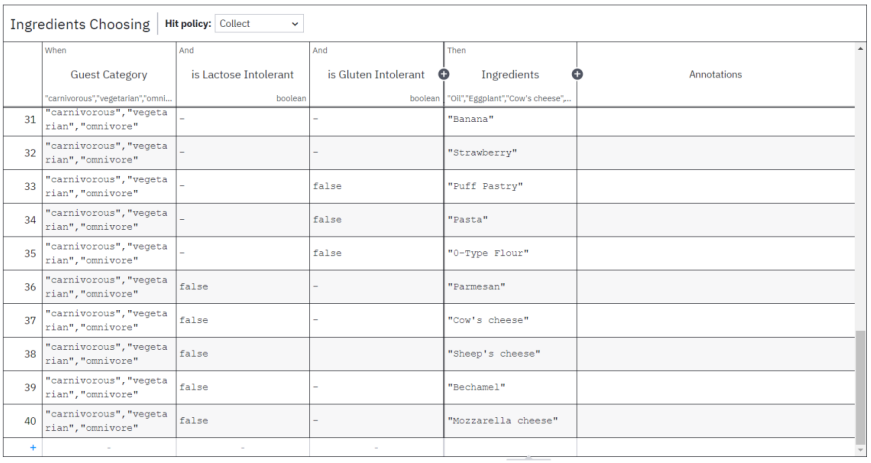
In such cases, if the user has the corresponding intolerance, the ingredient will be removed, and any meals containing it will not be recommended.



Ingredients Choosing | Hit policy: Collect

	When	And	And	Then	
	Guest Category "carnivorous","vegetarian","omni...	is Lactose Intolerant boolean	is Gluten Intolerant boolean	Ingredients "Oil","Eggplant","Cow's cheese",...	Annotations
1	"vegetarian","omnivore"	-	-	"Carrot"	
2	"vegetarian","omnivore"	-	-	"Cabbage"	
3	"vegetarian","omnivore"	-	-	"Lettuce"	
4	"vegetarian","omnivore"	-	-	"Eggplant"	
5	"carnivorous","omnivore"	-	-	"Shrimp"	
6	"carnivorous","omnivore"	-	-	"Ciauscolo"	
7	"carnivorous","omnivore"	-	-	"Salame"	
8	"carnivorous","omnivore"	-	-	"Ham"	
9	"carnivorous","omnivore"	-	-	"Pork cheek"	
10	"carnivorous","omnivore"	-	-	"Sausage"	

Figure 3.2: Ingredients choosing (1)



Ingredients Choosing | Hit policy: Collect

	When	And	And	Then	
	Guest Category "carnivorous","vegetarian","omni...	is Lactose Intolerant boolean	is Gluten Intolerant boolean	Ingredients "Oil","Eggplant","Cow's cheese",...	Annotations
31	"carnivorous","vegetarian","omnivore"	-	-	"Banana"	
32	"carnivorous","vegetarian","omnivore"	-	-	"Strawberry"	
33	"carnivorous","vegetarian","omnivore"	-	false	"Puff Pastry"	
34	"carnivorous","vegetarian","omnivore"	-	false	"Pasta"	
35	"carnivorous","vegetarian","omnivore"	-	false	"0-Type Flour"	
36	"carnivorous","vegetarian","omnivore"	false	-	"Parmesan"	
37	"carnivorous","vegetarian","omnivore"	false	-	"Cow's cheese"	
38	"carnivorous","vegetarian","omnivore"	false		"Sheep's cheese"	
39	"carnivorous","vegetarian","omnivore"	false	-	"Bechamel"	
40	"carnivorous","vegetarian","omnivore"	false	-	"Mozzarella cheese"	
+					

Figure 3.3: Ingredients choosing (2)

- Meal Suggestion: This table uses two inputs provided by the user—one obtained from the previous Decision Table and the other from the Literal Expression. The first parameter, Course Type, specifies the meal category (e.g., Appetizer, First Course, etc.), while the second, Calorie-Conscious Level, represents the user's level of attention to calorie intake, ranging from 0 to 3. Based on these inputs, the table generates a list of meals filtered according to the selected calorie level, with 0 set as the default. A level of 0 indicates no specific concern regarding calorie content, whereas levels 1, 2, and 3 reflect progressively greater attention to calorie intake, with level 3 corresponding to meals containing fewer than 250 kcal.

Figure 3.4: Meals suggestion (1)

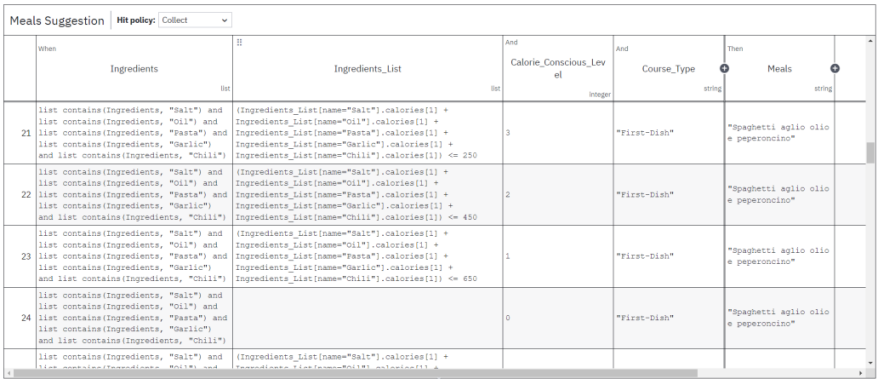


Figure 3.5: Meals suggestion (2)

As illustrated in Figures 3.4 and 3.5, the chosen Hit Policy is "Collect," which aggregates results according to defined criteria. The table performs four evaluations for each meal, corresponding to the user's selected Calorie-Conscious Level. While fewer levels could have been implemented, using four allows for greater precision and stricter filtering.

The first column, Ingredients, checks whether all required ingredients are available. For example, if a user is gluten intolerant, Puff Pastry will be excluded, and the meal will not be recommended. The same logic applies to lactose intolerance.

Proceeding across the table, the meal's total calorie count is compared against a threshold that varies by level: fewer than 250 kcal for level 3, fewer than 450 kcal for level 2, and fewer than 650 kcal for level 1. For level 0, no calorie restriction is applied.

Alongside the Calorie-Conscious Level, the user also specifies the Course Type, which is used to further refine the selection and produce a list of suitable meal options. The Decision Table's output is therefore a tailored list of meals matching the user's preferences.

3.1.4 Literal Expression

To optimize the workflow, we created a Literal Expression that stores ingredients alongside their respective calorie values in a Map structure, where each ingredient is paired

with a numerical value indicating its caloric content. A partial view of Literal Expression 3.6 is shown below.

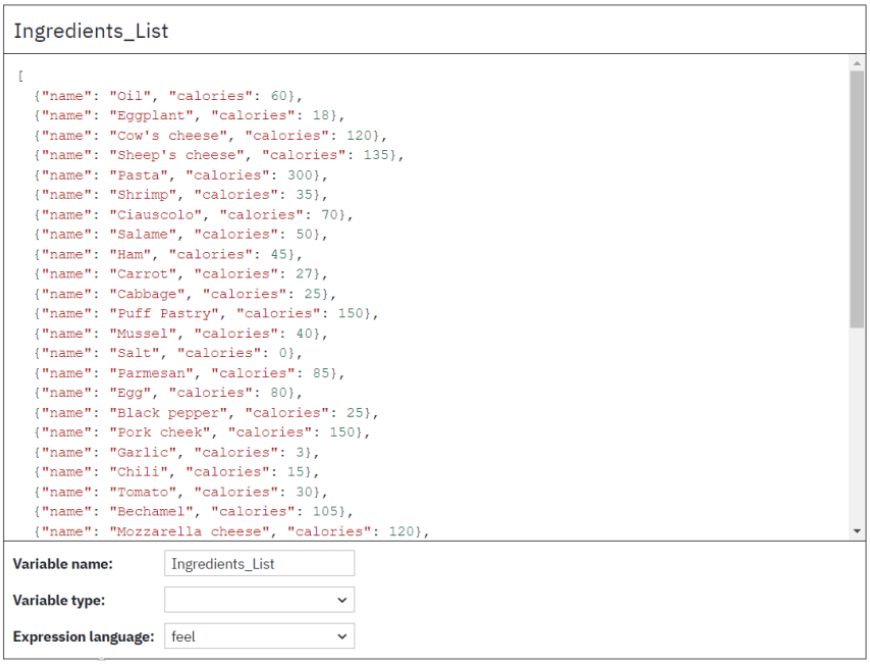


Figure 3.6: Ingredients list literal expression

3.1.5 DMN simulation

Finally, we provide an example of meal recommendations generated by our model through a DMN Simulator. In this initial simulation, the 'Guest Category' was set to 'Omnivore', 'Lactose intolerance' was selected, the 'Calorie-Conscious Level' was set to 2, and 'Appetizer' was chosen as the preferred course. As illustrated in Figure 3.7, the resulting output consisted of 'Involtini Primavera' and 'Affettati Misti'.

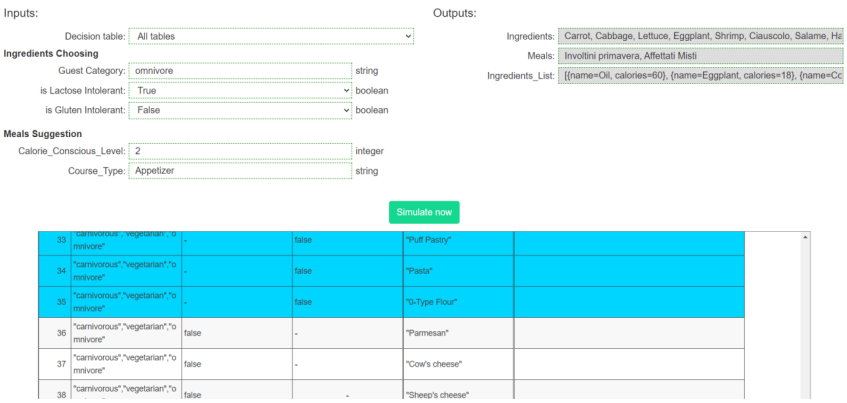


Figure 3.7: DMN first implementation

If the 'Guest Category' is changed to 'Vegetarian', 'Affettati Misti' will no longer appear in the results, leaving only 'Involtini Primavera', since it is lactose-free and made exclusively with vegetarian ingredients, as shown in Figure 3.8.

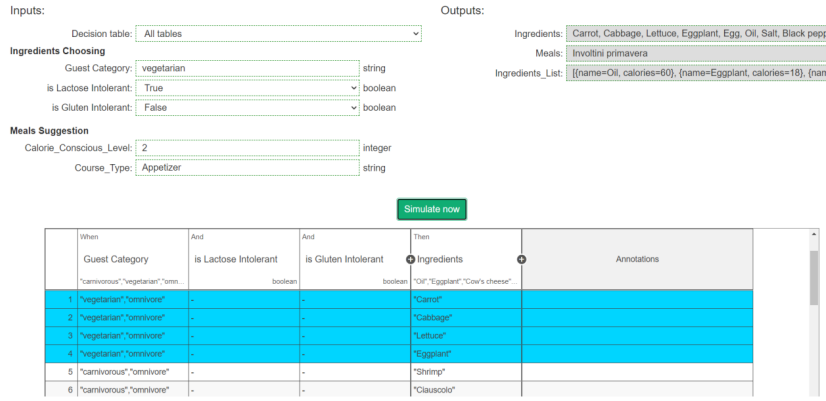


Figure 3.8: DMN second implementation

3.2 Prolog

Logic programming employs logic as a declarative representation language, using backward chaining as its primary inference mechanism. Prolog[4] is a logic programming language closely linked to artificial intelligence and computational linguistics, whose strength lies in its logical foundations.

The syntax of Prolog includes:

- **Symbols:** (and), ; (or), :- (if), not (not);
- **Variables:** a sequence of letters, digits, and underscores (_), beginning with either an uppercase letter or an underscore;
- **Facts:** predicate statements that declare something about the problem domain;
- **Rules:** predicate statements that use logical implication (:-) to define relationships between facts. A Prolog rule has the structure:

$$left\_hand\_side :- right\_hand\_side.$$

- **Queries:** evaluated against the set of facts and rules stored in the database. When a query is submitted, Prolog attempts to prove its validity. If successful, it returns "yes" and displays all variable bindings that led to the answer; if it cannot be proven, it returns "no."

Prolog also uses backtracking, a process in which the interpreter evaluates different predicates to verify their validity, systematically exploring possible paths until it finds one that satisfies the goal.

Additionally, Prolog includes numerous built-in predicates—native rules that provide useful functionality, such as length(?List, ?Length), which returns the length of a list. Due to its expressive power, Prolog can define a knowledge base in only a few lines of code. In our project, we leveraged this capability to build a rule-based system for managing meals and ingredients.

[4] *Prolog Wikipedia.* URL: https://it.wikipedia.org/wiki/Prolog

3.2.1 Prolog Implementation

We implemented the model defined in the DMN file using Prolog and tested it through SWISH Prolog[5].

Similarly to Task 1.1, we restructured the Prolog file so that restrictions are first applied at the ingredient level and then at the meal level.

With this approach, if an ingredient is classified as vegetarian and a meal contains that ingredient (provided that it does not also include a carnivorous ingredient), the meal is categorized as vegetarian. The same rule applies to carnivorous ingredients. However, if a meal contains both carnivorous and vegetarian ingredients, it is not classified as either category and is therefore considered omnivorous (note that all carnivorous and vegetarian meals inherently fall into the omnivorous category as well). Meals containing neither carnivorous nor vegetarian ingredients remain available to both carnivores and vegetarian users.

The Prolog code begins by defining a list of ingredients used across various meals. Each ingredient is declared through the 'ingredient' predicate, which enumerates all available components for meal preparation. This list encompasses a wide range of items, including vegetables, meats, dairy products, and staples such as flour and oil. This foundational step supports the subsequent stages of the program, where each ingredient will be linked to its caloric value and dietary classification.

```
% Define the list of ingredients used in meals
ingredient(salt).
ingredient(oil).
ingredient(carrot).
ingredient(cabbage).
ingredient(puff_pastry).
ingredient(ciauscolo).
ingredient(salame).
ingredient(ham).
ingredient(black_pepper).
ingredient(cow_cheese).
ingredient(sheep_cheese).
ingredient(parmesan).
ingredient(mozzarella_cheese).
ingredient(pasta).
ingredient(egg).
ingredient(pork_cheek).
ingredient(shrimp).
ingredient(mussel).
ingredient(chili).
ingredient(garlic).
ingredient(tomato).
ingredient(bechamel).
ingredient(minced_meat).
ingredient(pork_chop).
ingredient(pork_rib).
ingredient(sausage).
ingredient(steak).
ingredient(calamari).
ingredient(sword_fish).
ingredient(salmon).
ingredient(potato).
ingredient(eggplant).
ingredient(type_0_flour).
ingredient(water).
ingredient(sugar_and_food_coloring).
ingredient(corn).
ingredient(lettuce).
ingredient(apple).
```

[5] *Prolog Testing.* URL: https://swish.swi-prolog.org/

```
ingredient(banana).
ingredient(strawberry).
```

Listing 3.1: Ingredient facts

This section defines the caloric content of each ingredient through the 'kcal_ingredient' predicate, associating every ingredient with its corresponding value in kilocalories. Assigning these values is essential for assessing the nutritional profile of meals. The provided data enables precise caloric calculations, which are fundamental to dietary planning and nutritional evaluation.

```
% Define the caloric content of each ingredient
kcal_ingredient(salt, 0).
kcal_ingredient(oil, 60).
kcal_ingredient(carrot, 27).
kcal_ingredient(cabbage, 25).
kcal_ingredient(puff_pastry, 150).
kcal_ingredient(ciauscolo, 70).
kcal_ingredient(salame, 50).
kcal_ingredient(ham, 45).
kcal_ingredient(black_pepper, 25).
kcal_ingredient(cow_cheese, 120).
kcal_ingredient(sheep_cheese, 135).
kcal_ingredient(parmesan, 85).
kcal_ingredient(mozzarella_cheese, 120).
kcal_ingredient(pasta, 300).
kcal_ingredient(egg, 80).
kcal_ingredient(pork_cheek, 150).
kcal_ingredient(shrimp, 35).
kcal_ingredient(mussel, 40).
kcal_ingredient(chili, 15).
kcal_ingredient(garlic, 3).
kcal_ingredient(tomato, 30).
kcal_ingredient(bechamel, 105).
kcal_ingredient(minced_meat, 120).
kcal_ingredient(pork_chop, 125).
kcal_ingredient(pork_rib, 70).
kcal_ingredient(sausage, 205).
kcal_ingredient(steak, 485).
kcal_ingredient(calamari, 50).
kcal_ingredient(sword_fish, 120).
kcal_ingredient(salmon, 150).
kcal_ingredient(potato, 450).
kcal_ingredient(eggplant, 18).
kcal_ingredient(type_0_flour, 500).
kcal_ingredient(water, 0).
kcal_ingredient(sugar_and_food_coloring, 465).
kcal_ingredient(corn, 50).
kcal_ingredient(lettuce, 25).
kcal_ingredient(apple, 35).
kcal_ingredient(banana, 40).
kcal_ingredient(strawberry, 22).
```

Listing 3.2: Ingredients/Kcals association

Here are ingredients that are classified as carnivorous. If a meal contains a carnivorous ingredient and no vegetarian ingredients, then that meal is considered carnivorous.

```
% Define which ingredients are carnivorous
ingredient_carnivore(ciauscolo).
ingredient_carnivore(salame).
ingredient_carnivore(ham).
ingredient_carnivore(pork_cheek).
ingredient_carnivore(minced_meat).
ingredient_carnivore(pork_rib).
ingredient_carnivore(sausage).
ingredient_carnivore(shrimp).
ingredient_carnivore(mussel).
```

```
ingredient_carnivore(steak).
ingredient_carnivore(calamari).
ingredient_carnivore(sword_fish).
ingredient_carnivore(salmon).
```

Listing 3.3: Carnivorous ingredients

Here are ingredients that are classified as vegetarian. If a meal contains a vegetarian ingredient and no carnivorous ingredients, then that meal is considered vegetarian.

```
% Define which ingredients are vegetarian
ingredient_vegetarian(carrot).
ingredient_vegetarian(cabbage).
ingredient_vegetarian(eggplant).
ingredient_vegetarian(lettuce).
ingredient_vegetarian(apple).
ingredient_vegetarian(banana).
ingredient_vegetarian(strawberry).
```

Listing 3.4: Vegetarian ingredients

Here are all the ingredients that contain allergens (gluten or lactose).

```
% Define which ingredients cause lactose intolerance
ingredient_with_lactose_intolerance(cow_cheese).
ingredient_with_lactose_intolerance(sheep_cheese).
ingredient_with_lactose_intolerance(parmesan).
ingredient_with_lactose_intolerance(mozzarella_cheese).
ingredient_with_lactose_intolerance(bechamel).

% Define which ingredients contain gluten
ingredient_with_gluten_intolerance(type_0_flour).
ingredient_with_gluten_intolerance(pasta).
ingredient_with_gluten_intolerance(puff_pastry).
```

Listing 3.5: Ingredients associated with an intolerance

This section defines meals using the meal predicate, which specifies the meal name, the course type (such as appetizer or first dish), and a list of ingredients. Structuring meals in this way enables precise querying and analysis of available options based on their composition. Such organization supports the generation of tailored meal recommendations and detailed nutritional evaluations.

```
% Define various meals along with their courses and ingredients
meal(involtini_primavera, appetizer, [salt, oil, carrot, cabbage, puff_pastry]).
meal(affettati_misti, appetizer, [salt, ciauscolo, salame, ham, black_pepper]).
meal(formaggi_misti, appetizer, [salt, cow_cheese, sheep_cheese, parmesan]).
meal(tagliatelle_alla_marinara, first_dish, [salt, oil, pasta, shrimp, mussel]).
meal(rigatoni_alla_carbonara, first_dish, [salt, oil, pasta, sheep_cheese, egg,
    black_pepper, pork_cheek]).
meal(trofie_cacio_e_pepe, first_dish, [salt, oil, pasta, black_pepper,
    sheep_cheese]).
meal(spaghetti_aglio_olio_e_peperoncino, first_dish, [salt, oil, pasta, garlic,
    chili]).
meal(lasagne, first_dish, [salt, oil, carrot, pasta, parmesan, tomato, bechamel,
    minced_meat]).
meal(grigliata_di_maiale_mista, second_course, [salt, pork_rib, sausage, pork_chop]).
meal(bistecca_alla_fiorentina, second_course, [salt, oil, steak]).
meal(pesce_arrosto, second_course, [salt, oil, shrimp, mussel, calamari, sword_fish,
    salmon]).
meal(verdure_miste, second_course, [salt, cabbage, eggplant, lettuce]).
meal(patatine_fritte, sidedish, [salt, oil, potato]).
meal(melanzane_grigliate, sidedish, [salt, oil, eggplant]).
meal(insalata_mista, sidedish, [salt, oil, lettuce, corn]).
meal(pizza_margherita, main_dish, [salt, oil, cow_cheese, tomato, water,
    type_0_flour]).
meal(pizza_bianca, main_dish, [salt, oil, water, type_0_flour]).
meal(water, drink, [water]).
```

```
meal(coca_cola, drink, [water, sugar_and_food_coloring]).
meal(fruit_salad, dessert, [apple, banana, strawberry]).
```

Listing 3.6: Meal facts

The predicates described below classify a meal according to its dietary category:

- vegetarian_meal checks that all ingredients are vegetarian or non-carnivorous.
- carnivorous_meal ensures that all ingredients are carnivorous or non-vegetarian.
- omnivore_meal identifies meals containing both carnivorous and vegetarian ingredients.

These predicates allow meals to be categorized based on dietary preferences, helping users select options that align with their nutritional 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 3.7: Rules for: meal categories

These predicates detect meals that include ingredients associated with gluten or lactose intolerance. They utilize findall to gather the relevant meal-course pairs and subsequently remove any duplicates. Such checks are essential for ensuring that meals comply with specific dietary restrictions and facilitate the creation of customized meal plans for individuals with food intolerances.

```
% 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 3.8: Rules for: meal allergies

Description:

- **meal_calories** calculates the overall caloric value of a meal by adding up the calories of its individual ingredients.
- **calorie_conscious_levels** classifies meals into different categories according to their total caloric content.

These computations play a key role in dietary management, offering clear insights into the energy value of meals and supporting users in making informed choices that align with their caloric requirements.

```
% Calculate the total caloric content of a meal
meal_calories(Meal, Course, TotalCalories) :-
   meal(Meal, Course, Ingredients),
   findall(Calories,
      (member(Ingredient, Ingredients),
      kcal_ingredient(Ingredient, Kcal),
      Calories is Kcal),
   CaloriesList),
   sum_list(CaloriesList, TotalCalories).

% 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 > 650 -> HighestLevel = 0;
   TotalCalories =< 250 -> HighestLevel = 3;
   TotalCalories =< 450 -> HighestLevel = 2;
   TotalCalories =< 650 -> HighestLevel = 1
   ),
% Generate all levels up to the highest applicable level
   findall(Level, (between(0, HighestLevel, Level)), Levels).
```

Listing 3.9: Rules to compute the total kcals of a meal, and calorie-conscious levels

The **guest_preferences** predicate filters meals according to category (carnivorous, vegetarian, or omnivore), calorie level, and potential allergens. By combining these conditions, it refines the selection process to generate more accurate recommendations. This approach ensures that meal suggestions are tailored to individual dietary needs and restrictions, supporting truly personalized meal planning.

```
% 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 3.10: Rule for: guest preferences

This Prolog implementation provides a structured system for managing meal ingredients, evaluating their nutritional values, and adapting to both dietary preferences and restrictions. The design of the predicates ensures flexibility and reliability when generating meal recommendations across different criteria.

Prolog Testing

To demonstrate its functionality, we include a set of test queries focusing on the guest_preferences predicate. In this example scenario, if the user is vegetarian and shows a moderate to high level of calorie awareness

Figure 3.9: Prolog Testing (1)

and has no allergies, the system returns all the dishes and the relative courses. In the other case described below, if the user is carnivorous, has a moderate to low calorie awareness per meal, and it has a lactose intolerance, the following meals are returned: 'affettati misti', 'spaghetti aglio olio e peperoncino', 'tagliatelle alla marinara', 'grigliata di magliale mista', 'water'. Here, we provide example queries to run in the simulator described before for other results.

```
% guest_preferences(vegetarian, 3, none, Meal, second_course).

% guest_preferences(carnivorous, 1, none, Meal, Course).

% guest_preferences(omnivore, 0, gluten, Meal, Course).
```

Listing 3.11: Prolog testing queries

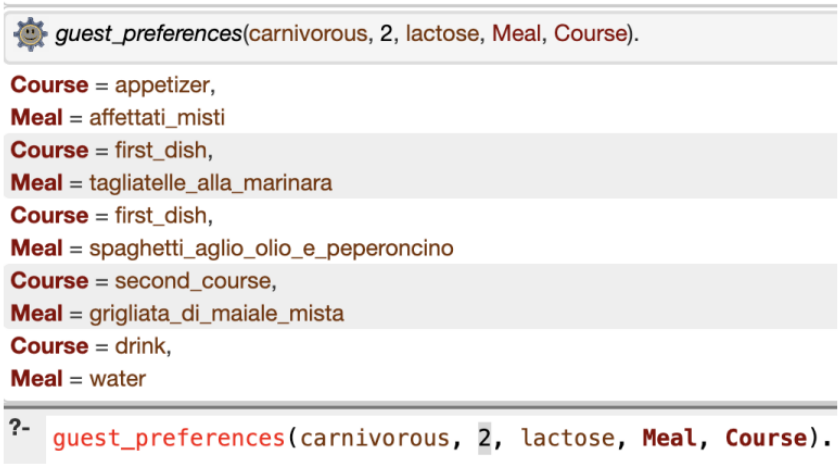


Figure 3.10: Prolog Testing (2)

3.3 Ontology Engineering

In Computer Science, ontology engineering is a discipline that focuses on methodologies for building ontologies. This involves the formal representation and definition of categories, properties, and relationships among concepts, data, and entities. Ontologies can be modeled using frameworks such as RDF, OWL, Neo4J, and others. For the construction of our ontology, we adopted RDF.

RDF

The Resource Description Framework (**RDF**) is a standardized model for representing and exchanging graph data in a general way. RDF allows for the description of resources that form the foundation of knowledge graphs. A **knowledge graph** represents a network of real-world entities—such as objects, events, situations, or concepts—while emphasizing the relationships between them.

Protégé

Protégé is a free, open-source ontology editor and knowledge management system. It supports the use of the Semantic Web Rule Language (**SWRL**), which enables the definition of inference rules to derive new knowledge from existing ontologies. Moreover, Protégé integrates the SPARQL Protocol and RDF Query Language (**SPARQL**), a semantic query language designed to retrieve and manipulate RDF data.

3.3.1 Our Ontology

Using Protégé, we developed our ontology by incorporating the core entities of our domain. Compared to the initial submission in June, we introduced several improvements:

Multiple SPARQL queries were added and described in detail within the report.

The SHACL file was revised to remove redundancies and refined so that constraints appear natural and consistent.

For ingredients, we introduced the Object Property foodHasCategory, which replaces the previous mealHasCategory property. This adjustment makes the property applicable

not only at the subclass level of Meal but also at the subclass level of Ingredient.

Regarding SWRL, we applied the necessary modifications and added two new Object Properties (mealNotContainsGlutenIntolerance and mealNotContainsLactoseIntolerance). These allow us to infer which meals are suitable for users based on their allergy type, dietary category, calorie-consciousness, and course preference.

Classes

The main entities identified earlier also serve as the primary classes of our ontology. In addition, we established a hierarchy that will support future extensions, such as the inclusion of new resources—potential allergens, types of ingredients, and other relevant elements.

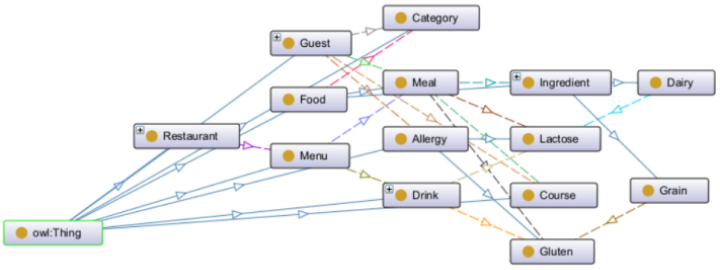


Figure 3.11: Our ontology graph in Protègè

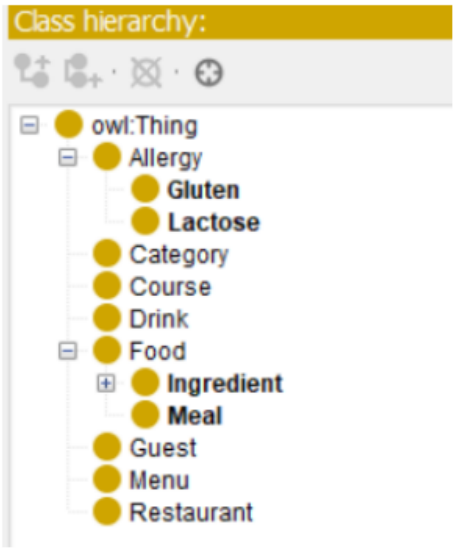


Figure 3.12: Our ontology model in Protègè

Object Properties

Object properties are used to represent relationships between classes. In our ontology, we defined the following:

- **restaurant hasMenu:** associates a menu with a restaurant

- **restaurant hasGuest:** links a guest to a restaurant
- **menu containsDrink:** associates a drink with a menu
- **menu containsMeal:** associates a meal with a menu
- **guest hasAllergies:** represents a guest who has an allergy
- **guest hasCategory:** links a guest to a category (e.g., carnivorous, vegetarian)
- **guest hasPreferenceCourse:** associates a guest with a preferred course
- **guest isAtRiskForFood:** a SWRL rule inferred by the reasoner, used to detect potential food-related risks due to allergies
- **food hasCategory:** associates food (ingredient or meal) with a category
- **meal containsGlutenIntolerance:** SWRL rule inferred by the reasoner, identifies if a meal contains gluten-based ingredients (e.g., grains)
- **meal containsLactoseIntolerance:** SWRL rule inferred by the reasoner, identifies if a meal contains lactose-based ingredients (e.g., dairy)
- **meal notContainsGlutenIntolerance:** added to check whether a meal is gluten-free
- **meal notContainsLactoseIntolerance:** added to check whether a meal is lactose-free
- **meal hasCourse:** associates a course with a meal
- **meal hasIngredient:** links an ingredient to a meal
- **drink hasIngredient:** links an ingredient to a drink
- **drink containsGlutenIntolerance:** indicates drinks containing gluten
- **drink containsLactoseIntolerance:** indicates drinks containing lactose
- **dairy containsLactoseIntolerance:** links the dairy class with lactose intolerance
- **grain containsGlutenIntolerance:** links the grain class with gluten intolerance

Data Properties

Data properties describe attributes of classes. The following were defined:

- **thing hasName:** attribute shared by all subclasses of Thing; it is a string that specifies the object's name
- **food hasKcal:** attribute for subclasses of Food (Ingredient and Meal); it is an integer representing the caloric content of the food

- **guest hasLevelOfCalorieConscious:** attribute of the Guest class; it is an integer ranging from 0 to 3 that represents a guest's level of calorie-consciousness

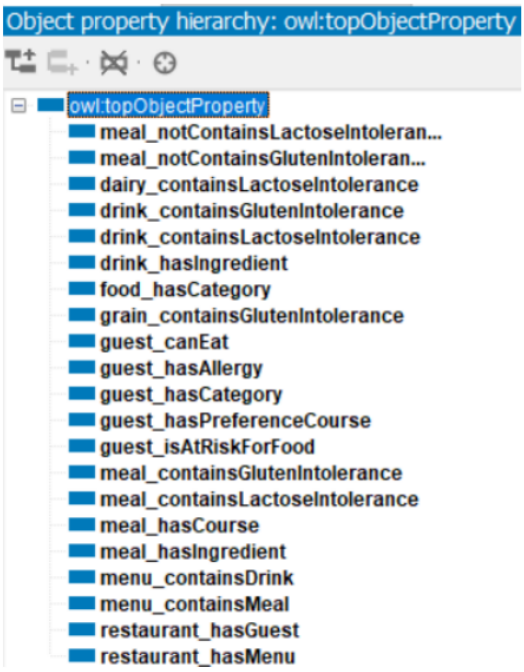


Figure 3.13: Our model's object's properties

- **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 3;

- **drink hasKcal:** this is an attribute that the class Drink possesses. It is an integer used to identify the kcal of the drink;

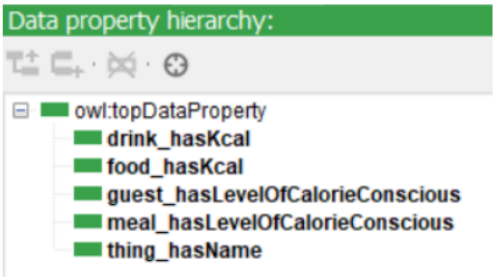


Figure 3.14: Our model's data properties

Individuals

Given the large number of individuals, we will only specify a few examples.

Figure 3.15: Some individuals in our model

For the class Restaurant, there is only one instance: restaurant our-restaurant. It is as follows:

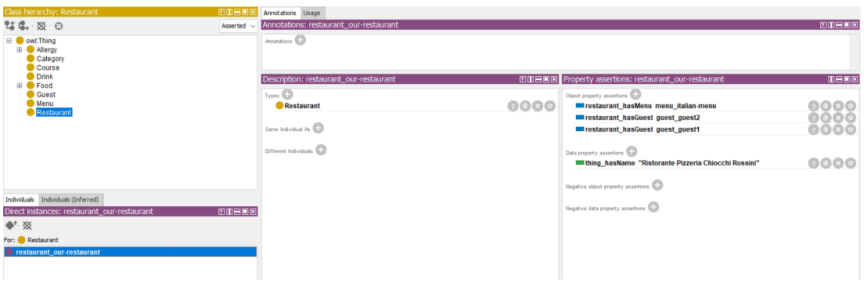


Figure 3.16: Restaurant individual in our model

As can be seen, it has the following characteristics: it includes a menu and two guests, and it has a name.

For the Menu class, there is a single instance that includes both drinks and meals:

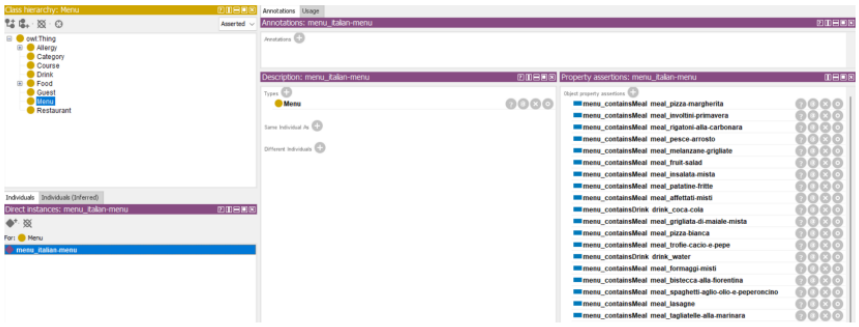


Figure 3.17: Individual 'Menu' in our model

Regarding the Guest class, we have two individuals. The following figure depicts guest1:

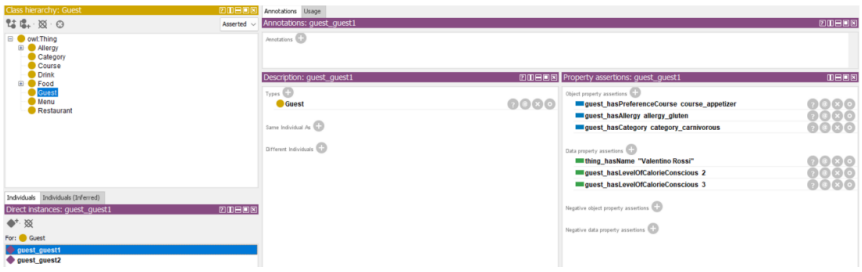


Figure 3.18: Individual 'Guest1' in our model

For the Meal class, there are 17 individuals. Below is an example of one individual: rigatoni alla carbonara.

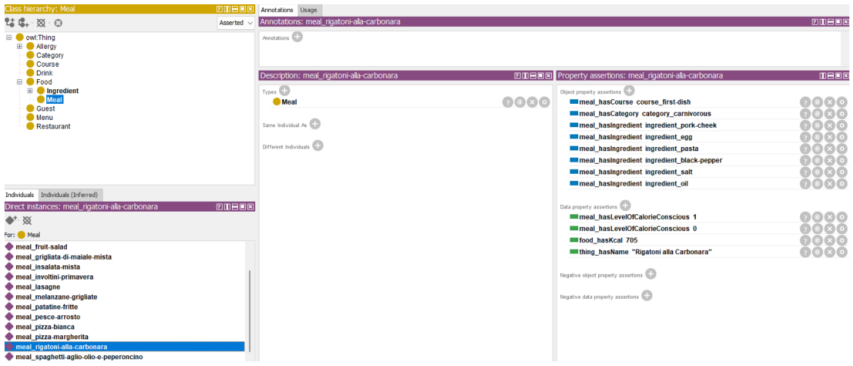


Figure 3.19: Individual 'Rigatoni alla Carbonara' in our model

3.3.2 SWRL Rules

We leveraged the Protégé reasoner to extend the ontology's inference capabilities by defining the following rules:

GuestIsAtRiskForFoodGluten

$$Guest(?\,guest) \wedge guesthasAllergy(?\,guest, ?\,allergy) \wedge Meal(?\,meal) \\ \wedge mealcontainsGlutenIntolerance(?\,meal, ?\,allergy) \\ \rightarrow guestisAtRiskForFood(?\,guest, ?\,meal)$$

This rule is designed to infer meals that pose a risk to the user based on whether the guest has an allergy, in this case gluten intolerance, or not, and to display them accordingly.

GuestIsAtRiskForFoodLactose

$$Guest(?\,guest) \wedge guesthasAllergy(?\,guest, ?\,allergy) \wedge Meal(?\,meal) \\ \wedge mealcontainsLactoseIntolerance(?\,meal, ?\,allergy) \\ \rightarrow guestisAtRiskForFood(?\,guest, ?\,meal)$$

This rule is designed to infer meals that pose a risk to the user based on whether the guest has an allergy, in this case lactose intolerance, or not, and to display them accordingly.

MealContainsGlutenIntolerance

$$Meal(?\,meal) \wedge mealhasIngredient(?\,meal, ?\,ingredient) \wedge Ingredient(?\,ingredient) \\ \wedge graincontainsGlutenIntolerance(?\,ingredient, ?\,gluten) \\ \rightarrow mealcontainsGlutenIntolerance(?\,meal, ?\,gluten)$$

This rule is used to infer meals that contain ingredients which include a gluten intolerance. Therefore, the reasoning is: if the meal contains an ingredient that has a gluten intolerance, then the meal itself has a gluten intolerance.

MealContainsLactoseIntolerance

$$Meal(?\,meal) \wedge mealhasIngredient(?\,meal, ?\,ingredient) \wedge Ingredient(?\,ingredient) \\ \wedge dairycontainsLactoseIntolerance(?\,ingredient, ?\,gluten) \\ \rightarrow mealcontainsLactoseIntolerance(?\,meal, ?\,gluten)$$

This rule is used to infer meals that contain ingredients which include a lactose intolerance. Therefore, the reasoning is: if the meal contains an ingredient that has a lactose intolerance, then the meal itself has a lactose intolerance.

Rule for inferring meals suitable for guests with gluten intolerance
This rule identifies which meals a guest can safely consume, taking into account both their dietary preferences and the presence of 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)
```

Figure 3.20: Rule to infere meals to eat - gluten intolerance

Rule for inferring meals suitable for guests with lactose intolerance
Similarly, this rule determines the meals a guest can eat based on their preferences and whether they are affected by 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)
```

Figure 3.21: Rule to infere meals to eat - lactose intolerance

Unfortunately, it was not possible to extend reasoning to the data properties guest hasLevelOfCalorieConscious and meal hasLevelOfCalorieConscious using the HermiT Reasoner (v1.4.3.456), as it does not support SWRL built-in atoms. We attempted to use the Pellet (incremental) Reasoner as an alternative; however, its performance proved inconsistent, sometimes producing correct results and at other times leading to errors. As a result, these fields were left without inference support, and we adopted a simpler yet still functional approach.Finally, the absence of negation support in SWRL affected the design of our model. To address this limitation, we introduced two additional object properties: meal notContainsGlutenIntolerance and meal notContainsLactoseIntolerance.

SWRL Testing

Below are examples of SWRL rule testing executed using Reasoner HermiT 1.4.3.456:

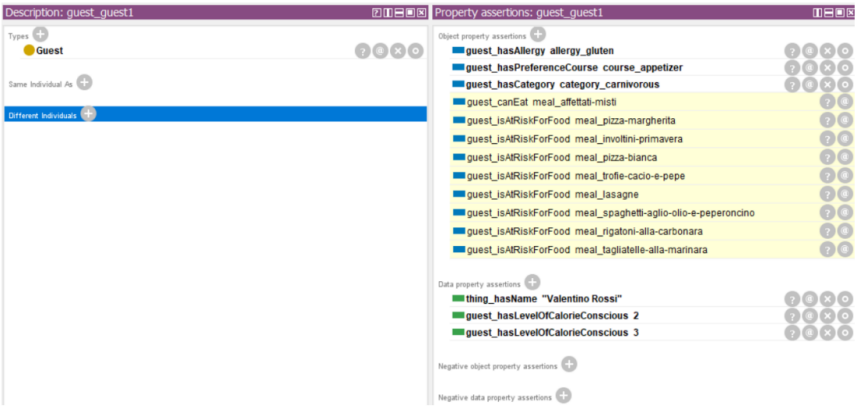


Figure 3.22: SWRL Testing (1) - 'Guest1' meals suggestion

As depicted in the first figure (Figure 3.22), the high-risk dishes for the user are immediately inferred, and the food that guest1 can safely consume (affettati misti) is correctly identified.

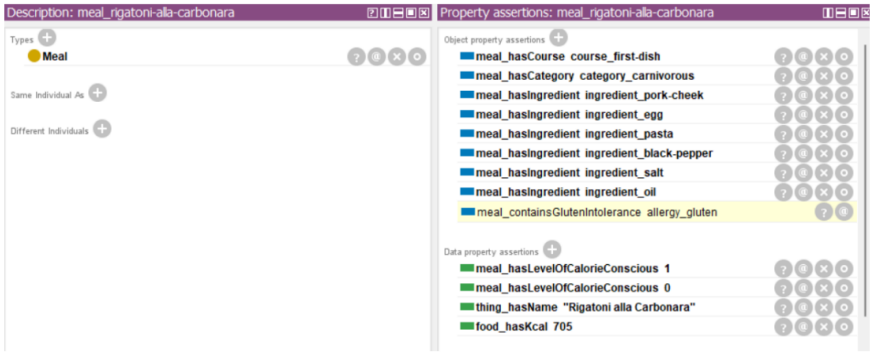


Figure 3.23: SWRL Testing (2) - 'Meal: Rigatoni all Carbonara'

In this case, the inference is based on whether the meal contains an "allergenic" ingredient or not. Below is the recommended meal inference for the guest based on their preferences: in this case, if the guest is omnivorous, highly calorie-conscious, has a gluten intolerance, and prefers the second course, roasted fish is recommended.

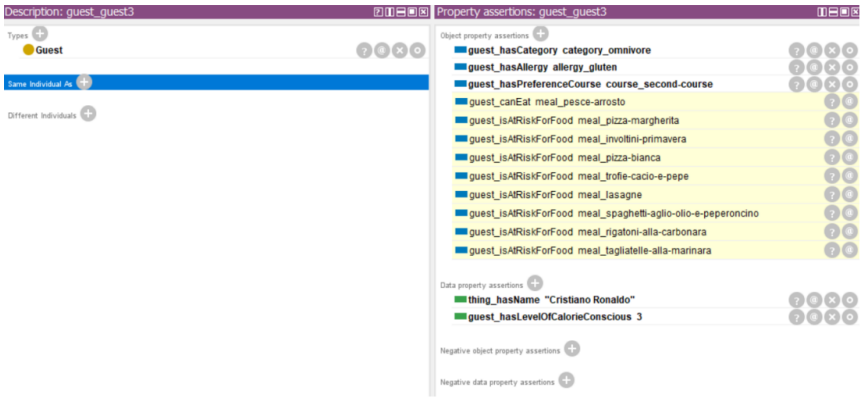


Figure 3.24: SWRL Testing (3) - 'Guest3' meals suggestion

3.3.3 SPARQL Query

Regarding the SPARQL queries, we designed a set of queries to filter and retrieve specific information, and tested them on both GraphDB and Protégé.

SPARQL (SPARQL Protocol and RDF Query Language) is a powerful language and protocol for querying and manipulating data stored in RDF format. It enables users to extract information from RDF graphs, perform complex searches, and integrate data from heterogeneous sources. By allowing precise data retrieval, SPARQL plays a key role in working with semantic web data and linked data applications.

In the following subsection, we present some of the queries contained in the file queries_sparql.txt[6], which collects all the queries we developed.

SPARQL Queries Examples

Below is the prefix to be added for the correct execution of the queries.

[6] *SPARQL example queries:* URL: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project/blob/main/Task1/Ontology/sparql_queries.txt

```
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/simone.micarelli-davide.nappo/kebi2025#>
```

Listing 3.12: Prefixes for all the SPARQL Queries

- **Query for Carnivorous:** The following SPARQL query is designed to retrieve all instances of meals that belong to the category 'carnivorous' from the dataset.

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

Listing 3.13: Query to retrieve all meals labeled as carnivorous

Executing this query on both Protege and GraphDB yields the results shown in the figures.

- **Query for retrieve all the ingredients in a meal:** This query is used to see all the ingredients in a specific meal, in this case 'pizza margherita'.

```
SELECT ?ingredient
WHERE {
   kebi:meal_pizza-margherita kebi:meal_hasIngredient ?ingredient .
}
```

Listing 3.14: Query to retrieve all the ingredients in a specific meal

Results:

Snap 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/simone.micarelli-davide.nappo/kebi2025#>

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

Execute

?meal
kebi:meal_pizza-margherita
kebi:meal_pizza-bianca
kebi:meal_lasagne
kebi:meal_tagliatelle-alla-marinara
kebi:meal_formaggi-misti
kebi:meal_grigliata-di-maiale-mista
kebi:meal_patatine-fritte
kebi:meal_trofie-cacio-e-pepe
kebi:meal_spaghetti-aglio-olio-e-peperoncino
kebi:meal_rigatoni-alla-carbonara
kebi:meal_pesce-arrosto
kebi:meal_affettati-misti
kebi:meal_bistecca-alla-fiorentina

13 results

Figure 3.25: SPARQL Query with Protègè: Carnivorous

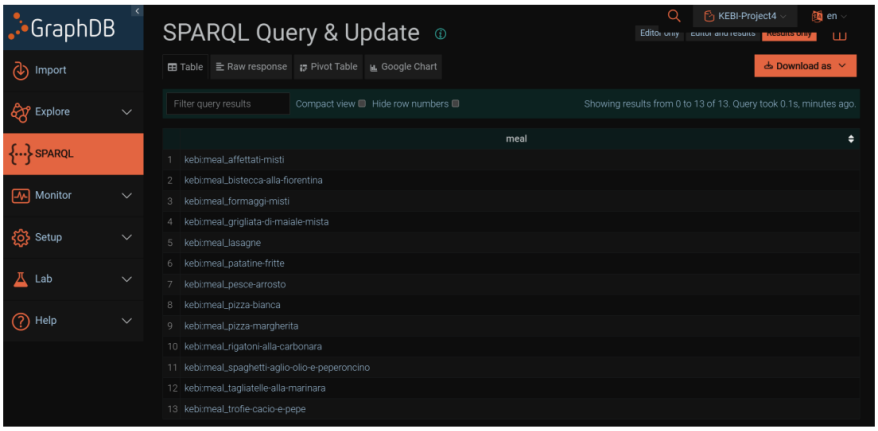


Figure 3.26: SPARQL Query with GraphDB: Carnivorous

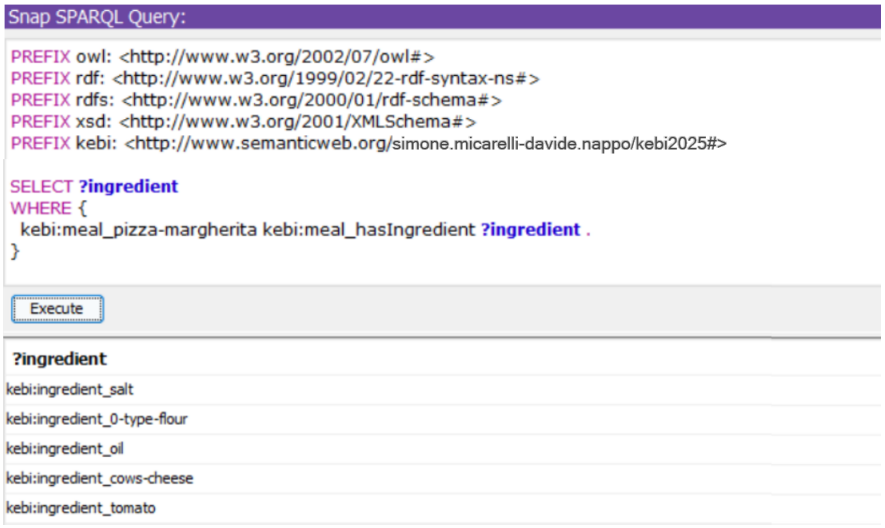


Figure 3.27: SPARQL Query with Protègè: Ingredients in a meal

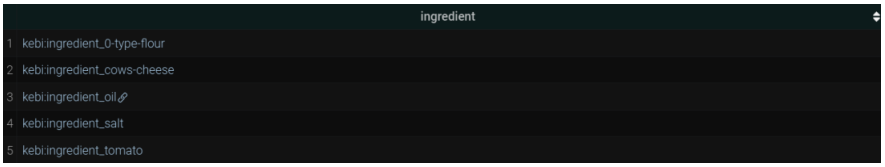


Figure 3.28: SPARQL Query with GraphDB: Ingredients in a meal

- **Suggested Meals:** Advanced query to display meals based on the preferences of the guest, in this case 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 }

}
```

Listing 3.15: Advanced query to display meals based on the preferences of the guest

'Guest1' properties:

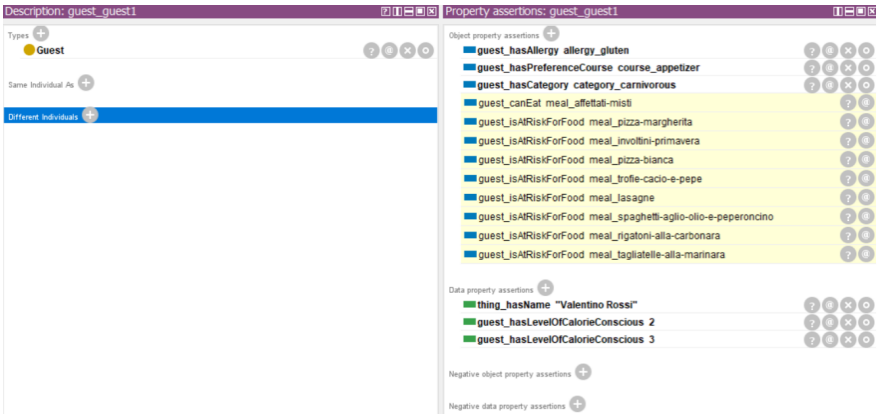


Figure 3.29: 'Guest1' properties

Results:

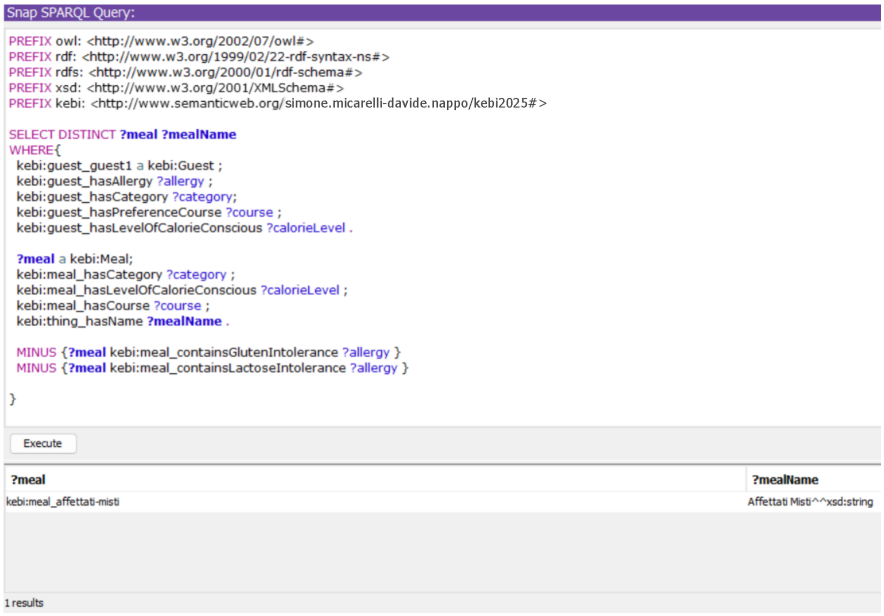


Figure 3.30: SPARQL Query with Protègè: 'Guest1' meals suggestion

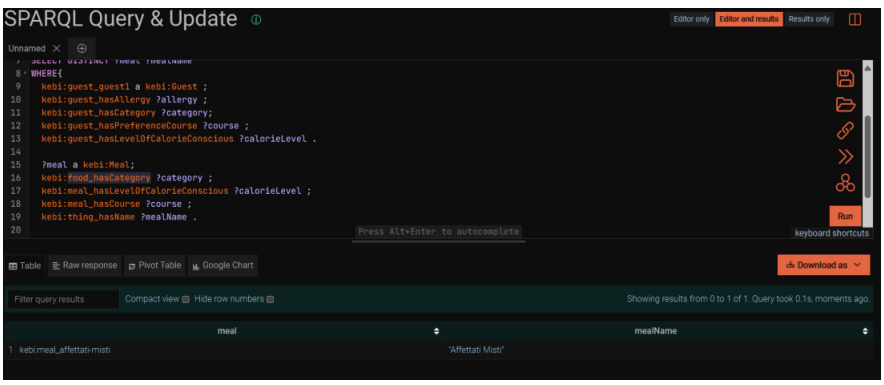


Figure 3.31: SPARQL Query with GraphDB: 'Guest1' meals suggestion

The remaining queries will be listed, but their results will not be provided due to the large number of queries.

- **Query to return vegetarian meals**

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

Listing 3.16: Vegetarian Meals

- **Query to view meals and their related calories**

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

Listing 3.17: Meals and related calories

- **Query to view the guests with allergies**

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

Listing 3.18: Guests with allergies

- **Query to view the ingredients kcals**

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

Listing 3.19: Kcal of all ingredients

- **Query to view the level 3 'calorie-conscious' meals**

```
SELECT DISTINCT ?meal ?calorie
WHERE {
   ?meal rdf:type ?type .
   ?meal kebi:meal_hasLevelOfCalorieConscious ?calorie .
   FILTER(?calorie = 3)
}
```

Listing 3.20: Meals with level 3 of Calorie Conscious

- **Query to view the kcals of a specific ingredient**

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

Listing 3.21: Kcal for a specific ingredient

- **Query to view ingredients associated with gluten intolerance**

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

Listing 3.22: Gluten intolerance: ingredients

- **Query to view ingredients associated with lactose intolerance**

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

Listing 3.23: Lactose intolerance: ingredients

- **Query to view the meals associated with gluten intolerance:** only with PROTEGE -Reasoner

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

Listing 3.24: Gluten intolerance: meals - REASONER Protègè

- **Query to view the meals associated with lactose intolerance:** only with PROTEGE -Reasoner

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

Listing 3.25: Lactose intolerance: meals - REASONER Protègè

- **Query to view the meals classified as 'Appetizers'**

```
SELECT ?meal
WHERE {
   ?meal rdf:type kebi:Meal .
   ?meal kebi:meal_hasCourse kebi:course_appetizer .
}
```

Listing 3.26: Meals classified as Appetizer

- **Query to view the meals classified as 'First Dish'**

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

Listing 3.27: Meals classified as First Dish

- **Query to view the ingredients classified as 'Seafood'**

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

Listing 3.28: Ingredient that are classified as Seafood

- **Query to view the meals with more than 500 kcals**

```
SELECT ?meal ?kcal
WHERE {
   ?meal kebi:food_hasKcal ?kcal.
   FILTER(xsd:decimal(?kcal) > 500).
}
```

Listing 3.29: Meals that have more than 500 kcal

3.3.4 SHACL Shape

SHACL (Shapes Constraint Language) is used to validate RDF (Resource Description Framework) data against predefined conditions, called shapes. These shapes describe the expected structure and content of RDF graphs, helping to ensure consistency and integrity within semantic web applications. SHACL can define constraints on node types, property cardinalities, and even allowable property values. By enforcing these rules, SHACL supports high data quality, reliable integration, and effective querying.

The SHACL Shape file, located in the ontology folder and named shacl_menu[7], includes the following:

- **RestaurantShape:** Requires each Restaurant instance to contain at least one Menu (MenuShape). We did not impose a maximum count, giving restaurant managers the flexibility to add more menus in the future. It also allows optional links to Guests (GuestShape), who are associated with the restaurant after scanning a QR code.

```
kebi2025:RestaurantShape a sh:NodeShape ;
   sh:targetClass kebi2025:Restaurant ;
   sh:property [
      sh:path kebi2025:restaurant_hasMenu ;
      sh:node kebi2025:MenuShape ;
      sh:minCount 1 ;
```

[7] *SHACL Shape File:* URL: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project/blob/main/Task1/Ontology/shacl_menu.txt

```
7        sh:message "It␣is␣impossible␣that␣a␣Restaurant␣does␣not␣have␣a␣menu!!" ;
8     ];
9     sh:property [
10       sh:path kebi2025:restaurant_hasGuest ;
11       sh:node kebi2025:GuestShape ;
12       sh:minCount 0 ;
13    ].
```

Listing 3.30: 'Restaurant' SHACL Shape

"If we delete the menu, for example, the following error will be returned: "It is impossible that a Restaurant does not have a menu".

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

```
1  kebi2025:MenuShape a sh:NodeShape ;
2     sh:targetClass kebi2025:Menu ;
3     sh:property [
4        sh:path kebi2025:menu_containsMeal ;
5        sh:node kebi2025:MealShape ;
6        sh:minCount 1 ;
7     ];
8     sh:property [
9        sh:path kebi2025:menu_containsDrink ;
10       sh:node kebi2025:DrinkShape ;
11       sh:minCount 1 ;
12    ].
```

Listing 3.31: 'Menu' SHACL Shape

Similarly, if the menu does not include any food or drinks, an error will occur.

- **GuestShape:**Validates instances of Guest, enforcing mandatory attributes such as dietary category (CategoryShape) and calorie-consciousness level, while allowing optional details like allergies (AllergyShape). The calorie-consciousness level is defined within a range from 1 to 4, since a value of 0 implicitly encompasses levels 1, 2, and 3. However, we were unable to employ SWRL due to its inference limitations when executed with a Reasoner, as it operates exclusively within Protégé. Consequently, on external platforms such as GraphDB or AOAME, certain data may not be preserved.

```
1  kebi2025:GuestShape a sh:NodeShape ;
2     sh:targetClass kebi2025:Guest ;
3     sh:property [
4        sh:path kebi2025:guest_hasAllergy ;
5        sh:node kebi2025:AllergyShape ;
6        sh:minCount 0 ;
7     ];
8     sh:property [
9        sh:path kebi2025:guest_hasCategory ;
10       sh:node kebi2025:CategoryShape ;
11       sh:minCount 1 ;
12    ];
13    sh:property [
14       sh:path kebi2025:guest_isAtRiskForFood ;
15       sh:node kebi2025:MealShape ;
16    ];
17    sh:property [
18       sh:path kebi2025:guest_hasLevelOfCalorieConscious ;
19       sh:datatype xsd:integer ;
20       sh:minCount 1 ;
```

```
        sh:maxCount 4 ;
    ].
```

Listing 3.32: 'Guest' SHACL Shape

A guest may optionally be associated with an allergy. In our implementation, we have defined only two types of allergies, though others (e.g., nut allergies) could also be considered. To ensure extensibility, no maxCount has been specified, leaving room for the addition of new types of allergies in the future.

- **MealShape:**Specifies constraints for instances of Meal, including optional intolerance to lactose (LactoseShape) and gluten (GlutenShape), categories, courses, and ingredients.

```
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 ;
    ].
```

Listing 3.33: 'Meal' SHACL Shape

A Meal may include lactose or gluten and can belong to one or more categories. For instance, a dish might be suitable for both vegetarian and non-vegetarian guests, and if it also fits vegan requirements, it would be included in that category as well. For this reason, no maxCount has been defined. In addition, every meal must be associated with a course, and it must contain at least one ingredient to be considered complete.

Validation

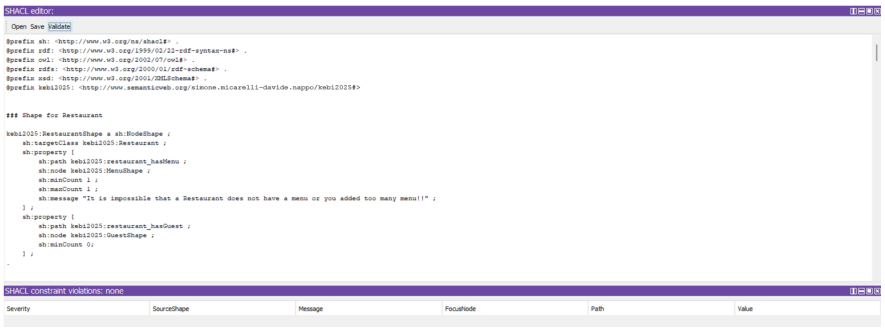


Figure 3.32: SHACL validator

As we can see at the end of image, no violations were found.

4. Agile and Ontology-based Meta-Modelling

In Section 4.1, we will describe what AOAME is, while in Section 4.2, we will outline what BPMN 2.0 entails and present our model. Here you can find the link to our project for Task 2[1].

4.1 AOAME

AOAME[2] is a prototype tool that applies an agile, ontology-based approach to meta-modeling. Its goal is to design and maintain schemas for Enterprise Knowledge Graphs (EKGs), which structure knowledge within a specific domain to support analysis, reasoning, and integration of heterogeneous data sources. A key challenge in this process lies in the need for expertise in both ontology engineering and the application domain.

The proposed approach combines traditional meta-modeling with agile principles, enabling domain-specific adaptations of modeling languages and real-time validation. This integration fosters the active involvement of domain experts in the engineering process. Developed according to the Design Science Research methodology, AOAME serves as a prototype to evaluate this approach. It offers features for extending, modifying, or removing modeling constructs, as well as hiding them from the tool palette. These operations are powered by meta-modeling operators that generate SPARQL queries and update the triplestore, ensuring alignment between human-readable and machine-readable knowledge.

The architecture of AOAME leverages Java Fuseki libraries to implement APIs that retrieve ontologies from the triplestore, display them in a graphical interface, and execute meta-modeling operators. Its effectiveness has been demonstrated through real-world implementations and case studies, confirming its applicability across diverse domains. By facilitating collaboration between domain experts and language engineers, AOAME enhances the adaptability and efficiency of domain-specific modeling language (DSML) engineering.

4.2 BPMN 2.0

Business Process Model and Notation (BPMN) is a graphical standard used to model business processes within a workflow. It provides a uniform way to visualize the sequence

[1] *Github Forked Project link:* URL: https://github.com/Elmicass/Ontology4ModelingEnvironment

[2] *AOAME Reference:* URL: https://emisa-journal.org/emisa/article/view/310

of activities and the exchange of information, ensuring clarity for all stakeholders, including business analysts, process participants, and technical developers.

The main elements of BPMN 2.0 include:

- Events: Occurrences that influence the process flow (e.g., start, intermediate, and end events).
- Activities: Tasks or units of work to be performed (e.g., user tasks, service tasks, manual tasks).
- Gateways: Decision points that govern the divergence and convergence of process paths (e.g., exclusive, parallel, inclusive gateways).
- Pools and Lanes: Represent key participants; pools correspond to organizations or entities, while lanes represent subdivisions within them.
- Artifacts: Supplementary elements, such as data objects, groups, and annotations, that provide additional context.

In this project, BPMN is applied to model the restaurant's customer ordering process. The diagram explicitly incorporates both allergies and dietary preferences. Customers are asked to declare any allergies and specify whether they are vegetarian, vegan, or non-vegetarian. Based on this information, they can then select the type of meals they wish to order. This ensures that the ordering process is personalized, improving both customer satisfaction and food safety.

4.2.1 Our BPMN 2.0 Model

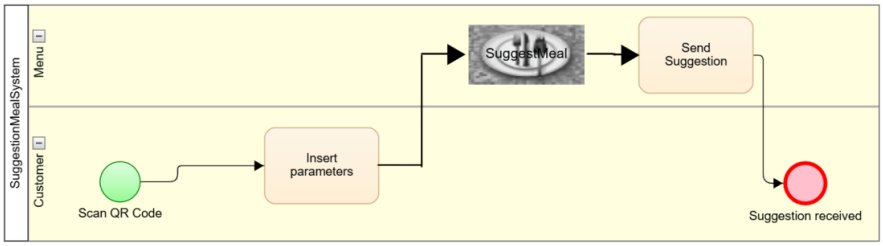


Figure 4.1: BPMN 2.0 graph built in AOAME

As shown in our work, we introduced specific elements to complete the BPMN diagram. In particular, we used a pool element to represent the interaction between the menu and the customer throughout the process. Concretely, the customer scans a QR code to access the menu, then inputs parameters such as allergies and dietary category. Based on this information, the system suggests suitable meal options.

In carrying out this task, we had the opportunity to work with AOAME (Agile and Ontology-Aided Modeling Environment), a methodology and toolset designed to improve software development and business process modeling by integrating agile principles with ontology-based techniques.

After designing the BPMN 2.0 model, we employed Apache Jena Fuseki to execute queries on the resulting graph. Jena Fuseki is a robust and scalable SPARQL server for efficiently managing RDF data. It provides a RESTful interface that supports:

- Executing SPARQL queries to retrieve data.
- Performing SPARQL updates to modify data.
- Loading, storing, and managing RDF datasets.

```
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/simone.micarelli-davide.nappo/kebi2025#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?property ?value
WHERE {
  mod:SuggestMeal_b8f9fb3d-e3c6-4224-9b85-057a7e4c59a2 ?property ?value .
}
```

Figure 4.2: Query 1: used for see the property and their relative value of our extended class SuggestMeal.

```
SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac 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 = "Appetizer", kebi:course_appetizer,
          IF(?course = "First Dish", kebi:course_first-dish,
             IF(?course = "Main Dish", kebi:course_main-dish,
                IF(?course = "Second Course", kebi:course_second-course,
                   IF(?course = "Sidedish", 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 4.3: Query 2: used for analyze the extended class and return the meals which satisfied all data property described.

The first query retrieves the properties and their relative value of SuggestMeal class, while the second one is used to analyze the graph and it provides the meal suggestions that satisfy the customer's parameters.

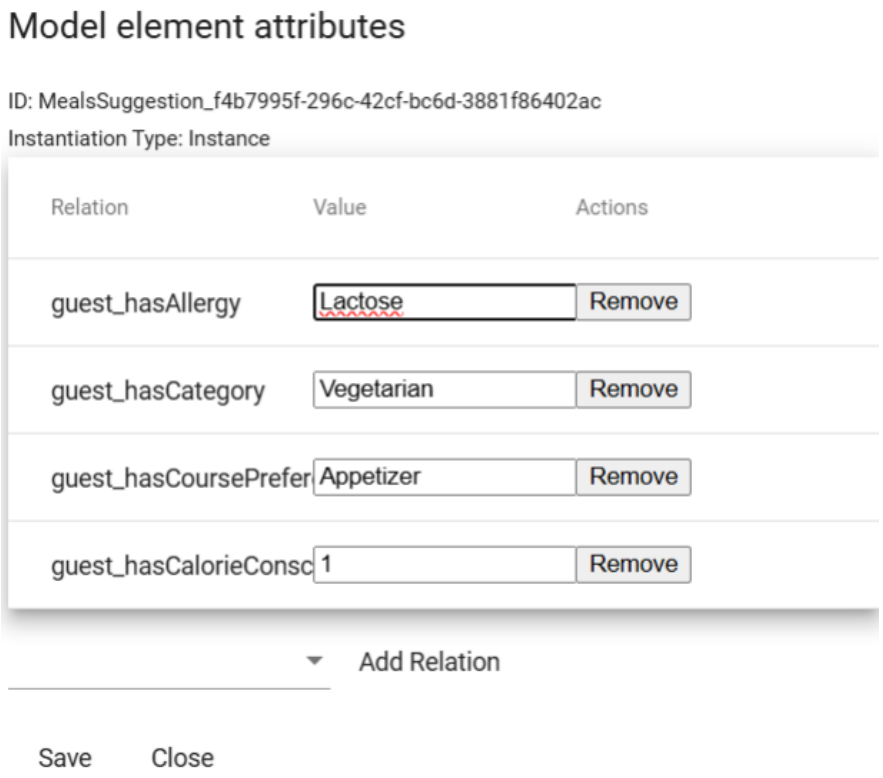


Figure 4.4: Define the attributes of our extended class. We set up the lactose allergy with vegetarian category.

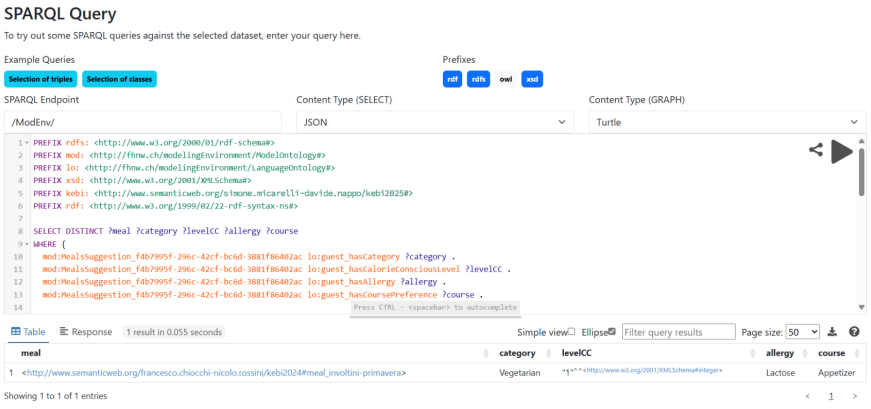


Figure 4.5: Result of query 2 with parameter of figure 4.4

For the first example, as we can see in the image above, we set up the attributes of our BPMN class and we fire the query in Jena Fuseki that gives back the result.

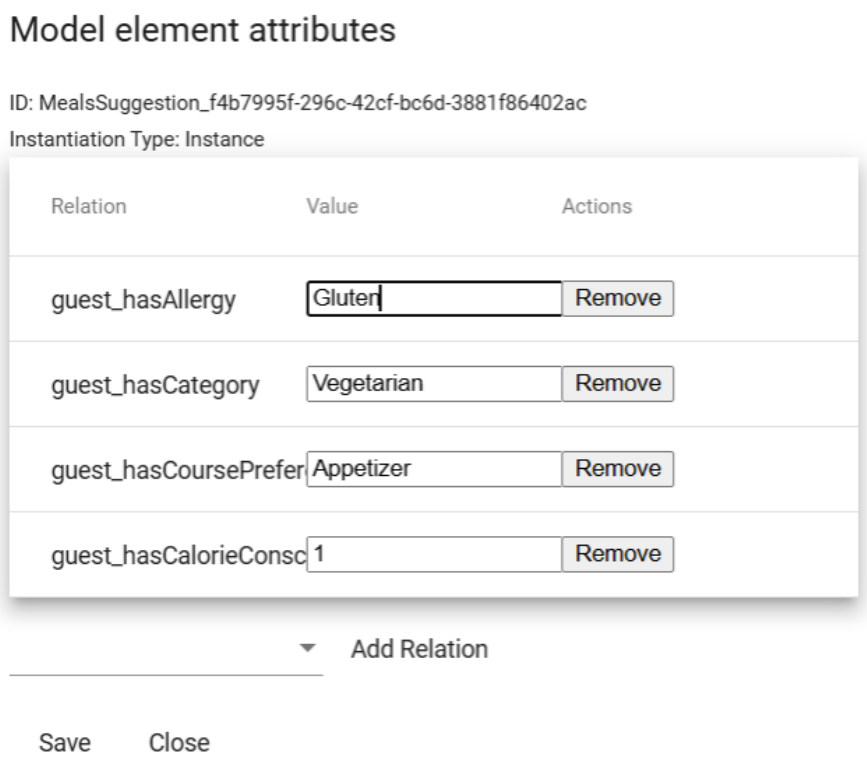


Figure 4.6: Define the attributes of our extended class. We set up the gluten allergy with vegetarian category.

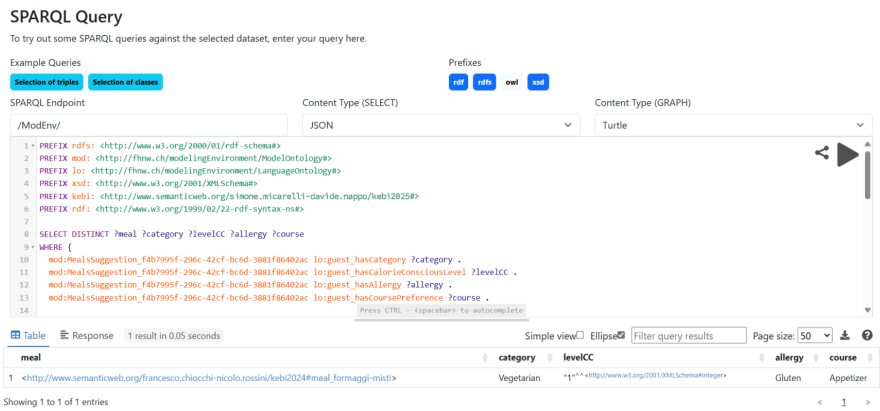


Figure 4.7: Result of query 2 with parameter of figure 4.6

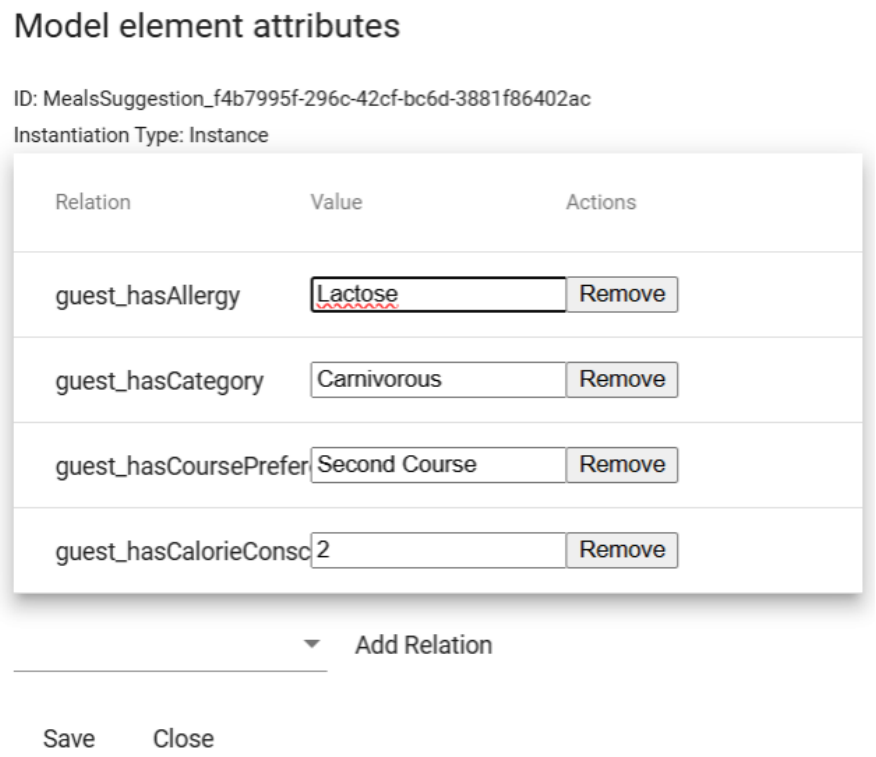


Figure 4.8: Define the attributes of our extended class. We set up the lactose allergy with carnivorous category.

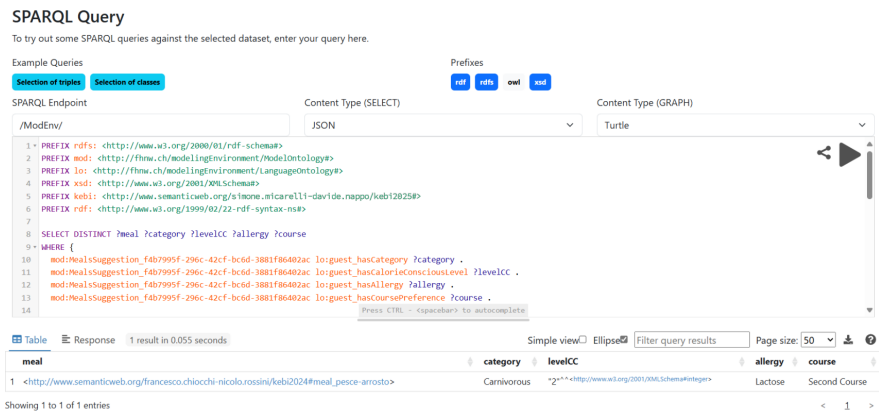


Figure 4.9: Result of query 2 with parameter of figure 4.8

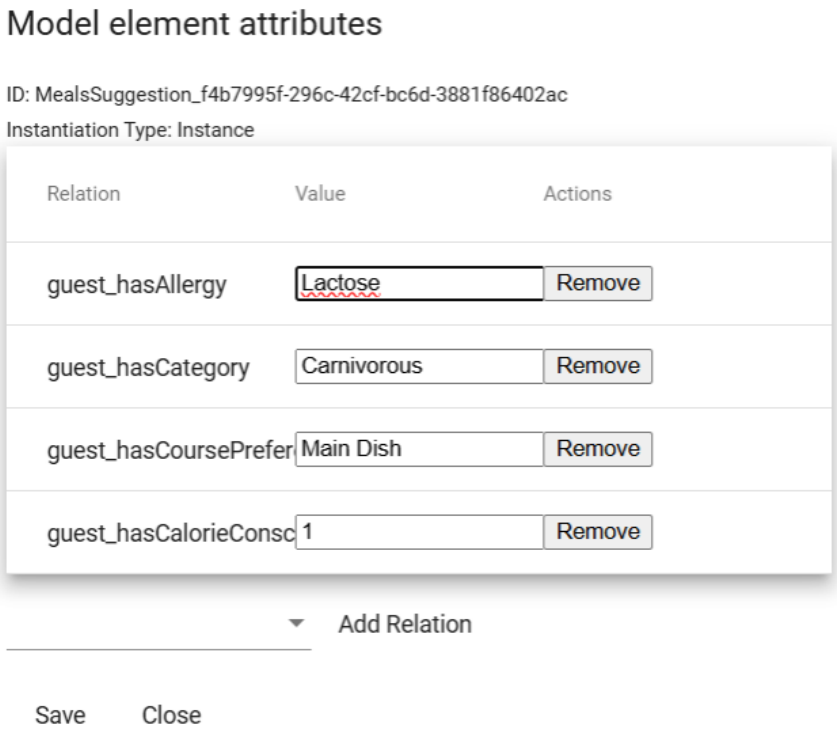


Figure 4.10: Define the attributes of our extended class. We set up the lactose allergy with carnivorous category and main dish.

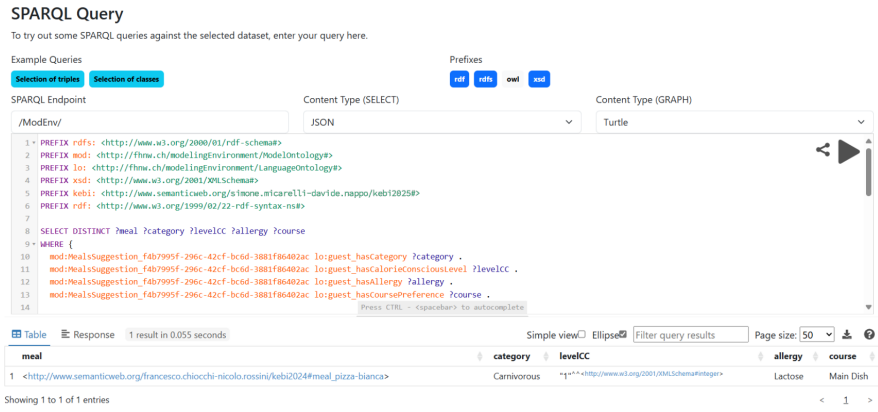


Figure 4.11: Result of query 2 with parameter of figure 4.10

5. Conclusions

5.1 Davide Nappo

During this project, my colleague and I explored a range of knowledge-based solutions, each presenting distinct advantages and limitations. Below, I reflect on my impressions of these approaches, highlighting both strengths and weaknesses.

- **Decision Tables**: Decision tables operate in a straightforward manner: inputs are matched with outputs according to predefined rules structured in rows. Output selection follows a Hit Policy, which dictates how results are grouped. In our project, we applied the Collect policy, allowing all rules to be evaluated concurrently and fully leveraging Decision Model and Notation (DMN). Under this logic, input conditions are connected by an "and" relationship within a column, while rows interact with each other through an "or" relationship. Outputs are aggregated into sets for each output column. This approach is intuitive and easy to implement. However, in contexts involving numerous variables or complex conditions, decision tables quickly become impractical and overly complicated. Data manipulation proved cumbersome, repetitive, and time-consuming, with the manual creation of entries turning monotonous. Debugging was also challenging, made worse by limited online tool support.

- **Prolog**: Working with Prolog was one of the most engaging parts of the project. It gave me the opportunity to experiment with a new programming language. Although not overly difficult to grasp, Prolog suffers from scarce and fragmented learning resources. Its programming paradigm emphasizes recursion, lacking conventional loop structures and straightforward arithmetic operations. Despite these limitations, Prolog excels in knowledge representation tasks, where retrieving and managing data from knowledge bases is intuitive. Nonetheless, its performance decreases significantly with large datasets, leading to long computation times. While I appreciated the learning experience, the lack of clarity in certain areas and its scalability issues were notable drawbacks.

- **Protégé**: I perceived Protégé as a hybrid approach between DMN and object-oriented programming. Within this environment, we could define objects, their attributes, and their relationships, while also extracting knowledge from these instances. What I particularly appreciated was the similarity of its query system to SQL, which made the tool easier to use. The SHACL validator was especially valuable, as it allowed us to verify whether a property was correctly defined—much like performing assertTrue or assertFalse in Java. Furthermore, SWRL rules enabled type inference, though I believe they should be applied sparingly. This view stems from my experience with tools like AOAME and GraphDB, which lack

built-in reasoning capabilities, requiring precise query construction. For instance, in our project, rules about meals demanded careful inference to identify allergenic ingredients based on meal components.

- **AOAME**: In my opinion, AOAME proved to be one of the most useful tools, mainly due to its integration of BPMN 2.0 with ontologies. This allowed us to design diagrams that were simpler and more intuitive compared to those produced without it. By creating the class Suggest Meal (a subclass of Task), we introduced a new graphical notation that was clear and practical for restaurant managers. We also extended the ontology by defining four new properties—course, allergy, calorie-conscious, and category—which we successfully combined into the modeling framework. Using Apache Jena Fuseki, we integrated our ontology into a triplestore and executed SPARQL queries to dynamically generate personalized meal suggestions. Query development was straightforward, thanks to similarities with Protégé. However, AOAME presented some challenges, including frequent bugs that undermined user experience and stability, as well as performance issues in its online version during high-traffic periods. Despite these flaws, AOAME remains a powerful and highly effective tool.

Conclusion

Each knowledge-based solution we explored has distinct merits and limitations. Decision tables are simple and user-friendly but become cumbersome in complex scenarios. Prolog provides intuitive data handling yet struggles with scalability and suffers from limited resources. Protégé offers a flexible and SQL-like approach to ontology modeling, but requires careful management of rules and reasoning. AOAME stands out for its ability to integrate meta-modeling with ontologies, enabling intuitive diagrams and dynamic reasoning, though it faces stability and scalability issues. Ultimately, the most appropriate solution depends on the project's requirements, the complexity of the knowledge base, and the desired level of customization.

5.2 Simone Micarelli

The project aimed to enhance the dining experience by designing a system that personalizes restaurant menus according to guest preferences and dietary restrictions. This approach addresses the challenge of navigating large digital menus on small screens, ensuring that guests only see meals they can or want to consume.

To achieve this, we developed several knowledge-based solutions to recommend meals based on guest profiles, using different representation methods:

- **Decision Tables** Decision tables proved practical and straightforward for managing rule-based logic. They provide a structured framework where inputs are mapped to outputs according to predefined rules. Their simplicity makes them ideal for clear, rule-driven processes. However, this simplicity also becomes a limitation when handling numerous variables and conditions. In such cases, decision tables can become unwieldy, leading to cumbersome implementations. Manual creation and debugging can also be repetitive and time-consuming, especially given the limited support from online tools. In summary, decision tables are well-suited for simple scenarios but less effective in complex, large-scale applications.

- **Prolog** Working with Prolog introduced a new paradigm, highlighting its strengths in logical reasoning and querying knowledge bases. Its declarative nature makes reasoning about data intuitive and powerful. However, challenges arose with its ambiguous syntax, steep learning curve, and reduced efficiency when managing large knowledge bases. While highly effective for logic-intensive tasks, Prolog is less practical for projects requiring fast, large-scale data processing or straightforward implementations.

- **Knowledge Graphs and Ontologies** We built an ontology to represent detailed knowledge about meals, ingredients, and guest preferences, using SWRL for rule definition, SPARQL for querying, and SHACL for validation. This provided a rich, extensible framework for capturing and reasoning about domain knowledge.

  SWRL extends OWL by enabling advanced rule-based reasoning, such as automatically inferring whether a meal contains allergens based on its ingredients. While powerful, it can become computationally heavy with large datasets or complex rules.

  SHACL validates RDF graphs against defined constraints, ensuring data consistency and integrity. Its strength lies in providing robust validation mechanisms, similar to unit testing, but it requires careful and often complex definitions as ontologies scale.

  By combining SWRL's reasoning capabilities with SHACL's validation, our ontology framework achieved both flexibility and rigor, though at the cost of added complexity and potential performance overhead.

Building on this ontology, we employed BPMN 2.0 to model the meal-selection process. The graphical notation made the decision-making steps intuitive for restaurant managers. Using Apache Jena Fuseki, we integrated the ontology into a triple store and ran SPARQL queries to dynamically generate personalized menu suggestions.

Overall, the system has strong potential to enhance guest satisfaction by delivering customized dining experiences while simplifying decision-making for restaurant managers. It demonstrates how advanced knowledge representation techniques can be effectively applied to practical challenges in the restaurant industry, paving the way for smarter and more user-friendly digital menu systems.

Bibliography

[Aoa] *AOAME Reference:* URL: https://emisa-journal.org/emisa/article/view/310.

[Cama] *Camunda.* URL: https://camunda.com/.

[Camb] *Camunda Documentation.* URL: https://docs.camunda.org/manual/7.19/reference/dmn/feel/.

[Git] *Github Forked Project link:* URL: https://github.com/Elmicass/Ontology4ModelingEnvironment.

[Proa] *Prolog Testing.* URL: https://swish.swi-prolog.org/.

[Prob] *Prolog Wikipedia.* URL: https://it.wikipedia.org/wiki/Prolog.

[Sha] *SHACL Shape File:* URL: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project/blob/main/Task1/Ontology/shacl_menu.txt.

[Spa] *SPARQL example queries:* URL: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project/blob/main/Task1/Ontology/sparql_queries.txt.

[Wik] Wikipedia. *Decision Table.* URL: https://en.wikipedia.org/wiki/Decision_table.