University of Camerino

School of Science and Technology

Course of Computer Science (Class LM-18)

Smart Personalized Menu System for Restaurants Using Knowledge-Based Solutions and Ontology-Driven Meta-Modelling

Students
Andrea Mazzetti
Lorenzo Paolucci

Supervisors
Prof. Knut Hinkelmann
Prof. Emanuele Laurenzi

A.A. 2024/2025

Contents

List of Figures

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 our 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: (https://github.com/LorenzoPaolucci000/KEBI-Project-2025.git) one detailing the development of the knowledge-based decision support system, and the other (https://github.com/LorenzoPaolucci000/Ontology4ModelingEnvironment.git) focusing on the integration of agile and ontology-based meta-modelling.

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, we will highlight our knowledge based solutions and the methods by which they were implemented to ensure that the appropriate meals are recommended to all types of guests. Firstly, we developed a Prolog script using the functionalities provided by SWI-Prolog[1]. Next, we used Knowledge Graphs to offer a convenient format for storing data, allowing us to query the information as if it were stored in a database. Finally, using Trisotech[2] and its decision modeler platform, we created the Decision Tables to match user preferences and filter out unwanted meals.

2.1 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 [Proa].

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, and this is the sense of the backtracking.

One of the main tool to write in Prolog is SWI-Prolog, a free implementation of the programming language Prolog, commonly used for teaching and semantic web applications. It has a rich set of features, libraries for constraint logic programming,

[1]SWI-Prolog: https://www.swi-prolog.org/
[2]Trisotech: https://www.trisotech.com/

multithreading, unit testing, GUI, interfacing to Java, ODBC and others, literate programming, a web server, SGML, RDF, RDFS, developer tools (including an IDE with a GUI debugger and GUI profiler), and extensive documentation [Swi].

2.1.1 Our Prolog

The first step to create the Prolog was to define the meal and the intolerances to take care of, then we started to write all the ingredients.

```
% drink
ingredient(water).
% Seasoning
ingredient(chili).
ingredient(garlic).
ingredient(lemon).
ingredient(oil).
ingredient(pepper).
ingredient(salt).
% Sweetener
ingredient(food_coloring).
ingredient(sugar).
% Aromatic Herbs
ingredient(basil).
ingredient(parsley).
ingredient(rosemary).
ingredient(sage).
% Vegetables
ingredient(arugula).
ingredient(capsicum).
ingredient(eggplant).
ingredient(mushroom).
ingredient(onion).
ingredient(tomato).
ingredient(zucchini).
% Tuber
ingredient(potato).
% Plant products
ingredient(chocolate).
% Nut
ingredient(almond).
ingredient(pistachio).
% Meat
ingredient(bacon).
ingredient(chicken).
ingredient(ham).
ingredient(pepperoni).
ingredient(sausage).
ingredient(steak).
% Fish
ingredient(anchovy).
ingredient(cuttlefish).
ingredient(octopus).
ingredient(salmon).
ingredient(shrimp).
ingredient(squid).
ingredient(tuna).
```

```
% Dairy
ingredient(butter).
ingredient(cream).
ingredient(milk).
ingredient(mozzarella_cheese).
ingredient(pecorino).
ingredient(ricotta).
% Animal Products
ingredient(egg).
ingredient(honey).
ingredient(parmesan).
% Grain Product
ingredient(bread).
ingredient(pasta).
ingredient(type_0_flour).
% Cereal Product
ingredient(buckwheat_pasta).
ingredient(buckwheat_flour).
% Fruits
ingredient(apple).
ingredient(banana).
ingredient(olive).
ingredient(strawberry).
```

Listing 2.1: Prolog Ingredients

For each ingredient were defined the calories.

```
% KCAL
kcal_ingredient(almond, 90).
kcal_ingredient(anchovy, 19).
kcal_ingredient(apple, 35).
kcal_ingredient(arugula, 3).
kcal_ingredient(bacon, 198).
kcal_ingredient(banana, 40).
kcal_ingredient(basil, 1).
kcal_ingredient(bread, 135).
kcal_ingredient(buckwheat_flour, 103).
kcal_ingredient(buckwheat_pasta, 278).
kcal_ingredient(butter, 75).
kcal_ingredient(capsicum, 11).
kcal_ingredient(chicken, 324).
kcal_ingredient(chili, 15).
kcal_ingredient(chocolate, 109).
kcal_ingredient(cream, 40).
kcal_ingredient(cuttlefish, 72).
kcal_ingredient(egg, 80).
kcal_ingredient(eggplant, 18).
kcal_ingredient(food_coloring, 5).
kcal_ingredient(garlic, 3).
kcal_ingredient(ham, 45).
kcal_ingredient(honey, 46).
kcal_ingredient(lemon, 4).
kcal_ingredient(mushroom, 15).
kcal_ingredient(milk, 128).
kcal_ingredient(mozzarella_cheese, 100).
kcal_ingredient(octopus, 60).
```

```
kcal_ingredient(oil, 60).
kcal_ingredient(olive, 47).
kcal_ingredient(onion, 13).
kcal_ingredient(parmesan, 75).
kcal_ingredient(parsley, 1).
kcal_ingredient(pasta, 300).
kcal_ingredient(pecorino, 80).
kcal_ingredient(pepper, 13).
kcal_ingredient(pepperoni, 150).
kcal_ingredient(pistachio, 120).
kcal_ingredient(potato, 450).
kcal_ingredient(ricotta, 89).
kcal_ingredient(rosemary, 2).
kcal_ingredient(sage, 2).
kcal_ingredient(salmon, 150).
kcal_ingredient(salt, 0).
kcal_ingredient(sausage, 205).
kcal_ingredient(shrimp, 35).
kcal_ingredient(squid, 69).
kcal_ingredient(steak, 485).
kcal_ingredient(strawberry, 22).
kcal_ingredient(sugar, 39).
kcal_ingredient(tomato, 30).
kcal_ingredient(tuna, 158).
kcal_ingredient(type_0_flour, 500).
kcal_ingredient(water, 0).
kcal_ingredient(zucchini, 13).
```

Listing 2.2: Prolog Ingredients Calories

To better identify the ingredient, it was labeled as carnivorous or vegetarian and considering the intolerance (gluten, lactose, nut).

```
% Carnivorous Ingredient
ingredient_carnivore(anchovy). % F
ingredient_carnivore(bacon). % M
ingredient_carnivore(chicken). % M
ingredient_carnivore(cuttlefish). % F
ingredient_carnivore(ham). % M
ingredient_carnivore(octopus). % F
ingredient_carnivore(pepperoni). % M
ingredient_carnivore(salmon). % F
ingredient_carnivore(sausage). % M
ingredient_carnivore(shrimp). % F
ingredient_carnivore(squid). % F
ingredient_carnivore(steak). % M
ingredient_carnivore(tuna). % F
% Vegetarian Ingredient
ingredient_vegetarian(arugula).
ingredient_vegetarian(capsicum).
ingredient_vegetarian(eggplant).
ingredient_vegetarian(mushroom).
ingredient_vegetarian(potato).
ingredient_vegetarian(olive).
ingredient_vegetarian(onion).
ingredient_vegetarian(tomato).
ingredient_vegetarian(zucchini).
```

```
ingredient_vegetarian(apple).
ingredient_vegetarian(banana).
ingredient_vegetarian(strawberry).
% lactose Intolerance
ingredient_with_lactose_intolerance(butter).
ingredient_with_lactose_intolerance(cream).
ingredient_with_lactose_intolerance(milk).
ingredient_with_lactose_intolerance(mozzarella_cheese).
ingredient_with_lactose_intolerance(pecorino).
ingredient_with_lactose_intolerance(ricotta).
% Gluten Intolerance
ingredient_with_gluten_intolerance(bread).
ingredient_with_gluten_intolerance(type_0_flour).
ingredient_with_gluten_intolerance(pasta).
% Nut Intolerance
ingredient_with_nut_intolerance(almond).
ingredient_with_nut_intolerance(pistachio).
```

Listing 2.3: Prolog Type of Ingredients

In the end, meals were written, specifying the meal name, course type (e.g., appetizer, first dish), and a list of ingredients. The structured meal definitions provide a foundation for querying and analyzing meal options based on their composition. This organization facilitates the creation of meal recommendations and nutritional assessments.

```
% Appetizer
meal(bruschette, appetizer, [basil, bread, ham, mushroom,
    mozzarella_cheese, oil, pepper, salt, tomato]).
meal(cheese_platter_and_honey, appetizer, [honey, parmesan, pecorino
    , ricotta]).
meal(french_fries, appetizer, [oil, potato, salt]).
meal(omelette, appetizer, [egg, oil, onion, pepper, salt, zucchini])
    .
meal(seafood_salad, appetizer, [basil, cuttlefish, octopus, oil,
    salt, shrimp, squid]).
% First Dish
meal(carbonara_pasta, first_dish, [oil, pepper, salt, bacon, egg,
    pecorino, pasta]).
meal(gricia_pasta, first_dish, [oil, pepper, salt, bacon, pecorino,
    buckwheat_pasta]).
meal(norcina_pasta, first_dish, [oil, pepper, salt, parmesan,
    sausage, cream, pasta]).
meal(norma_pasta, first_dish, [oil, salt, eggplant, tomato, ricotta,
     pasta]).
meal(spaghetti_garlic_oil_and_chili, first_dish, [oil, garlic, salt,
     chili, buckwheat_pasta]).
% Second Course
meal(grilled_salmon, second_course, [lemon, oil, salt, chili,
    parsley, salmon, pistachio]).
meal(ratatouille, second_course, [oil, garlic, salt, basil, capsicum
    , eggplant, onion, parsley, tomato, zucchini]).
meal(roasted_chicken, second_course, [oil, garlic, pepper, salt,
    rosemary, sage, chicken]).
meal(steak, second_course, [lemon, oil, pepper, salt, steak,
    rosemary]).
```

```
meal(tuna_fishballs, second_course, [oil, pepper, salt, parsley,
    anchovy, tuna, egg, parmesan, ricotta, bread]).
% Main Dish
meal(pizza_margherita, main_dish, [oil, salt, water, basil, tomato,
    mozzarella_cheese, type_0_flour]).
meal(pizza_ortolana, main_dish, [oil, salt, water, basil, capsicum ,
     eggplant, onion, tomato, zucchini, buckwheat_flour]).
meal(pizza_tri-color, main_dish, [oil, salt, water, arugula, tomato,
     parmesan, buckwheat_flour]).
meal(pizza_contadina, main_dish, [oil, salt, water, olive, ham,
    mozzarella_cheese, type_0_flour]).
meal(pizza_pepperoni, main_dish, [oil, salt, water,
    mozzarella_cheese, sausage, pepperoni, type_0_flour]).
% Dessert
meal(fruit_salad, dessert, [almond, apple, banana, strawberry, sugar
    ]).
meal(crepes, dessert, [butter, chocolate, egg, milk, type_0_flour,
    sugar]).
% Drink
meal(water, drink, [water]).
meal(coca_cola, drink, [food_coloring, sugar, water]).
```

Listing 2.4: Prolog Meal

At this point, we started to create the predicates to determine the meal considering:

- vegetarian meal when there is at least one ingredient vegetarian or non-carnivorous;
- carnivorous meal when there is at least one ingredient carnivorous or non-vegetarian;
- omnivore meal all type of ingredients that is composed of;
- meal with gluten intolerance when there is at least one ingredient with gluten;
- meal with lactose intolerance when there is at least one ingredient with lactose;
- meal with nut intolerance when there is at least one ingredient with nut;
- the meal calories;
- the calories conscious levels of the meal;
- the preferences of the guest.

```
% Vegetarian Meal
vegetarian_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    % Ensure all ingredients are either vegetarian or not
       carnivorous
    forall(member(Ingredient, Ingredients),
           (ingredient_vegetarian(Ingredient); \+
              ingredient_carnivore(Ingredient))).
% Carnivorous Meal
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))).
% Omnivore Meal
omnivore_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    forall(member(Ingredient, Ingredients), ingredient(Ingredient)).
% Meal with Gluten Intolerance
meal_with_gluten_intolerance(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    member(Ingredient, Ingredients),
    ingredient_with_gluten_intolerance(Ingredient).
% Meals with Lactose Intolerance
meal_with_lactose_intolerance(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    member(Ingredient, Ingredients),
    ingredient_with_lactose_intolerance(Ingredient).
% Meals with Nut Intolerance
meal_with_nut_intolerance(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    member(Ingredient, Ingredients),
    ingredient_with_nut_intolerance(Ingredient).
% Total calories 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).
% Calorie-conscious levels (based on total calories)
calorie_conscious_levels(Meal, Course, Levels) :-
    meal_calories(Meal, Course, TotalCalories),
    % 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).
% Guest Preferences (based on 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
```

```
64      ; ( Allergies = gluten -> not(meal_with_gluten_intolerance(Meal,
            Course))
65        ; Allergies = lactose -> not(meal_with_lactose_intolerance(
              Meal, Course))
66        ; Allergies = nut -> not(meal_with_nut_intolerance(Meal,
              Course))
67      )
68      ).
```

Listing 2.5: Prolog Meal Predicates

This Prolog code offers a comprehensive system for managing meal ingredients, analyzing their nutritional content, and accommodating dietary preferences and restrictions. The organization of predicates ensures flexibility and precision in meal recommendations based on various criteria.

2.1.2 Prolog Testing

Our objective in the testing phase was to show some scenarios that may happen with the guests. The query used to test the Prolog is the guest_preferences, that need the category of the guest, his calorie consciousness levels, the allergy if he has one and it wants the meal and the course in response. The testing is about:

- omnivore with 0 calorie conscious levels and without allergy;
- omnivore with 1 calorie conscious levels and nut allergy;
- omnivore with 0 calorie conscious levels and gluten allergy;
- carnivorous with 2 calorie conscious levels and without allergy;
- carnivorous with 0 calorie conscious levels and without allergy;
- carnivorous with 0 calorie conscious levels and lactose allergy;
- vegetarian with 3 calorie conscious levels and without allergy;
- vegetarian with 2 calorie conscious levels and nut allergy;
- vegetarian with 1 calorie conscious levels and without allergy.

```
1  % Some queries for testing
2  guest_preferences(omnivore, 0, none, Meal, Course).
3  guest_preferences(omnivore, 1, nut, Meal, Course).
4  guest_preferences(omnivore, 0, gluten, Meal, Course).
5  guest_preferences(carnivorous, 2, none, Meal, Course).
6  guest_preferences(carnivorous, 0, none, Meal, Course).
7  guest_preferences(carnivorous, 0, lactose, Meal, Course).
8  guest_preferences(vegetarian, 3, none, Meal, Course).
9  guest_preferences(vegetarian, 2, nut, Meal, Course).
10 guest_preferences(vegetarian, 1, none, Meal, Course).
```

Listing 2.6: Queries for Prolog Testing

The Figure 2.1 shows the results of query guest_preferences(vegetarian, 2, nut, Meal, Course).

Figure 2.1: Prolog testing

2.2 Knowledge Graph and Ontology Engineering

In computer science, an ontology is a formal representation of knowledge as a set of concepts within a specific domain and the relationships between those concepts. It is essentially a structured way to organize and share information, enabling computers to understand and reason about data in a meaningful way. Think of it as a sophisticated data model that goes beyond simple databases by incorporating semantic meaning and relationships.

2.2.1 Protégé and GraphDB

Protégé is a free, open-source ontology editor and framework for building intelligent systems. It can be adapted to build both simple and complex ontology-based applications, define classes, properties, and individuals, use reasoners (like HermiT, Pellet) to infer logical consequences and integrate the output of Protégé with rule systems or other problem solvers to construct a wide range of intelligent systems [Prob]. The rules and constraints used for the project are:

- **SWRL (Semantic Web Rule Language) Rules:** a rule-based language used to express logical rules in OWL ontologies (usually built in tools like Protégé) in order to infer new knowledge from existing facts;
- **SPARQL (SPARQL Protocol and RDF Query Language):** a query language designed specifically for querying and retrieving data stored in RDF (Resource Description Framework) format. RDF is a standardized approach utilized for describing and exchanging graph data in a general manner. It enables the description of resources that are utilized in constructing knowledge graphs;
- **SHACL (Shapes Constraint Language):** a W3C standard used to validate RDF data—it checks whether the data conforms to a set of conditions or "shapes".

It ensure data conforms to ontology rules and report violations when data does not match expectations.

Figure 2.2: Protégé Logo

SPARQL queries were tested even using GraphDB, an enterprise ready Semantic Graph Database, compliant with W3C Standards. Semantic graph databases (also called RDF triplestores) provide the core infrastructure for solutions where modelling agility, data integration, relationship exploration and cross-enterprise data publishing and consumption are important [Gra].

Figure 2.3: GraphDB Logo

2.2.2 Our 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 inclusion of other resources, such as potential allergens, types of ingredients and more.

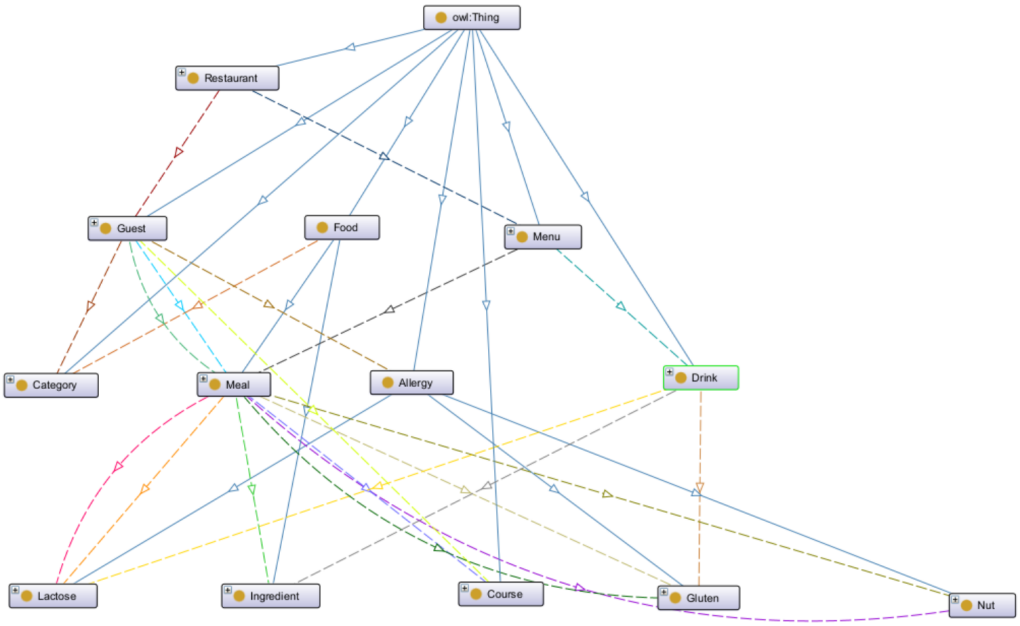


Figure 2.4: Protégé Ontology Graph

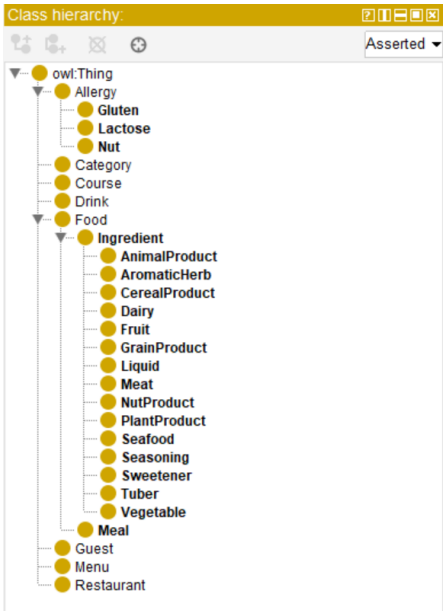


Figure 2.5: Protégé Classes

Object Properties

Object properties are used to represent the relationships between classes. We defined several object properties for the classes:

- **restaurant_hasMenu:** to associate a menu with a restaurant;
- **restaurant_hasGuest:** to associate guest with a restaurant;
- **menu_containsDrink:** to associate a drink with a menu;
- **menu_containsMeal:** to associate a meal with a menu;
- **guest_hasAllergies:** it represents a guest that has an allergy;
- **guest_hasCategory:** to associate a category (Carnivorous, Vegetarian, Omnivore) with a guest.
- **guest_hasPreferenceCourse:** to associate a specific course with a guest;
- **guest_foodWithIntoleranceRisk:** 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_containsNutIntolerance:** this is an SWRL rule inferred by the reasoner, which identifies if the meal contains ingredients with nut (NutProduct);
- **meal_notContainsGlutenIntolerance:** added to check if a meal is gluten-free;
- **meal_notContainsLactoseIntolerance:** added to check if a meal is lactose-free;
- **meal_notContainsNutIntolerance:** added to check if a meal is nut-free;t
- **meal_hasCourse:** to associate a course with the meal;
- **meal_hasIngredient:** to associate an ingredient with a meal;
- **drink_hasIngredient:** to associate an ingredient with a drink;
- **drink_containsGlutenIntolerance:** to associate a drink with gluten intolerance;
- **drink_containsLactoseIntolerance:** to associate a drink with lactose intolerance;
- **drink_containsNutIntolerance:** to associate a drink with nut intoleranc;
- **grainProduct_containsGlutenIntolerance:** to associate the gluten intolerance to class Grain;

- **dairy_containsLactoseIntolerance:** to associate the lactose intolerance to class Diary;
- **nutProduct_containsNutIntolerance:** to associate the nut intolerance to class NutProduct.

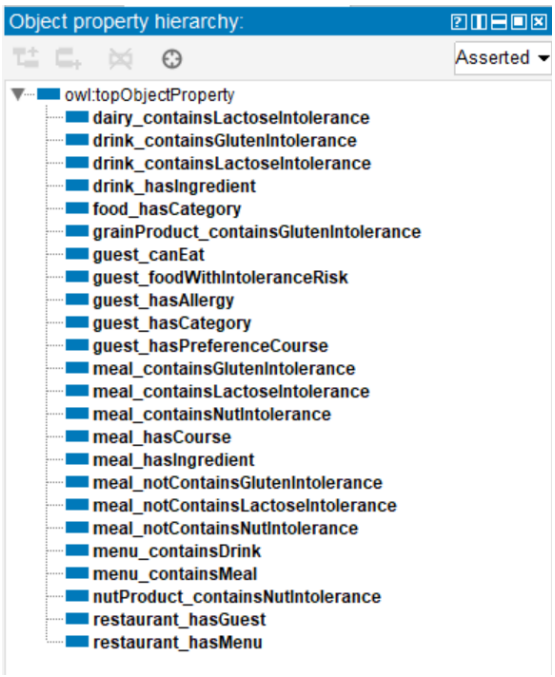


Figure 2.6: Protégé Object Properties

Data Properties

Data Properties represent the attributes of the classes. Below, all the Data Properties are illustrated:

- **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 (in this case we have only 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 3;
- **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.

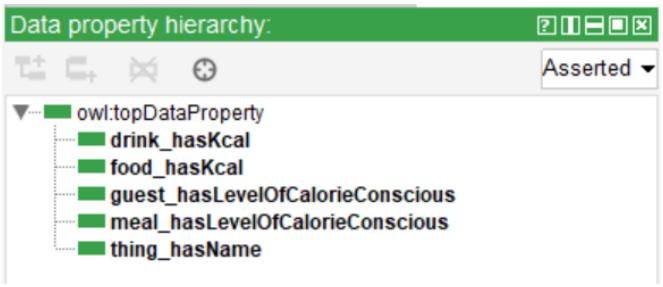


Figure 2.7: Protégé Data Properties

Individuals

We created many individuals for all necessary drinks, meals and ingredients.

The Restaurant class has one instance with a name, a menu and some guests.

The Menu class has one instance with drinks and meals.

The Guest class has four instance to consider different cases of guest with or without intolerance.

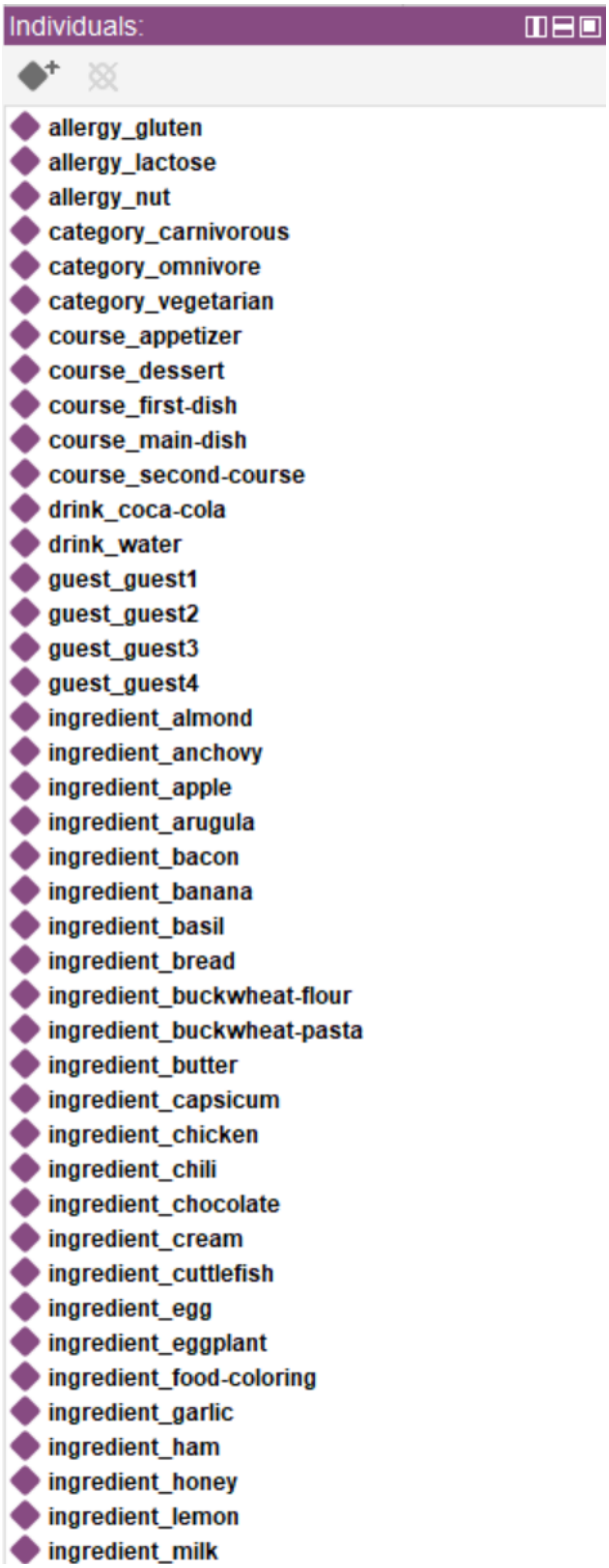


Figure 2.8: Some Individuals

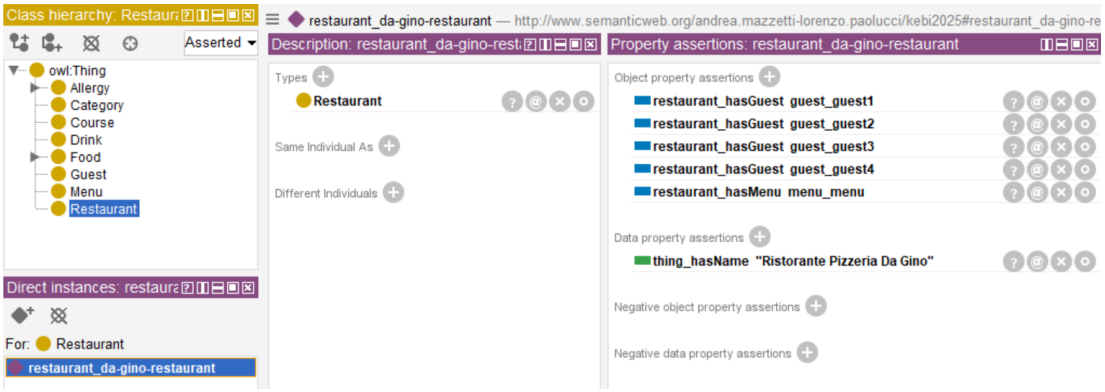


Figure 2.9: Protégé Restaurant Individual

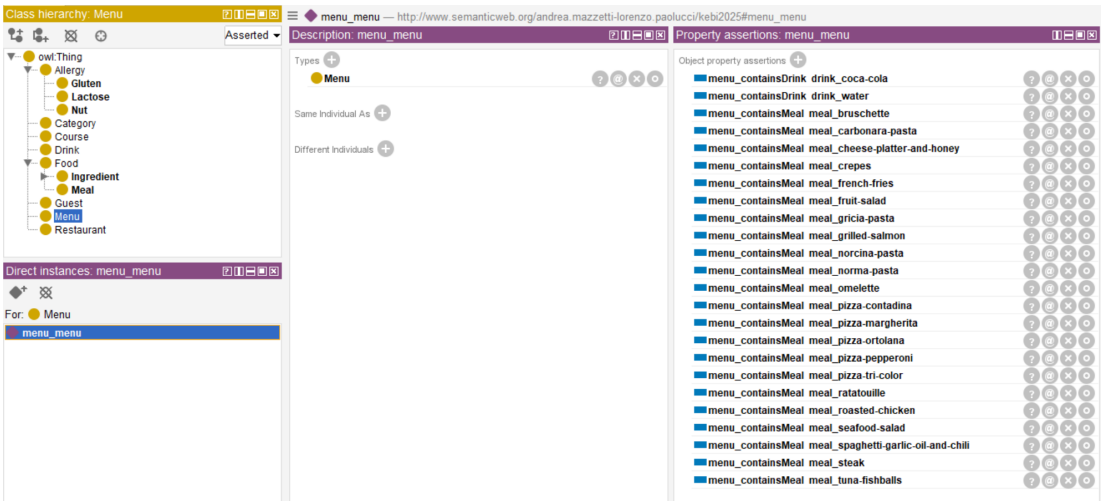


Figure 2.10: Protégé Menu Individual

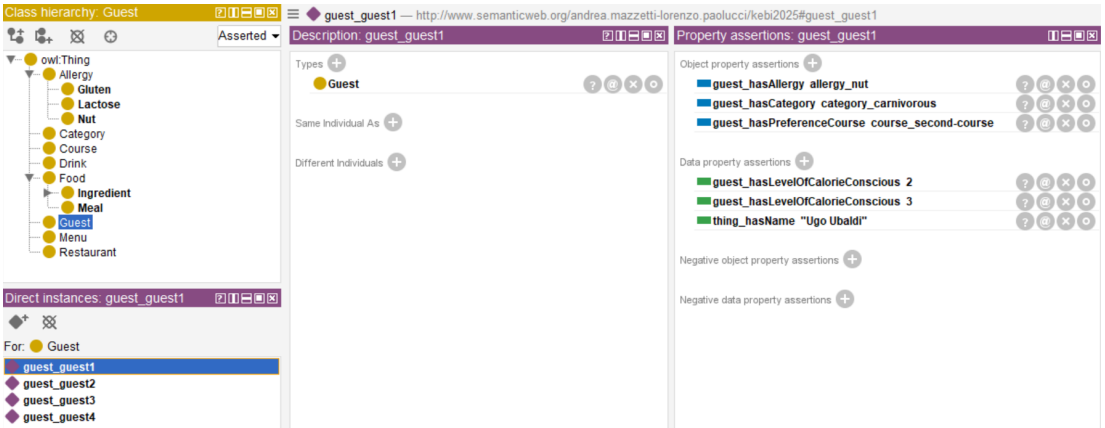


Figure 2.11: Protégé Guest Individual

The Meal class has 17 individuals; figure 2.12 represents one of the individual meals created.

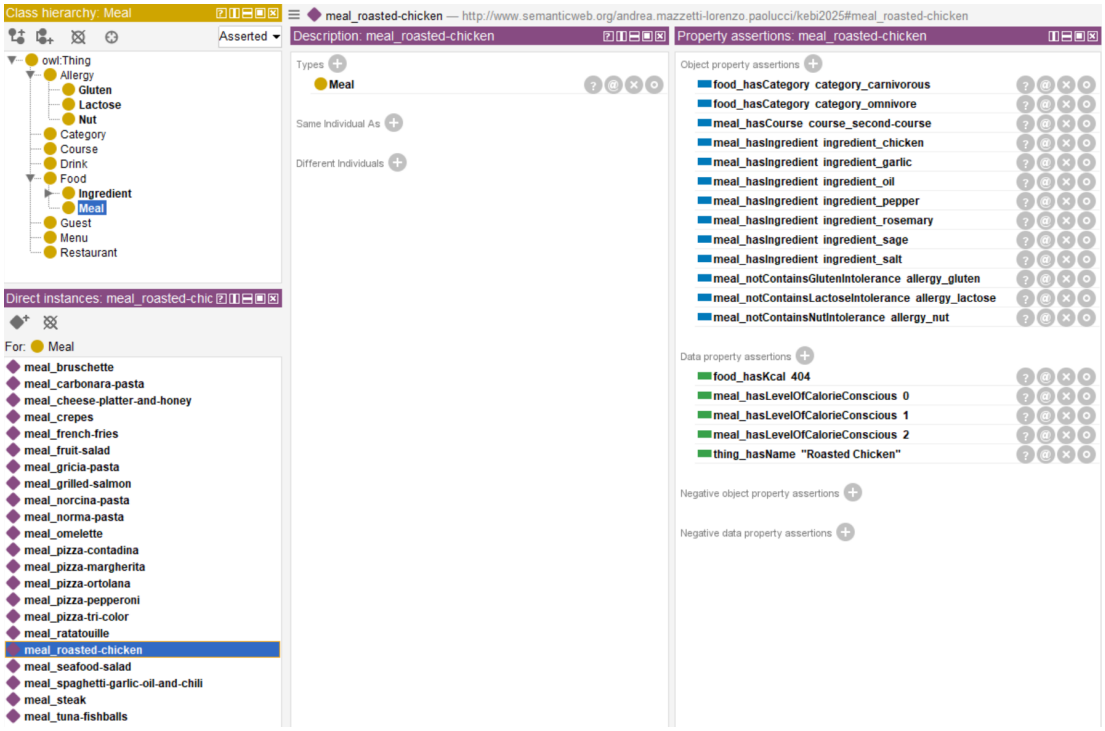


Figure 2.12: Protégé Meal Individual

2.2.3 SWRL

When all classes, object properties, and data properties were created, we started to write the rules and constraints of our ontology.

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

  kebi2025:Meal(?meal) $\wedge$
  kebi2025:Ingredient(?ingredient) $\wedge$
  kebi2025:meal_hasIngredient(?meal, ?ingredient) $\wedge$
  kebi2025:grainProduct_containsGlutenIntolerance(?ingredient, ?gluten)
  -> kebi2025:meal_containsGlutenIntolerance(?meal, ?gluten)

- **MealContainsLactoseIntolerance:** this rule helps to infer meals that contain ingredients with gluten intolerance by adding lactose intolerance to the meal.

  kebi2025:Meal(?meal) $\wedge$
  kebi2025:Ingredient(?ingredient) $\wedge$
  kebi2025:meal_hasIngredient(?meal, ?ingredient) $\wedge$
  kebi2025:dairy_containsLactoseIntolerance(?ingredient, ?dairy)
  -> kebi2025:meal_containsLactoseIntolerance(?meal, ?dairy)

- **MealContainsNutIntolerance:** this rule helps to infer meals that contain ingredients with gluten intolerance by adding nut intolerance to the meal.

kebi2025:Meal(?meal) ∧
kebi2025:Ingredient(?ingredient) ∧
kebi2025:meal_hasIngredient(?meal, ?ingredient) ∧
kebi2025:nutProduct_containsNutIntolerance(?ingredient, ?nut)
-> kebi2025:meal_containsNutIntolerance(?meal, ?nut)

- **GuestWithGlutenIntolerance:** this rule helps detect dangerous situations for individuals with food allergies. It enables the ontology to automatically infer risk relationships between guests and meals, looking for gluten allergy among meal contents.

  kebi2025:Guest(?guest) ∧
  kebi2025:Meal(?meal) ∧
  kebi2025:guest_hasAllergy(?guest, ?allergy) ∧
  kebi2025:meal_containsGlutenIntolerance(?meal, ?allergy)
  -> kebi2025:guest_foodWithIntoleranceRisk(?guest, ?meal)

- **GuestWithLactoseIntolerance:** this rule helps detect dangerous situations for individuals with food allergies. It enables the ontology to automatically infer risk relationships between guests and meals, looking for lactose allergy among meal contents.

  kebi2025:Guest(?guest) ∧
  kebi2025:Meal(?meal) ∧
  kebi2025:guest_hasAllergy(?guest, ?allergy) ∧
  kebi2025:meal_containsLactoseIntolerance(?meal, ?allergy)
  -> kebi2025:guest_foodWithIntoleranceRisk(?guest, ?meal)

- **GuestWithNutIntolerance:** this rule helps detect dangerous situations for individuals with food allergies. It enables the ontology to automatically infer risk relationships between guests and meals, looking for nut allergy among meal contents.

  kebi2025:Guest(?guest) ∧
  kebi2025:Meal(?meal) ∧
  kebi2025:guest_hasAllergy(?guest, ?allergy) ∧
  kebi2025:meal_containsNutIntolerance(?meal, ?allergy)
  -> kebi2025:guest_foodWithIntoleranceRisk(?guest, ?meal)

- **Inferring meals to eat - gluten intolerance:** this rule infers the meals checking the guests' preferences, calorie-conscious level, food category, and gluten intolerance.

  kebi2025:Category(?category) ∧
  kebi2025:Allergy(?allergy) ∧
  kebi2025:Meal(?meal) ∧
  kebi2025:Guest(?guest) ∧
  kebi2025:food_hasCategory(?meal, ?category) ∧
  kebi2025:guest_hasAllergy(?guest, ?allergy) ∧
  kebi2025:guest_hasCategory(?guest, ?category) ∧
  kebi2025:guest_hasPreferenceCourse(?guest, ?course) ∧
  kebi2025:guest_hasLevelOfCalorieConscious(?guest, ?level) ∧
  kebi2025:meal_hasLevelOfCalorieConscious(?meal, ?level) ∧
  kebi2025:meal_hasCourse(?meal, ?course) ∧

kebi2025:meal_notContainsGlutenIntolerance(?meal, ?allergy)
-> kebi2025:guest_canEat(?guest, ?meal)

- **Inferring meals to eat - lactose intolerance:** this rule infers the meals checking the guests' preferences, calorie-conscious level, food category, and lactose intolerance.

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

- **Inferring meals to eat - nut intolerance:** this rule infers the meals checking the guests' preferences, calorie-conscious level, food category, and nut intolerance.

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

SWRL Testing

The testing was performed running the reasoner, in our case 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.

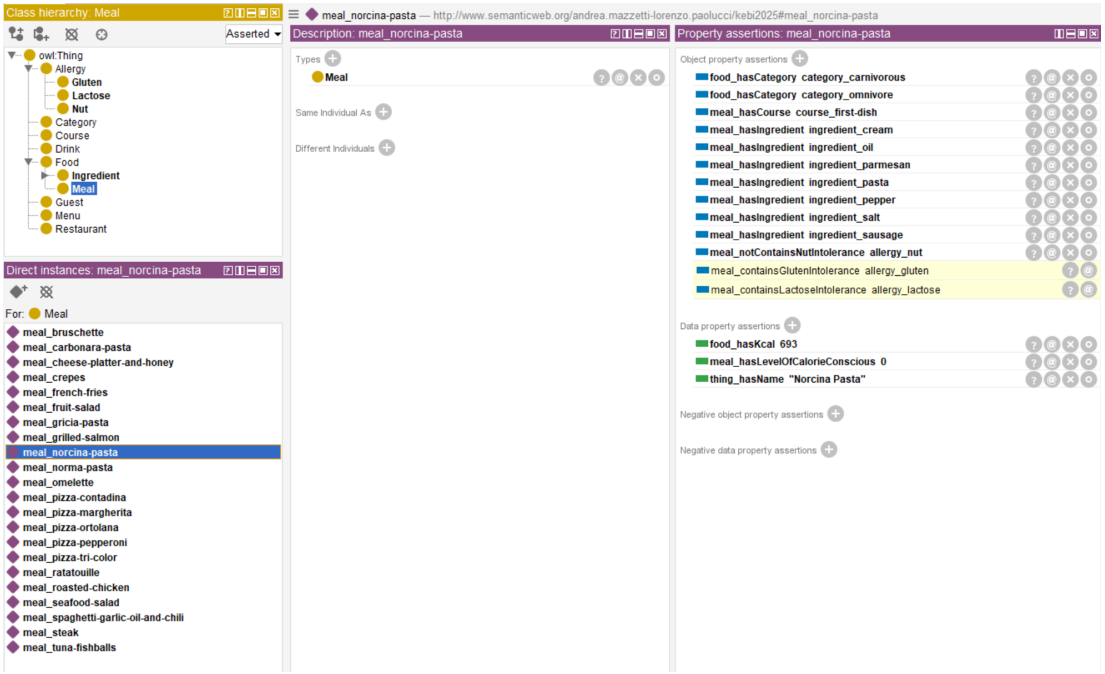


Figure 2.13: Meal Contains Gluten and Lactose Intolerance Test

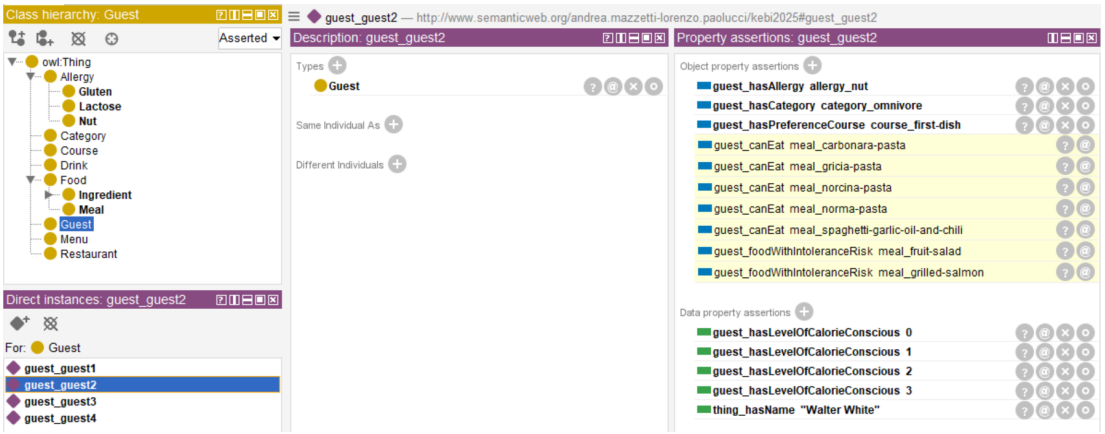


Figure 2.14: Guest with Nut Intolerance Test

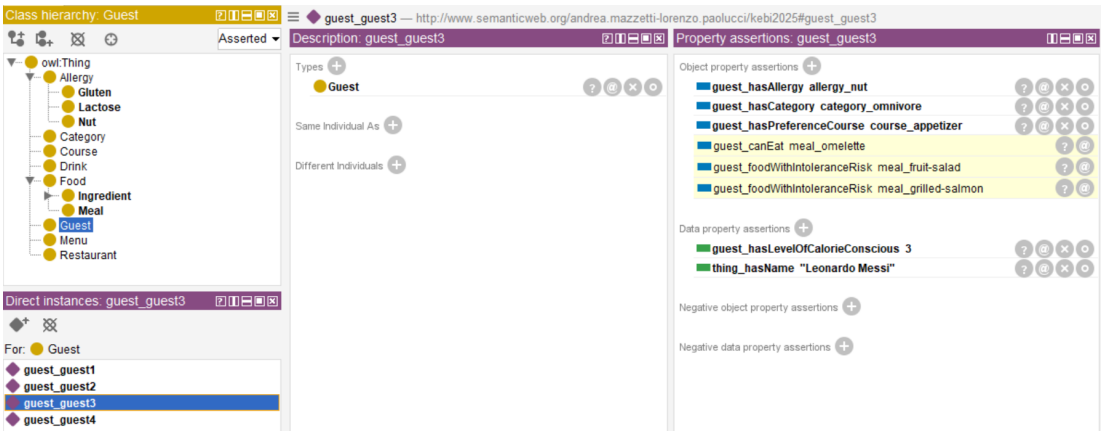


Figure 2.15: Inferring Meals to Eat - Lactose Intolerance Rule Test

2.2.4 SPARQL

With SPARQL we extracted data from dataset using queries in Protégé and GraphDB.

- **Prefix:** they are essential to make SPARQL work.

```
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/andrea.mazzetti-lorenzo
    .paolucci/kebi2025#>
```

Listing 2.7: Prefix

- **Vegetarian Meal:** this query search all instances of meals that belong to a specific category, in this case "vegetarian".

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

Listing 2.8: Vegetarian Meal

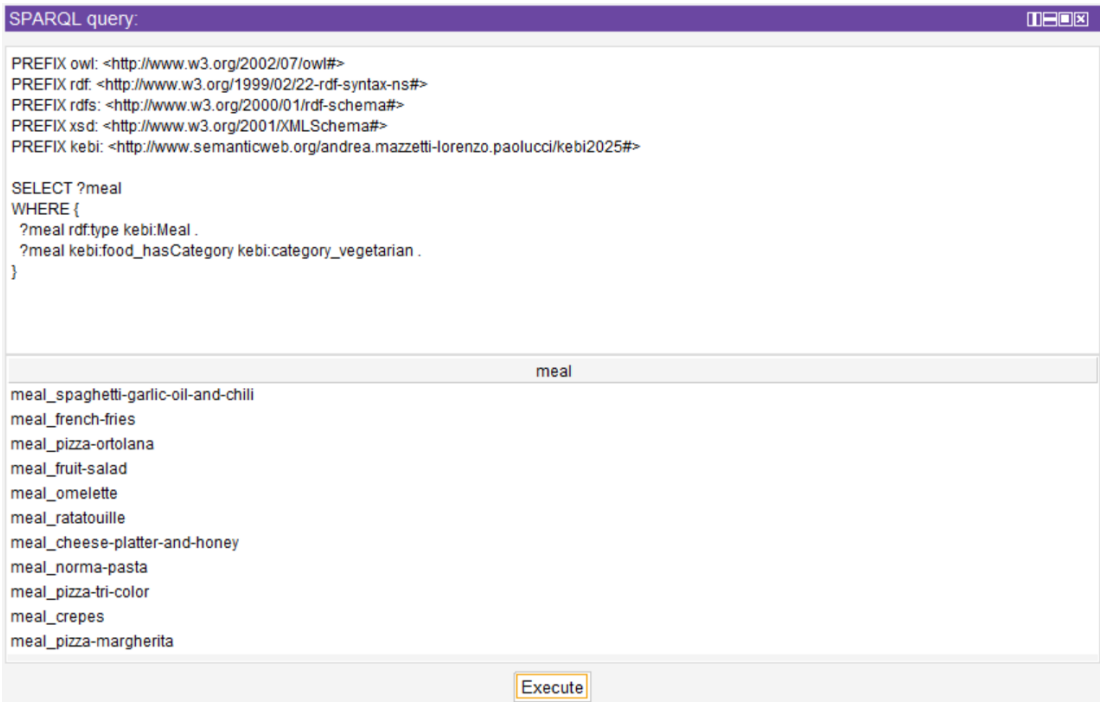


Figure 2.16: Protégé - SPARQL Query for Vegetarian Meal

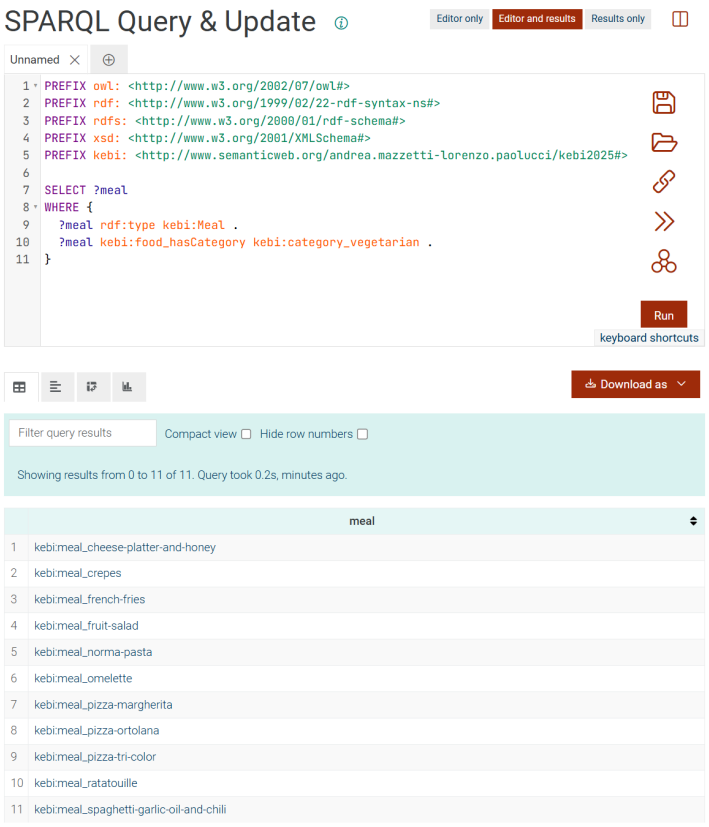


Figure 2.17: GraphDB - SPARQL Query for Vegetarian Meal

- **Meal Ingredients:** this query shows the ingredients of a specific meal, in this case "tuna-fishballs".

```
SELECT ?ingredient
WHERE {
  kebi:meal_tuna-fishballs kebi:meal_hasIngredient ?ingredient .
}
```

Listing 2.9: Tuna-fishballs Ingredients

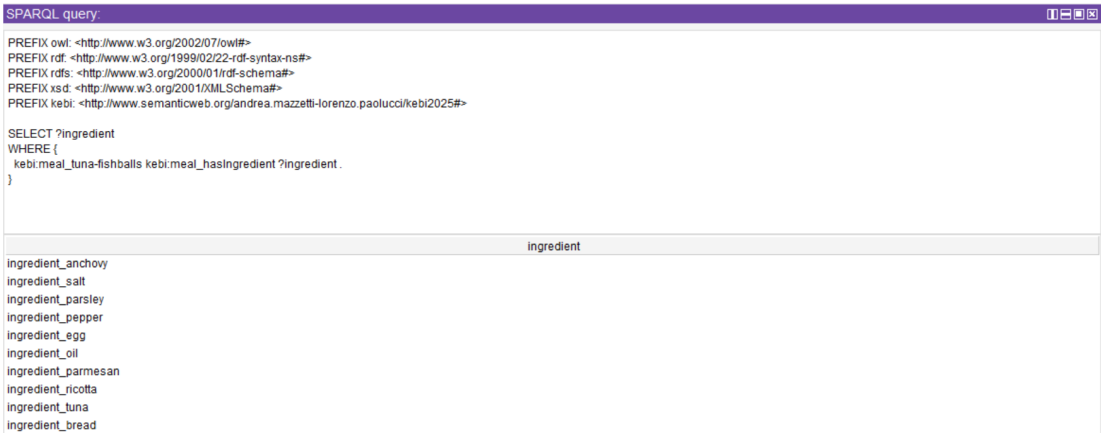


Figure 2.18: Protégé - SPARQL Query for Meal Ingredients

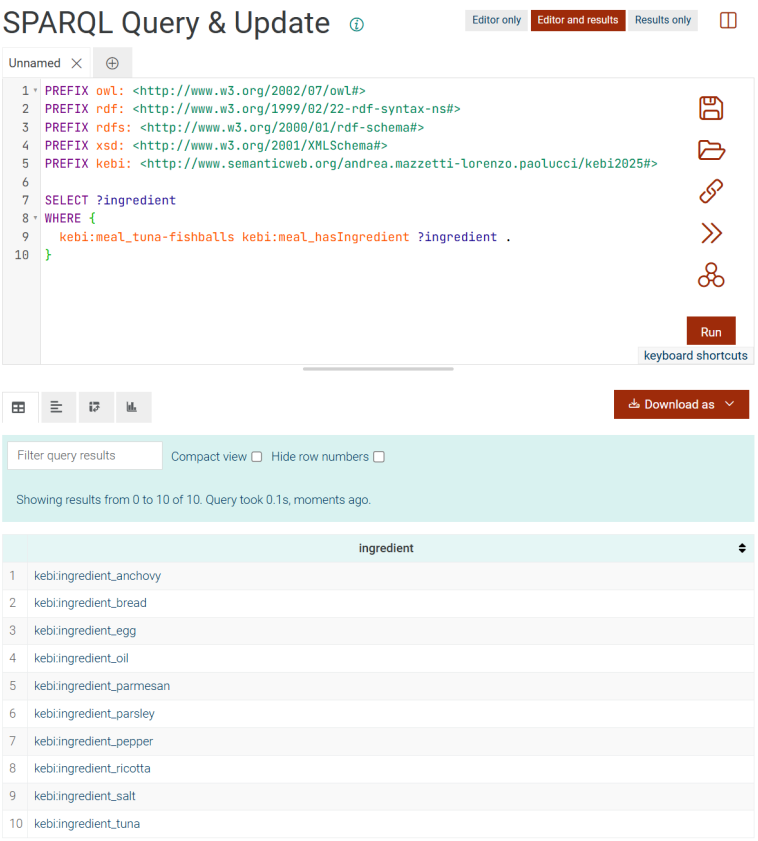


Figure 2.19: GraphDB - SPARQL Query for Meal Ingredients

- **Guest Preferences:** this query shows the meals based on the preferences of a specific 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 2.10: Guest Preferences

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/andrea.mazzetti-lorenzo.paolucci/kebi2025#>

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 }
  MINUS {?meal kebi:meal_containsNutIntolerance ?allergy }
}
```

meal	mealName
meal_roasted-chicken	"Roasted Chicken"^^<http://www.w3.org/2001/XMLSchema#string>
meal_grilled-salmon	"Grilled Salmon"^^<http://www.w3.org/2001/XMLSchema#string>

Figure 2.20: Protégé - SPARQL Query for Guest Preferences

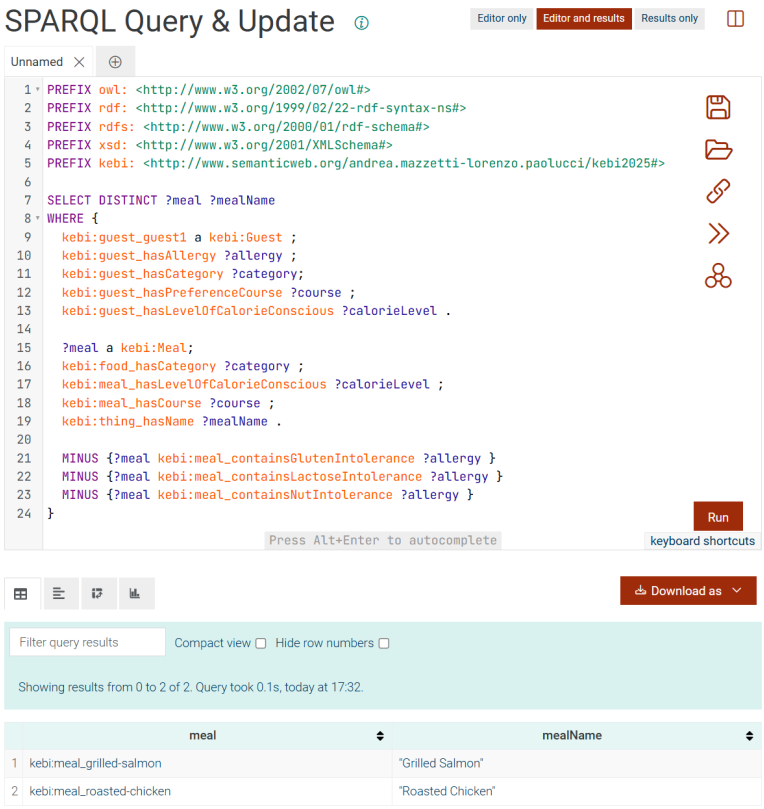


Figure 2.21: GraphDB - SPARQL Query for Guest Preferences

We created more queries to achieve all ingredients and specific ingredients with kcal, all meals a specific meal with kcal, meals with level 3 of conscious calorie, guests with allergy, ingredients with gluten, lactose, and nut intolerance, meals with gluten, lactose, and nut intolerance, the type of ingredient and the type of course of the meal; not all queries can be executed in GraphDB because to show the meals with some intolerance it is necessary Protégé reasoner. The reasoner is a tool that automatically infers logical consequences from the facts and rules in an ontology, it helps to deduce new knowledge based on existing data and class or property relationships.

2.2.5 SHACL

The last step of the ontology was to write the SHACL to validate data based on a set of conditions, known as shapes.

- **Restaurant Shape:** this shape check if the restaurant must have at least one menu that must pass MenuShape validation, and verify if it has a guest and if the guest passes GuestShape validation. If the restaurant does not have a menu, then return the error message "Restaurant does not have a menu!".

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

Listing 2.11: Restaurant Shape

- **Menu Shape:** this shape requires every menu to include at least one meal and at least one drink, each conforming to their respective shapes.

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

Listing 2.12: Menu Shape

- **Guest Shape:** this shape requires each guest to have at least one category and a calorie-consciousness level between 1 and 4, may have allergies, and can be linked to meals with intolerance risks.

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

    sh:property [
        sh:path kebi:guest_foodWithIntoleranceRisk ;
        sh:node kebi:MealShape ;
    ] ;
    sh:property [
        sh:path kebi:guest_hasLevelOfCalorieConscious ;
        sh:datatype xsd:integer ;
        sh:minCount 1 ;
        sh:maxCount 4 ;
    ] ;
.
```

Listing 2.13: Guest Shape

- **Meal Shape:** this shape defines that every meal must have exactly one course and at least one ingredient, may have categories, and can have zero or one intolerance for lactose, gluten, or nuts.

```
kebi:MealShape a sh:NodeShape ;
    sh:targetClass kebi:Meal ;
    sh:property [
        sh:path kebi:meal_containsLactoseIntolerance ;
        sh:node kebi:LactoseShape ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi:meal_containsGlutenIntolerance ;
        sh:node kebi:GlutenShape ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi:meal_containsNutIntolerance ;
        sh:node kebi:NutShape ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
```

```
        sh:path kebi:meal_hasCategory ;
        sh:node kebi:CategoryShape ;
        sh:minCount 0 ;
    ] ;
    sh:property [
        sh:path kebi:meal_hasCourse ;
        sh:node kebi:CourseShape ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi:meal_hasIngredient ;
        sh:node kebi:IngredientShape ;
        sh:minCount 1 ;
    ] ;
.
```

Listing 2.14: Meal Shape

Those are the main examples of SHACL shapes written; the others are about all classes and courses created.

Validation

The validation was made by checking one by one the shapes using the SHACL editor; if the editor did not find any violations, the validation is positive.

Figure 2.22: SHACL Validation

2.3 Decision Model

The Decision Model and Notation (DMN) has emerged as a powerful tool in the field of business analysis, offering a standardized approach to modeling and executing decision-making processes. Developed by the Object Management Group (OMG), DMN provides a common language for business analysts, data scientists, and IT professionals to collaborate effectively on decision management projects [Obj16]. This notation system enables organizations to clearly define, document, and automate complex decision logic, thereby enhancing operational efficiency and ensuring consistency in decision outcomes. By separating decision logic from process logic, DMN allows for greater flexibility and reusability of decision models across various business contexts. As businesses increasingly rely on data-driven decision-making, DMN has become an essential framework for translating business rules and policies into executable decision models that can be integrated with existing business process management systems.

2.3.1 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):** FEEL[3] is a Low-Code language used for evaluating expressions found within decision tables.

2.3.2 Trisotech

Trisotech is a Canadian company offering a cloud-based, low-code platform that bridges business and IT through open standards like BPMN, CMMN, and DMN. Founded in 1996[4] and based in Montreal, it enables organizations to model and automate processes and decisions collaboratively.

Figure 2.23: Trisotech Logo

Its main product, the Digital Enterprise Suite (DES)[5], includes the Digital Modeling Suite for visual modeling of processes, cases, and decisions, and the Digital Automation Suite, which executes these models via APIs, microservices, RPA, and AI/ML integrations. Used across sectors like healthcare, finance, and government, the platform supports initiatives such as BPM+ Health[6]. For this project, we used the Decision Modeler, which is a web application, part of the Trisotech Digital Enterprise Suite, where you can draw and model business decision models.

2.3.3 Our Decision Model

Our decision model is organized as illustrated in Figure 2.24. The following section will provide an introduction and detailed description of each component of the model.

2.3.4 Decision Tables

In this section, we will describe each decision table with reference to the preceding image. As you can see, there are two main decision tables:

- **Ingredients:** This decision table (Figure 2.25) processes multiple input parameters to identify suitable ingredients for a user based on their dietary restrictions and preferences. The main input is the Guest Category, which defines the user's general diet:

[3]FEEL: https://fhnw.trisotech.com/help/decision-modeler/feel.html
[4]Trisotech Foundation: https://www.trisotech.com/company
[5]Trisotech DES: https://www.trisotech.com/digital-enterprise-suite/
[6]BPM+ Community: https://www.bpm-plus.org/about-us.htm

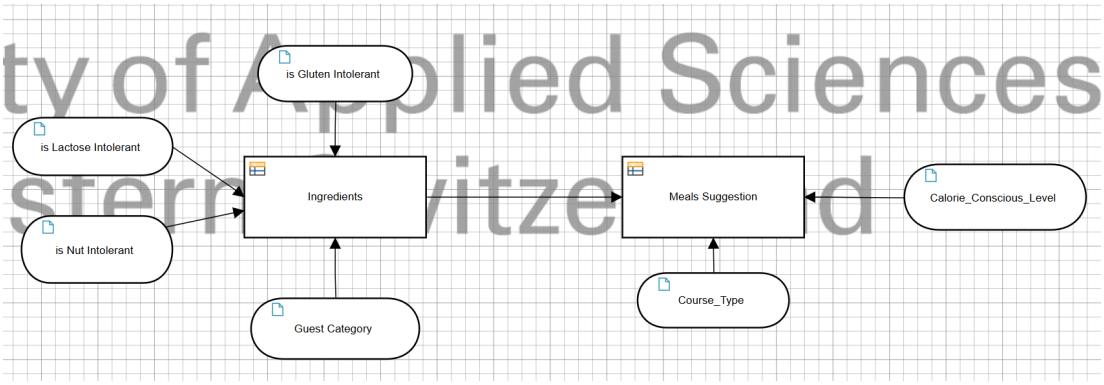


Figure 2.24: Decision Model in Trisotech

- Carnivorous: users who consume meat but generally avoid vegetables.
- Vegetarian: users who exclude meat from their diet.
- Omnivorous: users who consume both meat and vegetables.

This structure has been designed with extensibility in mind: for example, it can easily accommodate additional categories such as Vegan in the future, enabling support for further dietary types. In addition to the Guest Category, the table considers three boolean intolerance parameters: is Lactose Intolerant, is Gluten Intolerant, is Nut 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 2.26, 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, along with the caloric value (in kcal) of each ingredient.

- **Meals Suggestion:** This decision table (Figure 2.27) 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 category (carnivorous, omnivorous, vegetarian), along with intolerance filters (lactose, gluten, and nuts). 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. In addition, this decision table requires two input parameters to refine and tailor the suggestions:
  - Calorie_Conscious_Level: a custom integer attribute ranging from 0 to 3 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.

      $$TotalCalories > 650$$

    - 1: the guest has a low concern for calories but prefers meals that are not excessively high in energy.

      $$450 < TotalCalories < 650$$

Ingredients

C	inputs				outputs		annotations
	Guest Category	is Lactose Intolerant	is Gluten Intolerant	is Nut Intolerant	Ingredients		
					Name	Calories	
	Text	*Boolean*	*Boolean*	*Boolean*	*Text*	*Number*	
1	"vegetarian","omnivore"	-	-	-	"Apple"	35	
2	"vegetarian","omnivore"	-	-	-	"Arugula"	3	
3	"vegetarian","omnivore"	-	-	-	"Banana"	40	
4	"vegetarian","omnivore"	-	-	-	"Capsicum"	11	
5	"vegetarian","omnivore"	-	-	-	"Eggplant"	18	
6	"vegetarian","omnivore"	-	-	-	"Mushroom"	15	
7	"vegetarian","omnivore"	-	-	-	"Olive"	47	
8	"vegetarian","omnivore"	-	-	-	"Onion"	13	
9	"vegetarian","omnivore"	-	-	-	"Potato"	450	
10	"vegetarian","omnivore"	-	-	-	"Strawberry"	22	

Page 1 | Meals Suggestion | Ingredients | All

Figure 2.25: Ingredients Decision Table in Trisotech

	Guest Category	is Lactose Intolerant	is Gluten Intolerant	is Nut Intolerant	Name	Calories	
24					"Steak"	485	
25	"carnivorous","omnivore"	-	-	-	"Tuna"	158	
26	"carnivorous","vegetarian","omnivore"	-	-	false	"Almond"	90	
27	"carnivorous","vegetarian","omnivore"	-	-	-	"Basil"	1	
28	"carnivorous","vegetarian","omnivore"	-	false	-	"Bread"	135	
29	"carnivorous","vegetarian","omnivore"	-	-	-	"Buckwheat flour"	103	
30	"carnivorous","vegetarian","omnivore"	-	-	-	"Buckwheat pasta"	278	
31	"carnivorous","vegetarian","omnivore"	false	-	-	"Butter"	75	
32	"carnivorous","vegetarian","omnivore"	-	-	-	"Chili"	15	

Figure 2.26: Intolerances within Ingredients Decision Table

* 2: the guest is moderately calorie-conscious and prefers balanced meals.

$$250 < TotalCalories < 450$$

* 3: the guest is highly calorie-conscious and prefers low-calorie, fitness-oriented meals.

$$0 < TotalCalories < 250$$

– Course_Type: a string indicating the desired type of meal course, such as "Appetizer", "First Dish", "Second Course", "Main Dish", "Dessert", or "Drink". 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 accompanied by 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 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 evaluate to true, and the calorie threshold (based on Calorie_Conscious_Level) is not exceeded, the meal will be included in the list of recommendations, otherwise, the meal will be null and total calories will be equal 0.

Meals Suggestion

	inputs		outputs		annotations
C	Calorie_Conscious_Level	Course_Type	Meals Suggestion		
			Meals Suggestion	Total Calories	
	tCalorie_Conscious_Level (0..3)	*Text*	*Text*	*Number*	
1	0,1	"Appetizer"	if list contains(Ingredients.Name, "Basil") and list contains(Ingredients.Name, "Oil") and list contains(Ingredients.Name, "Salt") and list contains(Ingredients.Name, "Pepper") and list contains(Ingredients.Name, "Bread") and list contains(Ingredients.Name, "Ham") and list contains(Ingredients.Name, "Mushroom") and list contains(Ingredients.Name, "Mozzarella cheese") and list contains(Ingredients.Name, "Tomato") then "Bruschette" else null	if list contains(Ingredients.Name, "Basil") and list contains(Ingredients.Name, "Oil") and list contains(Ingredients.Name, "Salt") and list contains(Ingredients.Name, "Pepper") and list contains(Ingredients.Name, "Bread") and list contains(Ingredients.Name, "Ham") and list contains(Ingredients.Name, "Mushroom") and list contains(Ingredients.Name, "Mozzarella cheese") and list contains(Ingredients.Name, "Tomato") then sum(for i in Ingredients return if i.Name in ["Basil", "Oil", "Pepper", "Salt", "Bread", "Ham", "Mushroom","Mozzarella cheese", "Tomato"] then i.Calories else 0) else 0	
2	0,1,2	"Appetizer"	if list contains(Ingredients.Name, "Honey") and list contains(Ingredients.Name, "Parmesan") and list contains(Ingredients.Name, "Pecorino") and list contains(Ingredients.Name, "Ricotta") then "Cheese Platter and Honey" else null	if list contains(Ingredients.Name, "Honey") and list contains(Ingredients.Name, "Parmesan") and list contains(Ingredients.Name, "Pecorino") and list contains(Ingredients.Name, "Ricotta") then sum(for i in Ingredients return if i.Name in ["Honey", "Parmesan", "Pecorino", "Ricotta"] then i.Calories else 0) else 0	

Figure 2.27: Meals Suggestion Decision Table in Trisotech

2.3.5 Input Configuration and Result Output

This section presents an overview of the Decision Service model by illustrating both the user input configuration and the corresponding output results, as processed through the linked decision tables. The first image 2.28 shows the input of the decision service,

where the user is required to enter values for the attributes defined in the input columns of both decision tables: Ingredients and Meal Suggestion.

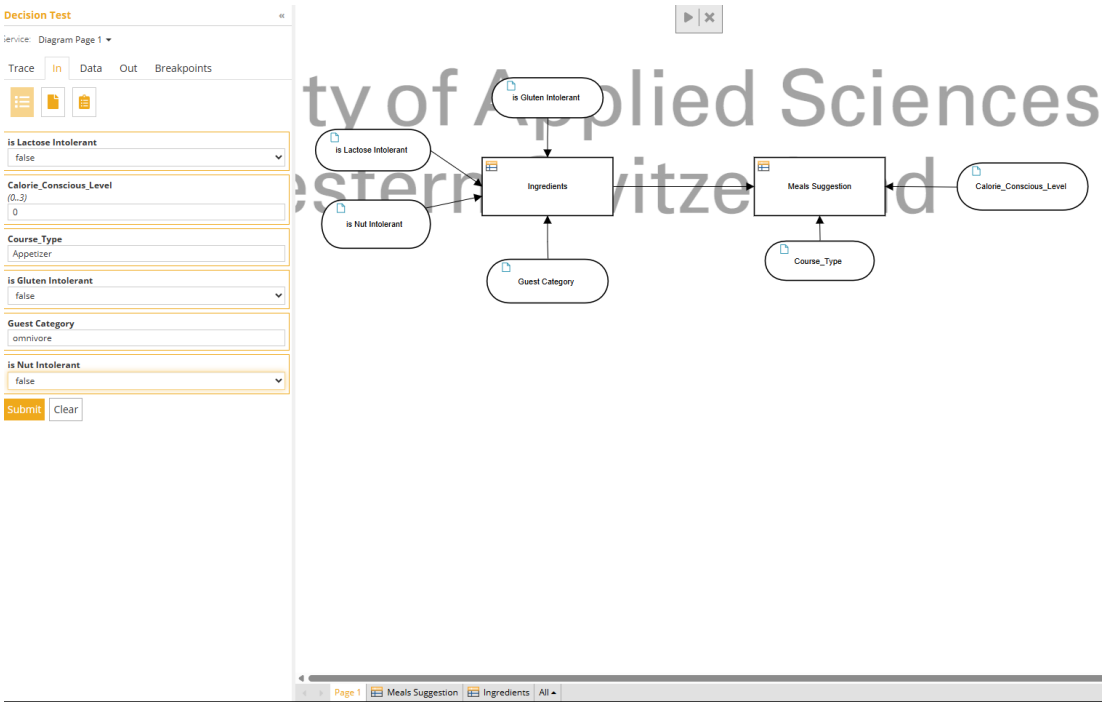
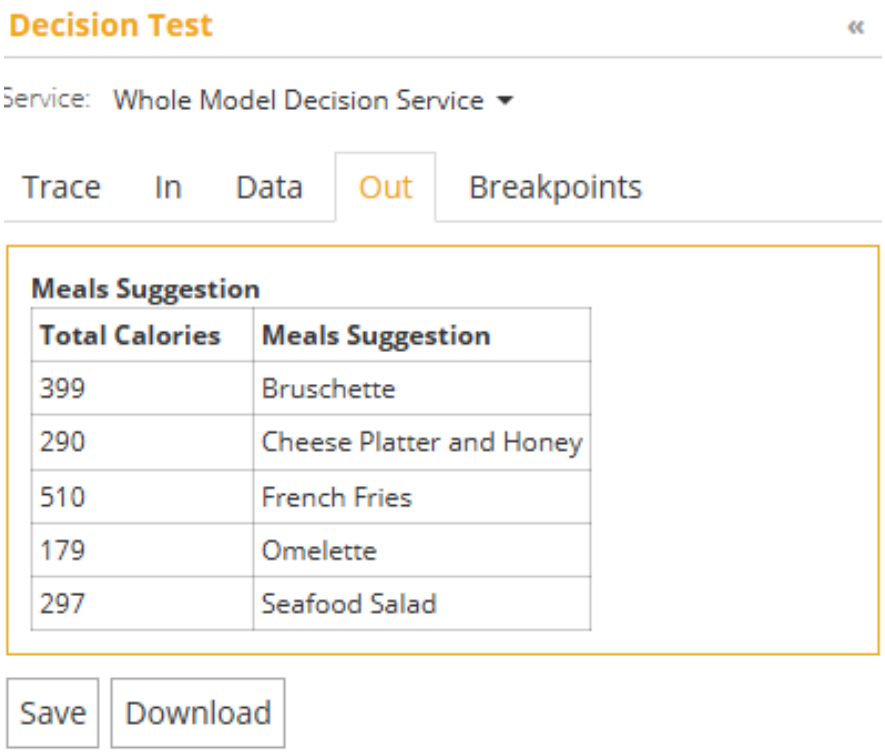
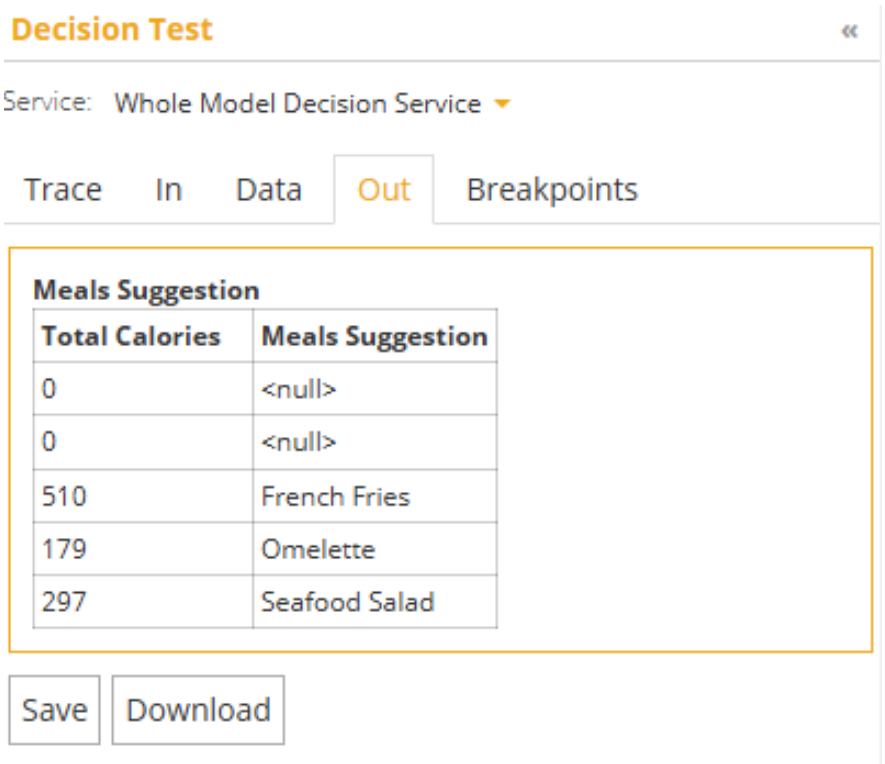


Figure 2.28: Input Form

The second image 2.29a shows the output interface, which displays the final results of the decision evaluation. If we set the Lactose Intolerant value to true the first two meals that include lactose will not be included in the output form as you can see in this figure 2.29b.



(a) Output Form



(b) Output Form with Lactose Intolerance

Figure 2.29: Output comparison between two different configurations

3. Agile and Ontology-based Meta-Modelling

In this chapter, we explore how agile modeling principles and ontology-based meta-modelling were applied to our project. The first section introduces AOAME, a tool that integrates domain ontologies into BPMN 2.0 diagrams. The second section focuses on BPMN 2.0, describing its key elements and how we used it to model the core process of our system. The full implementation is available in the project's GitHub forked repository.[1]

3.1 AOAME

AOAME[2] is a prototypical tool developed to support an agile, ontology-based approach to meta-modelling [Lau24]. This approach focuses primarily on the design and evolution of enterprise knowledge graphs (EKGs), structured representations of domain knowledge that enable advanced reasoning, integration, and analysis of heterogeneous data sources.

One of the main challenges in building and maintaining EKG schemas lies in the need for combined expertise in both ontology engineering and the target application domain. AOAME addresses this by extending traditional meta-modelling techniques with agile principles, allowing for flexible, real-time adaptations of domain-specific modelling languages (DSMLs). This enables domain experts to actively participate in the design cycle, reducing dependency on ontology specialists and improving collaboration throughout the modelling process.

Developed using the Design Science Research methodology, AOAME acts as a prototype for evaluating this agile, ontology-aware framework. It provides an interactive modelling environment where users can extend, modify, or remove modelling constructs, and dynamically update the modelling palette. These changes are handled through a set of meta-modelling operators that automatically generate the corresponding SPARQL queries, ensuring that the underlying RDF triplestore remains synchronized with the graphical interface. This guarantees consistency between human-interpretable models and machine-interpretable ontologies.

From a technical standpoint, AOAME's architecture is based on a Java backend that integrates with Apache Jena Fuseki[3], which serves as the triplestore for storing ontological data. The backend exposes APIs to retrieve and manipulate ontology con-

[1] GitHub Forked Repository Link:
https://github.com/LorenzoPaolucci000/Ontology4ModelingEnvironment.git
[2] AOAME GitHub Repository: https://github.com/BPaaSModelling/AOAME.git
[3] Apache Jena Fuseki: https://jena.apache.org/documentation/fuseki2/

tent, while the graphical user interface (GUI) displays the modelling environment and supports user interaction. The tool's dynamic features support the agile adaptation of DSMLs without requiring a complete redefinition of the schema, thus promoting incremental, iterative development.

AOAME has been validated through several real-world case studies, demonstrating its flexibility and applicability across diverse domains. By enabling continuous collaboration between domain experts and language engineers, the tool enhances both the adaptability and effectiveness of the meta-modelling process, making it particularly well-suited for environments where knowledge evolves rapidly and system requirements are subject to change.

3.2 BPMN 2.0

Business Process Model and Notation (BPMN) 2.0 is a standardized graphical language developed by the Object Management Group (OMG) for modeling business processes in a way that is both understandable to non-technical stakeholders and precise enough for implementation [Sig]. Its primary goal is to bridge the communication gap between business analysts and developers by offering a common notation that is simple yet expressive.

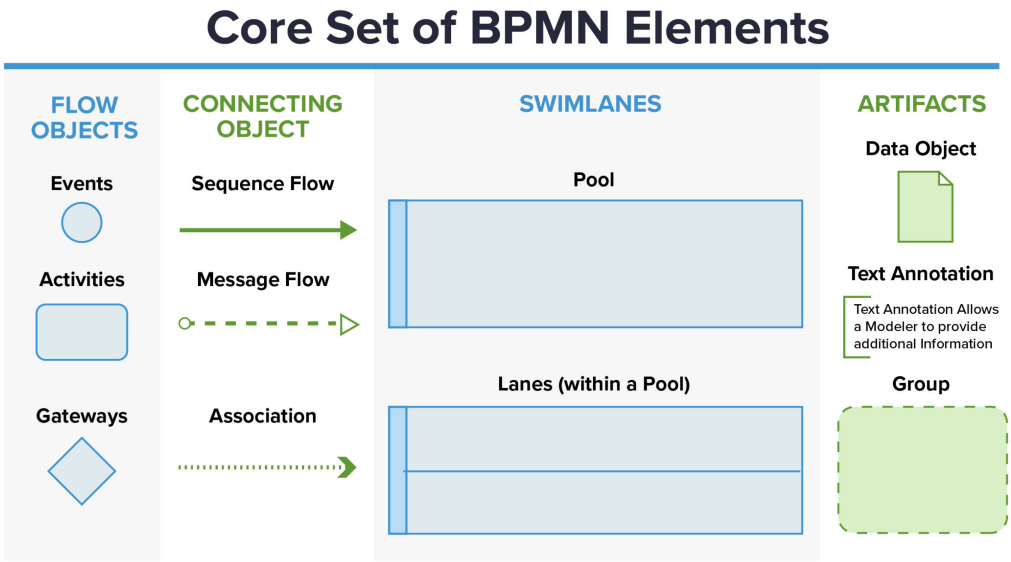


Figure 3.1: BPMN 2.0 Core Elements [Pro23]

BPMN 2.0 includes a set of standardized elements organized into key categories:

- **Flow Objects**, which define the core structure of the process [Pro23].

  These include:

  - **Events:** such as start, intermediate, and end events, which signal something that happens within the process, like a trigger or a result. They often mark the beginning or completion of a process flow.

  - **Activities:** represent tasks or work to be performed. These can be atomic or decomposed into subprocesses, and are essential for modeling operational logic.
  - **Gateways:** control the flow based on conditions or events. They define points where paths diverge or converge and help manage process complexity.

- **Connecting Objects**, which define the relationships and interactions between elements:
  - **Sequence Flow:** indicate the execution order of the activities.
  - **Message Flow:** show communication between different pools or participants.
  - **Associations:** link artifacts like data objects or annotations to elements. These connectors help define both control logic and data interaction.

- **Swimlanes**, which organize the model based on roles or organizational units:
  - **Pools:** represent participants in a process (e.g., departments, systems, environments).
  - **Lanes:** subdivide pools to show internal responsibilities. This structure clarifies who is responsible for each step in the process.

- **Artifacts**, which provide supplementary information that enhances understanding:
  - **Data Objects:** represent the information consumed or produced by activities.
  - **Groups:** can be used to visually cluster related elements.
  - **Annotations:** offer textual explanations to improve readability and documentation.

In our project, we used BPMN 2.0 to model the Meal Suggestion process within a restaurant scenario. This model represents the logical flow from the collection of user preferences to the generation of personalized meal suggestions based on the ontology and reasoning modules described in previous chapters. The use of BPMN allowed us to describe the system in a clear and structured way, facilitating communication among team members and improving the alignment between the design and implementation phases.

3.2.1 Our BPMN 2.0 Model

The figure 3.2 presents the BPMN model we developed to represent the Meal Suggestion System within a hypothetical restaurant scenario. The model is structured as a pool divided into two lanes: Guest and Restaurant, clearly delineating the responsibilities of each participant.

The process is initiated by the guest, who begins by scanning a QR code provided at the table. This action leads to the submission of personal preferences and individual characteristics, such as dietary category, allergies, and calorie-conscious level.

Once the guest's input is received, the restaurant lane processes this information by invoking a custom task called Meal_Suggestion. This task represents the core logic

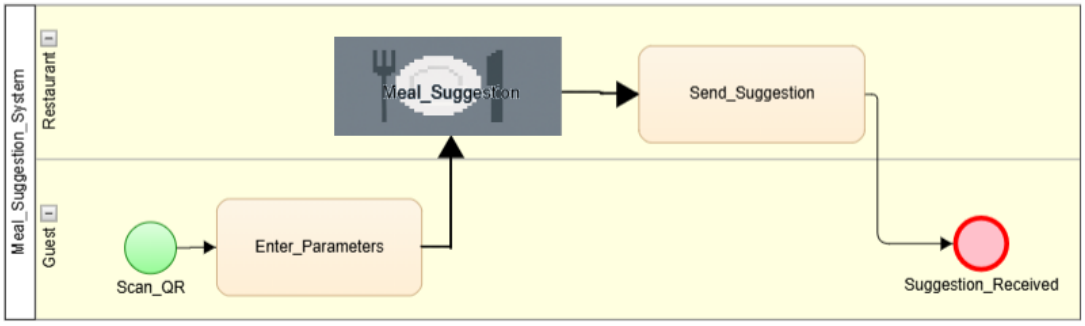


Figure 3.2: BPMN 2.0 Model built in AOAME

of the system, as it determines which meals are suitable for the guest based on the provided parameters. The reasoning considers dietary restrictions, allergen avoidance, and nutritional preferences 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 dining experience.

After designing the BPMN 2.0 model, we used Apache Jena Fuseki to execute SPARQL queries and explore the structure and content of the underlying RDF knowledge graph generated by our ontology-based system. Apache Jena Fuseki is a robust and scalable SPARQL server that facilitates the efficient handling of RDF data. It offers a RESTful interface that allows users to retrieve data through SPARQL queries, apply updates to modify existing triples, and manage RDF datasets by loading, storing, or organizing them programmatically. In our project, Fuseki was essential for testing and validating the semantic correctness of the information produced by the Meal_Suggestion task, ensuring that the system returned consistent and relevant results based on user preferences.

The SPARQL query shown in Listing 3.1 is designed to retrieve a set of personalized meal suggestions for a restaurant guest based on their individual preferences and dietary restrictions. It begins by extracting the guest's selected category (such as Vegetarian, Carnivorous, or Omnivore), calorie-consciousness level, allergy type, and preferred course (e.g., Main Dish or Dessert) from a specific instance of the Meal_Suggestion task. These input values are then dynamically mapped to the corresponding ontology concepts using BIND expressions, allowing the query to reason over structured RDF data. The query searches for meals that match the calorie-conscious level of the guest, belong to the appropriate food category, and correspond to the requested course type. In addition, it ensures that the suggested meals do not contain ingredients associated with the guest's allergy, if specified.

```
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/andrea.mazzetti-lorenzo.
    paolucci/kebi2025#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
```

```
10  mod:Meal_Suggestion_f907089d-8120-4f63-b6a1-81446d9edc8f lo:
        guest_hasCategory ?category .
11  mod:Meal_Suggestion_f907089d-8120-4f63-b6a1-81446d9edc8f lo:
        guest_hasCalorieConsciousLevel ?levelCC .
12  mod:Meal_Suggestion_f907089d-8120-4f63-b6a1-81446d9edc8f lo:
        guest_hasAllergy ?allergy .
13  mod:Meal_Suggestion_f907089d-8120-4f63-b6a1-81446d9edc8f lo:
        guest_hasCoursePreference ?course .
14
15  BIND(IF(?category = "Vegetarian", kebi:category_vegetarian,
16           IF(?category = "Carnivorous", kebi:category_carnivorous,
17             IF(?category = "Omnivore", kebi:category_omnivore, ?
                   category))) AS ?finalCategory) .
18
19  BIND(IF(?course = "Appetizer", kebi:course_appetizer,
20           IF(?course = "First Dish", kebi:course_first-dish,
21              IF(?course = "Main Dish", kebi:course_main-dish,
22                 IF(?course = "Second Course", kebi:course_second-
                       course,
23                    IF(?course = "Sidedish", kebi:course_sidedish,
24                       IF(?course = "Dessert", kebi:course_dessert,
                            ?course)))))) AS ?finalCourse) .
25
26  BIND(IF(?allergy = "Gluten", kebi:grain_containsGlutenIntolerance,
27              IF(?allergy = "Lactose", kebi:
                       dairy_containsLactoseIntolerance,
28              IF(?allergy = "Nut", kebi:
                       nutProduct_containsNutIntolerance,
29                 IF(?allergy = "none", "none", ?allergy)))) AS ?
                       finalAllergy) .
30
31  ?meal a kebi:Meal .
32  ?meal kebi:meal_hasLevelOfCalorieConscious ?kcal .
33  FILTER(?kcal = ?levelCC)
34
35  ?meal kebi:food_hasCategory ?finalCategory .
36  ?meal kebi:meal_hasCourse ?finalCourse .
37
38  OPTIONAL {
39    ?meal kebi:meal_hasIngredient ?ingredient .
40    ?ingredient rdf:type ?ingredientType .
41    FILTER ((?finalAllergy = "none") ||
42            (?finalAllergy = kebi:grain_containsGlutenIntolerance &&
                    ?ingredientType = kebi:Grain) ||
43            (?finalAllergy = kebi:dairy_containsLactoseIntolerance
                   && ?ingredientType = kebi:Dairy) ||
44            (?finalAllergy = kebi:nutProduct_containsNutIntolerance
                   && ?ingredientType = kebi:NutProduct))
45  }
46  FILTER (!BOUND(?ingredient))
47 }
```

Listing 3.1: SPARQL query for personalized Meal Suggestion

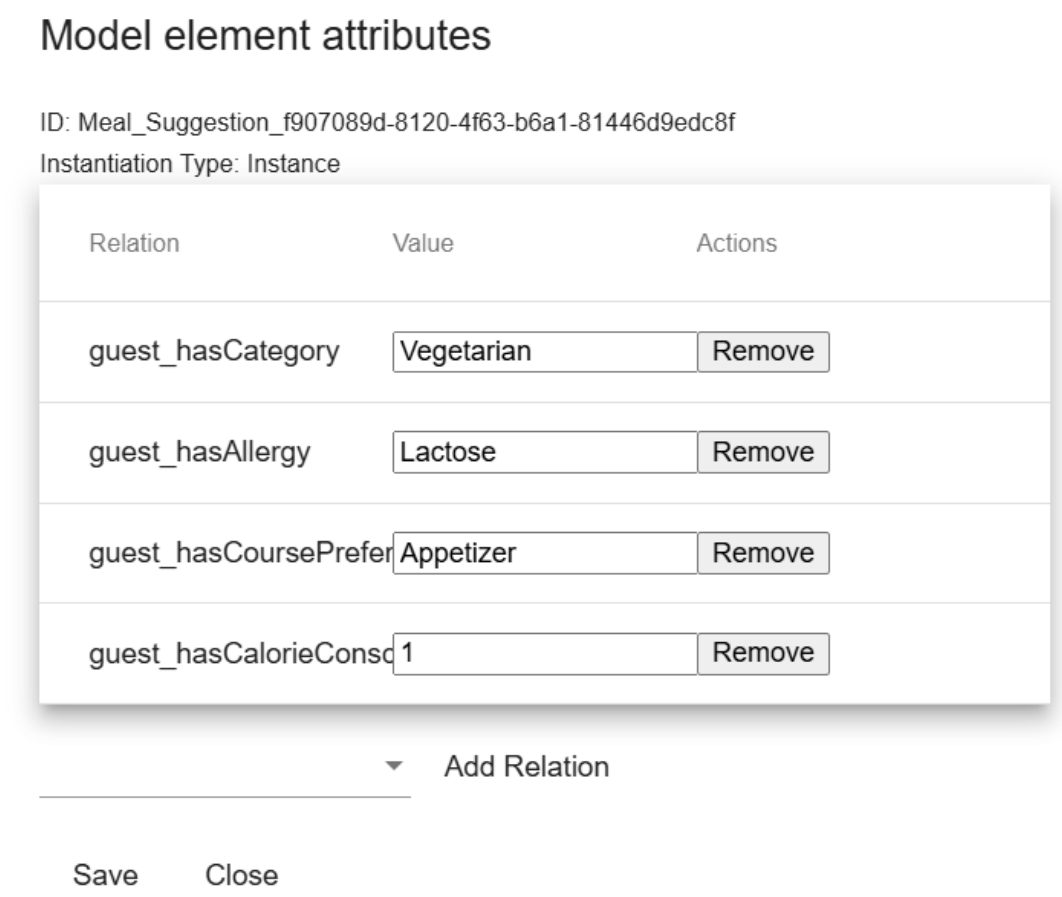
To validate the behavior of our BPMN model and the underlying query logic, we configured the attributes of the custom Meal_Suggestion task using three different guest

profiles.

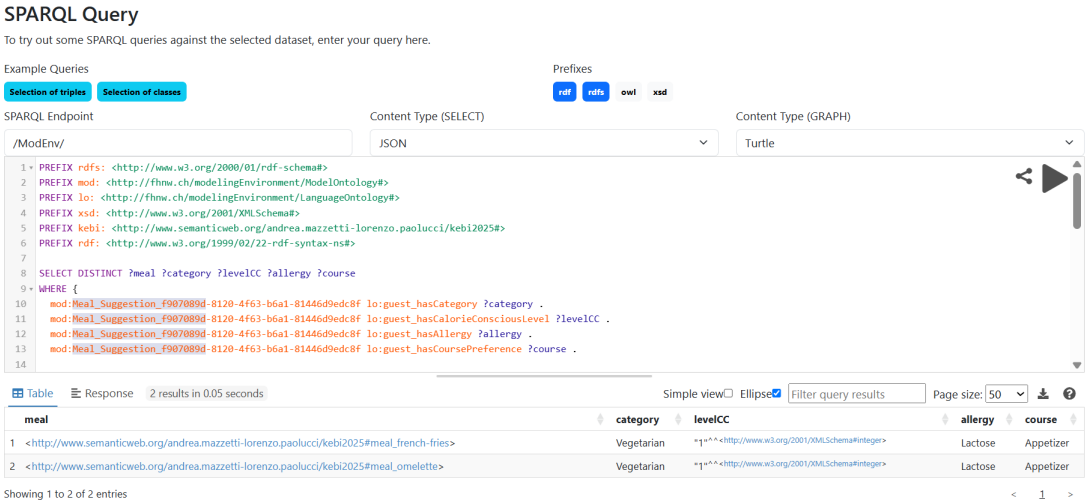
In the first scenario (Figure 3.3), the guest is vegetarian, lactose intolerant, wants an appetizer, and has a calorie-conscious level of 1. As shown in the corresponding output image 3.3b, the system returns only 2 suitable appetizers out of 5. The remaining options are excluded because they either contain lactose, include ingredients like fish (which are not aligned with a vegetarian diet), or exceed the guest's specified calorie sensitivity.

In the second configuration (Figure 3.4), the guest is omnivorous, has a gluten intolerance, prefers a first dish, and has a calorie-conscious level of 2. In this case 3.4b, the query returns only one valid result: spaghetti with garlic, oil, and chili, which is prepared using gluten-free buckwheat pasta. Other pasta-based meals are excluded either due to their higher caloric content or the presence of gluten.

In the final example (Figure 3.5), the guest is carnivorous and nut intolerant, prefers a second course, and has no calorie restrictions (calorie-conscious level set to 0). The output 3.5b, shows 3 out of 5 possible second courses. The system excludes a salmon dish garnished with pistachios, due to the nut intolerance, and a vegetable-based ratatouille, as it does not align with a carnivorous diet.

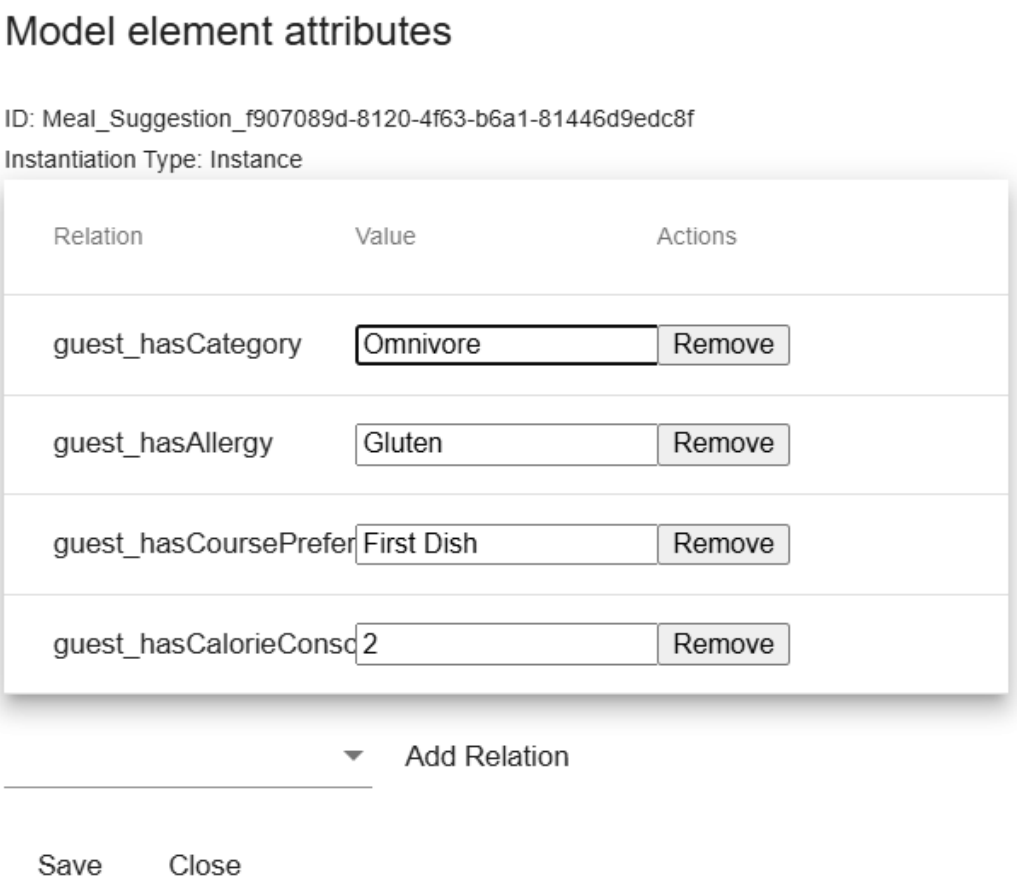


(a) Meal_Suggestion Attributes Configuration

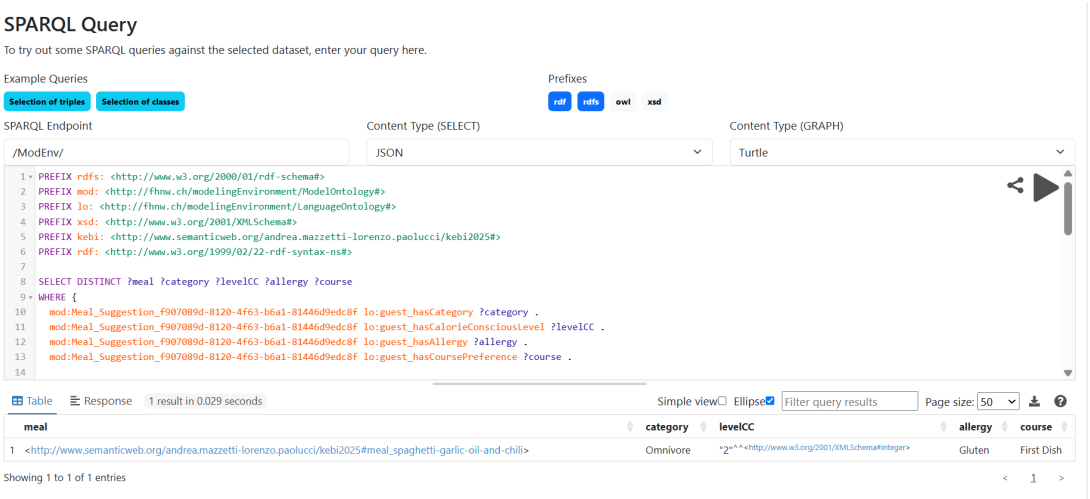


(b) Query Output

Figure 3.3: 1° Example

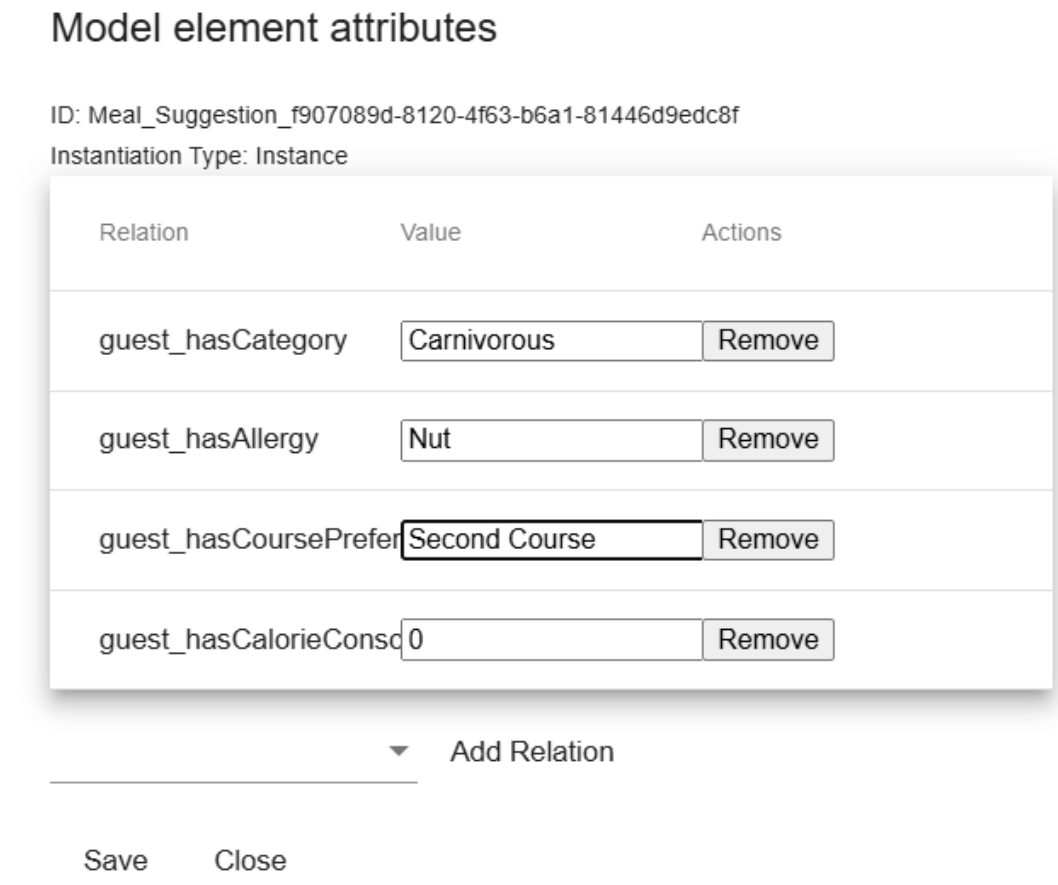


(a) Meal_Suggestion Attributes Configuration

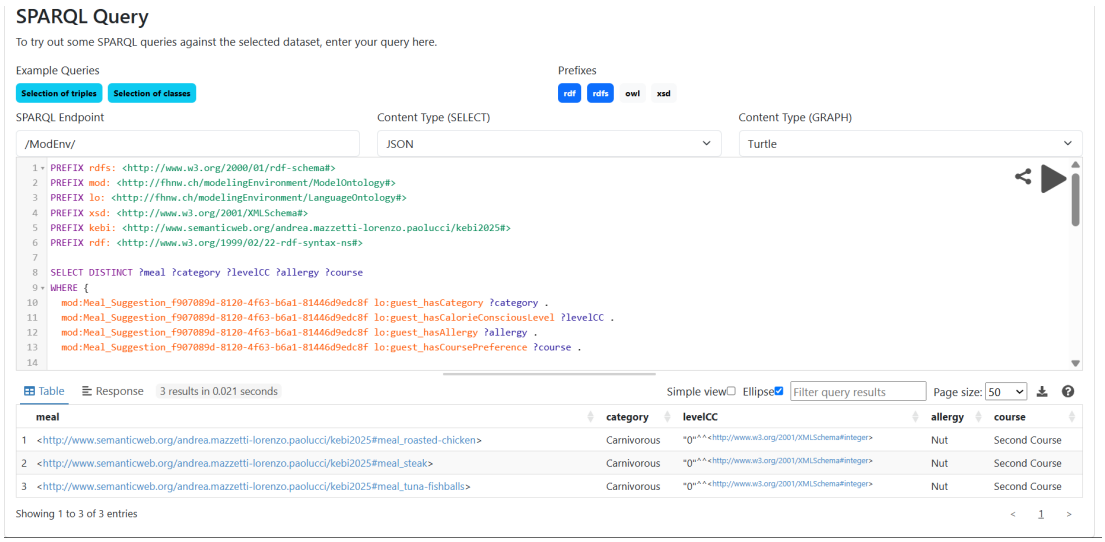


(b) Query Output

Figure 3.4: 2° Example



(a) Meal_Suggestion Attributes Configuration



(b) Query Output

Figure 3.5: 3° Example

4. Conclusions

In this final chapter, each team member reflects on the work carried out throughout the project, offering individual perspectives on the tools, methodologies and tasks developed. We discuss the strengths and challenges encountered during implementation, as well as the overall effectiveness and applicability of the proposed solutions within the context of knowledge-based systems and agile and ontology-based meta-modelling.

4.1 Andrea Mazzetti

This project made me very stimulated, enriching my experience and continuously challenging me to solve the problems that occurred. In particular, some steps of Task 1 have been very hard to make and have kept my interest very high to understand how the tool works and how I can obtain the best outcome. So the knowledge of this topic not only deepened my understanding of the subject matter, but also improved my problem-solving abilities, resilience, and capacity, allowing me to find out different ways to solve the problem. These are the reasons that make the project very rewarding to do.

4.1.1 Decision Tables

Trisotech's Decision Table tool has been an instructive experience, particularly in the context of modeling business logic and decision-making processes in a structured and transparent way. The platform offers a user-friendly interface for creating DMN (Decision Model and Notation) decision tables, which can be used to create complex decision logic readable by humans and executable by machines.

The interface is well-designed and visually intuitive, making it easy to define inputs, outputs, and rules. The tool also provides real-time validation of rules, maintaining the correctness of the decision logic and alerting the user to incomplete entries, overlapping rules, or inconsistencies.

Although the tool is intuitive, the first time it has been difficult to understand the DMN concepts, and it has required me time to study the correct uses of those. Another problem has been the type of the table that made it often difficult to manage or read when its dimensions became high, reducing clarity and increasing the risk of errors. Even the small community has been a problem because searching for some solution, nothing useful has been found.

4.1.2 Prolog

Prolog has been the first approach that I had for this project and has been very funny to do. The language is very interesting, particularly due to its unique approach to

problem-solving through logic-based paradigms.

Unlike imperative languages that rely on sequences of instructions, Prolog is declarative, allowing the developer to define what the program should realize rather than how. This aspect encourages a different way of thinking, more focused on formal logic and relationships between data. Through the use of facts, rules, and queries, Prolog allows for the representation of complex relationships and the modeling of knowledge in a clear way.

This makes Prolog not only an excellent tool for exploring fields such as artificial intelligence, natural language processing, and knowledge representation, but also a powerful mental exercise that refines logical thinking and problem decomposition skills, making it rewarding and creatively satisfying.

Despite this, at the beginning it has been difficult to understand how the language works with its logical relationships and rules, making it sometimes less intuitive. Another problem has been the small ecosystem, considering that it has a small community that makes searching difficult or useless for some issues.

4.1.3 Protégé

The first approach that I had about Protégé has been very confusing because I did not know how to use the tool and where I had to start. After some study to understand how I had to move, the next step of defining classes, properties, and individuals has been quite easy.

I have found the SHACL validator very useful to test the correct construction of the ontology and SPARQL to get data from queries, interesting even to create a personalized menu, for example, based on the preferences of the guests. SWRL rules have been useful in Protégé to infer the type of the meal based on the ingredients it was made, and this has been an important aspect that has helped to find the correct meal for the guest using SPARQL queries.

One of the negative aspects has been the long time spent to create all the data because there is nothing that makes the creation of data more quickly, but you have to create the data one by one and write the content every time. About SWRL rules, the problem was in GraphDB, which lacks built-in reasoning capabilities for type inference, so it was not possible to use the queries that need reasoner to find the infered meals.

4.1.4 AOAME

I found AOAME a very interesting tool that we used to create BPMN 2.0 diagrams, allowing us to have a graphical visual of how our ontology works. The main part of this tool has been the creation of Meal_Suggestion, a graphical notation where the guest can compile category, allergy, course, and calorie-conscious attributes to get a personalized menu. This feature can help the restaurant make the menu fast and very simple to filter.

About Jena Fuseki, it has been useful to integrate our ontology into a triplestore and execute SPARQL queries to generate a personalized menu dynamically based on the guests' preferences. It has been interesting the power of Jena Fuseki and the integration with AOAME and with our ontology.

I have not found a particular problem about Task 2 and about using AOAME and Jena Fuseki; both tools are quite intuitive. Maybe at the beginning they can make the

user confused, but with a few times it is easy to understand how they work, and it is interesting the potential of them.

4.1.5 Final Considerations

All those tools have challenged me to understand and study the features that they give. Trisotech's Decision Table has learned me to represent decision rules in a clear, structured and understandable way, automating complex choices through conditions and outputs. Prolog taught me the basics of logic programming and how to represent knowledge through facts and rules. It is a useful tool for deductive reasoning. Protégé learned me to create an ontology defining classes, properties, individuals and relationships, and structuring knowledge in a semantic way to encourage sharing and automatic reasoning. Jena Fuseki taught me how to interrogate RDF knowledge bases via SPARQL while AOAME taught me how to model an ontology.

In conclusion, I have tried to create a restaurant menu that can be personalized considering different types of guests or their preferences. It has been very funny creating this menu and understanding how to make it easier for the guests to read it. All those tools are valid for the creation of a personalized menu, and the choices of one of them depend on the type of customization desired and the complexity of the knowledge base.

4.2 Lorenzo Paolucci

During this project, I had the opportunity to work closely with a variety of tools and technologies in the field of knowledge engineering. This experience allowed me to explore different paradigms and modeling approaches, each with its own practical applications and learning opportunities. What struck me the most was the versatility of these tools and their ability to support intelligent and adaptable systems across different layers of abstraction, from logic programming to visual modeling. In this section, I will share my personal impressions of the solutions we explored.

4.2.1 Decision Tables

Among all the tools used, Decision Model and Notation (DMN) through the Trisotech platform was the one I engaged with the most. I was immediately drawn to its structured and user-friendly interface, which allowed for the clear definition of rules through tabular logic. The intuitive layout of inputs and outputs, combined with the "Collect" hit policy, made it easy to evaluate multiple rules in parallel and aggregate results effectively.

I found DMN especially useful for representing business logic transparently, making it accessible even to users with non-technical backgrounds. The combination of logical structure and graphical representation made the tables highly maintainable and easy to verify. However, I also observed that as the complexity of the decision model increased with more inputs, conditions, and rule combinations the process of manually managing rows and entries became somewhat repetitive and time-consuming. Despite this, DMN remains a very effective tool for declarative decision-making, and I see great potential in its application to a wide range of domains, especially those requiring traceable and auditable logic.

4.2.2 Prolog

Exploring Prolog introduced me to the declarative programming paradigm, which offered a different way of thinking about problem-solving. I enjoyed experimenting with recursion and rule-based inference, which allowed for a very natural expression of logical relationships. The language encouraged a new way of thinking, focusing not on how to perform a task step by step, but rather on what conditions must be satisfied for something to be considered true. Although some aspects of the language required extra effort, especially in the absence of mainstream learning resources, I found Prolog to be very effective for managing and querying knowledge bases, particularly in scenarios where logic plays a central role.

4.2.3 Protégé

I found Protégé to be a powerful and structured environment for modeling ontologies. It felt like a bridge between decision logic and object-oriented thinking, with the added benefit of using SPARQL queries, which were familiar and intuitive thanks to their similarity with SQL. I especially appreciated the SHACL validator, which helped us test the structure and consistency of our ontology. This feature reminded me of unit testing in traditional programming and proved useful in maintaining the consistency of our models. The use of SWRL rules allowed for deeper inference over the modeled knowledge, and while they require attention when used alongside external tools (e.g., AOAME, GraphDB), they added valuable reasoning capabilities to our system.

4.2.4 AOAME

Among the tools we used, AOAME stood out for its ability to combine BPMN 2.0 diagrams with semantic models. The visual integration of ontology concepts within process diagrams was particularly impactful, especially when defining user-centric tasks such as Meal_Suggestion. This approach made our models both expressive and understandable for stakeholders. By integrating AOAME with Apache Jena Fuseki, we could query our ontology dynamically through SPARQL, enabling personalized meal recommendations. Despite some occasional performance issues in the online version, the tool showed great potential for combining process modeling and semantic reasoning in an intuitive way.

4.2.5 Final Considerations

In summary, the development of this project allowed us to explore and compare multiple knowledge-based solutions, each offering distinct advantages depending on the context of use. DMN decision tables proved to be intuitive and easy to implement, especially for structured rule-based logic, though they became repetitive and less flexible in more complex scenarios. Prolog provided a powerful and logical way to query knowledge, but its scalability and learning curve could be limiting factors. Protégé offered a robust framework for modeling structured knowledge with strong querying capabilities, requiring careful rule and reasoning management. Lastly, AOAME stood out for its ability to visually integrate ontologies and processes, although small technical issues.

Overall, the project highlighted how the choice of tools should be guided by the specific requirements of the domain, the complexity of the knowledge involved, and the desired level of flexibility and maintainability. Combining these approaches allowed us

to build a well-rounded, adaptable system that balances usability, expressiveness, and reasoning power.

In conclusion, my understanding of knowledge engineering has grown as a result of this project, which has also demonstrated how these technologies can help develop intelligent, flexible systems for real-world uses, like those in the restaurant industry.

Bibliography

[Gra] *GraphDB*. URL: https://graphdb.ontotext.com/.

[Lau24] Emanuele Laurenzi. "Agile, Ontology-Based Meta-Modelling for Designing Enterprise Knowledge Graph Schemas". In: *Enterprise Modelling and Information Systems Architectures (EMISA Journal)* 19 (2024), pp. 1–29. DOI: 10.18417/emisa.18.310. URL: https://emisa-journal.org/emisa/article/view/310.

[Obj16] Object Management Group. "DMN 1.1 Specification". In: (2016). URL: https://www.omg.org/spec/DMN/1.1/PDF.

[Proa] *Prolog*. URL: https://en.wikipedia.org/wiki/Prolog.

[Prob] *Protégé*. URL: https://protege.stanford.edu/.

[Pro23] ProcessMaker. *What is the BPMN 2.0 Standard?* Accessed June 2025. 2023. URL: https://www.processmaker.com/blog/what-is-the-bpmn-2-0-standard/.

[Sig] Signavio GmbH. *BPMN 2.0 for Efficient Process Design*. Accessed June 2025. URL: https://www.signavio.com/bpmn-2-0-for-efficient-process-design/.

[Swi] *SWI-prolog*. URL: https://www.swi-prolog.org/features.html.