Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe LM-18)

Project "Personalized Menu"

Laureando

Luzi Lorenzo 126749

Contents

List of Figures

1. Introduction

This document outlines the development of a Knowledge Engineering and Business Intelligence project. Section 1.1 describes the system being developed, while Section 1.1.1 defines the tasks to be accomplished. The report then details the System Design phase, the implemented Knowledge-Based Solutions, and Agile and Ontology-based Meta-Modelling. The final chapter presents conclusions and reflections on the solutions provided.

1.1 Project Description

The aim of this project is to develop a system that can intelligently filter and present menu items based on individual guest preferences. This involves representing detailed knowledge about the meals offered and the various dietary needs of guests. The knowledge base will include meal, such as pizza, pasta, and main courses, with each meal broken down into its constituent ingredients. These ingredients will be categorized (e.g carnivorous, vegetarian, omnivore) and will include nutritional information like calorie content and allergy.

Guests can have diverse dietary profiles: some may be carnivores, others vegetarians, some may be calorie-conscious, and some may have allergies (such as lactose or gluten intolerance). The system will cater to these profiles, enabling a personalized dining experience by displaying only those meals that align with the guests' preferences and dietary requirements. In order to achieve this goal we defined knowledge-based solutions using differnt language like: Camunda, Prolog and Protégé to help with the selection task. With the ontolgy created during the definition of a knowledge-based solution we implemented an ontology-base meta-modelling solution.

1.1.1 Project Tasks

The project is divided into two main tasks:

1. Develop multiple knowledge-based solutions to recommend meals based on user profiles, using the following representation methods:
    - Decision Tables (including Decision Requirement Diagrams and decision tables)
    - Prolog (including facts and rules)
    - Knowledge Graph/Ontology (with SWRL rules, SPARQL queries, and SHACL shapes)

    The task 1 can be found here

2. Implement an agile and ontology-based meta-modelling approach to adapt BPMN 2.0 for meal recommendation. The ontology created in Task 1 can be extended for this purpose. A graphical notation should be designed for restaurant managers, making it easier to interpret results. The system should use a triple store interface (e.g., Jena Fuseki) to query and retrieve recommendations.
   The second task can be found here

2. System Design

This chapter presents the system's input and output formats, which serve as the foundation for data processing and meal suggestion. This step is used in order to create a starting point before to start with the first task.

2.0.1 Input Format

The system utilizes an input form designed with Google Modules to collect user preferences. Customers fill out this form to specify their dietary preferences and restrictions. The form includes:

- Dietary category selection: Carnivorous, Vegetarian, or Omnivore (single-choice option).
- Allergy information: Lactose intolerance, Gluten intolerance (multiple-choice option).
- Calorie-consciousness level: Ranging from 0 (not calorie-conscious) to 3 (strictly calorie-conscious).
- Course preference: Appetizer, First Course, Main Course, Second course, Pizza, Drink, or Dessert.

Once completed, this information is processed to generate personalized meal suggestions.

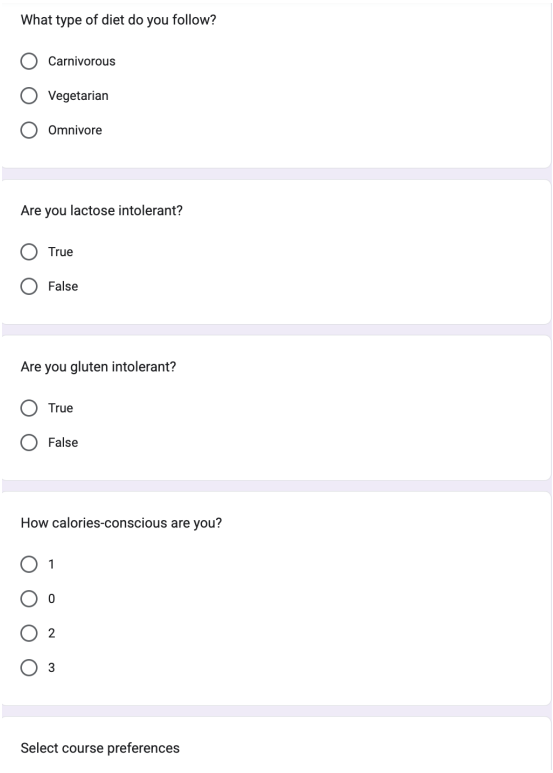

Figure 2.1: Google Modules input form example.

2.0.2 Output Format

Based on the user's input, the system returns a personalized meal list matching their preferences. The output format includes:

- A filtered list of meal options.
- Meals categorized by course type.
- Highlighted meals that align with dietary restrictions and calorie preferences.

An example of the output is the picture below

Figure 2.2: Hypothetical output based on customer preferences

3. Decision Model and notation

DMN, or Decision Model and Notation, is a standard for describing, creating, and visualizing decision-making processes within a business. Its primary purpose is to make decisions comprehensible to everyone involved. Essentially, DMN provides a simplified and standardized language for business processes, ensuring they are easily understandable and replicable across different companies or roles within the same organization, including upper management and specialists.

3.1 DMN Elements

The DMN [1] standard consists of several key elements:

- **Decision Requirements Diagrams (DRD)**: Illustrate dependencies between decision-making components.
- **Decision Tables**: Provide a structured format for defining conditions and corresponding actions.
- **Business Context**: Defines the external factors influencing the decision-making process.
- **Friendly Enough Expression Language (FEEL)**: A language used to evaluate expressions within decision tables.

3.2 Camunda

Camunda[2] is an open-source Java-based platform for workflow and decision automation. It enables the creation of BPMN process diagrams and DMN decision tables. Camunda also provides REST APIs for seamless integration with non-Java applications.

3.3 Our Decision Model

The decision model implemented in this project consists of two Decision Tables and one Literal Expression. The model first applies restrictions to ingredients, followed by constraints on meal selection.

[1] DMN
[2] Camunda link

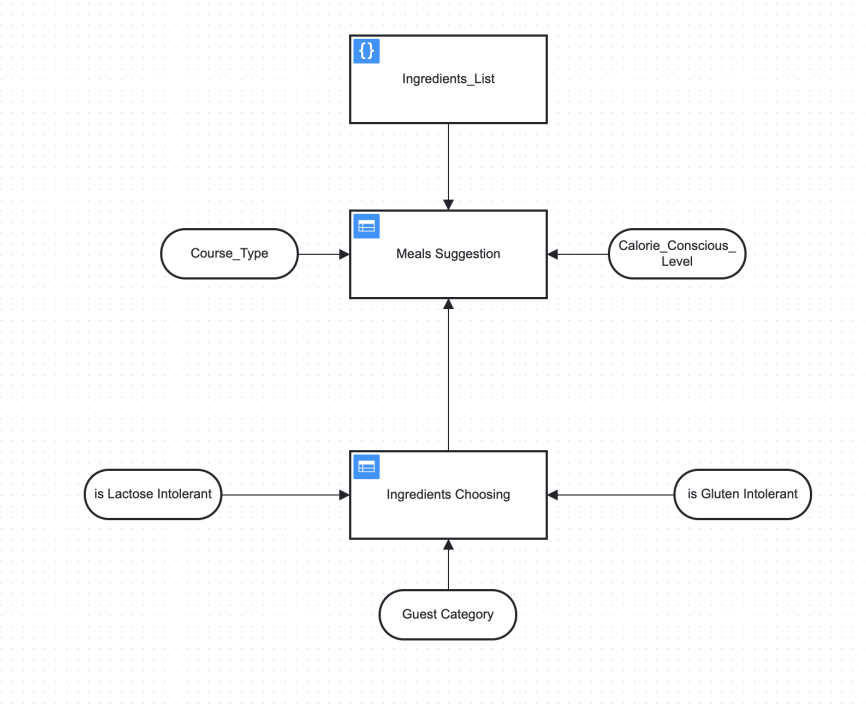

Figure 3.1: Decision Model implemented in Camunda.

Decision Tables

The system takes several input parameters to customize meal suggestions based on user preferences and dietary restrictions. One key input is the user's dietary category, which can be classified into three options:

- Carnivorous – Users who primarily consume meat and do not eat vegetables.
- Vegetarian – Users who avoid meat entirely.
- Omnivorous – Users who consume both meat and vegetables.

Additionally, the system considers two boolean parameters:

- Lactose Intolerance – A value of true indicates that the user is lactose intolerant, while false means they can consume dairy products.
- Gluten Intolerance – Similarly, a true value signifies gluten intolerance, whereas false means gluten can be included in meals.

The decision table processes these inputs and returns a filtered list of meals that align with the user's dietary needs. The structure of the table has been designed for flexibility and future expansion. For example, if a Vegan category needs to be introduced, it can be seamlessly added alongside the existing dietary categories, ensuring that only vegan-friendly ingredients are considered.

Regarding the boolean inputs, the decision table uses specific indicators:

A hyphen ("-") means that a particular ingredient is free from allergens. If false is entered, it indicates the presence of an allergen. In such cases, if the user has the corresponding intolerance, the system will exclude meals containing that ingredient

from the recommendations. This structured approach ensures that users receive meal suggestions tailored to their dietary preferences and restrictions, enhancing both safety and satisfaction.

Ingredients Choosing | Hit policy: Collect

	When: Guest Category "carnivorous","vegetarian","omni...	And: is Lactose Intolerant boolean	And: is Gluten Intolerant boolean	Then: 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"	–	–	"Pork chop"	
11	"carnivorous","omnivore"	–	–	"Steak"	
12	"carnivorous","omnivore"	–	–	"Salmon"	
13	"carnivorous","omnivore"	–	–	"Sword fish"	

Figure 3.2: Ingredient Choosing table

This table takes two user inputs: one from the previous table and another from the Literal Expression.

- The first input, Course Type, specifies the type of meal course, such as Appetizer or First Dish.
- The second input, Calorie Conscious Level, represents the user's awareness of calorie intake, with values ranging from 0 to 3.

Based on these inputs, the table returns a filtered list of meals. By default, the Calorie Conscious Level is set to 0, indicating that there is no concern about calorie content.

Meals Suggestion | Hit policy: Collect

	When: Ingredients list	And: Ingredients_List list	And: Calorie_Conscious_Level integer	And: Course_Type string	Then: Meals
1	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients, "Cabbage") and list contains(Ingredients, "Puff Pastry")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Carrot"].calories[1] + Ingredients_List[name="Cabbage"].calories[1] + Ingredients_List[name="Puff Pastry"].calories[1]) <= 250	3	"Appetizer"	"Involtini primav
2	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients, "Cabbage") and list contains(Ingredients, "Puff Pastry")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Carrot"].calories[1] + Ingredients_List[name="Cabbage"].calories[1] + Ingredients_List[name="Puff Pastry"].calories[1]) <= 450	2	"Appetizer"	"Involtini primav
3	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients, "Cabbage") and list contains(Ingredients, "Puff Pastry")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Carrot"].calories[1] + Ingredients_List[name="Cabbage"].calories[1] + Ingredients_List[name="Puff Pastry"].calories[1]) <= 650	1	"Appetizer"	"Involtini primav
4	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients, "Cabbage") and list contains(Ingredients, "Puff Pastry")	–	0	"Appetizer"	"Involtini primav
5	list contains(Ingredients, "Ham") and list contains(Ingredients, "Salame") and list contains(Ingredients, "Ciauscolo") and list	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Ham"].calories[1] + Ingredients_List[name="Salame"].calories[1] +	3	"Appetizer"	"Affettati Misti"

Figure 3.3: Meal suggestion

Literal Expression

A Literal Expression was developed to store ingredient-calorie mappings, enabling dynamic calorie-based filtering.

Ingredients_List


```
[
  {"name": "Oil", "calories": 60},
  {"name": "Eggplant", "calories": 18},
  {"name": "Cow's cheese", "calories": 120},
  {"name": "Sheep's cheese", "calories": 135},
  {"name": "Pasta", "calories": 300},
  {"name": "Shrimp", "calories": 35},
  {"name": "Ciauscolo", "calories": 70},
  {"name": "Salame", "calories": 50},
  {"name": "Ham", "calories": 45},
  {"name": "Carrot", "calories": 27},
  {"name": "Cabbage", "calories": 25},
  {"name": "Puff Pastry", "calories": 150},
  {"name": "Mussel", "calories": 40},
  {"name": "Salt", "calories": 0},
  {"name": "Parmesan", "calories": 85},
  {"name": "Egg", "calories": 80},
  {"name": "Black pepper", "calories": 25},
  {"name": "Pork cheek", "calories": 150},
  {"name": "Garlic", "calories": 3},
  {"name": "Chili", "calories": 15},
  {"name": "Tomato", "calories": 30},
  {"name": "Bechamel", "calories": 105},
  {"name": "Mozzarella cheese", "calories": 120},
  {"name": "Minced Meat", "calories": 120},
  {"name": "Pork chop", "calories": 125},
  {"name": "Steak", "calories": 485},
  {"name": "Calamari", "calories": 50},
  {"name": "Sword fish", "calories": 120},
```


Figure 3.4: Literal Expression for ingredient-calorie mapping.

Camunda Simulation

Unfortunately, at the time of writing this report, the service for testing the decision tables was unavailable. As a result, I was unable to provide or effectively test the decision tables.[3]

[3]Simulator page link

4. Rule-Based System

A rule-based system in computer science is a type of artificial intelligence system that operates on a set of conditional statements (rules) to perform reasoning, decision-making, or problem-solving tasks. These systems are also known as rule-based expert systems or production systems. In general here are the elements of a rule-based system:

- **Rule Base**: The list of rules specific to theri knowledge base;
- **Semantic Reasoner**: Infers information or takes actions based on the interaction between the input and the rule base;
- **Temporary working memory**: for data storing;
- **UI**: to send input and receive output;

4.1 Prolog

Prolog is a declarative programming language and a formal logic programming tool used primarily in artificial intelligence and computational linguistics. It stands for "Programming in Logic" and is based on formal logic, specifically first-order logic and Horn clauses. The prolog syntax is composed by:

- *Simbols*: ,(and) ; (or) :-(if) not (not);
- *Variables*: is a string of letters, digit
- *Facts*: Expression that make declerative statement about the problem domain;
- *Rule*: predicate expression that uses logical implication (:-) to describe relation between facts.
- *Queries*: The Prolog interprete answer queries on the facts and the rules represented in its database.

Backtracking is a technique implemented in Prolog. It is a process where Prolog examines the truthfulness of various predicates by evaluating their correctness. Essentially, it explore different potential paths through backtracking until it discovers a valid route that reach the final destination.

4.1.1 Prolog Implementation

We implemented the model defined in the DMN file using Prolog and tested it on SWISH Prolog6. Similar to Task 1.1, we restructured the Prolog file to first apply

restrictions at the ingredient level before categorizing meals at the meal level. The classification follows these rules:

- If an ingredient is labeled vegetarian and a meal contains that ingredient (without any carnivorous ingredients), the meal is classified as vegetarian.
- Similarly, if a meal contains carnivorous ingredients, it is classified as carnivorous.
- If a meal includes both carnivorous and vegetarian ingredients, it is classified as neither and is instead considered omnivorous (since all carnivorous and vegetarian meals are inherently omnivorous).
- Meals that do not contain either carnivorous or vegetarian ingredients remain accessible to both carnivorous and vegetarian users.

Ingredient Definitions

This section defines the list of ingredients used in various 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(cod).
ingredient(tomato).
ingredient(bechamel).
ingredient(minced_meat).
ingredient(pork_chop).
ingredient(steak).
ingredient(calamari).
ingredient(potato).
ingredient(eggplant).
ingredient(type_0_flour).
ingredient(water).
ingredient(sugar_and_food_coloring).
ingredient(corn).
ingredient(lettuce).
ingredient(apple).
ingredient(banana).
ingredient(strawberry).
```

Caloric Values

Each ingredient is associated with a caloric value to facilitate meal filtering based on calorie-conscious levels.

```
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(cod,84).
kcal_ingredient(tomato, 30).
kcal_ingredient(bechamel, 105).
kcal_ingredient(minced_meat, 120).
kcal_ingredient(pork_chop, 125).
kcal_ingredient(steak, 485).
kcal_ingredient(calamari, 50).
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).
```

Dietary Classifications

Ingredients are classified into categories such as carnivorous, vegetarian, and allergens.

```
ingredient_carnivore(ciauscolo).
ingredient_carnivore(salame).
ingredient_carnivore(ham).
ingredient_carnivore(pork_cheek).
ingredient_carnivore(minced_meat).
ingredient_carnivore(shrimp).
ingredient_carnivore(mussel).
ingredient_carnivore(steak).
ingredient_carnivore(calamari).
```

```
ingredient_vegetarian(carrot).
ingredient_vegetarian(cabbage).
ingredient_vegetarian(eggplant).
ingredient_vegetarian(lettuce).
ingredient_vegetarian(apple).
ingredient_vegetarian(banana).
ingredient_vegetarian(strawberry).
```

```
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).

ingredient_with_gluten_intolerance(type_0_flour).
ingredient_with_gluten_intolerance(pasta).
ingredient_with_gluten_intolerance(puff_pastry).
```

Meal Definitions

Meals are defined along with their associated 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(rigatoni_alla_carbonara, first_dish,
[salt, oil, pasta, sheep_cheese, egg, black_pepper, pork_cheek]).
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(tagliatelle_allo_scoglio, first_dish,
[salt, oil, pasta, shrimp, mussel, calamari, garlic, chili, tomato]).
meal(ramen, first_dish,
[salt, oil, pasta, egg, pork_chop, garlic, black_pepper, water]).
meal(bistecca_alla_fiorentina, second_course, [salt, oil, steak]).
meal(pesce_fritto, second_course, [salt, oil, shrimp, calamari,cod]).
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, mozzarella_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])
```

Meal Classification Rules

Rules are used to classify meals based on their ingredient composition.

```
% 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)).
```

```
% 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),
```

```
18    list_to_set(MealsWithLactoseIntolerance, UniqueMeals),
19    member(Meal-Course, UniqueMeals).
```

Calorie Calculation and Filtering

Rules are implemented to compute meal calorie values and categorize meals based on calorie-conscious levels.

```
1  meal_calories(Meal, Course, TotalCalories) :-
2     meal(Meal, Course, Ingredients),
3     findall(Calories,
4           (member(Ingredient, Ingredients),
5            kcal_ingredient(Ingredient, Kcal),
6            Calories is Kcal),
7           CaloriesList),
8     sum_list(CaloriesList, TotalCalories).
```

Guest Preferences

Meals are filtered based on guest preferences, including dietary restrictions and calorie-conscious levels.

```
1  guest_preferences(Category, CalorieLevel, Allergies, Meal, Course) :-
2     % Filtered meals by Category (carnivorous, vegetarian, omnivore)
3     ( Category = carnivorous -> carnivorous_meal(Meal, Course)
4     ; Category = vegetarian -> vegetarian_meal(Meal, Course)
5     ; Category = omnivore -> omnivore_meal(Meal, Course)
6     ),
7     % Filtered meals by calorie_consciou_level
8     calorie_conscious_levels(Meal, Course, Levels),
9     member(CalorieLevel, Levels),
10    % Filtered meals by allergies
11    ( Allergies = none -> true
12    ; ( Allergies = lactose -> not(meal_with_lactose_intolerance(Meal, Course))
13      ; Allergies = gluten -> not(meal_with_gluten_intolerance(Meal, Course))
14    )
15    ).
```

4.1.2 Prolog Testing

To validate our implementation, we tested various guest preference scenarios and examined the output.[1]

```
1  guest_preferences(carnivorous, 3, lactose, Meal, Course).
```

Below there is the result of the previous query

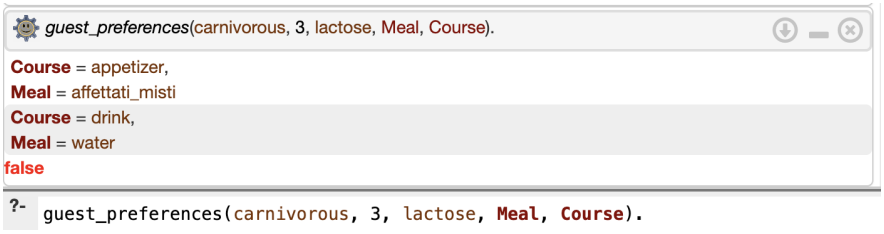

Figure 4.1: Result of prolog testing.

[1] Prolog test online link

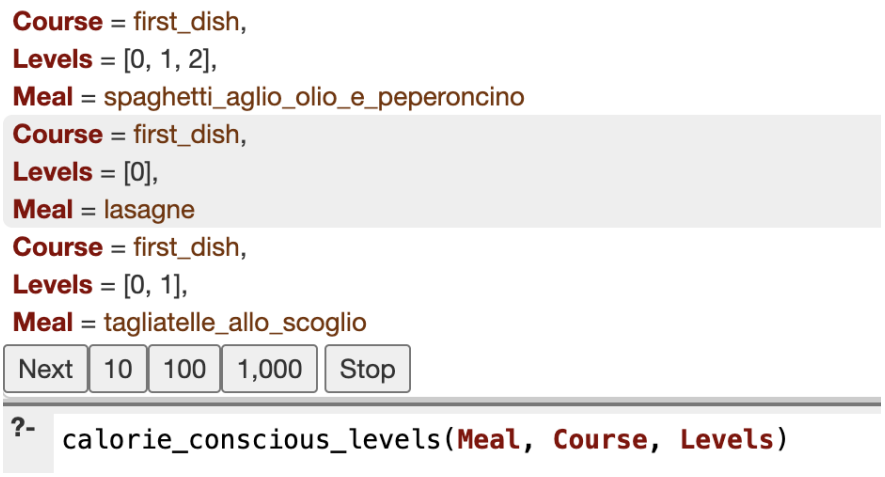

Figure 4.2: Result of prolog testing.

Figure 4.3: Result of prolog testing.

5. Ontology

In the field of Computer Science, ontology engineering focuses on the methodologies used to create and manage ontologies. This involves systematically defining and organizing concepts, their attributes, and the relationships between various entities and data. Different data models, such as RDF, OWL, and Neo4J, can be used to structure ontologies. In our case, we utilized RDF for building our ontology.

5.1 Protégé

Protégé is a free, open-source ontology editor and a knowledge management system. It allows the definition of Semantic Web Rule Language (SWRL) to create inference rules that can generate new data based on existing ontologies. Furthermore, it supports the Sparql Protocol And Rdf Query Language (SPARQL), which is a semantic query language for databases capable of retrieving and manipulating data stored in RDF format.And also the user can define constraints or "shapes" on their RDF data, ensuring that the data adheres to certain rules and structures through the use of SHACL.

RDF

The Resource Description Framework (RDF) is a standardized framework designed for representing and exchanging graph-based data. It allows for the structured description of resources, which are essential in constructing knowledge graphs.

Knowledge Graphs

A knowledge graph represents a network of interconnected entities—such as objects, events, and concepts—emphasizing the relationships between them.

Classes

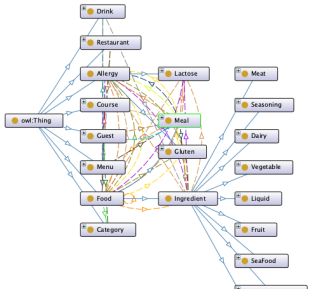

Figure 5.1: Class visual representation

The primary entities defined earlier also serve as our main classes. Additionally, we have established a hierarchy to facilitate the future inclusion of other resources, such as potential allergens, types of ingredients and more.

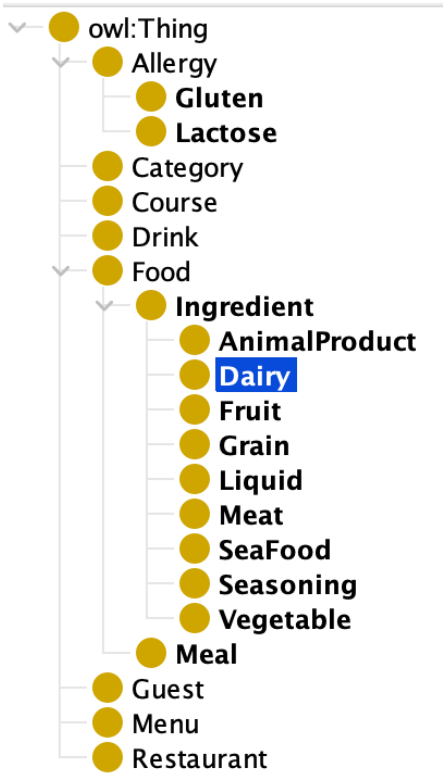

Figure 5.2: Classes hierarchy.

Object properties

Object properties are used to represent the relationships between classes.

- **Animal product contains lactose** : Properties used to link a food with a Allergy
- **drink has ingredient**: Properties to associate an ingredient to a drink

- **food has categories**:associate a food with the category
- **guest has Allergies**:link a guest with the allergy
- **guest has category** : associate a category (favourite category) to a guest
- **guest has preference course**: associate the guest with his course
- **guest is At risk allergies** : is a inferred properties
- **guest mealSuggested**: inferred rule to suggest a meal based on the guest preferences
- **ingredient has Gluten intolerance**: associate the gluten allergy to a ingredient
- **ingredient has Lactose intolerance**: associate the lactose allergy to a ingredient
- **meal has course** : link meal with the course
- **meal hasGlutenIntolerance** :Associate a meal with the gluten allergy
- **meal hasLactoseIntolerance**: Associate a meal with the lactose allergy
- **meal ingredients**: used to associate a meal with his ingredients
- **meal not has Gluten Intolerance**: It is use to associate a meal with an allergy
- **meal not has Lactose Intolerance**: It is used to associate a meal wiht an allergy
- **menu containsDrink**: associate the menu with the drink
- **Menu contains meal**: associate the menu with the meal
- **restaurant has guest**: to associate guest to a restaurant
- **restaurant has menu**: to associate menu to a restaurant

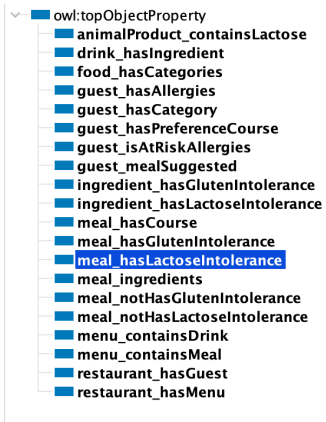

Figure 5.3: Object properties

Data properties

Data Properties represent the attributes of the classes.

- **meal caloriesConscious**: A data property representing an attribute with a literal value.

- **hasKcal**: A data property representing an attribute with a literal value.

- **guest CaloriesLevel**: A data property representing an attribute with a literal value.

- **hasName**: A data property representing an attribute with a literal value.

Figure 5.4: Data properties

Individuals

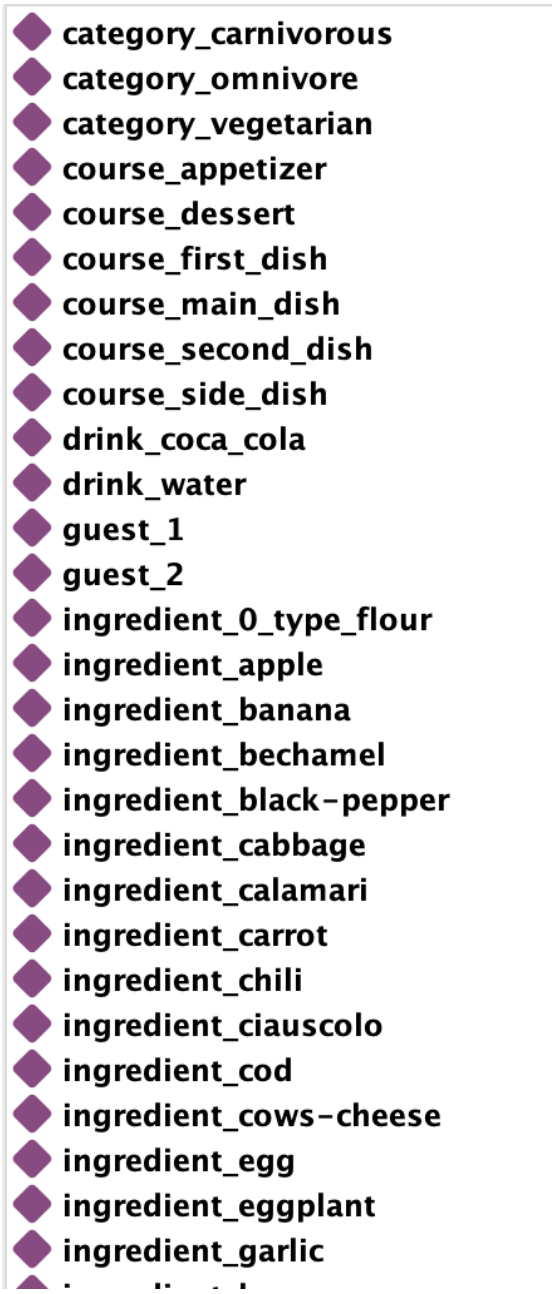

Figure 5.5: Individuals

Below some individuals association with the properties

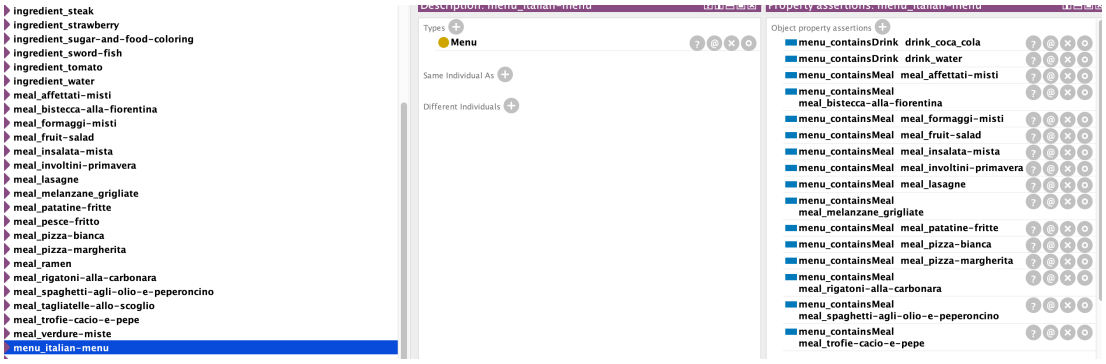

Figure 5.6: Restaurant entity in our model

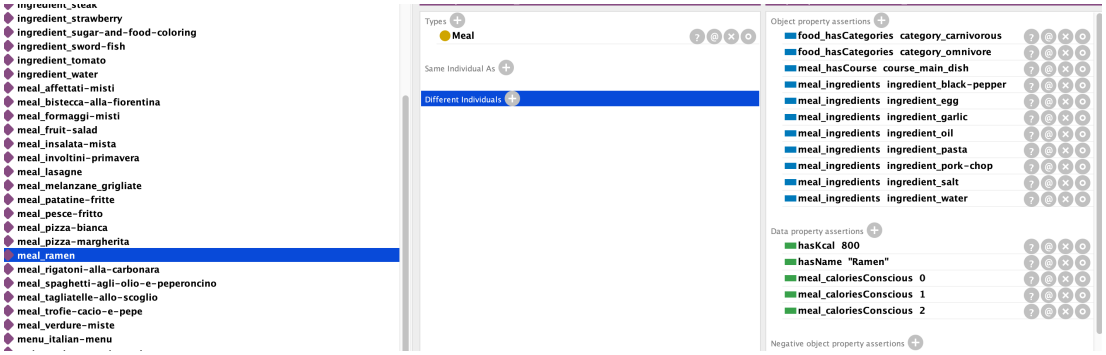

Figure 5.7: meal ramen entity in our model

SWRL Rules

We have exploited the reasoner provided by Protégé its power by defining the following rules:

S1: 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.

```
kebi2024:Guest(?guest) ^
kebi2024:guest_hasAllergies(?guest, ?allergy) ^
kebi2024:Meal(?meal) ^
kebi2024:meal_ingredients(?meal, ?ingredient) ^
kebi2024:ingredient_hasGlutenIntolerance(?ingredient, ?allergy)
-> kebi2024:guest_isAtRiskAllergies(?guest, ?meal)
```

S2: 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.

```
    kebi2024:Guest(?guest) ^
    kebi2024:guest_hasAllergies(?guest, ?allergy)^
    kebi2024:Meal(?meal) ^
    kebi2024:meal_ingredients(?meal, ?ingredient) ^
    kebi2024:ingredient_hasLactoseIntolerance(?ingredient, ?allergy)
    -> kebi2024:guest_isAtRiskAllergies(?guest, ?meal)
```

S3: 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.

```
kebi2024:Meal(?meal) ^
kebi2024:meal_ingredients(?meal, ?ingredient) ^
kebi2024:Ingredient(?ingredient) ^ kebi2024:ingredient_hasGlutenIntolerance
(?ingredient, ?gluten) ->
kebi2024:meal_hasGlutenIntolerance(?meal, ?gluten)
```

S4: 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.

```
kebi2024:Meal(?meal) ^ kebi2024:meal_ingredients(?meal, ?ingredient)^
kebi2024:Ingredient(?ingredient) ^
kebi2024:ingredient_hasLactoseIntolerance(?ingredient, ?dairy) ->
kebi2024:meal_hasLactoseIntolerance(?meal, ?dairy)
```

S5: This rule is used to infer the types of meals a guest can eat based on their preferences and whether they have a gluten intolerance.

```
kebi2024:Guest(?guest) ^
kebi2024:guest_hasCategory(?guest, ?category) ^
kebi2024:guest_hasAllergies(?guest, ?allergy) ^
kebi2024:guest_CaloriesLevel(?guest, ?level) ^
kebi2024:guest_hasPreferenceCourse(?guest, ?course) ^
kebi2024:Category(?category) ^
kebi2024:Allergy(?allergy) ^
kebi2024:Meal(?meal) ^
kebi2024:food_hasCategories(?meal, ?category) ^
kebi2024:meal_notHasGlutenIntolerance(?meal, ?allergy) ^
kebi2024:meal_caloriesConscious(?meal, ?level) ^
kebi2024:meal_hasCourse(?meal, ?course) ->
kebi2024:guest_mealSuggested(?guest, ?meal)
```

SWRL Testing

In the image below there is a exmaple of SWRL rule testing executed.

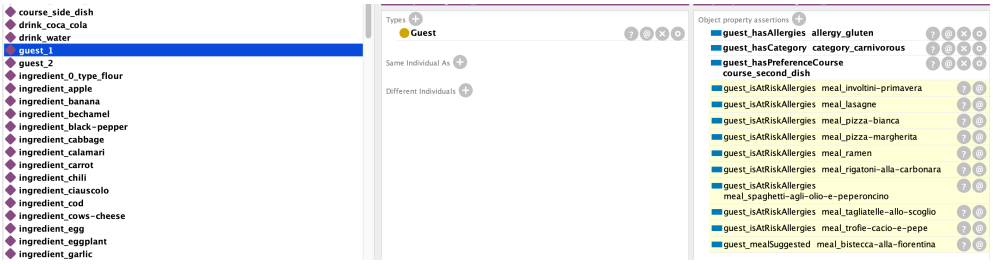

Figure 5.8: risk and suggestion meals

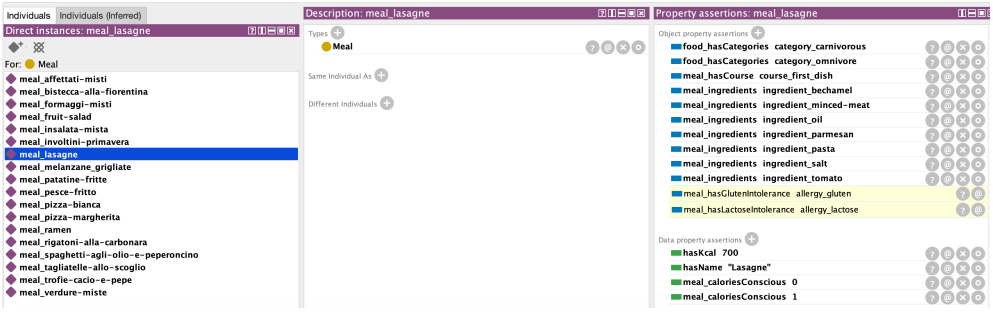

Figure 5.9: S3 and S4 in action

5.2 SPARQL Query

Regarding the SPARQL queries, we have developed these queries to filter and retrieve specific information, and subsequently tested them on both GraphDB and Protégé. SPARQL (SPARQL Protocol and RDF Query Language) is a powerful query language and protocol used for accessing and manipulating data stored in RDF format. It allows users to write queries to extract information from RDF graphs, perform complex searches, and integrate data from diverse sources. SPARQL queries enable precise data retrieval, making it an essential tool for working with semantic web data and linked data applications.

SPARQL query example

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

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix xml: <http://www.w3.org/XML/1998/namespace>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix kebi: <http://www.semanticweb.org/lauz/ontologies/2025/2/LuziLorenzoMenu/>
```

These are some of the query performed with SPARQL:

Query to retrieve all meal associate with carnivorous category

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

	meal
1	kebi:meal_affettati-misti
2	kebi:meal_bistecca-alla-fiorentina
3	kebi:meal_formaggi-misti
4	kebi:meal_lasagne
5	kebi:meal_patatine-fritte
6	kebi:meal_pesce-fritto
7	kebi:meal_pizza-bianca
8	kebi:meal_pizza-margherita
9	kebi:meal_ramen
10	kebi:meal_rigatoni-alla-carbonara
11	kebi:meal_spaghetti-agli-olio-e-peperoncino
12	kebi:meal_tagliatelle-allo-scoglio
13	kebi:meal_trofie-cacio-e-pepe

Figure 5.10: list of carnivorous meal

Query to retrieve all the ingredient in a meal

```
SELECT ?ingredient
WHERE {
      kebi:meal_ramen kebi:meal_ingredients ?ingredient.
}
```

1	kebi:ingredient_water
2	kebi:ingredient_black-pepper
3	kebi:ingredient_egg
4	kebi:ingredient_garlic
5	kebi:ingredient_oil
6	kebi:ingredient_pasta
7	kebi:ingredient_pork-chop
8	kebi:ingredient_salt

Figure 5.11: list of ramen's ingredients

Query to retrieve the calories related to the meal

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

	meal	calories
1	kebi:meal_affettati-misti	"200"^^xsd:integer
2	kebi:meal_bistecca-alla-fiorentina	"540"^^xsd:integer
3	kebi:meal_formaggi-misti	"300"^^xsd:integer
4	kebi:meal_fruit-salad	"90"^^xsd:integer
5	kebi:meal_insalata-mista	"130"^^xsd:integer
6	kebi:meal_involtini-primavera	"260"^^xsd:integer
7	kebi:meal_lasagne	"700"^^xsd:integer
8	kebi:meal_melanzane_grigliate	"80"^^xsd:integer
9	kebi:meal_patatine-fritte	"150"^^xsd:integer
10	kebi:meal_pesce-fritto	"500"^^xsd:integer
11	kebi:meal_pizza-bianca	"550"^^xsd:integer
12	kebi:meal_pizza-margherita	"700"^^xsd:integer
13	kebi:meal_ramen	"800"^^xsd:integer
14	kebi:meal_rigatoni-alla-carbonara	"705"^^xsd:integer
15	kebi:meal_spaghetti-agli-olio-e-peperoncino	"380"^^xsd:integer
16	kebi:meal_tagliatelle-allo-scoglio	"400"^^xsd:integer
17	kebi:meal_trofie-cacio-e-pepe	"510"^^xsd:integer
18	kebi:meal_verdure-miste	"50"^^xsd:integer

Figure 5.12: food calories

Query for the ingredient calories

```
Select ?ingredient ?calories
WHERE {
      ?ingredient rdf:type kebi:Ingredient.
   ?ingredient kebi:hasKcal ?calories.
}
```

Query to view the guest allergies

```
SELECT ?guest ?allergies
WHERE {
      ?guest a kebi:Guest;
   kebi:guest_hasAllergies ?allergies
}
```

Query for view the level of calories conscious of all meals

```
SELECT Distinct ?meal ?levelCalories
WHERE {
      ?meal rdf:type ?type.
   ?meal kebi:meal_caloriesConscious ?levelCalories.
}
```

Query for view the gluten intolerance of ingredient

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

Query to see the meal based on the guest preference

```
Select distinct ?meal ?mealName
Where{
   kebi:guest_1 a kebi:Guest;
   kebi:guest_hasAllergies ?allergy;
   kebi:guest_hasCategory ?category;
   kebi:guest_hasPreferenceCourse ?course;
   kebi:guest_CaloriesLevel ?caloriesLevel.

   ?meal a kebi:Meal;
   kebi:food_hasCategories ?category;
   kebi:meal_caloriesConscious ?caloriesLevel;
   kebi:meal_hasCourse ?course;
   kebi:hasName ?mealName.

   MINUS { ?meal kebi:meal_hasGlutenIntolerance ?allergy}
   MINUS{?meal kebi:meal_hasLactoseIntolerance ?allergy}
}
```

	meal	mealName
1	kebi:meal_bistecca-alla-fiorentina	"Bistecca alla fiorentina"

Figure 5.13: Meal suggested

Query to view the meal suggested

The difference from the previous query is that here we are using the properties inferred by the reasoner.

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix xml: <http://www.w3.org/XML/1998/namespace>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix kebi:<http://www.semanticweb.org/lauz/ontologies/2025/2/LuziLorenzoMenu/>

Select distinct ?meal ?mealName
Where{
   kebi:guest_2 kebi:guest_mealSuggested ?meal.
   ?meal kebi:hasName ?mealName
}
```

Execute

?meal	?mealName
kebi:meal_insalata-mista	Insalata mista^^xsd:string
kebi:meal_melanzane_grigliate	Melanzane grigliate^^xsd:string
kebi:meal_patatine-fritte	Patatine fritte^^xsd:string

Figure 5.14: Meal suggested

The other different from the previous query is that the first query is for the guest 1 instead the second query is for quest 2. The query can be found in the repository

5.3 SHACL

SHACL (Shapes Constraint Language) is a language used for validating RDF (Resource Description Framework) data against a set of conditions, known as shapes. These shapes define the expected structure and content of RDF graphs, ensuring data consistency and integrity within semantic web applications. SHACL shapes can specify constraints on the types of nodes, the cardinality of properties, and even the values that properties can take. By defining these constraints, SHACL helps maintain high data quality and enables more robust data integration and querying. [1]

Restaurant Shape

Ensure that each Restaurant instance has at least one Menu. For example if we delete the menu, the error "It is impossible that a Restaurant does not have a menu??".

```
1
2  kebi:RestaurantShape a sh:NodeShape ;
3     sh:targetClass kebi:Restaurant ;
4     sh:property [
5        sh:path kebi:restaurant_hasMenu ;
6        sh:node kebi:MenuShape ;
7        sh:minCount 1 ;
8        sh:message "It␣is␣impossible␣that␣a␣Restaurant␣does␣not␣have␣a␣menu!!" ;
9     ] ;
10    sh:property [
11       sh:path kebi:restaurant_hasGuest ;
12       sh:node kebi:GuestShape ;
13       sh:minCount 0 ;
14    ] .
```

Menu Shape

Ensuring that the menu contains at least one Meal and one Drink.

[1] Shape file here

```
kebi:MenuShape
   a sh:NodeShape ;
   sh:targetClass kebi:Menu ;

   # Constraint: At least one Meal in the Menu
   sh:property [
      sh:path kebi:menu_containsMeal ;
      sh:minCount 1 ;
      sh:message "Each menu must contain at least one meal." ;
   ] ;

   # Constraint: At least one Drink in the Menu
   sh:property [
      sh:path kebi:menu_containsDrink ;
      sh:minCount 1 ;
      sh:message "Each menu must contain at least one drink." ;
   ] .
```

Guest Shape

Validate instances of Guest, requiring mandatory information such as dietary category and calories-consciousness level. The calorie consciousness level range from 1 to 4 because if a guest has level of 0, it automatically inclueds the other levels.

```
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_isAtRiskForFood ;
      sh:node kebi:MealShape ;
   ] ;
   sh:property [
      sh:path kebi:guest_hasLevelOfCalorieConscious ;
      sh:datatype xsd:integer ;
      sh:minCount 0 ;
      sh:maxCount 4 ;
   ] ;
.
```

Meal Shape

Specifies constraints for instances of meals. In this case a meal can contain lactose or gluten, and it can have one or more categories. A meal must have a course and at least one ingredient

```
kebi:MealShape a sh:NodeShape ;
   sh:targetClass kebi:Meal ;
   sh:property [
      sh:path kebi:meal_hasLactoseIntolerance ;
      sh:node kebi:LactoseShape ;
      sh:minCount 0 ;
      sh:maxCount 1 ;
   ] ;
   sh:property [
      sh:path kebi:meal_hasGlutenIntolerance ;
```

```
11        sh:node kebi:GlutenShape ;
12        sh:minCount 0 ;
13        sh:maxCount 1 ;
14     ] ;
15     sh:property [
16        sh:path kebi:food_hasCategories ;
17        sh:node kebi:CategoryShape ;
18        sh:minCount 0 ;
19     ] ;
20     sh:property [
21        sh:path kebi:meal_hasCourse ;
22        sh:node kebi:CourseShape ;
23        sh:minCount 1 ;
24        sh:maxCount 1 ;
25     ] ;
26     sh:property [
27        sh:path kebi:meal_ingredients ;
28        sh:node kebi:IngredientShape ;
29        sh:minCount 1 ;
30     ] ;
31  .
```

The SHACL shape file can be find in the repository.

6. Agile and Ontology based Meta Modelling

6.1 AOAME

AOAME is a prototype tool designed to implement an agile, ontology-based approach to meta-modeling. This approach focuses on designing and managing schemas for Enterprise Knowledge Graphs (EKGs), which structure domain-specific knowledge to enable analysis, reasoning, and integration of data from multiple sources. One of the main challenges in developing and maintaining EKG schemas is the need for expertise in both ontology engineering and the specific application domain.

To address this challenge, AOAME extends traditional meta-modeling by incorporating agile principles. This allows for domain-specific adaptations of modeling languages and enables real-time testing, ensuring that domain experts can actively participate in the engineering process. Developed using the Design Science Research methodology, AOAME serves as a prototype to assess the effectiveness of this approach. It provides functionalities to extend, modify, remove, and hide modeling constructs within the tool palette. These actions are powered by meta-modeling operators that generate SPARQL queries and update the triplestore, maintaining consistency between human-interpretable and machine-interpretable knowledge.

The software architecture of AOAME utilizes Java libraries from Apache Fuseki to develop APIs that retrieve ontologies from the triplestore and display them in a graphical user interface (GUI). It also implements the meta-modeling operators necessary for schema adaptation. The tool has been validated through real-world scenarios and case studies, demonstrating its versatility across various application domains. By fostering close collaboration between domain experts and language engineers in the development of domain-specific modeling languages (DSMLs), AOAME enhances the flexibility and efficiency of the meta-modeling process. [1]

6.2 Key Elements of BPMN 2.0

BPMN 2.0 consists of several core elements that define business process flows:

- **Events**: Represent occurrences that influence the process flow, such as:
  - *Start events*: Indicate where the process begins.
  - *End events*: Mark the conclusion of the process.

[1] Forked repository here

  - *Intermediate events*: Occur between the start and end events, impacting process execution.

- **Activities**: Define tasks or work that must be performed, including:
  - *User tasks*: Performed by human users.
  - *Service tasks*: Automated system processes.
  - *Manual tasks*: Tasks performed outside of system automation.
- **Gateways**: Decision points that control the divergence and convergence of process flow, such as:
  - *Exclusive gateways*: Allow only one possible outcome.
  - *Parallel gateways*: Enable multiple simultaneous paths.
  - *Inclusive gateways*: Allow multiple paths to be taken if conditions permit.
- **Pools and Lanes**: Represent key participants in a process:
  - *Pools*: Represent different organizations or entities.
  - *Lanes*: Represent subdivisions within an organization.
- **Artifacts**: Provide supplementary information about the process, including:
  - *Data objects*: Represent data used or generated in the process.
  - *Groups*: Categorize elements without affecting process flow.
  - *Annotations*: Provide additional explanatory notes.

6.3 Application in Restaurant Order Management

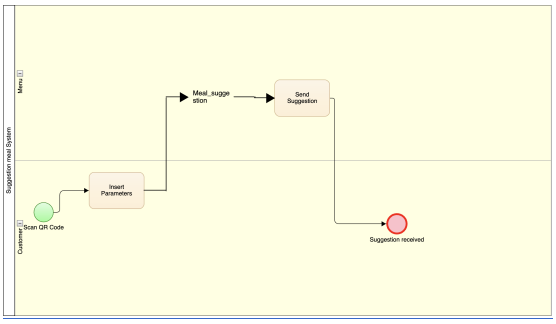

Figure 6.1: Meal suggested

In this project, BPMN is utilized to help a restaurant manager better understand the customer ordering process. The BPMN diagram will take into account customer allergies and dietary preferences. Customers will be required to declare any allergies and specify whether they are vegetarian, vegan, or carnivorous. Based on these preferences, they can proceed to select their desired meals. This structured approach enhances both customer satisfaction and safety by tailoring the ordering process to individual needs.

The BPMN diagram represents a meal suggestion system where a customer interacts with the system to receive meal recommendations. The workflow consists of the following steps:

- **Swimlanes:** The diagram contains two swimlanes:
  - *Customer*: Represents user interactions.
  - *Suggestion Meal System*: Represents the automated system processing the meal suggestion.
- **Start Event:** The process begins when the customer scans a QR code.
- **Insert Parameters:** The customer inserts meal preference parameters into the system.
- **Meal suggestion:** The system processes the input and generates a meal suggestion.
- **Send Suggestion:** The system sends the meal suggestion to the customer. (I had problem to upload a customize image for this task)
- **End Event:** The process concludes when the suggestion is received.

To use the diagram build in the BPMN we had use Jena Fuseki. Jena Fuseki is a robust and scalable SPARQL server designed to handle RDF data effiectly. It provides a interface to perform various operations on RDF dataset. The query below is used to get the meal suggested based on the properties selected in the BPM model.

```
prefix mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
prefix lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix xml: <http://www.w3.org/XML/1998/namespace>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix kebi: <http://www.semanticweb.org/lauz/ontologies/2025/2/LuziLorenzoMenu/>
SELECT Distinct ?meal ?category ?levelCC ?allergy ?course
WHERE {
      mod:Meal_suggestion_502661bc-058b-48d5-bb3b-6061f5544f0d
   lo:guest_hasCategory ?category.
 mod:Meal_suggestion_502661bc-058b-48d5-bb3b-6061f5544f0d
 lo:guest_hasAllergies ?allergy.
 mod:Meal_suggestion_502661bc-058b-48d5-bb3b-6061f5544f0d
 lo:guest_hasPreferenceCourse ?course.
 mod:Meal_suggestion_502661bc-058b-48d5-bb3b-6061f5544f0d
 lo:guest_CaloriesLevel ?levelCC.
      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_firsh_dish,
      if(?course= "main dish", kebi:course_main_dish,
        if(?course="second course", kebi:course_second_dish,
          if(?course="sidedish", kebi:course_side_dish,
            if(?course="desser", kebi:course_dessert,?course))))))as ?finalCourse).

 BIND(IF(?allergy="gluten",kebi:ingredient_hasGlutenIntolerance,
    if(?allergy="lactose",kebi:ingredient_hasLactoseIntolerance,
      if(?allergy="none","none",?allergy)))as ?finalAllergy).

 ?meal a kebi:Meal .
 ?meal kebi:meal_caloriesConscious ?kcal.
 Filter(?kcal = ?levelCC)

 ?meal kebi:food_hasCategories ?finalCategory.
```

```
 ?meal kebi:meal_hasCourse ?finalCourse.

 Optional{
   ?meal kebi:meal_ingredients ?ingredient.
   ?ingredient rdf:type ?ingredientType.
   Filter ((?finalAllergy="none") ||
     (?finalAllergy = kebi:ingredient_hasGlutenIntolerance &&
     ?ingredientType = kebi:Ingredient) ||
     (?finalAllergy = kebi:ingredient_hasLactoseIntolerance &&
     ?ingredientType= kebi:Ingredient))
           }
 filter (!bound(?ingredient))

}
```

Below we'll provide some example of result based on different properties selected by a customer.

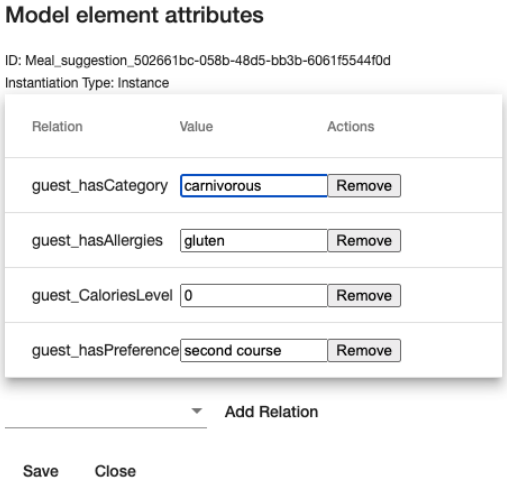

Figure 6.2: Properties selected into the BPMN model

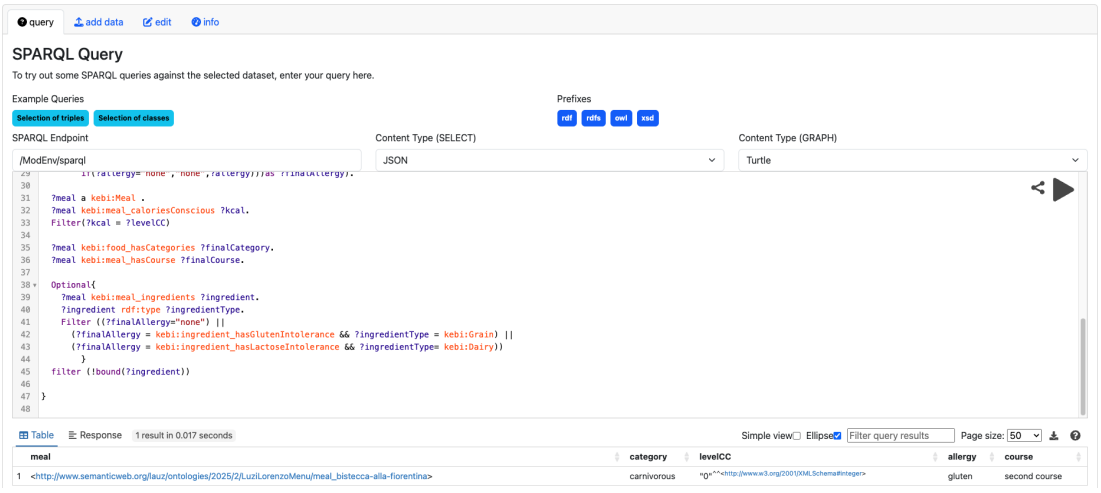

Figure 6.3: Result of the previous selection

This below is the same query but using different parameters

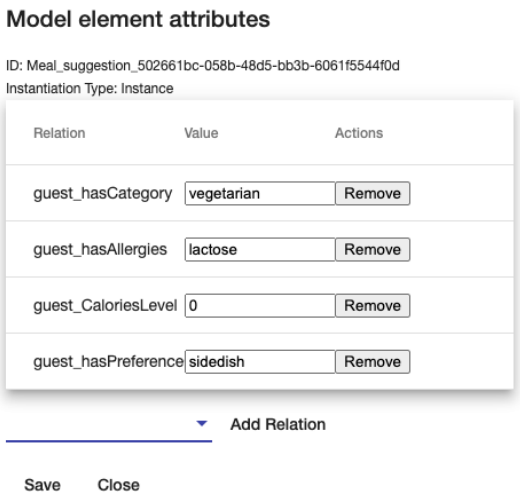

Figure 6.4: Properties selected into the BPMN model

	meal	category	levelCC	allergy	course
1	<http://www.semanticweb.org/lauz/ontologies/2025/2/LuziLorenzoMenu/meal_insalata-mista>	vegetarian	"0"^^<http://www.w3.org/2001/XMLSchema#integer>	lactose	sidedish
2	<http://www.semanticweb.org/lauz/ontologies/2025/2/LuziLorenzoMenu/meal_melanzane_grigliate>	vegetarian	"0"^^<http://www.w3.org/2001/XMLSchema#integer>	lactose	sidedish
3	<http://www.semanticweb.org/lauz/ontologies/2025/2/LuziLorenzoMenu/meal_patatine-fritte>	vegetarian	"0"^^<http://www.w3.org/2001/XMLSchema#integer>	lactose	sidedish

Figure 6.5: Result of the previous selection

We can try as many configuration as possible. Base on the meal on the menu with their specific properties it can happens that the system isn't able to suggest some meal.

```
prefix mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
prefix lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix xml: <http://www.w3.org/XML/1998/namespace>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix kebi: <http://www.semanticweb.org/lauz/ontologies/2025/2/LuziLorenzoMenu/>
SELECT Distinct ?properties ?value ?levelCC ?allergy ?course
WHERE {
    mod:Meal_suggestion_502661bc-058b-48d5-bb3b-6061f5544f0d ?properties ?value.


}
```

Figure 6.6: Query to get the properties of our extends calss

7. Conclusion

Throughout this project, my colleague and I explored various approaches to knowledge engineering, evaluating different methodologies and tools for their effectiveness in structuring and managing knowledge-based systems. This journey provided valuable insights into the strengths and limitations of each approach. Below, I share my perspective on these methods.

Decision Tables:

Decision tables provide a structured way to define rules by mapping inputs to outputs using a predefined logic. They follow a "Hit Policy," which determines how results are aggregated. In our case, we utilized the "Collect" policy, which allowed for simultaneous rule evaluations. While this approach ensures clarity and ease of use, it becomes impractical when dealing with complex datasets or intricate dependencies. Additionally, creating and managing large tables proved to be repetitive and time-consuming, making debugging a challenging task due to limited support from available tools.

Prolog:

Among all the tools and methodologies explored, I found Prolog to be the most enjoyable and intuitive. Learning a logic-based programming language like Prolog was a refreshing experience, especially due to its declarative nature, which allows for straightforward knowledge representation and retrieval. Unlike traditional procedural languages, Prolog emphasizes recursion and logical inference rather than loops and arithmetic operations. While I appreciated its ability to efficiently model logical relationships, I did encounter challenges, particularly in accessing well-structured learning resources. Additionally, Prolog's performance can degrade when handling large-scale knowledge bases, leading to increased computation times. Despite these limitations, I found Prolog to be a powerful tool for reasoning over structured knowledge, making it my preferred approach in this project.

Protégé:

This tool offers a hybrid approach, combining elements of Decision Model and Notation (DMN) with object-oriented principles. It enables the definition of entities, their attributes, and relationships, allowing for structured knowledge modeling. One aspect I particularly appreciated was its querying mechanism, which closely resembles SQL, making it intuitive to use. The SHACL validator was also beneficial, as it provided a mechanism for verifying property constraints, similar to assertion testing in Java. However, managing inference rules through SWRL requires careful consideration, as

not all querying tools support reasoning capabilities natively. This constraint became evident when working with AOAME and GraphDB, where queries had to be explicitly designed to account for inference logic.

AOAME:

AOAME proved to be a valuable tool for integrating meta-modeling with ontologies, allowing us to enhance Business Process Model and Notation (BPMN) diagrams. By extending BPMN with custom classes such as "Suggest Meal"—a subclass of "Task"—we were able to create a more intuitive graphical notation tailored to restaurant management. Additionally, by incorporating attributes like course type, allergy information, calorie preferences, and dietary categories, we enriched our ontology for more precise meal recommendations. The integration with Apache Jena Fuseki further streamlined our ability to execute SPARQL queries, facilitating dynamic menu generation. However, AOAME does have notable drawbacks, including persistent bugs and stability issues, particularly when handling high user traffic in its online version. Despite these challenges, its potential in ontology-driven process modeling remains significant.

7.1 Final Thoughts

In summary, this project provided an insightful exploration into various knowledge-based approaches, each with its own trade-offs. Decision Tables offer simplicity but can become impractical for complex use cases. Prolog enables intuitive data manipulation but presents challenges in scalability and accessibility. Protégé facilitates structured knowledge modeling but demands careful management of inference rules. AOAME stands out for its seamless ontology integration but requires refinement to enhance its usability and stability.

Ultimately, this project demonstrated that while the implemented solutions are functional, there is room for improvement in terms of efficiency and adaptability. Further refinements could optimize the system's performance, making it more effective for real-world applications. Despite the challenges faced, particularly working alone after the first deadline, this experience has significantly expanded my understanding of knowledge-based systems and reinforced my preference for working with Prolog as a reasoning tool. This project has been an invaluable experience, allowing me to explore new fields in knowledge engineering and experiment with various knowledge-based solutions. Throughout the process, my colleague and I evaluated different approaches, analyzing their strengths and limitations. However, after the first deadline, I was unable to contact my colleague, as It is an exchange student. As a result, I continued the project independently, using some elements from our initial version as a foundation while refining and expanding upon our work.