



University of Camerino

SCHOOL OF SCIENCES AND TECHNOLOGIES

Master degree in Computer Science (LM-18)

Personalized Menu Recommendations in Restaurants with Knowledge Graphs, Ontologies and Agile BPMN Processes

Author

Mathukiya Vaibhav Jagdish

Matricula 127837

Advisor

Prof. Knut Hinkelmann

Coadvisor

Prof. Emanuele Laurenzi

A.Y. 2023/2024

Contents

1	Introduction	9
1.1	Objectives	9
1.2	Composition of Menu	9
2	Decision table: Camunda	13
2.1	The Input Channel	13
2.2	Knowledge Sources	14
2.3	The Decision Model:	14
2.4	Decision tables	14
2.5	Testing and Simulation	15
3	Prologs	17
3.1	Rules	17
3.2	Simulation and checks	20
4	Knowledge Graph / Ontology	23
4.1	Syntax	23
4.2	Protege	23
4.2.1	The ontology	24
4.2.2	SPARQL Queries	25
4.3	SHACL Shapes	28
4.3.1	SWRL Rules	29
5	Agile and Ontology-Based Meta-modelling	31
5.1	AOAME	31
5.2	BPMN 2.0	31
5.2.1	The model	32
5.3	Jena Fuseki	33
5.3.1	Queries	33
6	Conclusion	37
6.1	Links	39

List of Figures

2.1	Input Sources	13
2.2	Knowledge Sources	14
2.3	Camunda Model	15
2.4	Decision table for Selected-Ingredients	16
2.5	Decision table for Final-dishes	16
3.1	Ingredients in Prolog	18
3.2	Meals in Prolog	18
3.3	Dishes and checks	19
3.4	Allergies and vegetarian	20
3.5	Checking Meals	20
3.6	Queries	20
3.7	Lactose, Gluten, No vegetarian for low calories	21
3.8	Lactose, Gluten, No vegetarian for high calories	21
4.1	The overview of the ontology	24
4.2	Ontology of Ingredients	26
4.3	Ontology of Meals	26
4.4	Ontology for Menu	27
4.5	Example for MealClass for Macedonia	27
4.6	Prefix	28
4.7	Basic Queries	28
4.8	SHACL Constraints	29
5.1	BPMN model	32
5.2	Example	32
5.3	Jena Fuseki Query	34
5.4	Results of query	35

List of Tables

1.1	Ingredients Dietary Information and Caloric Content	10
1.2	Meals with Ingredients and Caloric Values	11

1. Introduction

Nowadays, in a fast-changing technology world, many industries are using new tools to work smarter and stay competitive. Restaurants, for example, now often use QR codes to offer digital menus. This change saves on printing costs and makes updating of menu items much easier, while digital menus are great for restaurant owners, they can sometimes be hard or challenging for customers to navigate—especially on small screens.

That's where menu personalization can be useful. By changing the menu to each customer's needs—such as highlighting/showing options for those with lactose or gluten allergies, offering vegetarian choices, or filtering by calorie counts (low, medium, or high)—restaurants can create a more enjoyable and satisfying dining experience for customers.

1.1 Objectives

This project aims to create an advanced Knowledge Base that generates personalized menus based on the customer's dietary needs and preferences. To resolve this challenge, I used several tools and technologies:

- **Camunda (Chapter 2):** Used for modeling and managing decision-making processes.
- **Prolog (Chapter 3):** Shows the rule-based system behind the logic.
- **Protegé (Chapter 4):** Helps design a strong and flexible conceptual framework through ontology engineering.
- **AOAME and Jena Fuseki (Chapter 5):** Provide support for visual data representation and efficient query execution.

All files, implementation details, and documentation are available on GitHub: [LINK](#)

1.2 Composition of Menu

The system shows a menu with several unique ingredients and carefully chosen dishes. Each ingredient is detailed with its name, lactose and gluten content, vegetarian suitability, and calorie count. This information allows customers to easily filter ingredients based on different dietary needs, which is shown in the table below.

Every dish is defined by its name and its list of ingredients. Ingredients are selected based on their calorie counts, making sure that each dish meets specific nutritional

goals. This approach helps us create balanced meals and makes it simple to generate personalized menus.

By organizing the menu in this way, the system is flexible enough to handle a wide range of dietary requirements while maintaining consistency and balance.

Ingredients	Lactose Intolerant	Gluten Free	Vegetarian	Vegan	kcal/100g
Sausage	yes	yes	no	no	301
Milk	no	yes	no	no	42
Tomato	yes	yes	yes	yes	20
Coffee	yes	yes	yes	yes	0
Egg	yes	yes	yes	no	155
Bacon	yes	yes	no	no	541
Water	yes	yes	yes	yes	0
Onion	yes	yes	yes	yes	40
Flour	no	no	yes	yes	364
Mayonnaise	yes	yes	no	no	680
Salmon	yes	yes	no	no	208
Sparkling Water	yes	yes	yes	yes	0
Zucchini	yes	yes	yes	yes	20
Mushroom	yes	yes	yes	yes	22
Pasta	no	yes	yes	yes	131
Rosemary	yes	yes	yes	yes	131
Oil	yes	yes	yes	yes	884
Lettuce	yes	yes	yes	yes	15
Peppers	yes	yes	yes	yes	31
Mozzarella	no	yes	yes	no	280
Salt	yes	yes	yes	yes	0
Beef	yes	yes	no	no	250
Rice	yes	no	yes	yes	130
Eggplant	yes	yes	yes	yes	25
Carrot	yes	yes	yes	yes	32
Sugar	yes	yes	yes	yes	387
Tomato Sauce	yes	yes	yes	yes	29
Ham	no	yes	no	no	145

Table 1.1: Ingredients Dietary Information and Caloric Content

Meal	Ingredients	Kcal Value
Pizza Funghi e Prosciutto	Dough, Flour, Water, Oil, Mozzarella, Mushrooms, Ham, Tomato Sauce	High
Tiramisu	Mascarpone, Ladyfingers, Coffee, Eggs, Sugar, Cocoa	Low
Salad	Lettuce, Tomato, Cucumber, Carrots, Oil, Salt	Low
Vegetarian Ravioli	Pasta, Flour, Water, Oil, Onion, Mozzarella, Basil, Salt, Carrot, Spinach, Zucchini	High
Patate al Forno	Potatoes, Oil, Salt, Onion, Pepper	Low
Risotto with Vongole	Rice, Flour, Water, Oil, Vongole, Tomato Sauce, Zucchini, Onion, Wine	High
Tartar	Salmon, Salt, Avocado, Lemon, Tomato	High
Arancini al Ragù	Rice, Sugar, Salt, Water, Mozzarella, Onion, Beef, Oil, Tomato Sauce	High
Lasagna	Pasta, Flour, Water, Oil, Ragù, Mozzarella, Salt, Parmesan, Basil	High
Panna Cotta	Cream, Milk, Sugar, Gelatin, Vanilla	Low
Vegetable Pizza	Dough, Flour, Water, Oil, Mozzarella, Mushrooms, Peppers, Eggplant, Zucchini, Tomato Sauce	Medium
Ravioli with Ragù	Pasta, Flour, Water, Oil, Ground Beef, Tomato Sauce, Onion, Mozzarella, Basil, Salt, Carrot	High
Parmigiana di Melanzane	Eggplant, Oil, Tomato Sauce, Mozzarella, Salt	High
Pasta Salmon and Tomato	Pasta, Flour, Water, Oil, Mozzarella, Tomato Sauce, Salmon	High
Pizza Salmon	Dough, Flour, Water, Oil, Mozzarella, Tomato Sauce, Salmon	High
Pizza Patate e Bacon	Dough, Flour, Water, Oil, Mozzarella, Potatoes, Bacon, Tomato Sauce	High
Pizza Peperoni e Salsiccia	Dough, Flour, Water, Oil, Mozzarella, Sausage, Peppers, Tomato Sauce	High
Vegetarian Arancini	Rice, Salt, Water, Mozzarella, Oil, Tomato Sauce, Carrots, Peas, Zucchini	High
Pasta with Vongole	Pasta, Flour, Water, Oil, Vongole, Tomato Sauce, Zucchini	Medium
Risotto Salsiccia and Radicchio	Rice, Sausage, Onion, Salt, Water, Tomato Sauce, Mozzarella, Mushrooms, Wine	High
Ricotta al sugo	Ricotta, Oil, Onion, Tomato Sauce, Salt, Basil	Low
Pizza Margherita	Dough, Flour, Water, Oil, Mozzarella, Tomato Sauce, Basil	High

Table 1.2: Meals with Ingredients and Caloric Values

2. Decision table: Camunda

Camunda allows businesses to model their decision-making using the DMN standard, which visually and formally represents decision rules. This makes it easy to understand the logic behind decisions and build models that mirror company policies.

Using decision diagrams and tables, Camunda displays rules and criteria, simplifying even complex decisions. Once created, these decision models run directly on the Camunda platform, integrating seamlessly with business processes to improve efficiency and consistency.

2.1 The Input Channel

Input Data represents the information provided to the system for decision-making. These are variables entered by the user or another system during process execution. In DMN, Input Data is used in Decision Tables to produce an output.

In the diagram, four Input Data elements allow the customer to specify their dietary preferences and restrictions:

1. **Lactose-Intolerance:** Boolean value—enter true if lactose intolerant, false otherwise.
2. **Gluten-Intolerance:** Boolean value—enter true if gluten intolerant, false otherwise.
3. **Vegetarian:** Boolean value—enter true if vegetarian, false otherwise.
4. **Calorie:** Allows the customer to choose a preferred calorie range. If not specified, dishes of any calorie level will be considered.

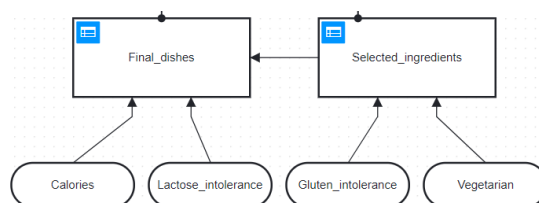


Figure 2.1: Input Sources

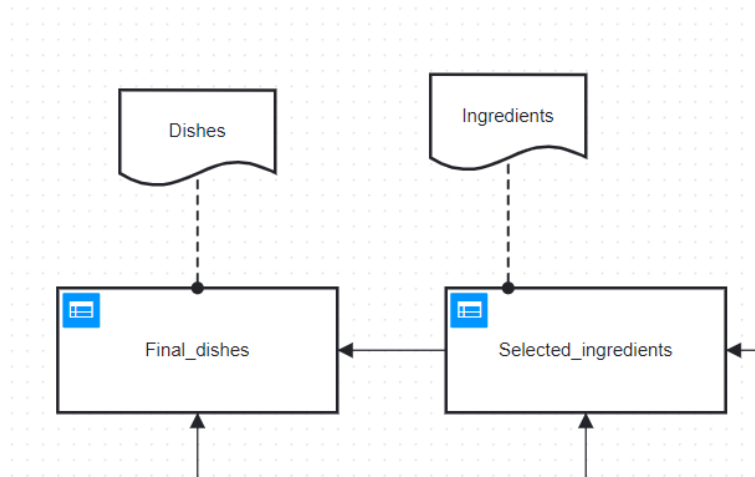


Figure 2.2: Knowledge Sources

2.2 Knowledge Sources

Knowledge Sources provide the main information needed for decision-making. They are the fixed resources that support applying decision rules to produce the best outcomes based on specific inputs.

In the diagram, there are two Knowledge Sources:

1. **Ingredients:** A comprehensive list of all available ingredients in the restaurant, detailing their properties.
2. **Dishes:** A complete list of all dishes offered by the restaurant.

These sources act as the main source for the decision process, making sure that choices are informed by accurate and latest information.

2.3 The Decision Model:

The image below illustrates the decision model. It takes two primary inputs—the list of available ingredients and the complete menu of dishes. In addition, it incorporates the customer's dietary preferences, including lactose intolerance, gluten intolerance, vegetarian choice, and preferred calorie level.

The following sections explain the rules in the decision tables in detail.

2.4 Decision tables

- **Selected Ingredients:**

This table helps identify which ingredients should be excluded from dishes based on customer preferences. A "-" symbol in the table means the condition is irrelevant for that rule, allowing flexibility. For example, if a rule has "-" in all conditions, the ingredient is always allowed, regardless of lactose intolerance, gluten intolerance, or vegetarian preferences. However, if an ingredient is marked as false for any condition, it will be excluded, and dishes containing it won't be available.

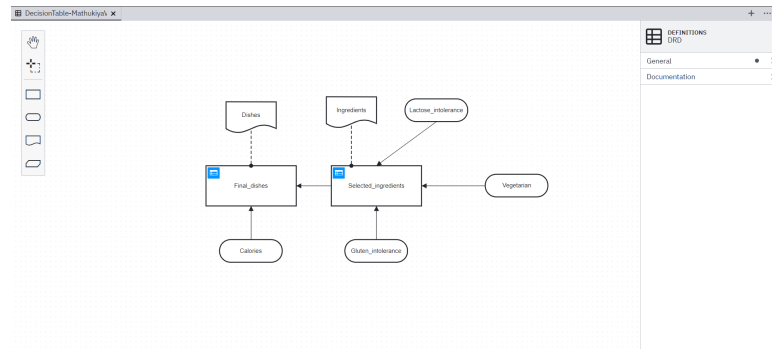


Figure 2.3: Camunda Model

The Collect hit policy ensures that multiple satisfied rules result in a list of selected ingredients, allowing a comprehensive selection based on the customer's dietary needs.

- **Final Dishes:**

This table determines and recommends dishes based on selected ingredients and the customer's desired calorie level (low, medium, or high). A dish is only included if all its ingredients are in the selected list, ensuring unwanted ingredients are avoided.

Using the Collect hit policy, the system gathers all suitable dishes, giving customers multiple options that fit their dietary preferences and calorie needs.

2.5 Testing and Simulation

As I have tried multiple times to search for the possibility of simulations and even contacting the mods of camunda, I didn't get a positive response and the simulation couldn't be carried out. Still I will try to write the hypothetical results.

Case 1: If a Client prefers lactose, gluten, meat and a medium calories level. In that way, all ingredients are considered and then the final dishes related to that will be within that calorie range will be returned.

Case 2: If a Client prefers no lactose, no gluten and high calories. So it shows that customers could've lactose and gluten tolerance as well, and they have specified high calories, so the dishes that are correct according to the filters will be returned. But also customers can be able to eat low calories.

DecisionTable-Mathukiyal x

Edit DRD Open overview

Selected_ingredients Hit policy: Collect

	When	And	And	Then	
	Lactose_intolerance	Gluten_intolerance	Vegetarian	Selected_ingredients	Annotations
	boolean	boolean	boolean	"Coffee","Water","Sparkling Wat...	
27	-	-	-	"Strawberry"	
28	-	-	-	"Peach"	
29	-	-	-	"Apple"	
30	-	-	-	"Ricotta"	
31	false	-	-	"Pecorino"	
32	false	-	-	"Parmesan"	
33	false	-	-	"Mozzarella"	
34	false	-	-	"Mascarpone"	
35	-	false	-	"Flour"	
36	-	false	-	"Pasta"	
37	-	false	-	"Bread"	
38	-	false	-	"Ladyfingers"	
39	-	-	false	"Beef"	
40	-	-	false	"Bacon"	
41	-	-	false	"Sausage"	
42	-	-	false	"Ham"	
43	-	-	-		

Figure 2.4: Decision table for Selected-Ingredients

DecisionTable-Mathukiyal x

Edit DRD Open overview

Final_dishes Hit policy: Collect

	When	And	Then	
	Selected_ingredients	Calories_level	output	
	list	"low","medium","high",""	string	
17	list contains(Selected_ingredients, "Pasta") and list contains(Selected_ingredients, "Salmon") and list contains(Selected_ingredients, "Tomato sauce") and list contains(Selected_ingredients, "Salt") and list contains(Selected_ingredients, "Oil")	"low"	"Pasta salmon and tomato"	Meat
18	list contains(Selected_ingredients, "Pasta") and list contains(Selected_ingredients, "Salmon") and list contains(Selected_ingredients, "Tomato sauce") and list contains(Selected_ingredients, "Salt") and list contains(Selected_ingredients, "Oil")	"medium"	"Pasta salmon and tomato"	Meat
19	list contains(Selected_ingredients, "Pasta") and list contains(Selected_ingredients, "Salmon") and list contains(Selected_ingredients, "Tomato sauce") and list contains(Selected_ingredients, "Salt") and list contains(Selected_ingredients, "Oil")	"high"	"Pasta salmon and tomato"	Meat
20	list contains(Selected_ingredients, "Pasta") and list contains(Selected_ingredients, "Salmon") and list contains(Selected_ingredients, "Tomato sauce") and list contains(Selected_ingredients, "Salt") and list contains(Selected_ingredients, "Oil")	""	"Pasta salmon and tomato"	Meat

Figure 2.5: Decision table for Final-dishes

3. Prologs

Prolog is a declarative programming language that uses logic-based rules and facts. It employs an inference engine to process queries and find solutions. In the project, I tried to implement the DMN model using Prolog.

The Prolog implementation is based on the DMN diagram created with Camunda. I first tried defining the ingredients and dishes, then implemented rules to generate personalized menus.

As shown in Image 3.1, ingredients are represented using the ‘ingredient’ predicate, which specifies an ingredient’s name and its key attributes:

- Lactose-free
- Gluten-free
- Vegetarian

For example, salmon is lactose-free, gluten-free, and vegetarian, while flour is lactose-free but contains gluten and is also vegetarian. This structured representation allows the program to match ingredients with dishes and generate personalized menus based on customer preferences.

Image 3.2 illustrates how dishes are represented. Each meal is defined using the meal predicate, which includes:

- Meal name
- List of ingredients
- Caloric value (categorized as low, medium, or high)

For example, Tartar consists of salt, salmon, avocado, lemon, and tomato and is classified as high in calories. Similarly, Arancini with Ragù includes bread, flour, rice, salt, water, mozzarella, onion, beef, oil, and tomato sauce, also categorized as high in calories.

By structuring meals this way, the program can filter dishes based on customer dietary preferences, ensuring a personalized menu tailored to individual needs.

3.1 Rules

This image shows whether the rule calories level finds all the meals that meets the calorie criteria, it uses numeric value to check meet values with 1,2,3, for checking low, medium,

```

1 /* Ingredients */
2
3 ingredient(tomato,lactose_free,gluten_free,vegetarian).
4 ingredient(tomato_sauce,lactose_free,gluten_free,vegetarian).
5 ingredient(mushrooms,lactose_free,gluten_free,vegetarian).
6 ingredient(truffle,lactose_free,gluten_free,vegetarian).
7 ingredient(avocado,lactose_free,gluten_free,vegetarian).
8 ingredient(ketchup,lactose_free,gluten_free,vegetarian).
9 ingredient(mayonnaise,lactose_free,gluten_free,vegetarian).
10 ingredient(ricotta,lactose_free,gluten_free,vegetarian).
11 ingredient(pecorino,lactose_free,gluten_free,vegetarian).
12 ingredient(parmesan,lactose_free,gluten_free,vegetarian).
13 ingredient(mozzarella,lactose_free,gluten_free,vegetarian).
14 ingredient(mascarpone,lactose_free,gluten_free,vegetarian).
15 ingredient(seabass,lactose_free,gluten_free,vegetarian).
16 ingredient(duck,lactose_free,gluten_free,no_vegetarian).
17 ingredient(vongole,lactose_free,gluten_free,vegetarian).
18 ingredient(beef,lactose_free,gluten_free,no_vegetarian).
19 ingredient(bacon,lactose_free,gluten_free,no_vegetarian).
20 ingredient(sausage,lactose_free,gluten_free,no_vegetarian).
21 ingredient(ham,lactose_free,gluten_free,no_vegetarian).
22 ingredient(egg,lactose_free,gluten_free,vegetarian).
23 ingredient(rice,lactose_free,gluten_free,vegetarian).
24 ingredient(strawberry,lactose_free,gluten_free,vegetarian).
25 ingredient(peach,lactose_free,gluten_free,vegetarian).
26 ingredient(apple,lactose_free,gluten_free,vegetarian).
27 ingredient(ladyfinger,lactose_free,gluten_free,vegetarian).
28 ingredient(salmon,lactose_free,gluten_free,vegetarian).
29 ingredient(coffee,lactose_free,gluten_free,vegetarian).
30 ingredient(water,lactose_free,gluten_free,vegetarian).
31 ingredient(sparkling_water,lactose_free,gluten_free,vegetarian).
32 ingredient(sugar,lactose_free,gluten_free,vegetarian).
33 ingredient(salt,lactose_free,gluten_free,vegetarian).
34 ingredient(rosemary,lactose_free,gluten_free,vegetarian).
35 ingredient(oil,lactose_free,gluten_free,vegetarian).
36 ingredient(flour,lactose_free,gluten_free,vegetarian).
37 ingredient(pasta,lactose_free,gluten_free,vegetarian).
38 ingredient(bread,lactose_free,gluten_free,vegetarian).
39 ingredient(lemon,lactose_free,gluten_free,vegetarian).
40 ingredient(onion,lactose_free,gluten_free,vegetarian).
41 ingredient(lettuce,lactose_free,gluten_free,vegetarian).
42 ingredient(zucchini,lactose_free,gluten_free,vegetarian).
43 ingredient(eggplant,lactose_free,gluten_free,vegetarian).
44 ingredient(peas,lactose_free,gluten_free,vegetarian).
45 ingredient(potato,lactose_free,gluten_free,vegetarian).
46 ingredient(carrot,lactose_free,gluten_free,vegetarian).

```

Figure 3.1: Ingredients in Prolog

```

1 /* Meals */
2 meal("Arrosti_di_pesce",[seabass,salmon,lemon,salt,oil,rosemary],medium).
3 meal("Origliata_di_pesce",[seabass,salmon,lemon,salt,oil,rosemary],low).
4 meal("Scoppione_di_fegato",[beef,champignon,flour,oil,salt,rosemary],low).
5 meal("Meatloaf",[beef,eggplant,tomato_sauce,onion,bread,oil,salt,parmesan],high).
6 meal("Insalata",[lettuce,tomato,carrot,oil,salt],low).
7 meal("Patate_al_forno",[potato,rosemary,oil,salt],medium).
8 meal("Patatine_fritte",[potato,oil,salt,ketchup,mayonnaise],high).
9 meal("Vongole_all'olio",[vongole,tomato_sauce,zucchini,oil,salt,rosemary],low).
10 meal("Pizza_margherita",[flour,water,salt,oil,mozzarella,tomato_sauce],medium).
11 meal("Pizza_salame",[flour,water,salt,oil,mozzarella,salmon],medium).
12 meal("Pizza_mozzarella",[flour,water,salt,oil,mozzarella,champignon,sausage],high).
13 meal("Pizza_morcina",[flour,water,salt,oil,mozzarella,truffle,champignon,sausage],high).
14 meal("Vegetable_Pizza",[flour,water,salt,oil,mozzarella,eggplant,onion,zucchini,carrot,tomato,champignon],medium).
15 meal("Pizza_potato_e_bacon",[flour,water,salt,oil,mozzarella,tomato_sauce,potato,bacon],high).
16 meal("Pizza_grosciatto",[flour,water,salt,oil,mozzarella,tomato_sauce,ham],high).
17 meal("Pizza_mozzarella_e_grosciatto",[flour,water,salt,oil,mozzarella,tomato_sauce,ham,eggplant],high).
18 meal("Pizza_zucchini_e_salsiccia",[flour,water,salt,oil,mozzarella,sausage,zucchini],high).
19 meal("Tiramisu",[ladyfinger,coffee,mascarpone],high).
20 meal("Macedonia",[peach,apple,strawberry,sugar,lemon],low).
21 meal("Torture",[salt,salmon,avocado,lemon,tomato],high).
22 meal("Risotto_al_rago",[bread,flour,rice,salt,water,mozzarella,onion,beef,oil,tomato_sauce],high).
23 meal("Vegetarian_grosciatto",[bread,flour,rice,salt,water,mozzarella,onion,oil,tomato_sauce],medium).
24 meal("Risotto_al_grosciatto",[bread,flour,rice,salt,water,mozzarella,ham],medium).
25 meal("Tagliere",[ham,mozzarella,pecorino,parmesan,ricotta],high).
26 meal("Insalata_russa",[potato,peas,carrot,mayonnaise,egg,oil,salt],high).
27 meal("Cordon_rosso",[pasta,egg,pecorino,bacon],high).
28 meal("Pasta_salame_pomodoro",[pasta,salmon,tomato_sauce,salt,oil],low).
29 meal("Pasta_alie_comuni",[pasta,clams,oil,salt],low).
30 meal("Gnocchi_al_sugo_di_papere",[flour,potato,egg,salt,duck,onion,carrot,tomato_sauce,parmesan,oil],high).
31 meal("Risotto_alie_vongole",[rice,clams,oil,salt],low).
32 meal("Risotto_salsiccia_papere",[rice,sausage,champignon,onion,oil,salt],medium).
33 meal("Risotto_al_rago",[pecorino,ricotta,salt,flour,egg,onion,tomato_sauce,beef,oil],high).
34 meal("Vegetarian_vegetal",[pecorino,ricotta,salt,flour,egg,onion,tomato_sauce,oil],medium).
35 meal("Purgifiume",[eggplant,tomato_sauce,mozzarella,parmesan,onion,oil,salt],high).
36 meal("Arrosti_di_carne",[beef,sausage,lemon,salt,oil,rosemary],medium).
37 meal("Origliata_di_carne",[beef,sausage,lemon,salt,oil,rosemary],low).
38

```

Figure 3.2: Meals in Prolog

```

1 /* Find dishes that meet the calorie criterion */
2 meal_by_calories(CalorieLevel, Meal) :-
3     calorie_order(CalorieLevel, Level),
4     meal(Meal, _, MealCalories),
5     calorie_order(MealCalories, MealLevel),
6     MealLevel <= Level.
7
8 check_preferences(Meal, Preferences) :-
9     (member(lactose_free, Preferences) -> meal_lactose_free(Meal) ; true),
10    (member(gluten_free, Preferences) -> meal_gluten_free(Meal) ; true),
11    (member(vegetarian, Preferences) -> meal_vegetarian(Meal) ; true).
12
13 find_meals(Preferences, CalorieLevel, Meals) :-
14     findall(
15         Meal,
16         (
17             meal(Meal, _, _),
18             check_preferences(Meal, Preferences),
19             meal_by_calories(CalorieLevel, Meal)
20         ),
21         Meals
22     ).
23

```

Figure 3.3: Dishes and checks

high respectively. And then the meal is returned. It shows the preferences, it evaluates whether the meal is returned as the preferences of the customer or not. It checks first when the meal is analyzed, and then second checks for gluten-free, lactose-free and vegetarian. If the preference is not provided this check is skipped and returns the subsequent meals. The `find_meals(Preferences, CalorieLevel, Meals)` predicate is designed to generate a list of meals that align with a customer's dietary preferences and calorie requirements.

It takes three arguments:

1. **Preferences** – A list of dietary restrictions (e.g., lactose-free, gluten-free, vegetarian).
2. **CalorieLevel** – The desired calorie range (low, medium, or high).
3. **Meals** – The resulting list of meals that meet the criteria.

The predicate uses `findall` to construct this list by ensuring:

- The meal exists in the database (`meal/3`).
- The meal matches the customer's preferences (`check_preferences/2`).
- The meal aligns with the specified calorie level (`meal_by_calories/2`).

Meals that meet all conditions are included in the result list, making this predicate a key component for generating personalized menu recommendations based on dietary and caloric needs.

The image below inserted shows the rules for the prolog where it returns the value according to the asked predicates. This rule contains lactose-free, gluten-free and vegetarian.

This image shows a few rules, for example, lactose free checks whether the meal is lactose free, gluten free checks whether the meal is gluten free and vegetarian checks whether the meal is vegetarian.

```
1 /*Allergies + Vegetarian*/
2 is_lactose_free(X) :- ingredient(X,lactose_free,_,_).
3 is_gluten_free(X) :- ingredient(X,_,gluten_free,_).
4 is_vegetarian(X) :- ingredient(X,_,_,vegetarian).
5 |
```

Figure 3.4: Allergies and vegetarian

```
1 /*check meals */
2 meal_lactose_free(Meal) :- meal(Meal, Ingredients, _),
3   forall(member(Ingredient, Ingredients), is_lactose_free(Ingredient)).
4
5 meal_gluten_free(Meal) :-
6   meal(Meal, Ingredients, _),
7   forall(member(Ingredient, Ingredients), is_gluten_free(Ingredient)).
8
9 meal_vegetarian(Meal) :-
10  meal(Meal, Ingredients, _),
11  forall(member(Ingredient, Ingredients), is_vegetarian(Ingredient)).
12
```

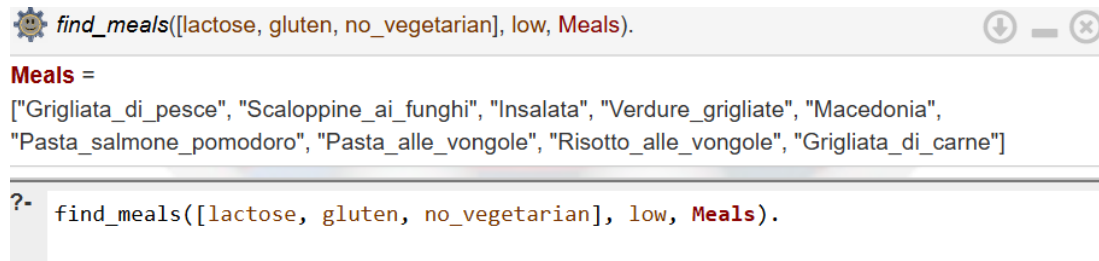
Figure 3.5: Checking Meals

3.2 Simulation and checks

The simulation has been performed with these queries inserted below. As well as the results for these queries are Inserted below.

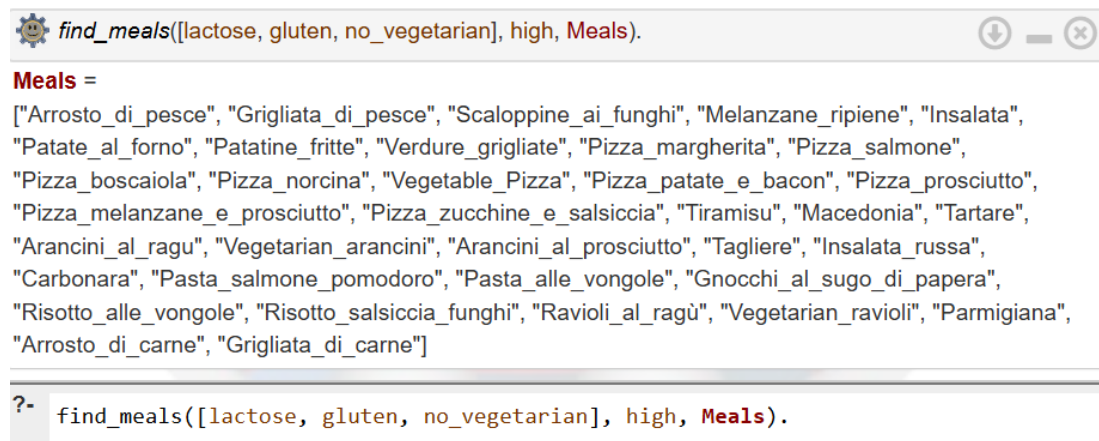
```
1 /*find_meals([lactose, gluten, no_vegetarian], low, Meals).*/
2 /*find_meals([lactose, gluten, no_vegetarian], high, Meals).*/
3 /*find_meals([lactose_free, gluten_free, vegetarian], low, Meals).*/
4 /*find_meals([lactose_free, gluten_free, vegetarian], high, Meals).*/|
```

Figure 3.6: Queries



```
find_meals([lactose, gluten, no_vegetarian], low, Meals).  
  
Meals =  
["Grigliata_di_pesce", "Scaloppine_ai_funghi", "Insalata", "Verdure_grigliate", "Macedonia",  
"Pasta_salmone_pomodoro", "Pasta_alle_vongole", "Risotto_alle_vongole", "Grigliata_di_carne"]  
  
?- find_meals([lactose, gluten, no_vegetarian], low, Meals).
```

Figure 3.7: Lactose, Gluten, No vegetarian for low calories



```
find_meals([lactose, gluten, no_vegetarian], high, Meals).  
  
Meals =  
["Arrosto_di_pesce", "Grigliata_di_pesce", "Scaloppine_ai_funghi", "Melanzane_ripiene", "Insalata",  
"Patate_al_forno", "Patatine_fritte", "Verdure_grigliate", "Pizza_margherita", "Pizza_salmone",  
"Pizza_boscaiola", "Pizza_norcina", "Vegetable_Pizza", "Pizza_patate_e_bacon", "Pizza_prosciutto",  
"Pizza_melanzane_e_prosciutto", "Pizza_zucchine_e_salsiccia", "Tiramisu", "Macedonia", "Tartare",  
"Arancini_al_ragu", "Vegetarian_arancini", "Arancini_al_prosciutto", "Tagliere", "Insalata_russa",  
"Carbonara", "Pasta_salmone_pomodoro", "Pasta_alle_vongole", "Gnocchi_al_sugo_di_papera",  
"Risotto_alle_vongole", "Risotto_salsiccia_funghi", "Ravioli_al_ragù", "Vegetarian_ravioli", "Parmigiana",  
"Arrosto_di_carne", "Grigliata_di_carne"]  
  
?- find_meals([lactose, gluten, no_vegetarian], high, Meals).
```

Figure 3.8: Lactose, Gluten, No vegetarian for high calories

4. Knowledge Graph / Ontology

Ontology engineering is a key area in knowledge representation that deals with creating, developing, and managing ontologies. In this context, an **ontology** is a structured and formal specification of a shared understanding within a domain.

It defines:

- **Entities** within a specific field
- **Relationships** between those entities
- **Rules** governing their interactions

By providing a clear framework for organizing knowledge, ontology engineering helps structure complex information, making it easier to understand, process, and utilize.

4.1 Syntax

The ontology I created is stored in a **Turtle (TTL) file**, a widely used syntax for representing **RDF (Resource Description Framework)** data in a structured yet human-readable format.

RDF is a framework for describing resources on the web using **triples**, which consist of:

- **Subject** (the resource being described)
- **Predicate** (the relationship or property)
- **Object** (the value or related resource)

Turtle improves readability by allowing **prefixes** to shorten long **URIs (Uniform Resource Identifiers)**, making RDF data easier to write, read, and understand compared to formats like **RDF/XML**.

4.2 Protege

I used **Protégé** for the ontology engineering tasks. Developed by **Stanford University**, Protégé is an open-source **ontology editor** that provides a powerful yet user-friendly environment for creating, editing, and managing ontologies.

Protégé supports multiple ontology languages, including:

- **OWL (Web Ontology Language)**

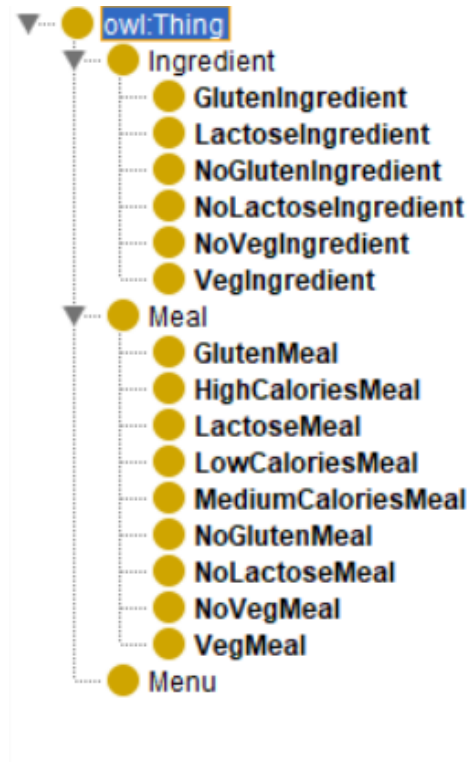


Figure 4.1: The overview of the ontology

- **RDF (Resource Description Framework)**

Its flexibility and comprehensive toolset make it an essential platform for developing structured knowledge representations.

4.2.1 The ontology

Classes

The ontology is structured around three primary classes: **Ingredient**, **Meal**, and **Menu**, as illustrated in Figure. Each class plays a distinct role in representing key elements of the domain. The **Ingredient** class encompasses all the basic components used to prepare meals (see Table 1.1). The **Meal** class defines individual dishes, characterized by their ingredients (see Table 1.2). Finally, the **Menu** class represents all possible combinations of meals, enabling the creation of personalized menus tailored to the preferences or dietary restrictions of a guest.

Ingredient Class

The **Ingredient** class is used to represent the basic building blocks of meals. Each **Ingredient** object is characterized by four key data properties that define its attributes, as illustrated in Figure :

- **hasLactose**: This property specifies whether an ingredient contains lactose. It has a range of `xsd:boolean`, allowing values of `true` or `false`.

- **isVegetarian:** This property indicates if an ingredient contains meat. Its range is `xsd:boolean`, with possible values of `true` or `false`.
- **hasGluten:** This property determines whether an ingredient contains gluten or is gluten-free. It has a range of `xsd:boolean`, meaning it can be either `true` or `false`.
- **hasName:** This property provides the name of the ingredient. Unlike the other three data properties, this one has a range of `xsd:string`, meaning it can hold any text value representing the ingredient's name.

Meal Class

The **Meal** class represents individual dishes and their properties, as well as their relationships to ingredients. Each **Meal** object is characterized by five key data properties, which provide detailed information about each meal and are essential for categorizing and managing meals within our system. These properties are illustrated in Figure 4.6:

- **hasName:** This property specifies the name of the meal. The range is `xsd:string`, allowing it to hold any text value representing the meal's name. In the example shown in Figure, this property is set to "Macedonia".
- **hasCalories:** This property indicates the caloric level of the meal. The range is `xsd:string`, which can represent various calorie levels, such as `Low`, `Medium`, or `High`. In the example shown in Figure, this property is set to "Low".
- **MealIsVegetarian:** This property denotes whether the meal is vegetarian. The range is `xsd:boolean`, indicating whether the meal is suitable for vegetarians (`true`) or not (`false`). In the example in Figure, this property is set to `true`.
- **MealHasGluten:** This property specifies whether the meal contains gluten. The range is `xsd:boolean`, with `true` indicating the presence of gluten and `false` indicating its absence. In the example in Figure, this property is set to `false`.
- **MealHasLactose:** This property indicates whether the meal contains lactose. Its range is `xsd:boolean`, with `true` denoting the presence of lactose and `false` denoting its absence. In the example in Figure 4.6, this property is set to `false`.

Upon selecting a Meal you can see all the ingredients inside the MealClass as an example here well as calories details also Lactose, Gluten and Vegetarian tables.

4.2.2 SPARQL Queries

SPARQL is a language and protocol designed for querying and managing data stored in **RDF** (Resource Description Framework) format. It allows us to retrieve and update information from RDF graphs, either on the web or within RDF databases.

In the project, I used **GraphDB**, a powerful RDF database that supports SPARQL queries. It provides advanced features to handle and query large-scale RDF datasets effectively.

While I ran many different queries, most of them were quite similar, with the main difference being the specific class of data they targeted. Therefore, we'll show just one

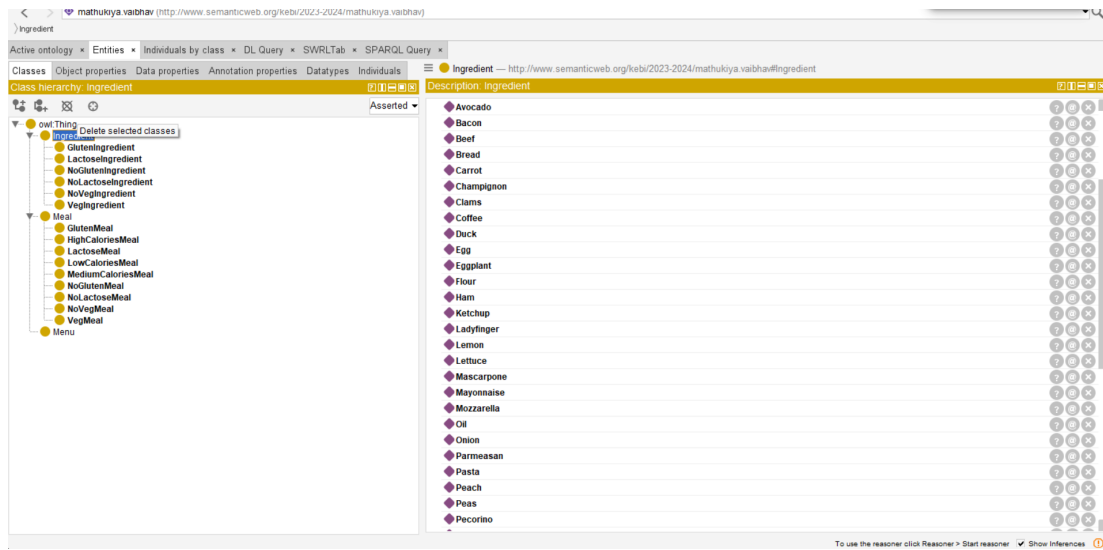


Figure 4.2: Ontology of Ingredients

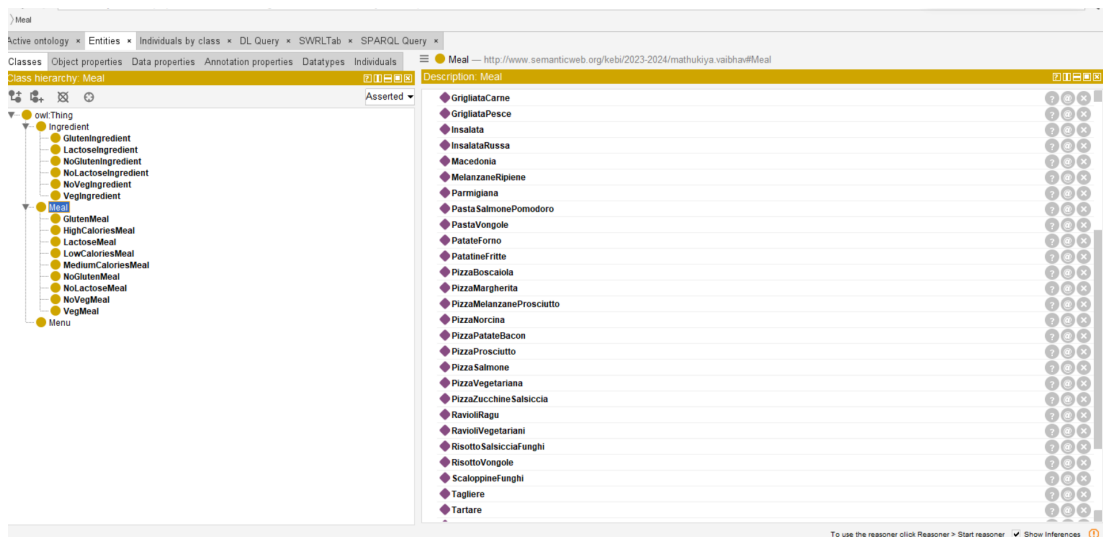


Figure 4.3: Ontology of Meals

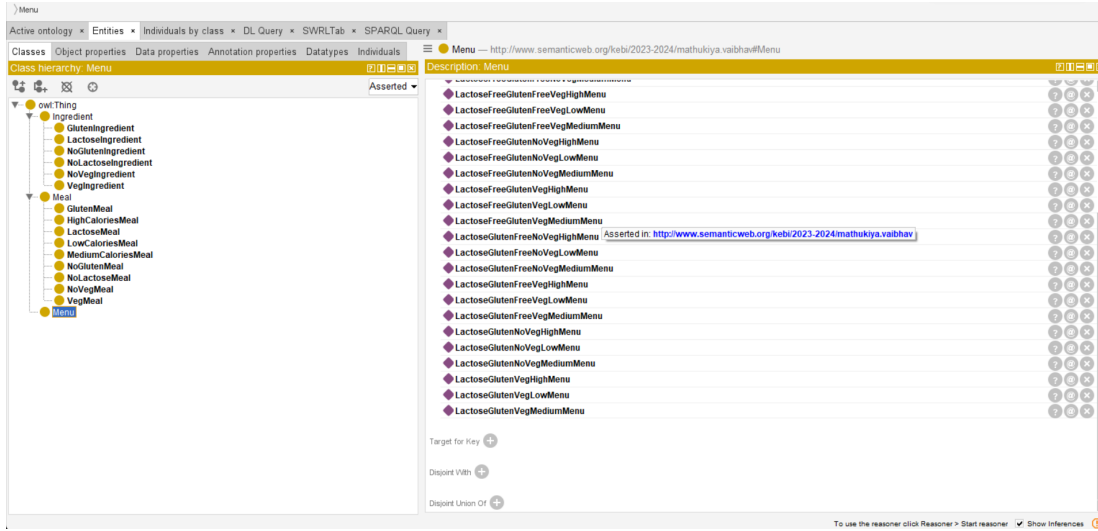


Figure 4.4: Ontology for Menu

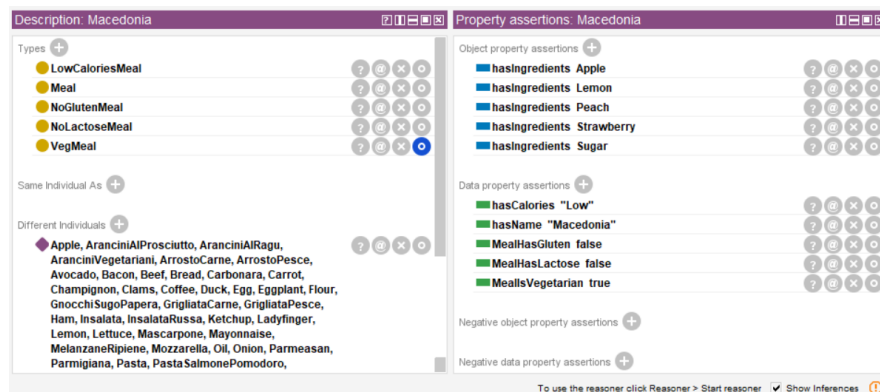


Figure 4.5: Example for MealClass for Macedonia

```
# Apply these prefixes to each query
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vaibhav: <http://www.semanticweb.org/kebi/2023-2024/mathukiya.vaibhav#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

Figure 4.6: Prefix

```
# Query to get only the preferred meals (simplified)
SELECT DISTINCT ?meal (STR(?rawName) AS ?name)
WHERE {
  # Define preferences
  BIND(false AS ?checkLactose)
  BIND(false AS ?checkGluten)
  BIND(false AS ?checkVegetarian)
  BIND("Low" AS ?caloriesLevel)

  # Basic meal information
  ?meal rdf:type vaibhav:Meal.
  ?meal vaibhav:hasName ?rawName.

  # Apply filters
  FILTER(
    # Lactose filter
    (?checkLactose = false || NOT EXISTS { ?meal rdf:type vaibhav:LactoseMeal }) &&

    # Gluten filter
    (?checkGluten = false || NOT EXISTS { ?meal rdf:type vaibhav:GlutenMeal }) &&

    # Vegetarian filter
    (?checkVegetarian = false || NOT EXISTS { ?meal rdf:type vaibhav:NoVegMeal }) &&

    # Calories filter (simplified)
    (?caloriesLevel = "" ||
     (?caloriesLevel = "Low" && EXISTS { ?meal rdf:type vaibhav:LowCaloriesMeal }) ||
     (?caloriesLevel = "Medium" && EXISTS { ?meal rdf:type vaibhav:MediumCaloriesMeal }) ||
     (?caloriesLevel = "High" && EXISTS { ?meal rdf:type vaibhav:HighCaloriesMeal })))
  )
}
```

Figure 4.7: Basic Queries

example of a typical query. We will also share the query that provides the final result of the project, which displays the meals chosen by the client.

To make the queries easier to understand, I used **prefixes** (as shown in **Figure**). Prefixes are shorthand for long **URIs** (Uniform Resource Identifiers), making the queries more readable by allowing us to use short, simple names instead of the full URLs.

For this I have created a basic set of queries which I have included for the basic interpretation of the set of query which I have uploaded on the Github.

4.3 SHACL Shapes

SHACL (Shapes Constraint Language) is a language used to validate RDF graphs by defining specific rules, known as *shapes*, which are also expressed as RDF graphs. These *shapes* specify the conditions that RDF data must satisfy to be considered valid.

SHACL shapes help in various tasks, including:

- **Data validation:** Ensuring the data meets certain criteria or rules.
- **User interface development:** Assisting in building interfaces that work with valid RDF data.
- **Integration:** Helping in the integration of RDF data from different sources by ensuring compatibility and correctness.

In short, SHACL provides a way to define structured criteria that RDF data must follow, helping ensure data integrity and consistency.

```

#Shape to add constraints to the GlutenIngredient class.
vaibhav:GlutenIngredientShape
  a sh:NodeShape ;
  sh:targetClass vaibhav:GlutenIngredient ; # Applies to all gluten ingredients
  sh:property [
    sh:path vaibhav:hasGluten ; #add constraints to hasGluten property
    sh:hasValue true;
  ].

#Shape to add constraints to the NoGlutenIngredient class.
vaibhav:NoGlutenIngredientShape
  a sh:NodeShape ;
  sh:targetClass vaibhav:NoGlutenIngredient ; # Applies to all non-gluten ingredients
  sh:property [
    sh:path vaibhav:hasGluten ; #add constraints to hasGluten property
    sh:hasValue false;
  ].

#Shape to add constraints to the LactoseIngredient class.
vaibhav:LactoseIngredientShape
  a sh:NodeShape ;
  sh:targetClass vaibhav:LactoseIngredient ; # Applies to all lactose ingredients
  sh:property [
    sh:path vaibhav:hasLactose ; #add constraints to hasLactose property
    sh:hasValue true;
  ].

#Shape to add constraints to the NoLactoseIngredient class.
vaibhav:NoLactoseIngredientShape
  a sh:NodeShape ;
  sh:targetClass vaibhav:NoLactoseIngredient ; # Applies to all non-lactose ingredients
  sh:property [
    sh:path vaibhav:hasLactose ; #add constraints to hasLactose property
    sh:hasValue false;
  ].

```

Figure 4.8: SHACL Constraints

I have added several constraints in SHACL for adding Shapes. Some of them will be mentioned here and more about it is all mentioned in the Github file.

I have inserted a figure here to demonstrate some of the constraints available, other constraints can't be possible to add her because of lack of space and length of the document.

4.3.1 SWRL Rules

SWRL (**Semantic Web Rule Language**) is a rule-based language that extends OWL (**Web Ontology Language**) by adding logic rules to ontologies. These rules consist of an **antecedent (body)** and a **consequent (head)**, forming an implication. Whenever the conditions in the antecedent are met, the consequent must also be true. SWRL rules allow for advanced reasoning in knowledge systems, making logical inferences based on defined data. I put some SWRL rules to categorize both **ingredients** and **meals** in the ontology.

1. **Rule for Ingredient with Lactose:** $\text{vaibhav:Ingredient}(?x) \wedge \text{vaibhav:hasLactose}(?x, \text{true}) \rightarrow \text{vaibhav:LactoseIngredient}(?x)$
2. **Rule for Ingredient without Lactose:** $\text{vaibhav:Ingredient}(?x) \wedge \text{vaibhav:hasLactose}(?x, \text{false}) \rightarrow \text{vaibhav:NoLactoseIngredient}(?x)$
3. **Rule for Meal with Lactose:** $\text{vaibhav:LactoseIngredient}(?x) \wedge \text{vaibhav:isInMeal}(?x, ?y) \rightarrow \text{vaibhav:LactoseMeal}(?y)$
4. **Rule for Meal without Lactose:** $\text{vaibhav:Meal}(?x) \wedge \text{vaibhav:MealHasLactose}(?x, \text{false}) \rightarrow \text{vaibhav:NoLactoseMeal}(?x)$

And like this there are many other rules which has been uploaded in my github.

5. Agile and Ontology-Based Meta-modelling

The methodology of **Agile and Ontology-Based Metamodeling** is the subject of this chapter and is directed towards making modeling languages more flexible for various domains. By bringing together **agile development principles** and **ontology-based frameworks**, this methodology provides a dynamic and adaptive way of developing and evolving complex models.

5.1 AOAME

For **agile and ontology-based metamodeling** task, I used **AOAME**. AOAME supports the creation and management of **Enterprise Knowledge Graph (EKG) schemas** for the organization of domain-specific knowledge for **analysis, reasoning, and integration** across different data sources.

AOAME supports **dynamic modifications** through **meta-modeling operators** that generate **SPARQL queries** and update the **triplestore** such that the model and data are consistent. It was implemented using **Jena Fuseki Java libraries** and is highly reusable for real-world applications. It played a important part in adapting **BPMN 2.0**, allowing to extend its **existing classes and properties** to meet our project's specific requirements.

5.2 BPMN 2.0

Business Process Model and Notation 2.0 (BPMN 2.0) is one of the most widely used graphical modeling standards for business processes. It provides **clear and structured notations** that enable organizations to communicate workflows more effectively, track performance, and optimize transactions.

BPMN 2.0 includes essential elements such as:

- **Activities, Events, and Gateways** – Representing activities, triggers, and gateways.
- **Flows** – Defining the sequence and interactions of processes.

By making it easier to solve **the gap between process design and execution**, BPMN 2.0 offers a **standardized and effective** way of **documenting and analyzing** business processes.

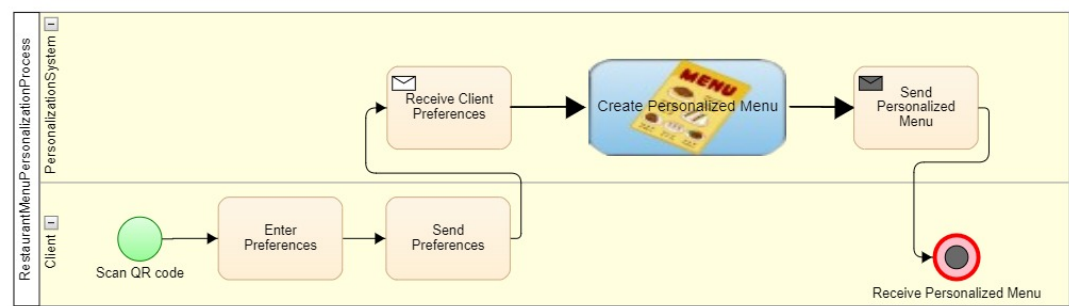


Figure 5.1: BPMN model

Relation	Value	Actions
isVegetarian	<input type="text" value="false"/>	<button>Remove</button>
isGlutenIntolerant	<input type="text" value="true"/>	<button>Remove</button>
isLactoseIntolerant	<input type="text" value="true"/>	<button>Remove</button>
preferredCaloriesLe	<input type="text" value="High"/>	<button>Remove</button>

lo:preferredCalories... ▼ Add Relation

Figure 5.2: Example

5.2.1 The model

Figure 1 illustrates the BPMN 2.0 model created using AOAME. It represents the process of generating a personalized menu for a client, organized into two distinct pools: one for the Client and one for the System, clearly defining their interactions.

The process begins when the client scans a QR code to initiate the workflow. The client then enters their preferences—such as dietary restrictions and allergies—and submits them to the system. Once received, the system forwards this information to an extended class, **CreatePersonalizedMenu**, specifically designed to select meals that align with the client’s preferences. After personalizing the menu, the system sends it back to the client, concluding the process when the client receives their customized menu.

The **CreatePersonalizedMenu** class extends the **Task** class and includes the data properties shown in Figure 5.2. The properties **isVegetarian**, **isGlutenIntolerant**, and **isLactoseIntolerant** are of boolean type, while **preferredCaloriesLevel** is a string with possible values of “Low,” “Medium,” or “High.”

5.3 Jena Fuseki

Jena Fuseki is a SPARQL server and a component of the Apache Jena framework, designed for hosting RDF (Resource Description Framework) data and providing a platform for querying and managing this data using SPARQL. Fuseki supports both SPARQL query (for reading data) and SPARQL update (for modifying data) operations. It can run as a standalone server, offering HTTP endpoints for querying and updating RDF data, or be embedded in applications, making it a versatile tool for developing semantic web applications and knowledge graph systems.

5.3.1 Queries

Understanding the Query: Finding Meals Based on User Preferences

This SPARQL query is designed to find meals based on specific preferences that a user may have, such as whether they are lactose intolerant, gluten intolerant, vegetarian, or what kind of calorie level they prefer in their meals. Let's break it down:

1. User Preferences:

- **Lactose Intolerance:** The user is not lactose intolerant (indicated by `false`).
- **Gluten Intolerance:** The user is not gluten intolerant (again, `false`).
- **Vegetarian:** The user is not specifically asking for vegetarian meals (again, `false`).
- **Calories:** The user prefers low-calorie meals ("`Low`").

These preferences are set at the beginning using the `BIND` command. This is like telling the query, "Here's what the user prefers."

2. Meal Information: The query looks for meals (`?meal`) that are of the type `vaibhav:Meal` and retrieves the name of each meal (`?rawName`).

3. Filters to Match Preferences: The query then uses **filters** to make sure the meals match the user's preferences.

- **Lactose Filter:** If the user isn't lactose intolerant, the query will ignore meals that contain lactose. But if the user were lactose intolerant, it would only return lactose-free meals.
- **Gluten Filter:** Similar to lactose, if the user isn't gluten intolerant, the query will ignore meals containing gluten. For a gluten-intolerant user, only gluten-free meals would be selected.
- **Vegetarian Filter:** If the user does not require vegetarian meals, the query excludes meals that are not vegetarian.
- **Calories Filter:** If the user prefers low-calorie meals, the query will only return meals that are categorized as low-calorie. If the user's preference were high or medium-calorie meals, the query would adjust accordingly.

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vaibhav: <http://www.semanticweb.org/kebi/2023-2024/mathukiya.vaibhav#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

# Query to get only the preferred meals
SELECT DISTINCT ?meal (STR(?rawName) AS ?name)
WHERE {
  # Define preferences - these can be set dynamically based on user input
  BIND(false AS ?checkLactose) # User is not lactose intolerant
  BIND(false AS ?checkGluten) # User is not gluten intolerant
  BIND(false AS ?checkVegetarian) # User does not require vegetarian
  BIND("Low" AS ?caloriesLevel) # User prefers low calorie meals

  # Basic meal information
  ?meal rdf:type vaibhav:Meal.
  ?meal vaibhav:hasName ?rawName.

  # Lactose filter
  FILTER(?checkLactose = false || NOT EXISTS { ?meal rdf:type vaibhav:LactoseMeal })

  # Gluten filter
  FILTER(?checkGluten = false || NOT EXISTS { ?meal rdf:type vaibhav:GlutenMeal })

  # Vegetarian filter
  FILTER(?checkVegetarian = false || NOT EXISTS { ?meal rdf:type vaibhav:NoVegMeal })

  # Calories filter
  FILTER(
    ?caloriesLevel = "" ||
    ?caloriesLevel = "High" ||
    (?caloriesLevel = "Low" && EXISTS { ?meal rdf:type vaibhav:LowCaloriesMeal }) ||
    (?caloriesLevel = "Medium" && EXISTS { ?meal rdf:type vaibhav:MediumCaloriesMeal })
  )
}

```

Figure 5.3: Jena Fuseki Query

4. What the Query Returns: The query will return a list of meals (?meal) that meet the user’s preferences (like lactose-free, gluten-free, vegetarian, or low-calorie) along with their names (?name). In short, this query helps find meals that match specific dietary needs based on what the user prefers.

This way, the query ensures that only the meals that match the user’s needs (like being lactose-free or low-calorie) are selected. The results are also mentioned below.

mealName	
1	Macedonia
2	Arrosto di carne
3	Arrosto di pesce
4	Tartare
5	Grigliata di carne
6	Insalata
7	Insalata russa
8	Risotto salsiccia e funghi
9	Risotto alle vongole
10	Verdure grigliate
11	Grigliata di pesce
12	Patatine fritte
13	Patate al forno

Showing 1 to 13 of 13 entries

Figure 5.4: Results of query

6. Conclusion

Camunda

In the project, **Camunda DMN** was used to manage the decision-making process for selecting dishes based on customer preferences, dietary restrictions, and calorie levels. We developed two main decision tables:

1. **Selected Ingredients:** This table filters available ingredients based on customer specifications, such as lactose intolerance, gluten intolerance, and vegetarian preferences.
2. **Final Meals:** This table connects to the first one and matches the filtered ingredients to available dishes. It applies additional filters based on calorie levels (e.g., "low," "medium," or "high"), providing a personalized list of dishes that fit both the dietary requirements and the customer's preferences.

The decision logic was automated using **DMN** (Decision Model and Notation), which eliminated the need for manual intervention and standardized the process. For example, the "list contains" function in the **Selected Ingredients** table automatically checks if each dish contains the required ingredients. The **Collect** hit policy was used to aggregate multiple results, providing a complete list of suitable dishes instead of just one option.

However, I faced some challenges:

- **Camunda** no longer offers an integrated environment for simulating DMN diagrams. This meant I couldn't directly test the model with real-world scenarios, requiring to make manual assumptions about the results.

Prolog

I used **Prolog** to manage the logic behind dish selection, particularly for filtering dishes based on customer dietary preferences. I defined **predicates** to describe ingredients and dishes, associating attributes like lactose, gluten, and vegetarian compatibility with each ingredient.

Key predicates included:

- `is_lactose_free`, `is_gluten_free`, and `is_vegetarian`: These predicates check if an ingredient meets specific dietary requirements.
- `check_preferences`: This predicate filters dishes based on user preferences, such as lactose intolerance, gluten intolerance, or vegetarian diet.

-
- **find_meals:** This predicate gathers all dishes that match the specified preferences and calorie level, using the `meal_by_calories` function.

While I attempted to modify the `check_preferences` code to align with **Prolog** standards, I faced difficulties in adapting the logic due to the complexity of the code. Despite efforts to simplify and optimize, I couldn't get the code to work as intended.

Protégé

Protégé proved to be a highly effective tool for structuring and managing information through ontology creation. It allowed me to define **classes**, **attributes**, and **entities**, organizing the data clearly and effectively. I used the tool to model complex relationships between different elements in our domain.

Key features of **Protégé**:

- **Graph Diagrams:** These helped visualize the hierarchical structure of our ontology.
- **OWL Support:** Protégé made it easier to define relationships and attributes using **OWL** (Web Ontology Language).
- **SWRL Rules:** I used **SWRL** (Semantic Web Rule Language) rules for type inference, though we found them useful only in simpler cases.
- **SHACL Validator:** This tool helped us validate RDF data against predefined constraints, ensuring data consistency and correctness.

Challenges included:

- **Complexity in Ontology Modeling:** I faced a steep learning curve and had to spend significant time understanding how to model the knowledge base effectively.
- **Manual Data Entry:** Entering large amounts of data manually was slow, error-prone, and time-consuming, especially for meal and ingredient instances.

AOAME

AOAME was valuable for integrating **BPMN 2.0** (Business Process Model and Notation) with the ontology. This tool allowed me to create intuitive, easy-to-read diagrams for restaurant managers. I also used **Apache Jena Fuseki** to integrate our ontology into a triplestore, enabling me to run **SPARQL** queries and dynamically generate personalized menu suggestions based on customer preferences.

Benefits of **AOAME**:

- It enabled the creation of a more readable diagram through **BPMN 2.0** integration.
- **SPARQL queries** provided an easy way to suggest meals based on user input.

Challenges:

- **Bugs and Stability Issues:** AOAME had several bugs affecting user experience and system stability.
- **Performance Issues:** The online version struggled during periods of high traffic, which affected the system's efficiency.

Despite these limitations, **AOAME** was a powerful tool for the project.

Future Developments

Looking ahead, there are several areas where the ontology and system could be enhanced:

- **Complex Relationships:** I could add more complex relationships between ingredients, dishes, and categories (e.g., temporal relationships for meals at different times of day like breakfast, lunch, or dinner).

This revision organizes the content into distinct sections and makes the information clearer and easier to follow.

6.1 Links

1. SPARQL - [LINK](#)
2. BPM -[LINK](#)
3. AGILE - [LINK](#)