



University of Camerino

SCHOOL OF SCIENCE AND TECHNOLOGY

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

Personalized Menu System

Authors:

Micarelli Simone

simone.micarelli@studenti.unicam.it

Supervisor

Prof. Knut Hinkelmann

Nappo Davide

davide.nappo@studenti.unicam.it

Prof. Emanuele Laurenzi

Contents

1	Introduction	9
1.1	Project Description	9
1.2	Project Tasks	9
2	System Design	11
2.1	Input format	11
2.2	Output format	13
3	Knowledge Based Solution	15
3.1	Decision Model	15
3.1.1	Camunda	16
3.1.2	Our solution	16
3.1.3	Decision Tables	16
3.1.4	Literal Expression	19
3.1.5	DMN simulation	19
3.2	Prolog	20
3.2.1	Prolog Implementation	21
3.3	Ontology Engineering	27
3.3.1	Our Ontology	28
3.3.2	SWRL Rules	33
3.3.3	SPARQL Query	36
3.3.4	SHACL Shape	43
4	Agile and Ontology-based Meta-Modelling	47
4.1	AOAME	47
4.2	BPMN 2.0	47
4.2.1	Our BPMN 2.0 Model	48
5	Conclusions	55
5.1	Davide Nappo	55
5.2	Simone Micarelli	56

Listings

3.1	Ingredient facts	21
3.2	Ingredients/Kcals association	22
3.3	Carnivorous ingredients	23
3.4	Vegetarian ingredients	23
3.5	Ingredients associated with an intolerance	23
3.6	Meal facts	24
3.7	Rules for: meal categories	24
3.8	Rules for: meal allergies	25
3.9	Rules to compute the total kcals of a meal, and calorie-conscious levels .	25
3.10	Rule for: guest preferences	26
3.11	Prolog testing queries	27
3.12	Prefixes for all the SPARQL Queries	36
3.13	Query to retrieve all meals labeled as carnivorous	36
3.14	Query to retrieve all the ingredients in a specific meal	38
3.15	Advanced query to display meals based on the preferences of the guest .	38
3.16	Vegetarian meals	40
3.17	Meals and related calories	40
3.18	Guests with allergies	40
3.19	Kcal of all ingredients	41
3.20	'calorie-conscious' meals of level 3	41
3.21	Kcal for a specific ingredient	41
3.22	Gluten intolerance: ingredients	41
3.23	Lactose intolerance: ingredients	41
3.24	Gluten intolerance: meals - REASONER Protégè	41
3.25	Lactose intolerance: meals - REASONER Protégè	42
3.26	Meals classified as 'appetizer'	42
3.27	Meals classified as 'first-dish'	42
3.28	Ingredient that are classified as 'Seafood'	42
3.29	Meals that have more than 500 kcal	42
3.30	'Restaurant' SHACL Shape	43
3.31	'Menu' SHACL Shape	43
3.32	'Guest' SHACL Shape	44
3.33	'Meal' SHACL Shape	44

List of Figures

2.1	Sample Input form for the customer	12
2.2	Sample filled out input form	13
3.1	Our decision model in Camunda	16
3.2	Ingredients choosing (1)	17
3.3	Ingredients choosing (2)	17
3.4	Meals suggestion (1)	18
3.5	Meals suggestion (2)	18
3.6	Ingredients list literal expression	19
3.7	DMN first implementation	20
3.8	DMN second implementation	20
3.9	Prolog Testing (1)	26
3.10	Prolog Testing (2)	27
3.11	Our ontology graph in Protégè	28
3.12	Our ontology model in Protégè	29
3.13	Our model's object's properties	30
3.14	Our model's data properties	31
3.15	Some individuals in our model	31
3.16	Individual 'our-restaurant' in our model	32
3.17	Individual 'italian-menu' in our model	32
3.18	Individual 'guest1' in our model	32
3.19	Individual 'rigatoni-all-a-carbonara' in our model	33
3.20	Rule to infere meals to eat - gluten intolerance	34
3.21	Rule to infere meals to eat - lactose intolerance	34
3.22	SWRL Testing (1) - 'guest1' meals suggestion	35
3.23	SWRL Testing (2) - 'Meal: 'rigatoni-all-a-carbonara'	35
3.24	SWRL Testing (3) - 'guest3' meals suggestion	36
3.25	SPARQL Query with Protégè: Carnivorous	37
3.26	SPARQL Query with GraphDB: Carnivorous	37
3.27	SPARQL Query with Protégè: Ingredients in a meal	38
3.28	SPARQL Query with GraphDB: Ingredients in a meal	38
3.29	'guest1' properties	39
3.30	SPARQL Query with Protégè: 'guest1' meals suggestion	39
3.31	SPARQL Query with GraphDB: 'guest1' meals suggestion	40

3.32 SHACL validator	45
4.1 BPMN 2.0 graph built in AOAME	48
4.2 Query 1: Used to view the properties and their associated values for the extended class 'SuggestMeal'.	49
4.3 Query 2: Used to analyze the extended class and return the meals that satisfy all the described data properties.	49
4.4 Defining the attributes of the extended class. In this example, the lactose allergy is set together with the vegetarian category.	50
4.5 Result of Query 2 using the parameters defined in figure 4.4	50
4.6 Defining the attributes of the extended class. In this example, the gluten allergy is set together with the vegetarian category.	51
4.7 Result of Query 2 using the parameters defined in figure 4.6	51
4.8 Defining the attributes of the extended class. In this example, the lactose allergy is set together with the carnivorous category.	52
4.9 Result of Query 2 using the parameters defined in figure 4.8	52
4.10 Defining the attributes of the extended class. In this example, the lactose allergy is set together with the carnivorous category and 'Main Dish' course type.	53
4.11 Result of Query 2 using the parameters defined in figure 4.10	53

1. Introduction

This document aims to outline the process undertaken to complete the Knowledge Engineering and Business Intelligence project. Section 1.1 will provide a description of the system to be developed, while Section 1.2 will define the tasks to be accomplished. Following this, we will examine the System Design phase (Chapter 2), the implemented Knowledge-Based Solutions (Chapter 3), and the description of our implementation of Agile and Ontology-based Meta-Modelling (Chapter 4). In the Conclusion chapter (Chapter 5), we will offer our personal perspectives on the solutions presented. The solution developed can be found at this link: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project

1.1 Project Description

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

1.2 Project Tasks

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

1. Create different knowledge-based solutions for recommending food depending on the profile of a guest (carnivores, vegetarians, calorie conscious, suffering from allergies, etc.) using the following representation languages:
 - Decision tables (including DRD with sub-decisions and corresponding decision tables);
 - Prolog (including facts and rules);

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

2. System Design

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

2.1 Input format

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

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

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

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

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

Input format

Personalized Menu System

B I U ↵ ⌂

@davide.nappo@studenti.unicam.it
@simone.micarelli@studenti.unicam.it

What type of user are you?

Vegetarian
 Carnivorous

Do you suffer from any allergies?

Lactose
 Gluten

How calorie-conscious are you?

0 1 2 3

If you want, type a meal for further information (ingredients and kcal)

Testo risposta breve

Type a meal's ingredient to see the calories

Testo risposta breve

Figure 2.1: Sample Input form for the customer

2.2 Output format

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

Personalized Menu System

*@davide.nappo@studenti.unicam.it
@simone.micarelli@studenti.unicam.it*

simone.micarelli@studenti.unicam.it Cambia account Bozza salvata

What type of user are you?

Vegetarian
 Carnivorous

[Cancella selezione](#)

Do you suffer from any allergies?

Lactose
 Gluten

How calorie-conscious are you?

0 1 2 3

[Cancella selezione](#)

If you want, type a meal for further information (ingredients and kcal)

Rigatoni alla carbonara

Type a meal's ingredient to see the calories

La tua risposta

Invia [Cancella modulo](#)

Figure 2.2: Sample filled out input form

3. Knowledge Based Solution

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

3.1 Decision Model

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

DMN Elements

The DMN standard consists of four main components:

- Decision Requirements Diagrams: Illustrate the relationships among the various decision-making elements, creating a network of dependencies.
- Decision tables¹: Present a clear and concise visual format for defining actions based on specific conditions.
- Business context: Refers to the set of contextual factors that influence the decision-making process within an organization.
- Friendly Enough Expression Language (FEEEL): A dedicated language used to evaluate expressions within decision tables.

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

3.1.1 Camunda

Camunda² is an open source platform developed in Java, designed to support workflow and process automation for organizations of any size. Its capabilities include creating decision models as well as, more broadly, BPMN process diagrams and DMN decision tables. In addition, Camunda provides REST APIs³, enabling developers who do not work with Java to access its functionalities. In our project, we used Camunda to build the knowledge base for meal management and to define decision-making processes and rules consistent with our organizational objectives.

3.1.2 Our solution

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

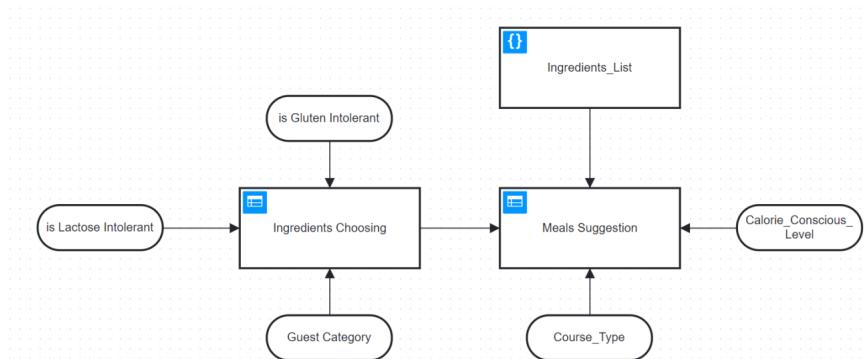


Figure 3.1: Our decision model in Camunda

3.1.3 Decision Tables

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

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

²Camunda. URL: <https://camunda.com/>

³Camunda Documentation. URL: <https://docs.camunda.org/manual/7.19/reference/dmn/feel/>

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

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

Figure 3.2: Ingredients choosing (1)

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

Figure 3.3: Ingredients choosing (2)

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

Decision Model

with level 3 corresponding to meals containing fewer than 250 kcal.

Meals Suggestion		Hit policy:	Collect						
When	Ingredients	And	Ingredients_List	And	Calorie_Conscious_Level	And	Course_Type	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 primavera"
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 primavera"
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 primavera"
4	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients,		0		"Appetizer"				"Involtini primavera"

Figure 3.4: Meals suggestion (1)

Meals Suggestion		Hit policy:	Collect						
When	Ingredients	And	Ingredients_List	And	Calorie_Conscious_Level	And	Course_Type	Then	Meals
21	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Pasta"].calories[1] + Ingredients_List[name="Garlic"].calories[1] + Ingredients_List[name="Chili"].calories[1]) <= 250	3		"First-Dish"				"Spaghetti aglio olio e peperoncino"
22	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Pasta"].calories[1] + Ingredients_List[name="Garlic"].calories[1] + Ingredients_List[name="Chili"].calories[1]) <= 450	2		"First-Dish"				"Spaghetti aglio olio e peperoncino"
23	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Pasta"].calories[1] + Ingredients_List[name="Garlic"].calories[1] + Ingredients_List[name="Chili"].calories[1]) <= 650	1		"First-Dish"				"Spaghetti aglio olio e peperoncino"
24	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")		0		"First-Dish"				"Spaghetti aglio olio e peperoncino"

Figure 3.5: Meals suggestion (2)

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

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

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

Alongside the Calorie-Conscious Level, the user also specifies the Course Type, which is used to further refine the selection and produce a list of suitable meal

options. The Decision Table's output is therefore a tailored list of meals matching the user's preferences.

3.1.4 Literal Expression

To optimize the workflow, we created a Literal Expression that stores ingredients alongside their respective calorie values in a Map structure, where each ingredient is paired with a numerical value indicating its caloric content. A partial view of Literal Expression 3.6 is shown below.

```
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}], Variable name: Ingredients_List, Variable type: , Expression language: feel
```

Figure 3.6: Ingredients list literal expression

3.1.5 DMN simulation

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

Inputs:

Decision table:	All tables
Ingredients Choosing	
Guest Category:	omnivore
is Lactose Intolerant:	True
is Gluten Intolerant:	False
Meals Suggestion	
Calorie_Conscious_Level:	2
Course_Type:	Appetizer

Outputs:

Ingredients:	Carrot, Cabbage, Lettuce, Eggplant, Shrimp, Ciauscolo, Salame, He...
Meals:	Involtini primavera, Affettati Misti
Ingredients_List:	{name=Oil, calories=60}, {name=Eggplant, calories=18}, {name=Cc...

Simulate now

Index	Guest Category	Calorie_Conscious_Level	Annotations
33	"carnivorous", "vegetarian", 0	-	false
34	"carnivorous", "vegetarian", 0	-	false
35	"carnivorous", "vegetarian", 0	-	false
36	"carnivorous", "vegetarian", 0	false	-
37	"carnivorous", "vegetarian", 0	false	-
38	"carnivorous", "vegetarian", 0	false	-

Figure 3.7: DMN first implementation

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

Inputs:

Decision table:	All tables
Ingredients Choosing	
Guest Category:	vegetarian
is Lactose Intolerant:	True
is Gluten Intolerant:	False
Meals Suggestion	
Calorie_Conscious_Level:	2
Course_Type:	Appetizer

Outputs:

Ingredients:	Carrot, Cabbage, Lettuce, Eggplant, Egg, Oil, Salt, Black pepper, Ch...
Meals:	Involtini primavera
Ingredients_List:	{name=Oil, calories=60}, {name=Eggplant, calories=18}, {name=Cc...

Simulate now

When	And	And	Then	Annotations
Guest Category	is Lactose Intolerant	is Gluten Intolerant	Ingredients	
"carnivorous", "vegetarian", omn...	boolean	boolean	"Oil", "Eggplant", "Cow's cheese"	
1 "vegetarian", "omnivore"	-	-	"Carrot"	
2 "vegetarian", "omnivore"	-	-	"Cabbage"	
3 "vegetarian", "omnivore"	-	-	"Lettuce"	
4 "vegetarian", "omnivore"	-	-	"Eggplant"	
5 "carnivorous", "omnivore"	-	-	"Shrimp"	
6 "carnivorous", "omnivore"	-	-	"Ciauscolo"	

Figure 3.8: DMN second implementation

3.2 Prolog

Logic programming employs logic as a declarative representation language, using backward chaining as its primary inference mechanism. Prolog⁴ is a logic programming language closely linked to artificial intelligence and computational linguistics, whose strength lies in its logical foundations.

The syntax of Prolog includes:

- **Symbols:** (and), ; (or), :- (if), not (not);

⁴ Prolog Wikipedia. URL: <https://it.wikipedia.org/wiki/Prolog>

- **Variables:** a sequence of letters, digits, and underscores (_), beginning with either an uppercase letter or an underscore;
- **Facts:** predicate statements that declare something about the problem domain;
- **Rules:** predicate statements that use logical implication (:-) to define relationships between facts. A Prolog rule has the structure:

left_hand_side :- right_hand_side.

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

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

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

3.2.1 Prolog Implementation

We implemented the model defined in the DMN file using Prolog and tested it through SWISH Prolog⁵.

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

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

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

```

1 % Define the list of ingredients used in meals
2 ingredient(salt).
3 ingredient(oil).
4 ingredient(carrot).
5 ingredient(cabbage).
```

⁵Prolog Testing. URL: <https://swish.swi-prolog.org/>

```

6  ingredient(puff_pastry).
7  ingredient(ciauscolo).
8  ingredient(salame).
9  ingredient(ham).
10 ingredient(black_pepper).
11 ingredient(cow_cheese).
12 ingredient(sheep_cheese).
13 ingredient(parmesan).
14 ingredient(mozzarella_cheese).
15 ingredient(pasta).
16 ingredient(egg).
17 ingredient(pork_cheek).
18 ingredient(shrimp).
19 ingredient(mussel).
20 ingredient(chili).
21 ingredient(garlic).
22 ingredient(tomato).
23 ingredient(bechamel).
24 ingredient(minced_meat).
25 ingredient(pork_chop).
26 ingredient(pork_rib).
27 ingredient(sausage).
28 ingredient(steak).
29 ingredient(calamari).
30 ingredient(sword_fish).
31 ingredient(salmon).
32 ingredient(potato).
33 ingredient(eggplant).
34 ingredient(type_0_flour).
35 ingredient(water).
36 ingredient(sugar_and_food_coloring).
37 ingredient(corn).
38 ingredient(lettuce).
39 ingredient(apple).
40 ingredient(banana).
41 ingredient(strawberry).

```

Listing 3.1: Ingredient facts

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

```

1 % Define the caloric content of each ingredient
2 kcal_ingredient(salt, 0).
3 kcal_ingredient(oil, 60).
4 kcal_ingredient(carrot, 27).
5 kcal_ingredient(cabbage, 25).
6 kcal_ingredient(puff_pastry, 150).
7 kcal_ingredient(ciauscolo, 70).
8 kcal_ingredient(salame, 50).
9 kcal_ingredient(ham, 45).
10 kcal_ingredient(black_pepper, 25).
11 kcal_ingredient(cow_cheese, 120).
12 kcal_ingredient(sheep_cheese, 135).
13 kcal_ingredient(parmesan, 85).
14 kcal_ingredient(mozzarella_cheese, 120).
15 kcal_ingredient(pasta, 300).
16 kcal_ingredient(egg, 80).
17 kcal_ingredient(pork_cheek, 150).
18 kcal_ingredient(shrimp, 35).
19 kcal_ingredient(mussel, 40).
20 kcal_ingredient(chili, 15).
21 kcal_ingredient(garlic, 3).
22 kcal_ingredient(tomato, 30).
23 kcal_ingredient(bechamel, 105).
24 kcal_ingredient(minced_meat, 120).

```

```

25 kcal_ingredient(pork_chop, 125).
26 kcal_ingredient(pork_rib, 70).
27 kcal_ingredient(sausage, 205).
28 kcal_ingredient(steak, 485).
29 kcal_ingredient(calamari, 50).
30 kcal_ingredient(sword_fish, 120).
31 kcal_ingredient(salmon, 150).
32 kcal_ingredient(potato, 450).
33 kcal_ingredient(eggplant, 18).
34 kcal_ingredient(type_0_flour, 500).
35 kcal_ingredient(water, 0).
36 kcal_ingredient(sugar_and_food_coloring, 465).
37 kcal_ingredient(corn, 50).
38 kcal_ingredient(lettuce, 25).
39 kcal_ingredient(apple, 35).
40 kcal_ingredient(banana, 40).
41 kcal_ingredient(strawberry, 22).

```

Listing 3.2: Ingredients/Kcals association

This section lists all ingredients classified as carnivorous. If a meal contains a carnivorous ingredient and no vegetarian ingredients, then that meal is considered carnivorous.

```

1 % Define which ingredients are carnivorous
2 ingredient_carnivore(ciauscolo).
3 ingredient_carnivore(salame).
4 ingredient_carnivore(ham).
5 ingredient_carnivore(pork_cheek).
6 ingredient_carnivore(minced_meat).
7 ingredient_carnivore(pork_rib).
8 ingredient_carnivore(sausage).
9 ingredient_carnivore(shrimp).
10 ingredient_carnivore(mussel).
11 ingredient_carnivore(steak).
12 ingredient_carnivore(calamari).
13 ingredient_carnivore(sword_fish).
14 ingredient_carnivore(salmon).

```

Listing 3.3: Carnivorous ingredients

This section lists all ingredients classified as vegetarian. If a meal contains a vegetarian ingredient and no carnivorous ingredients, then that meal is considered vegetarian.

```

1 % Define which ingredients are vegetarian
2 ingredient_vegetarian(carrot).
3 ingredient_vegetarian(cabbage).
4 ingredient_vegetarian(eggplant).
5 ingredient_vegetarian(lettuce).
6 ingredient_vegetarian(apple).
7 ingredient_vegetarian(banana).
8 ingredient_vegetarian(strawberry).

```

Listing 3.4: Vegetarian ingredients

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

```

1 % Define which ingredients cause lactose intolerance
2 ingredient_with_lactose_intolerance(cow_cheese).
3 ingredient_with_lactose_intolerance(sheep_cheese).
4 ingredient_with_lactose_intolerance(parmesan).
5 ingredient_with_lactose_intolerance(mozzarella_cheese).
6 ingredient_with_lactose_intolerance(bechamel).
7
8 % Define which ingredients contain gluten
9 ingredient_with_gluten_intolerance(type_0_flour).
10 ingredient_with_gluten_intolerance(pasta).
11 ingredient_with_gluten_intolerance(puff_pastry).

```

Listing 3.5: Ingredients associated with an intolerance

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

```

1 % Define various meals along with their courses and ingredients
2 meal(involtini_prima_vera, appetizer, [salt, oil, carrot, cabbage, puff_pastry]).
3 meal(affettati_misti, appetizer, [salt, ciauscolo, salame, ham, black_pepper]).
4 meal(formaggi_misti, appetizer, [salt, cow_cheese, sheep_cheese, parmesan]).
5 meal(tagliatelle_alla_marinara, first_dish, [salt, oil, pasta, shrimp, mussel]).
6 meal(rigatoni_alla_carbonara, first_dish, [salt, oil, pasta, sheep_cheese, egg,
7     black_pepper, pork_cheek]).
8 meal(trofie_cacio_e_pepe, first_dish, [salt, oil, pasta, black_pepper,
9     sheep_cheese]).
10 meal(spaghetti_aglio_olio_e_peperoncino, first_dish, [salt, oil, pasta, garlic,
11     chili]).
12 meal(lasagne, first_dish, [salt, oil, carrot, pasta, parmesan, tomato, bechamel,
13     minced_meat]).
14 meal(grigliata_di_maiale_mista, second_course, [salt, pork_rib, sausage, pork_chop]).
15 meal(bistecca_alla_fiorentina, second_course, [salt, oil, steak]).
16 meal(pesce_arrosto, second_course, [salt, oil, shrimp, mussel, calamari, sword_fish,
17     salmon]).
18 meal(verdure_miste, second_course, [salt, cabbage, eggplant, lettuce]).
19 meal(patatine_fritte, sidedish, [salt, oil, potato]).
20 meal(melanze_grigliate, sidedish, [salt, oil, eggplant]).
21 meal(insalata_mista, sidedish, [salt, oil, lettuce, corn]).
22 meal(pizza_margherita, main_dish, [salt, oil, cow_cheese, tomato, water,
23     type_0_flour]).
24 meal(pizza_bianca, main_dish, [salt, oil, water, type_0_flour]).
25 meal(water, drink, [water]).
26 meal(coca_colada, drink, [water, sugar_and_food_coloring]).
27 meal(fruit_salad, dessert, [apple, banana, strawberry]).
```

Listing 3.6: Meal facts

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

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

These predicates allow meals to be categorized based on dietary preferences, helping users choose options aligned with their nutritional requirements.

```

1 % Determine if a meal is vegetarian
2 vegetarian_meal(Meal, Course) :-
3     meal(Meal, Course, Ingredients),
4     forall(member(Ingredient, Ingredients),
5         (ingredient_vegetarian(Ingredient); \+ ingredient_carnivore(Ingredient))).
6
7 % Determine if a meal is carnivorous
8 carnivorous_meal(Meal, Course) :-
9     meal(Meal, Course, Ingredients),
10    % Ensure all ingredients are either carnivorous or not vegetarian
11    forall(member(Ingredient, Ingredients),
12        (ingredient_carnivore(Ingredient); \+ ingredient_vegetarian(Ingredient))).
13
14 % Define meals that contain both carnivorous and vegetarian ingredients
15 omnivore_meal(Meal, Course) :-
16     meal(Meal, Course, Ingredients),
```

```
17    forall(member(Ingredient, Ingredients), ingredient(Ingredient)).
```

Listing 3.7: Rules for: meal categories

These predicates identify meals containing ingredients associated with gluten or lactose intolerance. They use 'findall' to gather the relevant meal-course pairs and then remove duplicates. Such checks are essential to ensure meals comply with dietary restrictions and to facilitate the creation of customized meal plans for individuals with food intolerances.

```
1 % Find meals with gluten intolerance
2 meal_with_gluten_intolerance(Meal, Course) :- 
3   findall(Meal-Course,
4     (meal(Meal, Course, Ingredients),
5      member(Ingredient, Ingredients),
6      ingredient_with_gluten_intolerance(Ingredient)),
7      MealsWithGlutenIntolerance),
8      list_to_set(MealsWithGlutenIntolerance, UniqueMeals),
9      member(Meal-Course, UniqueMeals).
10
11 % Find meals with lactose intolerance
12 meal_with_lactose_intolerance(Meal, Course) :- 
13   findall(Meal-Course,
14     (meal(Meal, Course, Ingredients),
15      member(Ingredient, Ingredients),
16      ingredient_with_lactose_intolerance(Ingredient)),
17      MealsWithLactoseIntolerance),
18      list_to_set(MealsWithLactoseIntolerance, UniqueMeals),
19      member(Meal-Course, UniqueMeals).
```

Listing 3.8: Rules for: meal allergies

Description:

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

These computations are vital for dietary management, providing clear insights into the energy value of meals and helping users make informed choices aligned with their caloric needs.

```
1 % Calculate the total caloric content of a meal
2 meal_calories(Meal, Course, TotalCalories) :- 
3   meal(Meal, Course, Ingredients),
4   findall(Calories,
5     (member(Ingredient, Ingredients),
6      kcal_ingredient(Ingredient, Kcal),
7      Calories is Kcal),
8      CaloriesList),
9      sum_list(CaloriesList, TotalCalories).
10
11 % Determine calorie-conscious levels based on total calories
12 calorie_conscious_levels(Meal, Course, Levels) :- 
13   meal_calories(Meal, Course, TotalCalories),
14   % Determine the highest applicable level based on total calories
15   ( TotalCalories > 650 -> HighestLevel = 0;
16     TotalCalories =< 250 -> HighestLevel = 3;
17     TotalCalories =< 450 -> HighestLevel = 2;
18     TotalCalories =< 650 -> HighestLevel = 1
19   ),
20   % Generate all levels up to the highest applicable level
```

```
21    findall(Level, (between(0, HighestLevel, Level)), Levels).
```

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

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

```
1  % Filter meals based on guest preferences, including category, calorie level, and
2  allergies
3  guest_preferences(Category, CalorieLevel, Allergies, Meal, Course) :- 
4  % Filtered meals by Category (carnivorous, vegetarian, omnivore)
5  (Category = carnivorous -> carnivorous_meal(Meal, Course));
6  Category = vegetarian -> vegetarian_meal(Meal, Course);
7  Category = omnivore -> omnivore_meal(Meal, Course)
8  ),
9  % Filtered meals by calorie_conscious_level
10 calorie_conscious_levels(Meal, Course, Levels),
11 member(CalorieLevel, Levels),
12 % Filtered meals by allergies
13 (Allergies = none -> true;
14 Allergies = lactose -> not(meal_with_lactose_intolerance(Meal, Course));
15 Allergies = gluten -> not(meal_with_gluten_intolerance(Meal, Course))
16 ).
```

Listing 3.10: Rule for: guest preferences

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

Prolog Testing

To demonstrate its functionality, we include a set of test queries focusing on the '**guest_preferences**' predicate. In this example scenario, if the user is vegetarian, has a moderate to high level of calorie awareness and no allergies, the system returns all matching dishes and their respective courses.

```
 guest_preferences(vegetarian, 3, none, Meal, Course).
Course = second_course,
Meal = verdure_miste
Course = sidedish,
Meal = melanzane_grigliate
Course = sidedish,
Meal = insalata_mista
Course = drink,
Meal = water
Course = dessert,
Meal = fruit_salad

?- guest_preferences(vegetarian, 3, none, Meal, Course).
```

Figure 3.9: Prolog Testing (1)

In the other case described below, if the user is carnivorous, has a moderate to low calorie awareness, and is lactose intolerant, the system returns the following meals: ‘affettati misti’, ‘spaghetti aglio olio e peperoncino’, ‘tagliatelle alla marinara’, ‘grigliata di maiale mista’, and ‘water’.

```

 guest_preferences(carnivorous, 2, lactose, Meal, Course).

Course = appetizer,
Meal = affettati_misti
Course = first_dish,
Meal = tagliatelle_alla_marinara
Course = first_dish,
Meal = spaghetti_aglio_olio_e_peperoncino
Course = second_course,
Meal = grigliata_di_maiale_mista
Course = drink,
Meal = water

?- guest_preferences(carnivorous, 2, lactose, Meal, Course).

```

Figure 3.10: Prolog Testing (2)

Additional example queries can be run in the DMN simulator to generate other results.

```

1 % guest_preferences(vegetarian, 3, none, Meal, second_course).
2 % guest_preferences(carnivorous, 1, none, Meal, Course).
3 % guest_preferences(omnivore, 0, gluten, Meal, Course).
4

```

Listing 3.11: Prolog testing queries

3.3 Ontology Engineering

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

RDF

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

Protégé

Protégé is a free, open-source ontology editor and knowledge management system. It supports the use of the Semantic Web Rule Language (**SWRL**), which enables the definition of inference rules to derive new knowledge from existing ontologies. Moreover,

Protégé integrates the SPARQL Protocol and RDF Query Language (**SPARQL**), a semantic query language designed to retrieve and manipulate RDF data.

3.3.1 Our Ontology

Using Protégé, we developed our ontology by incorporating the core entities of our domain.

We complemented it with SPARQL queries to demonstrate how knowledge can be retrieved in practical scenarios.

A SHACL file was also defined to ensure that constraints on meals and ingredients are consistent and natural.

Furthermore, we introduced object properties to better capture relationships between meals and ingredients, and additional rules in SWRL to represent dietary restrictions and allergies (e.g., gluten and lactose intolerance). These elements allow the ontology to support reasoning about which meals are suitable for different user profiles.

Classes

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

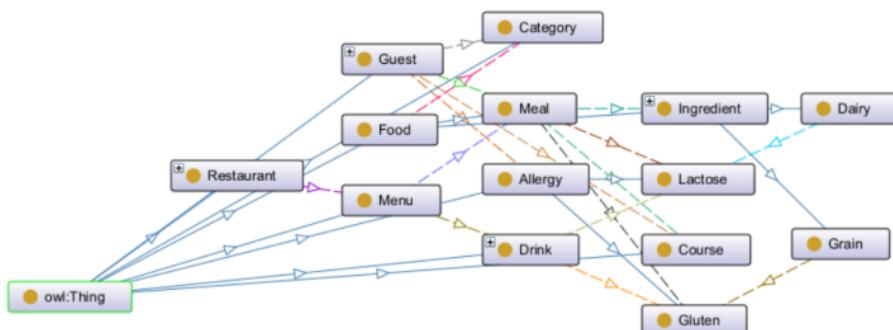


Figure 3.11: Our ontology graph in Protégé

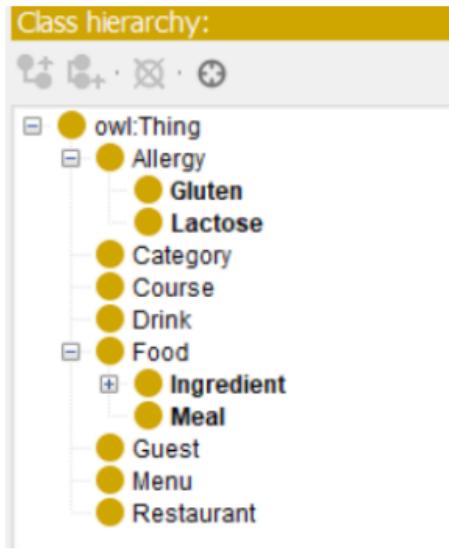


Figure 3.12: Our ontology model in Protègè

Object Properties

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

- **restaurant hasMenu:** associates a menu with a restaurant
- **restaurant hasGuest:** links a guest to a restaurant
- **menu containsDrink:** associates a drink with a menu
- **menu containsMeal:** associates a meal with a menu
- **guest hasAllergies:** indicates that a guest has an allergy
- **guest hasCategory:** links a guest to a dietary category (e.g., carnivorous, vegetarian)
- **guest hasPreferenceCourse:** associates a guest with a preferred course
- **guest isAtRiskForFood:** SWRL rule inferred by the reasoner, used to detect potential food-related risks due to allergies
- **food hasCategory:** associates a food item (ingredient or meal) with a category
- **meal containsGlutenIntolerance:** SWRL rule inferred by the reasoner, identifies meals containing gluten-based ingredients (e.g., grains)
- **meal containsLactoseIntolerance:** SWRL rule inferred by the reasoner, identifies meals containing lactose-based ingredients (e.g., dairy)
- **meal notContainsGlutenIntolerance:** used to check whether a meal is gluten-free

- **meal notContainsLactoseIntolerance:** used to check whether a meal is lactose-free
- **meal hasCourse:** associates a course with a meal
- **meal hasIngredient:** links an ingredient to a meal
- **drink hasIngredient:** links an ingredient to a drink
- **drink containsGlutenIntolerance:** indicates drinks containing gluten
- **drink containsLactoseIntolerance:** indicates drinks containing lactose
- **dairy containsLactoseIntolerance:** links the dairy class to lactose intolerance
- **grain containsGlutenIntolerance:** links the grain class to gluten intolerance

Data Properties

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

- **thing hasName:** attribute shared by all subclasses of 'Thing'; a string specifying the object's name
- **food hasKcal:** attribute for subclasses of 'Food' (Ingredient and Meal); an integer representing the caloric content
- **guest hasLevelOfCalorieConscious:** attribute of the 'Guest' class; an integer from 0 to 3 representing the guest's level of calorie-consciousness

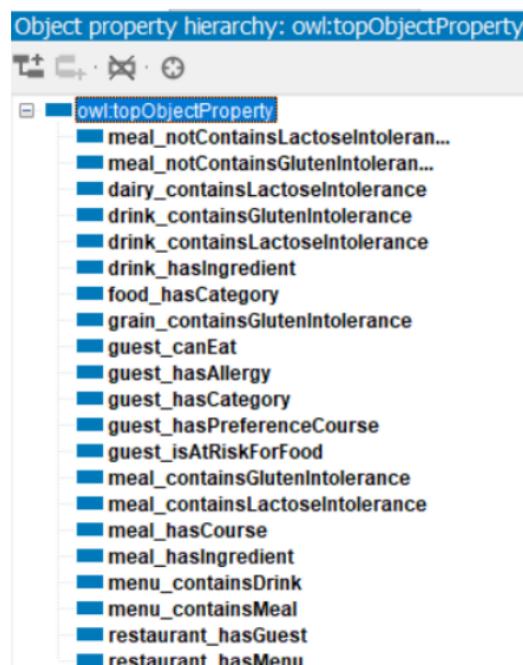


Figure 3.13: Our model's object's properties

- **meal hasLevelOfCalorieConscious:** attribute of the 'Meal' class; an integer from 0 to 3 indicating the level of calorie-consciousness of a meal
- **drink hasKcal:** attribute of the 'Drink' class; an integer representing the caloric content of the drink

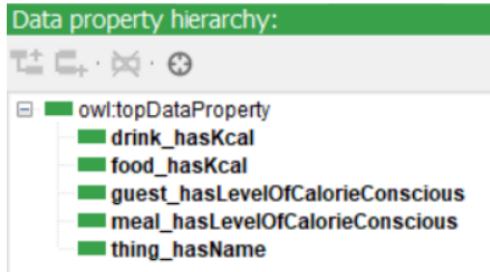


Figure 3.14: Our model's data properties

Individuals

Given the large number of individuals, we provide only a few examples:



Figure 3.15: Some individuals in our model

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

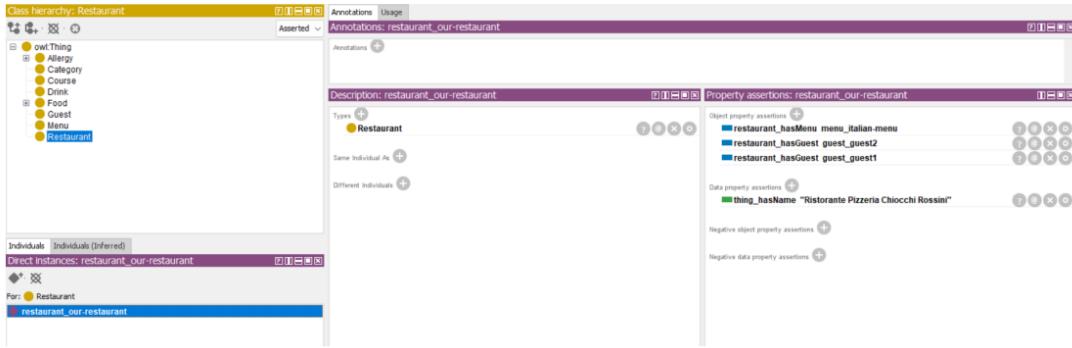


Figure 3.16: Individual 'our-restaurant' in our model

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

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

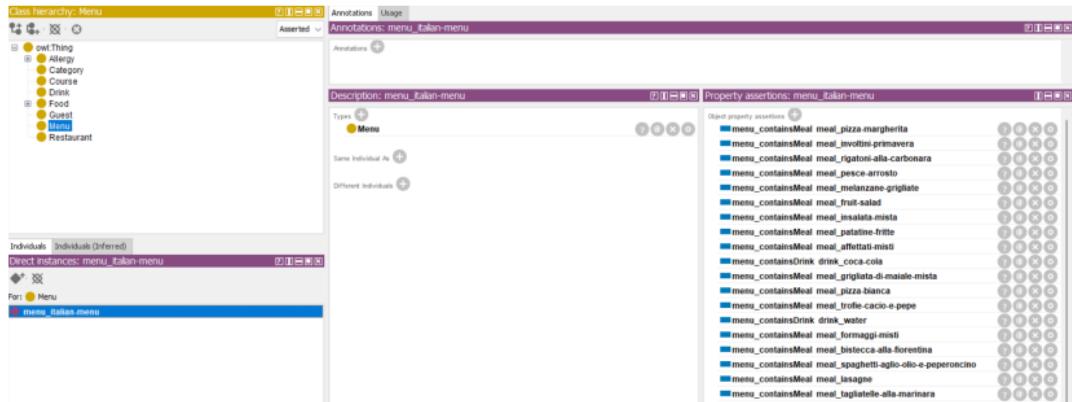


Figure 3.17: Individual 'italian-menu' in our model

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



Figure 3.18: Individual 'guest1' in our model

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

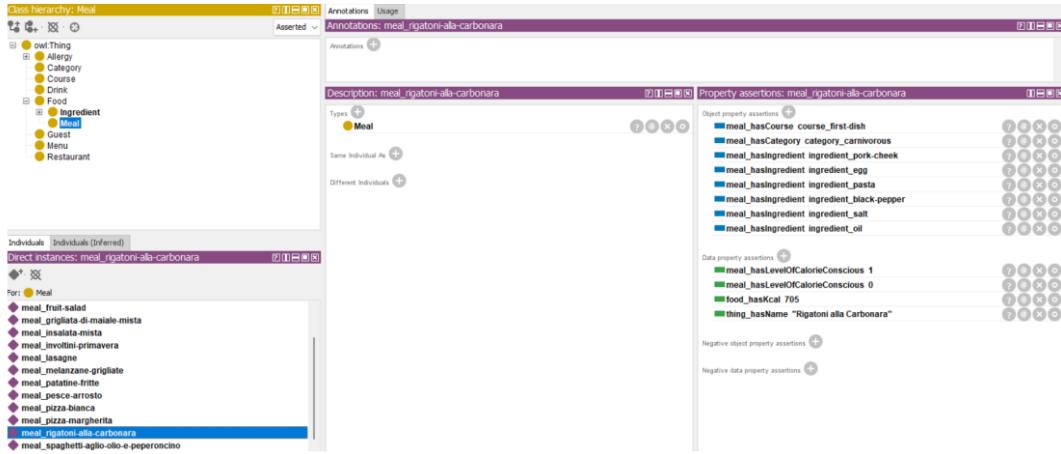


Figure 3.19: Individual 'rigatoni-alla-carbonara' in our model

3.3.2 SWRL Rules

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

GuestIsAtRiskForFoodGluten

$$\begin{aligned} & \text{Guest}(\text{?guest}) \wedge \text{guesthasAllergy}(\text{?guest}, \text{?allergy}) \wedge \text{Meal}(\text{?meal}) \\ & \wedge \text{mealcontainsGlutenIntolerance}(\text{?meal}, \text{?allergy}) \\ & \rightarrow \text{guestisAtRiskForFood}(\text{?guest}, \text{?meal}) \end{aligned}$$

Infers meals that pose a risk to the user based on whether the guest has an allergy (in this case, gluten intolerance) and displays them accordingly.

GuestIsAtRiskForFoodLactose

$$\begin{aligned} & \text{Guest}(\text{?guest}) \wedge \text{guesthasAllergy}(\text{?guest}, \text{?allergy}) \wedge \text{Meal}(\text{?meal}) \\ & \wedge \text{mealcontainsLactoseIntolerance}(\text{?meal}, \text{?allergy}) \\ & \rightarrow \text{guestisAtRiskForFood}(\text{?guest}, \text{?meal}) \end{aligned}$$

Similar to the above, but for lactose intolerance.

MealContainsGlutenIntolerance

$$\begin{aligned} & \text{Meal}(\text{?meal}) \wedge \text{mealhasIngredient}(\text{?meal}, \text{?ingredient}) \wedge \text{Ingredient}(\text{?ingredient}) \\ & \wedge \text{graincontainsGlutenIntolerance}(\text{?ingredient}, \text{?gluten}) \\ & \rightarrow \text{mealcontainsGlutenIntolerance}(\text{?meal}, \text{?gluten}) \end{aligned}$$

Infers meals containing ingredients classified as gluten intolerant. Reasoning: if a meal contains any ingredient with gluten intolerance, the meal itself is classified as containing gluten intolerance.

MealContainsLactoseIntolerance

$$\begin{aligned}
 & \text{Meal}(\text{? meal}) \wedge \text{mealhasIngredient}(\text{? meal}, \text{? ingredient}) \wedge \text{Ingredient}(\text{? ingredient}) \\
 & \wedge \text{dairycontainsLactoseIntolerance}(\text{? ingredient}, \text{? gluten}) \\
 & \rightarrow \text{mealcontainsLactoseIntolerance}(\text{? meal}, \text{? gluten})
 \end{aligned}$$

Infers meals containing ingredients classified as lactose intolerant. Reasoning: if a meal contains any ingredient with lactose intolerance, the meal itself is classified as containing lactose intolerance.

Rule for inferring meals suitable for gluten-intolerant guests
Identifies meals a guest can safely consume, considering both dietary preferences and gluten intolerance.

```

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

```

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

Rule for inferring meals suitable for lactose-intolerant guests
Performs the same function for lactose intolerance.

```

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

```

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

While reasoning over object properties worked as expected, extending inference to certain data properties (e.g., guestHasLevelOfCalorieConscious and mealHasLevelOfCalorieConscious) proved challenging, due to limited support for SWRL built-in atoms in current reasoners. Moreover, the absence of negation support in SWRL required us to introduce two additional object properties: mealNotContainsGlutenIntolerance and mealNotContainsLactoseIntolerance. These adjustments provided a functional way to capture dietary constraints while preserving the overall consistency of the ontology.

SWRL Testing

Below are examples of SWRL rules applied and tested on the ontology:

The screenshot shows two panels of the SWRL Testing interface. The left panel, titled 'Description: guest_quest1', shows a 'Types' section with a 'Guest' type selected. Below it are 'Same Individual As' and 'Different Individuals' sections. The right panel, titled 'Property assertions: guest_quest1', lists object and data property assertions for 'guest1'. Object properties include 'guest_hasAllergy allergy_gluten', 'guest_hasPreferenceCourse course_appetizer', etc. Data properties include 'thing_hasName "Valentino Rossi"', 'guest_hasLevelOfCalorieConscious 2', and 'guest_hasLevelOfCalorieConscious 3'. There are also sections for negative object and data property assertions.

Figure 3.22: SWRL Testing (1) - 'guest1' meals suggestion

As shown in Figure 3.22), high-risk dishes for the user are immediately inferred, and safe foods for 'guest1' (e.g., 'affettati-misti') are correctly identified.

The screenshot shows two panels of the SWRL Testing interface. The left panel, titled 'Description: meal_rigatoni-alla-carbonara', shows a 'Types' section with a 'Meal' type selected. Below it are 'Same Individual As' and 'Different Individuals' sections. The right panel, titled 'Property assertions: meal_rigatoni-alla-carbonara', lists object and data property assertions for 'meal_rigatoni-alla-carbonara'. Object properties include 'meal_hasCourse course_first-dish', 'meal_hasCategory category_carnivorous', etc. Data properties include 'meal_hasLevelOfCalorieConscious 1', 'meal_hasLevelOfCalorieConscious 0', 'thing_hasName "Rigatoni alla Carbonara"', and 'food_hasKcal 705'. There are also sections for negative object and data property assertions.

Figure 3.23: SWRL Testing (2) - 'Meal: 'rigatoni-alla-carbonara'

In this case, the inference is based on whether a meal contains any “allergenic” ingredient.

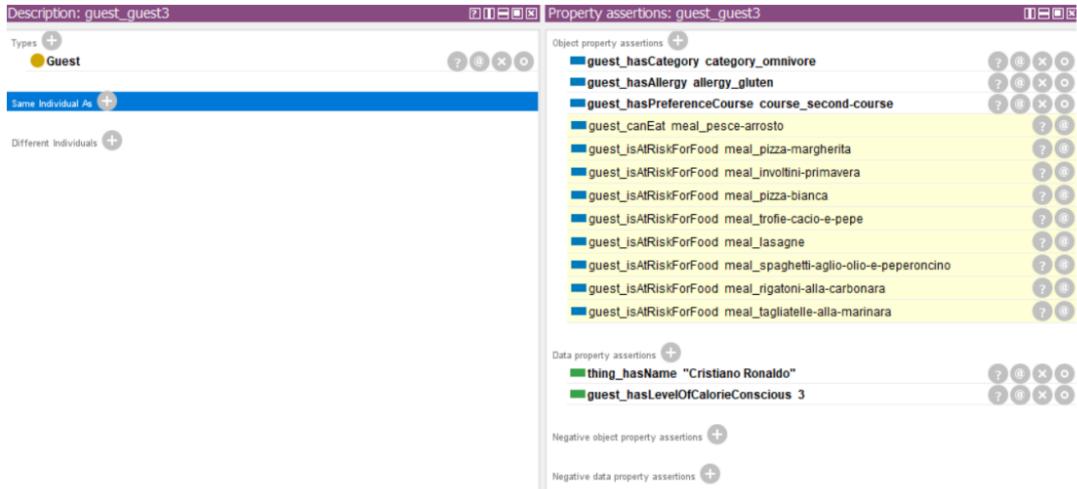


Figure 3.24: SWRL Testing (3) - 'guest3' meals suggestion

In this example, if a guest is omnivorous, highly calorie-conscious, has gluten intolerance, and prefers the second course, the system recommends 'pesce-arrosto'.

3.3.3 SPARQL Query

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

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

In the following subsection, we present some of the queries contained in the file `sparql_queries.txt`⁶, which contains all the queries we developed.

SPARQL Queries Examples

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

```

1 PREFIX owl: <http://www.w3.org/2002/07/owl#>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
5 PREFIX kebi: <http://www.semanticweb.org/simone.micarelli-davide.nappo/kebi2025#>
```

Listing 3.12: Prefixes for all the SPARQL Queries

- **Query for Carnivorous:** Retrieves all meal instances belonging to the 'carnivorous' category.

```

1 SELECT ?meal
2 WHERE {
3   ?meal rdf:type kebi:Meal .
4   ?meal kebi:food_hasCategory kebi:category_carnivorous .
```

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

5 }

Listing 3.13: Query to retrieve all meals labeled as carnivorous

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

Snap SPARQL Query:

```

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

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

```

Execute

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

13 results

Figure 3.25: SPARQL Query with Protègè: Carnivorous

The screenshot shows the GraphDB interface with the following details:

- Left Sidebar:** Contains links for Import, Explore (selected), Monitor, Setup, Lab, and Help.
- Top Bar:** Includes a search icon, "KEBI-Project4", language switch ("en"), and a refresh button.
- Toolbar:** Buttons for "Edit this query", "View raw results", "Results Only", and "Download as".
- Table Headers:** "Filter query results", "Compact view", "Hide row numbers", and a note indicating results from 0 to 13 of 13, with a query time of 0.1s.
- Table Data:** A table titled "meal" showing 13 rows of results, each starting with "kebi:meal_".

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

Figure 3.26: SPARQL Query with GraphDB: Carnivorous

- **Query for retrieve all the ingredients in a meal:** Lists all ingredients of a specific meal (e.g., 'pizza-margherita').

```

1 SELECT ?ingredient
2 WHERE {
3   kebi:meal_pizza-margherita kebi:meal_hasIngredient ?ingredient .
4 }
```

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

Snap SPARQL Query:

```

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

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

Execute

?ingredient

```

kebi:ingredient_salt
kebi:ingredient_0-type-flour
kebi:ingredient_oil
kebi:ingredient_cows-cheese
kebi:ingredient_tomato
```

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

ingredient

```

1 kebi:ingredient_0-type-flour
2 kebi:ingredient_cows-cheese
3 kebi:ingredient_oil
4 kebi:ingredient_salt
5 kebi:ingredient_tomato
```

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

- **Suggested Meals:** Advanced query to display meals based on the preferences of a specific guest (e.g., 'guest1').

```

1 SELECT DISTINCT ?meal ?mealName
2 WHERE {
3   kebi:guest_guest1 a kebi:Guest ;
4   kebi:guest_hasAllergy ?allergy ;
5   kebi:guest_hasCategory ?category ;
6   kebi:guest_hasPreferenceCourse ?course ;
7   kebi:guest_hasLevelOfCalorieConscious ?calorieLevel .
8
9   ?meal a kebi:Meal ;
10  kebi:food_hasCategory ?category ;
11  kebi:meal_hasLevelOfCalorieConscious ?calorieLevel ;
12  kebi:meal_hasCourse ?course ;
13  kebi:thing_hasName ?mealName .
14
15  MINUS { ?meal kebi:meal_containsGlutenIntolerance ?allergy }
16  MINUS { ?meal kebi:meal_containsLactoseIntolerance ?allergy }
```

18 }

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

'guest1' properties:

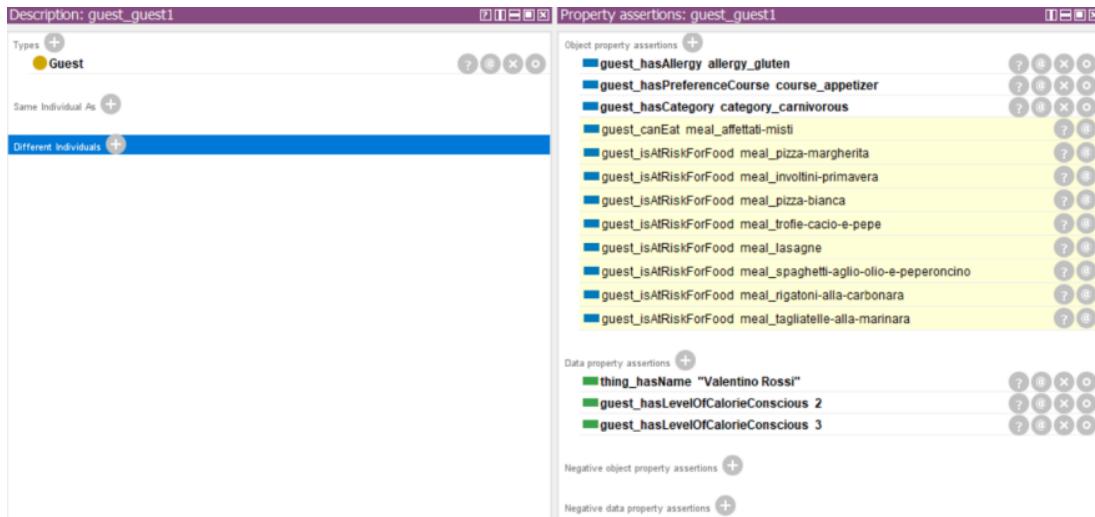


Figure 3.29: 'guest1' properties

Results:

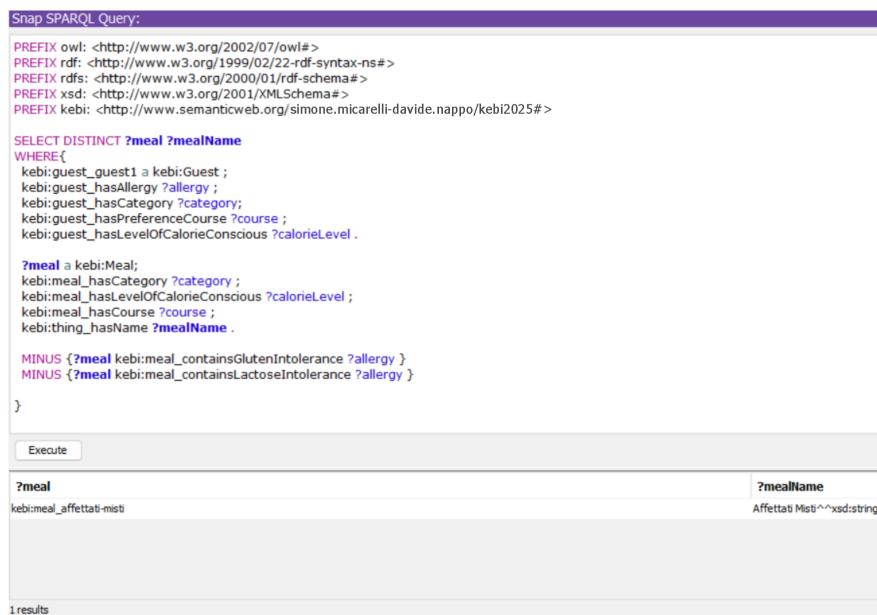


Figure 3.30: SPARQL Query with Protègè: 'guest1' meals suggestion

```

1  SELECT DISTINCT ?meal ?mealName
2  WHERE{
3    ?kebi:guest_guest1 a kebi:Guest ;
4    ?kebi:guest_hasAllergy ?allergy ;
5    ?kebi:guest_hasCategory ?category;
6    ?kebi:guest_hasPreferenceCourse ?course ;
7    ?kebi:guest_hasLevelOfCalorieConscious ?calorieLevel .
8
9    ?meal a kebi:Meal;
10   ?meal kebi:food_hasCategory ?category ;
11   ?kebi:meal_hasLevelOfCalorieConscious ?calorieLevel ;
12   ?kebi:meal_hasCourse ?course ;
13   ?kebi:meal_hasName ?mealName .
14
15
16
17
18
19
20

```

meal	mealName
1 kebi:meal_affettati-misti	'Affettati Misti'

Figure 3.31: SPARQL Query with GraphDB: 'guest1' meals suggestion

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

- **Query to return vegetarian meals**

```

1  SELECT ?meal
2  WHERE {
3    ?meal rdf:type kebi:Meal .
4    ?meal kebi:food_hasCategory kebi:category_vegetarian .
5  }

```

Listing 3.16: Vegetarian meals

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

```

1  SELECT ?meal ?calories
2  WHERE {
3    ?meal rdf:type kebi:Meal .
4    ?meal kebi:food_hasKcal ?calories .
5  }

```

Listing 3.17: Meals and related calories

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

```

1  SELECT ?guest ?allergy
2  WHERE {
3    ?guest a kebi:Guest ;
4    ?kebi:guest_hasAllergy ?allergy .
5  }

```

Listing 3.18: Guests with allergies

- **Query to view the ingredients kcals**

```

1 SELECT ?ingredient ?kcal
2 WHERE {
3     ?ingredient a kebi:Ingredient .
4     ?ingredient kebi:food_hasKcal ?kcal .
5 }
```

Listing 3.19: Kcal of all ingredients

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

```

1 SELECT DISTINCT ?meal ?calorie
2 WHERE {
3     ?meal rdf:type ?type .
4     ?meal kebi:meal_hasLevelOfCalorieConscious ?calorie .
5     FILTER(?calorie = 3)
6 }
```

Listing 3.20: 'calorie-conscious' meals of level 3

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

```

1 SELECT ?kcal
2 WHERE {
3     kebi:ingredient_apple kebi:food_hasKcal ?kcal .
4 }
```

Listing 3.21: Kcal for a specific ingredient

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

```

1 SELECT ?ingredient
2 WHERE {
3     ?ingredient rdf:type kebi:Ingredient .
4     ?ingredient kebi:grain_containsGlutenIntolerance kebi:allergy_gluten .
5 }
```

Listing 3.22: Gluten intolerance: ingredients

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

```

1 SELECT ?ingredient
2 WHERE {
3     ?ingredient rdf:type kebi:Ingredient .
4     ?ingredient kebi:dairy_containsLactoseIntolerance kebi:allergy_lactose .
5 }
```

Listing 3.23: Lactose intolerance: ingredients

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

```

1 SELECT ?meal
2 WHERE {
3     ?meal rdf:type kebi:Meal .
4     ?meal kebi:meal_containsGlutenIntolerance kebi:allergy_gluten .
5 }
```

Listing 3.24: Gluten intolerance: meals - REASONER Protégé

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

```

1 SELECT ?meal
2 WHERE {
3   ?meal rdf:type kebi:Meal .
4   ?meal kebi:meal_containsLactoseIntolerance kebi:allergy_lactose .
5 }
```

Listing 3.25: Lactose intolerance: meals - REASONER Protégé

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

```

1 SELECT ?meal
2 WHERE {
3   ?meal rdf:type kebi:Meal .
4   ?meal kebi:meal_hasCourse kebi:course_appetizer .
5 }
```

Listing 3.26: Meals classified as 'appetizer'

- **Query to view the meals classified as 'first-dish'**

```

1 SELECT ?meal
2 WHERE {
3   ?meal rdf:type kebi:Meal .
4   ?meal kebi:meal_hasCourse kebi:course_first-dish .
5 }
```

Listing 3.27: Meals classified as 'first-dish'

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

```

1 SELECT ?ingredient
2 WHERE {
3   ?ingredient rdf:type kebi:Seafood.
4 }
```

Listing 3.28: Ingredient that are classified as 'Seafood'

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

```

1 SELECT ?meal ?kcal
2 WHERE {
3   ?meal kebi:food_hasKcal ?kcal.
4   FILTER(xsd:decimal(?kcal) > 500).
5 }
```

Listing 3.29: Meals that have more than 500 kcal

3.3.4 SHACL Shape

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

The SHACL Shape file, located in the ontology folder and named shacl_menu⁷, includes the following:

- **RestaurantShape:** Requires each 'Restaurant' instance to contain at least one 'Menu' (MenuShape). No maximum count is enforced, giving restaurant managers the flexibility to add more menus in the future. Allows optional links to 'Guest' (GuestShape), who are associated with the restaurant after scanning a QR code.

```

1 kebi2025:RestaurantShape a sh:NodeShape ;
2   sh:targetClass kebi2025:Restaurant ;
3   sh:property [
4     sh:path kebi2025:restaurant_hasMenu ;
5     sh:node kebi2025:MenuShape ;
6     sh:minCount 1 ;
7     sh:message "It_is_impossible_that_a_Restaurant_does_not_have_a_menu!!" ;
8   ];
9   sh:property [
10    sh:path kebi2025:restaurant_hasGuest ;
11    sh:node kebi2025:GuestShape ;
12    sh:minCount 0 ;
13 ].
```

Listing 3.30: 'Restaurant' SHACL Shape

Example:

Deleting the menu produces the error:

'It_is_impossible_that_a_Restaurant_does_not_have_a_menu'.

- **MenuShape:** Ensures each 'Menu' contains at least one 'Meal' and one 'Drink'. Missing either triggers an error.

```

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

Listing 3.31: 'Menu' SHACL Shape

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

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

```

1 kebi2025:GuestShape a sh:NodeShape ;
2   sh:targetClass kebi2025:Guest ;
3   sh:property [
4     sh:path kebi2025:guest_hasAllergy ;
5     sh:node kebi2025:AllergyShape ;
6     sh:minCount 0 ;
7   ];
8   sh:property [
9     sh:path kebi2025:guest_hasCategory ;
10    sh:node kebi2025:CategoryShape ;
11    sh:minCount 1 ;
12  ];
13  sh:property [
14    sh:path kebi2025:guest_isAtRiskForFood ;
15    sh:node kebi2025:MealShape ;
16  ];
17  sh:property [
18    sh:path kebi2025:guest_hasLevelOfCalorieConscious ;
19    sh:datatype xsd:integer ;
20    sh:minCount 1 ;
21    sh:maxCount 4 ;
22 ].
```

Listing 3.32: 'Guest' SHACL Shape

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

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

```

1 kebi2025:MealShape a sh:NodeShape ;
2   sh:targetClass kebi2025:Meal ;
3   sh:property [
4     sh:path kebi2025:meal_containsLactoseIntolerance ;
5     sh:node kebi2025:LactoseShape ;
6     sh:minCount 0 ;
7     sh:maxCount 1 ;
8   ];
9   sh:property [
10    sh:path kebi2025:meal_containsGlutenIntolerance ;
11    sh:node kebi2025:GlutenShape ;
12    sh:minCount 0 ;
13    sh:maxCount 1 ;
14  ];
15  sh:property [
16    sh:path kebi2025:meal_hasCategory ;
17    sh:node kebi2025:CategoryShape ;
18    sh:minCount 0 ;
19  ].
```

```

20     sh:property [
21         sh:path kebi2025:meal_hasCourse ;
22         sh:node kebi2025:CourseShape ;
23         sh:minCount 1 ;
24         sh:maxCount 1 ;
25     ];
26     sh:property [
27         sh:path kebi2025:meal_hasIngredient ;
28         sh:node kebi2025:IngredientShape ;
29         sh:minCount 1 ;
30     ].

```

Listing 3.33: 'Meal' SHACL Shape

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

Validation

Severity	SourceShape	Message	FocusNode	Path	Value

Figure 3.32: SHACL validator

As shown in the final validation image, no violations were detected.

4. Agile and Ontology-based Meta-Modelling

In Section 4.1, we will describe what AOAME is, while in Section 4.2, we will outline what BPMN 2.0 entails and present our model. Here you can find the link to our project for Task 2¹.

4.1 AOAME

AOAME² is a prototype tool that applies an agile, ontology-based approach to meta-modeling. Its goal is to design and maintain schemas for Enterprise Knowledge Graphs (EKGs), which structure knowledge within a specific domain to support the analysis, reasoning, and integration of heterogeneous data sources. A key challenge in this process lies in the need for expertise in both ontology engineering and the application domain.

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

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

4.2 BPMN 2.0

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

¹Github Forked Project link: URL: <https://github.com/Elmicass/Ontology4ModelingEnvironment>

²AOAME Reference: URL: <https://emisa-journal.org/emisa/article/view/310>

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

The main elements of BPMN 2.0 include the following:

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

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

4.2.1 Our BPMN 2.0 Model

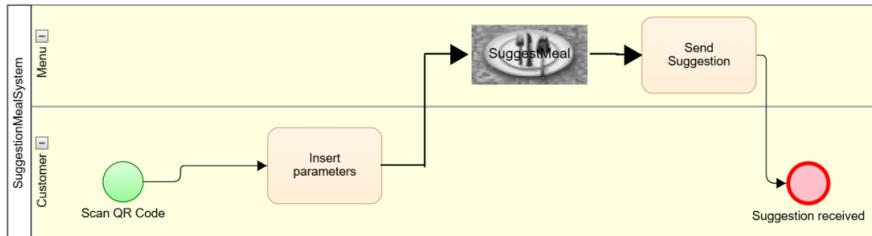


Figure 4.1: BPMN 2.0 graph built in AOAME

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

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

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

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

```

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

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

```

Figure 4.2: Query 1: Used to view the properties and their associated values for the extended class 'SuggestMeal'.

```

8  SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
9  WHERE {
10    mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
11    mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC
12    mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
13    mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCoursePreference ?course .
14
15    BIND(IF(?category = "Vegetarian", kebi:category_vegetarian,
16          IF(?category = "Carnivorous", kebi:category_carnivorous,
17              IF(?category = "Omnivore", kebi:category_omnivore, ?category))) AS ?finalCategory) .
18
19    BIND(IF(?course = "Appetizer", kebi:course_appetizer,
20          IF(?course = "First Dish", kebi:course_first-dish,
21              IF(?course = "Main Dish", kebi:course_main-dish,
22                  IF(?course = "Second Course", kebi:course_second-course,
23                      IF(?course = "Sidedish", kebi:course_sidedish,
24                          IF(?course = "Dessert", kebi:course_dessert, ?course)))))) AS ?finalCourse) .
25
26    BIND(IF(?allergy = "Gluten", kebi:grain_containsGlutenIntolerance,
27          IF(?allergy = "Lactose", kebi:dairy_containsLactoseIntolerance,
28              IF(?allergy = "none", "none", ?allergy))) AS ?finalAllergy) .
29
30    ?meal a kebi:Meal .
31    ?meal kebi:meal_hasLevelOfCalorieConscious ?kcal .
32    FILTER(?kcal = ?levelCC)
33
34    ?meal kebi:food_hasCategory ?finalCategory .
35    ?meal kebi:meal_hasCourse ?finalCourse .
36
37    OPTIONAL {
38      ?meal kebi:meal_hasIngredient ?ingredient .
39      ?ingredient rdf:type ?ingredientType .
40      FILTER ((?finalAllergy = "none") ||
41              (?finalAllergy = kebi:grain_containsGlutenIntolerance && ?ingredientType = kebi:Grain) ||
42              (?finalAllergy = kebi:dairy_containsLactoseIntolerance && ?ingredientType = kebi:Dairy))
43    }
44    FILTER (!BOUND(?ingredient))
45  }

```

Figure 4.3: Query 2: Used to analyze the extended class and return the meals that satisfy all the described data properties.

The first query retrieves the properties and their corresponding values of the 'SuggestMeal' class, while the second query analyzes the graph and returns the meal suggestions that meet the customer's parameters.

Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Lactose	<button>Remove</button>
guest_hasCategory	Vegetarian	<button>Remove</button>
guest_hasCoursePrefer	Appetizer	<button>Remove</button>
guest_hasCalorieConsc	1	<button>Remove</button>
Add Relation		
Save Close		

Figure 4.4: Defining the attributes of the extended class. In this example, the lactose allergy is set together with the vegetarian category.

SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries
 Selection of triples Selection of classes

SPARQL Endpoint /ModEnv/ Content Type (SELECT) JSON Content Type (GRAPH) Turtle

```

1: PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2: PREFIX mod: <http://fhmw.ch/modellingEnvironment/ModelOntology#>
3: PREFIX lo: <http://fhmw.ch/modellingEnvironment/LanguageOntology#>
4: PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
5: PREFIX kebl: <http://www.semanticweb.org/simone.micarelli1-davide.nappo/keb12825#>
6: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
7:
8: SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
9: WHERE {
10:   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac loguest_hasCategory ?category .
11:   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac loguest_hasCalorieConsciousLevel ?levelCC .
12:   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac loguest_hasAllergy ?allergy .
13:   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac loguest_hasCoursePreference ?course .
14: }
```

Press CTRL + Spacebar to autocomplete

Table Response 1 result in 0.055 seconds

meal	category	levelCC	allergy	course
< http://www.semanticweb.org/francesco.chiochi-nicolo.rossini/keb12024#meal_involtini-primavera >	Vegetarian	*+*+* < http://www.w3.org/2001/XMLSchema#integer >	Lactose	Appetizer

Showing 1 to 1 of 1 entries

Figure 4.5: Result of Query 2 using the parameters defined in figure 4.4

In the first example, as shown in the figure above, we define the attributes of our BPMN class and execute the query in Jena Fuseki, which returns the corresponding result.

Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Gluten	<button>Remove</button>
guest_hasCategory	Vegetarian	<button>Remove</button>
guest_hasCoursePrefer	Appetizer	<button>Remove</button>
guest_hasCalorieConsc	1	<button>Remove</button>

▼ Add Relation

[Save](#) [Close](#)

Figure 4.6: Defining the attributes of the extended class. In this example, the gluten allergy is set together with the vegetarian category.

SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries [Selection of triples](#) [Selection of classes](#)

SPARQL Endpoint [/ModEnv/](#) Content Type (SELECT) [JSON](#) Content Type (GRAPH) [Turtle](#)

Prefixes [rdf](#) [rdfs](#) [owl](#) [xsd](#)

```

1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX mod: <http://fhmw.ch/modellingenvironment/modelontology#>
3 PREFIX lio: <http://fhmw.ch/modellingenvironment/languageontology#>
4 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
5 PREFIX kebl: <http://www.semanticweb.org/silene/micarelli-davide.nappo/keb12025#>
6 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
7
8 SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
9 WHERE {
10   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac.loiquest_hasCategory ?category .
11   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac.loiquest_hasCalorieConsciousLevel ?levelCC .
12   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac.loiquest_hasAllergy ?allergy .
13   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac.loiquest_hasCoursePreference ?course .
14 }
```

Press CTRL + <spacebar> to autocomplete

Table Response 1 result in 0.05 seconds

Simple view □ Ellipse □ Filter query results Page size: 50 ↕ ↖ ↘ ↗

meal	category	levelCC	allergy	course
1 <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/keb12024#meal_formaggi-misti>	Vegetarian	1+*+* <http://www.w3.org/2001/XMLSchema#integer>	Gluten	Appetizer

Showing 1 to 1 of 1 entries

Figure 4.7: Result of Query 2 using the parameters defined in figure 4.6

Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Lactose	<button>Remove</button>
guest_hasCategory	Carnivorous	<button>Remove</button>
guest_hasCoursePrefer	Second Course	<button>Remove</button>
guest_hasCalorieConsc	2	<button>Remove</button>

▼ Add Relation

[Save](#) [Close](#)

Figure 4.8: Defining the attributes of the extended class. In this example, the lactose allergy is set together with the carnivorous category.

SPARQL Query
To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries [Selection of triples](#) [Selection of classes](#)

SPARQL Endpoint [/ModEnv/](#) Content Type (SELECT) [JSON](#) Content Type (GRAPH) [Turtle](#)

```

PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhwa.ch/modellingEnvironment/ModelOntology#>
PREFIX lo: <http://fhwa.ch/modellingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/simone.micarelli1-davide.nappo/kebi12825#>
PREFIX rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  ?meal mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac .
  ?meal mod:hasCategory ?category .
  ?meal mod:hasCalorieConsciousLevel ?levelCC .
  ?meal mod:hasAllergy ?allergy .
  ?meal mod:hasCoursePreference ?course .
}
  
```

Press CTRL + Spacebar to autocomplete

Table Response 1 result in 0.055 seconds

meal	category	levelCC	allergy	course
< http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#meal_pesce-arrosto >	Carnivorous	"2***< http://www.w3.org/2001/XMLSchema#integer >"	Lactose	Second Course

Showing 1 to 1 of 1 entries

Figure 4.9: Result of Query 2 using the parameters defined in figure 4.8

Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Lactose	<button>Remove</button>
guest_hasCategory	Carnivorous	<button>Remove</button>
guest_hasCoursePrefer	Main Dish	<button>Remove</button>
guest_hasCalorieConsc	1	<button>Remove</button>

▼ Add Relation

[Save](#) [Close](#)

Figure 4.10: Defining the attributes of the extended class. In this example, the lactose allergy is set together with the carnivorous category and 'Main Dish' course type.

SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries [Selection of triples](#) [Selection of classes](#)

SPARQL Endpoint /ModEnv/ Content Type (SELECT) [JSON](#) [Turtle](#)

Prefixes [rdf](#) [rdfs](#) [owl](#) [xsd](#)

```

1 <PREFIX rdfs: ><http://www.w3.org/2000/01/rdf-schema>
2 <PREFIX mod: ><http://fhmw.ch/modellingenvironment/ModelOntology>
3 <PREFIX lo: ><http://fhmw.ch/modellingenvironment/languageOntology>
4 <PREFIX xsd: ><http://www.w3.org/2001/XMLSchema>
5 <PREFIX kebl: ><http://www.semanticweb.org/thesone/micarelli1/davide.nappo@kebl12025a>
6 <PREFIX rdf: ><http://www.w3.org/1999/02/22-rdf-syntax-ns>
7
8 SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
9 WHERE {
10   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
11   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC .
12   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
13   mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCoursePreference ?course .
14 }
```

Press CTRL + <spacebar> to autocomplete

Table Response 1 result in 0.055 seconds

meal	category	levelCC	allergy	course
1 <http://www.semanticweb.org/francesco.chiochic-nicolo.rossini/kebl2024#meal_pizza-bianca>	Carnivorous	"1"^^<http://www.w3.org/2001/XMLSchema#integer>	Lactose	Main Dish

Showing 1 to 1 of 1 entries

Figure 4.11: Result of Query 2 using the parameters defined in figure 4.10

5. Conclusions

5.1 Davide Nappo

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

- **Decision Tables:** Decision tables operate in a straightforward manner: inputs are matched with outputs according to predefined rules structured in rows. Output selection follows a Hit Policy, which dictates how results are grouped. In our project, we applied the Collect policy, allowing all rules to be evaluated concurrently and fully leveraging Decision Model and Notation (DMN). Under this logic, input conditions are connected by an “and” relationship within a column, while rows interact with each other through an “or” relationship. Outputs are aggregated into sets for each output column. This approach is intuitive and easy to implement. However, in contexts involving numerous variables or complex conditions, decision tables quickly become impractical and overly complicated. Data manipulation proved cumbersome, repetitive, and time-consuming, with the manual creation of entries turning monotonous. Debugging was also challenging, made worse by limited online tool support.
- **Prolog:** Working with Prolog was one of the most engaging parts of the project. It gave me the opportunity to experiment with a new programming language. Although not overly difficult to grasp, Prolog suffers from scarce and fragmented learning resources. Its programming paradigm emphasizes recursion, lacking conventional loop structures and straightforward arithmetic operations. Despite these limitations, Prolog excels in knowledge representation tasks, where retrieving and managing data from knowledge bases is intuitive. Nonetheless, its performance decreases significantly with large datasets, leading to long computation times. While I appreciated the learning experience, the lack of clarity in certain areas and its scalability issues were notable drawbacks.
- **Protégé:** I perceived Protégé as a hybrid approach between DMN and object-oriented programming. Within this environment, we could define objects, their attributes, and their relationships, while also extracting knowledge from these instances. What I particularly appreciated was the similarity of its query system to SQL, which made the tool easier to use. The SHACL validator was especially valuable, as it allowed us to verify whether a property was correctly defined—much like performing `assertTrue` or `assertFalse` in Java. Furthermore, SWRL rules enabled type inference, though I believe they should be applied sparingly. This view stems from my experience with tools like AOAME and GraphDB, which lack

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

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

Conclusion

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

5.2 Simone Micarelli

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

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

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

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

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

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

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

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

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

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

Bibliography

- [Aoa] *AOAME Reference*: URL: <https://emisa-journal.org/emisa/article/view/310>.
- [Cama] *Camunda*. URL: <https://camunda.com/>.
- [Camb] *Camunda Documentation*. URL: <https://docs.camunda.org/manual/7.19/reference/dmn/feel/>.
- [Git] *Github Forked Project link*: URL: <https://github.com/Elmicass/Ontology4ModelingEnvironment>.
- [Proa] *Prolog Testing*. URL: <https://swish.swi-prolog.org/>.
- [Prob] *Prolog Wikipedia*. URL: <https://it.wikipedia.org/wiki/Prolog>.
- [Sha] *SHACL Shape File*: URL: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project/blob/main/Task1/Ontology/shacl_menu.txt.
- [Spa] *SPARQL example queries*: URL: https://github.com/Elmicass/PersonalizedMenu-KEBI_Project/blob/main/Task1/Ontology/sparql_queries.txt.
- [Wik] *Wikipedia. Decision Table*. URL: https://en.wikipedia.org/wiki/Decision_table.