University of Camerino

School of Science and Technologies

Master Degree In Computer Science (LM-18) - Knowledge Engineering

Intelligent platform for restaurant meal proposition, using knowledge-based solutions and meta-modeling structured according to ontological principles.

Students

Alessandro Vitali
Lorenzo Gezzi

Email

alessandr.vitali@studenti.unicam.it
lorenzo.gezzi@studenti.unicam.it

Supervisor

Prof. Knut Hinkelmann
Prof. Emanuele Laurenzi

A.Y. 2024/2025

Abstract

The digitization of restaurant menus through QR codes has improved accessibility and reduced the use of printed materials, but it also presents certain limitations, particularly regarding usability on mobile devices with small screens. Furthermore, the uniform presentation of the menu does not take into account the specific dietary preferences or needs of guests, such as vegetarians, calorie-conscious individuals, or those with intolerances or allergies. This project proposes the development of an intelligent system capable of filtering the dishes in a digital menu based on users' personal preferences. By utilizing a structured knowledge base containing information about typical Italian dishes and their ingredients, the menu display can be dynamically adapted. The system will enhance the user experience by providing a targeted and personalized selection of available options.

Contents

1. Introduction

In recent years, the widespread adoption of digital technologies has significantly transformed the restaurant industry, reshaping how customers interact with services and consume information. One of the most prominent and visible innovations has been the introduction of digital menus accessible via QR codes. By simply scanning a QR code at their table, customers can access the restaurant's menu on their smartphones. This shift has proven particularly useful in post-pandemic contexts, where contactless interaction and improved hygiene have become essential. Additionally, digital menus help reduce the use of printed materials, aligning with sustainability goals and operational efficiency.

However, while digital menus offer various benefits, they are not without limitations. One major drawback concerns their usability, particularly when viewed on small smartphone screens. Long or complex menus, such as those typically found in Italian restaurants with diverse offerings of appetizers, pasta, pizza, and main courses, can be difficult to navigate on mobile devices. Guests may struggle to gain a complete overview of their options, resulting in a frustrating and inefficient browsing experience.

Another significant limitation of current digital menus is the lack of personalization. Most systems display the full list of available dishes without taking into account the individual dietary needs or preferences of the customer. In a society where food choices are increasingly guided by health considerations, ethical beliefs, or medical requirements, this generic approach fails to meet the expectations of many users. Customers may follow specific diets—such as vegetarian, vegan, or low-calorie—or they may suffer from food intolerances or allergies, such as lactose or gluten intolerance. Navigating through all menu items to find those that are suitable can be time-consuming and discouraging, particularly for users unfamiliar with the dish names or their ingredients. As a result, the overall dining experience may be negatively impacted.

With this integration, Cosmari can better align its practices with compliance requirements, while also improving the accuracy and reliability of reporting. This upgraded system not only facilitates regulatory adherence but also drives the company towards a higher standard of efficiency, sustainability, and accountability in waste recycling. As a result, Cosmari will be empowered to uphold its environmental commitments while setting a benchmark for excellence and responsibility within the waste management industry.

1.1 Motivation and Objectives

The motivation behind this project stems from the desire to enhance the functionality and user-friendliness of digital menus by making them more adaptive and context-aware. The core objective is to design and implement an intelligent system capable of filtering and displaying only those dishes that align with a user's dietary profile and preferences. This approach aims to turn the menu from a static list into a dynamic, personalized tool that actively assists the guest in making informed and satisfying choices.

To achieve this, the system will rely on a structured knowledge base that represents the key components of a typical Italian restaurant menu. Dishes such as pasta, pizza, and main courses will be described along with their constituent ingredients. Each ingredient will be categorized based on relevant types—such as meat, vegetables, fruits, dairy, and others—and associated with nutritional information, including calorie content. This semantic structure will allow the system to reason about the contents of each meal and evaluate its suitability for different types of users.

In parallel, the system will model user profiles, which can include general dietary preferences (e.g., vegetarian, carnivore, low-calorie), medical constraints (e.g., lactose or gluten intolerance), and possibly other contextual information, such as time of day or personal goals. These profiles will serve as filters that guide the menu selection process, ensuring that only compatible dishes are shown to the guest.

The intended outcome is an improved user experience that combines clarity, efficiency, and personalization. By reducing cognitive load and eliminating irrelevant options, the system will help users quickly identify meals that match their needs and preferences. Moreover, it promotes inclusivity by accommodating a wide range of dietary requirements, contributing to greater customer satisfaction and loyalty.

In summary, this project seeks to bridge the gap between modern digital menu systems and the increasingly individualized expectations of restaurant customers. By leveraging knowledge representation and intelligent filtering, the proposed system aspires to deliver a smarter and more human-centered approach to digital dining.

1.2 Workflow

1. **Analysis of the provided specifications and project requirements for the system's development.**
2. **Create different knowledge-based solutions for recommending food depending on the profile of a guest**
    - DRD diagram with decision tables
    - Prolog: The system encodes domain knowledge using facts and rules, leveraging logic programming to infer suitable meals based on user profiles and ingredient data.
    - Knowledge Graph / Ontology:
        - Knowledge graph
        - SHACL shapes
        - queries in SPARQL
        - rules in SWRL
3. **Agile and ontology-based meta-modelling**
    - BMPN Diagram
    - SPARQL query in Jena Fuseki

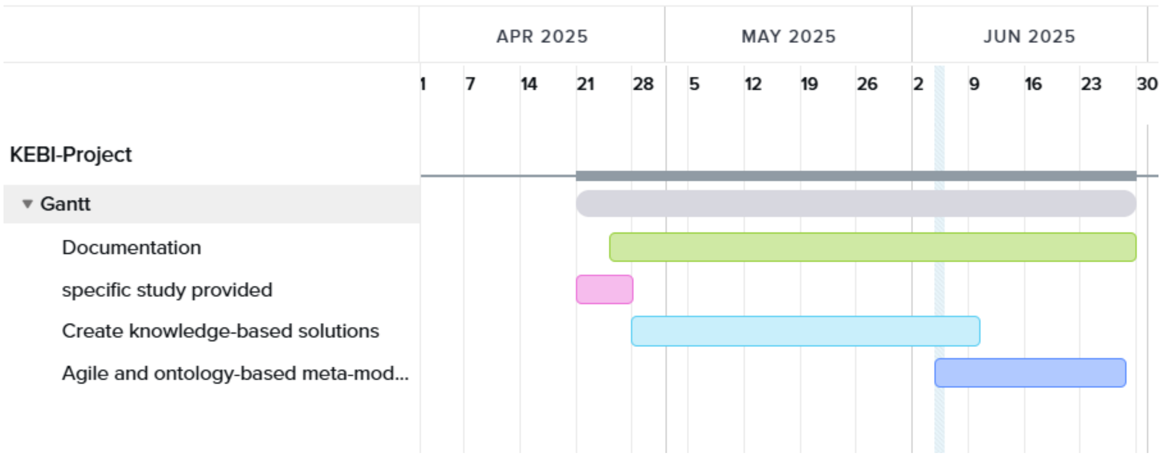


Figure 1.1: Gantt chart

2. Analysis of Requirements and Specifications

In this section, we provide an overview and analysis of the project specifications as outlined in the initial description. The project focuses on improving the digital dining experience in restaurants, particularly addressing the limitations of current menu presentation systems via QR codes on smartphones.

The provided text describes a common scenario where many restaurants have digitized their menus, allowing guests to access them through QR codes. While this offers convenience, it introduces usability challenges, especially when the menu is extensive and the device screen is small. Moreover, not all guests are interested in viewing the entire list of available meals. For example, vegetarians or individuals with specific dietary restrictions (such as allergies to lactose or gluten) might prefer to see only those meals that align with their preferences or health needs.

The core objective of the project is to represent and reason about knowledge related to meals and guest preferences. This involves designing a system capable of filtering and suggesting suitable meals based on the individual characteristics and restrictions of each guest. The underlying knowledge base is expected to include structured information about typical Italian restaurant dishes — such as pizza, pasta, and various main courses — along with the ingredients used in each meal. Additionally, ingredients are categorized by type (meat, vegetables, fruits, dairy) and are annotated with nutritional information, such as calorie content.

Guest profiles are also considered in the requirements. These include categories like carnivores, vegetarians, calorie-conscious individuals, and those with allergies or intolerances. The challenge lies in modeling this information semantically, using ontology-based techniques, and integrating it into a system that can dynamically adapt the menu presentation to show only relevant meal options for each guest.

This analysis lays the foundation for the subsequent phases of development, including knowledge modeling, system design, and user interface considerations.

3. Decision Model and Notation

In this section we will explain why the dmn diagram is used and how it was created to recommend dishes that the user can eat based on the preferences set.

3.1 Introduction to DMN

In modern organizations, decision-making often involves complex logic, regulatory requirements, and data-driven processes. The Decision Model and Notation (DMN), introduced by the Object Management Group (OMG), addresses this need by providing a standard to model and manage business decisions. [Dmn]

Two core elements of DMN are the **Decision Requirements Diagram (DRD)** and the **Decision Table**. Together, they separate the structure of a decision (the “what”) from the logic behind it (the “how”), enabling a transparent and maintainable decision model.

3.2 Introduction to DRD and Decision Table

This section describes the main specifications of the DRD and decision tables.

3.2.1 What Are Decision Requirements Diagrams?

A Decision Requirements Diagram (DRD) graphically represents the structure of decisions. It defines how individual decisions relate to input data, business knowledge, and other decisions. Each decision can be broken down into sub-decisions, forming a network of dependencies. DRD make it easier for stakeholders—both business and technical—to understand, analyze, and improve decision processes collaboratively.

DRD include several components:

- **Decisions** - defined by a question and possible answers.
- **Input Data** - external information required to make a decision.
- **Knowledge Sources** - the origin of business know-how (e.g., policies, regulations).
- **Business Knowledge Models** - reusable logic modules (such as decision tables).
- **Information Requirement** - shows that Input Data or decision output is required as an input by another Decision
- **Knowledge Requirement** - the invocation of a business knowledge model.

- **Authority Requirement** - shows the knowledge source of an element or the dependency of a knowledge source on input data.

3.2.2 What Are Decision Tables?

Decision Tables represent the logic of a decision in a compact, structured, and human-readable form. They specify which outputs (actions) result from given inputs (conditions), and are often assigned to decision elements within a DRD.

A Decision Table in DMN typically includes:

- **Condition Stubs** - input criteria affecting the decision.
- **Action Stubs** - potential outcomes.
- **Condition Entries** - values or expressions for each input.
- **Action Entries** - results for each combination of conditions.

DMN also defines **hit policies** to handle how multiple matching rules are treated (e.g., first match, collect all, prioritize).

Decision Tables are suitable for capturing business logic that is repeatable, testable, and can be directly automated—making them ideal for our project's goal of personalized meal recommendations based on guest preferences.

3.3 Motivations for Using DRD and Decision Tables

The objective of this project is to provide personalized meal recommendations that align with each guest's dietary preferences and restrictions. To address this challenge effectively, we adopted Decision Requirements Diagrams (DRD) and Decision Tables as core modeling tools. These methods offer a structured, transparent, and adaptable framework for representing decision logic.

Guests may follow a wide range of dietary patterns—such as vegetarian or vegan diets, lactose or gluten intolerance, or calorie-conscious plans—while the menu itself can consist of numerous dishes with varying ingredients and nutritional profiles. DRD help break down this complexity by mapping out the required decision components, such as allergen filtering, ingredient exclusions, and nutritional evaluations. Each component can then be addressed using Decision Tables, which formalize the conditions and outcomes that drive meal inclusion or exclusion based on the guest's profile.

In addition to supporting modular and maintainable design, DRD and Decision Tables improve cross-functional collaboration. Their visual and tabular formats make the logic comprehensible not only to developers but also to non-technical stakeholders such as restaurant staff and business analysts. Because the decision models are executable, the business logic defined during the design phase is the same logic enforced at runtime. This minimizes implementation errors, ensures consistency, and facilitates easy updates when dietary guidelines or menu offerings change.

3.4 Advantages and Disadvantages of DRD and Decision Tables

In our project, DRD and Decision Tables proved useful for organizing and automating the logic behind personalized meal recommendations. They allowed us to clearly define rules based on guest preferences—such as dietary type, allergies, or calorie limits—and made the system easier to understand, implement, and maintain. Their modular and visual nature also supported collaboration among team members with different backgrounds.

However, these tools also had some limitations. As the number of conditions grew, the tables became harder to manage, and adapting them to more dynamic or evolving user behavior was challenging. Building a complete and correct model took time, and required external tools like Trisotech, which may not always be available. Despite this, the advantages—especially in terms of structure, reusability, and transparency—made DRD and Decision Tables a solid choice for our use case.

3.5 Our Decision Model

We used decision tables in our project to recommend one or more dishes based on the preferences declared by the user. This approach allowed us to systematically organize various user profiles and match them with suitable meals in a clear and structured format. By doing so, the system could efficiently evaluate multiple input combinations—such as dietary choices, intolerances, and calorie limits—and generate personalized recommendations.

To model and simulate the decision tables, we used the cloud-based modeling platform **Trisotech Digital Enterprise Suite**, which fully supports the DMN standard. This tool provided a user-friendly environment to design the DRD and define decision logic, making it easier to test different scenarios and validate the results throughout the development process. [Tri]

3.5.1 Full Diagram

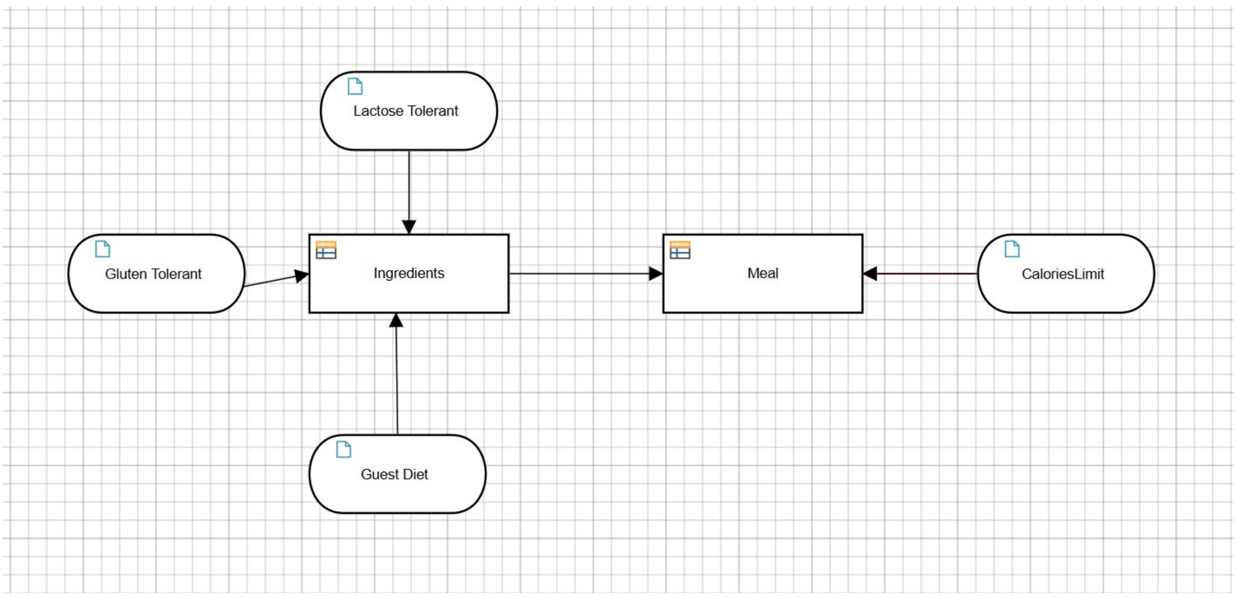
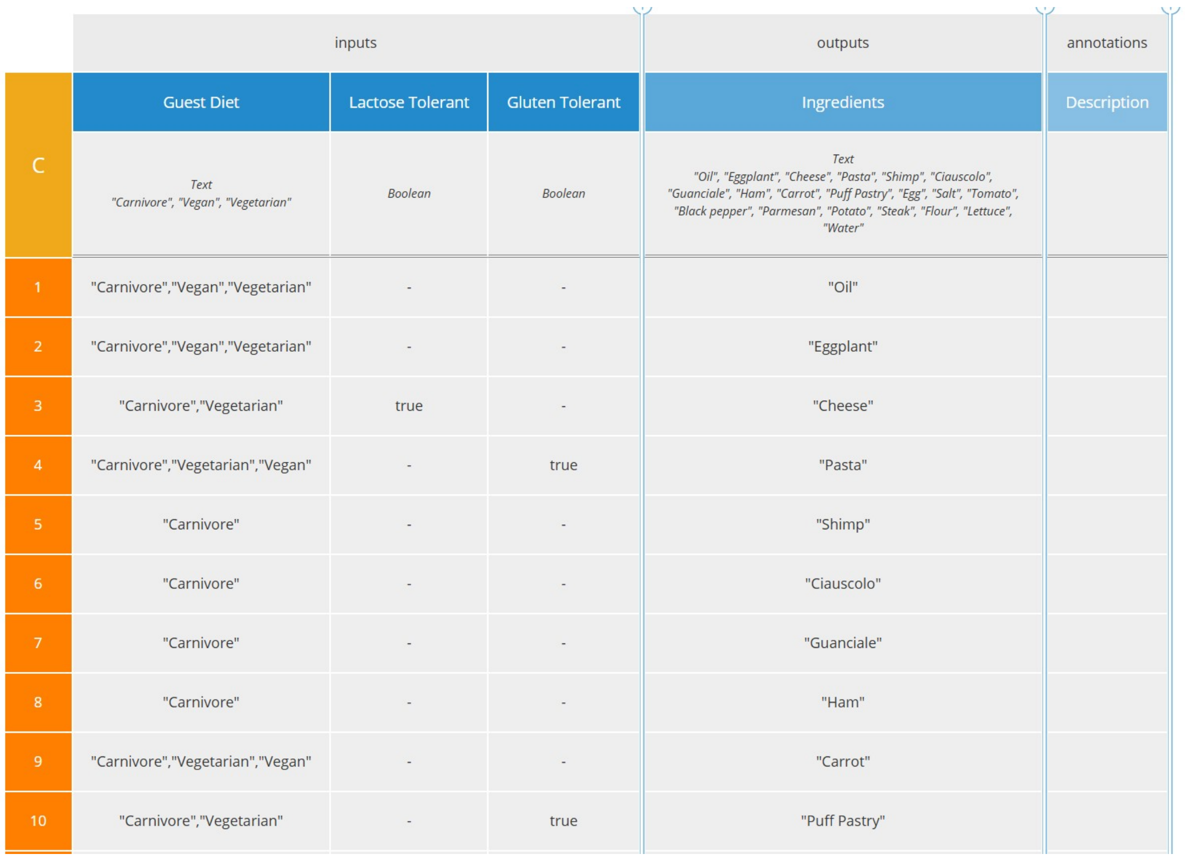


Figure 3.1: Full Diagram

This is the example of the complete diagram, which illustrates two Decision Tables and four Input Data elements. The first Decision Table, *Ingredients*, is used to filter or associate ingredients based on the type of diet and guest intolerances. As input, it takes *Guest Diet*, *Lactose Intolerance*, and *Gluten Intolerance*, and produces as output the corresponding ingredients that comply with these constraints. The second Decision Table is responsible for composing meals based on the filtered ingredients and a calorie limit. It takes as input the *Selected Ingredients* and the *Calorie Threshold*, and outputs the appropriate meals that match both nutritional and dietary requirements.

3.5.2 Ingredients Decision Table

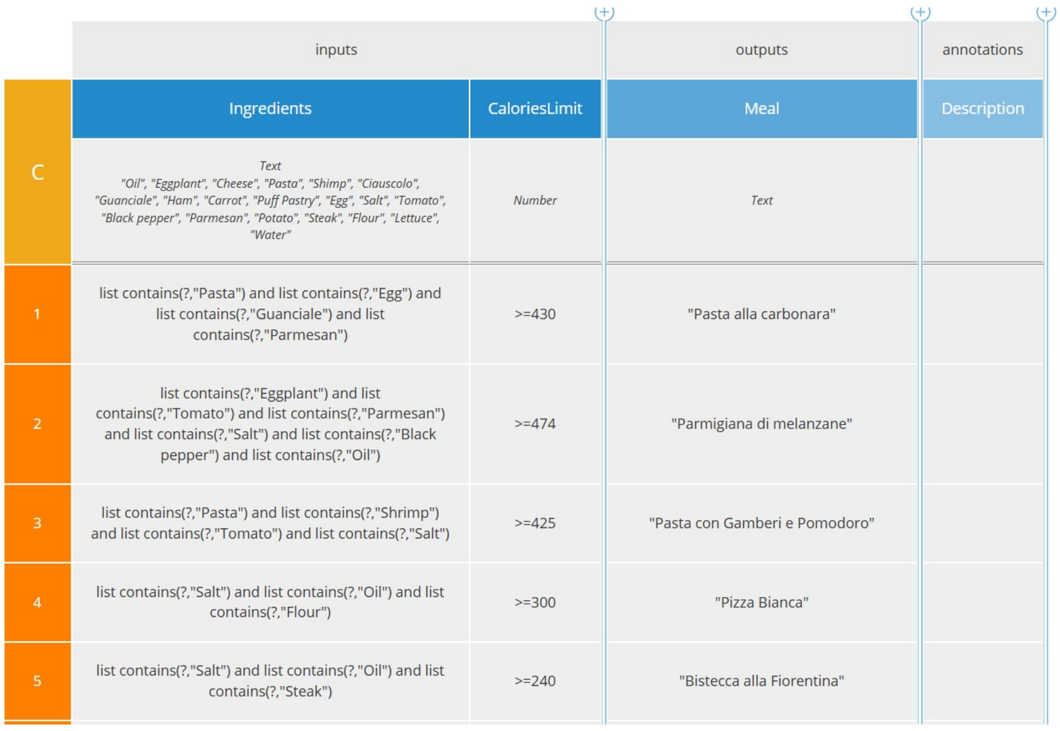


	inputs			outputs	annotations
C	Guest Diet	Lactose Tolerant	Gluten Tolerant	Ingredients	Description
	Text *"Carnivore", "Vegan", "Vegetarian"*	*Boolean*	*Boolean*	*Text* *"Oil", "Eggplant", "Cheese", "Pasta", "Shimp", "Ciauscolo", "Guanciale", "Ham", "Carrot", "Puff Pastry", "Egg", "Salt", "Tomato", "Black pepper", "Parmesan", "Potato", "Steak", "Flour", "Lettuce", "Water"*	
1	"Carnivore","Vegan","Vegetarian"	-	-	"Oil"	
2	"Carnivore","Vegan","Vegetarian"	-	-	"Eggplant"	
3	"Carnivore","Vegetarian"	true	-	"Cheese"	
4	"Carnivore","Vegetarian","Vegan"	-	true	"Pasta"	
5	"Carnivore"	-	-	"Shimp"	
6	"Carnivore"	-	-	"Ciauscolo"	
7	"Carnivore"	-	-	"Guanciale"	
8	"Carnivore"	-	-	"Ham"	
9	"Carnivore","Vegetarian","Vegan"	-	-	"Carrot"	
10	"Carnivore","Vegetarian"	-	true	"Puff Pastry"	

Figure 3.2: Ingredients Decision Table

As shown in this image, the *Ingredients* Decision Table takes as input the user's diet and various intolerances. For the user's diet, a `Text Enum` data type is used, which includes the three predefined diet types: *Carnivore*, *Vegan*, and *Vegetarian*. For intolerances or allergies, Boolean data types are used—`true` if the user has the specific intolerance, and `false` otherwise. The output of the table is the specific ingredient, represented using a `Text Expression` type, which filters ingredients according to the provided dietary and intolerance constraints.

3.5.3 Meal Decision Table

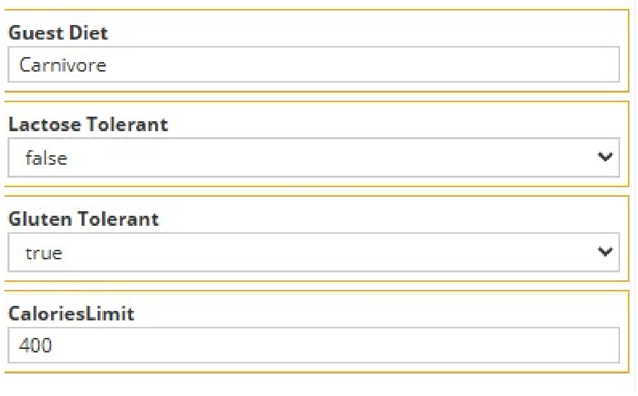


	inputs		outputs	annotations
C	Ingredients	CaloriesLimit	Meal	Description
	Text *"Oil", "Eggplant", "Cheese", "Pasta", "Shimp", "Ciauscolo", "Guanciale", "Ham", "Carrot", "Puff Pastry", "Egg", "Salt", "Tomato", "Black pepper", "Parmesan", "Potato", "Steak", "Flour", "Lettuce", "Water"*	*Number*	*Text*	
1	list contains(?,"Pasta") and list contains(?,"Egg") and list contains(?,"Guanciale") and list contains(?,"Parmesan")	>=430	"Pasta alla carbonara"	
2	list contains(?,"Eggplant") and list contains(?,"Tomato") and list contains(?,"Parmesan") and list contains(?,"Salt") and list contains(?,"Black pepper") and list contains(?,"Oil")	>=474	"Parmigiana di melanzane"	
3	list contains(?,"Pasta") and list contains(?,"Shrimp") and list contains(?,"Tomato") and list contains(?,"Salt")	>=425	"Pasta con Gamberi e Pomodoro"	
4	list contains(?,"Salt") and list contains(?,"Oil") and list contains(?,"Flour")	>=300	"Pizza Bianca"	
5	list contains(?,"Salt") and list contains(?,"Oil") and list contains(?,"Steak")	>=240	"Bistecca alla Fiorentina"	

Figure 3.3: Meal Decision Table

In this image, the *Meal* Decision Table is shown. It takes as input a list of ingredients—constructed using the `list contains` operator—and the calorie limit for the meal. Based on these inputs, the table returns as output the corresponding meal that is composed of the specified ingredients and adheres to the defined calorie threshold.

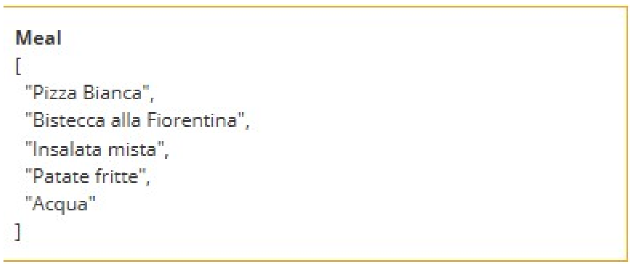
3.5.4 Results example



Guest Diet	Carnivore
Lactose Tolerant	false
Gluten Tolerant	true
CaloriesLimit	400

Figure 3.4: User Preferences

In this image, you can see set of user preferences. For this example user is carnivore, have lactose allergy and 400 of calorie limits.



```
Meal
[
  "Pizza Bianca",
  "Bistecca alla Fiorentina",
  "Insalata mista",
  "Patate fritte",
  "Acqua"
]
```

Figure 3.5: Final Menu

3.6 Conclusion

In this project, Decision Requirements Diagram (DRD) and Decision Tables helped us build a smart system for recommending meals based on each guest's dietary needs. DRD allowed us to break the decision process into smaller steps, like checking for allergens or calories. This made the logic easier to understand and manage. Decision Tables added clear rules that define exactly when a meal should be shown or hidden, depending on the guest's profile. Even though these tools can get complex with many rules, and don't handle dynamic behavior very well, their benefits are strong. They make the logic more transparent, reduce errors, and support automation. In the end, DRD and Decision Tables gave us a solid and flexible way to create a menu system that responds to each guest's preferences in a clear and consistent way.

4. Prolog Implementation of the DMN Logic

In this chapter we will illustrate what is and what is the utility of using prolog in our project.

4.1 What is Prolog

Prolog (Programmation en Logique) is a declarative programming language based on formal logic. It is widely used in artificial intelligence and computational linguistics for tasks that involve symbolic reasoning, rule-based inference, and knowledge representation. In Prolog, programs consist of a set of facts and rules, and computation is performed by querying this knowledge base. Rather than specifying how to solve a problem step by step, the programmer defines what is true in the domain of interest, and Prolog's inference engine derives answers through logical deduction. Typical applications include expert systems, natural language processing, and decision-making systems. [Pro]

4.2 Advantages and disadvantages of using prolog

Prolog is a logic programming language that is well suited for problems involving symbolic reasoning. One of its main advantages is its declarative nature, which allows programmers to focus on the problem, rather than on how to solve it. Prolog also has powerful built-in mechanisms for pattern matching and backtracking, making it efficient for finding and solving constraint-based problems. However, Prolog can be less intuitive for programmers accustomed to imperative languages, and its performance may not scale well for large, data-intensive applications. Additionally, debugging and understanding the flow of execution can be difficult due to its nonlinear control flow.

4.3 How we can use Prolog in our project

In our project, Prolog was used to model the reasoning logic required to match guest preferences with appropriate meals. Given the complexity of filtering dishes based on dietary restrictions, allergies, and nutritional considerations, Prolog allowed us to define a knowledge base of meals, ingredients, and guest profiles using logical rules. Each meal is described by its ingredients and calorie content, while guests are categorized by preferences such as vegetarian, carnivore, calorie-conscious, or allergic to certain ingredients. Prolog's inference engine was then used to query the knowledge base and return only those meals compatible with a guest's needs. This logic mirrors the decision structure defined in the DMN (Decision Model and Notation) diagram.

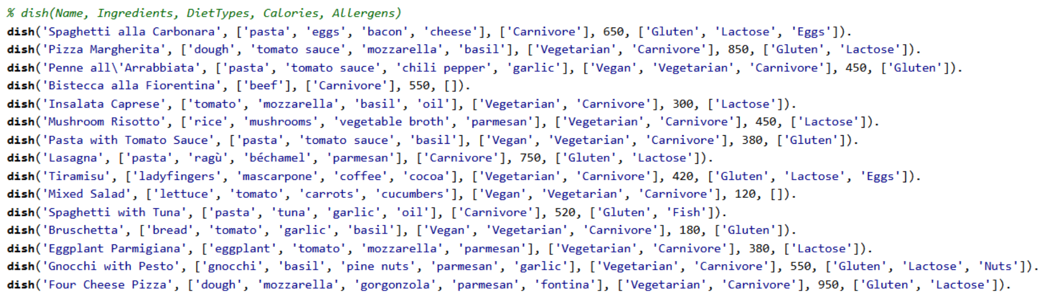
4.4 Prolog Implementation of the Recommendation System

This section presents the Prolog implementation developed to reason over the restaurant's digital menu and provide personalized meal recommendations based on guest profiles, dietary preferences, allergies, and calorie constraints.

4.4.1 Knowledge Base Representation

The knowledge base is composed of facts describing:

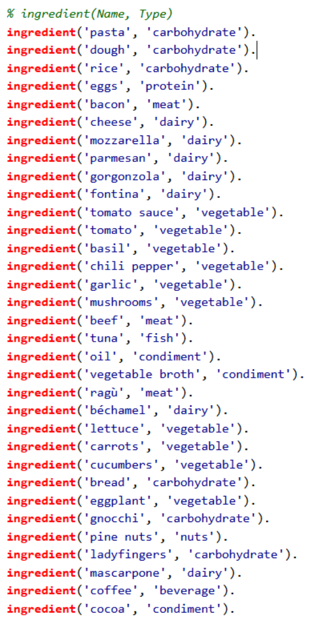
- `dish` — representing each menu item with its name, ingredients, compatible diet types, total calories, and potential allergens.



```
% dish(Name, Ingredients, DietTypes, Calories, Allergens)
dish('Spaghetti alla Carbonara', ['pasta', 'eggs', 'bacon', 'cheese'], ['Carnivore'], 650, ['Gluten', 'Lactose', 'Eggs']).
dish('Pizza Margherita', ['dough', 'tomato sauce', 'mozzarella', 'basil'], ['Vegetarian', 'Carnivore'], 850, ['Gluten', 'Lactose']).
dish('Penne all\'Arrabbiata', ['pasta', 'tomato sauce', 'chili pepper', 'garlic'], ['Vegan', 'Vegetarian', 'Carnivore'], 450, ['Gluten']).
dish('Bistecca alla Fiorentina', ['beef'], ['Carnivore'], 550, []).
dish('Insalata Caprese', ['tomato', 'mozzarella', 'basil', 'oil'], ['Vegetarian', 'Carnivore'], 300, ['Lactose']).
dish('Mushroom Risotto', ['rice', 'mushrooms', 'vegetable broth', 'parmesan'], ['Vegetarian', 'Carnivore'], 450, ['Lactose']).
dish('Pasta with Tomato Sauce', ['pasta', 'tomato sauce', 'basil'], ['Vegan', 'Vegetarian', 'Carnivore'], 380, ['Gluten']).
dish('Lasagna', ['pasta', 'ragù', 'béchamel', 'parmesan'], ['Carnivore'], 750, ['Gluten', 'Lactose']).
dish('Tiramisu', ['ladyfingers', 'mascarpone', 'coffee', 'cocoa'], ['Vegetarian', 'Carnivore'], 420, ['Gluten', 'Lactose', 'Eggs']).
dish('Mixed Salad', ['lettuce', 'tomato', 'carrots', 'cucumbers'], ['Vegan', 'Vegetarian', 'Carnivore'], 120, []).
dish('Spaghetti with Tuna', ['pasta', 'tuna', 'garlic', 'oil'], ['Carnivore'], 520, ['Gluten', 'Fish']).
dish('Bruschetta', ['bread', 'tomato', 'garlic', 'basil'], ['Vegan', 'Vegetarian', 'Carnivore'], 180, ['Gluten']).
dish('Eggplant Parmigiana', ['eggplant', 'tomato', 'mozzarella', 'parmesan'], ['Vegetarian', 'Carnivore'], 380, ['Lactose']).
dish('Gnocchi with Pesto', ['gnocchi', 'basil', 'pine nuts', 'parmesan', 'garlic'], ['Vegetarian', 'Carnivore'], 550, ['Gluten', 'Lactose', 'Nuts']).
dish('Four Cheese Pizza', ['dough', 'mozzarella', 'gorgonzola', 'parmesan', 'fontina'], ['Vegetarian', 'Carnivore'], 950, ['Gluten', 'Lactose']).
```

Figure 4.1: Prolog Dishes

- `ingredient` — defining each ingredient by its name, type (e.g., vegetable, dairy, meat)



```prolog
% ingredient(Name, Type)
ingredient('pasta', 'carbohydrate').
ingredient('dough', 'carbohydrate').
ingredient('rice', 'carbohydrate').
ingredient('eggs', 'protein').
ingredient('bacon', 'meat').
ingredient('cheese', 'dairy').
ingredient('mozzarella', 'dairy').
ingredient('parmesan', 'dairy').
ingredient('gorgonzola', 'dairy').
ingredient('fontina', 'dairy').
ingredient('tomato sauce', 'vegetable').
ingredient('tomato', 'vegetable').
ingredient('basil', 'vegetable').
ingredient('chili pepper', 'vegetable').
ingredient('garlic', 'vegetable').
ingredient('mushrooms', 'vegetable').
ingredient('beef', 'meat').
ingredient('tuna', 'fish').
ingredient('oil', 'condiment').
ingredient('vegetable broth', 'condiment').
ingredient('ragù', 'meat').
ingredient('béchamel', 'dairy').
ingredient('lettuce', 'vegetable').
ingredient('carrots', 'vegetable').
ingredient('cucumbers', 'vegetable').
ingredient('bread', 'carbohydrate').
ingredient('eggplant', 'vegetable').
ingredient('gnocchi', 'carbohydrate').
ingredient('pine nuts', 'nuts').
ingredient('ladyfingers', 'carbohydrate').
ingredient('mascarpone', 'dairy').
ingredient('coffee', 'beverage').
ingredient('cocoa', 'condiment').
```

Figure 4.2: Prolog Dishes

These facts serve as the foundation for reasoning and recommendation processes.

4.4.2 Guest Profiles and Constraints

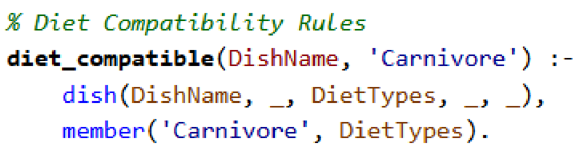
Several predefined guest profiles are defined using the predicate `guest_profile/3`, which includes:

- Dietary type (e.g., Carnivore, Vegetarian, Vegan)
- Allergy list (e.g., Gluten, Lactose, Nuts)
- Calorie limit (used for calorie-conscious guests)

4.4.3 Recommendation Logic

The system uses a set of rules to determine dish suitability:

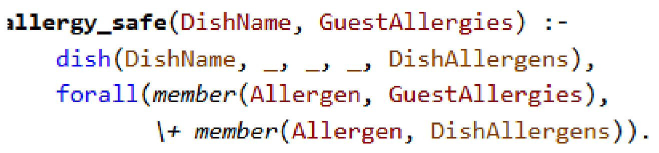
- `diet_compatible` — checks whether a dish complies with a guest's dietary restriction.



```
% Diet Compatibility Rules
diet_compatible(DishName, 'Carnivore') :-
    dish(DishName, _, DietTypes, _, _),
    member('Carnivore', DietTypes).
```

Figure 4.3: Filter by diet

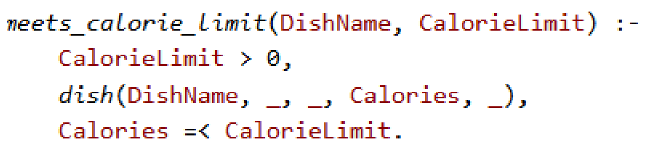
- `allergy_safe` — filters out dishes containing allergens listed by the guest.



```
llergy_safe(DishName, GuestAllergies) :-
    dish(DishName, _, _, _, DishAllergens),
    forall(member(Allergen, GuestAllergies),
           \+ member(Allergen, DishAllergens)).
```

Figure 4.4: Filter by allergy

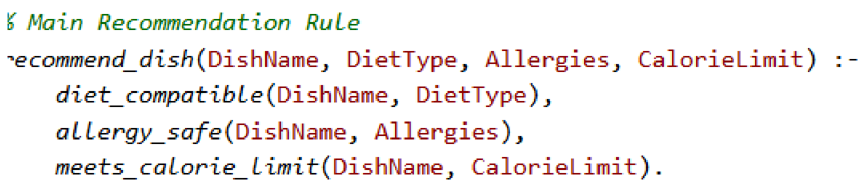
- `meets_calorie_limit` — validates that the dish's calories are within the specified limit.



```
neets_calorie_limit(DishName, CalorieLimit) :-
    CalorieLimit > 0,
    dish(DishName, _, _, Calories, _),
    Calories =< CalorieLimit.
```

Figure 4.5: Filter by calorie limit

- `recommend_dish` — the main rule that combines all constraints to determine if a dish is recommended.



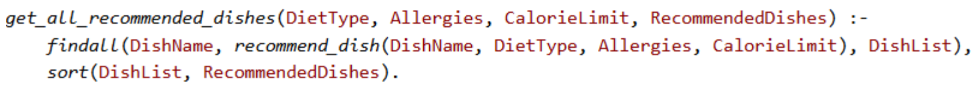
```
% Main Recommendation Rule
ecommend_dish(DishName, DietType, Allergies, CalorieLimit) :-
    diet_compatible(DishName, DietType),
    allergy_safe(DishName, Allergies),
    meets_calorie_limit(DishName, CalorieLimit).
```

Figure 4.6: Recommended dishes

4.4.4 Query and Output Rules

To enable easy use of the system:

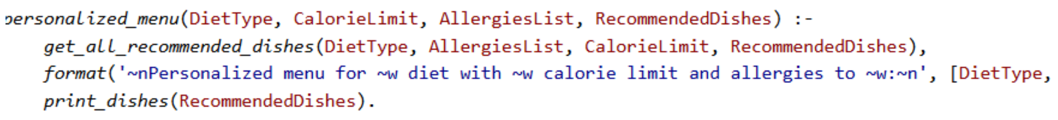
- `get_all_recommended_dishes` returns a list of all dishes that satisfy the given criteria.



```
get_all_recommended_dishes(DietType, Allergies, CalorieLimit, RecommendedDishes) :-
    findall(DishName, recommend_dish(DishName, DietType, Allergies, CalorieLimit), DishList),
    sort(DishList, RecommendedDishes).
```

Figure 4.7: All recommended dishes

- `personalized_menu` The personalized_menu function generates a personalized menu based on the type of diet, allergies, and calorie limit indicated by the user. To do this, it calls a predicate that selects the dishes compatible with these preferences and then prints a descriptive message. Finally, it displays the list of recommended dishes on the screen. In this way, the user receives targeted suggestions that are suitable for their dietary needs.



```
ersonalized_menu(DietType, CalorieLimit, AllergiesList, RecommendedDishes) :-
    get_all_recommended_dishes(DietType, AllergiesList, CalorieLimit, RecommendedDishes),
    format('~nPersonalized menu for ~w diet with ~w calorie limit and allergies to ~w:~n', [DietType,
    print_dishes(RecommendedDishes).
```

Figure 4.8: All recommended dishes

4.4.5 Conclusion

This Prolog program effectively models the logical decision-making needed to support personalized digital menus. It integrates dietary reasoning, allergy checks, and nutritional considerations, providing a flexible and extensible foundation for intelligent meal filtering in the restaurant domain.

5. Knowledge graph

In this chapter, we will describe how the knowledge graph was created respecting the logic defined in the previous chapters regarding the personalized diet for each user. The knowledge graph was built by connecting key concepts such as diet types, ingredients, dishes, allergies and caloric limits, allowing a structured representation of knowledge. Each node and relationship was defined in order to facilitate automatic querying and inference. This approach allows to generate consistent, personalized and easily updated recommendations based on user data. [Kno]

5.1 What is a Knowledge graph

A knowledge graph is a data structure that represents knowledge in a semantic and relational way, using a graph formed by nodes (representing entities such as objects, concepts, people, etc.) and arcs (representing the relationships between these entities). This type of representation allows you to connect information together in a logical way that can be easily interpreted by an intelligent system. Knowledge graphs are used to model complex contexts, such as personalized recommendations, automatic responses, or semantic analysis. In the case of a food system, for example, they can connect dishes, ingredients, diets, allergies, and nutritional preferences. The main advantage is the ability to perform advanced inferences and queries, offering more complete and relevant answers. In this way, the knowledge graph becomes a solid basis for intelligent knowledge-based applications.

5.2 How we can use Knowledge graph in our project

A knowledge graph can be used in this context to represent knowledge about dishes, ingredients and customer preferences in a structured way. Each menu has a list of dishes. Each dish is linked to its ingredients, nutritional values (such as calories) and categories (e.g. vegetarian, gluten-free). Similarly, guest profiles can be represented as entities with specific preferences or restrictions. The knowledge graph can be queried to filter and display only dishes that are compatible with their needs.

5.3 Structure of Knowledge graph

To build the Knowledge Graph for the menu of dishes, we used GraphDB Desktop [Gra], an open source software designed to manage and query knowledge bases structured in the form of RDF graphs. This tool allowed us to semantically model information about dishes, ingredients, food categories, allergies and user preferences. Through GraphDB, it was possible to create and save the graph within a dedicated database, thus ensuring centralized and easily accessible data management. In addition, the software offers an intuitive interface to perform SPARQL queries, which allowed us to efficiently extract dishes compatible with the user's criteria. The use of GraphDB made intelligent navigation of knowledge possible, making the system effective and customizable.

5.3.1 Components of Graph

The graph was created starting from the menu structure, which represents the starting point for knowledge modeling. Inside it, the menu contains a set of dishes, each of which has been represented as a node connected to other nodes that describe its specific characteristics. The graph provides a detailed and relational representation of the menu, useful for applying filters and generating personalized recommendations based on user needs.

- **Menu**: As described above the menu was built as a set of dishes represented in the figure after the menu is the central node and the dishes are all connected to the menu via outgoing arcs and with the `hasMeal` predicate.

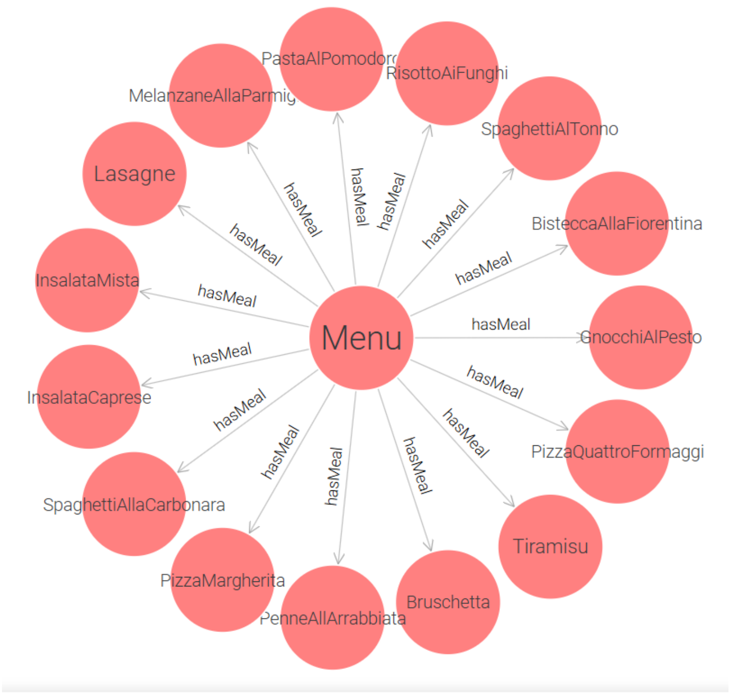


Figure 5.1: Menu Graph

- **Meal**: Each dish has four types of outgoing edges:
  - **hasIngredient**: ingredient or ingredients that the dish is made of.
  - **hasDietType**: type of diet with which the dish is associated.
  - **hasAllergen**: allergens present in the dish.
  - **hasCaloriesValue**: caloric value for a portion of that dish..

  Note that calories are associated with the dish.

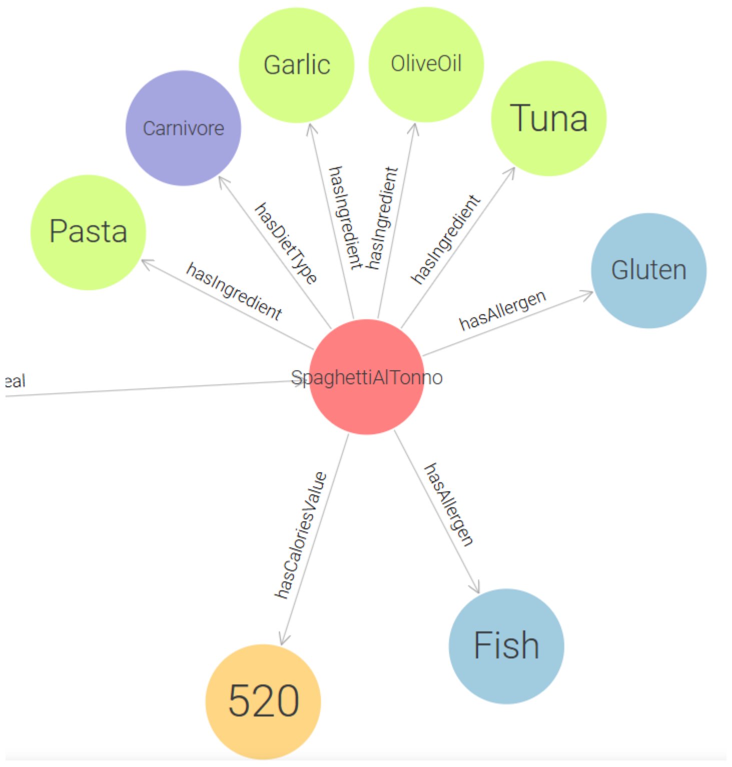


Figure 5.2: Meal Graph

5.4 SHACL for Graph Validation

This section describes what the SHACL language is and how it was used within the Knowledge Graph illustrated above. SHACL was used to define and apply validation rules on the graph data, ensuring that each entity (such as dishes, ingredients, allergens and dietary categories) respected a coherent and well-defined structure. Through the use of shapes, it was possible to specify which properties each node should have, with what type of value and with what cardinality. This allowed to improve the reliability of the graph and ensure a more controlled and precise management of the information.

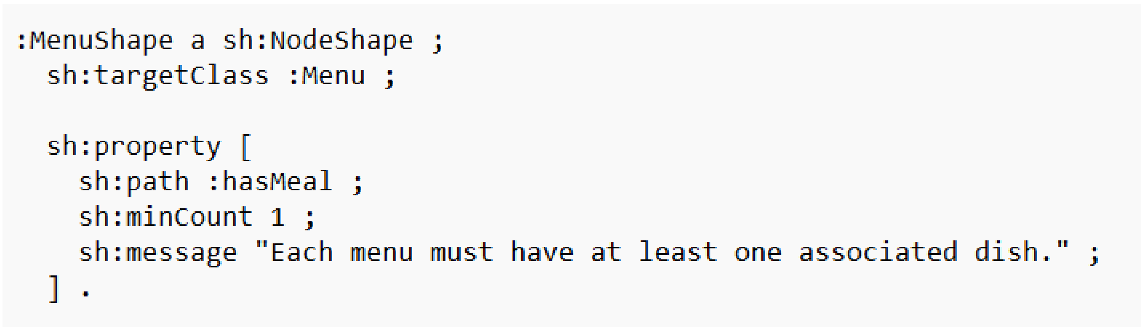
5.4.1 What is SHACL

SHACL (Shapes Constraint Language) is a language developed by the W3C to validate and describe data structures in RDF graphs (such as those used in knowledge graphs). It allows you to define shapes, or rules that describe which properties entities (nodes) must have and with which characteristics, such as the type of value, cardinality, or mandatory relationships. SHACL is useful for ensuring the consistency and quality of data, ensuring that the information inserted in the graph respects a well-defined schema. Furthermore, it facilitates the identification of errors and inconsistencies in the data, making the querying and processing of the knowledge graph more reliable.[Sha]

5.4.2 Constraint Using for Validate Graph

To validate the knowledge graph created on the menu we defined constraints for:

- **Menu**: A menu, to be considered as such, must contain at least one dish within it.



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

  sh:property [
    sh:path :hasMeal ;
    sh:minCount 1 ;
    sh:message "Each menu must have at least one associated dish." ;
  ] .
```

Figure 5.3: Menu Validation Rule

- **Meal**:To validate a dish it must contain: at least one ingredient, a minimum and maximum number of calories, at least one type of associated diet and allergens among those proposed.

```
:MealShape a sh:NodeShape ;
  sh:targetClass :Meal ;


  sh:property [
    sh:path :hasIngredient ;
    sh:minCount 1 ;
    sh:message "Each dish must have at least one ingredient." ;
  ] ;


  sh:property [
    sh:path :hasCalories ;
    sh:datatype xsd:integer ;
    sh:minInclusive 0 ;
    sh:maxInclusive 1500 ;
    sh:message "Calories must be a positive number between 0 and 1500." ;
  ] ;


  sh:property [
    sh:path :hasDietType ;
    sh:class :DietType ;
    sh:minCount 1 ;
    sh:message "Each dish must have at least one type of diet associated with it." ;
  ] ;


  sh:property [
    sh:path :hasAllergen ;
    sh:class :Allergen ;
    sh:minCount 0 ;
    sh:message "Allergens must be valid instances of Allergen." ;
  ] .
```

Figure 5.4: Meal Validation Rules

- **Guest**:To be considered valid, a guest must have valid type and allergens among those proposed.

```
:GuestShape a sh:NodeShape ;
  sh:targetClass :Guest ;


  sh:property [
    sh:path :hasDietPreference ;
    sh:class :DietType ;
    sh:minCount 0 ;
    sh:message "Dietary preferences must be instances of DietType." ;
  ] ;


  sh:property [
    sh:path :hasAllergy ;
    sh:class :Allergen ;
    sh:minCount 0 ;
    sh:message "Allergies must be valid instances of Allergen." ;
  ] .
```

Figure 5.5: Guest Validation SHACL

5.5 SPARQL query to query the Knowledge graph

In this section we will show the queries used to query the knowledge graph. These have been created in SPARQL in a simple, intuitive language and created specifically for these graphs.

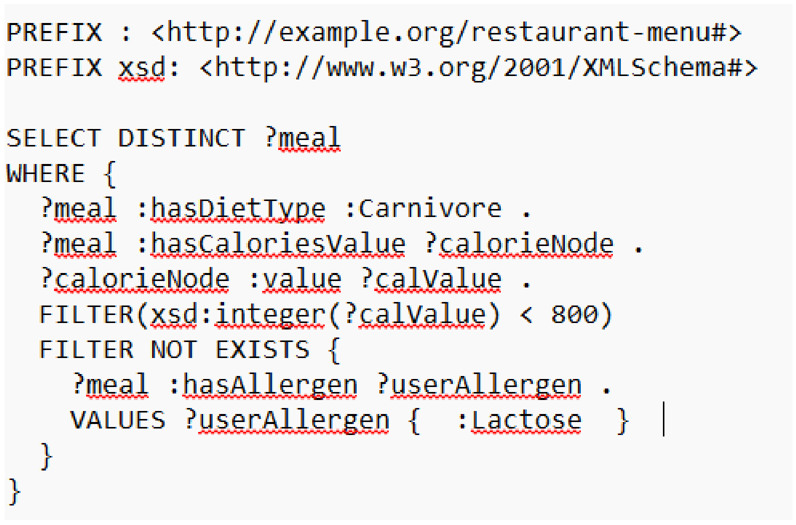
5.5.1 What are query in SPARQL

SPARQL is a query language designed to extract and manipulate data stored in RDF (Resource Description Framework) format. SPARQL queries allow you to search for information within knowledge bases structured as graphs, typical of the Semantic Web. Using SPARQL, you can specify triple subject-predicate-object patterns to return only the relevant data. It is used, for example, to query linked datasets such as DBpedia or Wikidata, obtaining complex answers efficiently. Queries can also filter, sort or combine the results. SPARQL supports advanced operations such as unions, optionality and aggregations. In practice, it is a fundamental tool for accessing and navigating structured and interconnected data at scale. [Spa]

5.5.2 Queries used

Below are some of the queries used to query the graph to help you understand if the data was loaded correctly.

- **All dishes on the menu with calories, type of diet and allergens proposed by the user**:

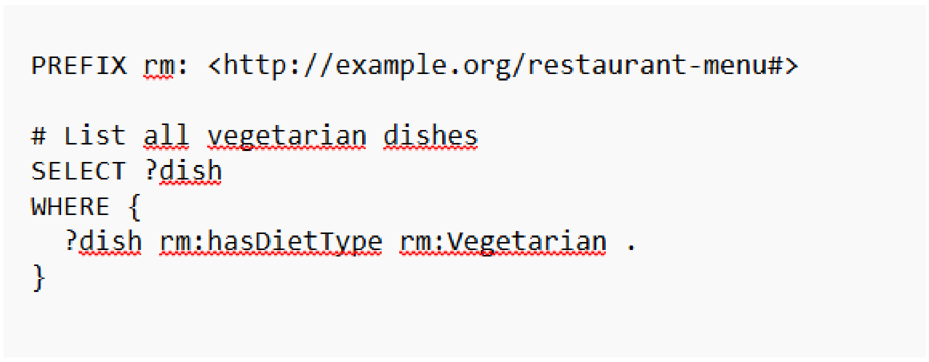


```
PREFIX : <http://example.org/restaurant-menu#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?meal
WHERE {
  ?meal :hasDietType :Carnivore .
  ?meal :hasCaloriesValue ?calorieNode .
  ?calorieNode :value ?calValue .
  FILTER(xsd:integer(?calValue) < 800)
  FILTER NOT EXISTS {
    ?meal :hasAllergen ?userAllergen .
    VALUES ?userAllergen {  :Lactose  }
  }
}
```

Figure 5.6: All dishes given by Guest Preferences

- **All the dishes on the menu with that type of diet**: List all vegetarian dishes

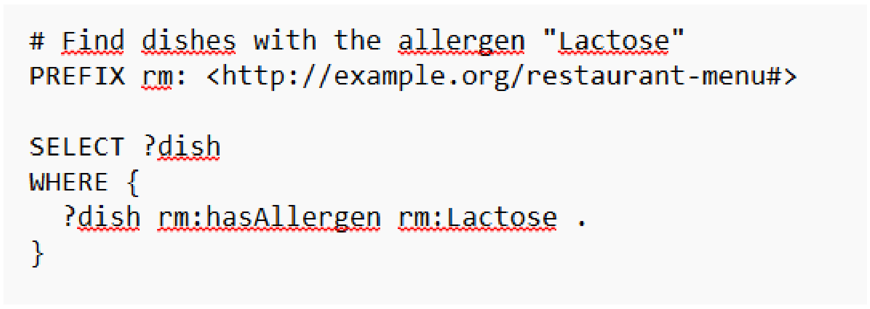


```
PREFIX rm: <http://example.org/restaurant-menu#>

# List all vegetarian dishes
SELECT ?dish
WHERE {
  ?dish rm:hasDietType rm:Vegetarian .
}
```

Figure 5.7: List all vegetarian dishes

- **All dishes on the menu with that type of allergen**:Find dishes with the allergen "Lactose"

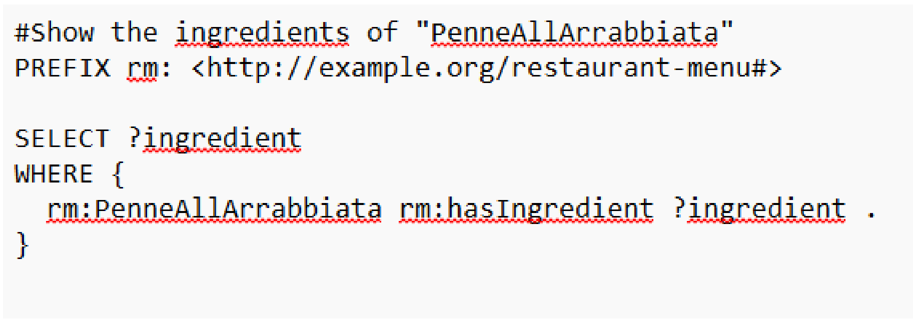


```
# Find dishes with the allergen "Lactose"
PREFIX rm: <http://example.org/restaurant-menu#>

SELECT ?dish
WHERE {
  ?dish rm:hasAllergen rm:Lactose .
}
```

Figure 5.8: Dishes with the allergen "Lactose"

- **All the ingredients of that specific dish.**:Show the ingredients of "PenneAllArrabbiata"

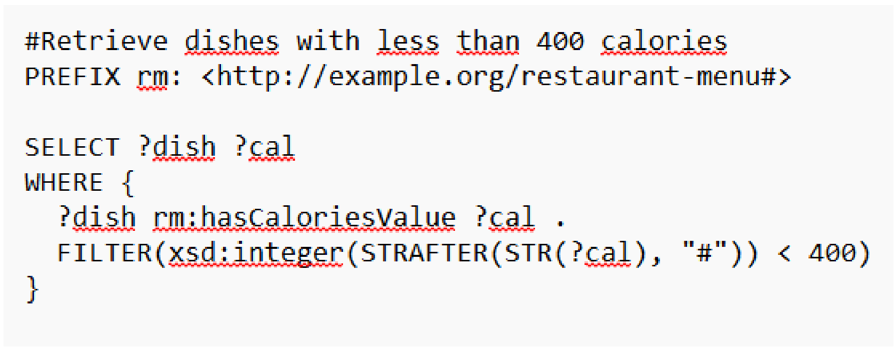


```
#Show the ingredients of "PenneAllArrabbiata"
PREFIX rm: <http://example.org/restaurant-menu#>

SELECT ?ingredient
WHERE {
  rm:PenneAllArrabbiata rm:hasIngredient ?ingredient .
}
```

Figure 5.9: Show the ingredients of "PenneAllArrabbiata"

- **All dishes on the menu with calories less than a certain number**:Retrieve dishes with less than 400 calories



```
#Retrieve dishes with less than 400 calories
PREFIX rm: <http://example.org/restaurant-menu#>

SELECT ?dish ?cal
WHERE {
  ?dish rm:hasCaloriesValue ?cal .
  FILTER(xsd:integer(STRAFTER(STR(?cal), "#")) < 400)
}
```

Figure 5.10: Retrieve dishes with less than 400 calories

5.6 SWRL Rules

In the last section of this chapter, we will discuss the use of rules in our project, illustrating how they have been implemented to enrich the ontology and improve the automatic inference process. Practical examples of SWRL rules applied to specific use cases will be presented, highlighting the added value in terms of coherence, completeness and deductive capacity of the system. This will allow us to concretely understand the central role of rules in knowledge modeling within the project.

5.6.1 What are SWRL's rules

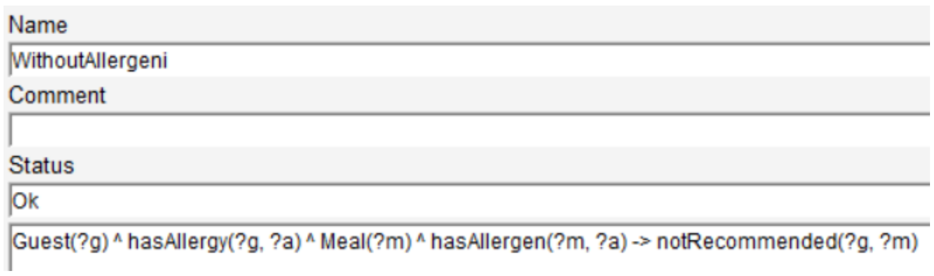
Semantic Web Rule Language (SWRL) rules are logical rules used to extend the inferential capabilities of OWL ontologies. They allow to express knowledge that goes beyond the possibilities of descriptive logic alone, by adding rules of the type "if... then...". A SWRL rule is composed of an antecedent (body) and a consequent (head): if the conditions of the body are true, then those of the head must also be true. These rules are used for automatic inferences, semantic validations or to enrich data with new inferred knowledge. They are often used in areas such as the semantic web, healthcare or intelligent systems. [Swr]

5.6.2 SWRL Rules Used

Below we will show the SWRL rules used to display the dishes that the user can eat. After adding the guest and specifying: type of diet, allergies, calorie limit.

There are three rules:

- **WithoutAllergens**:This rule states that if a guest (?g) has an allergy to a certain allergen (?a), and a meal (?m) contains that allergen, then the meal is not recommended for that guest. It is used to automatically exclude meals that are dangerous for those with allergies.



Name
WithoutAllergeni
Comment

Status
Ok

```
Guest(?g) ^ hasAllergy(?g, ?a) ^ Meal(?m) ^ hasAllergen(?m, ?a) -> notRecommended(?g, ?m)
```

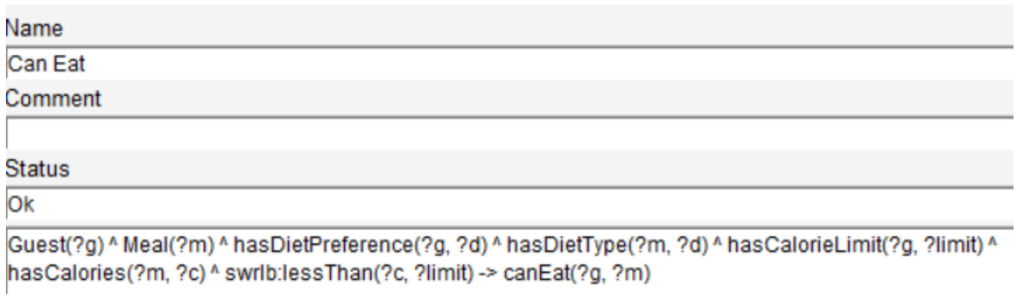
Figure 5.11: Exclude meals that are dangerous for those with allergies.

- **Can Eat**:This rule verifies that:

  -the meal (?m) is compatible with the guest's dietary preferences (?g);

  -the calorie content of the meal (?c) is less than the guest's calorie limit (?limit).

  If both conditions are true, the meal is suitable for consumption by the guest. This rule is useful for recommending personalized and healthy meals.



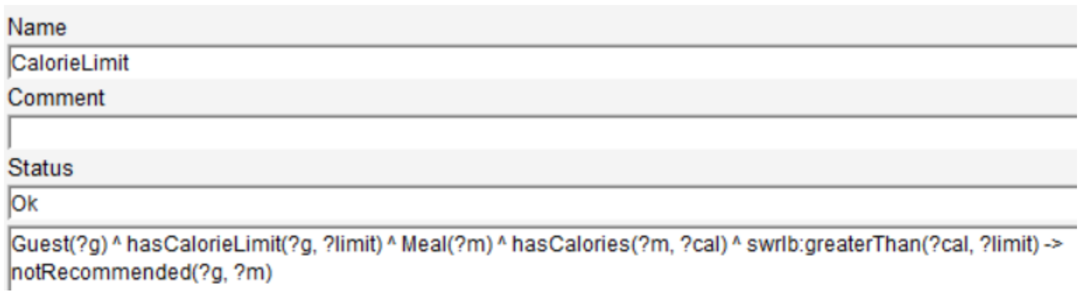
Name
Can Eat
Comment

Status
Ok

```
Guest(?g) ^ Meal(?m) ^ hasDietPreference(?g, ?d) ^ hasDietType(?m, ?d) ^ hasCalorieLimit(?g, ?limit) ^
hasCalories(?m, ?c) ^ swrlb:lessThan(?c, ?limit) -> canEat(?g, ?m)
```

Figure 5.12: Recommending personalized and healthy meals.

- **CalorieLimit**:This rule is the opposite of the previous one: if a meal has more calories than the guest's personal limit, then it is marked as not recommended. It is used to filter out meals that are too high in calories, even if they are perhaps dietary compatible.



Name
CalorieLimit
Comment

Status
Ok

```
Guest(?g) ^ hasCalorieLimit(?g, ?limit) ^ Meal(?m) ^ hasCalories(?m, ?cal) ^ swrlb:greaterThan(?cal, ?limit) ->
notRecommended(?g, ?m)
```

Figure 5.13: Retrieve dishes with less than calories

6. Ontology-Based BPMN Extension for Meal Recommendation

6.1 Overview

To enhance flexibility and automation in our personalized meal recommendation system, we explored an agile and ontology-based approach to extend BPMN 2.0. This method allows us to enrich traditional business process models with semantic knowledge, enabling more intelligent and context-aware decision-making. Specifically, we integrated BPMN tasks with ontology elements that describe guest preferences, meal ingredients, and dietary constraints, and connected these models to a knowledge graph through a triple store (Jena Fuseki) to retrieve recommendation results dynamically.

6.2 AOAME

AOAME is a prototype tool designed to apply an agile and ontology-based approach to meta-modeling. This method is intended to design and update schemas for Enterprise Knowledge Graphs (EKG), which organize the knowledge of a specific application domain, allowing the analysis, reasoning and integration of data from heterogeneous sources.

One of the main challenges in creating and managing an EKG schema is the need to combine skills in both ontology engineering and in-depth domain knowledge.

The proposed approach introduces elements of agile thinking into traditional metamodeling, offering the possibility to adapt modeling languages to domain specificities and to verify their effects in real time. This allows the direct involvement of domain experts in the knowledge engineering process.

AOAME, developed according to the principles of Design Science Research, has been created to evaluate the effectiveness of this approach. The prototype allows to extend, modify, delete or hide modeling constructs through meta-modeling operators that automatically generate SPARQL queries to update the triplestore, thus maintaining the alignment between the human-readable and machine-interpretable representations.[Aoa]

6.3 BPMN 2.0

BPMN, which stands for *Business Process Model and Notation*, is a standardized graphical language used for modeling business workflows. It offers a visual representation that outlines the sequence of tasks and the exchange of information within a business process. One of BPMN's primary strengths lies in its ability to be interpreted by a wide range of stakeholders, including business analysts, developers, and individuals directly involved in the process. [Bpm]

The BPMN 2.0 specification introduces several fundamental components that form the foundation of process diagrams:

- **Events**: Indicate things that happen and impact the process flow, such as start, end, or intermediate events.
- **Activities**: Define units of work or tasks to be performed, including manual tasks, user tasks, and automated service tasks.
- **Gateways**: Serve as control points for the process flow, managing decisions and parallelism (e.g., exclusive, parallel, and inclusive gateways).
- **Pools and Lanes**: Visualize the main participants in a process. Pools represent separate organizations or systems, while lanes subdivide roles or departments within a pool.
- **Artifacts**: Optional elements that enrich the process diagram by providing extra context, such as data objects, annotations, and groups.

In this project, the BPMN model is used to support the restaurant manager in visualizing and optimizing the customer order workflow. The BPMN model takes into account the customer's allergies, diet type, and other preferences such as the maximum calorie limit they want to consume.

During the ordering process, customers are asked to disclose any food allergies and to specify their dietary choice, i.e. whether they follow a vegetarian, vegan, or carnivorous diet. Once this information is collected, the customer can proceed to select the type of meal they want to order from those compatible with the preferences stated by the latter.

By integrating dietary considerations directly into the process model, the restaurant can better personalize the dining experience, thus increasing customer satisfaction and ensuring health and safety.

6.4 Our BPMN Diagram

We created the bpmn diagram by analyzing and following the flow of events and activities that are carried out by the user and the computer system of a possible restaurant that implements this approach during the process of choosing the dish that the user wants to eat among those proposed based on the preferences indicated.

The model is developed according to this execution flow:

- **Start Event**: The flow of the diagram begins and can be considered as the arrival of the customer at the restaurant.
- **Scan QR Code**: The customer scans the QR code to view the restaurant's complete menu.
- **Add User preferences**: The user specifies his/her preferences regarding the type of diet, allergies and maximum calorie limit he/she wishes to ingest.
- **Check custom preferences**: The system checks the preferences entered by the customer
- **Meal Suggestion**: The system suggests a set of dishes based on the preferences specified by the user
- **Show personalized menu**: The system shows the recommended dishes to the user.
- **Select Meal**: The user selects meals from those recommended by the system.
- **End Event**: Termination of the process

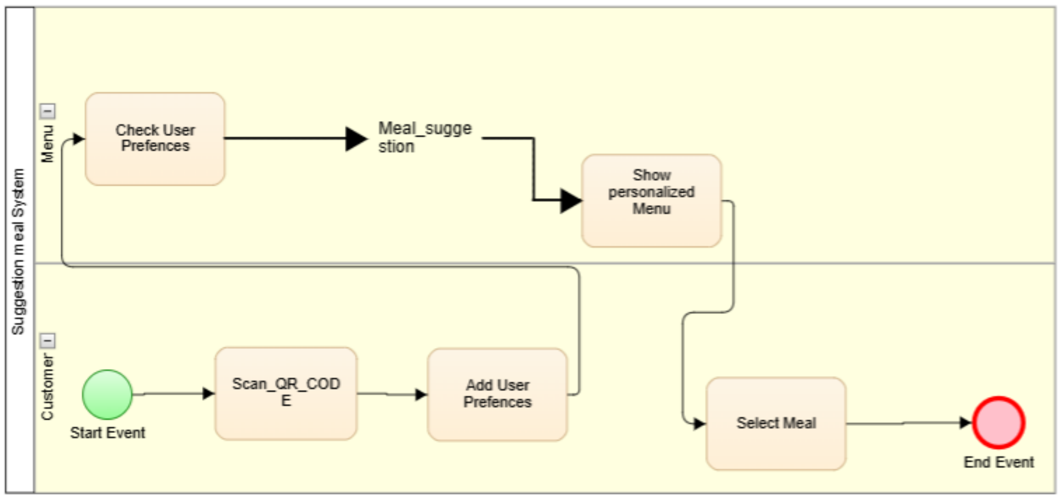


Figure 6.1: BPMN diagram

6.5 Query Execution with Jena Fuseki

Apache Jena Fuseki is a robust and scalable SPARQL server designed to handle RDF data efficiently. It provides a RESTful interface to perform various operations on RDF datasets, including: Execute SPARQL queries to retrieve data; Perform SPARQL updates to modify data; Load, store, and manage RDF datasets.[Jen]

Below we have listed all the SPARQL queries that were run to query the RDF graph and test whether the data was saved correctly in the graph.

- **Query that returns all dishes in the menù**

```
PREFIX : <http://example.org/restaurant-menu#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?dishName ?calorieCategory ?dietTypes ?allergens ?calories
WHERE {
  :Menu :hasMeal ?dish .
  BIND(STRAFTER(STR(?dish), "#") AS ?dishName)

  ?dish :hasCaloriesValue ?calorie .
  ?calorie :value ?calories .

  OPTIONAL {
    SELECT ?dish (GROUP_CONCAT(?dietType; separator=", ")
    AS ?dietTypes)
    WHERE {
      ?dish :hasDietType ?diet .
      BIND(STRAFTER(STR(?diet), "#") AS ?dietType)
    }
    GROUP BY ?dish
  }

  OPTIONAL {
    SELECT ?dish (GROUP_CONCAT(?allergenType; separator=", ")
    AS ?allergens)
    WHERE {
      ?dish :hasAllergen ?allerg .
      BIND(STRAFTER(STR(?allerg), "#") AS ?allergenType)
    }
    GROUP BY ?dish
  }
}
ORDER BY ?calories
```

This query returns all the dishes in the menu with their associated diet types, calories and allergens.

- **All dishes with a given type of diet, calorie limit and not have user allergen**

  ```
  PREFIX :<http://example.org/restaurant-menu#>
  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

  SELECT ?dishName ?calories
  WHERE {
    :Menu :hasMeal ?dish .
    BIND(STRAFTER(STR(?dish), "#") AS ?dishName)

    ?dish :hasCaloriesValue ?cal .
    ?cal :value ?calories .

    ?dish :hasDietType ?dietType .
    FILTER(STRAFTER(STR(?dietType), "#") = "Vegan")
    FILTER NOT EXISTS {
      ?dish :hasAllergen ?allergene .
      FILTER(STRAFTER(STR(?allergene), "#") = "Gluten")
    }
    FILTER(xsd:integer(?calories) < 600)
  }
  ORDER BY ?calories
  ```

  This query returns all dishes that are compatible with the type of diet, allergies, and calorie limit imposed by the user.

- **Dishes with a given type of diet**

  ```
  PREFIX : <http://example.org/restaurant-menu#>
  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  SELECT ?dishName
  WHERE {
    :Menu :hasMeal ?dish .
    ?dish :hasDietType ?dietType .
    FILTER(STRAFTER(STR(?dietType), "#") = "Vegan")
    BIND(STRAFTER(STR(?dish), "#") AS ?dishName)
  }
  ```

  This query contains all the dishes on the menu that are compatible with vegan users.

- **Dishes with less than a certain number of calories**

  ```
  PREFIX : <http://example.org/restaurant-menu#>
  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

  SELECT ?dishName ?calories
  WHERE {
    :Menu :hasMeal ?dish .
    ?dish :hasCaloriesValue ?cal .
    ?cal :value ?calories .
    FILTER(xsd:integer(?calories) < 500)
    BIND(STRAFTER(STR(?dish), "#") AS ?dishName)
  }
  ORDER BY ?calories
  ```

  This query contains all dishes with calories less than 500

- **Dishes that do not contain a given allergen**

  ```
  PREFIX : <http://example.org/restaurant-menu#>
  SELECT ?dishName
  WHERE {
    :Menu :hasMeal ?dish .
    BIND(STRAFTER(STR(?dish), "#") AS ?dishName)

    FILTER NOT EXISTS {
      ?dish :hasAllergen ?allergene .
      FILTER(STRAFTER(STR(?allergene), "#") = "Gluten")
    }
  }
  ```

  This query contains all that do not contain gluten

- **Ingredients of a certain dish**

  ```
  PREFIX : <http://example.org/restaurant-menu#>
  SELECT ?ingredientName
  WHERE {
    :SpaghettiAllaCarbonara :hasIngredient ?ingredient .
    BIND(STRAFTER(STR(?ingredient), "#") AS ?ingredientName)
  }
  ```

  This query contains all the ingredients of the dish ”SpaghettiAllaCarbonara”

6.6 Benefits of the Approach

By combining BPMN 2.0 with ontology-based extensions and semantic queries, we achieved the following:

- **Domain-awareness**: Process tasks are no longer abstract—they are connected to real-world concepts like guest profiles and ingredients.
- **Reusability and Modularity**: Ontological components and query templates can be reused across different processes or interfaces.
- **Stakeholder Accessibility**: Custom graphical elements improve communication and usability for business users.
- **Flexibility**: Changes in business rules (e.g., new dietary restrictions) can be handled by updating the ontology and queries, without changing the process structure.

6.7 Conclusion

The integration of BPMN with ontology-based modeling and semantic query execution (via Jena Fuseki) adds intelligence and flexibility to the meal recommendation system. This approach supports agile updates, reusable logic, and a user-friendly process view—all essential in dynamic environments like restaurants, where personalization and adaptability are key.

7. Conclusions

7.1 Alessandro Vitali

During the realization of this project, my colleague and I have been deeply engaged in knowledge engineering, exploring different knowledge-based solutions, each with its own advantages and disadvantages. Below I will explain my impressions of the various solutions developed, focusing on the positive and negative aspects.

- **DMN Diagrams:** DMN diagrams work in a simple and logical way, the main construct is decision tables, accepting inputs and generating outputs based on predefined rules organized in rows. The selection of outputs follows a "Hit Policy", which determines how the results are grouped. Our project used a "Collect" policy, which allows the simultaneous evaluation of all rules. According to this policy, the input conditions are structured in a simple format where the columns are connected by an "and" relationship, while the rows operate between them through an "or" relationship. The output results are aggregated in sets for the respective output columns. These decision tables have the advantage of being simple to understand and implement as they use a simple and intuitive logic.

  This operational clarity makes it intuitive and easy to implement.

  It must also be said that using these constructs in IT contexts, where there are several variables, complex operations to compute are not as efficient. Furthermore, the way our project was developed with the logic of the decision tables is not extendable and reusable at the IT level, because it is enough to add some ingredients or allergies and you have to change a lot of the decision tables, this goes against the rules of "good IT."

- **Prolog:** Instead, working with prolog was different because it is a programming language that is intuitive to understand and easy to use as it is a useful rules and facts scheme. The use of recursion structures makes it closer to pure functional languages. Using prolog you can overcome those disadvantages that I mentioned in decision tables, that is, you can create extensible and general programs.

  But prolog also has its disadvantages, such as the fact that it has difficulty managing large knowledge bases, which can put a strain on its interpreter, resulting in prolonged calculation times to obtain results. Furthermore, the behavior of Prolog can be non-transparent, bugs can be difficult to identify.

- **Knowledge Graph:** From my point of view, building knowledge graphs is a hybrid between the DMN and Prolog approaches, it is clear that as a structure it is a graph database.

  In this framework, we could define objects together with their relations and attributes, and extract knowledge from these object instances. This method of modeling knowledge made it easy to use and intrigued me, especially because of its query method very similar to standard SQL, which made it easier to use.

  The SHACL validator seemed very useful to me as it is very similar to the tests that are done in programming languages to find bugs.

  SWRL rules allow type inference, but in my opinion, its use should be limited. This perspective comes from experience with query tools like AOAME or GraphDB, which do not have built-in reasoning capabilities for type inference. As a result, queries had to be meticulously crafted, especially given the rules defined for meals, where inference determined whether a meal contained allergenic ingredients based on its components.

- **BPMN and Jena Fuseki:** To build the BPMN diagram we used, I found AOAME to be an extremely valuable tool due to its ability to adapt BPMN 2.0 to our ontology. This allowed us to create a much simpler and easier to read diagram than many others who do not use this tool. Specifically, creating a diagram that is easy to build and intuitive. Using Apache Jena Fuseki, we seamlessly integrated our ontology into a triplestore. This integration allowed us to run SPARQL queries, dynamically generating personalized menu suggestions that meet the specific needs of each guest. The process of creating queries to suggest meals based on user preferences was simple, largely due to the similarity in query development with Protege or GraphDB. Unfortunately, AOAME has many bugs that impact the user experience and system stability. Problems in saving diagrams that are not always done, or great difficulties in importing or exporting what is created with AOAME. Despite these disadvantages, AOAME remains a very powerful and very useful tool.

In conclusion, each knowledge-based solution has its own strengths and weaknesses as listed above. So, it can be concluded that the choice of the appropriate solution depends on the specific needs of the project, the complexity of the knowledge base, and the desired level of customization and scalability.

7.2 Lorenzo Gezzi

The project aimed to enhance the dining experience by creating a system that customizes restaurant menus according to guest preferences and dietary restrictions. We developed multiple knowledge-based solutions to recommend meals based on guest profiles using various representations:

- **Decsion Model and Notation:** During our project, I found that DRD and decision tables did a great job of organizing and automating the logic behind the personalized meal tips. They let me lay out clear, rules-based call-outs for every guest-whether the issue was diet type, allergies, or calorie limits-so the whole system stayed easy to read, plug in, and keep running. Because the charts are visual and modular, teammates with mixed technical backgrounds could still jump in and help without getting lost.

  Still, we ran into a few limits along the way. Once the number of rules piled up, the tables grew so big that managing them felt like juggling. Tweaking a table to match shifting user habits also wasnt always a smooth ride. On top of that, building a complete, accurate model took time and third-party tools like Trisotech, which we dont always have on hand. Even so, the clarity, reusability, and steady behavior they deliver make DRD and tables a solid choice for this kind of problem.

- **Prolog:** Prolog is great when you need to do symbolic reasoning. Its declarative style lets you say what the goal is without getting tangled up in the step-by-step mechanics. Built-in pattern matching and backtracking handle constraints naturally, saving you a lot of boilerplate code. Still, moving from an imperative world I stumbled over the non-linear control flow and the unique debugging speed bumps. Performance also dips once the data set grows or the rules stack up.

  In this project I relied on Prolog to match guests with meals that fit their diets. Filtering dishes for allergies, restrictions, and nutrition goals was just messy enough to suit Prologs strengths. I set up a knowledge base that listed meals by ingredients and calories and grouped guests as vegetarian, meat-lover, calorie-wise or allergic to nuts, shellfish, and so on. With a few logical rules Prologs inference engine scanned the facts and quickly spat out the right plates for each profile. The end logic closely mirrors formal specifications, making testing and validation straightforward.

- **Knowledge Graph:** For our project, we chose to build a knowledge graph to organize and show information about dishes, their ingredients, and what guests like or need. Working this way let me map how everything connects in a clear, flexible diagram. We tied each menu to a group of dishes, linked each dish to its ingredients, calorie counts, and tagsvegetarian, gluten-free, lactose-free. Guest profiles appear as their own nodes, describing any dietary limits, making it simple to check which meals fit which user.

  The biggest win has been how well the graph juggles tangled data. When query it-extract all vegetarian, gluten-free plates under, say, 600 calories-it reads almost like plain English and runs quickly, even as the set grows. Because the model speaks semantically, adding a new ingredient or guest trait later feels low-risk; the whole system stays understandable.

That said, building the graph hasnt been completely smooth. Laying out the ontology, spelling out each link, and guarding against overlap takes time and discussion. I also discovered that writing queries in SPARQL-or any similar dialect-can be tricky.

- **BPMN and Jena Fuseki:** What I liked about AOAME was how it seamlessly connected BPMN 2.0 to my ontology. This connection made the diagrams clearer and easier to follow than anything I had created with previous software. The notation spoke to a restaurant manager, not a programmer, which is exactly what I wanted.

  The BPMN was also very clear with specific colors for different tasks, gateways, etc.

  Finally, I added properties like "diet", "allergy", "calories" so the model was robust. Pushing the ontology into Apache Jena Fuseki completed the process. From there, I could run SPARQL queries against the triplestore and come up with meal ideas that would be appropriate for each customer. Creating these queries was easy, especially since I had already learned the basics from my previous job at Protégé.

  Of course, AOAME isn't perfect. I have encountered some bugs that occasionally made the tool unbalanced or difficult to manage, and the web version lagged when too many people were logged in at once. This could be improved, for example when I tried to insert an arrow it didn't do it correctly.

In conclusion, our project successfully demonstrates the integration of advanced knowledge representation techniques with practical applications in the restaurant industry, paving the way for more intelligent and user-friendly digital menu systems.

Acknowledgments

We are very grateful to Professors **Laurenzi and Hinkelmann** for the exceptional help in this project. Their recommendations and provision of tools and methods, were critical in allowing us to do our work. We value the ability to put theory and knowledge into practice with problem solving scenarios that gave a framework for learning about solution and how to implement those solutions in the real world. Their guidance and encouragement, have been vital for development and successful completion of this project.

Bibliography

[Aoa] URL: https://www.hinkelmann.ch/knut/lectures/nemo2019/Laurenzi_et_al_2018-AOAME-POEM_preprint.pdf.

[Bpm] URL: https://www.omg.org/spec/BPMN/2.0/.

[Dmn] URL: https://blog.maxconsilium.com/2014/09/introduction-to-decision-model-notation.html.

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

[Jen] URL: https://jena.apache.org/documentation/fuseki2/.

[Kno] URL: https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/.

[Pro] URL: https://www.swi-prolog.org/.

[Sha] URL: https://www.w3.org/TR/shacl/.

[Spa] URL: https://www.w3.org/TR/sparql11-query/.

[Swr] URL: https://www.w3.org/submissions/SWRL/.

[Tri] URL: https://www.trisotech.com/.