University of Camerino
School of Advanced Studies
Master of Science in
Computer Science & Mathematics

KEBI: Project Report

Supervisor
Prof. Knut Hinkelmann

Students
Francesco Finucci

Doctoral Examination Committee
Prof. YYYY
Prof. ZZZZ

Academic Year 2023-2024

IMPRINT

Title Title Title

Published by the University of Camerino, Camerino, Italy.

COLOPHON

This thesis was typeset using LaTeX and the `memoir` documentclass. It is based on Aaron Turon's thesis *Understanding and expressing scalable concurrency*[1], itself a mixture of `classicthesis`[2] by André Miede and `tufte-latex`[3], based on Edward Tufte's *Beautiful Evidence*.

The bibliography was processed by Biblatex. All graphics and plots are made with PGF/TikZ.

The body text is set 10/14pt (long primer) on a 26pc measure. The margin text is set 8/9pt (brevier) on a 12pc measure. Matthew Carter's Charter acts as the text typeface. Monospaced text uses Jim Lyles's `Bitstream Vera Mono` ("Bera Mono"). The display typeface is Knuth's `Concrete Modern`.

[1] `https://people.mpi-sws.org/~turon/turon-thesis.pdf`

[2] `https://bitbucket.org/amiede/classicthesis/`

[3] `https://github.com/Tufte-LaTeX/tufte-latex`

Declaration

I herewith declare that I have produced this paper under the supervision of Prof. XXXX at the University of Camerino, without the prohibited assistance of third parties and without making use of aids, other than those specified. Notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in an identical or similar form to any other Italian or foreign examination board.

XXXXXXXXXXXXXXXXX
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the School of Advanced Studies of University of Camerino.

XXXXXXXXXXXXXX,
XXXXXXXXXXXXXX
—XXXXX

Abstract

write the abstract

Contents

List of Publications

- xxxxx

List of Algorithms

List of Figures

List of Tables

Part I

PROLOGUE

1

Introduction

"In the end, it's not the technology that matters, but our ability to use it to improve and enrich the lives of people."
—Tim O'Reilly

In today's digital era, many restaurants have transitioned to digitized menus accessible through QR codes. While this technological advancement offers convenience, it also presents challenges, especially for guests with specific dietary preferences or restrictions. The small screen size of smartphones can make it difficult to get a comprehensive and full view of a menu, and guest often have to shift through numerous options that may not align with their dietary needs.

To address these issues, this project aims to develop a knowledge-based system that tailors menu recommendations based on individual guest preferences. By leveraging knowledge engineering techniques, we can create a system that filters and presents only those meals that match the guest's dietary profile. This enhances the dining experience by providing a more personalized and manageable menu.

Knowledge engineering plays a crucial role in this project, as it involves the creation, representation and utilization of knowledge to solve complex problems. The solutions created in this project include decision tables, Prolog, and knowledge graphs/ontologies, each offering unique advantages for representing and querying knowledge.

In developing our solution, we utilized the menu from the Italian restaurant Nero Balsamico as a case study (https://www.nerobalsamico.it/). This real-world example provided a rich dataset of typical Italian meals, including pizzas, pastas, and main dishes, along with detailed information about their ingredients and nutritional content. It surely added complexity rather than mocking a simple and common Menu but we gladly went through with the challenge.

The report is structured as follows:

1. **Introduction**
    - Overview of the project, its objectives, and the importance of knowledge engineering.

2. **Decision Tables and DRD**
    - Explanation of decision tables and Decision Requirements Diagrams (DRD).
    - Creation and implementation of decision tables for menu recommendations.
3. **Prolog Implementation**
    - Introduction to Prolog and its use in knowledge representation.
    - Development of Prolog facts and rules for guest-specific meal recommendations.
4. **Knowledge Graphs and Ontologies**
    - Explanation of knowledge graphs and ontologies.
    - Use of SWRL, SPARQL, and SHACL for meal recommendation queries and rules.
5. **AOAME**
    - Introduction to the AOAME tool.
    - Modeling of our ontology leveraging the power of AOAME.
6. **Evaluation and Conclusions**
    - Evaluation of the system's performance.
    - Discussion of results and potential improvements made by each individual member of the project.

By the end of this report, we will have demonstrated how knowledge-based systems can significantly enhance the dining experience by providing personalized and relevant menu recommendations. This project not only showcases the practical application of knowledge engineering but also highlights the potential for such systems to be integrated into modern restaurant operations. You can view all the project implementation and try it out yourself by checking the following **Github Repository**: (https://github.com/Meguazy/project_KEBI).

2

General Logic

In this chapter, we will provide a comprehensive analysis and explanation of the key logical concepts that recur throughout our paper. These concepts form the backbone of our approach, enabling us to develop solutions that are both efficient and adaptable. By understanding these foundational principles, the reader will gain deeper insights into how our system operates and how it manages complex decision-making processes, particularly in the context of dietary restrictions and meal recommendations.

2.1 HIERARCHICAL LOGIC IN DIETARY RESTRICTIONS

In this project, we implemented a hierarchical logic system to effectively manage dietary restrictions across all the solutions. This hierarchy plays a crucial role in streamlining the decision-making processes, particularly when determining the suitability of ingredients for various diets. The hierarchy is structured from the most general to the most specific dietary categories: **omnivore**, **vegetarian**, and **vegan**.

2.1.1 *Structure of the Hierarchy*

The logic behind this hierarchical structure is simple yet powerful. It categorizes ingredients based on their compatibility with different dietary restrictions, enabling a more efficient and flexible filtering process.

- **Omnivore**: This is the most general category. Ingredients labeled as omnivore include both animal-based and plant-based foods. These ingredients are only suitable for individuals who follow an omnivorous diet, meaning they do not avoid any food groups. By labeling an ingredient as omnivore, we indicate that it is not suitable for vegetarians or vegans.
- **Vegetarian**: Moving one step more specific, vegetarian ingredients exclude meat and fish but may include dairy and eggs. These ingredients are suitable for both vegetarians and omnivores. Vegetarian ingredients are a middle ground, offering more specificity than omnivore ingredients while still being less restrictive than vegan ingredients.

- **Vegan**: At the highest level of specificity, vegan ingredients exclude all animal products, including meat, dairy, eggs, and any other animal-derived substances. These ingredients are the most restrictive but also the most inclusive in terms of dietary compatibility. Vegan ingredients can be consumed by everyone—vegans, vegetarians, and omnivores alike.

2.1.2 *Application of the Hierarchy*

This hierarchical system is applied across all the solutions in the project to simplify the logic and make the decision-making process more efficient. By categorizing ingredients according to this hierarchy, the system can quickly and accurately filter out unsuitable options based on the client's dietary needs.

Example:

- If a client is vegan, only vegan ingredients are considered, ensuring that the meal recommendations are strictly aligned with their dietary preferences.
- For vegetarians, both vegetarian and vegan ingredients are available, but omnivore ingredients are excluded from the options.
- Omnivores have the most flexibility, with all ingredients being considered unless specific allergens are noted.

2.1.3 *Benefits of Hierarchical Logic*

The use of hierarchical logic in this project brings several key benefits:

- **Efficiency**: The hierarchy allows the system to quickly filter and identify suitable ingredients, reducing the complexity of the logic needed to make these decisions.
- **Scalability**: As new ingredients are introduced, they can be easily integrated into the existing hierarchy without requiring major changes to the system's logic.
- **Flexibility**: This approach ensures that the system can cater to a wide range of dietary needs, from the most general to the most specific, without sacrificing accuracy or personalization.

In summary, the hierarchical logic system is a fundamental component of the project's overall structure. It ensures that the solutions are both adaptable and robust, capable of delivering accurate and personalized recommendations across a diverse set of dietary requirements.

Part II

DECISION TABLES

3

Decision Tables and DRD

"A decision table is the simplest form of representing the decision logic in a tabular manner, which helps in organizing and ensuring completeness."
—Anonymous

3.1 DRD DESCRIPTION

Decision tables were the first step towards our solutions; they are a powerful tool used in decision-making processes. They provide a structured way to represent decision logic by mapping different conditions to corresponding actions through a tabular representation. We leveraged the power of decision tables to model our system with rules that allowed us to filter the initial ingredients list to ensure we would consider the Guest's dietary needs and allergens. During our solutions, we made extensive use of the FEEL language, which enabled us to create a more dynamic system. We followed a course on Camunda about Decision Tables to deeply understand their advantages and managed to elaborate every single table in our DRD with powerful FEEL expressions.

To have a deep understanding of the solution we provided through the decision tables, we shall start by describing our DRD, a Decision Requirements Diagram, which illustrates the decision-making process for recommending meals based on client-specific dietary information.

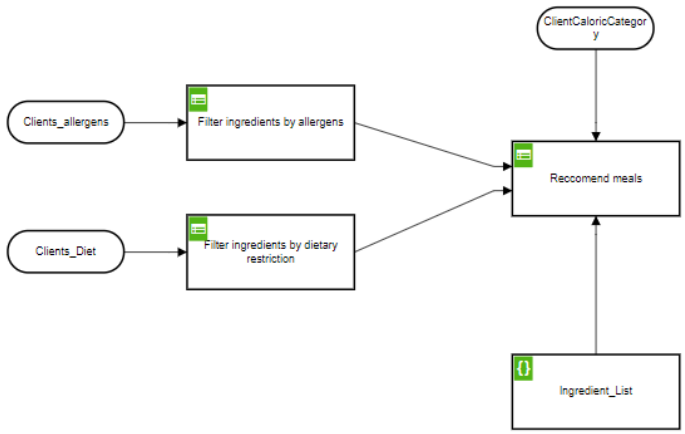

FIGURE 3.1: Decision Requirements Diagram (DRD) for Recommending Meals Based on Dietary Preferences

As we can see from Figure 3.1, this DRD is composed of several elements. We will analyze each element individually and provide an accurate description. The diagram is made up of the following components:

- **Filter Ingredients By Allergens:** This is a decision table that filters out ingredients containing allergens specified by the clients.
  - These allergens are provided as input data by the component **Clients_allergens.**
- **Filter By Dietary Restriction:** This decision table is responsible for filtering out ingredients based on the client's dietary restrictions, such as Vegan, Vegetarian, and Omnivore.
  - The specific diet of the client is given by an input data shape called **Clients_Diet.**
- **Ingredient_List:** This, unlike the others, is a literal expression and represents the comprehensive list of all possible ingredients available for meal preparation, mapped to their kcal values.
- **Recommend Meals:** This central decision node integrates filtered results based on allergens, dietary restrictions, and caloric categories to recommend appropriate meals based on the client's needs. It represents the heart of our DRD and is responsible for collecting and combining information from other decision tables. As we can see from the inputs, it is linked with almost every element. There is also a new element called **ClientsCaloricCategory**, which we will discuss further in the project report.

First and foremost, it is important to note that we initially used a sequential filtering approach. We later concluded that a sequential approach might have some potential problems in the future regarding extensibility and maintainability, as it was very difficult to retrieve information from previous tables that weren't directly linked. The new DRD's modular design allows each decision component (e.g., filtering by allergens, dietary restrictions, and caloric categories) to be developed, tested, and maintained independently. We have a single responsibility where each table has one logical responsibility. This modularity makes it easier to add new decision criteria or modify existing ones without affecting the entire system.

Another significantly improved aspect is the **Scalability**. As new types of dietary requirements or allergens might emerge, additional decision nodes can be incorporated into the DRD without overhauling the entire decision-making process. This ensures the system remains scalable and adaptable to evolving needs. In fact, unlike sequential filtering, where each step depends on the previous one, the DRD allows for parallel processing of decision rules, leading to more efficient data processing as multiple conditions can be evaluated simultaneously.

The DRD also enhances **Maintainability** by making it straightforward to update specific decision rules or add new ones. This flexibility is crucial for adapting to new dietary trends and regulations, ensuring that the system remains relevant and accurate over time.

In summary, the use of DRD in our project significantly improved the system's extensibility, scalability, and maintainability, providing a robust framework for dynamic and personalized meal recommendations.

3.2 DECISION TABLES ANALYSIS

In this section we are going to analyze the single decision tables, explaining their implementation and semantics.

3.2.1 *Filter Ingredients By Allergens Decision Table*

The **Filter Ingredients by Allergens** decision table plays a crucial role in ensuring that the meal recommendations are safe for clients with specific allergen sensitivities. This decision table utilizes the FEEL (Friendly Enough Expression Language) to dynamically filter out ingredients that contain allergens specified by the client. Below, we describe the table in detail and explain the FEEL expressions used.

Filter ingredients by allergens | Hit policy: Collect

	When	Then	Annotations
	Clients_allergens string	Filtered_Ingredients_By_Allergens string	
1	-	"lemon_sorbet"	
2	-	"fresh_fruit_salad"	
3	-	"caramelized_orange"	
4	not(contains(lower case(?), "gluten"))	"pink_peppercorn_tuille"	
5	not(contains(lower case(?), "lactose"))	"anise_panna_cotta"	
6	-	"fresh_raspberries"	
7	not(contains(lower case(?), "lactose"))	"zabaglione_chantilly_cream"	
8	not(contains(lower case(?), "gluten"))	"crisp_wafer"	
9	not(contains(lower case(?), "gluten") or contains(lower case(?), "lactose") or contains(lower case(?), "nuts"))	"chocolate_macaron"	
10	-	"custard"	
11	-	"alchermes"	
12	not(contains(lower case(?), "gluten") or contains(lower case(?), "lactose"))	"sponge_cake"	
13	-	"coffee_coral"	
14	not(contains(lower case(?), "lactose"))	"tiramisu_parfait"	
15	not(contains(lower case(?), "lactose"))	"mascarpone_cream"	
16	not(contains(lower case(?), "lactose"))	"barozzi_cake"	

FIGURE 3.2: Decision Table: Filter Ingredients by Allergens

3.2.2 *Understanding the Decision Table*

The decision table is structured into the following main columns:

- **When (Conditions)**: This column checks the presence of specific allergens in the client's allergen list.

- **Then (Actions)**: If the condition is met, the action specifies which ingredient should be included or excluded from the filtered list based on the allergen content.

The decision table operates under the **Hit policy: Collect**, which means that all applicable rules are executed, and their results are collected. This policy is particularly useful in this case since we need to filter **multiple** ingredients based on the client's allergen profile.

FEEL Expressions in Detail

The FEEL expressions used in the **When** column are as follows:

- **Expression 1:** `not(contains(lower case(?), "gluten"))`
  This expression checks if the client's allergen list does not contain the term "gluten". The function `lower case(?)` converts the allergen list to lowercase to ensure case-insensitive matching, and `contains` checks for the presence of the string "gluten". If "gluten" is not found, the corresponding ingredient (in the **Then** column) is considered safe and added to the filtered list.
- **Expression 2:** `not(contains(lower case(?), "lactose"))`
  Similar to the first expression, this checks for the absence of "lactose" in the allergen list. If lactose is not present, the ingredient is deemed safe for inclusion.
- **Expression 3:** `not(contains(lower case(?), "egg"))`
  This checks the absence of egg in the client's allergen list.
- **Expression 4:** `not(contains(lowercase(?),"nuts"`
  This last expression checks for the absence of nuts in the client allergens list.
- **Expression 5:** `-`
  A hyphen (`-`) in the **When** column indicates a default rule that always applies if no specific condition is matched. In this table, it typically allows non-allergenic ingredients to pass through the filtering process.

The previous FEEL expressions take the client's data as an input and suggest ingredients that do not contain something that could harm the client based on his allergens. In our example we considered just gluten,lactose,egg and nuts. Since they are the most common ones, but in future implementations we could consider inserting more allergens to make the system more inclusive. It is important to note that in our table there are **combinations of previous FEEL expressions**, in fact by using a logic OR, we contemplated the cases where an ingredient contains more than just one allergen.

The output of the *Filter Ingredients By Allergens Table* will be a list of Ingredients that are suitable for the client allergens.The use of FEEL

expressions allows for a flexible and dynamic approach to filtering ingredients, making the system both powerful and adaptable to various client needs. This list will be used later on on our main table.

3.2.3 *Filter Ingredients By Dietary Restriction Decision Table*

The **Filter Ingredients by Dietary Restriction** decision table is essential for tailoring meal recommendations according to the client's dietary preferences, such as omnivore, vegetarian, and vegan diets. This table, similar to the allergen filtering table, uses FEEL (Friendly Enough Expression Language) to dynamically filter ingredients based on the client's diet type. Below, we describe the table in detail and explain the FEEL expressions used.

Filter ingredients by dietary restriction | **Hit policy:** Collect

	When Clients_Diet string	Then Filtered_Ingredients_By_Diet string	Annotations
1	contains(lower case(?), "omnivore")	"prosciutto_crudo"	
2	contains(lower case(?), "omnivore")	"coppa"	
3	contains(lower case(?), "omnivore")	"salame"	
4	contains(lower case(?), "omnivore")	"mortadella"	
5	contains(lower case(?), "omnivore")	"beef"	
6	contains(lower case(?), "omnivore")	"mullet_roe"	
7	contains(lower case(?), "omnivore")	"octopus"	
8	contains(lower case(?), "omnivore")	"meat_broth"	
9	contains(lower case(?), "omnivore")	"beef_ragu"	
10	contains(lower case(?), "omnivore")	"red_shrimp_tartare"	
11	contains(lower case(?), "omnivore")	"beef_filet"	
12	contains(lower case(?), "omnivore")	"tuna"	

FIGURE 3.3: Decision Table: Filter Ingredients by Dietary Restriction

3.2.4 *Understanding the Decision Table*

The decision table is structured into the following main columns:

- **When (Conditions)**: This column checks the specific dietary restriction of the client.
- **Then (Actions)**: If the condition is met, the action specifies which ingredient should be included or excluded from the filtered list based on the client's dietary restriction.

The decision table, like the previous one, operates under the **Hit policy: Collect**, which means that all applicable rules are executed, and their results are collected.

FEEL Expressions in Detail

The FEEL expressions used in the **When** column are as follows:

- **Expression 1:** `contains(lower case(?), "omnivore")`
  This expression checks if the client's diet is omnivore. The function `lower case(?)` converts the diet input to lowercase to ensure case-insensitive matching. If the client's diet is omnivore, the corresponding ingredient (in the **Then** column) is considered suitable and added to the filtered list.
- **Expression 2:** `contains(lower case(?), "vegetarian")`
  This expression checks if the client's diet is vegetarian. It filters out ingredients that are not compatible with a vegetarian diet.
- **Expression 3:** `contains(lower case(?), "vegan")`
  This expression checks if the client's diet is vegan. It filters out ingredients that are not suitable for a vegan diet, such as any animal products.
- **Expression 4:** `-`
  A hyphen (`-`) in the **When** column indicates a default rule that always applies if no specific condition is matched. This typically allows ingredients that are universally acceptable (i.e., suitable for all diet types) to pass through the filtering process.

Note: The FEEL expressions are **case-sensitive**, meaning that the input data must match the expected case format. This is why the function `lower case($)` is used to normalize the input, ensuring that the condition check is not affected by case variations in the input data.

Example Rules and Outcomes

- **Rule 1**: `contains(lower case(?), "omnivore")` ⇒ `"prosciutto_crudo"`
  If the client's diet is omnivore, "prosciutto_crudo" is considered suitable and added to the filtered ingredients list.
- **Rule 2**: `contains(lower case(?), "omnivore")` ⇒ `"coppa"`
  If the client's diet is omnivore, "coppa" is included in the filtered list.
- **Rule 3**: `contains(lower case(?), "omnivore")` ⇒ `"salame"`
  If the client's diet is omnivore, "salame" is considered appropriate and added to the list.

3.2.5 *Conclusion*

The *Filter Ingredients By Dietary Restriction* decision table dynamically adjusts the list of suitable ingredients based on the client's dietary profile. The use of FEEL expressions allows for a flexible and robust filtering mechanism, ensuring that the system can adapt to various dietary needs while maintaining accuracy and relevance in meal recommendations.

This filtered list will then be utilized in subsequent decision-making steps to recommend meals that align with the client's dietary restrictions.

3.2.6 *Recommend Meals Decision Table*

The **Recommend Meals** decision table integrates the filtered results from both allergens and dietary restrictions, along with the client's desired caloric intake, to recommend appropriate meals. This table uses FEEL expressions to dynamically assess the suitability of a meal based on the combined criteria. Below, we describe the table in detail, including how the ingredients are mapped to their caloric values using the **Ingredient_List**.

Recommend Meals | Hit policy: Collect

	When	And	And	And	Then	And	Annotations
	Filtered_Ingredients_By_Allergens	Filtered_Ingredients_By_Diet	Ingredient_List	ClientCaloricCategory	Reccomended_Meals	Menu_Category	
	list	list	list	"low","medium","high"	string	string	
7	list contains(Filtered_Ingredients_By_Allergens, "spinach") and list contains(Filtered_Ingredients_By_Allergens, "parmesan_cheese") and list contains(Filtered_Ingredients_By_Allergens, "mullet_roe")	list contains(Filtered_Ingredients_By_Diet, "spinach") and list contains(Filtered_Ingredients_By_Diet, "parmesan_cheese") and list contains(Filtered_Ingredients_By_Diet, "mullet_roe")	(Ingredient_List[name="spinach"].calories[1] + Ingredient_List[name="parmesan_cheese"].calories[1] + Ingredient_List[name="mullet_roe"].calories[1]) <=400	"low"	"Spinach flan on Parmigiano Reggiano cream with mullet roe"	"Appetizers"	
8	list contains(Filtered_Ingredients_By_Allergens, "spinach") and list contains(Filtered_Ingredients_By_Allergens, "parmesan_cheese") and list contains(Filtered_Ingredients_By_Allergens, "mullet_roe")	list contains(Filtered_Ingredients_By_Diet, "spinach") and list contains(Filtered_Ingredients_By_Diet, "parmesan_cheese") and list contains(Filtered_Ingredients_By_Diet, "mullet_roe")	(Ingredient_List[name="spinach"].calories[1] + Ingredient_List[name="parmesan_cheese"].calories[1] + Ingredient_List[name="mullet_roe"].calories[1]) <=900	"medium"	"Spinach flan on Parmigiano Reggiano cream with mullet roe"	"Appetizers"	
9	list contains(Filtered_Ingredients_By_Allergens, "spinach") and list contains(Filtered_Ingredients_By_Allergens, "parmesan_cheese") and list contains(Filtered_Ingredients_By_Allergens, "mullet_roe")	list contains(Filtered_Ingredients_By_Diet, "spinach") and list contains(Filtered_Ingredients_By_Diet, "parmesan_cheese") and list contains(Filtered_Ingredients_By_Diet, "mullet_roe")		"high"	"Spinach flan on Parmigiano Reggiano cream with mullet roe"	"Appetizers"	

FIGURE 3.4: Decision Table: Recommend Meals

3.2.7 *Understanding the Decision Table*

The decision table is structured into several key columns:

- **Filtered_Ingredients_By_Allergens**: This column contains the list of ingredients that have passed the allergen filtering step.
- **Filtered_Ingredients_By_Diet**: This column contains the list of ingredients that have passed the dietary restriction filtering step.
- **Ingredient_List**: This column uses the provided **Ingredient_List** to calculate the total caloric value of the selected ingredients.
- **ClientCaloricCategory**: This column represents the caloric category selected by the client (e.g., "low", "medium", "high").
- **Recommended_Meals**: If all conditions are satisfied, this column specifies the meal to be recommended.
- **Menu_Category**: This column identifies the category of the recommended meal (e.g., "Appetizers", "Main Course").

The decision table operates under the **Hit policy: Collect**, meaning that all applicable rules are executed and their results are collected. This is crucial for ensuring that the system can recommend multiple meals that fit the client's preferences and dietary needs.

Ingredient List and Caloric Mapping

The **Ingredient_List** (shown in Figure 3.5) maps each ingredient to its corresponding caloric value. This list is used in the **Recommend Meals** table to calculate the total caloric content of the ingredients in a meal. The FEEL expressions used for this purpose include the following:

- Expression: `Ingredient_List[name="spinach"].calories[1] + Ingredient_List[name="parmesan_cheese"].calories[1] + Ingredient_List[name="mullet_roe"].calories[1]`
  This expression sums the caloric values of "spinach", "parmesan_cheese", and "mullet_roe" to determine the total calories of a potential meal. The `calories[1]` syntax indicates that we are accessing the first (and typically only) entry of the caloric value for each ingredient in the list.

Caloric Categories: Low, Medium, and High

Clients can specify their desired caloric intake by selecting a caloric category:

- **Low**: Meals with a total caloric content of **400 calories or less**.
- **Medium**: Meals with a total caloric content between **400 and 900 calories**.
- **High**: Meals with a total caloric content of **over 900 calories**.

The decision table uses these categories to ensure that the recommended meals align with the client's caloric preferences.

3.2.8 *Example Rules and Outcomes*

- **Rule 7**: If the ingredients "spinach", "parmesan_cheese", and "mullet_roe" are present in both the allergen and diet-filtered lists, and their combined caloric value is **less than or equal to 400 calories**, then recommend "Spinach flan on Parmigiano Reggiano cream with mullet roe" as a low-calorie appetizer.
- **Rule 8**: If the same ingredients are present and the combined caloric value is **between 400 and 900 calories**, then recommend the same dish as a medium-calorie appetizer.
- **Rule 9**: If the caloric value exceeds **900 calories**, recommend the same dish as a high-calorie appetizer.

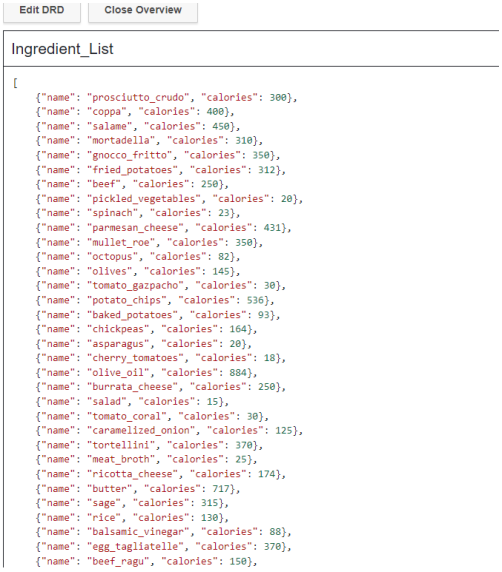

FIGURE 3.5: Ingredient List with Caloric Mapping

3.2.9 *Conclusion*

The *Recommend Meals* decision table effectively integrates dietary restrictions, allergen considerations, and caloric preferences to provide personalized meal recommendations. By leveraging the **Ingredient_List** for caloric calculations, the system ensures that clients receive meals that are not only safe and suitable but also align with their nutritional goals.

3.3 SIMULATION EXAMPLES

This section explains the results generated by the Camunda simulator based on various inputs. We will explore how the inputs lead to the specific outputs and how the decision tables are executed to provide meal recommendations.

3.3.1 *Simulator Inputs and Outputs Overview*

The first image (3.6) shows the interface where inputs are provided and outputs are generated. The inputs include the decision table selection, allergens, dietary restrictions, and caloric category. The outputs display the filtered ingredients and the recommended meals.

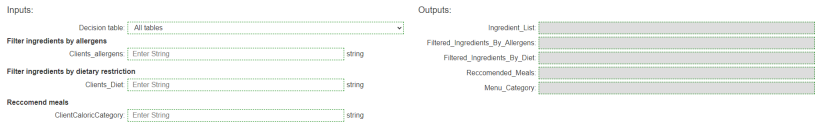

FIGURE 3.6: Camunda Simulator Inputs and Outputs Interface

Inputs:

- **Decision Table**: The user can select "All tables" to apply the full decision logic across all available decision tables.

- **Filter Ingredients by Allergens**:
  - **Clients_allergens**: Specifies any allergens the client may have, such as "lactose" and "gluten".
- **Filter Ingredients by Dietary Restriction**:
  - **Clients_Diet**: Specifies the dietary preferences of the client, such as "Vegetarian", "Vegan", or "Omnivore".
- **Recommend Meals**:
  - **ClientCaloricCategory**: Specifies the desired caloric intake category: "low", "medium", or "high".

Outputs:

- **Ingredient_List**: The complete list of ingredients with their respective caloric values.
- **Filtered_Ingredients_By_Allergens**: Ingredients that have passed the allergen filtering step.
- **Filtered_Ingredients_By_Diet**: Ingredients that match the client's dietary restrictions.
- **Recommended_Meals**: The meals that meet all the client's criteria, including allergens, diet, and caloric category.
- **Menu_Category**: The category under which the recommended meal falls, such as "Appetizers" or "Main Dishes".

3.3.2 *Scenario 1: Vegetarian Client with Lactose and Gluten Allergies*

In the second scenario (3.7), the inputs are as follows:

- **Clients_allergens**: "lactose, gluten"
- **Clients_Diet**: "Vegetarian"
- **ClientCaloricCategory**: "medium"

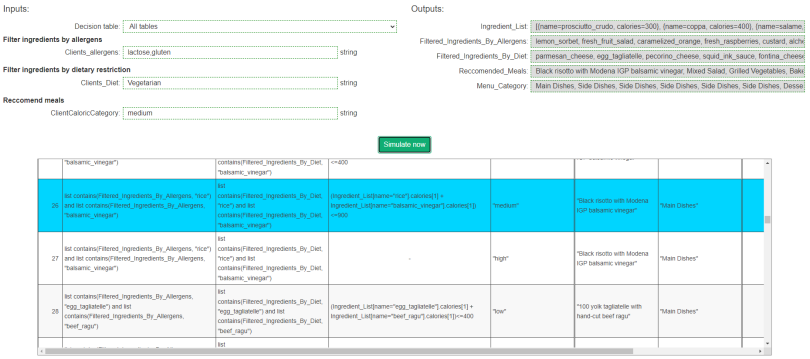

FIGURE 3.7: Example Scenario: Vegetarian Client with Lactose and Gluten Allergies

Outputs:

- **Filtered_Ingredients_By_Allergens**: The list includes ingredients like "lemon sorbet", "fresh fruit salad", and "custard", which are free from lactose and gluten.
- **Filtered_Ingredients_By_Diet**: The list includes vegetarian-friendly ingredients such as "parmesan_cheese", "egg_tagliatelle", and "pecorino_cheese".
- **Recommended_Meals**: The simulator recommends "Black risotto with Modena IGP balsamic vinegar", which fits the medium-calorie requirement and is suitable for the client's dietary needs, and all the others meals that suit the requirements.
- **Menu_Category**: The recommended meal is categorized under "Main Dishes" and "Side Dishes".

3.3.3 *Scenario 2: Omnivore Client with High-Calorie Preference*

In the third scenario (3.8), the inputs are:

- **Clients_allergens**: None (input left empty)
- **Clients_Diet**: "Omnivore"
- **ClientCaloricCategory**: "high"

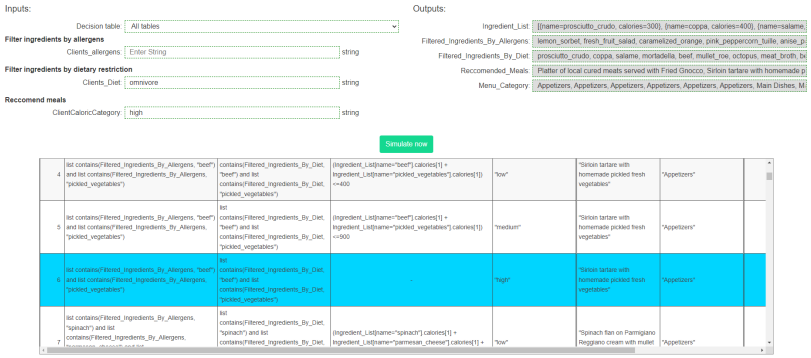

FIGURE 3.8: Example Scenario: Omnivore Client with High-Calorie Preference

Outputs:

- **Filtered_Ingredients_By_Allergens**: Since no allergens were specified, the output includes all ingredients.
- **Filtered_Ingredients_By_Diet**: As the client is omnivorous, the list includes all types of ingredients, including meat.
- **Recommended_Meals**: The simulator recommends "Sirloin tartare with homemade pickled fresh vegetables" as a high-calorie option.
- **Menu_Category**: This meal is categorized under "Appetizers".

These scenarios demonstrate how the Camunda simulator processes the inputs and provides personalized meal recommendations based on the client's dietary needs and caloric preferences, you can download the DMN file called DecisionTablesFinalVersion.dmn on https://github.com/

Meguazy/project_KEBI/tree/main/Part%20One/Decision%20tables, go on the camunda web simulator and test it with different allergens and dietary restrictions.

Part III

PROLOG

4

Prolog

"XXXX"
—XXX

4.1 PROLOG CODE EXPLANATION

After our approach with decision tables, we transitioned into Prolog to further refine our meal recommendation system. While decision tables are highly effective for straightforward decision-making processes, they have problems regarding high complex decision involving several components, while Prolog offers several advantages that make it particularly suited for more complex and dynamic systems.

This section provides a detailed explanation of the Prolog code used to implement a meal recommendation system for the Italian restaurant Nero Balsamico. The system considers the client's dietary restrictions, allergens, and preferred caloric intake to suggest suitable dishes. The approach leverages Prolog's logical inference capabilities to filter and recommend meals based on the client's preferences.

4.1.1 *Defining Ingredients and Their Attributes*

The first part of the code defines a list of ingredients available at Nero Balsamico, along with their associated attributes such as calories, allergens, and dietary restrictions. This information is crucial for filtering ingredients based on client preferences.

Listing 4.1: Defining Ingredients and Their Attributes

```
% Define the facts with ingredient, calories, allergens,  and dietary
    restrictions
ingredient_info(prosciutto_crudo, 250, [], [omnivore]).
ingredient_info(coppa, 300, [], [omnivore]).
ingredient_info(salame, 400, [], [omnivore]).
ingredient_info(mortadella, 350, [lactose], [omnivore]).
ingredient_info(gnocco_fritto, 300, [gluten], [vegan]).
ingredient_info(fried_potatoes, 300, [], [vegan]).
```

In this section, each ingredient is associated with:

- **Calories**: The caloric content of the ingredient.
- **Allergens**: Any allergens present in the ingredient (e.g., lactose, gluten).
- **Dietary Restrictions**: The suitability of the ingredient for different diets, such as **omnivore**, **vegetarian**, or **vegan**.

Example: The ingredient `mortadella` has 350 calories, contains `lactose`, and is suitable for `omnivores`.

4.1.2 *Dish Definition and Meal Assembly*

Dishes are defined by listing their ingredients and categorizing them into meal types, such as appetizers, main dishes, second meals, side dishes, and desserts.

Listing 4.2: Defining Dishes

```
% Appetizers dish facts
dish('Platter␣of␣local␣cured␣meats␣served␣with␣Fried␣Gnocco', appetizer, [
    salame, gnocco_fritto]).
dish('Sirloin␣tartare␣with␣homemade␣pickled␣fresh␣vegetables', appetizer,
    [beef, pickled_vegetables]).
dish('Spinach␣flan␣on␣Parmigiano␣Reggiano␣cream␣with␣mullet␣roe',
    appetizer, [spinach, parmesan_cheese, mullet_roe]).
```

Each dish is associated with:

- **Dish Name**: The name of the dish.
- **Course Type**: The type of course (e.g., appetizer, main dish).
- **Ingredients**: A list of ingredients that make up the dish.

Example: The dish `'Spinach flan on Parmigiano Reggiano cream with mullet roe'` is categorized as an `appetizer` and includes ingredients such as `spinach`, `parmesan_cheese`, and `mullet_roe`.

4.1.3 *Client Preferences and Meal Recommendation Logic*

Client preferences are stored as facts, detailing their preferred caloric category, allergens, and dietary restrictions.

Listing 4.3: Client Preferences and Meal Recommendation Logic

```
% Client preferences fact
client_preferences(client1, low, [lactose], vegetarian).
client_preferences(client2, high, [gluten], vegetarian).
client_preferences(client3, medium, [], vegan).
client_preferences(client4, medium, [egg], omnivore).
```

Client Preferences:

- **Caloric Category**: The client's desired caloric intake level (e.g., low, medium, high).
- **Allergens**: A list of allergens the client is sensitive to.
- **Dietary Restrictions**: The client's dietary preferences, such as **omnivore**, **vegetarian**, or **vegan**.

The Prolog code then uses these preferences to find suitable dishes by matching the client's preferences with the dish ingredients.

Listing 4.4: Finding Suitable Dishes

```
% Find suitable dish for the client based on client preferences
find_suitable_dish(ClientID, DishName, CourseType, [TotalCalories,
    AllAllergens, HighestDietaryRestriction], [ClientCaloriePref,
    ClientAllergens, ClientDietaryRestriction]) :-
    client_preferences(ClientID, ClientCaloriePref, ClientAllergens,
        ClientDietaryRestriction),
```

```
assemble_dish(DishName, CourseType, [TotalCalories, AllAllergens,
    HighestDietaryRestriction], [CalorieCategory]),
matches_calorie_preference(ClientCaloriePref, CalorieCategory),
% Ensure the dish does not contain any allergens the client is
    allergic to
forall(member(Allergen, ClientAllergens), \+ member(Allergen,
    AllAllergens)),
% Ensure the dish matches the client's dietary restrictions
matches_dietary_restriction(ClientDietaryRestriction,
    HighestDietaryRestriction).
```

This predicate works by:

- **Assembling the Dish**: Calculating total calories, determining calorie category, aggregating allergens, and identifying the most restrictive dietary requirement among the dish ingredients.
- **Matching Preferences**: Ensuring that the dish's calorie category, allergens, and dietary restrictions align with the client's preferences.

4.1.4 *Reasoning on Omnivore, Vegetarian, and Vegan*

The Prolog system categorizes ingredients and dishes based on their compatibility with omnivore, vegetarian, and vegan diets. The reasoning process includes the following considerations:

- **Omnivore**: Suitable for all diets; includes all ingredients unless otherwise restricted by allergens.
- **Vegetarian**: Excludes any meat or fish but includes dairy and eggs.
- **Vegan**: Excludes all animal products, including meat, fish, dairy, and eggs.

This categorization allows the system to filter ingredients and recommend dishes that align with the client's dietary restrictions.

4.1.5 *Advantages of the Prolog Approach*

Using Prolog for this task has several advantages:

- **Logical Inference**: Prolog's logical inference allows for flexible and dynamic filtering of ingredients and meals based on varying client preferences.
- **Scalability**: New ingredients, allergens, and dietary restrictions can be added to the knowledge base without needing to alter the core logic.
- **Declarative Nature**: Prolog's declarative syntax makes the rules easy to understand and maintain.

This approach ensures that the meal recommendations are both accurate and personalized, meeting the unique needs of each client at Nero Balsamico.

Part IV

ONTOLOGY AND KNOWLEDGE GRAPHS

5

Ontology & Knowledge Graphs

"XXXX"
—XXX

Part V

AOAME

6

AOAME Implementation

"XXXX"
—XXX

Part VI

EPILOGUE

7

Conclusion

"XXXX"
—XXX

Abbreviations

HTML hypertext markup language

List of Symbols

Latin Letters

l	length

Greek Letters

η	labeling

Superscripts

$\mathcal{G}$	graph

Subscripts

ρ	environment

I would like to thank ...