University of Camerino

School of Science and Technology

Master Degree in Computer Science (Knowledge Engineering and Business Intelligence course)

Knowledge Engineering Project

Candidate

Veronika Moriconi
Francesca Morici

Supervisor

Dr. Knut Hinkelmann

Co-supervisor

Dr. Emanuele Laurenzi

Academic Year 2024/2025

Table of Contents

List of Figures

1. Introduction

In recent years, the digital transformation of the restaurant industry has significantly changed the interaction between customers and menu information. QR-code-based digital menus have become increasingly common, offering a practical and costeffective alternative to traditional printed menus. However, despite their convenience, digital menus often present a major limitation: the difficulty for customers to quickly identify dishes that fit their specific dietary needs. Users affected by allergies, intolerances, dietary restrictions, or caloric preferences may find it challenging to manually filter the available dishes displayed on a small smartphone screen.

Personalizing the menu according to user preferences therefore becomes essential for improving the customer experience. A system capable of automatically analysing dietary constraints and recommending only compatible meals can provide clearer navigation, reduce cognitive load, and increase customer satisfaction. To address this need, this project integrates different knowledge-based technologies, combining rule-based reasoning, ontological modelling, and process-oriented decision logic.

The methodological approach adopted in this work relies on four complementary tools:

- **Trisotec**: used for the modelling of decision logic through Decision Requirement Diagrams (DRD) and decision tables.
- **Prolog**: used to implement a rule-based reasoning system capable of checking compatibility between meals and dietary preferences.
- **Protégé**: adopted for designing the ontology, defining classes, properties, and formal relations between ingredients, meals, and dietary constraints. The ontology is enriched with SWRL rules, SHACL constraints, and SPARQL queries.
- **Jena Fuseki and AOAME**: used to deploy and query the knowledge graph and to support ontology-based BPMN meta-modelling for representing the personalized menu generation process.

The integration of these tools enables the development of a semantically rich, modular, and scalable knowledge-based system capable of dynamically selecting meals that align with a customer's dietary restrictions and caloric preferences. This demonstrates how decision models, rule-based systems, and ontologies can be combined into a unified pipeline for intelligent food recommendation.

1.1 Project Objectives

The main objective of this project is to design and implement an integrated knowledge-based system capable of generating a personalized restaurant menu tailored to the user's dietary preferences. The system must evaluate dietary needs such as allergies, intolerances, vegetarian choices, and caloric limits, and recommend only meals that satisfy these constraints.

To achieve this goal, the project incorporates multiple knowledge engineering tools, each contributing a different aspect of the reasoning process:

- **Decision Modelling with Trisotec**: Decision tables and DRDs are used to represent the logical dependencies involved in meal selection.
- **Prolog-based Reasoning**: A set of declarative rules processes ingredient information, dietary constraints, and caloric levels to infer which meals are suitable for the user.
- **Ontology Development in Protégé**: The domain of ingredients, meals, nutritional values, and user profiles is formally modelled using OWL, complemented by SWRL rules, SHACL validation, and SPARQL queries.
- **Knowledge Graph Querying and BPMN Meta-Modelling**: Jena Fuseki enables SPARQL-based retrieval of customized menus, while AOAME supports the extension of BPMN for representing the menu personalization workflow.

Through the combination of these technologies, the system provides robust, semantically consistent, and automated meal recommendations grounded in explicit domain knowledge.

1.2 Structure of the Personalized Menu

To ensure consistency across all knowledge-based components of the project (Prolog, ontology, SPARQL queries), the personalized menu is structured around a curated set of atomic ingredients. Each ingredient is characterized by properties relevant for dietary filtering: lactose content, gluten content, and vegetarian compatibility. These properties reflect the classifications implemented in the Prolog knowledge base and the corresponding ontology.

Table 1.1 provides an overview of a representative subset of the ingredients included in the knowledge base. Each ingredient is annotated with dietary information extracted from the Prolog file `Prolog-Moriconi-Morici.pl`.

This structured set of ingredients serves as the foundation for all reasoning modules. It enables the system to apply dietary constraints consistently across Prolog rules, ontology reasoning, and SPARQL-based filtering, ultimately supporting the generation of a personalized and coherent menu.

Table 1.1: Overview of Ingredients Used in the Personalized Menu

Ingredient	**Lactose Free**	**Gluten Free**	**Vegetarian**	**Notes**
anchovy	yes	yes	no	Fish-based topping
basil	yes	yes	yes	Herbs, sauces
bechamel	no	yes	yes	Used in lasagna
beef_steak	yes	yes	no	Meat dish
bread	yes	no	yes	Contains gluten
butter	no	yes	yes	Dairy product
chicken_breast	yes	yes	no	Meat ingredient
egg	yes	yes	no	Used in pasta/recipes
eggplant	yes	yes	yes	Vegetarian dishes
flour	yes	no	yes	Gluten ingredient
gorgonzola	no	yes	yes	Dairy cheese
ham	yes	yes	no	Meat topping
mascarpone	no	yes	yes	Dairy ingredient
milk	no	yes	yes	Dairy ingredient
mozzarella	no	yes	yes	Pizza cheese
mushrooms	yes	yes	yes	Pizza, pasta
olive_oil	yes	yes	yes	Condiment
pancetta	yes	yes	no	Carbonara ingredient
parmigiano	no	yes	yes	Dairy cheese
pecorino	no	yes	yes	Dairy cheese
penne	yes	no	yes	Gluten pasta
pesto_sauce	yes	yes	yes	Pasta sauce
prosciutto	yes	yes	no	Meat ingredient
ricotta	no	yes	yes	Dairy ingredient
sausage	yes	yes	no	Meat topping
spaghetti	yes	no	yes	Gluten pasta
tomato	yes	yes	yes	Sauces, base
tomato_sauce	yes	yes	yes	Pizza/pasta base
zucchini	yes	yes	yes	Vegetarian dishes

2. Decision Model and Notation with Trisotech

Decision Model and Notation (DMN) provides a standardized way to model, document, and execute decision logic using a combination of Decision Requirement Diagrams (DRD) and decision tables. In our Personalized Menu system, DMN is used to formalize the decision logic that determines the most suitable meal for a customer based on their dietary restrictions and caloric preferences.

Unlike previous work that employed Camunda for DMN modeling, our project uses **Trisotech Digital Enterprise Suite**, which offers a fully integrated environment for DMN modeling, simulation, validation, and Test Case execution. This allows us to evaluate decision rules directly within the platform and ensure that the logic behaves correctly under different customer profiles.

The decision logic is structured into two main decision tables:

- **Ingredients_Res**: filters compatible ingredients based on dietary preferences (gluten, lactose, vegetarian).
- **Suggested Meal**: identifies a recommended dish based on the compatible ingredients and the customer's caloric threshold.

All model components—decision tables, DRD structure, and test executions—are exported from Trisotech and documented in this chapter.

2.1 Decision Requirements Diagram (DRD)

The DRD defines the dependencies between the input data and the two decision nodes. The main decision, *Suggested Meal*, depends on two inputs:

- **Customer Profile**: may include Vegan, Vegetarian, GlutenFree, LactoseFree, or None.
- **Max Kcal**: the caloric limit that the recommended meal must not exceed.

These inputs feed into the *Ingredients_Res* decision, which determines the set of acceptable ingredients. The result is then used by the *Suggested Meal* decision to propose a suitable dish.

The DRD clearly shows how customer-provided information flows into the decision tables, ensuring transparency and traceability of the meal recommendation process.

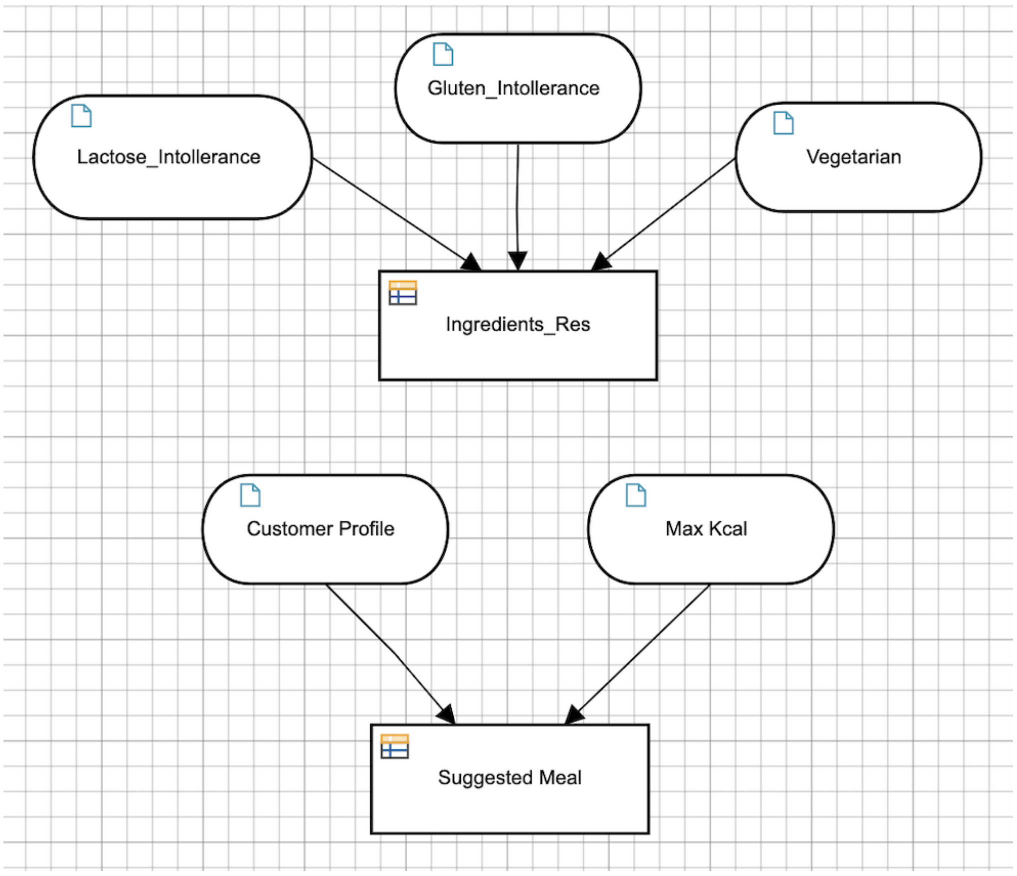


Figure 2.1: Decision Requirements Diagram (DRD) of the DMN model implemented in Trisotech.

2.2 Decision Tables

Decision tables implement the business logic of the model. In our Trisotech implementation, both tables adopt the **COLLECT** hit policy, allowing multiple rules to be satisfied simultaneously.

2.2.1 Ingredients_Res Decision Table

The *Ingredients_Res* decision evaluates each ingredient and determines whether it should be included in the set of compatible ingredients, based on the customer's dietary profile.

Each rule in the table checks three boolean conditions:

- Gluten intolerance
- Lactose intolerance
- Vegetarian preference

If the ingredient meets all required conditions, it is added to the resulting list.

A practical example of executing this table is shown in Figure 2.3, where the Trisotech test console displays the input profile and the generated set of compatible ingredients.

2.2.2 Suggested Meal Decision Table

The *Suggested Meal* decision takes two inputs:

- The filtered list of ingredients produced by *Ingredients_Res*
- The customer's *Max Kcal* threshold

Each rule describes a combination of dietary profile and caloric limit, mapping it to a specific meal suggestion and an optional textual description.

By adopting the **COLLECT** hit policy, the table can return multiple valid meal options, ensuring flexibility when several dishes satisfy the customer's constraints.

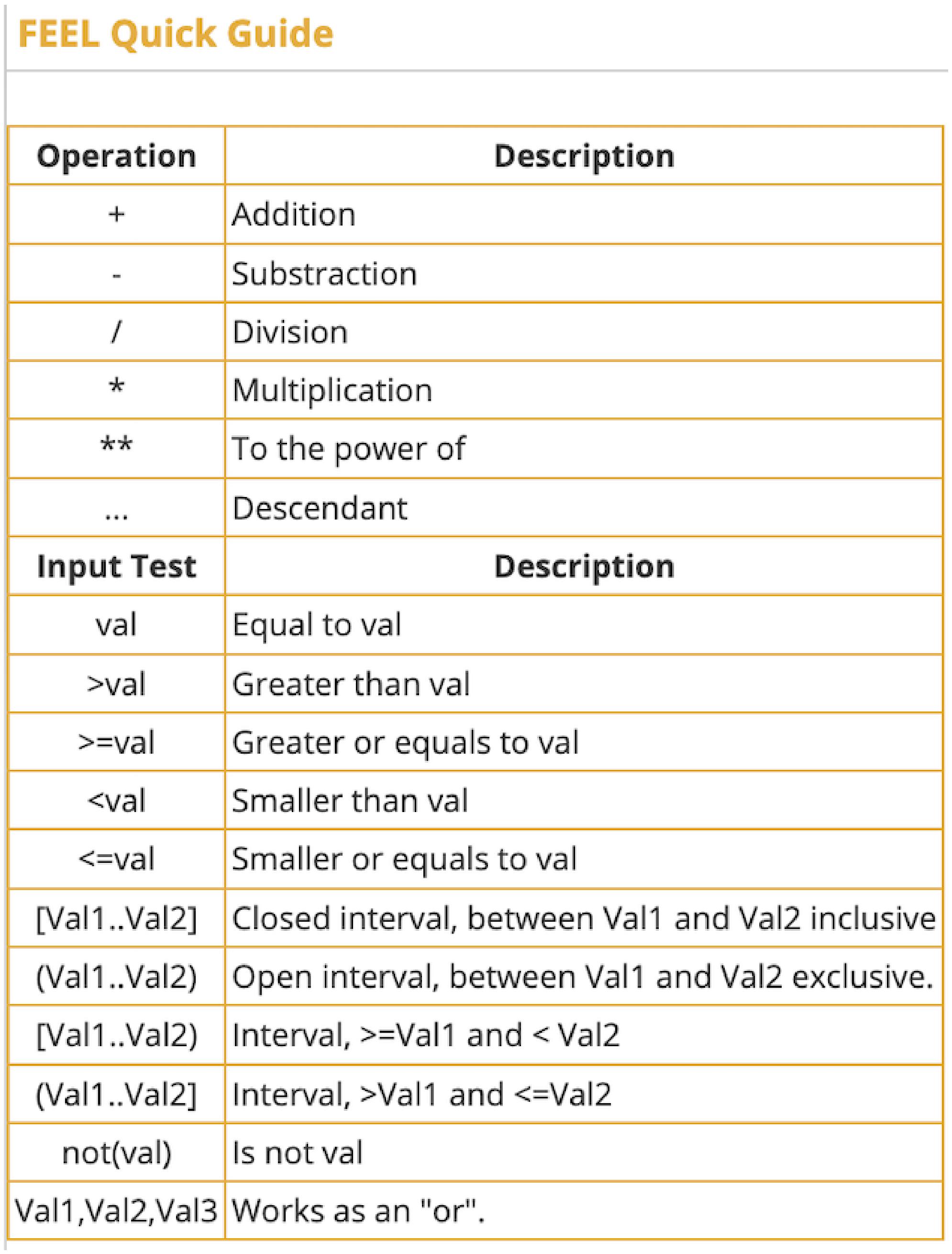
2.3 Model Simulation and Validation in Trisotech

A major advantage of using Trisotech is the availability of integrated **Test Cases** for validating DMN models. We defined a suite of test cases covering different customer scenarios, including Vegan, GlutenFree, and unrestricted users with different caloric limits.

Trisotech evaluates each test case by comparing the actual output of the decision model with the expected outcome, ensuring correctness and consistency.

These tests confirm that the decision logic performs as intended:

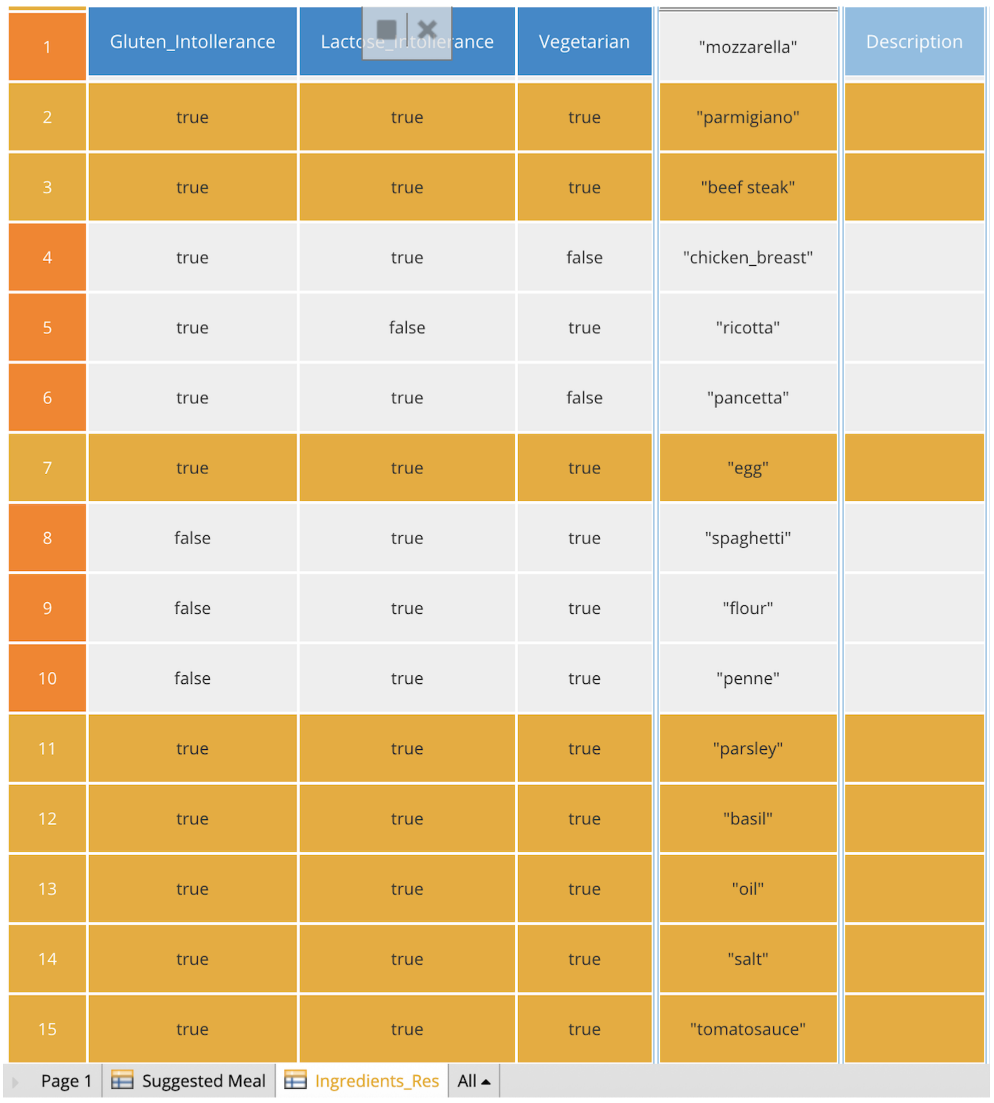
- Vegan profile (800 kcal) $\rightarrow$ Vegan Lasagna
- GlutenFree profile (600 kcal) $\rightarrow$ Grilled Chicken
- GlutenFree profile (900 kcal) $\rightarrow$ Beef Steak
- No restrictions (500 kcal) $\rightarrow$ Margherita Pizza



FEEL Quick Guide

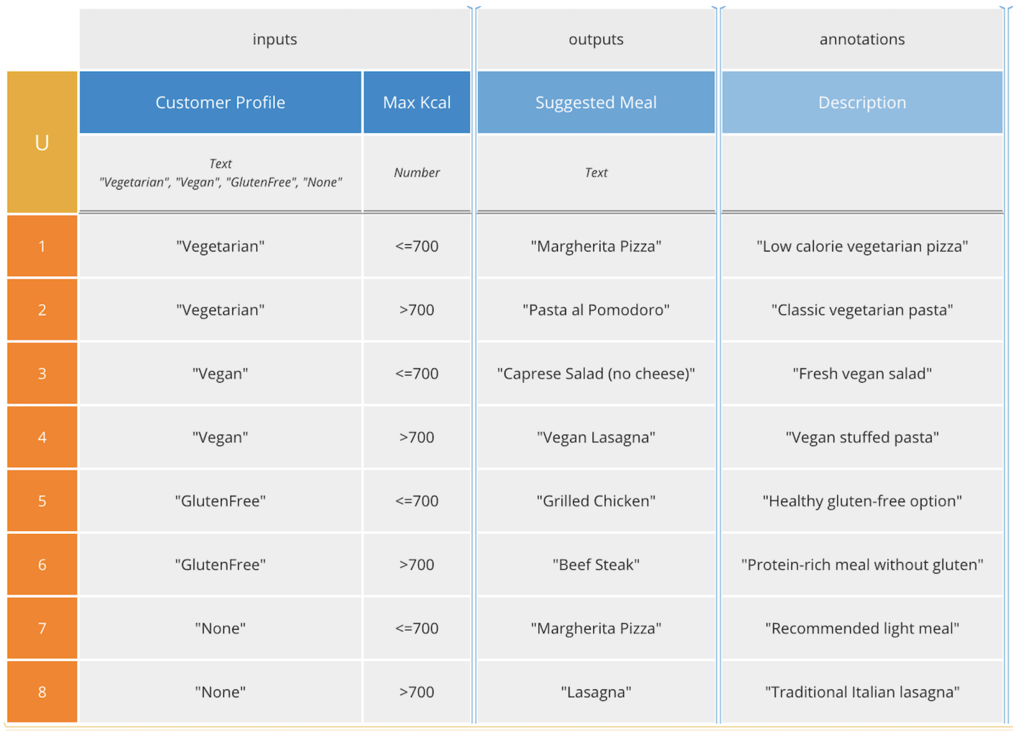
Operation	Description
+	Addition
-	Substraction
/	Division
*	Multiplication
**	To the power of
...	Descendant
Input Test	**Description**
val	Equal to val
>val	Greater than val
>=val	Greater or equals to val
<val	Smaller than val
<=val	Smaller or equals to val
[Val1..Val2]	Closed interval, between Val1 and Val2 inclusive
(Val1..Val2)	Open interval, between Val1 and Val2 exclusive.
[Val1..Val2)	Interval, >=Val1 and < Val2
(Val1..Val2]	Interval, >Val1 and <=Val2
not(val)	Is not val
Val1,Val2,Val3	Works as an "or".

Figure 2.2: Decision table *Ingredients_Res* used to filter ingredients based on the customer's dietary preferences.



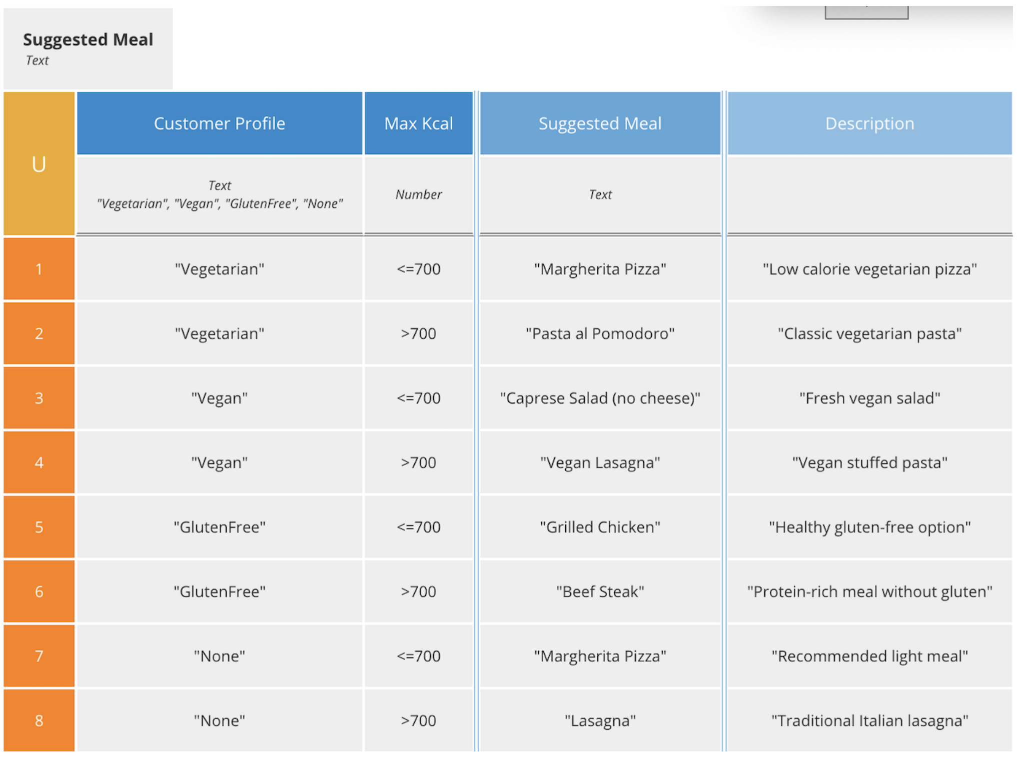
1	Gluten_Intollerance	Lactose_Intollerance	Vegetarian	"mozzarella"	Description
2	true	true	true	"parmigiano"	
3	true	true	true	"beef steak"	
4	true	true	false	"chicken_breast"	
5	true	false	true	"ricotta"	
6	true	true	false	"pancetta"	
7	true	true	true	"egg"	
8	false	true	true	"spaghetti"	
9	false	true	true	"flour"	
10	false	true	true	"penne"	
11	true	true	true	"parsley"	
12	true	true	true	"basil"	
13	true	true	true	"oil"	
14	true	true	true	"salt"	
15	true	true	true	"tomatosauce"	

Figure 2.3: Test execution of the *Ingredients_Res* decision in Trisotech, showing the resulting filtered ingredient list.



U	inputs		outputs	annotations
	Customer Profile	Max Kcal	Suggested Meal	Description
	Text "Vegetarian", "Vegan", "GlutenFree", "None"	*Number*	*Text*	
1	"Vegetarian"	<=700	"Margherita Pizza"	"Low calorie vegetarian pizza"
2	"Vegetarian"	>700	"Pasta al Pomodoro"	"Classic vegetarian pasta"
3	"Vegan"	<=700	"Caprese Salad (no cheese)"	"Fresh vegan salad"
4	"Vegan"	>700	"Vegan Lasagna"	"Vegan stuffed pasta"
5	"GlutenFree"	<=700	"Grilled Chicken"	"Healthy gluten-free option"
6	"GlutenFree"	>700	"Beef Steak"	"Protein-rich meal without gluten"
7	"None"	<=700	"Margherita Pizza"	"Recommended light meal"
8	"None"	>700	"Lasagna"	"Traditional Italian lasagna"

Figure 2.4: Decision table *Suggested Meal*, which defines the recommended dish for each combination of dietary restrictions and caloric limit.



Suggested Meal
Text

U	Customer Profile	Max Kcal	Suggested Meal	Description
	Text "Vegetarian", "Vegan", "GlutenFree", "None"	*Number*	*Text*	
1	"Vegetarian"	<=700	"Margherita Pizza"	"Low calorie vegetarian pizza"
2	"Vegetarian"	>700	"Pasta al Pomodoro"	"Classic vegetarian pasta"
3	"Vegan"	<=700	"Caprese Salad (no cheese)"	"Fresh vegan salad"
4	"Vegan"	>700	"Vegan Lasagna"	"Vegan stuffed pasta"
5	"GlutenFree"	<=700	"Grilled Chicken"	"Healthy gluten-free option"
6	"GlutenFree"	>700	"Beef Steak"	"Protein-rich meal without gluten"
7	"None"	<=700	"Margherita Pizza"	"Recommended light meal"
8	"None"	>700	"Lasagna"	"Traditional Italian lasagna"

Figure 2.5: Trisotech Test Cases panel for the *Suggested Meal* decision, showing multiple validated scenarios.

- No restrictions (1000 kcal) → Lasagna

The execution traces generated by Trisotech confirm that the model consistently activates the correct rules for each scenario.

2.4 Conclusion

The DMN model implemented in Trisotech provides a robust, transparent, and maintainable structure for automating the personalized meal recommendation process.

Compared to Camunda-based implementations, Trisotech offers:

- Native simulation and step-by-step decision tracing
- Built-in support for DMN TCK-compliant test cases
- Clear visualization of rule activation and DRD flow

These features significantly enhance the model's reliability and interpretability. The resulting system is flexible, scalable, and fully aligned with modern standards for automated decision-making.

3. Prolog

Prolog was employed in this project as a rule-based reasoning engine capable of evaluating dietary constraints and generating meal recommendations according to user preferences. Its declarative nature makes it particularly suitable for representing domain knowledge through logical relations, enabling automated inference based on ingredients, dietary restrictions, and caloric levels.

The Prolog knowledge base developed for this project is contained in the file `Prolog-Moriconi-Morici.pl`.
It is organized into four main components: (1) ingredient classification, (2) meal representation, (3) rules for dietary compliance, and (4) recommendation rules. This section provides an overview of these components and describes how they interact to produce personalized menus.

3.0.1 Ingredient Classification

The knowledge base begins by defining all atomic ingredients through the predicate:

```
ingredient(X).
```

Ingredients are then annotated with dietary properties that indicate whether they contain lactose, gluten, or are incompatible with vegetarian or vegan diets. These properties are represented as unary predicates, for example:

```
lactose(milk).
gluten(flour).
incompatible_vegetarian(anchovy).
incompatible_vegan(egg).
```

From these predicates, the system derives positive dietary properties through negation-as-failure:

```
is_lactose_free(I) :- \+ lactose(I).
is_gluten_free(I) :- \+ gluten(I).
is_vegetarian(I) :- \+ incompatible_vegetarian(I).
is_vegan(I) :- \+ incompatible_vegan(I).
```

This structure mirrors the reasoning strategy used in the ontology and in the decision tables, ensuring coherence across all knowledge-based components.

3.0.2 Meal Representation

Each meal is defined using the predicate:

```
meal(Name, Ingredients, CaloriesLevel).
```

An example extracted from the knowledge base is:

```
meal(margherita_pizza,
     [mozzarella, tomato_sauce, basil, olive_oil],
     medium).
```

The list of ingredients allows the system to evaluate dietary eligibility for each meal by checking whether any ingredient violates the user's constraints. The caloric level (`low`, `medium`, or `high`) is used to filter meals according to the user's nutritional preferences.

3.0.3 Dietary Compliance Rules

Rules are defined to check whether a meal is compatible with specific dietary requirements. For example, a lactose-free meal is defined as one that contains no lactose ingredients:

```
meal_lactose_free(Meal) :-
    meal(Meal, Ingredients, _),
    \+ (member(I, Ingredients), lactose(I)).
```

Similar rules are implemented for gluten-free, vegetarian, and vegan meals:

```
meal_gluten_free(Meal) :-
    meal(Meal, Ingredients, _),
    \+ (member(I, Ingredients), gluten(I)).

meal_vegetarian(Meal) :-
    meal(Meal, Ingredients, _),
    \+ (member(I, Ingredients), incompatible_vegetarian(I)).

meal_vegan(Meal) :-
    meal(Meal, Ingredients, _),
    \+ (member(I, Ingredients), incompatible_vegan(I)).
```

This structure ensures alignment with the ontology restrictions (e.g., SHACL shapes and SWRL rules), in which meals must not include forbidden ingredient classes.

3.0.4 Calorie-Based Reasoning

Calories are handled using a simple hierarchical ordering:

```
calorie_order(low, 1).
calorie_order(medium, 2).
calorie_order(high, 3).
```

The rule for selecting meals by caloric level is:

```
meal_by_calories(RequiredLevel, Meal) :-
    meal(Meal, _, Level),
    calorie_order(Level, N1),
    calorie_order(RequiredLevel, N2),
    N1 =< N2.
```

This means that requesting a `medium` meal also returns all `low` meals, while requesting `high` returns every meal in the knowledge base.

3.0.5 Checking User Preferences

User preferences are represented as a structured list, for example:

```
[lactose_free, vegetarian, medium_calories]
```

These preferences are evaluated through the predicate:

```
check_preferences(Meal, Preferences).
```

Each preference triggers a dedicated rule, e.g.:

```
check_lactose(Meal, lactose_free) :- meal_lactose_free(Meal).
check_gluten(Meal, gluten_free) :- meal_gluten_free(Meal).
check_vegetarian(Meal, vegetarian) :- meal_vegetarian(Meal).
```

Preferences not present in the list are automatically skipped, implementing an opt-out mechanism.

3.0.6 Final Recommendation Rule

The main predicate that produces the personalized menu is:

```
find_meals(Preferences, CalorieLevel, Meals).
```

This rule performs the following steps:

1. Enumerates all meals in the knowledge base.
2. Checks that each meal satisfies the user's dietary preferences.
3. Ensures compatibility with the requested caloric level.
4. Aggregates all valid meals into a final list using `findall/3`.

This predicate represents the core of the recommendation engine and ties together all elements of the knowledge base, providing an automated and logically coherent way to generate personalized menus.

4. Ontologies

The ontology developed in this project formalizes the domain of ingredients, meals, nutritional values, and dietary restrictions. Implemented in OWL using Protégé, it provides a structured semantic model that supports automated reasoning, constraint validation, and SPARQL-based retrieval. The inferred ontology graph (`MenuOntology-Moriconi-Morici-Inferred.ttl`) includes the full classification of meals and ingredients, enriched through SWRL rules and validated via SHACL constraints.

The ontology is organized into three main conceptual layers: (1) classes and taxonomies, (2) object and data properties, and (3) individuals representing concrete meals and ingredients.

4.0.1 Classes and Structure

The core of the ontology is the definition of the class `ex:Meal`, representing all dishes available in the personalized menu. Each meal is connected to its components through the object property:

`ex:hasIngredient`

Ingredients are organized into semantic categories relevant to dietary restrictions, including:

- **ex:Meat**
- **ex:Fish**
- **ex:DairyIngredient**
- **ex:GlutenIngredient**
- **ex:LactoseIngredient**
- **ex:EggProduct**

These categories are used extensively in both SWRL rules and SHACL validations. Each meal also has a caloric value expressed through:

- `ex:hasCalories` (datatype property, integer)
- `ex:hasCaloriesLevel` (object property with values `ex:Low`, `ex:Medium`, `ex:High`)

The inferred ontology shows (i) classification of ingredients into their dietary-relevant categories and (ii) automatic assignment of calorie levels based on numeric thresholds.

4.0.2 SWRL Rules

SWRL rules were used to extend the ontology with additional reasoning capabilities. The full rule set is defined in the file `SWRL-Rules-Moriconi-Morici.txt`, and includes:

1. **Calorie classification rules** Assigning `High`, `Medium`, or `Low` caloric levels based on numeric thresholds:

   ```
   Meal(?m) ^ hasCalories(?m, ?c) ^ swrlb:greaterThan(?c, 800)
     -> hasCaloriesLevel(?m, High)
   ```

2. **Intolerance-based restriction rules** For example, meals containing gluten ingredients are inferred as not recommended for gluten-intolerant guests:

   ```
   Meal(?m) ^ hasIngredient(?m, ?i) ^ GlutenIngredient(?i)
     -> notRecommendedFor(?m, guestNoGluten)
   ```

3. **Vegetarian and vegan restriction rules** Meat, dairy, fish, egg products automatically mark meals as not recommended:

   ```
   Meal(?m) ^ hasIngredient(?m, ?i) ^ Meat(?i)
     -> notRecommendedFor(?m, guestVegan)
   ```

4. **Combined reasoning rules** For example, low-calorie meals containing gluten are explicitly linked to `guestNoGluten`.

These rules enrich the ontology with implicit knowledge, which becomes explicit in the inferred file, making it possible to query dietary recommendations using SPARQL.

4.0.3 SPARQL Queries

A comprehensive set of SPARQL queries was implemented to extract knowledge from the ontology. The file `SPARQL-Queries-Moriconi-Morici.txt` includes more than 20 queries covering various scenarios.

Representative examples include:

Listing all meals

```
SELECT ?meal ?label WHERE {
  ?meal rdf:type ex:Meal .
  OPTIONAL { ?meal rdfs:label ?label }
}
```

Vegan or vegetarian meals

```
SELECT ?meal WHERE {
  ?meal rdf:type ex:Meal .
  FILTER NOT EXISTS { ?meal ex:hasIngredient ?i . ?i rdf:type ex:Meat }
  FILTER NOT EXISTS { ?meal ex:hasIngredient ?i . ?i rdf:type ex:Fish }
  FILTER NOT EXISTS { ?meal ex:hasIngredient ?i . ?i rdf:type ex:DairyIngredie
}
```

Meals inferred as not recommended

```
SELECT ?meal ?label WHERE {
  ?meal ex:notRecommendedFor ex:guestVegan .
  OPTIONAL { ?meal rdfs:label ?label }
}
```

High-calorie meals

```
SELECT ?meal WHERE {
  ?meal ex:hasCaloriesLevel ex:High .
}
```

These queries allow the ontology to serve as a reasoning engine for meal recommendations, dietary filtering, and knowledge extraction.

4.0.4 SHACL Shapes

To ensure structural and semantic consistency of the ontology, a complete SHACL validation model was developed (`SHACL_SHAPES-Moriconi-Morici.txt`). The SHACL shapes address three main validation categories:

1. **Data quality constraints** For example, calories must be non-negative integers:

   ```
   sh:datatype xsd:integer ;
   sh:minInclusive 0 ;
   ```

2. **Dietary compliance shapes** Including vegan, vegetarian, gluten-free, and lactose-free checks:

   ```
   SELECT $this WHERE {
     $this ex:hasIngredient ?i .
     ?i rdf:type ex:DairyIngredient .
   }
   ```

3. **Calories consistency checks** Ensuring alignment between numeric calories and declared level:

   ```
   FILTER(xsd:integer(?c) <= 800)
   ```

Each shape ensures that asserted and inferred meal classifications are aligned with domain rules.

4.0.5 SHACL Validation Report

The report (`REPORT_SHACL-with-violation.txt`) confirms that the ontology does **not** fully conform to the SHACL constraints, as expected. Violations include:

- Vegan meals containing dairy, meat, fish, or eggs

- Vegetarian meals containing meat or fish
- Gluten-free meals containing `ex:GlutenIngredient`
- Lactose-free meals containing `ex:LactoseIngredient`
- Inconsistencies between calorie levels and numeric calorie values

Examples from the report include:

- `ex:QuattroFormaggiPizza` violating vegan rules
- `ex:CarbonaraPasta` violating gluten-free and lactose-free rules
- `ex:GrilledChicken` failing calorie-level consistency

These violations are intentional and reflect the ontology's purpose: SHACL ensures that meals are correctly classified and that inferred dietary restrictions match structural constraints.

4.0.6 Summary

The ontology provides a complete semantic backbone for the project, integrating explicit modelling, SWRL reasoning, validation via SHACL, and knowledge extraction through SPARQL. Together, these components enable a robust, interpretable, and semantically consistent representation of the personalized menu domain.

4.1 Ontology Simulation

This section illustrates how the ontology behaves when queried, validated, and enriched through automated reasoning. Simulations were performed using the inferred ontology file `MenuOntology-Moriconi-Morici-Inferred.ttl`, the SHACL validation rules, and a set of SPARQL queries deployed on Jena Fuseki. The goal of this simulation is to demonstrate that the semantic model correctly supports dietary classification, caloric reasoning, and constraint checking.

4.1.1 SWRL Rule Inference

After applying the SWRL rule set, several new assertions are inferred in the ontology. These inferences appear in the *inferred* version of the ontology and confirm that the rule base is behaving as expected.

Calorie Level Classification

The calorie thresholds defined in SWRL correctly assign calorie levels to meals. For instance:

- Meals with more than 800 calories (e.g., `ex:PestoPasta`) are inferred as `ex:High`.
- Meals between 600 and 800 calories (e.g., `ex:GrilledChicken`) are inferred as `ex:Medium`.

- Meals below 600 calories (e.g., `ex:CapreseSalad`) are inferred as `ex:Low`.

These inferences match the numeric values defined in the ontology and ensure consistency across SPARQL and validation tasks.

Dietary Restriction Inference

SWRL also infers incompatibilities between meals and specific guest profiles.

Examples include:

- **Vegan Guest:** Meals containing dairy, meat, fish, or egg products receive: `ex:notRecommendedFor ex:guestVegan` (e.g., `ex:QuattroFormaggiPizza`, `ex:CarbonaraPasta`).
- **Vegetarian Guest:** Meals containing meat or fish are marked as: `ex:notRecommendedFor ex:guestVegetarian`.
- **Gluten Intolerance:** Meals containing a `ex:GlutenIngredient` (e.g., pasta dishes) are associated with: `ex:notRecommendedFor ex:guestNoGluten`.
- **Lactose Intolerance:** Meals containing `ex:LactoseIngredient` are similarly linked to: `ex:guestNoLactose`.

These inferred links are crucial for the final decision layer of the system, allowing meal recommendations to be computed directly from the knowledge graph.

4.1.2 SPARQL Query Simulation

Multiple SPARQL queries were executed against the ontology using Jena Fuseki. This subsection presents representative examples and the expected results.

Query 1 — Vegan Meals

```
SELECT ?meal ?label WHERE {
  ?meal rdf:type ex:Meal .
  FILTER NOT EXISTS { ?meal ex:hasIngredient ?i . ?i rdf:type ex:Meat }
  FILTER NOT EXISTS { ?meal ex:hasIngredient ?i . ?i rdf:type ex:Fish }
  FILTER NOT EXISTS { ?meal ex:hasIngredient ?i . ?i rdf:type ex:DairyIngred
  FILTER NOT EXISTS { ?meal ex:hasIngredient ?i . ?i rdf:type ex:EggProduct
  OPTIONAL { ?meal rdfs:label ?label }
}
```

Expected result: Only meals made exclusively of plant-based ingredients, such as:

- `ex:MarinaraPizza`
- `ex:ArrabbiataPasta` (if no dairy)

Query 2 — Meals Not Recommended for Vegan Guests

```
SELECT ?meal ?label WHERE {
  ?meal ex:notRecommendedFor ex:guestVegan .
  OPTIONAL { ?meal rdfs:label ?label }
}
```

Expected result (based on SWRL inference):

- `ex:QuattroFormaggiPizza`
- `ex:CarbonaraPasta`
- `ex:Lasagna`
- `ex:BeefSteakDish`
- others containing dairy, meat, fish, or eggs

Query 3 — Gluten-Free Meals

```
FILTER NOT EXISTS {
  ?meal ex:hasIngredient ?i .
  ?i rdf:type ex:GlutenIngredient .
}
```

Expected result: Dishes not containing pasta, flour, bread, or other gluten ingredients.

Query 4 — High-Calorie Meals

```
SELECT ?meal WHERE {
  ?meal ex:hasCaloriesLevel ex:High .
}
```

Expected result:

- `ex:PestoPasta`
- `ex:QuattroFormaggiPizza`
- other meals inferred over 800 kcal

4.1.3 SHACL Validation Simulation

The SHACL shapes were applied to validate the ontology against structural and logical constraints.

The validation report (`REPORT_SHACL-with-violation.txt`) indicates:

- **The ontology does not conform** to all SHACL shapes (as expected).

Main Violations

- **Vegan violations:** meals containing dairy or meat flagged by `ex:VeganMealShape`.
- **Vegetarian violations:** meals including `ex:Meat` or `ex:Fish`.
- **Gluten violations:** meals including pasta or flour flagged by `ex:GlutenFreeMealShape`.
- **Lactose violations:** meals containing cheese or milk flagged by `ex:LactoseFreeMealShape`.
- **Calorie-level inconsistencies:** meals whose numeric calories do not match their declared level.

Example Violations

- `ex:QuattroFormaggiPizza` violates vegan and lactose-free constraints.
- `ex:Lasagna` violates gluten-free, lactose-free, and vegan constraints.
- `ex:CarbonaraPasta` violates gluten, lactose, vegetarian, and vegan shapes.
- `ex:GrilledChicken` marked with incorrect calorie level in the ontology.

The presence of these violations confirms that the SHACL shapes are correctly identifying inconsistencies arising from the data and inference layers.

4.1.4 Simulation Summary

The ontology simulation demonstrates the integration of semantic reasoning, constraint validation, and SPARQL querying. The inferred ontology accurately reflects SWRL-based logic, SHACL shapes successfully detect inconsistencies, and SPARQL queries retrieve meaningful dietary classifications.

This confirms that the ontology is functioning as the semantic backbone of the personalized menu system and supports coherent interaction with Prolog rules, decision models, and the process layer implemented in AOAME.

5. Agile and Ontology-based Meta-modelling with AOAME and Jena Fuseki

In addition to DMN, Prolog, and the ontology-based knowledge graph, our solution also exploits **Agile and Ontology-based Meta-modelling (AOAME)** to bridge business process models with the semantic layer of the system. AOAME enables us to extend standard BPMN 2.0 elements with domain-specific attributes that are directly linked to the ontology and to executable SPARQL queries. In this chapter we describe how AOAME has been used to:

- customise a BPMN process for personalised menu recommendation, and
- automatically query the ontology via Jena Fuseki, using the data stored in the extended BPMN model.

5.1 AOAME Overview

AOAME is designed to support the creation and management of Enterprise Knowledge Graph schemas and their connection to business process models. It provides meta-modelling operators that:

- allow the extension of existing modelling languages (such as BPMN 2.0) with domain-specific properties,
- automatically generate SPARQL queries to keep the triplestore and the process model consistent,
- and support agile evolution of the schema when new requirements emerge.

In our context, AOAME plays a key role in customising the BPMN model of the personalised menu process. The central BPMN task *Construct Personalised Menu* is extended with additional data properties that represent the customer preferences collected at runtime. These properties are then reused in the SPARQL query that retrieves the recommended meals from the ontology.

5.2 BPMN Model Solution with AOAME

The BPMN flow models the end-to-end interaction between the customer and the personalised menu system. The main goal of the process is to:

collect customer preferences (allergies, dietary restrictions, calorie preferences) and generate a customised menu that is sent back to the user.

The process is structured into two lanes:

- **Client lane**: contains the activities performed directly by the customer.
- **Menu recommendation system lane**: contains the automated tasks that receive preferences, compute the recommended meals, and deliver the result.

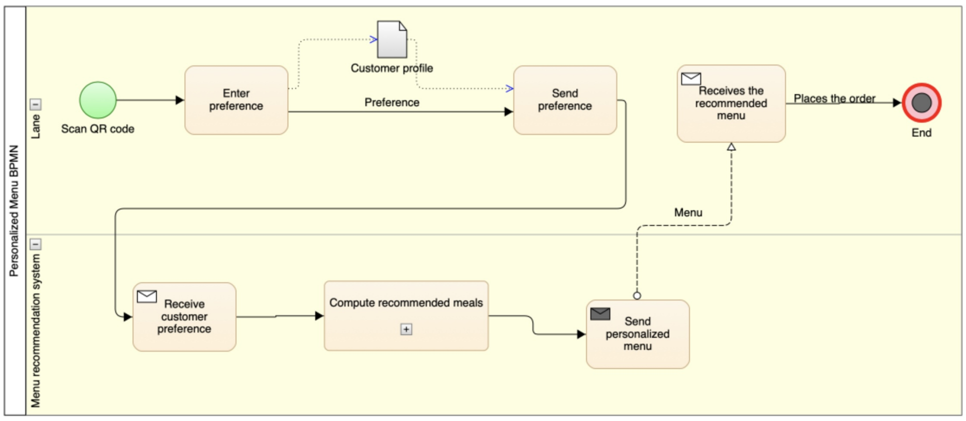


Figure 5.1: BPMN process for personalised menu generation, extended with AOAME.

The main stages of the process are:

Scan QR Code The process starts when the customer scans a QR code placed on the restaurant table or menu. This action identifies the session and opens the digital interface.

Enter Preferences The customer specifies food preferences such as: lactose or gluten intolerance, vegetarian or vegan diet, and a preferred calorie level (e.g. low, medium, high).

Send Preferences Once completed, the preferences are sent to the backend system. This marks the end of the interactive part on the client side and triggers the automated processing.

Receive Customer Preference The personalised menu system receives and validates the preference data.

Compute Recommended Meals This is the core task of the process. In AOAME, the corresponding BPMN task *Construct Personalised Menu* is modelled as an *extended* class of the standard BPMN `Task`. The extension introduces several data-type properties:

- `isGlutenIntolerant` (boolean),
- `isLactoseIntolerant` (boolean),
- `isVegetarian` (boolean),

- `isVegan` (boolean),
- `preferenceCaloriesLevel` (string, with values `"Low"`, `"Medium"`, or `"High"`).

An instance of this extended task (`mod:constructMenuInstance1`) stores the concrete preferences for a given customer. AOAME ensures that these properties are mapped to RDF triples in the triplestore, making them directly accessible to SPARQL queries.

Send Personalised Menu / Receive Personalised Menu After the recommendations are computed, the personalised menu is sent back to the customer, who views it on their device. The process terminates once the menu has been received and the customer can place an order.

By enriching the BPMN task with semantic properties, AOAME creates a tight coupling between the business process and the ontology: the process does not simply call a black-box service, but explicitly exposes the preferences that are later reused in the query logic.

5.3 Jena Fuseki Integration

To execute semantic queries over the ontology, we use **Apache Jena Fuseki** as SPARQL endpoint. Fuseki hosts the RDF dataset that contains:

- the domain ontology (ingredients, meals, calorie levels),
- the instances imported from the DMN and ontology work,
- and the AOAME-generated triples that represent the extended BPMN task instance.

Fuseki provides a web-based interface where SPARQL queries can be written, executed and inspected. In our project, AOAME generates a SPARQL query that reuses both the ontology and the AOAME-specific properties of the *Construct Personalised Menu* task. The query is then run in Fuseki to obtain a list of meals that satisfy all customer constraints.

5.3.1 SPARQL Query Based on AOAME Properties

The core idea behind the query is as follows:

1. Read the customer preferences that AOAME stored on the extended BPMN task instance (`mod:constructMenuInstance1`).
2. Retrieve all meals and their calorie level from the ontology.
3. Filter out meals that violate any active intolerance or diet restriction (gluten, lactose, vegetarian, vegan).
4. Apply the calorie-level preference, if specified.

The actual SPARQL query used in our implementation is shown below.[1]

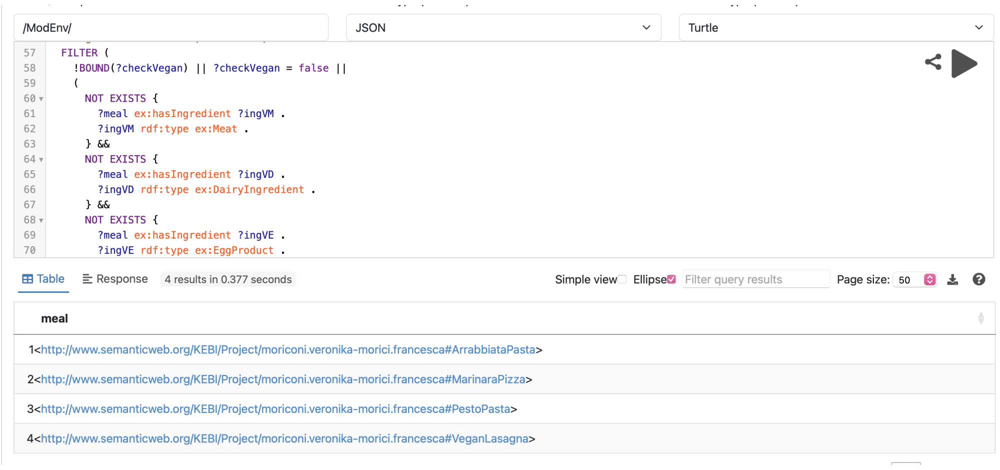
Codice 5.1: SPARQL query that reads AOAME preferences and returns compatible meals.

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex:   <http://www.semanticweb.org/KEBI/Project/moriconi
  .veronika-morici.francesca#>
PREFIX mod:  <http://fhnw.ch/moriconi-morici/model#>

SELECT DISTINCT ?meal WHERE {

  # ==== 1) I read the preferences from the AOAME task ====
  OPTIONAL { mod:constructMenuInstance1 mod:isGlutenIntolerant
          ?checkGluten . }
  OPTIONAL { mod:constructMenuInstance1 mod:
    isLactoseIntolerant      ?checkLactose . }
  OPTIONAL { mod:constructMenuInstance1 mod:isVegetarian
                 ?checkVegetarian . }
  OPTIONAL { mod:constructMenuInstance1 mod:isVegan
                     ?checkVegan . }
  OPTIONAL { mod:constructMenuInstance1 mod:
    preferenceCaloriesLevel ?prefCal . }

  # ==== 2) I take all the meals and their calorie level ====
  ?meal rdf:type ex:Meal .
  OPTIONAL { ?meal ex:hasCaloriesLevel ?mealCalLevel . }

  # ---- I normalize calorie preference (string or individual)
    ----
  BIND(
    IF(?prefCal = ex:Low      || ?prefCal = "Low",      "Low",
      IF(?prefCal = ex:Medium || ?prefCal = "Medium", "Medium"
        ,
        IF(?prefCal = ex:High || ?prefCal = "High", "High", ""
          )
      )
    ) AS ?caloriesLevel
  )

  # ==== 3) Constraints on intolerances / diet ====

  # no gluten if the customer is intolerant
  FILTER (
    !BOUND(?checkGluten) || ?checkGluten = false ||
    NOT EXISTS {
      ?meal ex:hasIngredient ?ingG .
      ?ingG rdf:type ex:GlutenIngredient .
```

[1]The code assumes that the ontology defines classes such as `ex:GlutenIngredient`, `ex:LactoseIngredient`, `ex:Meat`, `ex:DairyIngredient`, and `ex:EggProduct`, and that meals are linked to ingredients via the property `ex:hasIngredient`.

```
    }
  )

  # no lactose if the customer is lactose intolerant
  FILTER (
    !BOUND(?checkLactose) || ?checkLactose = false ||
    NOT EXISTS {
      ?meal ex:hasIngredient ?ingL .
      ?ingL rdf:type ex:LactoseIngredient .
    }
  )

  # vegetarian: no meat
  FILTER (
    !BOUND(?checkVegetarian) || ?checkVegetarian = false ||
    NOT EXISTS {
      ?meal ex:hasIngredient ?ingM .
      ?ingM rdf:type ex:Meat .
    }
  )

  # vegan: no meat, dairy, eggs
  FILTER (
    !BOUND(?checkVegan) || ?checkVegan = false ||
    (
      NOT EXISTS {
        ?meal ex:hasIngredient ?ingVM .
        ?ingVM rdf:type ex:Meat .
      } &&
      NOT EXISTS {
        ?meal ex:hasIngredient ?ingVD .
        ?ingVD rdf:type ex:DairyIngredient .
      } &&
      NOT EXISTS {
        ?meal ex:hasIngredient ?ingVE .
        ?ingVE rdf:type ex:EggProduct .
      }
    )
  )

  # ==== 4) Calorie level restriction ====
  # we use the normalized string "Low/Medium/High"
  # + ex:Low, ex:Medium, ex:High

  FILTER (
      !BOUND(?caloriesLevel)
      || ?caloriesLevel = ""
      || ?caloriesLevel = "High"
      || ( ?caloriesLevel = "Low"     && ?mealCalLevel = ex:Low
            )
        || ( ?caloriesLevel = "Medium" && ?mealCalLevel = ex:
          Medium )
    )
}
```

Figure 5.2 shows the execution of this query in Jena Fuseki. The upper part of the screenshot displays the query itself, while the bottom part lists the meals that satisfy all the active constraints (for example, vegan meals without meat, dairy or egg products).



```
FILTER (
  !BOUND(?checkVegan) || ?checkVegan = false ||
  (
    NOT EXISTS {
      ?meal ex:hasIngredient ?ingVM .
      ?ingVM rdf:type ex:Meat .
    } &&
    NOT EXISTS {
      ?meal ex:hasIngredient ?ingVD .
      ?ingVD rdf:type ex:DairyIngredient .
    } &&
    NOT EXISTS {
      ?meal ex:hasIngredient ?ingVE .
      ?ingVE rdf:type ex:EggProduct .
```

4 results in 0.377 seconds

	meal
1	<http://www.semanticweb.org/KEBI/Project/moriconi.veronika-morici.francesca#ArrabbiataPasta>
2	<http://www.semanticweb.org/KEBI/Project/moriconi.veronika-morici.francesca#MarinaraPizza>
3	<http://www.semanticweb.org/KEBI/Project/moriconi.veronika-morici.francesca#PestoPasta>
4	<http://www.semanticweb.org/KEBI/Project/moriconi.veronika-morici.francesca#VeganLasagna>

Figure 5.2: Execution of the AOAME-generated SPARQL query in Jena Fuseki, returning meals compatible with the customer preferences.

5.4 Discussion

The integration of AOAME and Jena Fuseki adds an agile, meta-model-driven layer on top of the ontology and BPMN model:

- **Semantic enrichment of BPMN:** by extending BPMN tasks with domain-specific attributes, the process model becomes self-descriptive and tightly connected to the ontology.
- **Consistent data flow:** the same preference values entered by the customer in the BPMN process are propagated to the RDF graph and reused in SPARQL queries, eliminating duplication and reducing the risk of inconsistencies.
- **Agile evolution:** if new dietary constraints or preference types are introduced, AOAME allows the BPMN meta-model and the corresponding SPARQL queries to be adapted with limited effort.

Overall, AOAME and Jena Fuseki provide a flexible and robust framework for connecting process models with semantic data, ensuring that the personalised menu recommendations remain both explainable and adaptable to changing requirements.