



University of Camerino

SCHOOL OF SCIENCES AND TECHNOLOGIES

Master degree in Computer Science (LM-18)

Wine Suggestion System

KEBI - Project

Group Members

Filippo Lampa - 124079

Francesco Moschella - 122435

Supervisors

Prof. Dr. Knut Hinkelmann

Prof. Dr. Holger Wache

A.A. 2022/2023

Contents

1	Introduction	5
1.1	Project Description	5
1.2	Project Tasks	6
2	System Design	9
2.1	Input Format	9
2.2	Output Format	10
3	Knowledge Based Solution	13
3.1	Decision Tables	13
3.2	Prolog Software	14
3.3	Knowledge Graph	16
3.4	Ordering Reccomendations	17
4	Graphical Model	21
4.1	Metamodelling	21
4.2	Modelling	22
5	Conclusions	27
5.1	Filippo Lampa Conclusions	27
5.1.1	DMN Tables	28
5.1.2	Prolog	28
5.1.3	Knowledge Graphs	28
5.1.4	Comparison	29
5.2	Francesco Moschella Conclusion and Preferences	29
5.2.1	Decision Tables	29
5.2.2	Prolog Implementation	30
5.2.3	Knowledge Graphs	30
5.3	Comparison	30

1. Introduction

This document aims to describe the process followed to complete the Knowledge Engineering and Business Intelligence project. In Section 1.1 the description of the system to be realised will be reported, while the tasks to accomplish will be defined in Section 1.2.

Following, we will explore the System Design (Chapter 2) phase, the implemented Knowledge Based Solutions (Chapter 3) and finally the Graphical Model (Chapter 4). Finally, we draw some Conclusions (Chapter 5) on the project and present our personal points of view on the presented solutions.

All the solutions developed for this project and discussed within the following chapters can be found at <https://github.com/HarlockOfficial/Knowledge-Engineering-and-Business-Intelligence/tree/main/Exam>.

1.1 Project Description

With COVID-19 many restaurants have their menus digitised. Guests can scan a QR code and have the menu presented on their smartphones. A disadvantage is that the screen is very small, and it is difficult to get an overview. This is especially true if a restaurant offers a big wine menu.

Usually customers have preferences. Some prefer wine only from a specific country, e.g. Italy, or a specific region¹ like the Lombardy. Others exclude or prefer specific grapes², e.g. some don't like Pinot Noir. Some wines are made out of several grapes. Most people are not wine experts and would like to select their wine by describing the taste (i.e. dry/not-dry, tannin/less-tannin).

But a very prominent decision influencer is the meal³. Red wine usually is offered to meat dishes; white wine usually to fish. But there are exceptions⁴, e.g. white wine with chicken.

The objective of the project is to represent the knowledge about wines. Menus and guest preferences are needed to support the selection process (i.e. they are input). Create a system that allows to select those wines that fit the guest preferences and the menu.

¹https://en.wikipedia.org/wiki/List_of_wine-producing_regions

²https://en.wikipedia.org/wiki/List_of_grape_varieties

³<https://winefolly.com/wine-pairing/getting-started-with-food-and-wine-pairing>

⁴<https://media.winefolly.com/food-and-wine-poster.jpg>

The knowledge base shall contain information about typical wines (of an international restaurant) with wines from different regions and countries. For the taste, the grapes and the meals we focused on five major representatives. As for the Taste, we decided to allow the user to filter their favourite wine using the following flavour selectors:

- Bold
- Dry
- Fruity
- Savory
- Tannin

For each of the flavours, the user can decide on an amount from 1 to 5.

As for the Grapes, we allow the user to either include or exclude grapes from the following list:

- Cabernet Franc
- Merlot
- Pinot Noire
- Syrah
- Viura

The same goes for Meals, the user can either include or exclude a dish from the following list:

- Beef
- Crab
- Maitake Mushroom
- Parmesan
- Turkey

Is also possible for the user to either include or exclude countries and regions from which the wine comes.

1.2 Project Tasks

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

1. Define input and output of the Knowledge-Based system
2. Create different Knowledge-Based solutions based on

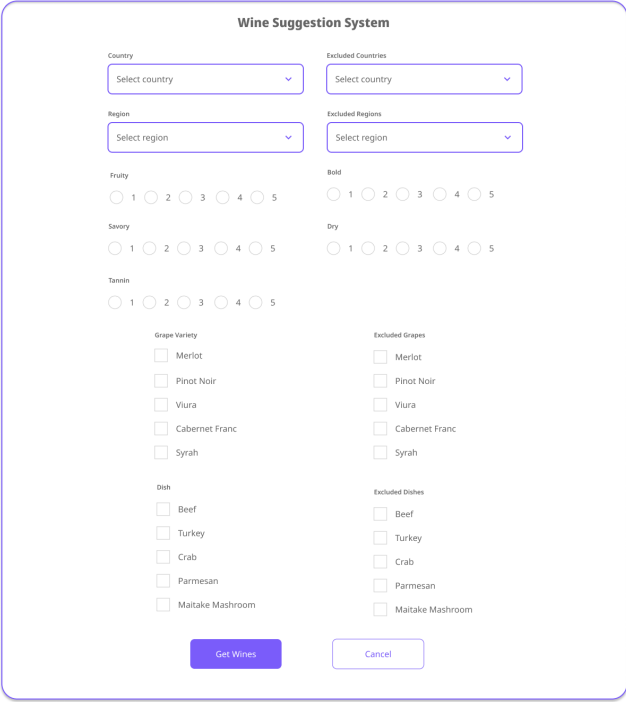
- (a) Decision Tables
 - (b) Prolog
 - (c) Knowledge Graph
3. Design a graphical modelling language, which allows a chef to represent meals and wines in a graphical way, such that it contains all information relevant for the customers to select according to their preferences.

2. System Design

In this Chapter, we will explore our idea about the data input and output that the system will analyse

2.1 Input Format

Based on the already presented homework and on the exercises completed during the several lectures, we came up with our idea of an input form on which we will base the rest of this project. Following a picture representing our idea of input form is shown (Figure 2.1). The form is supposed to be filled out by the customer in order to help it get the best wine.



The image shows a web form titled "Wine Suggestion System". It is organized into several sections. At the top, there are two dropdown menus for "Country" and "Excluded Countries", both with "Select country" as the placeholder text. Below these are two more dropdown menus for "Region" and "Excluded Regions", both with "Select region" as the placeholder text. The next section contains five rows of radio button scales, each with a label and a scale from 1 to 5: "Fruity", "Savory", "Tannin", "Bold", and "Dry". Each scale has five radio buttons labeled 1, 2, 3, 4, and 5. The final section is divided into two columns. The left column is titled "Grape Variety" and lists five options: Merlot, Pinot Noir, Viura, Cabernet Franc, and Syrah, each with a checkbox. The right column is titled "Excluded Grapes" and lists the same five options, each with a checkbox. Below these columns are two more sections, each with a title and a list of items with checkboxes: "Dish" (Beef, Turkey, Crab, Parmesan, Maitake Mushroom) and "Excluded Dishes" (Beef, Turkey, Crab, Parmesan, Maitake Mushroom). At the bottom of the form are two buttons: "Get Wines" (a solid blue button) and "Cancel" (a white button with a blue border).

Figure 2.1: Example Input form for the customer

In this form, presented in Figure 2.1, we can distinguish four groups of rows, country and region, tastes, grapes variety and dishes. Except for the taste-related group, all the rows are divided into two columns, the left one that allows the user to select elements to take into account and the right one that allows to mark elements to avoid. In the tastes group, we have all the tastes, the user can choose a value between 1 and 5 for

each of them, with 1 meaning the absence (or the wine with as less as possible) of the specified taste, and 5 meaning the wine with the strongest presence of that taste. In each field of each group, is also possible for the user to not provide an answer, in that case, we assume that the specified parameter does not matter and should not be used to filter out wines.

2.2 Output Format

The purpose of the project is to obtain the most suitable wine to match a specified meal in a restaurant. We are also aware that the best wine from a mathematical point of view might not be the best choice for the taste of a customer. Therefore we decided to provide as output a list containing the most suitable wines that match the user preferences with a certain tolerance. The list will be ordered and will provide as the first result the most appropriate match, followed by the wines that seem to match most of the user's requirements and preferences up to a certain threshold. Following we present a hypothetical case scenario, with a filled-out form and the related wine list that we ought to get from the solution.

Wine Suggestion System

Country: Excluded Countries:

Region: Excluded Regions:

Fruity: ☐ 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 Bold: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Savory: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 Dry: ☐ 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5

Tannin: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Grape Variety: ☐ Merlot ☐ Pinot Noir ☐ Viura ☐ Cabernet Franc ☐ Syrah

Excluded Grapes: ☐ Merlot ☐ Pinot Noir ☐ Viura ☒ Cabernet Franc ☐ Syrah

Dish: ☒ Beef ☐ Turkey ☐ Crab ☒ Parmesan ☐ Maitake Mushroom

Excluded Dishes: ☐ Beef ☐ Turkey ☐ Crab ☐ Parmesan ☐ Maitake Mushroom

Figure 2.2: Sample filled out form

Figure 2.2 presents a possible selection made by a customer. Here we find the preferred wine to be originated from either Italy or France, with no specified region. Also, the wine should be highly Fruity (4) and not Dry (2). The customer does not like the wines from the Champagne region nor wines containing the Cabernet Franc grapes. The customer wants to drink this wine while eating Parmesan and beef.

Figure 2.3 presents the output provided by the parameters provided by the form in Figure 2.2. We can notice the presence of the Syrah wine, listed as the most suitable wine that follows the user's choices followed by other wines that follow the customer's choices but with less strictness.

Wine Suggestion System

Best Choice

1) Syrah

Alternatives

2) Rhone

3) Supertuscan

4) Bordeaux

Close

Figure 2.3: Sample output matching the form in Figure 2.2

3. Knowledge Based Solution

Relying on the input form and format previously defined, we can query our knowledge base in three different ways. Firstly, by using Decision Tables (Section 3.1) to match user preferences and filter out undesired wine or grape qualities. With Prolog (Section 3.2), we can extend the logic provided by the Decision Tables by applying the functionalities provided by a programming language. This allows the users to manipulate data and perform even more actions. Finally, using Knowledge Graphs (Section 3.3), we can store our data in a convenient format and query them as if they were stored in a database. Following, in this chapter, we will discuss each of the three solutions, their main advantages and drawbacks, after which, we will propose a solution regarding the recommended wine ordering (Section 3.4).

3.1 Decision Tables

After defining our knowledge base and the user input form. The decision tables were the first step towards our solution. We initially followed a brute force approach by creating a unique table containing all records and combinations. After that, since the result we were about to obtain was a gigantic table that would have been complicated to read, understand and eventually fix, we decided to apply a more scalable solution. Therefore, we split the table into smaller ones, one for each decision and filter we had to take into account and procedurally filter out unwanted wines.

Specifically, in our Decision Table solution, we have five tables:

1. Filter By Dish
2. Filter By Nation
3. Filter By Region
4. Filter By Taste
5. Wine

These tables follow each other in the order they are listed. Each time, some wines are removed from the list. After the last table, a list containing all the wines that can be associated with the meal is retrieved. For each parameter, is possible to express the absence of a preference, usually, this is accomplished by not providing input, in the specific case of the taste, a placeholder parameter is required. Each of these tables is further described below.

Filter By Dish The initial table uses the dishes that a user provides as input for both categories, dishes that the user is going to eat and dishes that the user wants to exclude, and produces a list of wines that are related to the specified meal.

Filter By Nation Using the previous table output with the nations specified by the user, this table makes ulterior filtering, thus lowering the list of matching wines.

Filter By Region Using the output from the Nation table, along with the specified user region preferences, this table filters the input list of wines to produce a subset of the latter and provides its result to the following table.

Filter By Taste This table is used to match the taste preferences of a user with the available wines. Requires as input the list of wines filtered from the previous step. Also, a value between 1 and 5 (both included) is required for each of the taste parameters:

- Fruitness
- Boldness
- Savoriness
- Dryness
- Tannity

If the user does not want to express a preference for the specified parameters, it is possible to use the value “-1”. After this filter, the biggest among the five, we reach the final table.

Wine This table uses the list of wines provided as input, along with the grapes that users want and do not want in their wine, to match the perfect list of wines.

This approach sure was simple to implement, simplicity is one of the core points in DMNs, but because of this simplicity, some problems occur. Firstly, the whole work is quite mechanical. Also, eventual changes, updates and edits in the list of wines or the conditions, requires a careful update of all the tables, even a minor error might completely change the outcome. Finally, in this approach, expressing concepts like ranges is not trivial, for example, if a user wants the Tannity to be between 2 and 4, in the current DMN is necessary to search three times, each time changing the Tannity level, and if we want to change the tables to allow for ranges, as said before, the change is far from trivial.

3.2 Prolog Software

Having our knowledge base defined and some initial work on the Decision tables, we implemented a software equivalent to the tables in Prolog. Here, having all the means of a programming language, we further expanded the functionalities provided by the DMN diagram and allowed the user to perform a thresholded search in our knowledge base. For simplicity, maintainability and extensibility, we divided the software into two files. One contains all the knowledge base, while the other includes all the horn clauses.

Knowledge Base This file has been organised in 9 “blocks” and in triples as much as possible, the reason behind the use of triples is the later simplicity in accessing the data and transforming the Prolog into a Knowledge Graph. Each block contains some pieces of information. Firstly we have the block containing all the countries. Followed by the block where regions are listed and connected to a country. After that is possible to find the block where grapes are defined. Right after grapes, the list of dishes is determined. Finally, the last five blocks are used to respectively match each wine with the grape that contains, the region from which the wine comes or where is produced, the taste of the mapping between the wine and fruitness, boldness, savoriness, dryness and tannicity levels, the matching dish and finally the wine name. This file as is, makes simple the maintenance of the list of wines, countries, regions, dishes, tastes and matches. Also, to add a single wine, only a few lines have to be added to the respective block.

Horn Clauses This file contains all the rules used to provide the desired matching wines list to the user. There are only two functions directed to the user. The clause “query(...)” that requires as an input:

- a list of countries or “[A]” if there are no preferences
- a list of regions or “[B]” if there are no preferences
- a list of grapes or “[C]” if there are no preferences
- the value of the fruitness level or “D” if there are no preferences
- the value of the boldness level or “E” if there are no preferences
- the value of the savoriness level or “F” if there are no preferences
- the value of the dryness level or “G” if there are no preferences
- the value of the tannicity level or “H” if there are no preferences
- a list of dishes or “[I]” if there are no preferences
- a list of countries to exclude or “[J]” if there are no preferences
- a list of regions to exclude or “[K]” if there are no preferences
- a list of grapes to exclude or “[L]” if there are no preferences
- a list of dishes to exclude or “[M]” if there are no preferences

The other clause is called “softened_query(...)” and requires as input all the previously mentioned parameters and adds a threshold parameter. This parameter defines a circular range around each value provided as input for the taste levels. As a side note, in this clause, is mandatory to specify a preference for each parameter regarding the taste.

The other clauses present in the file, although callable by the user, should be almost meaningless by themselves.

Those clauses are mainly used to verify the input data and to find the matching values for the wine pairings, origin and name.

In this task, we faced the problem of testing and learning a new programming language while working with it. It was a nice experience and the previously learned programming languages helped in rapidly understanding this one. We found the absence of an IDE to negatively impact the software developer experience, also the testing was initially slow because we only relied on the provided Prolog website. After some time, we decided to install a Prolog interpreter on our machines and development become much faster and smoother.

3.3 Knowledge Graph

Last but not least, part of this project's tasks was focused on the knowledge graphs. These graphs map the Prolog knowledge base to a graph where more information on the data may be discovered by using the connections among information. To accomplish this subtask, we used the Protégé software by Stanford¹. This task's result is a single RDF file. We can identify four major parts:

- Classes
- Object Properties
- Data Properties
- Individuals

Classes The classes can be mapped directly from Object Oriented Programming, indeed, the classes were the types used to initialise objects and provide them functionalities and parameters. In our project, we distinguish five sibling classes.

- Country
- Dish
- Grape
- Region
- Wine

Each of which defines a core component of our model.

Object Properties The object properties can be mapped to the functions provided by each object. In our model we defined four object properties.

- <region> belongs to <country>
- <wine> belongs to region <region>
- <wine> contains grape <grape>
- <wine> is paired with <dish>

In the previous list, we mentioned the classes between angular brackets.

As is possible to notice, these kinds of property effectively relate two classes of objects one to the other.

¹The tool website can be found at the following link <https://protege.stanford.edu/>

Data Properties The data properties are parameters of an object and define indeed some values directly related to the object itself. In our project, we identified six data properties.

- name
- bold
- dry
- fruity
- savoury
- tannin

Except for the first one, all the properties belong to the wine. The first one, belongs to each class since each class can have a personal name. Also, providing names this way lowers the coupling between the “Individuals” names and their actual names.

Individuals The individuals can be seen as the instances of the classes of OOP. we have a discrete amount of individuals, therefore we are not going to list them here. We have a total of three countries, four regions for each country, five dishes and ten wines. After defining all these entities, that as for the DMN was quite mechanical work, we defined a way to query the data and reported a SPARQL Query. This query can be used in two ways, firstly to list all the entries in our dataset, and with a minor change to filter the dataset and obtain wines related to a user’s taste.

A nice feature provided by the Protégé software is the Reasoner, this tool uses the provided knowledge base, creates a graph and infers other information. Sadly not all the inferred data can be used, for example, in our project, as previously presented, we had the parameter name, this parameter was part of each class and therefore part of each object. After starting the reasoner, we found that following its logic, since the wine had a name, and the country had a name, an individual of type wine was also typed as a country. Of course, this behaviour has to be prevented and may lead to unwanted results or problems.

3.4 Ordering Recommendations

In our opinion, an important feature of a wine suggestion system is the possibility to order and rank results by matching criteria. The three kinds of knowledge solutions require different approaches to obtain this result.

Following we will discuss possible approaches to apply with Decision Tables, Prolog and Knowledge Graphs.

Decision Tables In this kind of knowledge-based solution, applying a form of output ordering is not possible. Wines are decided with a Collect policy, therefore the output is treated as a set, and not as an ordered object. All rules are indeed applied at the same time to the whole input, therefore is not possible to provide ordering of the suggested wines. On the other hand, it is possible to apply a form of weak suggestion. To apply a

weak suggestion, we can let the user define a threshold, we can then use the threshold to filter for the wine tastes. This would actually require a new set of rules, indeed, a simple threshold of ± 1 would require, only for the taste the initial rule with the correct values, then 5 more rules in which one single value is changed by $+1$ or -1 , then also all the combinations of these five rows have to be added, resulting in a huge amount of rows. After this change we should also have to allow changes for higher thresholds, in our use case, since the value ranges from 1 to 5, we should at least allow a threshold that goes up to 3. Other than allowing for a threshold in these values, we could also allow the customer to specify for all fields on which a decision was made (for example the grape variety has to be only Merlot and Syrah) if that decision could be ignored. This opens the Pandora's box, resulting in a lot more rows necessary to filter out a wine, but to also retain it if the choice was marked as possible to ignore.

Of course, in this case, is important to notice that strongly suggested wines and weakly suggested wines will end in the same set, therefore they will not be distinguishable, or at least if we stick to the single-column output, the strongly suggested and weakly suggested wines within the set will be indistinguishable.

If we try to move to a double-column output, we will be able to distinguish strongly and weakly suggested wines. But this will open a higher set of problems, requiring either to duplicate most of the tables and create a pipeline that will only filter suggested wines and the other that will provide the weakly suggested ones. Or try and filter for both of them altogether, ending up with a huge amount of rows for each table.

Prolog Moving to Prolog, a real programming language, matters become simpler. Indeed we already provide a form of weak suggestion system. As presented, we provide the two clauses “query(...)” and “softened_query(...)”, where the latter is indeed a softened version of the former. The softening only regards the strictness of the taste values but can be easily extended to also provide support for other parameters, such as the chosen dishes or countries. This change might require some more horn clauses and few testing, but the core idea should be to have a specific parameter in the clause, such as `CAN_SKIP_DISH` expecting a parameter of either 1 if the dish selection can be skipped or 0 if the dish selection has to be followed. After this, inside the horn clause, we should call some other clauses that will require as input the dish list and the parameters, then will either retain or clean up the provided input list. As for the ordering, we can rely on the Prolog interpreter, as for the tests done in our local machines and in the online web service, results are already returned with a specific ordering, which is firstly based on the order present in the knowledge base, and then based on the rules that matched before. Indeed, improving our version of the softened query and allowing the threshold to increase during the query execution may allow the engine to return an ordered list of suggested wines that will be ordered following the strength on which the taste and parameters will match the user choices.

Knowledge Graphs Actually, when defining a Knowledge Graph we are defining a specific form of database, then the reasoner will apply logic to the database and allow us to extract knowledge. Since the query language used to retrieve information from the data in the Knowledge Graph is similar in a way to standard SQL, and knowing that SQL allows results ordering, we can also correctly assume that the SPARQL language allows to order results and query the data in many ways.

Actually, applying strong and weak suggestions to this solution might result simpler

than expected. What we should do is create a specific query that retrieves all the results and adds to each resulting row a “special” identifier, the difference between the user choice and the result. This way the “special” column will hold a value representing the distance between the user choice and the actual result, we can then reverse-order the results following this column and we can also cut out the results with a value higher than a specified threshold.

This will not only allow the user to choose from a higher amount of wines but also can show what changes between its choice and the result, allowing a more aware decision of the wine.

4. Graphical Model

Differently from the previous tasks, the graphical modelling of the system aims to be as human-readable as possible. The goal of a conceptual model is to provide a visual representation of a system or concept by depicting entities, their relationships, and the flow of information, to create a clear and intuitive representation that aids understanding and communication. The process of modelling can be split into two main parts: metamodelling, i.e. the definition of domain-specific concepts, which will be taken into account in Section 4.1 and the modelling phase where the previously defined concepts get instantiated, described in Section 4.2.

4.1 Metamodelling

This phase focuses on the construction of the general knowledge about the domain, which will subsequently allow the actual modelling of the system. This can be obtained through the definition of concepts, attributes and relationships, by exploiting the functionalities of the ADOxx Development Toolkit ¹. The first step was therefore to create those classes and relation classes along with their attributes within a new ADOxx library, namely the WineLib library, associated to a new user. Through the specification of a hierarchy among the classes, ADOxx provides inheritance mechanisms which make the definition of entities and attributes more straightforward, by using an OOP style. Out of fairness, no sub-classes have been created during the development of our project, since it seemed to be a stretch, actually not needed. The only super-class extended is the construct, which acts as the root of the class hierarchy. Following the structure used to construct the knowledge graph in Section 3.3, the classes that we can find within our library are the following:

- Wine
- Grape
- Region
- Country
- Dish

Additionally, to allow clear inter-model referencing, three more classes have been added:

¹The official documentation of the tool can be found at <https://www.adoxx.org/live/introduction-to-adoxx>

- RegionWineRef
- WineGrapeRef
- WineDishRef

Their goal is to redirect the user to a different model when clicked, defining a flow between the models and allowing to better address the user through the exploration of the knowledge base, towards the choice of the right wine. We will get back to the underlying concept later in Section 4.2, when we will specifically talk about the actual models. For now, what needs to be underlined is that these links can be easily implemented in ADOxx by adding an INTERREF attribute to the class we want to make clickable, further defining the reference type (Model reference or Instance reference) and the target model/class. Within our implementation we used only Instance references, to draw a more precise path between the entities. An additional step that needs to be carried on within the development toolkit consists of the editing of the GraphReps of the several classes and relationships. GraphReps are specific textual attributes which, by being interpreted as a script by the GRAPHREP interpreter, allow to define the style and appearance of the classes within the actual model in the ADOxx Modelling Toolkit, aiming to enhance the expressiveness of the model. This field allows not only to define standard geometric shapes such as rectangles to represent an entity, but also to edit text, labels, and to dynamically apply icons based on the instance's attributes by exploiting conditional scripting constructs such as the IF-ELSIF one. Custom icons and graphics can be applied also to the relation classes to personalise, for instance, the appearance of the arrows. All the classes and the relation classes defined in the hierarchy, will not be available within the modelling toolkit until we specify them in the add-ons section of the library attributes. Each model is specified through the MODELTYPE keyword, and the classes to be included in the model are set by using the INCL keyword. Always based on our protege work, we can find four models in our implementation:

RegionBelongsToCountry draws the relations of belonging between regions and countries

WineBelongsToRegion defines the relations of belonging between each wine and its region of production

WineContainsGrape explains the grapes contained inside each wine

WinePairedWith matches each wine with the foods which better suit its flavour

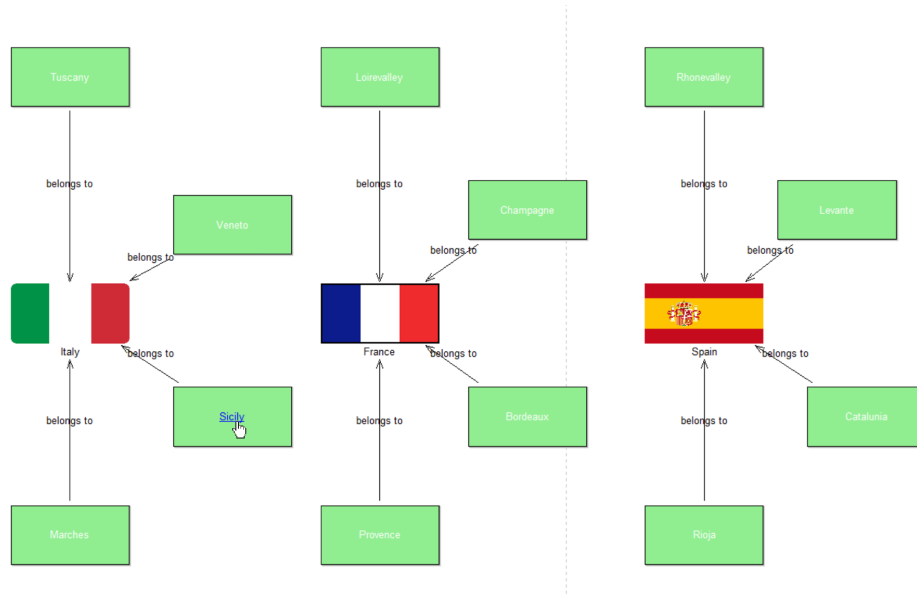
4.2 Modelling

Within the ADOxx Modeling Toolkit is where the actual instantiation of the meta-model takes place. Here it is possible to use the concepts defined in the development toolkit, specifically the ones declared within the add-ons section of the library attributes, to draw the models and the relationships among the classes in a visual way. It starts by laying down the entities, giving them names of actual instances. It is also possible, as we did for example for the wine instances, to give values to the attributes of the class, for instance, the values from 1 to 5 for Fruitiness, Boldness, Savoriness, Dryness and Tannity. After having our instances placed in each corresponding model,

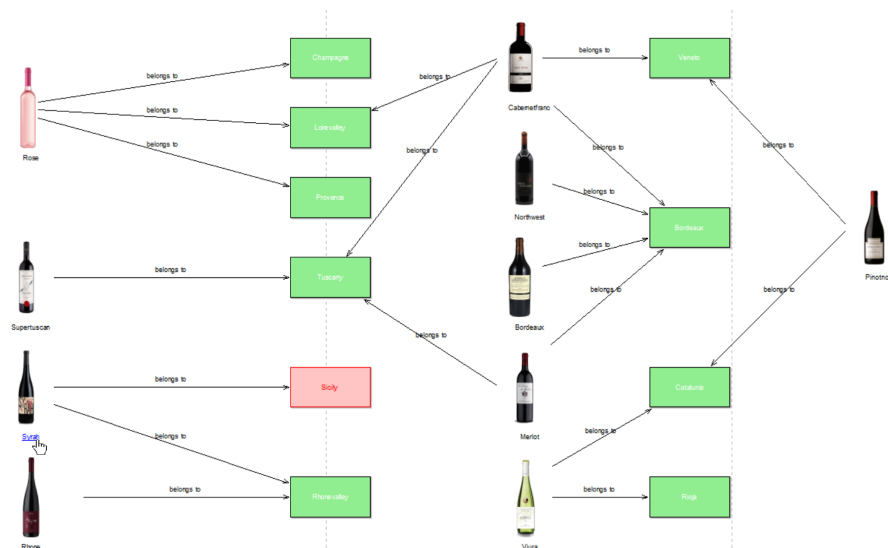
the relationships between them can be drawn, providing some semantic and meaningfulness to the models. As mentioned in Section 4.1, the path to the best wine is given by the inter-model linking of the various instances. In the modelling tool, by opening the description of an entity, it is possible to specify the precise instance we want to link it with, so that, when clicked, the redirection will take place and the referenced entity in the destination model will be highlighted.

This mechanism allows us to guide the user through a precise flow of information, which is the following:

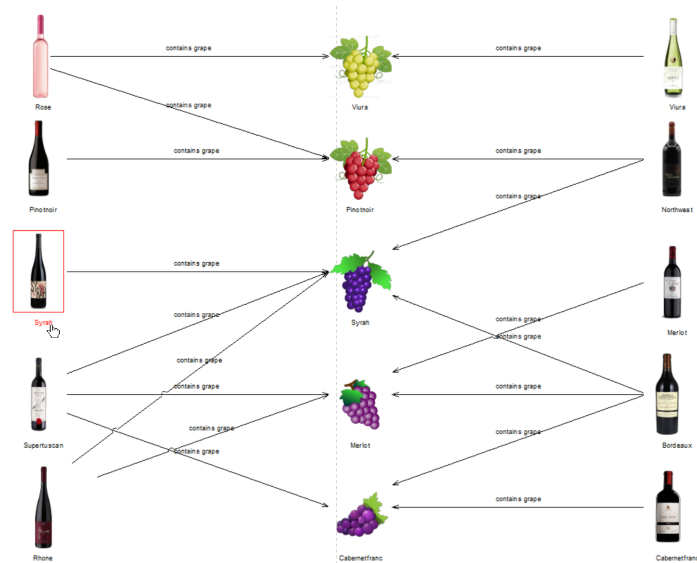
We start from the RegionBelongsToCountry model.



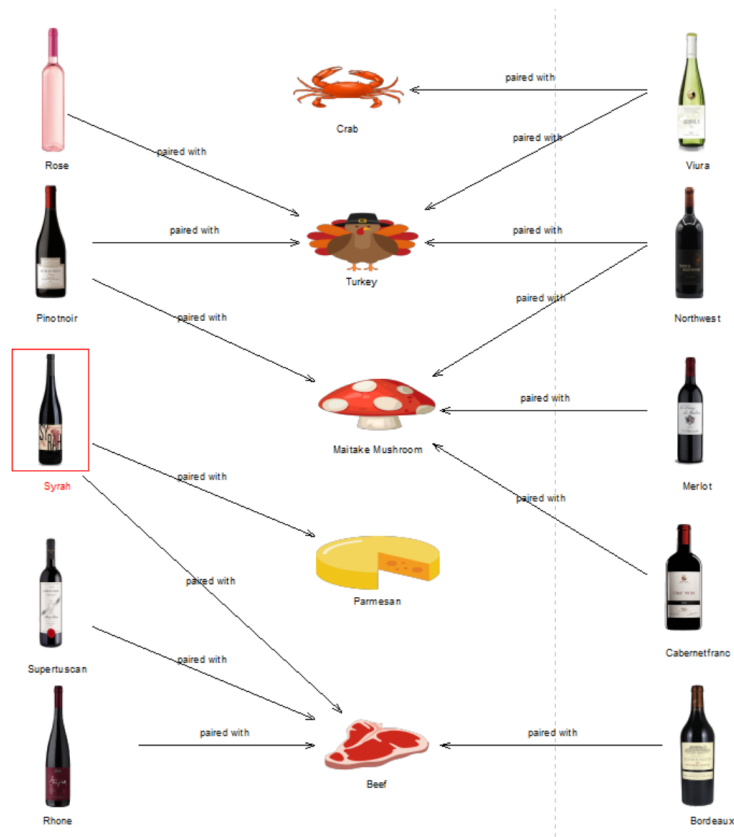
Here we can explore the available wines in a region by clicking the region's name, being redirected to the WineBelongsToRegion model



Choose a wine based on the preferred region and click on its name to be redirected to the WineContainsGrape model in which all the grapes used to make that specific wine are described



Choose a wine based on the preferred grapes and click on its name to be redirected to the WinePairedWith model in which all the dishes matching the wine are drawn, allowing the user to make its final choice based on it.



On top of that, within this model the user has also the possibility to open the description of a wine through a double click to see the values associated to its taste

The screenshot shows a software window titled "Syrah (Wine)". On the left side, there are six input fields with labels: "Name:", "Fruity:", "Savory:", "Bold:", "Tannin:", and "Dry:". The "Name" field contains the text "Syrah". Below these fields are two buttons: "Close" and "Reset". On the right side, there is a vertical panel with a tab labeled "Description". The main area of the window is currently empty.

After following this flow, the user will either be aware of the wine that better suits their preferences, or start the choice process from the beginning, changing their ideas and following different paths.

5. Conclusions

In this report, we described the process followed to complete the Knowledge Engineering and Business Intelligence project. Starting from the project description, the goal was to develop a wine suggestion system that allows restaurants to provide a simple, yet detailed and user-friendly interface to customers that wish to receive help in choosing the perfect wine for their meal. Moving to the project tasks development, the first step was to define and describe an input format. The input format consisted of two columns and four rows, the columns were to decide elements to either include or exclude. The rows were mono-thematic, with the first one allowing the customer to decide on country and region to include or exclude, the second row related to the user's wine tastes, the third about the grapes present in the wine, and the last one related to the dish. After this form and a sample case of its use, we moved to the actual implementation, starting with Decision Tables, a strict, yet simple method to represent knowledge and to query a dataset to retrieve answers. The subsequent step consisted of the Prolog implementation. This time, a programming language allowed for a more flexible implementation, that as we have shown can easily scale in both the number of wines and the rules that can be applied to query the knowledge base. As for the last part of the chapter, we presented a solution using Knowledge Graphs, a form of database, that uses a reasoning software and allows extracting knowledge from a set of data. After the Knowledge Graphs, we discussed the ordering recommendation possibilities and some plausible ways to implement this feature. We have shown that in certain cases providing this kind of feature to the user is far from trivial and implies the use of cumbersome and time-consuming solutions. We also have shown that in other cases, such as Prolog and Knowledge Graphs, the implementation of a weak recommendation system in the former was partially implemented or simple to implement, and non-necessary in the latter, since we could take advantage of the query language to filter and sort data as preferred. The following chapter was destined for the last task of the project. We presented the graphical model, with a small digression in metamodeling and modelling and how those were applied using the ADOxx software.

The project followed a bottom-up approach, with each step being the base for the next one, and we both are overall satisfied with the learning outcomes.

Following in this chapter we will present our point of view on the project, talking about the most liked parts and the ones that for us proved to be the most boring.

5.1 Filippo Lampa Conclusions

The development of the project described in this report allowed both me and my colleague to get hands-on with several tools and techniques. All of them proved to have strengths and weaknesses, therefore in this section, I will compare and discuss the

ones that, in my own opinion, are the pros and cons of every knowledge-based solution used within this project.

5.1.1 DMN Tables

The first feature the eye falls on is the ease of understanding and visualising data: DMN tables provide a representation that is intuitive to both write and comprehend, making it easier to get the key relationships between the data within them. On top of that, even if we did not take advantage of that in this specific project, DMN tables can be quickly integrated with business process modelling tools, enhancing business automation capabilities. This makes them a powerful tool if used in the right context and with awareness, providing the possibility of both quick implementation and fast queries. On the other hand, if used when should not, this kind of solution rapidly shows its drawbacks. DMN tables may not be the best choice to represent highly complex and intricate knowledge or reasoning, since their ease of usage does not come without some limitations given by the tabular representation, which may restrict the flexibility required for capturing complex relationships and constraints present in some domains, where there could even show some non-deterministic behaviours impossible to capture with this kind of structure. Moreover, as decision models grow in size and complexity, managing DMN tables becomes more and more challenging, leading to increased maintenance efforts and tables exponentially bigger, especially if designed without attention.

5.1.2 Prolog

Being based on a powerful logical reasoning syntax and on predicates with high expressiveness power, it proves to be efficient even where sophisticated inference capabilities are needed. It allows for easy manipulation and extension of knowledge bases, making it adaptable to the most disparate problem domains and making it rather scalable, allowing it to handle complex bases just as easily as simple ones if the right design choices are made. The first drawback of this language is its syntax, not too complex but still rather counter-intuitive. Most of the operators and constructs do not follow the standard mechanisms found in most of the other programming languages, making the learning curve steeper and making it sometimes necessary to check the documentation when trying to represent slightly more complex scenarios. The second problem is, Prolog may suffer from performance issues when dealing with large-scale data processing or computationally intensive tasks. It is not rare to see the engine working for high amounts of time after querying the knowledge base, sometimes even not replying at all due to the bad management of exceptions and errors.

5.1.3 Knowledge Graphs

This structure provides a flexible and extensible way to model complex knowledge domains and capture relationships between entities effectively. Especially in Protégé, the ontology editor used during the development of this project, powerful reasoning capabilities are offered, allowing for advanced inferencing and semantic querying. This flexibility and reasoning capabilities, unfortunately, do not come for free. To exploit the reasoner and the inferences at their best, significant efforts in terms of ontology design, data modelling, and ontology population are required to build and maintain knowledge graphs. Adding more classes, relationships, properties and individuals may

result in a cumbersome and time-consuming operation, resulting even in a threat to scalability, not to mention the querying and reasoning operations which can become computationally expensive as knowledge graphs grow in size and complexity.

5.1.4 Comparison

To sum up, DMN tables are user-friendly and excel at representing decision logic but may lack expressiveness for complex knowledge domains. We can find more expressiveness in Prolog, which provides powerful logical reasoning capabilities but has a slightly steeper learning curve and may occasionally show performance issues, given sometimes also the massive need for recursion enforced by its design. Knowledge Graphs with Protégé offer representational flexibility, and advanced semantic reasoning but require significant knowledge engineering efforts and high efforts to maintain and update the knowledge base. Ultimately, it is quite clear that the choice of the knowledge-based solution depends on the specific requirements of the problem domain, the complexity of knowledge, the reasoning involved and the available knowledge base, without forgetting to look ahead and take into account the needed scalability.

5.2 Francesco Moschella Conclusion and Preferences

Throughout the course of this project, my colleague and I had the opportunity to work on various aspects of knowledge engineering. All the knowledge-based solutions present pros and cons. In this section, I am going to present my own impressions about the provided solutions, also taking into account what, from my point of view, were the positive and negative aspects.

5.2.1 Decision Tables

These tables are quite straightforward. They receive an input and provide an output, to accomplish this, they follow some rules, specified as rows in the table and the output is collected following a so-called “Hit Policy”. There are many policies to collect the results and provide different features, advantages and disadvantages.

In this project, we used a “Collect” policy that allows all the rules to be checked concurrently, thus exploiting the full potential of the DMNs.

Using this policy, the input-related part of the tables follow a simple format, the columns are bonded by an “and” relation, and all rows can be considered as using an “or” relationship with each other, while the output of the matching conditions is just collected and add to a set for each output column.

This way of functioning is indeed simple to understand and easy to apply.

But, if we try to use these tables in highly complicated contexts, where there are many variables to take into account and the conditions are not so simple, the application of this method may become impossible or extremely complex.

As an example, in the project development chapter, we outlined the complexity related to maintaining or enhancing the behaviour provided by the current tables.

Also, this method of manipulating data is quite cumbersome and repetitive, even in our project, at a certain point, we thought of creating a Python script to simplify our work. The creation of all the entries was quite monotonous and at a certain point

tedious. Also, the debugging was not simple, indeed we only had access to an online tool that did not completely support our solution.

5.2.2 Prolog Implementation

The Prolog development was the part that I enjoyed the most. I had the possibility to learn a new programming language and experiment with it. This language does not present a steep learning curve, albeit finding sources and material to study is not simple. It enforces a high use of recursion and does not provide other loop-related operators. Also, classic mathematical operations are not straightforward. Definitely, this language has to be used in its own context, where it performs really well. In fact, managing the data part of our knowledge base and retrieving information from those data is quite simple and intuitive. Although learning and using this language was quite interesting and fun, I also have to recognise that Prolog presents some major problems. First of all, and as mentioned, there is no clear documentation online. Each document that I found, refers to its own dialect of Prolog, therefore it is never possible for the developer to be certain that a provided solution or function will work in its dialect, is important to notice that this enlarges the development time since each function has to be tested singularly and with the others, to check whether the presented behaviour matches the expected one. Again related to the presence of many dialects and the testing problems, we can find the lack of standards. Since anyone can define their dialect of Prolog, the function signature may change for each version, furtherly complicating this language. Finally, even if Prolog is highly performing and optimised to work with identifiers, working with a big knowledge base, may slow down the interpreter, resulting in computation that requires lots of time to provide a result.

5.2.3 Knowledge Graphs

I saw this solution as a midway between DMN and Object Oriented programming languages. Here we are allowed to define objects, with their relations and fields and extract knowledge from the instances of these objects. It was intriguing to model knowledge this way, also the querying method was close to standard SQL, which made the use of it simple and precise. Unfortunately, in knowledge graphs, I found some negative sides that resemble the ones present in Decision Tables. In my opinion, this directly derives from the Protégé software that was used to model the solution. Indeed, the creation of the model structure and of the instances was quite repetitive and uninteresting. There was no way to automate this part, and we had to repeat the same operations several times. After developing the project, we tried to use the reasoner to extract information from our data. As presented in the Knowledge Graphs related section, we discovered that the provided reasoner was not context-aware and among the extracted information, there were invalid or useless pieces of information. The presence of this non-necessary information could have impacted in a negative way the querying process used to retrieve knowledge from the data.

5.3 Comparison

In conclusion, while there were parts of the project that I thoroughly enjoyed, such as Prolog Software development and the SPARQL Query creation, there were also areas that I found less appealing, such as Decision Tables definition, and Knowledge Graphs

creation. Nevertheless, this project provided me with valuable experiences, growth opportunities, and a deeper understanding of the subject matter. After completing this project, I think that each of the presented and used technologies has its personal field of use, and is important to not misuse them.