
Supervised and Experimental Learning

CBR System for a Synthetic Task: Cocktail Creator

Professor

SANCHEZ MARRE, Miquel

Team members

BEJARANO SEPULVEDA, Edison Jair
edison.bejarano@estudiantat.upc.edu

REYES FERNANDEZ, Grecia
greCIA.carolina.reyes@estudiantat.upc.edu

DEL REY JUÁREZ, Santiago
santiago.del.rey@estudiantat.upc.edu

ZURITA MARTEL, Yazmina
yazmina.zurita@estudiantat.upc.edu

Contents

1	Introduction	1
1.1	Application Domain	1
2	Methodology	4
2.1	Roles	4
2.2	Communication	4
2.3	Tools	4
2.4	Tasks	5
3	Requirements Analysis	6
3.1	Functional Requirements	6
3.2	Non-functional Requirements	7
4	Functional Architecture	8
4.1	CBR Scheme	8
4.2	Components of the System	9
4.3	User Interaction	9
5	Proposed Solution Design	11
5.1	Case representation and Case Library structure designed	11
5.2	CBR Cycle	12
5.2.1	Retrieval	12
5.2.2	Adaptation	13
5.2.3	Evaluation	14
5.2.4	Learning	15
6	Testing and Evaluation	16
6.1	Manual Testing	16
6.2	Automatic Testing	18
7	Conclusion and Future Work	19
	Bibliography	20
A	Automatic testing	21

List of Figures

1.1	Distribution of category types	2
1.2	Distribution of alcohol types	2
1.3	Distribution of basic taste types	2
1.4	Distribution of glass types	2
1.5	Distribution of garnish types	3
1.6	Number of NaNs of each attribute	3
2.1	General Roadmap tasks	5
4.1	General system diagram	8
4.2	Class diagram of the system	9
4.3	Sequence diagram of the system	10
5.1	Case structure example	11
5.2	Case Library structure schema	12
6.1	Cocktail Creator GUI - Constraints	16
6.2	Cocktail Creator GUI - Adapted case	17
6.3	Cocktail Creator GUI - Constraints	17
6.4	Cocktail Creator GUI - Adapted case	17
6.5	Time vs Queries	18

Chapter 1

Introduction

The goal of this project is to implement a Case-Based Reasoning (CBR) system for a cocktail recipe creator. A CBR system operates on the main assumption that similar problems have similar solutions and that a solution that worked in the past is likely to work again in the future. Moreover, it can adapt one or more solutions in order to meet the requirements of its current problem.

In this context, we present the technical details of our CBR system, that can recommend a cocktail recipe based on the constraints described by the user in the graphical user interface GUI or CLI.

The set of stored cases is the Case Library, it is organized in a hierarchical memory way to make retrieval time more efficient. The cases stored in the case library are formatted as a structured XML. Real cases and others that are learned are stored, so that they can be used later. Each case incorporates a set of features: name, category, glass, ingredients, preparation, utility, derivation, evaluation, success_count, and failure_count.

For each phase of the CBR system, methods were implemented: retrieval, adaptation, evaluation and learning, the entire implementation was in Python and will be reviewed in detail in the following chapters.

1.1 Application Domain

The dataset used in this project is extracted from Kaggle¹, which is in CSV format. This dataset has 473 unique cocktails with each ingredient labeled and measured.

The dataset contains for each cocktail recipe the required ingredients, measurements and cocktail preparation steps.

The exploration of the dataset shows that the alcohol type, basic taste, measure, quantity, unit and garnish type are attributes of the ingredients. Whereas the category and glass type are features referring to the cocktails, this means a cocktail will belong to a single category and will be served in a glass of a certain type.

The attributes of the dataset are the following:

- Ingredients (303): They can be ingredients that refer to alcoholic drinks, non-alcoholic drinks or others, for example: Chocolate Ice-Cream, Cherries, egg, etc.
- Categories (9): Types of categories of alcoholic drinks or non-alcoholic drinks.
- Alcohol Type (25): Types of alcoholic drinks, related only to cocktails.
- Basic Taste (9): Taste types of ingredients other than alcoholic drinks, for example: Sour, Sweet, Salty, etc.
- Glass Type (35): Types of glasses for each cocktail, including Coffee Mug.
- Garnish type (15): Types of garnish, related only to non-alcoholic drinks in basic taste attribute.

Data exploration: To better understand the application domain, we will present the frequency of instances of each attribute in the following graphs.

¹<https://www.kaggle.com/datasets/svetlanagruzdeva/cocktails-data>

Figure 1.1 shows the number of times each instance is repeated in the different category types. The most frequent category type is the Ordinary Drink (59%), then the Cocktail (13%) and the other 7 categories represent 28% of the total instances.

In Figure 1.2 we have more alcohol types or brands of Sweet Liquors (13%) and Vodka (12%). Whereas for Absinthe and Cachaça only have one recipe that includes it. On the other hand, 850 instances are null because not all recipes contain some alcohol type, this is 48% of the total instances.

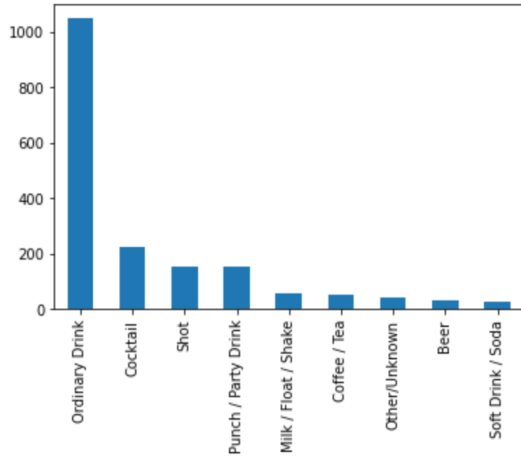


Figure 1.1 – Distribution of category types

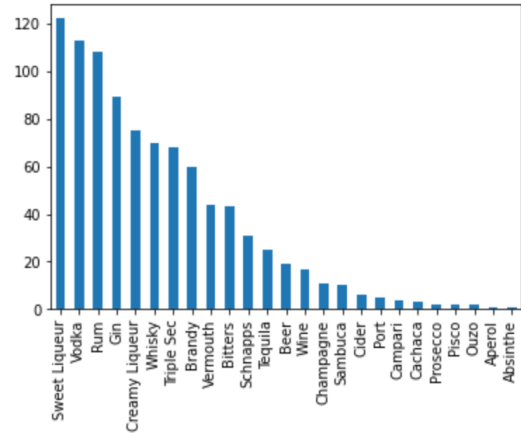


Figure 1.2 – Distribution of alcohol types

Figure 1.3 shows the different types of basic taste, 22% of the recipes contain the instance Sweet as basic taste. This is to be expected as most cocktails contain juice or soda. The other basic taste represent 25% and 52% are null (complement of the previous attribute types of alcohol).

Figure 1.4 shows the different glass types, the most frequent instance is Cocktail glass (25%), cocktails can be presented in different types of glasses and this is one of the most used, whereas Cordial glass and Copper Mug are the least used.

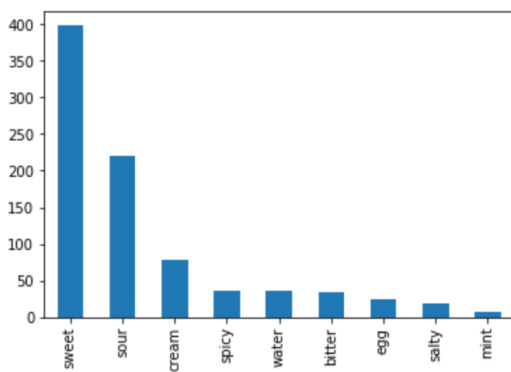


Figure 1.3 – Distribution of basic taste types

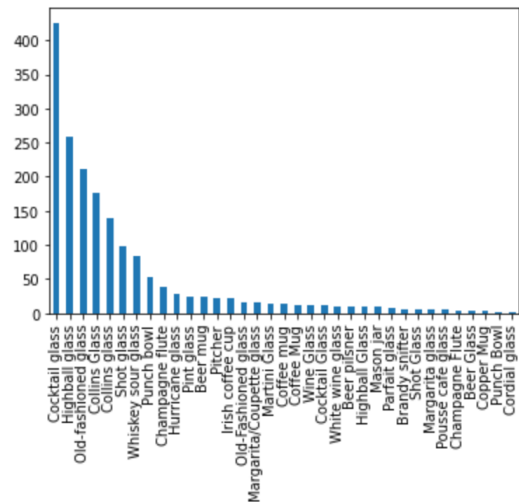


Figure 1.4 – Distribution of glass types

Figure 1.5 shows the instances of garnish type, this attribute is only related to non-alcoholic drinks (basic taste), consequently, 90% of the instances are null. The most frequent garnish types are berry, slice and twist, which represent more than 60% of the instances.

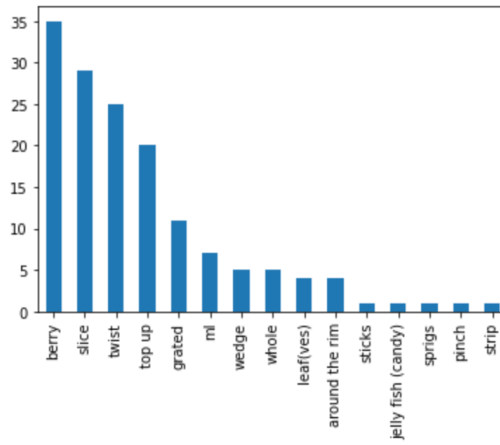


Figure 1.5 – Distribution of garnish types

Data pre-processing: We check data types, then replace non-numeric measures in Garnish_amount and transform them to float values.

We use the bar chart (Figure 1.6) to show the attributes with NaN values, in the case of numeric attributes we replace them with 0. We convert all string names to lowercase, except cocktail names. Finally, typos are corrected and ingredient names are updated.

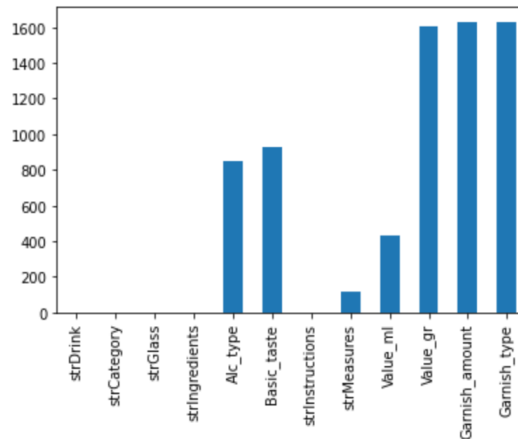


Figure 1.6 – Number of NaNs of each attribute

Chapter 2

Methodology

A well-defined plan is the cornerstone of a successful project, and for this project, a detailed plan has been defined, divided into different parts among the team members and implementing agile methodologies to optimize the results. Likewise, GitHub and Jira were used as complementary tools, the first for version control and the second to manage times, tasks and processes.

2.1 Roles

As a work strategy, it was decided to use an agile methodology where work roles were assigned. In particular, we used a Scrum-like methodology where Yazmina Zurita acted as Product Owner, Santiago del Rey as Scrum Master, and Grecia Reyes and Edison Bejarano as developers.

2.2 Communication

One of the main factors for the optimal development of the project was good communication, for this it was decided to adopt some strategies of agile methodologies, such as daily meetings, which were adapted to be held every 3 or 4 days to follow up on the tasks for each of the members, presenting the difficulties encountered along the way and discussing the possible solutions or paradigms found in the project. Likewise, an attempt was made to maintain a balance between high communication without harming productivity. In a certain aspect, this represented some challenges when trying to coordinate the schedules among the members due to the different obligations of each member. Nevertheless, it is worth highlighting the will and good disposition of all the members, which helped to facilitate communication in the group.

2.3 Tools

Github: As previously mentioned, some tools are implemented to facilitate project management, one of them was GitHub, which was used to control, share and manage code versions, which can be found in the following link: https://github.com/EjbejaranosAI/SEL_PW3.

Jira: It was used to register, manage and control the different tasks of the project, likewise, a roadmap was designed to plan the development of these tasks and be able to meet the stipulated delivery date. As shown in Figure 2.1, the project was divided into 4 parts, the first where time was allocated to read, understand and investigate the best development of the project and answer questions.

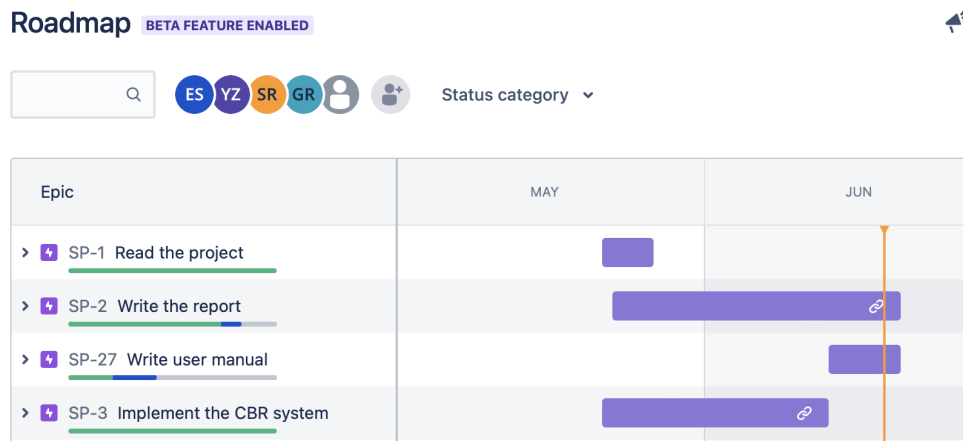


Figure 2.1 – General Roadmap tasks.

2.4 Tasks

At the time of starting the project, it was decided to carry out a division of work, according to the main components of the CBR. The case representation and library structure was assigned to Santiago del Rey. The retrieval phase was assigned to Grecia Reyes. The adaptation phase was developed by Yazmina Zurita. The evaluation and learning was implemented by Edison Bejarano. Additionally, the GUI was developed by Santiago del Rey and the CLI tool by Yazmina Zurita. All the members contributed to the elaboration of the final report, the user manual and the presentation.

However, it must be taken into account that as the project developed, the intervention of the different members in other tasks was necessary to understand, facilitate and adapt the new components in the system, so in the end an average of 62 hours per team member, in the following distribution:

	Time spent (h)			
	Edison	Grecia	Santiago	Yazmina
Case representation	0	0	14:00:38	0
Case library structure	0	0	20:31:14	0
Case retrieval and case similarity	0	42:00:00	0:48:44	0
Case adaptation	0	0	0	37:00:00
Case evaluation and learning	23:00:00	0	3:18:45	0
Create GUI	9:00:00	0	19:45:08	0
Create CLI	0	0	0	4:00:00
Testing	4:00:00	2:30:00	3:11:41	9:00:00
Writing report	18:00:00	12:00:00	5:24:14	7:00:00
Meetings	4:00:00	4:00:00	4:45:32	4:45:32
Total	58:00:00	60:30:00	71:45:56	61:45:32

Chapter 3

Requirements Analysis

In this chapter, we perform a requirement analysis on the system in order to define what functionalities the system should have and what are the acceptance criteria to consider them fulfilled.

3.1 Functional Requirements

First we defined the functional requirements that the system had to fulfill. In particular, we focused on the user requirements. The defined requirements were specified as user stories.

Select drink type

As a user, I want to be able to select the drink type, **in order to** be able to filter recipes by a given type of drink.

Acceptance criteria:

- The user can only select one type of drink at a time.
- Only drink types in the Case Library are allowed.

Select glass type

As a user, I want to be able to select the glass type, **in order to** be able to filter recipes by a given type of glass.

Acceptance criteria:

- The user can only select one type of glass at a time.
- Only glass types in the Case Library are allowed.

Include alcohol type

As a user, I want to be able to include one or more alcohol types to the recipe, **in order to** be able to get a recipe that fits my preferences.

Acceptance criteria:

- The user must be able to add or remove an alcohol type from the list of included alcohols.
- The system should show available alcohols based on the user input.
- If an alcohol type has already been selected the system should not show it as an available alcohol type.
- There must not be repeated alcohol types in the lists of inclusions.
- Only alcohol types in the Case Library are allowed.

Include basic taste

As a user, I want to be able to include one or more tastes to the recipe, **in order to** be able to get a recipe that fits my preferences.

Acceptance criteria:

- The user must be able to add or remove a basic taste from the list of included tastes.
- The system should show available basic tastes based on the user input.
- If a basic taste has already been selected the system should not show it as an available basic taste.
- There must not be repeated basic tastes in the lists of inclusions.
- Only basic tastes types in the Case Library are allowed.

Include ingredients

As a user, I want to be able to include or exclude one or more ingredients to the recipe, **in order to** be able to get a recipe that fits my preferences.

Acceptance criteria:

- The user must be able to add or remove an ingredient from the list of included ingredients.
- The user must be able to add or remove an ingredient from the list of excluded ingredients.
- An ingredient can not be in the list of included ingredients and the list of excluded ingredients at the same time.
- If an ingredient is excluded and is an alcohol, the system should add ingredient's alcohol type to the list of excluded alcohols unless the alcohol is explicitly included by the user.
- The system should show available ingredients based on the user input.
- If an ingredient has already been selected the system should not show it as an available ingredient.
- There must not be repeated ingredients in the lists of inclusions or exclusions.
- Only ingredients in the Case Library are allowed.

Evaluate recipe

As a user, I want to be able to evaluate the recipe generated by the system, **in order to** be able qualify how good is the proposed recipe according to my preferences.

Acceptance criteria:

- The must be able to introduce an evaluation score between 0 and 10.
- The score must have a precision of 1 decimal.

3.2 Non-functional Requirements

Non-functional requirements are an important part of a system definition since they specify how the overall system should function and what are the minimum standards that it must follow. In this project, we defined the non-functional requirements using the ISO/IEC 25010 as a guideline [1]. The defined requirements are:

Performance efficiency:

- The system should respond to a user query in less than 3 seconds.
- The Case Library should not exceed the size of 1GB.

Usability:

- The system should be easy to use for new users.

Reliability:

- The mean time between failures should be > 300 queries.

Chapter 4

Functional Architecture

In this chapter, we detail some of the technical aspects of the system that were decided to have a general structure of the system before starting its implementation.

4.1 CBR Scheme

This project implements a Case-Based Reasoning system. Thus, the general structure of the system is composed of the four main parts building a CBR. These main components are retrieval, adaptation, evaluation and learning. Based on the above, and the context of the application, more components were added to adapt the system according to the decisions made in the group.

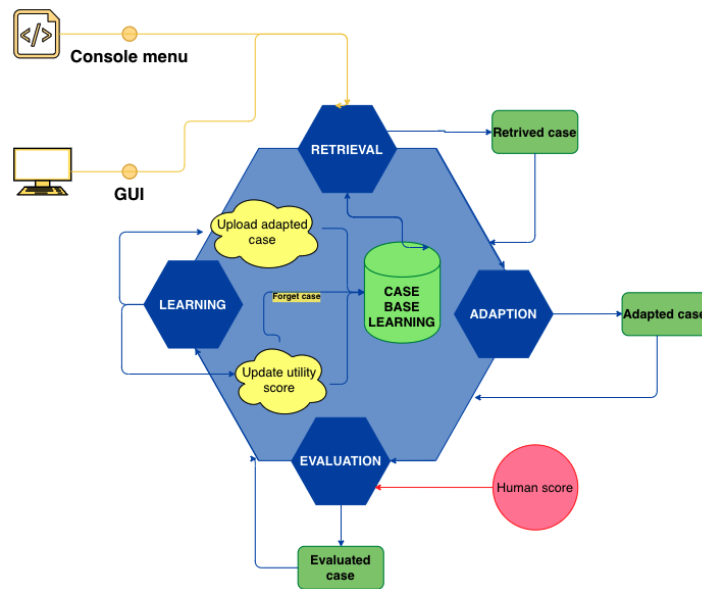


Figure 4.1 – General CBR system diagram.

As seen in Figure 4.1, the diagram shows the workflow of the system, starting from the point that the user can interact with the CBR in two different ways, through the console and the graphical user interface. These entries go to the retrieval phase where the most similar case is recovered taking into account the requested requirements.

The retrieval component is in charge of returning a set of retrieved cases which are obtained by accessing the case library and obtaining the most similar cases to the query, then they are delivered to the adaptation module, where the recipe most similar recipe is adapted to achieve the requested recipe. Continuing with the workflow, the adapted case is delivered to the user and a score is returned to the system to receive feedback on the process. The evaluation of the case is carried out through external interaction, and then with the evaluation completed, the learning process begins. Here a utility measure is computed to help in the forget step. Also, the case is classified as a success or failure and stored in the case library. Finally, the system forgets the cases that are normal and not useful.

4.2 Components of the System

The system we designed consisted of several components that worked together. In Figure 4.2, we show the specification and connection between these components. Overall, we have seven main components. The MainWindow is the class that manages the GUI. The CBR is where the main business logic is encapsulated. Then, the CBR can have multiple instances of Query, that is a data class representing a user query, and an instance of the CaseLibrary, which is the one managing the access to the data.

Additionally, we created some auxiliary classes. First, the ConstraintsBuilder, which is a class that builds the constraints in the particular format used by the CaseLibrary so that the developers do not have to worry about how the CaseLibrary internally works. Then, we created some data classes to represent the cocktails and their ingredients so that it would be easier to manage and specially print the different recipes.

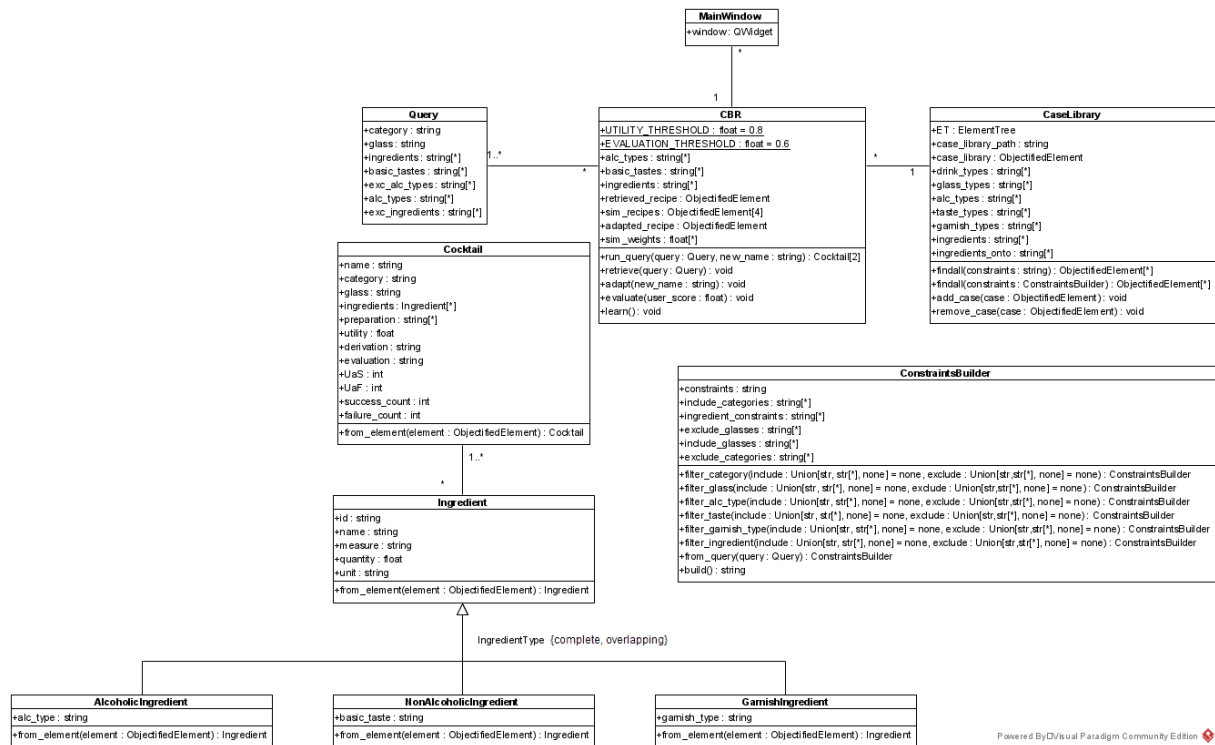


Figure 4.2 – Class diagram of the system.

4.3 User Interaction

As mentioned earlier, the user has the possibility to interact with the CBR through two options. To better understand the interaction between objects and the sequence of messages we show the sequence diagram in Figure 4.3. In it we can see the lifelines the different components and how they exchange messages over time.

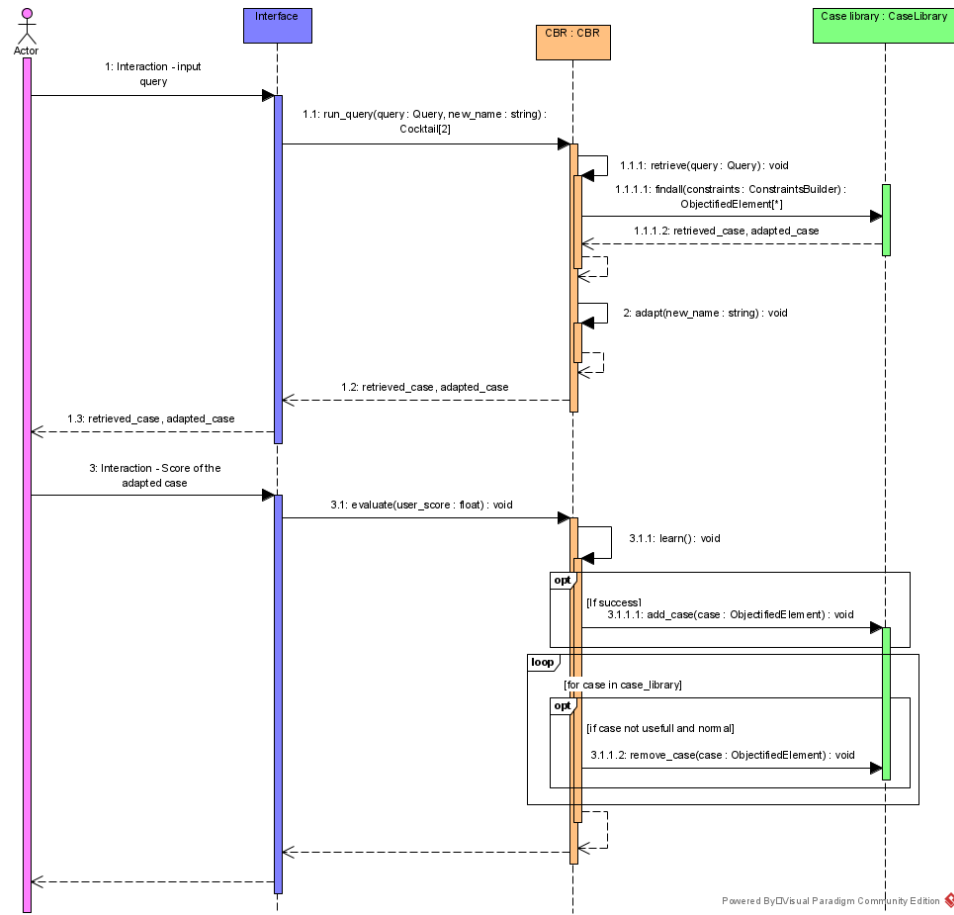


Figure 4.3 – Sequence diagram of the system.

The user is presented as the main actor, who initiates the message chain by filling an input query in the interface. Then, the interface send the query to the CBR, which starts the retrieval step and asks the Case Library for cases matching the given constraints. Then, it performs the adaptation step and sends back the retrieved and adapted recipes to the interface. After the user has received the retrieved and adapted recipes it sends back an evaluation score to the interface, which then forwards it to the CBR. Then, the CBR starts the evaluation and learning step. During this last step, the adapted case is stored in the Case Library if the adaptation was evaluated as a success. Finally, the CBR asks the Case Library to remove those cases that are normal and not useful.

Chapter 5

Proposed Solution Design

In this chapter, we describe in depth the different components that compose our CBR system. First, we talk about the Case representation selected and how is the Case Library structured. Then, we give a detailed explanation of each of the four steps in the CBR cycle: retrieval, adaptation, evaluation, and learning.

5.1 Case representation and Case Library structure designed

We begin this chapter by discussing the strategies used for the case representation and the Case Library. The first decision to make was how the cases should be represented. This decision was critical since depending on it, we would store the cases in one data format or another. This decision also impacted the Case Library structure.

In this project, the cases were cocktail recipes, which we considered to be complex objects. Since recipes contain several nested objects (e.g. the recipe steps or the ingredients), we considered that attribute-value vectors were not appropriate since they are too simple and can make the access to the recipes' properties very difficult. For this reason, we used a structured representation, which allowed us to represent these nested properties. In particular, we used the XML structure since it is a well-known and broadly supported format.

The cases initially contained the following information. The name of the recipe, the drink category, the glass type, the list of ingredients, and the steps of the recipe. For the ingredients, we also had their ID, the measure, the quantity and the unit of each of them. Also depending on the ingredient type, we had information about the type of alcohol, their basic taste and the garnish type. Additionally, we added some more information used to evaluate the recipes. This information was the case utility, whether it was an original case or an adapted case, the result of the evaluation, and some metrics for the computation of the case utility that will be described in Section 5.2.4. An example of the recipe structure is shown in Figure 5.1.

```
<cocktail>
  <name>Shot-gun</name>
  <category>shot</category>
  <glass>shot glass</glass>
  <ingredients>
    <ingredient id="ingr0" alc_type="whisky" basic_taste="" measure="1 part" quantity="30.0" unit="ml" garnish_type="">jack daniels</ingredient>
    <ingredient id="ingr1" alc_type="whisky" basic_taste="" measure="1 oz" quantity="30.0" unit="ml" garnish_type="">wild turkey</ingredient>
    <ingredient id="ingr2" alc_type="whisky" basic_taste="" measure="1 part" quantity="30.0" unit="ml" garnish_type="">jim beam</ingredient>
  </ingredients>
  <preparation>
    <step>pour one part ingr0 daneils and one part ingr2 into shot glass then float ingr1 on top</step>
  </preparation>
  <utility>1.0</utility>
  <derivation>original</derivation>
  <evaluation>success</evaluation>
  <uas>0</uas>
  <uaf>0</uaf>
  <success_count>0</success_count>
  <failure_count>0</failure_count>
</cocktail>
```

Figure 5.1 – Case structure example.

Once we defined the case representation, we proceeded to decide which was the most suiting structure for our Case Library. In our data, we had five features that we considered that could be used to classify the recipes: drink type, glass type, alcohol type, basic taste, and ingredient name. The two former are general features that apply to the hole recipe. The rest, are specific of each ingredient. At first, we

thought of using a discriminant tree or a k -d tree that used the five features mentioned. We decided to discard the latter since we considered that its implementation would be too difficult given that our data was completely categorical. Specially for the learning and forgetting steps. The discriminant tree seemed an approach more suitable for our CBR. However, using the five discriminant features would make the task of managing the Case Library too complex.

Given that we wanted to build a Case Library that was simple and easy to manage, but where the retrieval would be efficient, we finally decided to use a hybrid approach. We used the two more general features (i.e. drink type, glass type) to build a discriminant tree where the leaves are a group of cases structures as a flat memory (see Figure 5.2).

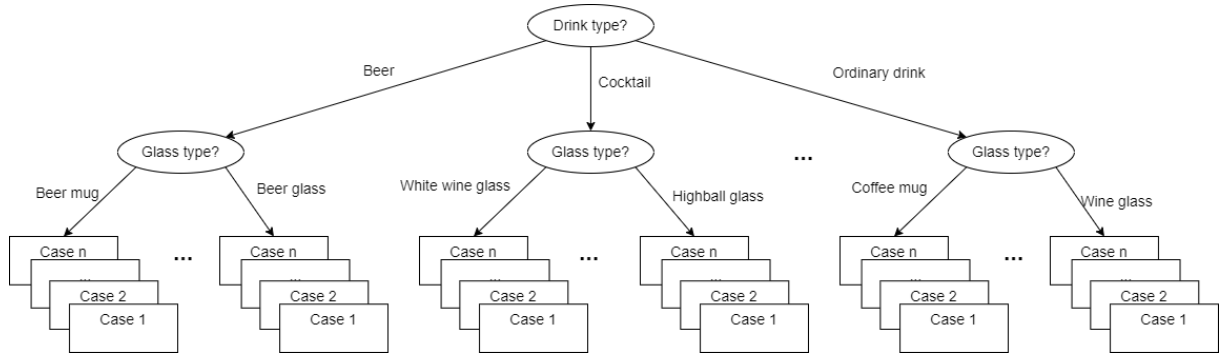


Figure 5.2 – Case Library structure schema.

5.2 Methods Implementing each CBR Cycle Step

5.2.1 Retrieval

The first step of the CBR system is the retrieval of a set of cases of cocktail recipes from the case library. This is, to find a subset of cases that are the most similar to the new case described by the user. With this we seek to maximize the similarity between the new case and those recovered.

The retrieval process consists of 2 stages: **Searching the most similar cases to the new case:** The objective of this step is to pre-select the cases that match with the constraints given by the user in the new case, the constraints are the inclusions and exclusions that the user enters in the GUI or in the CLI.

First, we filter the elements that match the constraints: category, glass, alcohol types, taste, ingredients (included and excluded). In case of having less than 5 recipes that match the user's constraints, these are progressively relaxed until having at least 5 cocktail recipes. For instance, if we only have 2 cases that meet the constraints requested by the user, the "ingredient" attribute will be the first to be ignored, that is, these constraints will not be taken into account. In case of continuing to have less than 5 cases, the second constraint to be ignored will be basic taste and so on progressively. The sequence is as follows:

1. Ingredients
2. Basic tastes
3. Alcohol types
4. Ingredients excluded
5. Glass
6. Category

This stage is very effective in retrieval with respect to simplicity and time, because we reduce to a subset of at least five cocktail recipe cases to assess similarity.

Selecting the best cases: In this stage, the objective is to select the most relevant cases among the subset retrieved in the previous step. That is, we calculate the similarity with each of the cocktails of

the subset. The cases retrieved with the highest similarity will be the closest. That is, the most similar to the new case. So by having similar cases we will have similar solutions.

To calculate the similarity we will define weights $w_k \in [0, 1]$, which have been defined by the team, to provide the system with the adaptability of the most relevant characteristics for the user based on their input preference.

Each constraint of the new case is evaluated one by one with the selected subset of cases, if the constraint ingredient is used in a cocktail the similarity increases. However, if the ingredient excluded from the constraint is used in the cocktail, the similarity decreases. This similarity depends on the weight of the feature. In case the constraint is not met, we add the weight to the normalization score. The weights defined for each characteristic are:

Constraints included:

- `ingr_match`: 1.0, the match is exact, it is important that the retrieved cocktail contains the ingredient.
- `alc_type_match` and `basic_taste_match`: 0.85, these constraints are important but not relevant for retrieving the most similar cases.
- `ingr_alc_type_match`, `ingr_basic_taste_match` and `glass_type_match` : 0.5, are not important constraints on the ingredients.

Constraints excluded:

- `exc_ingr_match` , `exc_alc_type` and `exc_basic_taste`: -1.0, these constraints are relevant, although negatively. The user requests that the constraints of these attributes are not considered in the cocktail.
- `exc_ingr_alc_type_match` and `exc_ingr_basic_taste_match` : -0.5, are not relevant constraints on ingredients.

The normalized similarity between two cases is expressed as follows:

$$s(x; y) = \frac{\sum_{k=1}^K w_k sim_k(x, y)}{\sum_{k=1}^K w_k} \quad (5.1)$$

where $s(x; y) \in [-1; 1]$ is the similarity measure between cases x and y , x is the instance retrieved from the case library, y is the instance of the new case given by the user, the top K closest indexes retrieved and $sim_k(x, y)$ is the similarity metric.

Finally, to estimate the possibilities that some of the pre-selected cases will be selected as a retrieved case, we use the utility function to multiply it with the normalized similarity for each case.

5.2.2 Adaptation

After the retrieval step, the system adapts the recipe that best satisfies the petition of the user to fulfil possible remaining constraints. The process follows a structural adaptation strategy that uses transformation methods to add, remove or substitute ingredients in the recipe. The main operations applied on a copy of the most similar case (the adapted recipe from now on) are:

- `include_ingredient`: The ingredient is added as a child in the node of ingredients. Its attributes are set to `measure='some'` and `id='ingrN'`, where N is the number of elements in the node of ingredients. If a measure M is passed as argument, then `measure='M'`. In addition, a step is added as the second element in the preparation node, with the text `add ingrN to taste` if a measure is not specified and `add ingrN` otherwise. Usually, the glass is specified in the first step (e.g.: fill

tall glass with ice cubes) and garnishes and serving instructions are specified in the last step (e.g.: garnish with lime slice). Thus, inserting new ingredients in the second step was the most sensible choice.

- **delete_ingredient:** The ingredient is removed from the node of ingredients. If it is present in a preparation step with other ingredients, its ID is replaced by the placeholder [IGNORE]. If it is the only ingredient present in a preparation step, the whole step is removed from the preparation node.
- **replace_ingredient:** Given two ingredients, if their attributes values for `alc_type` or `basic_taste` match, the text of the first is replaced by the text of the second.

Given the query of the user, the adapted recipe and the next four most similar recipes (the similar recipes from now on), the system executes the following four steps:

- **Exclude ingredients:** The system iterates through the list of ingredients to exclude, checking if any of these ingredients are present in the adapted recipe. If that is the case, we distinguish between two cases:
 - When the ingredient is not an alcohol, the system looks for an ingredient with the same `basic_taste` and calls `replace_ingredient`. It searches in the list of ingredients that the user wants in the recipe first, in the ingredients in the similar recipes second and in all the case library third. If there was more than one match in the third case, the system chooses a random one. If a match is not found, `delete_ingredient` is called.
 - When the ingredient is an alcohol (`alc_type` \neq 0), `delete_ingredient` is called directly. The reason is that, when a user wants to exclude "rum" from the recipe, for instance, it would usually mean that the user does not want any ingredient with `alc_type` = 'rum'. Thus, it would not make sense to replace "rum" with "raspberry rum" for instance, which is what would happen following the first approach.
- **Include ingredients:** The system iterates through the list of ingredients to include and checks if they are already in the adapted recipe. If it is not the case, it searches for an ingredient with the same `alc_type` or `basic_taste` in the similar recipes to obtain the measure. Then, `include_ingredient` is called, passing the measure as an argument if one was found.
- **Adapt alcohol types and basic tastes:** The system iterates through the lists of alcohol types and basic tastes to include and checks if they are already satisfied in the adapted recipe. If it is not the case, the system searches for an ingredient with the basic taste or alcohol type desired. It searches in the similar recipes first and in the case library second. If such an ingredient is found, `include_ingredient` is called.

After this process, all the user constraints should be satisfied.

5.2.3 Evaluation

As is mentioned in [2], "Evaluation is an important issue for every scientific field and a necessity for emerging software technology like case-base reasoning". It is basically a strategy to quantify the efficiency of the process and with that, it is possible to measure how well the solution that is developed is. The evaluation can be observed for different approaches, one of them is to validate the system, the other to compare between different variables or characteristics [2]. For this project, the evaluation consisted of comparing if the solution given for the system satisfied the user's needs.

Following this previous approach, there was a lot of discussion about how to apply this evaluation to this project, which gave rise to some interesting development ideas.

On the one hand, carrying out an evaluation taking into account the query that was delivered to the system and the adapted recipe using similarity as a tool to find a quantification of its quality. However, this initiative was abandoned since by doing so, this would mean that in most cases, a value similar to 1

or very close to it would be obtained, since one of the bases of the system was that it needed to meet the requirements requested in all categories. So we considered that it did not make sense to carry out this type of evaluation.

On the other hand, we tried to think of performing an evaluation based on the similarity between the most similar case and the adapted one, which would result in taking the evaluation result with the highest similarity found in the case library by the user's query, which would not make sense either since it is expected that the new case is not similar to the adapted one. Also, it would heavily depend on the user's query.

Finally, we decided that according to the CBR methodology that was developed in this project, the most appropriate thing would be to have a rating by the user. This rating is interpreted as the user's level of satisfaction with the resulting recipe. In this way, it will be possible to have a quantification by the user that determines how well or poorly the system is working.

Based on the above, it was determined that for the evaluation process the user would input a score between 0 and 10 allowing the system to take this value as reference to determine if the adaptation was a success or not. In particular, the system deems as successful those adaptations that receive a user score > 0.6 .

This can be interpreted as questioning to a human expert for feedback from the real world and allowing the system to learn from experience after being validated from an expert.

5.2.4 Learning

There are many types of learning depending on the topology of the system we want to implement. In our system we have decided to only store those cases that we deem successful. Thus, we can classify it as learning from experience. We also considered enabling the system with the capacity of learning from failures, but given the time constraints and the complexity of this approach we finally decided to discard it.

The learning process works as follows. After receiving the result of the evaluation step, the system saves the new case if it was classified as successful. Otherwise it is not saved. Additionally, we update the utility measure for all the retrieved cases by applying the following:

$$UM(C) = \frac{\frac{\#UaS}{\#S} - \frac{\#UaF}{\#F} + 1}{2} \quad (5.2)$$

where C is the retrieved case, UaS is the number of the times that the case was used and there was a success, S is the total amount of successes when the case was among retrieved cases, UaF is the number of times that the case was used and there was a failure, and F is the total amount of failures when the case was among retrieved cases.

This measure allows having a parameter that feeds back to the system according to the failed and successful cases, which allows the system with a criterion to find those cases that are not useful.

To decide if a case should be forgotten, the system looks at the utility measure. If it is below a threshold of 0.8 the system considers the case as not useful and looks if it is a normal case. If the system finds that the case is normal then it forgets it. We consider a case to be normal if there is at least one more case having the same drink type, glass type, and alcohols or basic tastes. This allows the CBR to meet the criteria of competence and efficiency, thus saving only the cases of interest and that are satisfactory, using the minimum amount of space possible.

Chapter 6

Testing and Evaluation

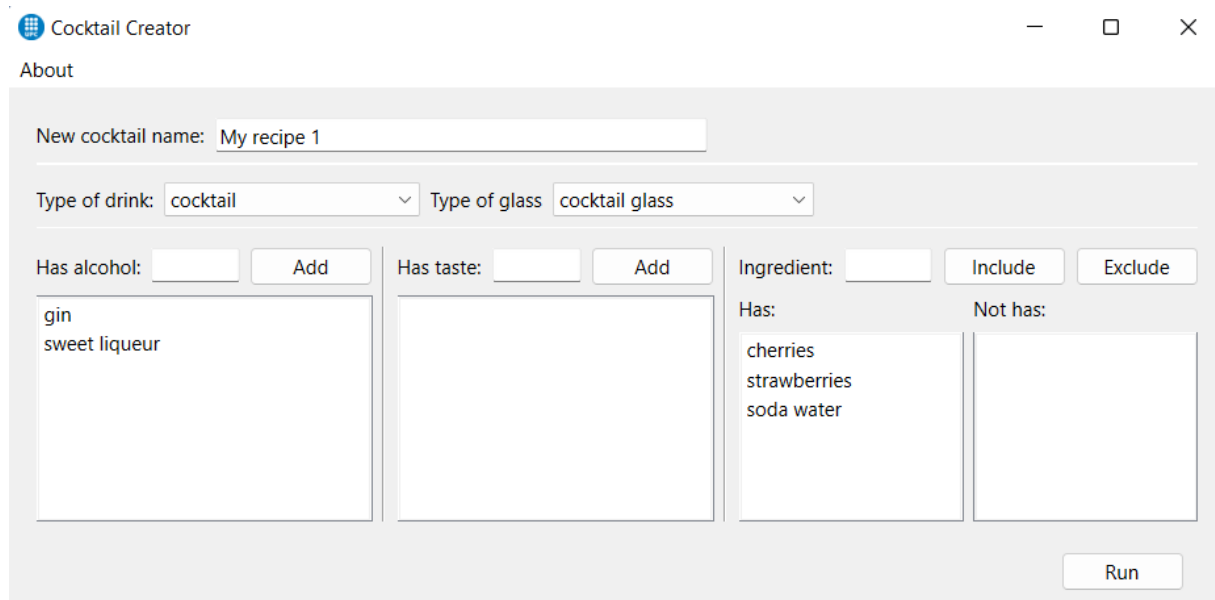
In this chapter, we will present some of the experiments used for evaluating the proposed system and the associated results. The experiments will be done by using the GUI interface and a script to perform automatic testing.

It should be noted that during the implementation of the system, several tests were carried out to check the correct functionality and coherence of the system.

In the following sections we analyze the results of the proposed system: the behavior of the system and its adaptation to changes in constraints, and the evaluation of system performance.

6.1 Manual Testing

In this first experiment we enter only preferences, we do not include ingredient exclusions (ingredients we do not want). We call this new case “My recipe 1”, we choose the drink type: cocktail, the glass type: cocktail glass, the alcohol type: gin and sweet liqueur, we leave the basic taste empty and the ingredients to include will be cherries, strawberries and soda water, as shown in the Figure 6.1.



The screenshot shows the 'Cocktail Creator' application window. At the top, there's a title bar with a grid icon, the text 'Cocktail Creator', and window control buttons (minimize, maximize, close). Below the title bar is an 'About' button. The main interface has a light gray background. At the top, there's a text input field labeled 'New cocktail name:' with the value 'My recipe 1'. Below this are two dropdown menus: 'Type of drink:' set to 'cocktail' and 'Type of glass:' set to 'cocktail glass'. Further down, there are three sections for constraints. The first section is 'Has alcohol:' with a text input field containing 'gin' and 'sweet liqueur', and an 'Add' button. The second section is 'Has taste:' with an empty text input field and an 'Add' button. The third section is 'Ingredient:' with a text input field and two buttons: 'Include' and 'Exclude'. Below the 'Ingredient:' section, there are two columns: 'Has:' and 'Not has:'. The 'Has:' column contains a list of ingredients: 'cherries', 'strawberries', and 'soda water'. The 'Not has:' column is empty. At the bottom right, there is a 'Run' button.

Figure 6.1 – Cocktail Creator GUI - Constraints

By clicking on the “Run” button, the system gives us the “Retrieved Case” most similar to the entered constraints and also the “Adapted Case”, which for this simple case no adaptation was needed to meet all constraints. To finish we select the score between 0 to 10, in this case it is 10 because it satisfies all the constraints. The adapted recipe includes the ingredients and the alcohol types requested (note that the ingredient “galliano” satisfies the constraint of “sweet liqueur” since it has `alc_type = "sweet liqueur"`).

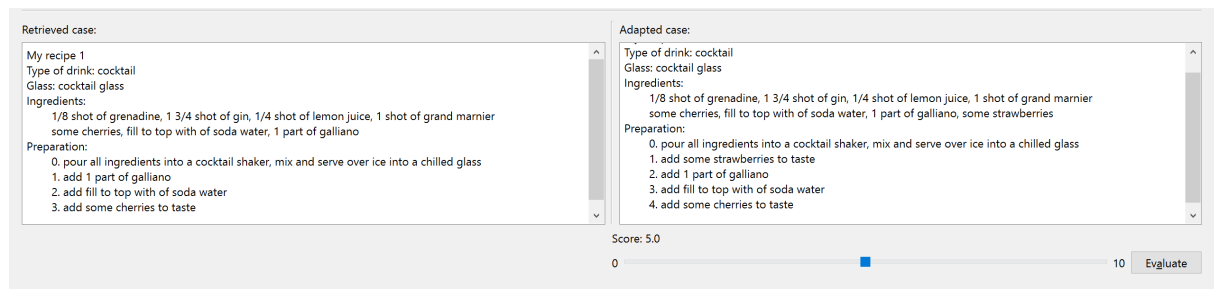


Figure 6.2 – Cocktail Creator GUI - Adapted case

In the second experiment, we enter the same constraints as in the previous example and we exclude some of the ingredients that the system gives us in the recovered or adapted recipe (the same), for example, the recipe includes in the preparation of the cocktail a 1/4 shot of lemon juice and 1 shot of grand marnier, Figure 6.3 . We enter these two ingredients as exclusion and we will review what the system returns.

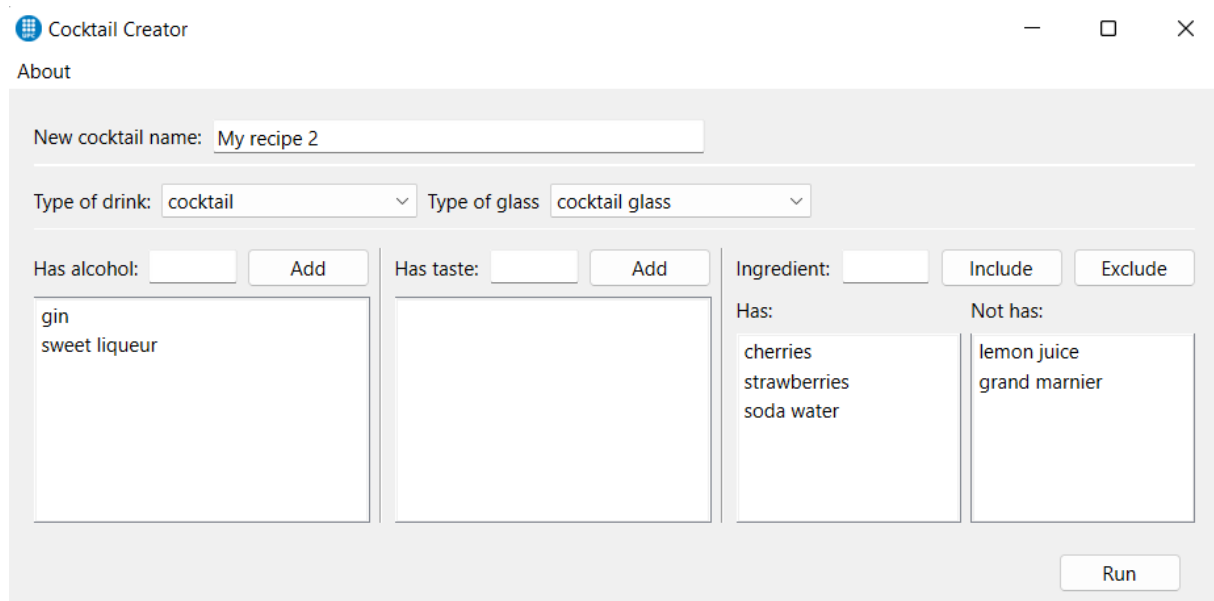


Figure 6.3 – Cocktail Creator GUI - Constraints

Once we run the new case with the preferences and exclusions, we see that the retrieved recipe called “Addington” is the most similar case in the case library. On the other hand, the adapted case gives an excellent response to our request, since it satisfies all the constraints as in the previous example. To finish we select the score between 0 to 10, in this case it is 9 because it satisfies almost all the requested ingredients.

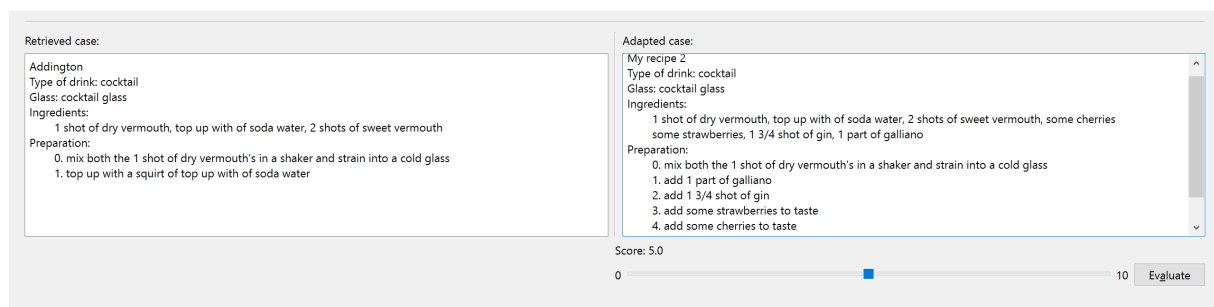


Figure 6.4 – Cocktail Creator GUI - Adapted case

As we have seen in these experiments, the system can retrieve and adapt a cocktail recipe to match the

constraints given by a user, as well as learn from the experiences of it. Also, the system can correctly manage: addition, replacement and removal of ingredients to meet user requests either by GUI or CLI interface.

6.2 Automatic Testing

For the automatic tests, the number of queries was tested against time, and for this the `system_test.py` script was implemented to perform the necessary tests. The script generates a number of queries randomly that satisfy the same restrictions as when the user inputs the query manually. It also saves the query, the retrieved case, the adapted recipe generated, the total number of queries and the total time that the system spent processing them in a text file.

To test the performance of the system, we performed several consecutive queries and computed the required time by the system to answer all of them. Specifically, we executed the same experiment several times by changing the number of queries from 50 to 500 in increments of 50. To obtain more accurate results, we executed 11 times each of the experiments. The complete table of results is included in Appendix A. In Figure 6.5, we show the time averaged over the 11 runs for each number of queries.

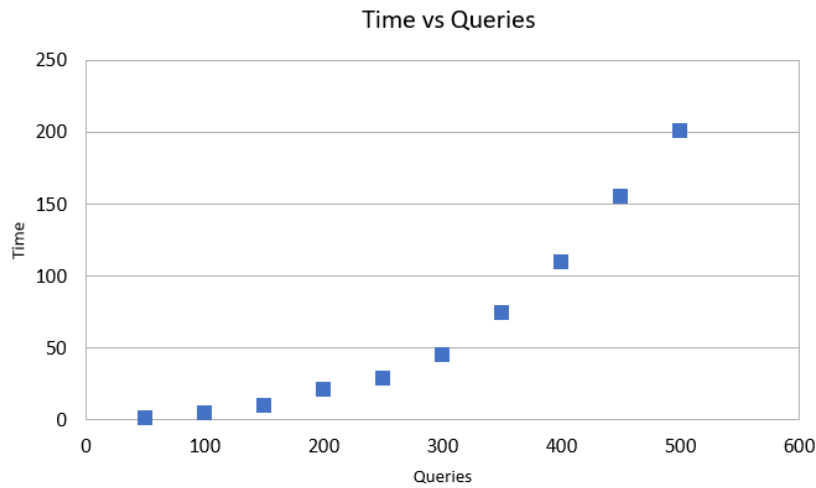


Figure 6.5 – Time vs Queries.

As can be seen in Figure 6.5, the processing time grows exponentially with the number of queries. The reason is that the case library grows considerably while processing a large number of queries. Consequently, the system has to search among a higher number of cases in the retrieve step which increases the processing time. As a reference, the size of the case library is 522 KB before executing the test and 3.22 MB after processing 500 queries. To try to decrease the growing rate of the case library, we tried several values for the thresholds involved in the learn and forget steps. However, we were not able to improve the scalability of the system with respect to the number of queries. Looking at the results, we can give a rough estimation of the mean time spent processing a single query in the worst case explored. Since processing 500 queries takes around 201 seconds, processing one takes around 0.40 seconds, which is way below the limit of 3 seconds specified in the technical requirements.

Chapter 7

Conclusion and Future Work

We started this project with the goal of making a CBR system able to generate a cocktail recipe based on some constraints that a user inputs. Through this report, we have detailed what approach we have taken to achieve this goal. First, we performed the planning and the requirements analysis to define the scope of the solution and give some guidelines on how it should be implemented. Second, we have detailed how our proposed solution implements the four basic steps of the CBR cycle. Finally, we performed both manual and automatic testing on our solution and evaluated the results obtained to assess the quality of our system. From the manual testing we observed that the system performed as expected to common queries by returning an adapted case that matched our requirements. To put the system under more stress and test its response to uncommon queries we performed the automatic testing. Our results show that the system performs well when receiving several consecutive queries with random constraints and that it complies with the performance efficiency requisites of time and space efficiency.

Although we have achieved our original goal, we feel that the proposed solution can be further improved. We have performed some data preprocessing in order to clean the data before using it, we feel that further improvements can be done in this direction. For example, we could apply different NLP techniques to better clean the data and to obtain better information, specially from the ingredients and the recipe steps. To further improve the CBR, we could also explore using an alternative similarity metric in the retrieval step like Nearest Neighbour or K-Nearest Neighbours [3]. In the evaluation phase, we could introduce an auxiliary similarity measure to evaluate how good is the adapted solution with respect to the input constraints. Then use this auxiliary measure together with the external evaluation given by the user to improve the CBR learning. In addition, we could improve the scalability of the system with respect to the number of queries by using a method to forget a higher number of cases in the forget step or to learn fewer in the learn step. Regarding the user experience, we could perform some improvements mostly to avoid contradictory inputs. For instance, we could verify that if a given ingredient must be excluded but its alcohol type must be included there should exist at least one more ingredient with the same alcohol type. Otherwise, we should warn the user that there is a conflict in their request and ask him whether to remove the ingredient from the exclusion list or the alcohol type from the included types. Finally, we should perform more testing of the whole system to look for possible inconsistencies.

In conclusion, we consider that the main objective has been reached. We implemented a Case-Based Reasoning (CBR) system for a cocktail recipe. That is, the system can recommend a cocktail recipe based on the constraints described by the user in the GUI or CLI. The developed system can successfully retrieve and adapt a cocktail recipe to match the constraints given by a user, as well as learn from its experiences.

Bibliography

- [1] ISO/IEC 25010:2011, “Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models,” International Organization for Standardization, Geneva, CH, Standard, Mar. 2011.
- [2] K.-D. Althoff, “Evaluating case-based reasoning systems,” in Jul. 2000.
- [3] I. Watson, “An introduction to case-based reasoning. progress in case-based reasoning.,” in Jul. 1996, pp. 3–16.

Appendix A

Automatic testing

Table A.1 – Processing time with different number of queries.

Time	Queries	Time	Queries	Time	Queries
1.38493490219116	50	4.29196810722351	100	9.06908798217773	150
1.37345027923584	50	4.38534283638001	100	9.01099705696106	150
1.37767744064331	50	4.38283634185791	100	9.06769371032715	150
1.41104555130005	50	4.39605236053467	100	9.28728032112122	150
1.37394189834595	50	4.3205292224884	100	9.35278463363647	150
1.37864995002747	50	4.45460200309753	100	9.36301898956299	150
1.36660313606262	50	4.41691064834595	100	9.33764123916626	150
1.37254500389099	50	4.40092825889587	100	9.40368914604187	150
1.40381217002869	50	4.54373335838318	100	9.33054113388062	150
1.39853549003601	50	4.61489725112915	100	9.4474663734436	150
1.37680625915527	50	4.66941547393799	100	9.32127189636231	150
Mean value		Mean value		Mean value	
1.38345473462885	50	4.44338326020674	100	9.27195204388012	150
Time	Queries	Time	Queries	Time	Queries
21.3479435443878	200	27.1378765106201	250	42.0983436107636	300
21.7094576358795	200	27.3972456455231	250	43.3202357292175	300
22.8747620582581	200	27.9866280555725	250	43.8733096122742	300
21.6608307361603	200	28.7319781780243	250	45.1996734142304	300
23.1608324050903	200	29.2954306602478	250	46.258882522583	300
21.8523716926575	200	29.0745108127594	250	44.6144993305206	300
21.4457993507385	200	28.3104608058929	250	44.5775504112244	300
22.1023795604706	200	29.9571568965912	250	46.539737701416	300
20.646683216095	200	27.4093127250671	250	44.1315064430237	300
21.1820690631866	200	27.7862620353699	250	46.6845042705536	300
15.3601479530334	200	29.7556948661804	250	44.401998758316	300
Mean value		Mean value		Mean value	
21.2130252014507	200	28.4402324719862	250	44.700021982193	300
Time	Queries	Time	Queries	Time	Queries
70.1726493835449	350	106.257402181625	400	176.946761369705	450
69.604856967926	350	105.136089801788	400	149.70193696022	450
74.7269940376282	350	102.265642404556	400	147.591487169266	450
76.846312046051	350	114.891226291657	400	156.176809310913	450
69.9400939941406	350	113.277559757233	400	146.027337551117	450
73.4041233062744	350	111.890584230423	400	143.69486951828	450
73.0571157932282	350	113.659188985825	400	166.420283794403	450
74.8357214927673	350	113.084007978439	400	156.471763372421	450
76.0785222053528	350	111.604095935822	400	160.020894289017	450
74.922265291214	350	108.04275560379	400	151.154732704163	450
<i>Continues in the next page</i>					

Table A.1 – <i>Continuation of the previous page</i>					
Time	Queries	Time	Queries	Time	Queries
81.8831627368927	350	107.267331123352	400	150.505618572235	450
Mean value		Mean value		Mean value	
74.1338015686382	350	109.761444026774	400	154.973863146522	450
Time	Queries				
224.331243515015	500				
197.819222688675	500				
190.784683704376	500				
188.836595535278	500				
193.160325527191	500				
191.601882219315	500				
189.65607380867	500				
201.083418130875	500				
209.451683044434	500				
207.879523277283	500				
213.778276920319	500				
Mean value					
200.762084397403	500				