Design and Development of a Case-Based Reasoning Shell integrated in an Intelligent Data Analysis Tool

Beatriz Sevilla

September 4, 2009

There are some CBR shells available in the literature. Nevertheless, most of them do not allow implementing the whole basic CBR cycle in a very flexible way.

"That is what learning is. You suddenly understand something you've understood all your life, but in a new way." By Doris Lessing.

*"Human beings, who are almost unique in having the ability to learn from the experience of others, are also remarkable for their apparent disinclination to do so."*By Douglas Adams.

"Don't just learn something from every experience, learn something positive." By Al Neuharth.

Acknowledgments

Firstly, I would like to dedicate this master thesis to my father and sister. I really appreciate that they have always support me although they do not understand why I keep studying and not working. Many thanks for having not questioned my decision.

Secondly, but not less important, I would like to thank Alex Jurado for supporting to me in most of the process and helping me to improve this documentation.

In addition, I would like to be grateful to Roberto Confalonieri and Javier Vázquez for helping me with the English and with any question that I asked to them.

Thanks, also to David Artiga for being always my *"javaman"* in the implementation issue.

I should thank Zoraida Hidalgo and MariSol Sánchez for their psychological support, facing the pressure that the presentation of this project means.

I cannot forget to thank my advisor Dr. Miquel Sánchez for guiding me and teaching me all what I know about CBR.

Eventually, I would like to remark that if there had not been people who was dedicated to research - in this case CBR research, then the realization of this project would have not been possible either. For this reason thank Roger Schank, Cristopher Riesbeck, Agnar Aadmodt, Ian Watson, David Leake, Ramón López de Mantaras, Enric Plaza, Juan Corchado, etc; for devoting part of their life to research in this topic, CBR.

Contents

List of Figures

Chapter 1

Introduction

The objective of this master thesis is to create a Case-Based Reasoning shell integrated in an Intelligent Data Analysis Tool, called GESCONDA which is a software developed in our research group (KEMLg[1]). Our project's aims to create a system which integrates CBR with some other techniques from artificial intelligence and statistics. This integration is done in such a way, that every technique is used by other techniques for its benefit.

CBR is gaining attention, as it does not require an explicit domain model and therefore elicitation becomes a task of gathering case histories [Watson & Marir, 1994]. On the other hand, existing Case-Based Reasoning implementations are mostly for a fixed domain. So, our propose is to create a CBR Shell where no fixed domain exists and where the expert/user creates its own domain. In this way, the task of building a new CBR application for a new target domain becomes easier.

Most of the learning algorithms can be classified in supervised and unsupervised algorithms. The supervised ones expect to get prototypes to predict new results. Clustering algorithms analyze what semblances or relations there are between instances. CBR is closer to supervised algorithms because it expects to solve a new situation.

Almost all supervised algorithms are prepared to predict the class of the new instance, so the class attribute uses to be a discrete attribute. Thus, these features restrict the real environment. The problems have to be reduced to an instance list where each instance is a set of attributes and one of these attributes is the solution that it could be a discrete attribute.

Nevertheless in CBR , or rather Data-Intensive Case-Based Reasoning, the problem is still an instance list, but the solution can be an attribute list of any type.

Currently, our research group is involved in a European Project called ModSimTex[3] where we carry the Artificial Intelligence research work. In this context, we will create a CBR system to analyze the data for achieving an approximation of the textile machine parameters.

In the ModSimTex project, the data belongs to a real domain where the problem is defined as an attribute set and each new query or new problem could

[1] This thesis is the result of the research work of the author within the Knowledge Enginering and Machine Learning Group[2]

[3] `http://www.modsimtex.eu/`

have more than one attribute. Besides, these attributes can change in every case or new situation. So, we need a system that offers the possibility to have more than one attribute as response and that the role of the attributes can change any time the user wants. Therefore, the management of some real-world domains requires a flexible CBR tools. As it will be described in section 2.3, mostly available CBR shells do not afford this skill. Thus, this is one of the motivations of our work.

Until now, we explained the advantages of CBR, but a Data Mining process normally takes some phases. One of the most known methodologies, that defines these phases and how these are connected, is CRISP[4]. This methodology remarks the importance of data preparation and modeling, and also that these two phases can be a cycle by themselves.

There are some tools that offer data preparation and learning algorithms, but if we want to use CBR, there are not available tools offering both - CBR and techniques to prepare the data.

For these reasons, we found interesting both ideas of using CBR and integrating it in a intelligent Data Analysis Tool such as GESCONDA.

The document is organized as follows:

Firstly, in next sections we explain the history of the project to understand our motivation and the goals that we want to achieve.

In chapter 2 we introduce this project background explaining what is CBR and GESCONDA. And we talk about the existing CBR shells to understand the lack of flexible and general CBR tools in this area. In chapter 3, we describe the CBR shell functionalities that we want to create and the several possible implementations. Once the project is described, in chapter 4 the design and decision made to develop the system are presented . After the whole system is explained, in the evaluation chapter we show some experiments that has been done to test the tool. A comparison of the CBR shell against the shells that has been described in related work(2.3). Finally, we conclude with the discussion of the project (chapter 6), drawing some conclusions and outlying future work directions.

1.1 Motivation

A few years ago, I has met Case Base Reasoning(CBR) (see description in background 2.1). It was introduced as a technique which is based on the same functionality as the human brains. The idea behind CBR was simple and in fact, when I thought in myself in front of a new situation, I imagine myself remembering what experiences I have to solve it.

Until that moment I had studied learning techniques which just tried to generalize in order to extract conclusions out of the analyzed data. And, in the case of supervised learning, how to classify a new instance based on the generalizations. In these processes we always talk about *success percent*. But I had never thought that instances which were not solved because they did not belong to these generalizations could have been solved by looking at similar instances in a smaller, particular domain for the new instance.

As I was thinking more and more about the subject, new questions and worries came to me. When I went to ask my advisor questions like: *"if we*

[4] `http://www.crisp-dm.org/`

create prototypes, then search can be faster" or *"why not creating a memory organization which maintains relationships between non linear cases"*, he kept answering to me that of course we could make these extensions. Thus, he proposed me some articles through I began to understand that this was a very flexible field.

I kept exploring some shells and contexts in which CBR had been applied. In help desk field it has been used a lot, with the aim of finding answers to questions already formulated, like GizmoTapper [giz, 1996] or the one developed by Compaq [com, 1992]. It has also been used in diagnostics domain, like CASEY [Koton, 1989] or PROTOS [E.R.Bareiss, 1988].

One of the CBR systems that most attracted my attention was CHEF [Hammond & Head, 1990], CHEF is a recipe adaptation system. This system implements all stages of the CBR cycle, but it is highly domain-dependent. It looks for recipes given some directions like the possibility of including a particular ingredient, or the way it is cooked. If it does not find a case which has all of the preconditions, then - providing some rules it has - it changes the most similar recipe it founds in order to fit the conditions. Per example, exchanging a vegetable for another which was a precondition for the other recipe. And it saves the recipe together with user's evaluation of it, being able to create new recipes if the recipe has been negatively evaluated.

Then, after looking to some of the shells and systems, I concluded that most of them were focused on the retrieval part. Moreover, those that implemented some phase else, they do it tied to the domain, because they are used to belong to organizations interested in analyzing and solving a concrete problem.

Then, having a bit more knowledge I saw that CBR seemed to be focused just on retrieval part, at least most of the implemented CBR systems. The rest of the phases were very domain dependent or were directly obviated.

Abstracting what CBR means, I reached the conclusion that it could be seen as a methodology where each part can be extended with different techniques that deal with improving the performance. Moreover, it caught my attention the fact that in the literature, they do not refer to a class attribute anymore, but they do to the solution which could be a more generic concept.

Coming back to supervised learning, one could think in the process as a CBR system, where the case base was the instances which got organized according to the used technique. The prototype to which a new instance belongs was searched and finally, it was adapted by copying the prototype's solution (class) to the new instance. In case of incremental techniques, they save that instance, while the non-incremental ignore it.

We noticed then that it existed the need to create a generic CBR system suitable to be used in any domain and also extensible and malleable for each domain.

We started to specify how such a system should behave and we realized that it could be more interesting if it was embedded in a larger data analysis system like GESCONDA. Doing that way, both parts could benefit each other. There is a lemma in programming: DRY ("Don't Repeat Yourself"). If there was already a system including several techniques, why not use it? At that point, we decided to focus on Data-Intensive Case-Based Reasoning - instance-based - because GESCONDA is conceived to work with instances.

We think that case structure'loss is justified because the GESCONDA functions can be applied to the case base. Besides, the knowledge available generally

when doing Data Mining is represented by instances.

That is when the idea of this project arose: *implementing a CBR which had all of its phases flexible enough to cope with the maximum number of domains and extend it to improve its functionality.* Besides, it may be used as a final system to cover all phases of Data Mining and possible become an application that could be used not only in the academic environment.

Our final target was set. That is why we have studied some generic applications to see how they tried to generalize the CBR phases (see related work 2.3). We also have made an study on some fields to see how they faced each phase. Moreover, we have looked for systems that join different techniques together. They can be found in the literature as "hybrid CBR systems".

1.2 Goals

Our target is the implementation of a new CBR module inside GESCONDA (to be precise, GESCONDA II, which is the last GESCONDA version). This implementation allows the use of all techniques at any moment. That is, although data have to be loaded as cases, they may be used also in any functionality of GESCONDA. On the other hand, if data are loaded as instances, they may be also used inside CBR. This way we have then two systems benefiting each other.

We will go into CBR in depth. We will look at which systems are already developed, what they cover and how they have been conceived. About each CBR task, we will study which solutions have been proposed and how have they been developed. In CBR part, we will try to make it as flexible as possible in order it to cover as many domains as possible or, rather to, be adjusted as much as possible to each domain. Moreover, in order not to loose the knowledge about the domain, the CBR shell should allow to specialize each configuration to a more concrete domain.

During its executions, the system should also create a text file describing what it is executing, together with the results. Moreover, once the desired configuration is known, the user will be able to use the CBR shell by command line, without the need to load the graphical interface.

Regarding the technical part, the implementation has to be highly flexible, in order to be able to keep adding more and more functionalities without the need to change the code. This implementation has to be also independent enough to be used as an API (Application Programming Interface).

Finally, the Graphical User Interface should be easy to use. Any user with the minimum required knowledge should be able to execute CBR .

Chapter 2

Background

In this chapter we introduce the reader with background knowledge about the thesis, which will help him to have a better understanding of the document.

We start with an explanation of what Case Based Reasoning is, by means of a short description of the general concept. Afterwards, all its steps and options are explained in detail.

Next, a brief description of GESCONDA system follows, allowing the reader to understand how is it designed and what it can offer. This system has its own documentation, hence we thought it is important to have just a general view which provides which type of system is and to have an idea of what is this thesis about.

Finally, we conclude with an introduction of some applications which implement CBR to see what they offer, which are their general features and lacks. We base part of our CBR system in what these applications do and how they do it.

2.1 Case-Based Reasoning

Case-Based Reasoning (CBR) [de Mantaras et al., 2006, Kolodner, 1993, Riesbeck & Schank, 1989] is a problem solving paradigm [Aamodt & Plaza, 1994]. Such a system uses the human reasoning model [de Mantaras et al., 2006] as a base. Humans make use of past experiences to resolve new situations or problems, reasoning by analogy [Ross, 1989].

The foundations of Case-Based Reasoning rely on the early work done by Schank and Abelson [Schank & Abelson, 1977], where they proposed that our general knowledge about situations is recorded as scripts. The cognitive paradigm behind the Case-Based Reasoning is based on Schank's Dynamic Memory theory [Schank, 1982], which introduces indexing as the key to use experience in understanding. The main premise was that remembering, understanding, experiencing, and learning cannot be separated from each other, and that the human memory is dynamic, and changes as a result of its experiences.

The reasoning by analogy of CBR is based in collecting a lot of relevant cases. Those cases could be considered as past experiences or solved problems [Althoff, 1999] in a particular domain. Storing a case means to keep a description of the experience as well as the solution provided to that experience. The set of stored

cases or experiences is usually named "Case Library" or "Case Base" or "Case Memory".

Case-Based Reasoning is a paradigm that - in many aspects - is fundamentally different from other major AI approaches [Aamodt & Plaza, 1994]. Instead of relying on general knowledge of the problem domain, or generalized relations between problem description and solution, CBR is able to utilize the specific knowledge of previously experienced cases. A new problem is solved by finding a similar past case, and reusing it in the new problem situation. A second important difference is that CBR is also an approach to incremental, sustained learning, since a new experience could be retained each time a problem has been solved, making it immediately available for the next problems.

The CBR formalization is summarized in the basic CBR system reasoning cycle, proposed by Aamodt and Plaza in [Aamodt & Plaza, 1994], called the "4 RE's"(figure 2.1).

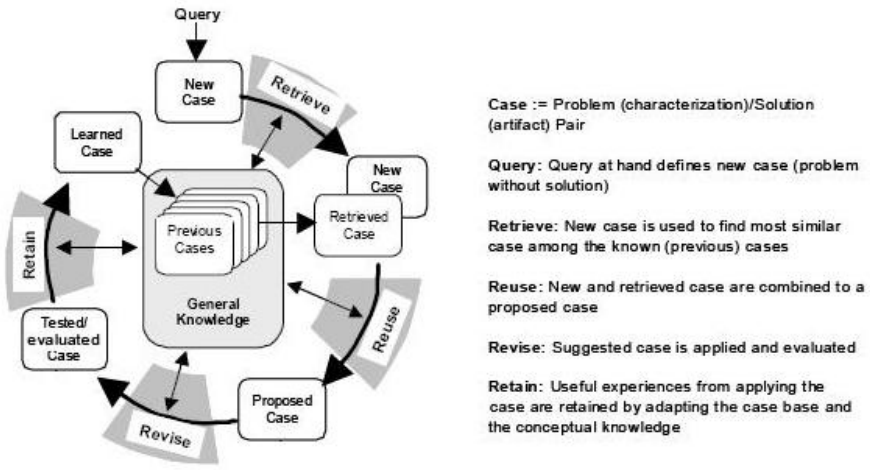


Figure 2.1: Typical CBR cycle

According to CBR, solving a problem involves:

1. Obtain the problem description
2. Measure the similarity of the current problem to previous problems stored in a case base (or memory) with their known solutions, retrieving one or more similar cases
3. Adapt (reuse) the solution of one or more of the retrieved cases, possibly after adapting it to account for differences in problem descriptions
4. The solution proposed by the system is then evaluated (revised)
5. The problem description and its new solution can be retained (stored) as a new case, and the system has learnt to solve a new problem.

Therefore, a CBR system will be efficient and reliable as long as its case base is representative of the domain, and its ability to retrieve the most similar cases as an answer to a new situation is good [de Mantaras et al., 2006].

As a matter of fact, two aspects have been more important from the definition of Aamodt and Plaza [Aamodt & Plaza, 1994]. Now, the learning or retain phase copes only with the functionality of storing or not the new experience with some criteria. These aspects are indexation of cases and case base maintenance, which are enough important to be separated in two tasks more.

The first one - indexation - is the organization of the case base, with the aim of ensuring a faster retrieval without quality affecting. The second one - case base maintenance - is a task that should be executed to ensure the coherence of the organization of the cases and tries, to maintain optimal efficiency and competence of the system [Orduña-Cabrera & Sánchez-Marré, 2008].

Some authors consider CBR as a methodology instead of a technique [Sovat et al., 2001], but there is no a clear distinction. Initially, CBR was a more specific model, but along the time the different phases of the CBR cycle have been implemented with different techniques. This can be seen, for example, in the hybrid system for forecasting presented by Corchado and Aiken [Corchado et al., 2004]. This system uses MLHL-SIM(Maximum Likelihood Hebbian learning - Scale Invariant Map) for indexing and retrieval, unsupervised kernel methods for reusing and kernel methods for learning. Therefore, the initial CBR definition has evolved and it is not wrong to think about it CBR as a methodology.

2.2 GESCONDA

GESCONDA is an intelligent data analyzer for the management of environmental data (***GES**tió del **CON**eixement de **D**ades **A**mbientals*), but it is suitable to with other domain data

GESCONDA [1] is a project under development by KEMLg. This project was funded by the Spanish Research Council(CICYT) in 2000 and 2004[2].

The main goal of the project is to design and develop a prototype tool for intelligent data analysis and implicit knowledge management of data bases, with special focus on environmental databases. The latter are remarkable due to the high amount of heterogeneous information and knowledge patterns implicit in large data bases coming from the monitoring of any system or dynamical environmental process [Sánchez-Marré et al., 2004].

Although in the literature other Knowledge Discovery(KD) tools or commercial systems exist, such as WEKA[3] or RapidMiner[4], none of them strongly integrates statistical and machine learning methods together. Besides the possibility of explicit management of the produced knowledge in Knowledge Bases (in the classical AI sense), mixed techniques that can cooperate among them to discover and extract the knowledge contained in data or dynamical data analysis in a single tool, allowing interaction among all methods [Gibert et al., 2006].

Our implementation wraps GESCONDA II [Margarit, 2007]. GESCONDA II is a new version of the original GESCONDA, with the integration of different components, redesign and implementation of new functionalities.

2.2.1 Architecture, Specification and Design

This new version of GESCONDA II has been developed in Java 5 (GESCONDA uses a previous version of Java). In addition, the graphics in the presentation layer have been drawn using a LGPL library: *JFreeChart* which allows to create powerful and adaptable graphics.

The final product has been packaged on one jar using a special tool[5] that replaces the ClassLoader from Java Virtual Machine for one that is able to decompress the project in runtime.

The design methodology that has been followed is **Scrum**[6], which is included in the agile methodologies of developing. This methodology owes its name to the concept of *Scrum* in Rugby. *Scrum* is a simplification of *eXtreme Programming* that allows to complex development or managing complex work in a short time.

GESCONDA II has followed the Model-View-Controller (MVC) architecture pattern. Along the following sections we can briefly see how is the behaviour of each layer.

Model

In GESCONDA, the database is implemented as a plain file that contains a set of instances (or observations). This file can have different formats(see at 2.2.2

[1] `https://kemlg.upc.edu/menu2/current-projects-1/gesconda-1/gesconda`
[2] TIC 2000-1011, TIC 2004-1368
[3] `http://www.cs.waikato.ac.nz/ml/weka/`
[4] `http://rapid-i.com/`
[5] `http://one-jar.sourceforge.net`
[6] `http://en.wikipedia.org/wiki/Scrum_(development)`

the functionality of GESCONDA).

Each instance is set up with a set of values belonging to an attribute, where all instances share the same set of attributes. Some of the values can be *missing* values as GESCONDA supports *missing* values

The types of the attributes could be summarized as real numbers and character streams. Formally, the instantiated types are the following:

Quantitative/Numerical: Real number, are implemented as float numbers.

Qualitative/Categorical: All the possibilities have to be specified or can be increased in GESCONDA while it's running.

Ordered The values follow an order, so the distances between two different values are dependent on the distance of the order.

Unordered No relation exits among the possible values.

As a result, GESCONDA defines an Instance as a set of attribute values. In the figure 2.2 it is possible to observe that *CMTaula* is and object that contains all the instances (*CMInstancia*) and also, a list of attributes (*CMAtribut*). Thus, the data is a list of instance, while instances themselves are a list of values (*CMValor*). Every instance has the same attributes and one value for each of the attributes. The most important classes used are the following:

CMModel Follows the *Singleton* pattern. It acts as a front-end interface among the rest of the classes. It has a executed algorithm (*CMAlgorisme*) historical. Also, it has access to the data (*CMTaula*). And finally, the current distance (*CMDistancia*).

CMTaula This class gathers together the different parts of the data, the specification of the attributes among *CMAtribut* and *CMInfoAtribut* and the data (list of *CMInstancia*).

List of *CMInstancia* It is a vector of all the instances

CMInstancia Is a vector of values (*CMValor*). One value for each attribute that is defined.

List of *CMAtribut* A vector of the attributes which belong to instances. *CMAtribut* is an abstract class, so a type has to be selected for instantiation. In this class, the attribute is described: its type, its possible values, whether is it categorical and its order, if any. It provides functions to consult the values, which are stored in the different instances, the possibility to update with a new value or resetting the information created from the values. Each *CMAtribut* will be instantiated as a subclass as it is possible to see in figure 2.2: numerical, categorical and ordered categorical. Moreover, every attribute has its extra information(*CMInfoAtribut*), which is an abstract class as well.

CMInfoAtribut This class contains extra information that can be assessed and updated while the program is running. The information about the values are: mean, mode, number of different values, variance, etc .

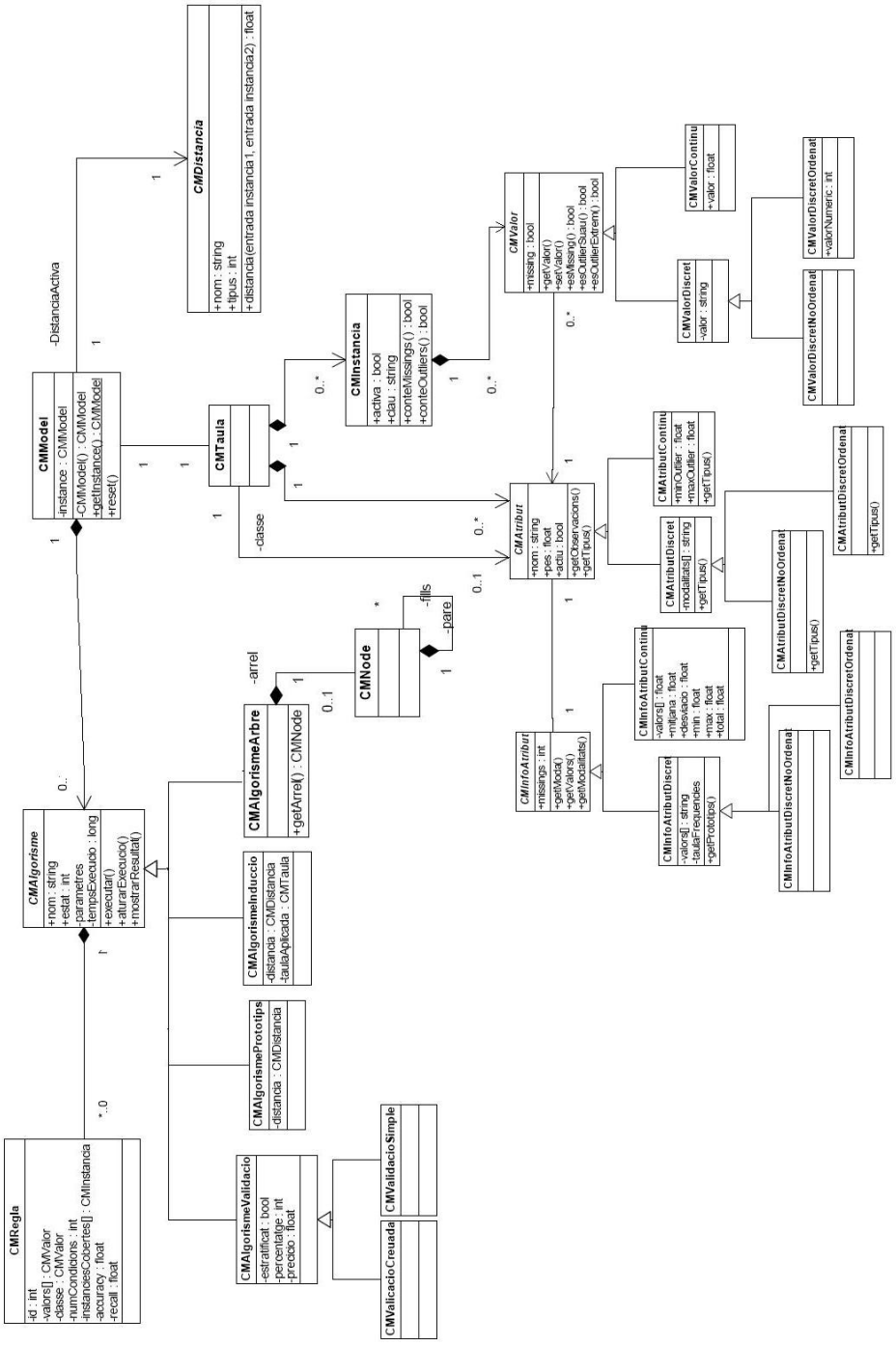


Figure 2.2: Class Diagram of GESCONDA Model

CMValor It is an abstraction of the types and contains the value and whether it is an outlier or not. Of course, it has to be specified according to the subtypes of this *CMValor*, *CMAtribut* and *CMInfoAtribut*.

Class Attribute It is an attribute of *CMTaula* that links to a categorical attribute which will act as the class for supervised learning (see 2.2.2)

CMAlgorisme This is an abstract class, provides the template of the implemented algorithms with simple functions such as execute. It also has information about the current state. In figure 2.2 is this class with its subsequent subclasses depicted. Many of these subclasses are the abstraction of different groups of algorithms and they have, in turn, subclasses implementing specific algorithms. To examine some examples about these implementations, please consult *kMeans, ID3, Rise* (see section 2.2.2)

This class is included in the model because *CMModel* has an executed algorithms historical log during the session. It also has the possibility to recover them without the need to execute it again in the case data are the same.

CMDistancia It is also an abstraction. Some measures are developed as *Euclidean, Manhattan, Canberra, L'Eixample*, etc. (please refer to appendix A for an explanation about each distance function). This class is in the model because there is an active distance stored in *CMModel*.

One of the special characteristics is that most of the results of the algorithms are included in the data as a new variable. In the instances, new qualitative variables are generated by the clustering algorithm, thus classifying them into the different clusters, and also allowing to visualize the data base into prototypes.

Rule inductive algorithm generates a rule set that can be applied later on the database. These rules must be modeled for ensuring the correct treatment and persistence. Rules can be exported to the CLIPS format.

Decision tree algorithms generate a tree. A tree is a set of nodes and father-child relations. Each node contains information about the attributes and conditions that are applied to itself.

View

In this section we explain the main features selected to develop the CBR module. However, it is out of the thesis scope to study in depth the details of the architecture, design and implementation of GESCONDA.

Figure 2.3 shows the class diagram describing the main classes in this layer. *Desktop* is a unique instance (Singleton pattern) that can be to access from any part of the code. In figure 2.4 we can see the visualization of this class. On the left there is a variable view, and the rest of the screen is organized by tabs. All of the components use *Observer* pattern to be notified when a change occurs in the model.

As it has been said before, the language used is Java, using Swing libraries for the views and *jFreeChart* for graphics and data representation.

The application menu is dynamically built from a *xml* file. This could be the main feature when including the CBR module with almost no modification of the code.

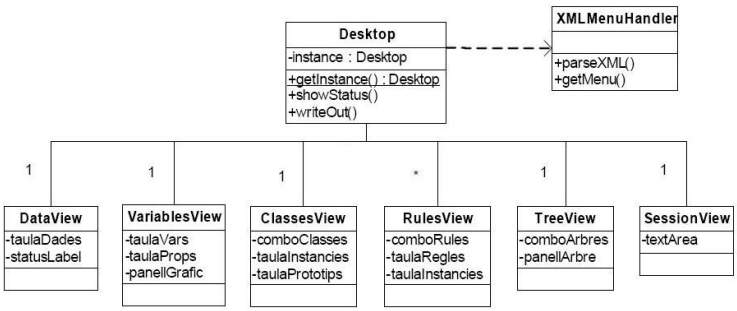


Figure 2.3: Class Diagram of GESCONDA View

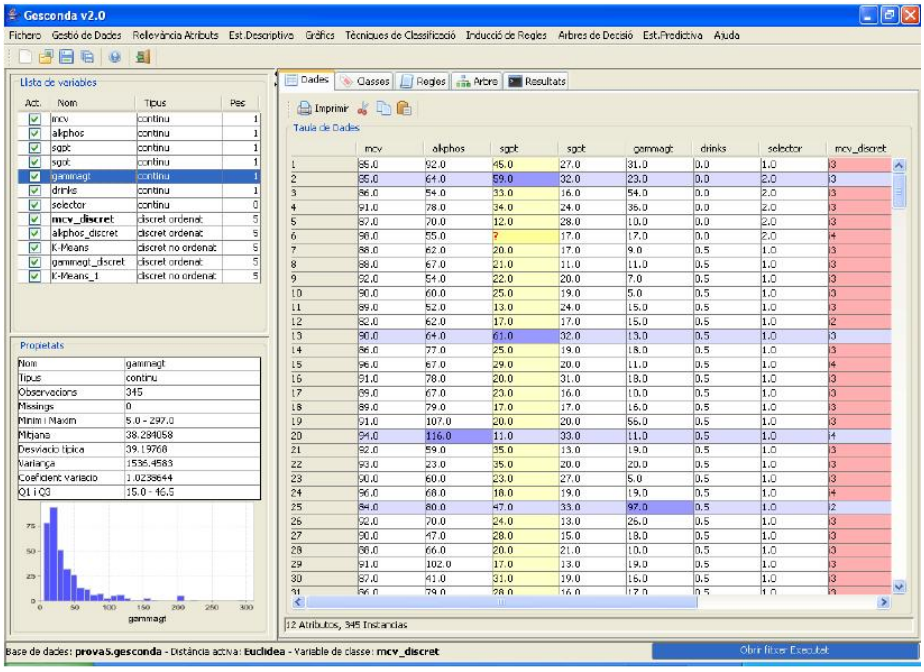


Figure 2.4: Screenshot of GESCONDA GUI

Controller

The Controller layer of the MVC model is represented in GESCONDA as an action set providing functions for showing dialogs, executing algorithms and notifying views.

Most of the actions are linked to the menu through the *xml* file(see previous section 2.2.1). Besides, a set of actions manage the algorithm executions in the stack of the application. For this propose, each algorithm is launched in a new thread.

2.2.2 GESCONDA Functionality

To explain the functionality, we divide the functions into logical groups. On the basis of previous experiences, GESCONDA as a multi-layer architecture of 4 levels has been designed connecting the user with the system or process [Gibert et al., 2006], as we it ca be seen in figure 2.5. These 4 levels are the following: Data preparation, recommendation and meta-knowledge managment, knowledge discovery and knowledge management.

In order to manage the data, typical functions are provided: open, save, close, import, export. Different formats are supported:

GCDA This format is called "GESCONDA project" because it is possible to store more than the data. For instance, the algorithm results like rules or trees can be stored. It is *xml*-structured.

GSP GSP is the old format of the first GESCONDA version. It is created with the Java serialization.

CSV Delimited by semicolons.

TXT Text format with the data delimited by tabs.

DAM Data and Meta-data allows to store the data and more related information, like the attributes and their weights.

GESCONDA II offers all the functions to create the data from scratch. In addition, when either the structure of the data is specified or the attributes are defined , it supports copying the data from an excel sheet. To facilitate the visualization of the data missing values, soft and hard outliers are highlighted in different colors.

About data filtering, GESCONDA II provides tools for manipulating the data and its attributes. The function provided are the following:

- Creating a new attribute
  - from scratch
  - as a constant
  - as a sequence
  - as a random variable following one of these distributions: Bernoulli, Binomial, Discrete, Uniform, Poisson, Normal and Exponential

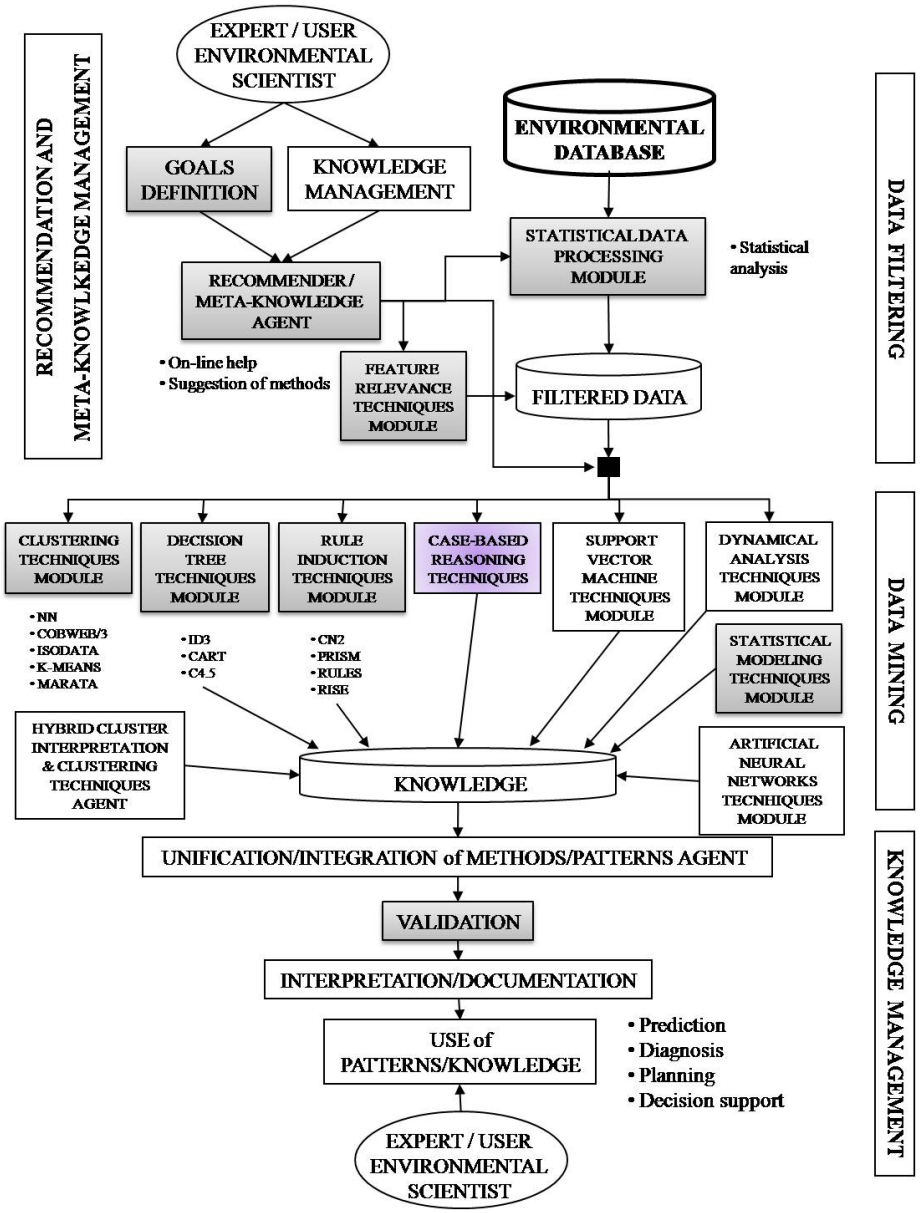


Figure 2.5: Architecture of GESCONDA

- Deleting an attribute
- Defining the Class attribute.
- Discretization of an attribute:
  - custom
  - equidistant
  - based on boxplots [Gibert & Perez-Bonilla, 2006]
- Standardization of a specific attribute, thus creating a new one.
- Replacing missing values.
- Managing outliers, both soft and hard ones.
- Defining the active distance.
- Calculating the mean distance among all the instances.
- Relevance techniques to weight the attributes:
  - custom
  - Supervised
    - EBL (Entropy based local weighting) [Núñez et al., 2003]
    - PROJ (Projection of Attributes)
    - IG (Information Gain)
    - CVD (Class Value Distribution) [nez et al., 2003]
  - Unsupervised
    - UEB(Unsupervised Entropy Based)-1
    - UEB(Unsupervised Entropy Based)-2 [Núñez & Sánchez-Marré, 2004]
    - Gradient descent
- Graphical visualization:
  - bar diagram
  - histogram
  - TSplot
  - Bivariant plot (XYPlot)
  - Letterplot

About Recommendation and Meta-Knowledge Management, some tools are provided by GESCONDA for recommending a proper way to face the analysis in order to extract the more useful knowledge regarding the concrete problem to be solved.

- Problem goal definition
- Method suggestion
- Parameter setting

In Knowledge discovery, GESCONDA offers some other tools which apply techniques from Artificial Intelligence and Statistics:

- Clustering (Machine Learning and Statistical):
  - k-Means
  - Nearest-Neighbour
  - Marata
  - Isodata
  - Cobweb
  - Bagging strategies
- Decision tree induction:
  - ID3
  - C4.5
  - CART
- Classification rule induction: option to export(*txt* or *CLIPS* format)
  - Rules
  - Prism
  - CN2
  - Rise
- Statistical Modeling:
  - Linear regression
  - Variance Analysis: one and two factors

In Knowledge management, once the algorithms are applied, GESCONDA allows to use and validate the solutions:

- Integration of different knowledge patterns for a predictive task, or planning, or system supervision.
- Validation of the acquired knowledge pattern.

2.3 CBR Tools

In the literature it is possible to find some tools for the CBR system development. Some of them are developed by commercial companies and others are deployed by research university groups. In the next subsection we describes some of these CBR shells with the goal to know what is currently offered and what is our contribution with the thesis presented in this document. As a matter of the fact, these systems are Data-Intensive, although some of them could be considered as Knowledge Intensive CBR shells. In the literature, it is distinguished between Data-Intensive CBR (DI-CBR) and Knowledge-Intensive CBR (KI-CBR). DI-CBR systems share characteristic features such as the use of learning from examples, sparse use of domain and simple case structure, etc. Nevertheless, KI-CBR share characteristic features such as the intensive use of domain knowledge, complex case structure or developments based on ontologies and description logics, etc.

We will explain first the academic shells and then, the commercial ones.

2.3.1 Caspian

Caspian has been built at the Center for Intelligent Systems (University of Wales). [7]

This Shell comes with its own language for defining cases (CASL[8]) and creating the database.

The general structure of the case file described by CASL [cas, 1995a]:

Introduction Contains an introductory text which gets displayed once the program has finished checking the case file.

Case Definition Defines the types and weights of the attributes that may appear in a case.

Index Definition Defines the attributes used as indexes when searching for a matching case. At least one is needed and its type must be an enumerated type.

Modification Definition Defines the modification rules for providing a means of:

- specifying that certain symbols or numbers are similar (matching purpose). In the case of the number, this is done by specifying ranges.
- specifying symbols as abstractions of others (useful for making the search more general or for defining generalized cases).

Repair Rule Definition Contains the repair rules. Both the Modification Definition and the Repair Rule Definition may be omitted.

Case Instance Such instances are the ones that make up the case base.

[7] `http://www.aber.ac.uk/compsci/Research/mbsg/cbrprojects/getting_caspian.shtml`
[8] CASL is a language used for Case-Based Reasoning

The CBR cycle starts when the user specifies a new case. In this process, it is possible to skip some attributes but not those that are used for indexing.

Then the retrieve task, which is separated in two subtasks [cas, 1995b]. The first one performs the index search, the program searches the case base for the subset of cases which exactly matches all the index constraints. The second subtask consists in selecting one case. The user can do it manually or let the program scan the subset of cases to find the one with the highest weight value. That is, calculating the greatest sum of the weights of all the attributes that matched. Finally, the attributes which do not match exactly but are defined to be similar by the modification rules, return a value which is three-quarters of the attributes's normal weight.

The next step after selecting the case is to apply the repair rules to it, in the same way that is done in ESTEEM (see section 2.3.9).

In the aftermaths of the retrieved case when it is repaired, two things happen. First, the original values in the problem section are replaced by new values which have been entered by the user. Then the repair rules are applied. These rules examine the problem fields and the solution fields of the retrieved case for certain combinations of values that would cause problems. If this is the case, then the repair rule fires and changes are made to the solution part of the retrieved case. When Caspian applies the repair rules, it cycles through each of them in order until no more repair rules fire. A rule is only allowed to fire once. Since it is possible for one set of changes to cause another one to fire, it is possible to decompose the repair rules, so that single rule can tweak the changes made by a previous rule activation.

2.3.2 jCOLIBRI

JColibri built by GAIA(Group for Artificial Intelligence Applications).[9]This is a complete CBR development architecture supporting many features like graphical interfaces, description logics and ontologies, textual CBR, evaluation, etc. Nowadays, a new version exists (jCOLIBRI 2) which includes mechanisms to Retrieve, Reuse, Revise and Retain cases and is designed to be easily extended with new components. It offers two basic methods in reusing. The first one is copying and the second one is a numerical direct proportion among the attributes of the query and the case. This is a system for building CBR applications that is an evolution of previous work on knowledge intensive CBR. The first version of COLIBRI1 was prototyped in LISP and was far from being usable outside of the research group which developed it. jCOLIBRI was designed as a technological evolution of COLIBRI which incorporates, in a 3-tier architecture, an object-oriented framework in Java and a number of GUI-based tools for assembling a CBR application from reusable components. We are going to concentrate on the jCOLIBRI2, that this year has been considered the main framework to develop a cbr system[10].

We introduce jCOLIBRI2 in more detail in the following sections:

[9] http://gaia.fdi.ucm.es/projects/jcolibri/

[10] http://www.madrimasd.org/informacionidi/noticias/noticia.asp?id=37686&tipo=g

Retrieval

At this point, the configuration and precycle is completed. This step obtains the most similar cases given a query. The main method is nearest neighbour numeric scoring comparing attributes. It uses global similarity functions to compare compound attributes (CaseComponents) and local similarity functions to compare simple attributes.

The selection of the cases to use in the adaptation can be performed by means of two criteria. The first one consists in selecting the most similar and the second in choosing those that are more diverse. jCOLIBRI2 offers methods for both of them.

Reuse

The reuse step adapts the solution of the retrieval cases to the requirements of the query. jCOLIBRI2 considers this task is very domain dependent and leaves this step open to developers. Anyway, it offers the following methods:

Copy This method copies the value of an attribute in the new case.

Direct Proportion Performs numerical direct proportion among the attributes of the query and the case.

Besides these methods, a package is included that contains some classification reuse methods implemented by Lisa Cummins & Derek Bridge (University College Cork, Ireland). On the other hand, ontologies can be used to guide the adaptation of the cases.

Revise

In the revise step the proposed solution is tested for success. For instance, by being applied to the real world environment or evaluated by a domain expert, and repaired if failed.

This step is also very domain dependent and may change among applications.

Retain

jCOLIBRI2 just offers an option to store the new cases, but there is no automatic mechanism to decide whether to store them or not.

jCOLIBRI2 also presents a textual CBR, but there does not appear to be a consensus about the structure of a textual CBR system, mainly due to the different application domains. For classification applications typically only a basic stemmer algorithm and a cosine similarity function is needed, while with other applications more intense NLP derived structures are employed [Recío-García et al., 2008].

jCOLIBRI2 includes several other features implemented by external contributors. We briefly describe them:

- Visualization of a Case Base. This tool computes the distance among all cases and visualizes them according to that distance.
- Classification and Maintenance. jCOLIBRI2 includes several methods in order to improve the CBR performance or its accuracy. It's been developed by Lisa Cummins and Derek Bridge (University College Cork, Ireland).

2.3.3 IUCBR

The Indiana University Case-Based Reasoning Framework (IUCBRF)[11] is a freely available open-source framework, written in Java, to facilitate the development of case-based reasoning (CBR) systems [Bogaerts & Leake, 2005].

A case can be defined as a set of attributes, each one belonging to one of the following groups:

- Problem
- Solution
- Inactive Contexts
- Use Counts
- Time Of Creation
- Source

Both a problem and a solution consist mainly of a feature collection.

A case also maintains a *use count* and a *successful use count*. Each time a case is retrieved, its use count is incremented. If the resulting solution is deemed of high quality, then each contributing case's successful use count is incremented. This data can be useful for maintenance purposes.

Cases record the point in the system's history they were added (the *time of creation*). This is recorded in terms of the number of problems previously processed. This data can be useful for maintenance purposes. For example, older cases may be considered more likely to be out of date and targeted for removal.

Each case is also associated with a source. For example, if a case came from a successful problem solving episode, then the case source would be "system-generated". More specific sources can also be created. Source data may be useful in maintenance.

The attribute types that IUCBR support are the following:

- Double
- Integer
- String (with some special handling for long strings)
- Boolean
- The set {"Yes", "No"}
- The set {"very mild", "mild", "moderate", "severe", "very severe"}
- Term vector
- Internet address

Any customized attribute type can be created by extending an existing abstract feature.

The case base is indexed by one of the two following simple indexing schemes:

[11] `http://www.cs.indiana.edu/~sbogaert/CBR/`

- Flat case base
- B-tree-backed case base

The framework provides the option of storing the cases using a B-tree-backed case base. With this approach, selected features are identified by the system designer as indices to be held in memory and used for similarity comparisons in retrieval. When the most similar cases are identified according to these indexes, the full cases are retrieved from a B-tree-backed file.

About the CBR cycle, each part is implemented and it's easy to be extended:

Retrieval To find the most similar cases, a k-nearest-neighbor (k-NN) algorithm is implemented.

Reuse IUCBRF provides the following small set of domain-independent case adaptation techniques:

- No Adaptation: Simply returns the solution of the first retrieved case as the system solution.
- Weighted Average: Given a list of cases and weights, returns a weighted average of the solution values as the adapted solution.
- Weighted Majority: Given a list of cases and weights, takes a weighted vote and returns the winning value. In the event of a tie, whichever value is associated with a higher-ranked case is declared the winner.

The weights used can either be provided statically, or be determined dynamically in each problem solving episode. One example of dynamic weighting are the distances of retrieved cases (saved in the retrieval step), with closer neighbors given higher weights.

Retain The framework provides simple facilities for case base maintenance include case removal and addition. Triggering mechanisms for both operations can be customized, and two implementations currently exist. One simple implementation, "null maintenance", never adds nor removes any cases. Another implementation, "basic maintenance" periodically checks for infrequently-used cases to remove, and accepts for addition any new case corresponding to a problem that was just solved successfully.

IUCBR also provides a CCBR (Conversational CBR) system that incrementally refines case selections by interacting with the user.

Case List Refiner It determines the contents and order of the list of potentially useful cases. The one case list refiner that is implemented orders the cases depending on the difference between each case and the current problem.

Question Selector It determines which question should be asked next in the conversation. The framework contains three implementations :

Ordered Selection It simply asks the next question on the list.

Flow Chart Selector It determines the next question by asking a flow chart provided by the designer.

Frequency This selector determines which unknown attribute in the current problem has a known value in the largest number of cases under consideration. The question corresponding to this attribute is the next question asked.

2.3.4 AIAICBR

AIAICBR Shell Deployed by the Artificial Intelligence Applications Institute[12]. The information about this system is scarce and in its home page is detailed how to use and focused on genetic algorithms.

This shell is a generic tool for CBR . The structure of the case restricts the solution to a unique nominal attribute.

The tool performs classification based on case comparison. The parameters of the algorithm to define are the number of nearest neighbours and the weights that can be set up manually, or optimized by a genetic algorithm. The accuracy of the algorithm is measured by a leave-one-out-evaluation.

Genetic Algorithms optimize the weight structure. Each chromosome represents the weights of each attribute or field belonged to a case. This algorithms don't alter the value of k in nearest neighbour.

In order to run this algorithm, filling the following parameters is needed:

- number of chromosomes
- mutation rate(%)
- word length(bits)
- mapping from bits to weights can be configured

On the other hand, the file data structure must be in comma delimited form, newline delimiting a case. The first line of the case base must contain the name of the key file while the second one states the goal field (i.e. the class to which the data/record/case belongs). Only one attribute can belong to the *case solution*

Furthermore, it is possible to modify a file that defines the type of matching that is done on each field (attribute) in each case. The matching types include:

- numerical comparison by evaluating the ratio of 2 numbers
- comparison of strings/sentences/paragraphs by tri-gram matching

2.3.5 *myCBR*

MyCBR[13] is a project at German Research Center for Artificial Intelligence (DFKI) [Stahl & R.Roth-Berghofer, 2008]. They built a case based reasoning tool that can be used as standalone application as well as a *Protégé*[14] plug-in. *myCBR* consists of the four following modules:

[12] `http://www.aiai.ed.ac.uk/`

[13] `http://www.mycbr-project.net`

[14] *Protégé* is a free, open source ontology editor and knowledge-base framework (see `http://protege.stanford.edu/`)

Modelling Tools: These tools extend the existing functionality of *Protégé* for creating domain models and case instances and adding the missing functionality for defining similarity measures.

Retrieval GUI: The retrieval GUI provides powerful features for analysing the quality of the defined similarity measures. Moreover, it can also serve as the user interface of first prototypical CBR applications.

Retrieval Engines: For executing the similarity-based retrieval, different retrieval engines are provided.

Explainer: A dedicated explanation component provides modelling support information as well as explanations of retrieval results for quicker roundtrips of designing and testing.

To create a new CBR application [Zilles, 2009], the starting point is the collection of the data. That data is mainly intended for structural CBR that make use of rich attribute-value based or object-oriented case representations. Data can be imported from a CSV file and specify a class in the Class Hierarchy of the *Protégé* . Then the data rows become instances of this class.

About the CBR cycle, *myCBR* is focused in the retrieval task. As has been introduced, they present a functionality to model the similarity measure, both global for calculating the final similarity value and local for each attribute:

$$Sim(q, c) = \sum_{i=1}^{n} w_i * sim_i(q_i, c_i)$$

Retrieval Engine

After defining all necessary similarity measures, the next step is to define the query, or rather, the new case. This can be done manually or using one from the case base.

Finally, the system shows the most similar cases to the new case according to the defined similarity measures.

Similarity possibilities

The global similarity ($Sim(q, c)$) for the current class could be calculated using one of the following approaches: weighted sum, euclidean, minimum, maximum. For each row, similarity measure options are specified for an attribute:

- whether it is discriminant or not
- a weight (range which is unbounded)
- local similarity measure

The local similarities (sim_i) depend on the attribute type:

Numerical The user can choose between some typical and adjustable functions or define a minimum and maximum value and optional similarity points, in order to obtain a linear interpolation over all similarity points.

Symbolic The user can specify all the similarity values between each pair of possibilities, treat them as numerical - giving an order - or create a taxonomy for performing automatic similarity calculations.

Text User select word or character based and after, those can be configured.

Set Depending on the chosen settings, the mapping between new case values and case values is calculated differently. For example, one value out of the set or by configuring a similarity for each value.

2.3.6 INRECA

The acronym INRECA stands for "**IN**duction and **RE**asoning from **CA**ses" and it is the name of a European consortium that jointly executed two large CBR projects named INRECA (1992 - 1995) and INRECA-II (1996 - 1999). The projects have been funded by the European Commission's ESPRIT program, as part of the 3rd and 4th funding framework.

In spite of this is not a generic shell, we believe it's worth the effort, because it's a methodology that has been a reference for a wide range of CBR tools.

The consortium consisted of the following partners and some of the applications developed during the project [Bergmann, 2001]:

- AcknoSoft (now renamed to Kaidara)[15]
  - A maintenance system to support troubleshooting of the CFM 56-3 aircraft engines for the BOEING 737.
  - A helpdesk system for supporting the robot diagnosis procedure at the after-sales service for SEPRO Robotique.
  - At ALSTOM, an application for improving train availability to optimize operating cost.
  - A rapid cost estimation application for plastic parts production.
  - For Odense Steel Shipyard, AcknoSoft developed an application that integrates with multimedia tools for improving the performances of ship welding robots.
- IMS (now renamed to IMS MAXIMS)[16]
  - Application for assessing wind risk factors for Irish forests at COILLTE[17].
  - At Analog Devices, IMS developed an operational amplifiers product catalog application using the tools and the support provided by AcknoSoft, tec:inno, and the University of Kaiserslautern.
  - A CBR expertise knowledge base at Irish Research Scientists Association.
- TEC:INNO (now renamed to Empolis Knowledge management GmbH, which is part of the Bertelsmann Mohn Media Group)[18]

[15] www.kaidara.com
[16] www.imsmaxims.com
[17] http://www.coillte.ie/
[18] http://www.empolis.com

  - A tourist information system on the Internet for the region of Müritz (Germany).
  - For Siemens, TEC:INNO developed the SIMATIC Knowledge Manager which provides service support for the SIMATIC industrial automation system.
  - For the Institute of Microtechnology in Mainz (Germany), it developed the compendium "Precision from Rhineland-Palatinate".

- University of Kaiserslautern, Artificial Intelligence - Knowledge-Based Systems Group [19]
  - They conducted a feasibility study (together with DaimlerBenz) for applying INRECA technology to the task of supporting the reuse of object-oriented software.

- DaimlerBenz (now renamed Daimler)[20]
  - The main INRECA-II application called HOMER was developed by TEC:INNO at Daimler. HOMER is an intelligent hotline support tool for CAD/CAM workstations.

Nowadays, Kaidara (Acknosoft) offers *Advisor* that is a helpdesk system based on INRECA. Empolis (TEC:INNO)developed *empolis orenge* [Schumacher, 2002] based in other tools as CBR-Works [Schulz, 1999], CBR-Sells and CBR-answer.

Empolis Orenge[21] is a flexible and scalable Case-Based Reasoning shell for industrial applications that also contains many components which provide functionality beyond the basic CBR paradigm. Orenge consists of a set of components or services which provide different solutions for CBR :

- retrieval engines: CRN and database retrieval.
- text mining component for information extraction.
- rule processing system for completion and reuse task.
- data import from a document or database

The editing of models and configuration files is supported by Orenge: Creator, which provides graphical user interfaces to edit the models and contains wizards to assist the user in this task.

Briefly, we explain the four points on which INRECAis centered are introduced: vocabulary, similarity measures, solution transformation and case base.

Case Representation (Vocabulary)

INRECAuses an object-oriented technique for case representation. This representation is appropriate for complex domains in which casex with different structures exist. Each case is represented by an object which is a collection of

[19] `http://www.dfki.de/`
[20] `www.daimler.com`
[21] `http://www.km.empolis.com`

attribute-value pairs itself. The structure of an object is described by an *object class* that includes the set of attributes together with a *type* (possible values or subobjects). The attributes can be distinguished between simple or relational. The last ones represent a direct binary relation and are used to represent complex case structures.

Moreover, these *object classes* can form a hierarchy where subclasses inherit from the parent classes.

The object-oriented representation has been implemented in CASUEL[22], which is a Common Case Representation language developed within the INRECA project in 1992.

CASUEL is the interface language between all the INRECA component systems [Bergmann, 2001], but it is also intended to serve as the interface language between the INRECA integrated system and the external world, and as a standard for exchanging information between classification and diagnostic systems which use cases.

CASUEL is a flexible, object-oriented frame-like language for storing and exchanging descriptive models and case libraries in ASCII files. It is designed to naturally modelling the complexities of real cases. CASUEL represents domain objects in a class hierarchy using inheritance, the slots used to describe the objects, typing constraints on slot values, as well as different kinds of relationships between objects. CASUEL additionally supports a rule formalism for exchanging case completion rules and case adaptation rules, as well as a first mechanism for defining similarity measures.

Similarity Measures

The objective is to determine the similarity between two objects, so-called *the global similarity*, whereas the distance between two attributes is *local similarity measure*. Lastly, for each relational slot, an object similarity (global) measure recursively compares the two related sub-objects.

Then, the similarity values from the local similarity measures and the object similarity measures, respectively, are aggregated to the object similarity between the objects being compared. This may be done by using a weight model.

This initial approach to similarity, however, did not specify how the class hierarchy influences similarity assessment. In INRECA-II a framework was developed for object similarities that allowed to compare objects of different classes while considering the knowledge contained in the class hierarchy itself.

The model presented afterwards distinguished between an inter-class and an intra-class similarity measure [Bergmann & Stahl, 1998]:

Intra-Class Similarity The common properties of the two objects can be used to the intra-class similarity. For doing this it is necessary to take the most specific common class of the two objects and to compute the similarity based on the attributes (together with its weights or not) of this class only.

Inter-Class Similarity It is important to note that the difference between two objects is not represented by their shared attributes but by the structure of the class hierarchy. Therefore, inter-class similarity represents the

[22] `http://www.wi2.uni-trier.de/de/cms/projects/CASUEL/CASUEL2_toc2.04.fm.html`

highest possible similarity of two objects, being independent of their attribute values, but dependent on the positions of their object classes in the hierarchy. Formally, the inter-class similarity is defined over the classes of the objects from the query(new case) and case being compared.

The final object similarity between two objects can then be computed by the product of the inter- and the intra-class similarity.

Solution transformation

In many situations, additional general knowledge is required to cope with the requirements of an application. In INRECA, such general knowledge is expressed by three different kinds of rules:

Exclusion rules are entered by the user during consultation and describe hard constraints on the cases being retrieved.

Completion rules are defined by the knowledge engineer during system development. They describe how to infer additional features out of known features of an old case or the current case.

Adaptation rules are also defined by the knowledge. They describe how a retrieved case can be adapted to fit the current case.

2.3.7 CBRExpress

CBR Express, produced by Inference Corporation, has been one of the most successful CBR tools. Inference's CBR Express is a tool with a comfortable user interface, primarily designed for help-desk applications and after, extended for similar interactive selection tasks [Althoff et al., 1995]. It is itself an application written with Inference's ART-IM expert system toolset, and can be extended with this toolset [3cb, 1992]. The user interface is written with Asymetrix ToolBook, which makes it somewhat slow but malleable.

CBRExpress works well in domains that can be represented by a set of vectors of pairs attribute-value. If domain modelling requires background knowledge such as rules, formulas, constraints or taxonomies, CBRExpress tool is not the best choice, unless it's used as a component of another application, instead of using it as a standalone application [Althoff et al., 1995]

A key feature of CBR-Express is its ability to handle free-form text. This was felt to be vital to the help desk market since it lets customers describe their problems in their own words rather than being taken through a decision tree style question and answer session. CBR-Express ignores words such as: and, or, I, there, etc., it can use synonyms, and represents words as a set of trigrams [Watson & Marir, 1994].

Originally, they wanted a large case base to cover more problems. Instead, some sample cases were constructed, with proper questions. With a smaller, hand-built case base, it's easier to avoid redundant cases differing only by a word or two. One important task was creating a synonym list. In addition, they created a style guide to force consistency on the four developers and eventually four domain experts (senior support reps) who create cases for the system. Questions get default weights from CBR Express, or the builder-user can weight them according to relative importance, including absolute scores.

CBR Express includes automatic heuristics to reduce the search space to a subset for most queries. Moreover, a hierarchical system is fairly rigid in its classifications and expects correct answers; nearest-neighbor is extremely tolerant of inaccuracies and missing data - especially appropriate for help-desk problems.

The system is good enough to present the right answer to the operator with a couple of iterations of the process. The point of CBR Express is to make it simple, and it is in its basic form without ART-IM extensions. The problem with CBR Express is that it's so simple it misses some powerful constructs.

A typical CBR Express process: The user starts by typing in a short problem description. The first section of the runtime module uses simple text-search techniques to rank probable cases by matching the significant words and trigrams, throwing out (temporarily) any cases with no words in common with the input case. This is similar to most natural-language categorization or database front-end systems, which just throw out extraneous words, anyway - as is opposed to parsers, which depend on grammar and more verbose formulations. The top-ranking cases/questions are then displayed to the user. Obviously, the operator has to enter relevant information, using words that discriminate well between cases. Whereas a direct user of a public system might type a lot of irrelevant stuff, an experienced operator will type only terse, useful phrases, such as "Printer failure", whatever the customer on the line might say. In the end, the system depends on sensible operators, and a particular case base contains a limited domain. Now the user must answer some of the questions listed with the top-ranked cases; usually five or ten of them are displayed, a number set by the user.

The questions are listed in order of weight for the highest-ranked case. The user answers whichever question prefers. While there's a large, possibly proliferating number of discrete, unique cases, the system's efficiency is maximized and the user's time is minimized if questions are kept to the minimum. The more cases a single question can discriminate among, the better.

Using the new answers, CBR Express conducts a second search, to find cases that match the answers given for the input case. It scores a subset of the case base again, allocating an optional proportion of the score (typically 20 percent) to the description match, and the rest to the values of answers to the questions. Once again, it produces a list of the highest-ranked matches to the target case. At this point, the operator can select a case and see the recommended action, or answer further questions based on a new set of questions that appears, corresponding again to the highest-ranking possibilities.

Once again, the system adjusts the scores of the cases involved, and produces a new list. This goes on until the system flashes a match, the user gives up, or the system declares that nothing matches the user's answers. In that case, the user tags the new case as "unresolved," saving the description and all the questions and their answers. The case is forwarded to an expert who solves it, and either determines that it is in fact an existing case or enters it as a new case. The expert just stores it with a new name, along with any new questions and their answers.

Special characteristics of CBRExpress retrieval are that it is fast. This becomes important for case bases with thousands of cases where lack of an inductive component may become a problem. And another characteristic that makes a difference from others applications, it is the solution that is not just an

unique attribute, else a data record that associates additional information with the case.

CBRExpress distinguishes between user and maintenance mode. Only the maintenance mode allows to changes of the similarity measure, entering questions o reindexing the case base. In user mode, most of the windows are removed to prevent the user from destroying the case base. The menus are reduced to only those features necessary for the retrieval.

In summary, CBR-Express is extremely well suited to help desk applications and has also been used successfully for intelligent task assistance, information access systems and knowledge publishing. It is very easy to use, reliable, network ready, and notable for its intelligent handling of text.

2.3.8 ReMind

ReMind, produced my Cognitive Systems Inc., was developed with support from the US DARPA programme. It is available as a C library to be embedded in other applications, as well as an interactive development environment [Watson & Marir, 1994].

Cognitive takes a hybrid approach, combining decision trees and template matching with nearest-neighbor searches, and stresses its flexibility as a competitive advantage. Generally, matching cases can be retrieved using the decision tree. The answer to each question rapidly eliminates a huge portion of the case base and allows the search to focus on a tighter and tighter section of the case base. When a selection of cases is retrieved, or when there is no exact match, nearest-neighbor searching comes into play.

ReMind offers template, nearest neighbour, inductive, and knowledge-guided inductive retrieval that can be done automatically with no user involvement or the user can create a qualitative model to guide the induction algorithm. The template retrieval supports simple SQL-like queries returning all cases that fall within set parameters. The nearest neighbour retrieval is informed by user defined importance weightings that can be placed on case features. Inductive retrieval involves building a decision tree that indexes the cases. This can be done automatically by ReMind with no user involvement or the user can create a qualitative model to guide the induction algorithm.

Qualitative models are created graphically to indicate which concepts (case features) are dependent on other concepts. Qualitative weightings can be placed on these dependencies. Then, ReMind uses the qualitative model to guide the induction algorithm (hence knowledge-guided induction) resulting in decision trees that more closely reflect the causal relationship of concepts in the cases. Interestingly, different qualitative models can be created to explore different theories about the domain or to allow what-if questions to be asked.

Case adaptation is provided by creating adaptation formulae that adjust values based on the difference between the retrieved and the new case. These are also graphically created using a visual programming technique. Although it takes a little in getting used to the extremely close typing of case features combined with the close typing of the operators, it does reduce syntax errors. Finally, ReMind can create case bases from existing databases making it potentially useful for data mining projects.

Cognitive's ReMind is closer to a toolbox than a tool [3cb, 1992]; it lets the savvy user-builder manipulate data and cases with just about every version of

CBR technique around, including many techniques described above. You can build an extremely clever system if you know what you're doing, and you can also make it fairly intelligible to a normal end-user if you have the interface-building skills and sensibility - but the system doesn't do it for you. Where ReMind shines is the underlying technology of matching, indexing and induction, and the richness of the development environment.

It's hard to describe the typical ReMind system, since the tool lets the builder-user build almost anything. Customers so far include some government agencies, but also commercial companies such as Motorola (help-desk, warehouse management to find appropriate parts and computer security auditing), Boeing (engineering decisions and production scheduling), British Airways (747 maintenance), American Express (to detect when its small-business accounts are spending out of pattern), Nestlé, Barclay and NCR. The firm also has a long list of prospects, many of which it has been done research for.

2.3.9 ESTEEM

ESTEEM, from Esteem Software Inc.[23], is written in Intellicorp's Kappa-PC [24] It supports applications that access multiple case bases and nested cases. This means that one can reference another case base through an attribute slot in a case. ESTEEM supports various similarity assessment methods including feature counting, weighted feature computation and inferred feature computation. It can also automatically generate feature weights, either using an ID3 weight generation method, or a gradient descent weight generation method. Moreover, users can incorporate their own similarity functions into ESTEEM.

ESTEEM is very reasonably priced and is well suited as a teaching tool or an entry level CBR product. That's the reason why Esteem is one of the major commercial CBR tools.

It's considerated an expert system, focusing not just on case retrieval but on generally rule-based case adaptation. In this respect, although it's a less ambitious system, it's closer to ART-IM than to either ReMind (see 2.3.8) or CBR Express(see 2.3.7). One other distinguishing feature is Esteem's ability to nest cases, so that a case can derive a feature from cases in another case base.

The Esteem product grew from a number of customer-specific applications. One example is the Bidder's Associate, currently in use by salespeople at Enginetics, an aerospace parts manufacturer in Dayton, Ohio. Those salespeople use spreadsheets to record and manage their bids for parts manufacturing contracts. Basically, the spreadsheet is a smart form into which salespeople enter the customers' requirements and do a few calculations; then they draw up a proposal. The requirements and constraints all follow a similar pattern, but each set is unique; moreover, some win bids and some lose bids. The Esteem system took its spreadsheet, with its significant features in it, as the basic format for a case.

This shell is a Windows-based software tool that enables individuals (both non-programmers and programmers) to quickly construct decision enabling applications which utilize Case-Based Reasoning technology [3cb, 1992]. ESTEEM delivers a set of technologies including Case-Based Reasoning, hybrid cases and

[23] `http://www.esteem.co.uk/`

[24] Kappa-PC is a Hybrid Knowledge-based Systems Environment which incorporates multiple knowledge representation schemes, multiple inferencing capabilities, options for the choice of search and the ability to incorporate standard procedural coding into you application.

rules, similarity assessment, and learning through adaptation of prior experience.

The case structure is defined by the user and every feature from the *description case* can have the following types:

- Text
- Numeric
- One of a list
- Yes or No
- Case (a feature can be a case from another case base)

ESTEEM provides the ID3 algorithm for indexing (organizing the case base). Multiple features or attributes can be selected to build more complex indices, in order to use those features as the primary search attributes.

The similarity is defined as well, and can be customized for each feature depending of what type are:

- Exact
- Partial match (letters)
- Partial word match (words)
- Range (numeric range)
- Fuzzy Range (tolerance)
- Equal
- Inferred (uses rules to determine match)

In addition, the assessment can include feature counting, weighted feature counting, inferred feature computation and weighted inferred feature computation.

ESTEEM's Similarity Definition Editor is used to define how similarity is assessed between a new problem description and the case base. Similarity is performed both at the case level (comparing case to case) as well as at the feature level (comparing the value of each case's feature value to the new entered feature values).

Rules are used to compute similarity and to adapt a retrieved similar case to better the needs of the new problem. The rules used in adaptation are domain specific, that is, each CBR application will have its own rules for adaptation. A developer should be careful by limiting the use of rules as much as possible [3cb, 1992]. Also, he must try to place as many values as he can into the creation of the case base and into the knowledge the cases will contain.

Chapter 3

The CBR System Description

In this chapter we explain every phase in our CBR shell. What is needed to assess a cycle, what are the parts that are required to the whole system and what can be done to improve the performance.

This CBR shell, which is integrated in GESCONDA, pretends to be as flexible as possible. All phases allow to be parameterized in the sense of using them according to the needs of the user, with the idea to adapt them to the new domain or context.

In the following subsections main features and functionalities of the CBR shell will be detailed: case structure, case library, configuration, how to create a new problem and the four steps of the classical CBR cycle.

3.1 Case Structure

A case is a contextualized piece of knowledge that represents an experience. To store that concept we try to take profit from what the GESCONDA data is offering.

Normally, the case structure is divided into Case Description and Case Solution. Most attributes will belong to one or other category. But, some other attributes are interesting to improve the CBR performance, such as an evaluation of the proposed solution or a measure to assess the utility of the case in the past.

Both attributes *evaluation* and *utility* could be useful to improve the performance of CBR. The first one, *evaluation*, is an attribute that contains the information of the evaluation or revise task, and it is useful to know what are the cases in the case base which have been evaluated positively or, on the contrary, negatively (see section of evaluation 3.7).

The second one is *utility*, that stores how useful a case has been during the CBR life. This attribute maintains counters or frequencies of how often this case has been used and whether it was successful (see section utility 3.9).

3.2 Case Library

The system maintains the memory as is used in GESCONDA, that is, as a flat memory. In figure 4.5 there's the model for implementing different types

of indexing the case library but, at present, only the plain (also called flat) indexing schema exists. We explain the indexations that will be implemented soon because the system allows to easily integrate another indexation schema.

3.2.1 Flat Memory

Flat memory is a list of all the cases. In our case, this kind of memory is inherited from GESCONDA.

Plain or flat memories always retrieve the set of cases best matching the input case. Moreover, adding new cases is cheap, but the retrieval time is expensive since every case in the memory is matched against the input case.

This kind of memories could could have good time efficiency if policies success to maintain this library clean without redundant cases or cases which have never been used (see retain section 3.8).

3.2.2 Hierarchical Memory

In hierarchical memories, the matching process and retrieval time are more efficient, due to the fact that only few cases are considered for similarity assessment purposes, after a prior discriminating search in the hierarchical structure. Anyway, they also have some disadvantages. Keeping the hierarchical structure in optimal conditions requires reorganizing the case library's structure [Veloso & Carbonell, 1993] in front of new experience, otherwise the retrieval process could miss some optimal cases searching a wrong area of the hierarchical memory.

This kind of memories can be split up in three types depending on how they have been built up:

Share Feature Networks/Trees The cases are stored in a tree, and are located depending on the characteristic attributes (clustering). In other words, the common characteristics are nodes where all the cases sharing this characteristic hang.

This process is computationally more expensive because the tree needs more space and its storing and updating is more complex.

The retrieval task is cheap because only a subset of cases is analyzed, but its result depend on how the hierarchy is established.

The quality of the results depends on the establishment of an importance characteristic hierarchy.

Discriminant Tree with priorities This kind of tree solves the problem of searching even when the input case is incomplete. The main idea is that each node contains a question to all subnodes which provide alternative responses. The most important questions are formulated first, higher in the hierarchy.

As in common characteristics tree, discrimination trees subdivide the set of cases and share most of the advantages and disadvantages of this technique. The questions can be implemented more efficiently than when matching in each subnode. At the same time, to separate the attributes in their particular values makes easier to identify which attributes have been more useful for the case characterization.

The disadvantages that arise are those of the Share Feature Networks/Trees: some significant cases may be ignored if the ordering of questions is not optimal.

Redundant Discriminant Trees or Dynamic Memory Model These trees solve the problem of not retrieving the optimal case by organizing them in different trees, each one with a different order of the questions. Most of them use the share feature networks to maintain their size controlled.

This method has been one of the most implemented hierarchical memories, excluding flat memory.

This model has been developed from the MOP (Memory Organization Packet) theory [Aamodt & Plaza, 1994]. This structure has been postulated by Roger Schank as a more general knowledge structure to account for the diverse and heterogeneous nature of episodic knowledge. More important than the MOP knowledge structure was the new emphasis on the basic memory processes of reminding and learning [Slade, 1991].

The basic idea is to organize specific cases which share similar properties under a more general structure, a Generalized Episode(GE). These episodes contain three different types of objects:

Cases Set of cases in the GE.

Norms Common characteristics of all GE's cases.

Indices Features that discriminate between GE's cases. An index can point a case or another GE.

This schema is called redundant since it is possible to reach the same case or GE via multiple paths. When a new case is given, the best matching is searched and the new case is allocated in the tree starting from the root node.

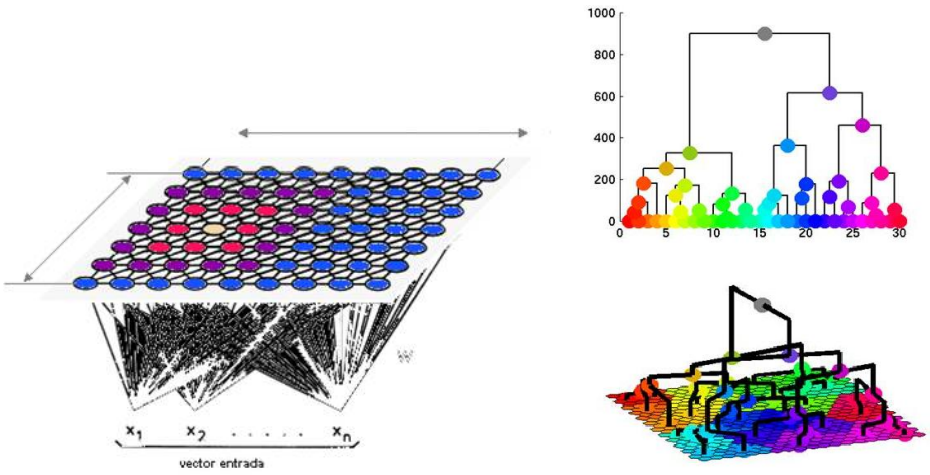
To retrieve a case, the GE with the most common norm is searched, then the indexes are used to look for the case with more additional common characteristics.

One of the problems is that the number of indexes can excessively grow when the number of cases increase.

Finally, this organization could be seen as a memory that integrates knowledge of specific episodes with knowledge of general episodes.

3.2.3 Self-Organizing Maps

Self-Organizing Maps (SOM) belong to the topographic organized maps. The last kind of maps specifically allows us to understand how the data are sorted according to their characteristics. The models will usually be trained on a unsupervised learning, so that cases depend on the natural form. With a trained model, each case can be represented in such a way that it is easy to identify the subgroups and areas that are more intermediated. These maps are nests of points that form a mesh (neurons, in instance, for SOM), which besides can be interpreted as prototypes. In the figure 3.1 we can see how the input instance actives a neuron and how its neighbourhood is affected and, in the right subfigures, how it is possible to interpret them as prototypes.



(a) Active Neuron Influence (b) Clustering View

Figure 3.1: Self-Organizing Maps Structure

SOM are a model created by the professor Teuvo Kohonen. Self-Organizing Maps are commonly called the *Kohonen Maps*. Although, in the literature, sometimes SOM are defined as more general and include Kohonen Maps [Fodor, 2002].

They are considered a competitive neural network based on the biological model of the memory function, in how information is organized in the brain. The theory says that, during learning, when a neuron reinforces itself, it reinforces all neurons that are near to it. The transition between to strengthen neighbour neurons (cooperative learning) to inhibit them (competitive), is smooth as the distance between neurons increases, so the information is sorted topographically if we observe the neurons that are responsible for it.

Teuvo Kohonen, based on these ideas, in 1982 presented a model with a similar behavior. There are two variants: LVQ (Learning Vector Quantization) and SOM (Self-organizing Maps), both based on the principle of topological map formation to establish the common characteristics among the input data. Also, both differ in the dimension of the map, being on a single dimension in the case of LVQ and bidimensional or three-dimensional in SOM.

The most common use is related to visualization. The most important reason is that data with high dimension is transformed into a latent space with low dimension. In addition, the new space follows a topological order.

It is considered a unsupervised learning technique because it is not focused on the relationship between the descriptive attributes and the class variable. Learning in the Kohonen model is offline, which distinguishes a learning(training) stage and performance. During the learning, the values (weights) of connections (feedforward) between the input and output layer are set up. This network uses an unsupervised competitive learning, neurons in the output layer competing for being activated and only one remaining active at a given information. Then, the weights of connections are adjusted depending on the winner neuron. Besides of the connections between input and output layer, the neurons consti-

tuting the output layer interact laterally such that the weights between neurons nearby (neighbouring) are similar. These lateral connections are not physical connections.

The goal of SOM is to find a similarity function that allows classifying the data set in different output neurons (prototypes).

An important concept in the Kohonen network is the neighbourhood around the winner neuron. The weights of the neurons who are in this area shall be updated along with the weight of the winning neuron. That's an example of cooperative learning.

Learning is an iterative process to adjust the weights of a network and reducing the number of neurons that are activated for an example or instance. The Kohonen network could be defined as a classification task, since the output neuron that is activated by an entry represents the class of such instance. In addition, a similar instance shall activate the same output neuron, or other close to it, due to the similarity between classes, thereby guaranteeing that neurons that are topological nearby are sensitive to a similar inputs, so the network is especially useful for establishing previously unknown relationships among data sets.

3.3 Configuration

This part of the project might be the most important one since our objective is to build up a flexible system.

To cope with this requirement, we want to create a system that can be configured from different points. Since our system works both as an interface and as a shell, we thought that both ways must have a mechanism to get all the variables configured.

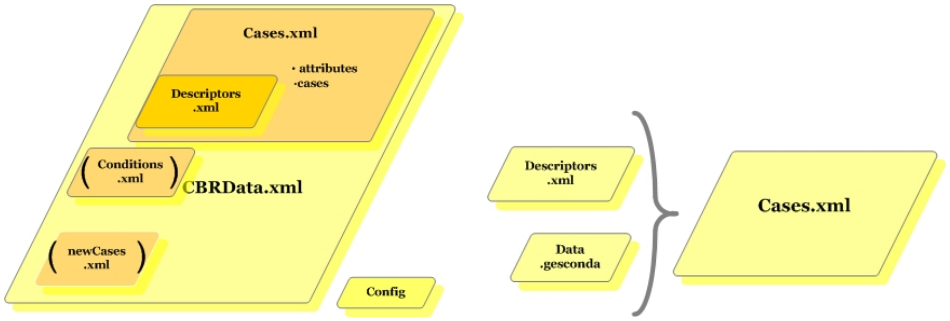
When it is run as a GUI, all the variables can be asked to the user using dialogs In section 4.3.3, where the views are explained, it is possible to see some examples of these dialogs.

Besides, if it is run as a shell then a configuration file is used. Inside of this file are all the global parameters which are possible to change together with their default values. In appendix B this configuration file is included, where all the global variables can be set up. This file is loaded into the system. The GUI mode is using it as well, but it can also use other values by changing them through dialogs. These new values are temporary and do not override the ones from the file.

Our purpose is that the CBR system can be used from the command line only indicating the file with the case base and its structure. In trying to achieve this goal, we propose all the configuration related to data to be in files apart from the configuration one.

From now on, it is possible for our CBR to load the data from GESCONDA II, so we need to load the rest of the configuration that is related to the data. We have thought in two ways to cope with this problem. The first one, it is a specific *xml* file that contains all the information about the data. The second one, it is to have the information split into different files. One of these files could be one of the supported GESCONDA II formats (described in section 2.2.2). We follow our purpose to be as flexible as possible, so the user can decide how to manage the data, but following the *xml* structure that we define.

In addition, figure 3.2 is included in order to understand how the files can be loaded to run the system. This figure 3.2 shows a configuration file (*config*) and data file. The data file contains the information about the domain and it can be loaded as unique file (CBR project format). Otherwise, this data can be shared into different files containing: descriptors, new cases, or the data with any GESCONDA format.



(a) Organization of files

(b) How CBR data file can be substitute using GESCONDA format file

Figure 3.2: Files that can be used by CBR system

In the next list we can see what is the information that should be loaded to have everything needed to run the CBR shell. In this list, all of the items added with this CBR module are highlighted:

- data base
- case structure
  - Attributes
    - Type = Continuous, DiscreteNonOrdered, DiscretOrdered, **List, Utility**
    - **Descriptor = Description, Solution, Evaluation, Utility, Cbr**
    - Weight: Importance of this attribute. **The sum of all weights should be one.**
    - **Distance: What distance is better for this attribute**.

3.4 Start a new case

The beginning of every CBR cycle is to define the new case wanted to be resolved. So, to do this task our CBR system offers different ways to get the data from:

By user Manually introducing the attribute values.

From Case Library Choosing one from the current case library. In this case, the user can decide if he wants to use a copy or the real case of the library. In this case, any change on it will be reflected in the library. In case of a *Battery*, instead of choosing one, the user can select a percentage of the first ones, as well as a random set.

From File In the case of a *Battery* the list of new cases can be loaded from a file. Of course, the cases of this file must have the same structure as the cases that are loaded.

3.5 Retrieve

In general, the retrieve phase involves finding the most similar case/s to a given case. This task starts with a (partial) problem description, and ends when best matching previous case has been found [Riesbeck & Schank, 1989]. It is usual to divide this task in two parts. The first one tries to select similar cases and the second one chooses the best matches.

Really, this division into two tasks makes sense when a memory organization (indexation) exists. Then, a set of cases is returned as similar cases. For instance, in a hierarchical structure, the cases of a substructure would be returned. The second subtask takes place by selecting the most similar case/s.

One of the goals is to have this organization or indexing of the cases but, currently, this objective is not reached. Besides of that, the CBR design is prepared to have it in the sense that, when a new problem is presented, the algorithm of retrieval is divided in these two subtasks.

First of all, the case library is asked to return the set of cases that are candidates to be compared with the new case. Actually, the library returns all the cases that are stored. But the idea is that an active indexation returns theses cases.

Afterwards, we focus on the second task, starting from the precondition that the case library must return a set of cases given a new case. And from them, the retrieve task might be defined as the selection of the n most similar cases to new given case. To make this task as flexible as possible, we have parametrized all the possible variables.

The parameters can be separated in two groups: the global ones and the ones related with the current data.

3.5.1 Global parameters

Number of cases to retrieve(n) The algorithm will rate all the cases according to its similarity. The n^{st} most similar cases will be shown or, if the next m cases have the same distance than the n^{st} case, then the $n+m$ cases will be shown.

Use only description This variable indicates if only the case description is used to find the most similar, otherwise the case solution will be used.

Use the current case if it is in the library When the new case is selected from the case library, there is the possibility to use it in the retrieval system or, rather, remove it temporarily from the library. Note that it may be interesting to test the system if the case is in the library. It could be possible to see if the system is doing a good retrieval. Also, if an indexation exists, it could also be possible to see whether it is correctly built up.

Distance The user not only can select the similarity functions that are implemented in GESCONDA II but also customize their parameters (See

description of distances in the appendix A). The parameters for each distance are the following:

L'Eixample:
- $\alpha \rightarrow$ Threshold that defines the boundary for using quantitative or qualitative values for continuous attributes.
- Accepting the weights already set or setting them up again

Euclidean:
- Choosing whether or not standardizing the data
- Accepting the weights already set or setting them up again

Manhattan:
- Choosing whether or not standardizing the data
- Accepting the weights already set or setting them up again

Minkowsky:
- $r \rightarrow$ variable power to specify the distance type. $r = 1 \rightarrow$ Manhattan distance, $r = 2 \rightarrow$ Euclidean distance.
- Choosing whether or not standardizing the data
- Accepting the weights already set or setting them up again

Cosinus None

Canberra None

Clark None

3.5.2 "Local" Parameters

The next variables depend on the data:

Attributes As we explained, the case structure (3.1) is changeable in that it is possible to select which are the attributes of the description and which are the ones of the solution. Besides, GESCONDA II offers the mechanism to activate and desactivate the attributes, so just the active attributes will be used for assessing CBR algorithms.

Weights It was explained in GESCONDA functionality (2.2.2) that several algorithms exist to automatically calculate the weights of the attributes. So, this system is reused to generate the weights prior to retrieve, but it is configurable in retrieve dialog.

GESCONDA provides for itself the functionality of active and desactive attributes and our CBR respect this configuration.

3.6 Reuse

Reuse is, inherently, one of the most complex task in the CBR cycle, since it requires a deep understanding of the represented situation in the retrieved case/s in order to be able to modify or set up the solution. Typically, this understanding is domain dependent. Thus, it is important to *define generic reuse methods or methods which would be able to capture automatically the domain knowledge.*

In general, it is considered that the CBR inference is based on the principle [Lieber, 2007]:

"Similar problems have similar solutions"

The retrieved case solution adaptation - in the context of the new case - focuses on the differences among the past and the current case and in what part of the retrieved cases can be transferred to the new case. The selection of the cases to be used for adaptation from the retrieved cases can be either guided by an *Utility* attribute or selected by the user. Its value can be determined by the number of times that a case has been retrieved, the number of times it has been used for adaptation or the number of times it has been successfully used (positive evaluation). Also, it can be a combination of all these criteria (see 3.9). In the literature, the reuse task can be separated in different methods:

Null or Copy The small differences are abstracted away and they are considered as non relevant. So the solution of the retrieved cases is directly transferred to the new case as its solution. This is a trivial type of reuse, but it is widely used in many CBR systems because it is domain independent. In our CBR system, it is possible to apply this kind of method. When more than one retrieved case exists, then a *mean or mode value is computed*.

Some authors do not consider this method as an "adaptation method" [Aamodt & Plaza, 1994]. Nevertheless, when a mean or mode are applied, it can be considered transformational reuse because it is a formula that transforms the retrieved solutions in a new one. Another possibility is to assess a pondered average, or rather, a weight average where the weight are inversely proportional to the distance to the new case. Or even, considering the *Utility* attribute by giving more importance to more "useful" cases.

Transformational Reuse The past case solution is not directly a solution for the new case, but there exists some knowledge in the form of transformational operators. To develop these operators, knowing some extra information is needed, which is domain dependent. Our CBR system provides a module where *numerical formulas* can be defined for the numerical attributes belonging to the solution of the new case. Thus, when the system is used by an expert, he/she defines the best transformation scheme for the solutions.

Derivational Reuse Reuses the algorithms and methods that produced the original solution in order to create a new solution to the current problem. Also, the planning sequence that generated the original solution has to be stored in memory along with the solution. This approach is sometimes called *reinstantiation*, and it can only be used for cases that are well understood. This kind of reuse is not available in our system because it is domain dependent and conceived for other type of data different from data-intensive CBR systems, as may be the case that represents a process.

Another point to take in account is that our system can *support more than one attribute belonging to the Case Solution* and this classification between description and solution can vary when the user wants. So, in the literature we have found some solutions but most of them are oriented to solve only one solution attribute. Most of the proposals use techniques from the supervised learning field and have only one attribute to predict [Fdez-Riverola & Corchado, 2000]. For example, a system to predict oceanographic temperatures [Corchado & Lees, 2001]. This system retrieves the most similar cases and retrain a radial basis network with them to create a new solution. Or WeVoS-CBR, which is a hybrid intelligent system to forecast the presence or not of oil slicks in a certain area of the open sea after an oil spill. In this system, the data is indexed by a Visualization Induces Self Organizing Map and it uses a growing neural network to guide the retrieval and create a new solution with these recovered cases

retraining the network.

One requirement of our system was that it has to be as flexible as possible in the sense to offer a CBR that could be adapted to any problem. That is the reason to think in what are the possibilities facing this *Adaptation* challenge. Since the problems that we cover are data intensive, we have to look for solutions which don't rely on an existant knowledge.

For the time being, CBR module implements three adaptations: null, mean, weighted mean and formula. The first one copies the values of each attribute in retrieved case solution to the new case solution. And the second one computes the mean/mode depending on the attribute type, and copies this value to the new case solution. But we split them because the calculus can be weighted by either the *similarity* or the *utility*(see 3.9):

$$s_i(new\ case) = \frac{\sum_j^R s_i(retrievedCase_j) * weigth(retrievedCase_j)}{\sum_j^R weight(retrievedCase)},$$

$$where,$$

$$s_i : \text{solution attribute with index i}$$

$$R : \text{number of retrieved cases}$$

$$weight(retrievedCase) : \begin{cases} 1 - distance(retrievedCase, newCase) \\ utility(retrievedCase) \end{cases} \tag{3.1}$$

Guided by the transformational reuse, we create a module where each attribute can be defined by a user formula (the third adaptationscheme). Each attribute from the solution can be computed with the different options. In this point, it is possible to use the GESCONDA functionality to create new attributes with the solution values from different algorithms.

Our formula module is an initial approximation. In the future that module will be able to be extended to cover more formulas and to be configured through a file. Currently, by means of the GUI, the user can introduce a formula including operators, numbers or the own continuous attributes values. To simplify the process, if there is more than one retrieved case, then the mean value for these cases will be used. The next equation 3.2 is an expression representing the kind of formulas which are supported:

$$sol_i = sim(operator\ sim)^*\ ,\ sim = \begin{cases} number \\ value\ of\ selected\ attribute \\ \text{mean of the values of selected attribute} \\ \quad \text{of the retrieved cases for adaptation} \end{cases} \tag{3.2}$$

In configuration 3.3 section, we explained that both the configuration and the acquisition of the parameters are achieved through the interface or by means of files.

In this phase, all the options - excepting the formula adaptation option - are possible to be configured by means of a file.

Using the GUI, the user can select the retrieved cases that wants to use for adaptation or just use all of them. The next step is to choose what kind of adaptation wants for each solution case:

Null: copying the solution of the most similar case out of the chosen ones in the previous step to the new case

Mean/Mode: computing the mean or mode for each attribute from the solution depending on the attribute type

WeightedMean/WeightedMode: this time the mean is weighted by the similarity that has been calculated in the retrieval task to give more importance to the most similar cases. As an alternative, it can be calculated using the Utility attribute, thus giving more importance to the cases that are supposedly more useful.

Formula: This option is only available for the continuous attributes. A new dialog is opened to introduce a formula created by the combination of numbers, retrieved case attributes and operators (see figure 4.9 in section 4.3.3).

This part is easily extensible so, if considered worthy in the future, this module can be extended by any Java programmer without knowing all the implementation. Moreover, if an automatic generator of formulas is provided, it should be easily to introduced in the system.

3.7 Revise

This task can be really important because it gives the opportunity to learn whether the previous tasks run correctly or not. Therefore, this part of the cycle consists in evaluating the proposed solution to the new case.

This phase is maybe the most difficult one, because there is none automatic method implemented in the literature. Most of the papers that treat case-based reasoning either avoid this topic or leave it to the expert who is using the system.

It is supposed that this evaluation takes place after the solution is applied in a real environment. Then, an expert evaluates how useful has been the proposed solution. Hence, there is little implemented about automatic and generic methods.

In [Michael et al., 2001] they present the "utility" as we understand the evaluations. To them, this evaluation is an important term that encompass in a more generic way what has to be retrieved: not only the most similar case, but the most useful case. In the next figure 3.3 is well-explained this concept.

Despite the use of the term "utility", they describe this along the text as a criterion used by the user to evaluate how useful has been the solution to him. They introduce a methodology to improve the performance of the retrieval phase if this evaluation is approximated before selecting the case. Anyway, to approximate this evaluation they outline that it is domain dependent and has to be specified. However, there are some possible approximations, but those depend on the type of use. If it is for testing, the evaluation could be done like in supervised learning. The simple idea is create a percentage of how equal is the proposed solution to the real one.

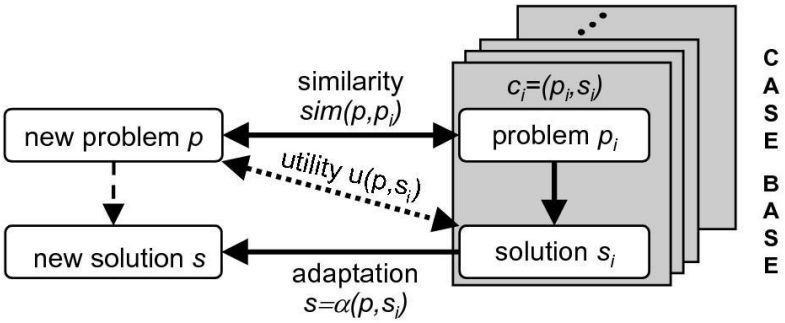


Figure 3.3: Comparison between similar case and useful case

An approximation of the value would be done by looking at the evaluations of the cases used to set up this on. When the utility attribute is created, also a new attribute is created: *SourceCases*, which is a list of references to the cases that have been used to create the solution of the current case. Thus, this attribute could be used to access the precedent cases and form an approximation to the new case evaluation.

In the literature, we found that this step is made up of two subtasks. The first one is the evaluation of the case itself. This task usually takes place when applied to the real environment or in simulations. The second one consists on adapting the solution again [Aamodt & Plaza, 1994]. This step could be interpreted like going back in the cycle and return to the reuse or adaptation phase. First, the errors have to be detected and then retrieve or generate explanations for them. After, theses explanations are used to modify the solution in such a way that the failures do not occur. Other authors prefer to have the evaluation in order to not proposing a bad solution again [Baccigalupo & Plaza, 2007], though. This provides an opportunity to learn from failure.

In our system, currently is a percentage of how good has been the case, but has to be introduced by the end user or expert, unless the default evaluation value is used. Graphically, it is possible to easily change it. It is a simple scroll bar. Moreover, the default value could be modified in the configuration file (see configuration section 3.3).

3.8 Retain

Learning by (own) experience is the last task of the cycle in Case Based Reasoning. After an evaluation step, the opportunity to increase the problem solving capabilities of the system arises. So, it can learn from the new experience or not. If the proposed solution has been a successful one, the system can learn from this fact in the sense that, if this experience is stored in memory, when a new similar case to this one appears, it can be solved like it (learning from success). If the system has failed, it must be able to prevent itself from making the same mistake in the future (learning from failure). Not all Case-based systems have both kinds of learning.

When a case is evaluated as a failure, if it is assumed that the adaptation

method is correct, two reasons could haver originated the case to be a failure:

- The retrieved case is the best one for solving this new situation, given the current case library, although it is not very similar to the new case. The problem here is that there are not enough cases (experience) in the case library to cover the whole space of cases.

- The optimal or nearest case has not been retrieved due to the indexation of the case base. Thus, there is something wrong in the retrieval process.

When the new case has failed, there are several possible actions to be taken, in order to ensure that this failure cannot be repeated in the future. First, the CBR system can store the failed case into its memory to prevent taking, another time, the same failing solution for similar cases to this one. Some case-based systems maintain a separate case library of failed cases, while others maintain only one case library structure. The latter is our situation, because our evaluation system is not just based on "success" or "failure". It could be evaluated with intermediate values.

In each domain and application it is necessary to decide what, when and how a new experience should be stored. This is a process of incorporating what is useful to retain from the new problem, that is, whether the new proposed solution deserves to be in the case base as a new knowledge because is considered as a useful experience. Sometimes, it's not worth to store it, i.e, the new case is almost equal to a case in the library. That is the reason for creating a functionality where the user can indicate what are *his learning criteria*, in order to configure a criterion being dependent to his/her *specific domain*.

Our intention is that every domain or application has its own retain criterion. Initially, the case base may be empty, so it is not necessary to control it until the number of cases is growing too much - with the consequence of increasing case base all times without improving the results. Uncontrolled Case Base growth can cause serious performance problems as retrieval efficiency degrades and incorrect or inconsistent cases become increasingly difficult to detect.

The following characteristics influence in the quality of the data:

Relevance cases that are not relevant could have a negative impact

Correctness wrong cases can derivate in wrong solutions

Memory space the more cases exist, the more cost of storing

Retrieve cost if the memory is large the cost to find good cases is increased as well

The *Retain* task would be seen as one policy of the maintenance of the case base because its purpose is to avoid incorporating a new data in case it could cause noise [Leake & Wilson, 2000]. Maintenance in CBR can mean a number of different things: out-of-date, redundant, or inconsistent cases may be deleted, groups of cases may be merged to eliminate redundancy and improve reasoning power,cases may be re-described to repair inconsistencies [Iglezakis et al., 2004].

Therefore, the retention of a new experience is not an easy task. Some domains change along the time and recent experiences are more important than the old ones. Nevertheless, some domains are temporarily episodic, or rather

the old cases could be useful again. For instance, data depending on the weather is different for each season.

Besides, the utility attribute can be interesting in this phase by improving the efficiency of the system by ruling out selected knowledge [Minton, 1990] This should be done using metrics of use frequency and usefulness of a determinated case to the knowledge. This metric is what the *Utility* attribute is storing (see section 3.9).

After understanding the importance of retaining a new case in the library and that it depends on the domain and the indexation, we decided to create a functionality which may be easily extensible and flexible to be configured. Thus, our shell offers *an interface that allows the user to add conditions which have to be satisfied by the case before to be learnt or not, as we can see in figure 4.10 in section 4.4.8.*

The idea is that the user configures which condition needs to ensure that the new case will be useful to the case base. This is achieved by joining conditions with the logical connector AND($\wedge$). These conditions are loaded from a file. There are two ways for selecting which are the conditions to follow. The first one is to select all conditions in the file: this is appropriate if the CBR is run from the command line. And the second one is through the interface: the user can select the conditions to use and, for each condition, which are the values wanted for selection. In order to show these conditions graphically we had to decide a representation. So, we decided that a condition consists of three strings at any given moment:

1. Name: Identifier of the condition.

2. Operator: Identifier of the condition function.

3. Value: Identifier of the value to use.

This is a general formalization to not restrict what a condition may be. Therefore, within the file we can define the condition as one identifier and a list of operators and, for each operator, a value list or a number. Hence, the function is run in the itself class and the retain algorithm only asks to them whether they are satisfied or not. For a better understanding of this abstraction we explain some examples are described below.

3.8.1 Distance

The first and simple condition is not to store the cases that are equal or very similar to other one which is already in the case base. In order to define the condition, we can formulate: "If similarity is bigger than 90% then do not store it". In this case, the three identifiers could be: "similarity", "$<$", "90". To use this condition - "similarity" - it is possible to define different operators and values. In fact, the value could be a representation of a number and the user may introduce any numerical value.

Thence, the condition identified by "similarity" may have different operators("$>$", "$<$", "equal", "$>$90") and each operator a numerical value or a list of values. For instance, the operator "$>$90" has the values "yes" or "not".

3.8.2 Evaluation

As with *Distance*, the conditions of the values using their evaluation work in the same way (as it happens with *Utility*), so we skip these conditions to explain a more complex one.

If this new case has not been evaluated, in order to not have failed cases, the case base is searched to find which case that has a negative evaluation matches the new input case. This is done to avoid repeating the failure, or rather to avoid having similar incorrect cases in the library. Then the condition needs two values, the first one could be the evaluation and the second one, the similarity. But we have already explained that the value is a string, so this string must depict both values. This could be done by introducing a list of values:

name: "Negative Similar Case"

operator: "negativeSimilar"

values :numberEvaluation, numberSimilarity

Then, if the case retrieved has an evaluation smaller than the value "numberEvaluation" and its similarity is bigger than the value "numberSimilarity", the case has to be ignored.

3.8.3 Recursive Cluster Elimination (RCE) method

In a context, the training data are the past cases of diagnoses. These records are organized as description-solution associative pairs. With an adapted clustering based learning algorithm similar to the RCE method, the algorithm is outlined as follows:

In this algorithm, each training case is an associative pair (X, Y), where X is the input description vector and Y corresponds to the solution component. Each cluster center has a radius of attraction R_j, whose value lies between its allowable maximum and minimum values. When a new training vector X is entered, the algorithm computes distance scores D_j for all existing cluster centers C_i.

If D_j is less than R, for some j, it means that the input pattern is covered by C_j. Then the algorithm checks if the solution of C_j is the same as Y in the training record. If they are the same, then the training record is marked "covered" and nothing else needs to be done. Otherwise, it means that the radius R_j is too large, then it has to be reduced. If the radius R_j has already been reduced to its allowable minimum, then the cluster center C_j is made to represent more than one solution. This means that the set of descriptions, X, does not uniquely identify the solution and hence it recommends multiple possible solutions. In this situation, the case has to be stored. The user will then need to verify each of these suggestions.

If the input record is not covered by any cluster (ie. $D_j > R_j$ for all j), then a new cluster is formed. Its center will take the input vector values, its radius of attraction initiated to the maximum allowable value and its solution set to Y. Notice that when the radius of a cluster center is reduced, some previously covered input patterns may become exposed again. Thus, whenever the radius of some cluster center is decreased, the system is marked "unstable", and all

the training data will be presented for another iteration, until the system finally becomes stable.

Whenever a new training record is available, the same learning algorithm can be applied to incorporate the new case into the existing knowledge base in a relatively fast manner [Lim et al., 1991].

Then the condition could be reduced to indicate what is the minimum value of the radius: {"RCE", "minimum Radius" and "numerical value"}.

3.9 Utility

"The utility problem in artificial intelligence system occurs when knowledge learned in an attempt to improve a system's performance degrades performance instead" [Minton, 1990]; [Francis & Ram, 1994]

One of the main problems in CBR is the maintenance of the case base [Orduña-Cabrera & Sánchez-Marré, 2008] [Orduña-Cabrera & Sánchez-Marré, 2009]. When the number of cases reaches high values then some problems can occur. Next, we explain some of them:

- When the number of cases grows, the cost to find a case grows as well.
- A case base containing obsolete cases means that noise exists and performance could decrease.

We mentioned the *Utility* attribute several times in previous sections. *"Utility"* is a concept that tries to keep the usefulness of the case in the case base. Of course, this may be an abstract problem depending on the point of view. Sometimes, the utility can be how often a case is used to resolve new problems while other times is more important to just count how may times this case is near to the new presented problems.

The question is: where could be useful this *Utility* attribute to be used? As a matter of fact, this attribute could be useful in almost all CBR tasks:

Retrieval From a set of cases that are similar to the new case, select those that are most useful.

Adaptation If there is a group of cases to use in adaptation, *Utility* could give more relevance to those that are more useful.

Case Maintenance The policies to delete some cases to reduce the size of the library could be guided by this attribute avoiding to delete those cases which have been used the most. Or to delete those that have not been used in a long time.

In [Francis & Ram, 1994] they suggest that we could improve the behaviour of their CBR approach by assessing cases quality, and using the quality in order to prefer useful cases in the retrieval.

In the following lines we introduce an example in which we use this attribute to maintain the case base. As a consequence, retrieval time forgetting is reduced, that is another different point of view from the perspective seen until now.

The article *Forgetting Reinforced Cases* [Romdhane & Lamontagne, 2008] presents a study comparing the performance of Case Usage versus Case Value

criteria for forgetting cases, hence bounding the number of cases to be explored during retrieval. In this study they conclude that case usage is the most favourable criteria for selecting cases to be forgotten prior to retrieval. In addition, they try mixing case usage and case value obtaining some improvements.

Computational time in a CBR cycle depends on the time dedicated to retrieval and adaptation. In a nearest neighbour setting, retrieval time depends on the number of cases being considered for making recommendations. Hence, to meet time constraints, a CBR system would either have to forget cases prior to case retrieval or to filter cases while performing retrieval. For this work, they adopt a forgetting approach to limit retrieval time.

In the same article, the goal is to determine whether the values obtained through reinforcement learning could provide a good indication on which cases a CBR system should forget to reduce the size of a case base. This work is conducted by a case study using Tetris, a game with simple rules presenting relevant time constraints.

In figure 3.4 we can see how is the performance based on the number of lines eliminated in tetris game. From this experiment, they conclude that case usage (U) provides a better decision criterion for progressive forgetting of cases. Results also indicate that reinforcement value (V) is not by itself an informative criterion for reducing the size of the case base. It is also interesting to note that the size of the CBR memory can significantly be reduced without severely impacting on its performance. The combination of both criteria improves the results,especially when the case base size is tiny.

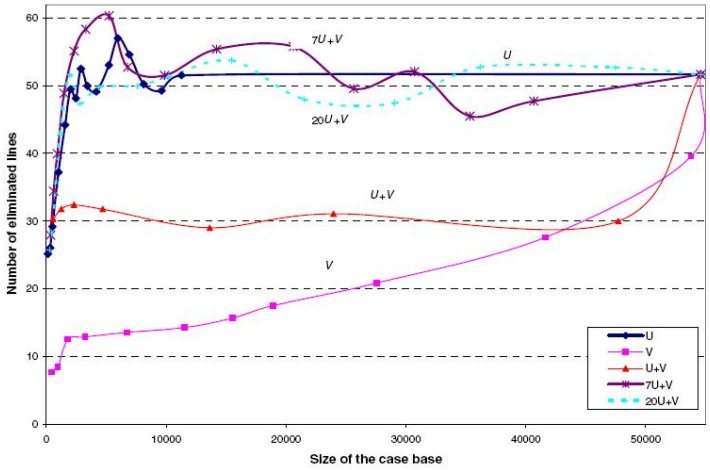


Figure 3.4: Performance profiles for case usage(U), case reinforcement value(V) and combination of they two. Performance profiles obtained by progressively removing cases from an initial case base of 55000 cases

But it's not all advantages when always selecting the most useful cases. If doing so, then the same cases are selected every time, thus creating a new problem of leverage. Hence, *Utility* creates a dilemma to deal with the exploration vs. exploitation trade-off: when there is no good case for recommending a node, we may prefer the less used cases (exploration), or the most used ones

(exploitation).

3.9.1 Our approximation of *Utility*

What are the factors that make a case become useful? This question is a new challenge that has to be solved by thinking in all domains, or rather by a generic definition. The next points are analyzed: determining how often the case in the system has been used and determining how long since the last time that has been used. The assessment of the answer to these question could be done defining this information for each case:

- Total number of times the CBR cycle has been run, which should be common to all the cases (Num of Runs).
- Number of times a case has been retrieved (Num of Retrieves).
- Number of times a case has been selected to create a new solution (Num of adaptations). Notice that it is a subgroup of the retrieved cases.
- Number of times a case has been selected to create a new solution, becoming this new solution positively evaluated (Num of success).
- Date of case creation.
- Date of the last use of the case.

Utility could be seen as a frequency determined by a fraction [de la Rosa et al., 2007], so the numerator and the denominator must be specified. One approximation could be how often a case has been successful:

$$utility = \frac{numerator}{denominator} = \frac{Num\ of\ success}{Num\ of\ runs\ or\ retrieves}$$

To make *Utility* attribute more flexible, it is possible to decide what are the numerator and the denominator through the GUI and the configuration file. Remember that the evaluation attribute can be avoided, so "Num of success" would not have sense. Then, "Num of adaptations" could be used instead as a good metric of how useful has been the case along the CBR life.

Chapter 4

The CBR Shell Development

In this chapter, most of the skills, described in the last chapter 3, are designed and developed in the current CBR shell prototype. During the CBR design specification we have been aware that one of ours main objectives is to have an extensible CBR shell. Thus, some decisions have been into account for designing and implementing.

After describing functional and non-functional requirements of the system (Section 4.1), we describe the main functions (Section 4.2). Finally, in Section 4.3 we provide the design of the system and we remark important points of the implementation and the algorithms. In this chapter, we do not specify the CBR shell in details as the system has been informally presented in the last chapter 3.

4.1 Software Requirements

In the following we explain the main software requirements of the system. Some of them are restrictions due to the integration with GESCONDA, besides to the functionalities that this new module should offer.

4.1.1 Functional requirements

Our aim is to create a CBR shell that can be integrated in the last version of GESCONDA II. One of the main requirement, and also the most difficult, we want to fulfill is the shell flexibility. The shell has to be as flexible as possible in all the CBR phases and sufficiently generic to be able to cover more than one domain.

The following requirements have been used as a starting point:

- to store the case base in xml files
- to import/export in both cases' types and GESCONDA cases' instances.
- to be used for different domains
- to offer the possibility to launch a battery of queries
- to trace logs

- to have a flexible structure. The structure should not be fixed as changes in the case structure should be easy to apply
- to have some indexing technique. The indexing should allow the possibility to order the cases in some structure
- to be able to select the number of cases in the retrieval step
- to select the distance to use and its parameters
- to select the cases from the retrieved to be used in the adaptation step
- to light feedback in the evaluation step
- to be able to specify options to store or not the solved cases

4.1.2 Non-Functional Requirements

These requirements are those that our system should satisfy. As previously said, some are restriction due to the integration of the CBR shell with GESCONDA II:

- the GESCONDA code must not be modified or at least the minimum. In principle, GESCONDA II offers a simple system to integrate new functionalities that will be accessible from the main menu.
- the data should be not duplicated. CBR should use the same objects in memory that are created by GESCONDA .
- the front-end should be usable both for application and library modes
- the use of the libraries available in the group should be maximizaed
- the system should be as flexible and generic as possible, both to extend and to configure the parameters of CBR
- the systems should be a multi-platform tool

4.2 Use Cases of the CBR shell

The functionalities of this CBR prototype are represented in the next use case diagram 4.1. In this figure we distinguish between two actors, the user and the system. And the inclusions among functionalities- if one functionality uses another one.

Finally, we briefly specify the main functionalities in the following subsections.

4.2.1 Define Case Structure

Since we want a Data-Intensive Case-Based Reasoning, our cases are constituted as a set of attributes. These attributes can belong to case description, case solution, evaluation, utility or other attributes that internally are used by CBR such as source case.

In order to have a more flexible case description, the user can define to which group the description belongs. It is also possible not to create any utility nor

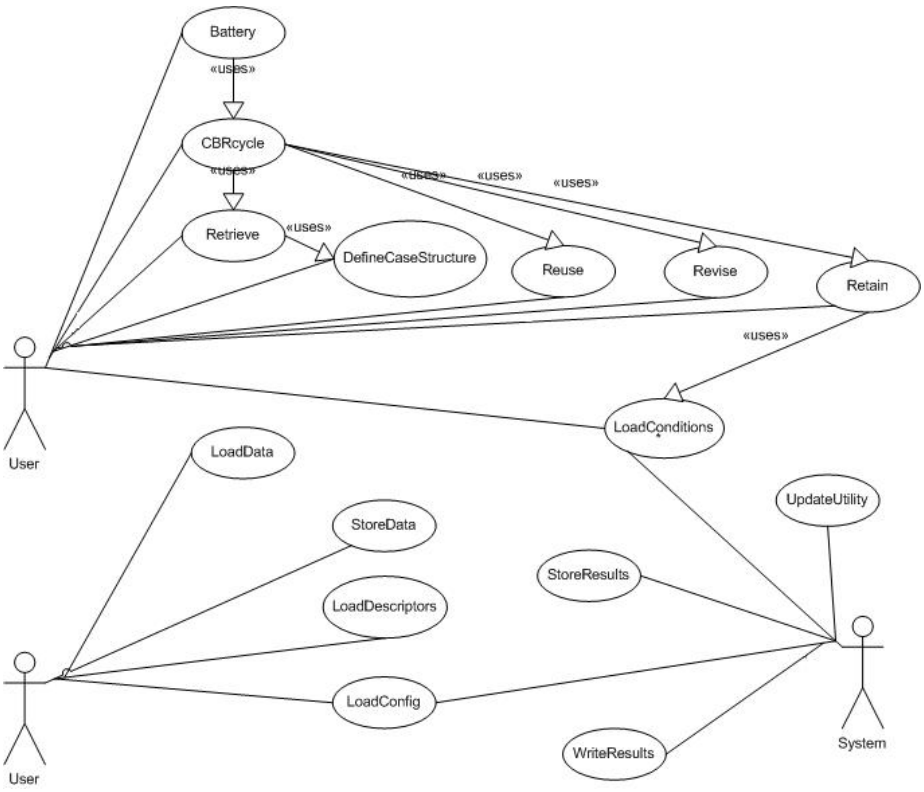


Figure 4.1: Use Case Diagram of the CBR Shell

any evaluation. In this way the case is more similar to a typical instance. In addition, if just one attribute is selected to case solution, these cases are like instances with an attribute as a class, as in the supervised learning field.

4.2.2 Select New Case

A new case definition is possible whenever a new CBR cycle is started as a whole cycle or just as only the retrieval phase. For the case definition, the user is allowed to create a new case case and specify all the values, or pick up one case from the library or even, to edit the values of an existing case picked up from the library.

4.2.3 Select New Case List

A new cases list can be loaded from a file or picked up from the library, for example when the user wants to run a battery of CBR cycles,(to solve a battery of new cases). If cases are chosen from the library it is possible to take a percentage of the first cases or take a percentage of random cases. At this moment, the user is asked whether he wants to remove the cases from the case base or to maintain. If they are maintained, the cases will be used as well as cases in the library. The use of each new case to retrieve depends on the

configuration. It is possible to compare with itself or to ignore when the cases are the same.

When the case or cases are selected from the library, the user can choose if he/she wants to use the case/s uniquely as new cases or also, as case base. Depending on the configuration those will be compared against themselves in the retrieve algorithm or ignored when both the compared cases are the same.

4.2.4 CBR cycle and Battery

We mentioned these two functionalities. The CBR cycle implies to run the whole cycle in one time, without stopping between each CBR step:

1. Set up the new case (4.2.2)
2. Retrieve (4.2.5)
3. Adapt (Reuse)(4.2.6)
4. Evaluate (Revise)(4.2.7)
5. Learn (Retain)(4.2.8)

It can be configured at the beginning with the same parameters as if every phase is run separately.

And the battery uses this CBR cycle to compute more than one consecutive cycle. To do it, first the battery dialog offers the functionality to load or select a new case list (see 4.2.3). If it is run by the command line, the new case list should be loaded from a file.

4.2.5 Retrieve

In this first step, the user is required to set up the new case (see 4.2.2). Then, the case library or case base, that is in charge to maintain these cases, return a set of retrieved cases when is asked with the new case as parameter. The set of cases represent the similar ones under the library criteria. The library criteria depends on the indexation that was set up. For flat indexation, it returns all the cases. This function can be seen as the first part of the retrieve (selection).

The next step is to look for the n (number of retrieved cases) cases that are more similar depending on the selected distance and its own distance configuration.

4.2.6 Reuse

The reuse phase, first load the reuse configuration, both introduced by the user or by the configuration file. In the case that is configured by the user, this phase allows to introduce custom formulas to calculate the adapted solution.

Secondly, compute the adaptation for each solution attribute according to its configuration as *NULL, MEAN/MODE, WEIGHTED MEAN/MODE or FORMULA*(see algorithms in section 4.4.6).

4.2.7 Revise

Currently, this step is introduced by the user. The user can use the default value and change it. Or, also can revise once the new case is already adapted.

4.2.8 Retain

Retains is twofold: first, there is the option to load the available retaining conditions and after that, the user can configure as many conditions as necessary, and each one with one custom configuration. Second, to decide whether to retain or not depend on the adaption mode (*Learn, NoLearn , LearnIf*). In the case of conditions, those have to be satisfied for learning the current case.

If this phase is configured by the GUI, then the user can check if the current case satisfy the set of conditions. If it is run by command line, then all the conditions in the *conditions.xml* will be loaded and configured with default values. However, the current case can be retained or not by user demand or by satisfying all the configured conditions.

4.2.9 Load Conditions

The conditions are stored in a *xml* file. Then, the user can load a new file with his/her preferences.

4.2.10 Update Utility

Any time that any algorithm related wit CBR is run, there is a listener that is in charge to update the *Utility* attribute.

- After Retrieve: it is updated the number of retrieves of each case that has been retrieved.
- After Reuse: it is updated the number of use field in all the cases that has been used to adapt the current case.
- After Revise: it is updated the number of success field of all the cases that has been used to created the adapted solution(we call the "source cases") to the revised case.

Besides to update the *utility* attribute, also, it creates the files with all the results.

4.2.11 Load Configuration

CBR has a global configuration of all its phases. This configuration can be specified through a configuration file. To make the process faster, the user can load another configuration file with the same format, but it is not necessary to include all the field.

4.3 Design

We have designed the CBR shell taking into account the system specification provided in chapter 3 (CBR description). In the next sections we are explained the main design decisions with respect to the architecture and the design patterns. Of course, in the design of the shell we have considered the design of GESCONDA II as well in order to guarantee a coherent integration between the two systems.

In next figure 4.2 is schematic depicted the whole CBR cycle:

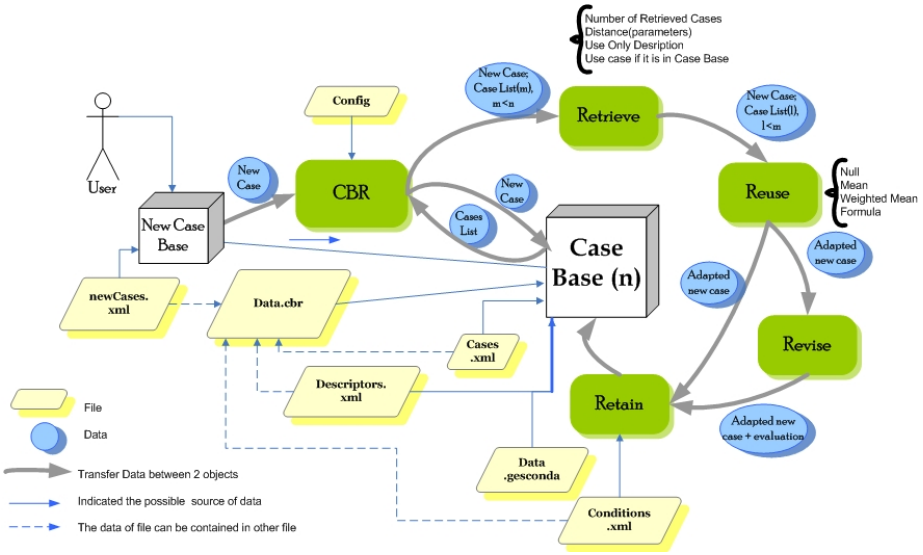


Figure 4.2: CBR schema of how it works

Finally, the figure 4.3 draws how the CBR works and when the configured file and data files are needed.

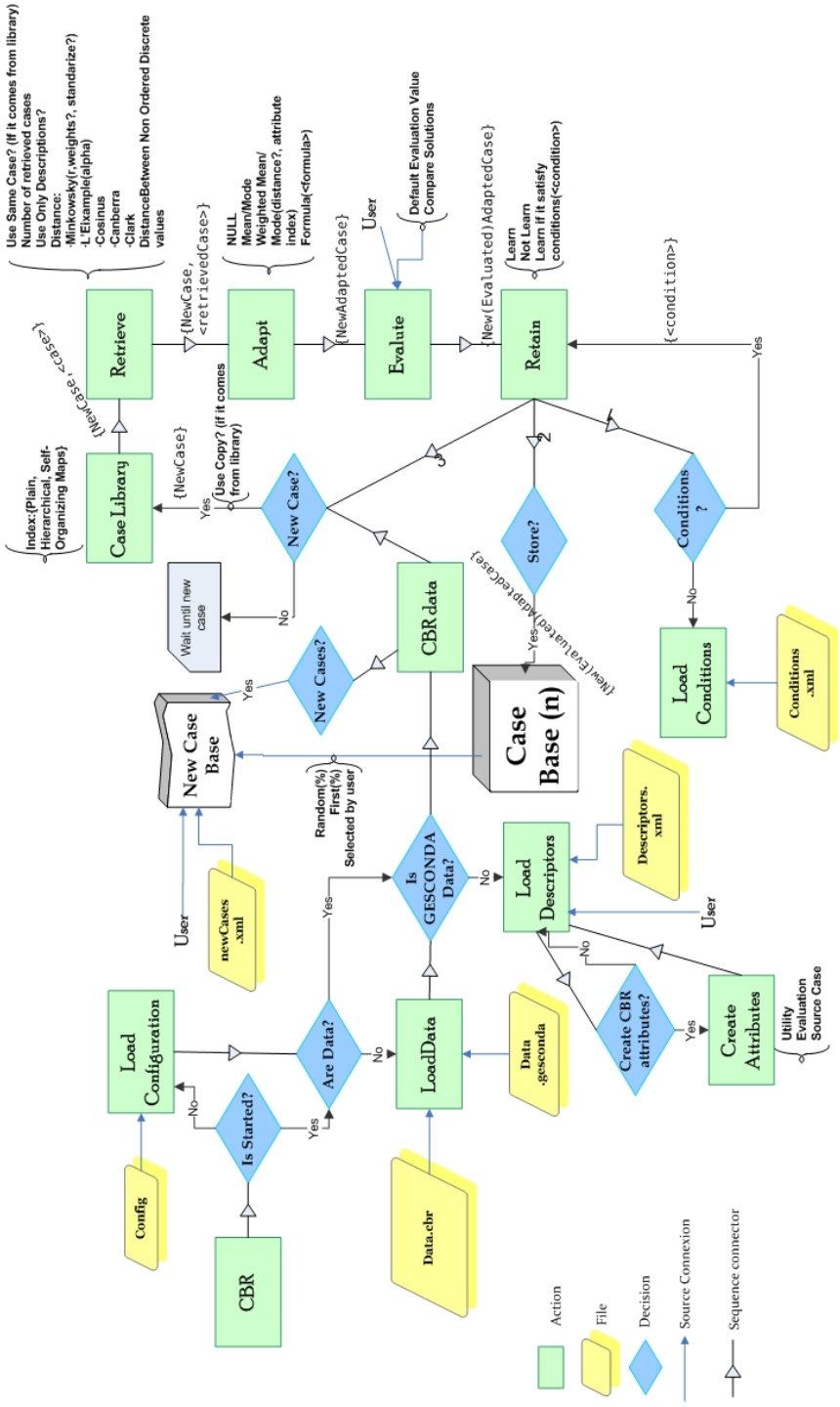


Figure 4.3: CBR diagram of how it works

4.3.1 Architecture

As far as technologies are concerned, we have reused the technologies used to implement GESCONDA II. Thus, we use Java, and *jFreeChart*. We have switched to Java 6 because it does not run into conflicts with the rest of code and it offers some advantages over version 5.

For the design we have followed the MVC architecture pattern used in GESCONDA II with the difference that we want to build up a system that could be used as a shell (by command line) as well. So, we have changed some aspects to achieve an independence from the interface. For instance, all parameters and variables to configure the system can be introduced by the interface or by means of an *xml* file (see CBR configuration 3.3).

4.3.2 Model

One of the main restrictions is that the GESCONDA structure cannot be modified in order to guarantee a smooth integrated system running. One of the non-functional requirement is to not modify GESCONDA code as far as possible. Also, it is not allowed to duplicate the data because is expensive in terms of memory, and after duplication, it would be not possible to use the GESCONDA functionality because it would not understand the new case structure as a GESCONDA instance (please refer to the GESCONDA structure defined in Section 2.2.1 to have a better understanding).

Thus, the idea is that CBR module cannot modify the GESCONDA data, and after CBR is executed, GESCONDA functionalities continue should working. In this way, the results of GESCONDA can be used in CBR because, most of the algorithms store the result in new variables which could be useful to improve the retrieval. Also, the functionality to prepare the data could be used, such as to change the current weights of the attributes.

Therefore, we already have a data structure called *CMInstancia*(instance), which is a vector of attributes. These attributes can be continuous or nominals, and one of the nominals can be selected as class attribute to use in supervised learning. Thus, it has been necessary to create a structure which allows to define which attributes belongs to the *Case Description* or to the *Case Solution* part. The user can define for each attribute its kind, and in addition, there is the chance to load this information by an *xml* file. Besides, the option to create an evaluation or utility attributes is offered as well. Hence, this structure enables the flexibility to select which are the *Case Description* or *Case Solution*, to identify others attributes which will be used by CBR, and to change such configuration at any time.

So, we had to forget about the typical CBR model (see figure 4.4), and therefore, to reorganize it to wrap the GESCONDA model ensuring that GESCONDA keeps on working correctly.

Our case is an imaginary class that includes *CMInstance* and all the descriptions of its attributes are common for all the cases. For this reason cases can be stored in the same place where all the cases are stored. This place is *CMTaula* for attributes and *CMCaseLibrary* for descriptors.

Figure 4.5 shows the initial GESCONDA model and the classes added to model by the CBR system (inside the rounded black rectangle). The rest of the classes inherited from the GESCONDA classes are not included (see

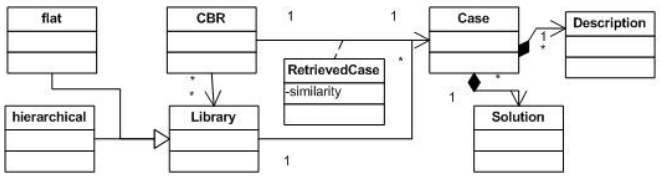


Figure 4.4: Typical CBR Model

GESCONDA model 2.2.1 for details). The following list describes every class:

CBR The main class that controls the data, configuration and the state of the CBR cycle in case that it is separately executed.

CBRConfig This class contains all the variables which do not depend on the data. In fact, this class maps the configuration file (see configuration 3.3).

CBRListener This class implements the observer pattern and its main function is to perform extra functions of CBR like updating the utility attribute or collecting data for future statistics.

CaseLibrary This class is in charge of maintaining the data. From this class it is possible to access data and the descriptors that are common for all the cases or instances.

IndexedCaseList It is an abstract class and depending on the configuration a subclass is chosen for selecting which kind of index to use. Its function *getCases(Case)* returns a list of cases that vary depending which one is selected (plain, hierarchical or Self-Organizating Maps).

Utility To include Utility, it is created as a new type of attribute. Thus, it needs the attribute, infoAttribute and value.

List It is a new type of attributes and it stores a list of *CMInstancia*. This attribute is useful to maintain the source cases, i.e. the cases that have been used to create the case solution.

Algorithm subclasses These classes are inherited from the Algorithm to be included as an algorithm of GESCONDA. Most of them are the algorithm used in the CBR cycle. For instance, the Battery algorithm executes a CBR cycle from a list of new cases with a specific configuration. Condition that are used in the Retain task, this is an abstract class that has the *satisfy* function and has to be coherent with the *condition.xml* that contains all the conditions and its values to be shown in the interface.

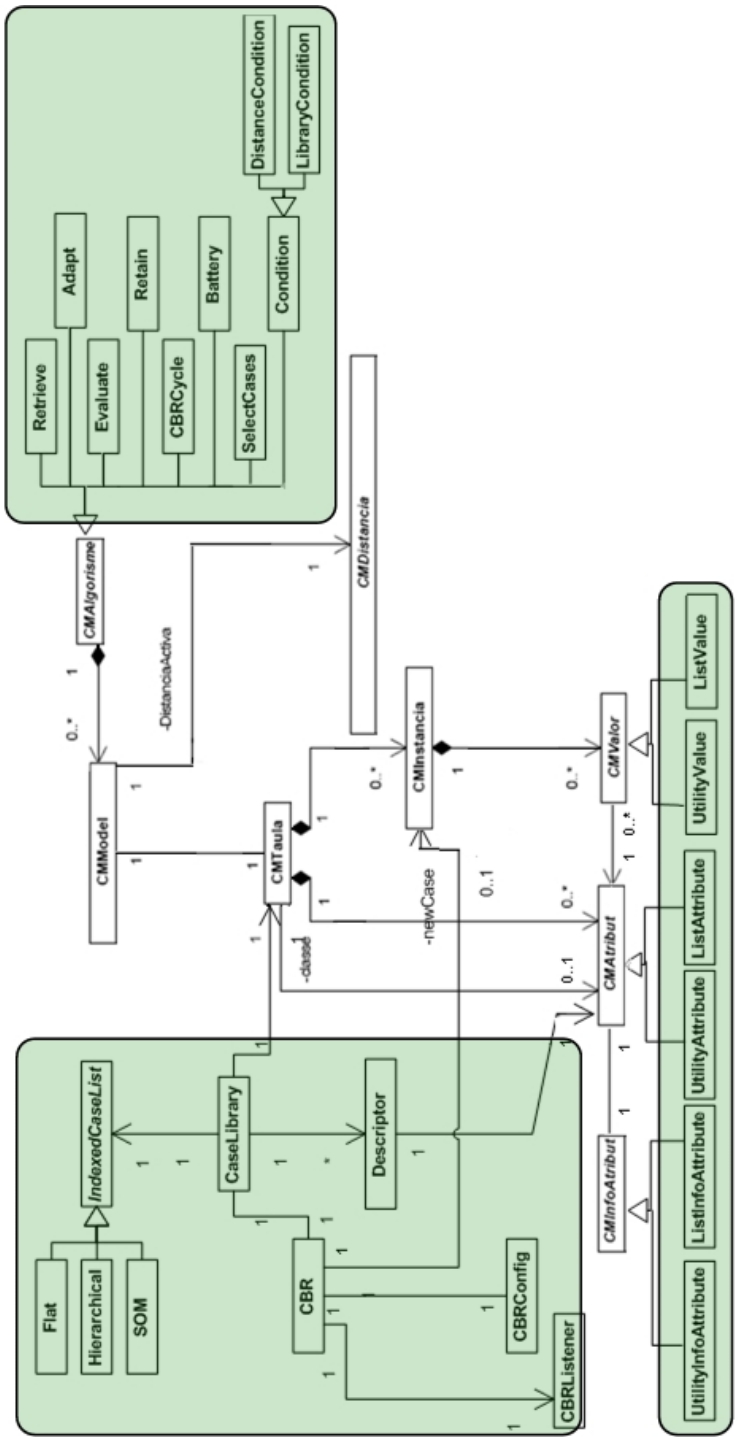


Figure 4.5: Gesconda Model with CBR Model

To adapt the CBR model to the GESCONDA model we introduce *CBR*, a class used to manage and configure the system to execute CBR. For instance, it maintains the task scheduling preventing the execution of a posterior task in the case the anterior has not been run yet and it stores the last solution of the cycle. *CBR* is a *Singleton* pattern and thus it is accessible by all the classes.

One of the most important classes of our model is the *CaseLibrary* which deals with the maintenance of data and their attributes. As seen before, the *CMTaula* contains the instances and their attributes and the *CaseLibrary* represents its complementary class. Thus, it has *Descriptors* and manages them to have one for each attribute. Furthermore it manages the data access. This class also creates the CBR attributes according to the configuration. Right now the indexation is only plain, so instances are returned as they come from *CMTaula*.

The *Descriptor* class contains the attribute to which refers and its type (description, solution, evaluation, utility or other cbr type). Its function is a tag for the attributes to recognize what type they are.

Since the attributes types of GESCONDA do not support all the data types, we create two more. The first *Utility* is an attribute that is a fraction. Its values are shown as continuous, but every case stores how many times has been retrieved, used to create a new solution and used successfully.

The second attribute type is *List*. This is a list of *CMInstancia* and is the most different of all because cannot be reduced as a number or a simple string. Anyway, its role is to store for each case what cases have been used to create its solution. Of course, we design a generic attribute for being used in the future as another type of attribute.

4.3.3 View

The model View contains the classes designed to support the interaction with the end-user. Following GESCONDA II, the functions offered in the main menu are introduced in *menu.xml* (see section 2.2). In this way code changes are not needed to add interface functionalities.

Nevertheless we have to introduce some changes to return the results in the same way that GESCONDA II does. Thus, we add a new tab in the main window responsible to show all the references to CBR. In next figure 4.6 the aspect of this tab integrated in GESCONDA II is shown.

The rest of view consists of independent dialogs for the configuration of each task. Each dialog is shown inside another one depending on the task flow. To reduce the complexity we have reduced the dialogs because the functions *CBR cycle* and *Battery* show all the parameters for each task in the cycle. Thus, if these tasks are executed separately a new dialog is opened only with the respective options for this task:

1. If data is loaded with any GESCONDA format then the descriptors do not exist. Thus, a dialog is opened to ask the user whether she/he wants to load data from a file or set up them manually. In figure 4.7 is described the manual setup.

2. After data and descriptors are loaded, if the function of battery is selected, then a configuration dialog is opened. This dialog is depicted in figure 4.8.

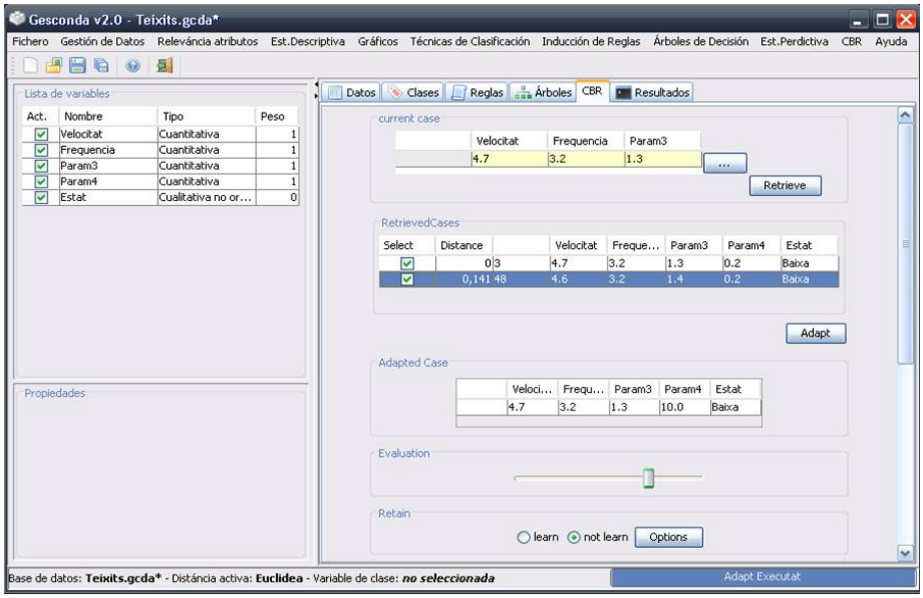


Figure 4.6: View of the CBR tab in GESCONDA II

This functionality has the option to load the cases which will be used as new cases. There are two sources. The first one is from the loaded data, and it can be the *first* one or *random*, and the user can indicate the percentage to be used. The second one is loaded from an external file.

3. In adaptation, for each attribute there are some options (see 4.4.6). For numerical ones the option to create a specific formula is provided as shown in figure 4.9

Figure 4.7: Dialogs that allows to set up or change the case structure defining which belong to the description case or solution case. Also, to create the evaluation, utility attribute or both.

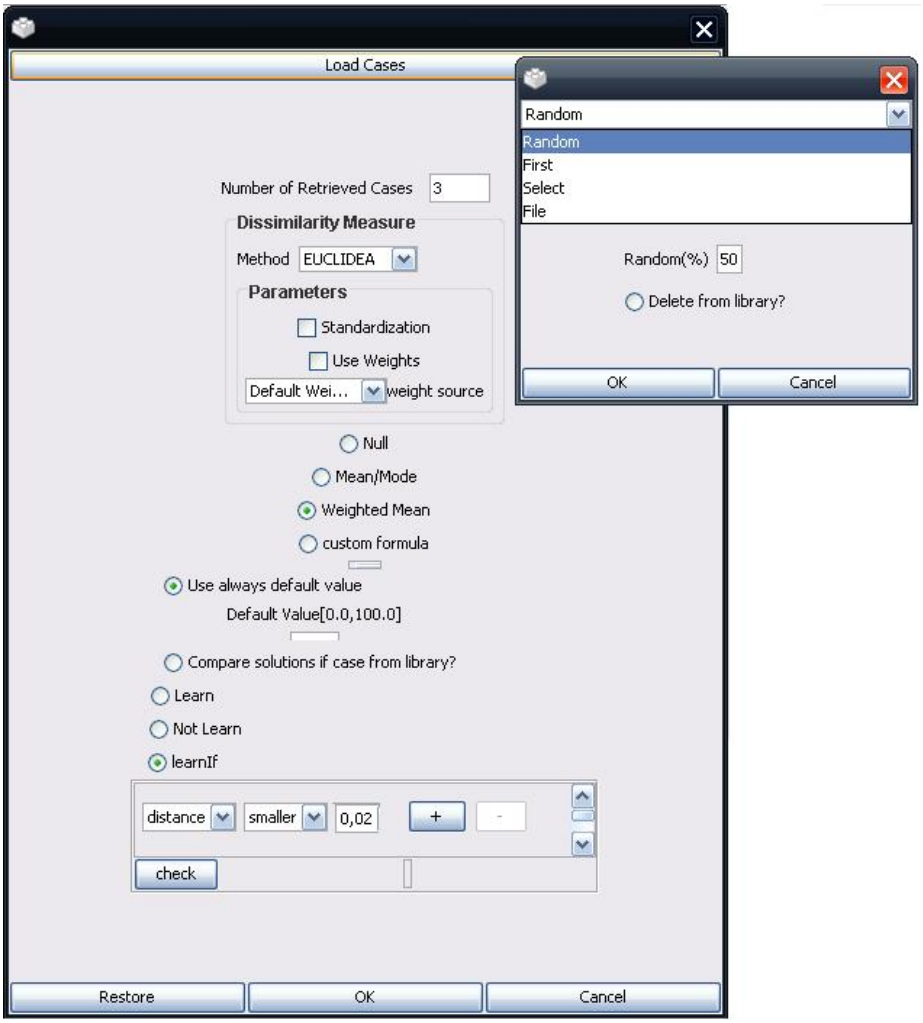


Figure 4.8: View of the Battery Configuration, where it is possible to distinguish all the phases of CBR

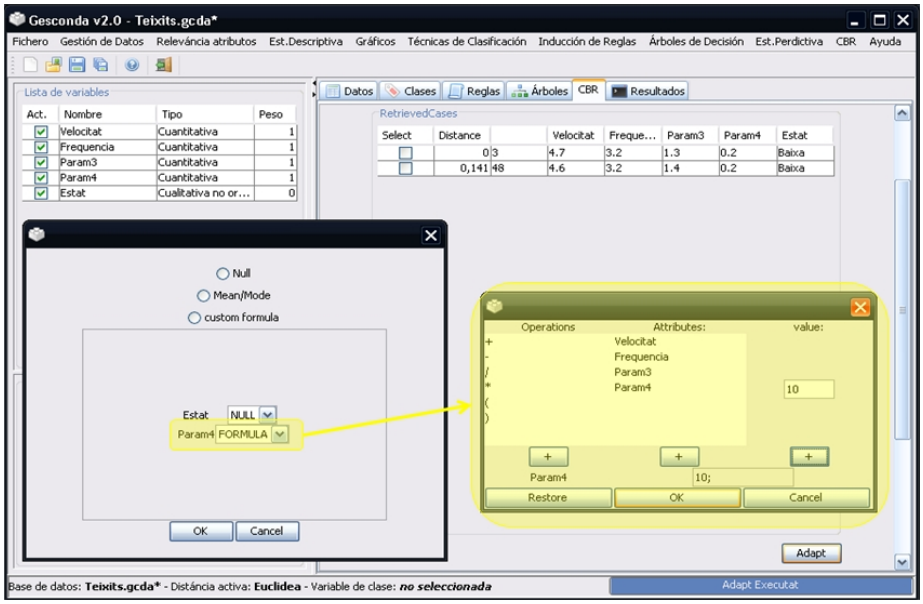


Figure 4.9: Dialog to create a formula for an attribute adaptation

4.3.4 Controller

4. Finally, when the battery is run, the CBR tab is returned showing the last cycle which has been assessed (see figure 4.6).

This part follows the schema of GESCONDA II controllers. Thus, all the actions are subclasses of the *CCAccio* (controller class). The main function of CBR controllers is to manage which dialogs to show for obtaining the parameters and launch the pertinent algorithms.

As it is mentioned in GESCONDA (see 2.2), there is a controller structure that allows to execute the algorithm in a new thread without blocking the whole application.

4.4 Implementation

In this section the development of this shell is detailed. As said in the previously sections, our implementation has been restricted by the fact that it has to adapt to the GESCONDA system and it cannot be intrusive.

In the implementation we have followed the GESCONDA II style, with the difference that we want to be able to execute from command line as well. For this reason our layers do not have any dependencies between them. Since GESCONDA II is implemented bounding the functionalities to its interface; we need that our model let be independent from the interface intervention. In this way, a custom configuration is loaded by a file as well.

We enumerate the style to name the classes in the following list to avoid confusions with the names:

CM* : Model Class

CC* : Controller Class

CCAccio* : Controller Class in charge of a function. We use "CCAction*" because our code is written in English.

4.4.1 Model

The model layer is responsible to manage different aspects of the CBR functionality. First, it loads the necessary data (instances, descriptors and configuration). Next, but not less important it collects information while CBR is running and it is in charge of updating information used, and/or generating logs used for statistics.

The main class is *CMCBR*. It uses the Singleton pattern to be unique and always accessible. This class is in charge of maintaining the state of the CBR cycle, of connecting with the case base and of configuration. It also contains a listener class that covers everything that is not related to the own CBR cycle. Finally, it stores all the results in logs. In addition, this layer contains classes which can parse xml files.

Configuration

When GESCONDA II is launched *CMCBR* is created. The *CMCBR* initialization creates the class *CBRConfig* that loads the "config" file. To load this file we use Java library which offers Java "Properties". Such library allows to treat a file as a hash map. For this purpose the "Config" file is appended (see for more detail section B)

Listener

The class *CBRListener* is the implementation of the *Observer* pattern. Through the classes "PropertyChangeSupport" and "PropertyChangeEvent", the class *CBRListener* becomes a listener of CBR .

Each time that any CBR algorithm finishes, the *CBRListener* updates the following information:

- The last date that has been used for each case involved in the algorithm.

- Retrieval:
  - Total number of retrieves of the system.
  - the number of retrieves for each retrieved case.
- Reuse: the number of reuse for all the cases that has been involved for solving the new solution
- Evaluation: if this phase has been successful, all the cases which have taken part in the reuse.[1]
- Retain: the date of creation if the new case has been stored. This date is also updated when the case is introduced as an instance through any functionality of GESCONDA .

Moreover every time that an algorithm finishes, *CBRListener* stores the information in a file. In this way a trace log of the system execution is always available. To maintain this file, we use a simple library for logging. Logging is a package of the Commons project included in Apache[2].

XML Readers

The data of CBR is stored in xml format. Thus, SAX[3] has been used to read and write xml files.

4.4.2 View

The view layer is implemented using the Java *Swing* library. Almost all dialogs of the CBR module collect the parameters and configuration needed to execute CBR algorithms. Besides, to the tab inserted in GESCONDA II that also shows the results.

The implementation is as reusable or modular as we need, in order to reuse all the panels that are similar. For example, the figure 4.8 in subsection view of design, it depicts that it is formed by the same panels that has been used in the different dialogs when the algorithms are executed separately.

One thing that all the collected dialogs and panels implement, is the *PropertyChange*. In fact, these fire a *PropertyChange* with the new parameters.

Eventually, we reuse all views from GESCONDA II which shown what we want. In other situations, we had to implement it again. For instance, the dialog from GESCONDA II that allows to change the distance just shows the names and we re-implement it for our module to configure all possible parameters. In figure 4.8 below to "Load Cases" there is the retrieval configuration with "Dissimilarity Measure" panel. Inside this panel it is possible to change distance measure, and to each distance the panel "parameters" is updated with its own parameters.

[1] In this point we had to create a new type of attribute List which stores the list of cases used to form the solution.

[2] `http://commons.apache.org/logging/`

[3] `http://www.saxproject.org/`

4.4.3 Controller

The controller layer includes classes which mediate the view class and model class. To connect them there is also an intermediary class to pass the parameters collected in views.

Most of these classes are initiated from the main menu of GESCONDA II with the aim of executing some algorithm. So, the controller shows the pertinent dialogs to collect the parameters and transfer those to algorithm while are launched.

Some actions are launched from the model class *CMCBR* like the action that allows to introduce the descriptors either by specifying through the GUI or by loading a file(*descriptors.xml*).

Actions can be launched from another action as well, such as the case of selecting a set of cases to launch a battery of cycles.

4.4.4 Algorithms

All the classes that implement an algorithm inherit it from *CMAlgorisme*. The main functions are *executar* and *algorisme*, both are abstract. The first function is called from the controller which manages the threads. *algorisme* is the algorithm itself.

Besides, it implements some functions to be called by the classes that manage the view and the time. The current instance is shown while the algorithm is running.

Initially the constructor asks for the name and the data (*CMTAula*). So, if the algorithm has parameters, then all the algorithms that we implement have to override the constructor to pass the parameters in a unique object. In alternative parameters can be specified one by one.

The unique object is a HashMap<DTOParamenter, Object>. This object represents the DTO pattern in the sense that is useful to transfer all the parameters among the classes. One example is when the configuration is obtained through the interface. Then, the controller can ask to this window for its parameters and directly transfer then to the algorithm before its execution. The advantage is that it is not necessary to fill all the parameters, but only the ones the user wants to change from the default values.

DTOParameter belongs to an enumeration of all the existing parameter types. Some examples are: name, attribute weights, learn mode, normalize.

4.4.5 Retrieval

The retrieval algorithm implements the second part of the Retrieval task. At this point, the selection of the cases which depends on the indexation has already been done. Particularly, in the plain indexation, all the cases are selected; in hierarchical indexation, the cases of a sub-hierarchy are the selected ones; and in Self-Organizing Maps, the cases belonging to a region.

The parameters needed by this algorithm are:

- New Case: the case to be compared with the others.
- Selected Cases: cases that has been selected as the proper cases to be compared with the new case depending on the indexation criteria.

- Distance: The type of distance to be used: *Euclidean, L'eixample, Caberra, etc.*. This distance will be already configured with its parameters.
- Weights: this is an option parameter that represents the weights to use in the assessment of the distance, and are not the weights which are defined in GESCONDA. These weights only exist in this class and do not affect any other class.
- Use Only Descriptions: The usual retrieve only compares the description attributes, but it is possible to include the rest of parameters configuring this parameter.
- Ignore Same Case: This option implies that the new case is from the library, and then if it is true the new case will be not compared with itself. It could be interesting to compare if the user is making some testing.

Notice that the "number of the retrieved cases" is not a parameter, because at the same time that is needed to select the n most similar cases, all the distance are assessed. Thus, all the cases with its distance are stored. So, there is a function (*topNRetrieved*) that given a number returns the number of cases that are most similar. A particularity of this function is that it returns the minimal sorted set of cases that contains at least the number of cases specified such that it does not exist another case whose distance is equal or smaller and it does not belong to this set.

Therefore, the retrieval algorithm iterates over all the cases from the selection of the library (depends on the indexing criteria) comparing with the new case and storing the distance between them.

4.4.6 Reuse

The adaptation of the new case depends on the way it has been configured. This algorithm needs to know the data which is available all the time, the configuration of the user, each selection of each attribute and whether it is a formula itself. The modes are: NULL, MEAN(MODE), WEIGHTED MEAN(MODE) or FORMULA.

The parameters needed by this algorithm are the following:

- New Case
- List of cases to be used in adaptation. By default they are the cases that have been retrieved. Before to adapt, when the retrieved cases are shown, the user can select a subset of them.
- Mode of reusing: global o for each solution attribute. If any mode is a formula, then this formula must be indicated.
- Index of an attribute: this parameter is optional and corresponds to the index of the attribute to be used for pondering the mean or mode.
- List of float: this parameter is optional and represents the distances for pondering the mean. This list should be ordered like the cases list.

Depending on the selected mode the algorithm is different.

NULL Adaptation

NULL adaptation copies directly each value of each solution attribute to the solution attribute of the new case. In this mode the number of the cases which have been selected to be used in adaptation does not matter, as the most similar case is taken.

MEAN or MODE adaptation

This reuse method depends on the type of attribute over which the adaptation is calculated. For continuous attributes the mean of the values of the selected cases is assessed. However for discrete attributes, the solution attribute is placed with the mode of this attribute values from the selected cases.

In fact, this adaptation is a particular case of weighted mean, where all weights are the same.

Weighted MEAN or MODE adaptation

When this algorithm was explained in 3.6, we described that it could be useful to ponder with the distance or utility of the selected cases. In this way the most useful or closest cases have a bigger influence over the solution. To make this algorithm more flexible, the index of the attribute that contains the values to ponder is not fixed, and by default, it corresponds to the utility.

As the index and the list of distance are optional, and there is not restrictions whether one exists and the other does not, we implement a more generic reuse:a generic weighting mean (depicted in equation 4.1).

$$
\begin{aligned}
s_i(newCase) = weightedMean = \sum_{j=0}^{R} s_i(retrievedCase_j) * weigth_j, \\
weight_j = \frac{p_j}{\sum_{k=0}^{R} p_k} \\
\text{where,} \\
s_i : \text{solution attribute with index i} \\
p_j : \begin{cases} 1 - distance(retrievedCase_j, newCase) \\ attribute_{index}(retrievedCase_j) \\ (1 - distance(retrievedCase_j, newCase)) * attribute_{index} \end{cases} \\
R : \text{number of retrieved cases} \\
index : \text{parameter index}
\end{aligned}
\tag{4.1}
$$

Finally, to calculate the weighted mode the next equation 4.2 is applied:

$$s_i(newCase) = weightedMode = max_{C_k}(\sum_{j=0}^{N} p_j C_{kj})$$

where,

s_i : solution attribute with index i

C_k : one discrete value from the set of k discretes values

$$p_j : \begin{cases} 1 - distance(retrievedCase_j, newCase) \\ attribute_{index}(retrievedCase_j) \\ (1 - distance(retrievedCase_j, newCase)) * attribute_{index} \end{cases}$$

N : number of retrieved cases that $s_i = C_k$

$index$: parameter index

(4.2)

FORMULA adaptation

This mode of reusing the retrieved cases is quite different in the sense that involves more classes. To each attribute that is going to be solved by a formula, a new class *Formula* is needed. As described in section reuse 3.6, the formula is introduced as a text. This text has a specific syntax according to the Formula class. The syntax is a list of symbols and each one is separated by the next with a "separator character". Every symbol is a number, attribute name or an operator. The symbols have to follow an order which permits that they are treated as a formula.

This Formula class is a first idea. As we will explain in future work 6.1, this class with its subsequent classes represented in the interface, could be changed with those that let introduce all type of formulas, including logical ones.

Listing 4.1: Formula reuse code

```
public abstract class CMAdapt extends CMAlgorisme
{
  private CMInstancia newCase;
    .
    .
    .
  /*This function will assess the entire reuse algorithm,
    applying to each attribute the subsequent algorithm
    depending on configuration  */
  @Override
  public int algorisme()
  {
    .
    .
    for(CMAtribut at: solutionAttributes)
    {
      switch(mode)
      {
        case NULL:
            .
```

```
                .
            case FORMULA:
              //obtain the formula introduced in the GUI
              String text = (String)modes.get(at);
              //new formula class is built up with the text
              Formula formula = new Formula(text);
              //getAttributeNames returns a list of attri-
              //bute'names that has been utilized in the
              //formula
              //getAttributesValues returns for each attri-
              //bute its value or its mean if there are more
              // than one case for the reuse
              HashMap<String,Double> values =
                   getAttributeValues
                              (formula.getAttributeNames());
              // calculate the formula
              double result = formula.evaluate(values);
              // new adapted value(result) is stored
              currentCase.setValorAtribut
                              (at.getPosicio(), result);
          }
        }
      }
      .
      .
}
```

4.4.7 Evaluation

The implementation of this algorithm is simple. Until now, there is not a mechanism which can be automatically used to assess the evaluation itself. Generally, it is supposed that only an expert could select the evaluation value, or alternatively, that a default value is provided. Hence, the algorithm just receives the evaluation value and assign it to the new case.

4.4.8 Retain

We present a flexible implementation where a user developer can add learning conditions. Learning conditions are described in a xml file(*conditions.xml*) and the class used to open the file must inherit from the interface condition (*CMCondition*). The formalization of a condition consists of three elements: name, the condition itself and a value. The *xml* tree is a list of conditions like type of the following (in listing 4.2 there is an example of this *xml* file):

Conditions Is a list which specifies the available conditions for this condition and the available values depending on the condition chosen

Condition Defines every new condition class. Its parameters are:

- **name** The name of the condition used to identify and to show it in the GUI
- **algorithmClass** The Java Class that extends *CMCondition Interface*

Operator Defines the condition function

name The string that indicates which operator is considered

valueType If it is numerical, the end-user or the person who configures the system will introduce a numerical value. Otherwise if categorical, a list of the possible values has to be specified (as we can see in the listing 4.2)

value contains the values that an operator can take. If the parameter *default* is *true*, then the value is the default value to be used for showing or if it is an automatic process to be used as election of the user. If they are values belonging to an operator with categorical type, then the values are the possible categories.

Listing 4.2: Condition file

```
<?xml version="1.0" encoding="UTF-8"?>
<conditions>
   <condition name="Distance"
         algorithmClass="gesconda.CBR.model.CMDistanceCondition">
      <operator name=">" valueType="numerical"/>
         <value default="true">4,5</value>
      <operator name="=" valueType= "categorical">
         <value>1</value>
         <value>3</value>
         <value>5</value>
         <value default="true">3</value>
      </operator>
   </condition>
   <condition name="Type_of_CaseBase"
         algorithmClass="gesconda.CBR.model.CMIndexingCondition">
      <operator name="is" valueType="categorical">
         <value>Flat</value>
         <value>Hierarchical</value>
         <value>SOM</value>
      </operator>
   </condition>
</conditions>
```

Therefore, the extension of the retain step has consisted in creating the new condition/s and its code. New conditions have access to all the data. The process consists of evaluating whether all the conditions are satisfied. In the figure 4.10 it is possible to see the GUI that allows to configure which conditions have to satisfy the new situation. The conditions are a conjunction, i.e. all the conditions must be satisfied before to learn or to store the case in the library.

As it has already been mentioned, to implement the conditions, the user needs to implement a new Java Class which extends CMCondition Interface. In the listing 4.3, there are the most important function of this class. To execute the retain algorithm, all the conditions will be evaluated calling the method *satisfy*.

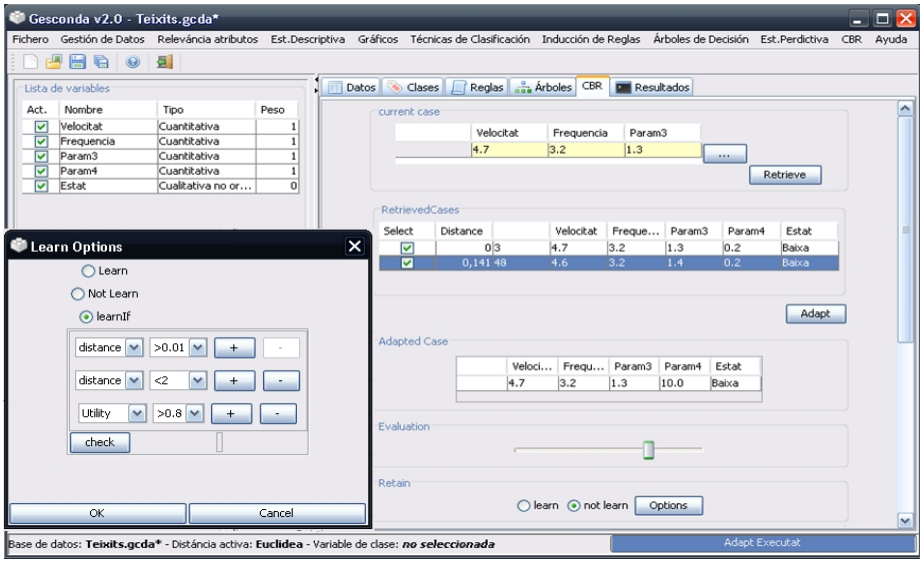


Figure 4.10: CBR tab with GUI retain for adding conditions

Listing 4.3: CMCondition Interface

```
public abstract class CMCondition extends CMAlgorisme
{
  protected String conditionName = null;
  protected String operatorName = null;
  protected String value = null;
     .
     .
     .
  /* This function will assess if the condition is
     satisfied in this moment for the new case */
  public abstract boolean satisfy(String value,
                        String operator, CMCBR cbr);

  public abstract String getConditionName();

  /* Return the possible operators */
  public abstract String[] getOperators();

  /* Return the current operator */
  public abstract String getCurrentOperator();

  /* If the condition is a fix value introduced
     by the user */
  public abstract String getValueName();

  /* If the condition values are a restringed set */
  public abstract String[] getPossibleValues();

  /* Add to the list of operators and its possible
     values or null if is numerical*/
  public void addOperator(String operator,
                          String[] values)
}
```

The main idea is to give the possibility to create a different criteria depending on the expert user and let the opportunity of extending the code easily without the need to modify the rest of the classes. The mechanism consists of creating a class that extends *CMCondition* which contains all the logic. The main functionality of the *xml* file is to define the condition identifier with its Java class.

Practical example: Negative Similar Case

This example has been described in section 3.6. This policy tries to avoid storing those cases that have a similar one and on top of that this similar case has been evaluated as negative.

The code 4.4 reflects the main function that has to be implemented to incorporate this condition in the system. The next code 4.5 has to be placed in the condition file.

Listing 4.4: NegativeSimilarCase: function satisfy

```
public class NegativeSimilarCase extends CMCondition
{
  public boolean satisfy(String value,
                         String operator, CMCBR cbr)
  {
    double evaluation = getEvaluation(value);
    double similarity = getSimilarity(value);

    CMInstance newCase = cbr.getNewCase();
    for(CMInstancia c: cbr.getCases())
    {
      if(c.evaluation() < evaluation)
      {
        if(distance(newCase,c)<=(1-similarity))
        {
          return false;
        }
      }
    }
    //There are not a case that satisfy these two
    //conditions, so the case could be stored.
    return true;
  }
}
```

Listing 4.5: Representation of Negative Similar Case in xml file

```
<condition name="NonNegativeSimilarCase"
      algorithmClass="gesconda.CBR.model.NegativeSimilarCase">
   <operator name="negativeSimilar" valueType="numerical"/>
      <value default="true">{0.5,0.9}</value>
   </operator>
</condition>
```

Chapter 5

Evaluation

The evaluation of the CBR shell deployed could be twofold: from a more theoretical point of view, and from an experimental assessment.

In advance and summarizing, the CBR shell developed is a promising tool for Data-Intensive Case-Based Reasoning Systems, and provides some advantages previously mentioned over existing CBR shells.

5.1 Theoretical assessment

From a theoretical perspective, out CBR shell could be analyzed and compared against some other CBR shells in the literature. The advantages and shortcoming can be taken into account, to get a final assessment of the CBR shells.

There are some CBR shells available in the literature. Some of them are commercial products deployed in software companies (Remind, CBRExpress, etc.), while others are academic products developed in some research groups in universities (Caspian, JColibri, etc.). Nevertheless, most of them do not allow implementing the whole basic CBR cycle in a very flexible way: retrieval, reuse, revise and retain. On the other hand, most of them do not provide the friendly use of some data analysis techniques integrated in the same tool as in our approach. In the next table we can see a comparison of the revised tools against our system:

Most of the shells are extensible, especially those that are written in Java.

JColibri2 is probably our strongest competitor when talking about functionalities. JColibri2 is one of shells that is the most popular nowadays, at least as an academic tool. Also, it is more used as an API and receives contributions from others entities extending its functionality.

The case structure for all of the reviewed systems is built up with a list of attributes. Some of these systems support text as CBRExpress, ReMind and AIAICBR. But only JColibri2 and IUCBRF along with our system have extra attributes for assessing CBR algorithms. The last thing we want to remark is that not all of the systems allow the solution to contain more than one attribute.

Our system has a lack of indexation, at least for the moment. Comparing with the rest of systems, IUCBRF and ESTEEM allow to be hierarchically organized, even though indexes are not automatically calculated.

As it was mentioned before, most of the systems are focused on retrieval.

Name	Language	Case	Indexing	Retrieval
Caspian	CASL	Attribute List Solution:n Evaluation	Flat	Search guided by user index Selection: most similar, manually
JColibri2	Java	Attribute List Solution:n Result Reason of result	Flat	knn Distance global and local Selection: more similars or more differents
IUCBRF	Java	Attribute List Solution:n Use Count Time of creation Creation' Source	Flat B-tree-backet (index by user)	knn
AIAICBR	Java	Attribute List Solution:1	Flat	knn weight: by user or genetic algorithms Distance:global Trigrams
myCBR	Java	Attribute List Solution:1	Flat	Distance: global and local GUI to define distances
CBRExpress	Asymetric ToolBox	Attribute List	Flat	Trigrams Heuristic
ReMind	C	Attribute List	Decision Tree	knn automatic or guied by user SQL-queries
ESTEEM	Intellicop's Kappa-PC; C++	Attribute List	ID3 automatic weighting index by user	ID3
CBR-GESCONDA	Java	Attribute List Solution:n Evaluation Utility Source Cases	Flat Hierarchical, SOM[a]	knn Distance: global

[a] not yet implemented

Name	Reuse	Revise	Retain	Extensible	Other
Caspian	Repair rules by user	user	user	-	-
JColibri2	Null Numerical Direct proportion	user	user	Easily	API Persistence: Ontologies, files or DDBB
IUCBRF	Null Weighted Average Weighted Majority	-	custom triggers Delete old cases periodically	Yes	API model
AIAICBR	-	-	-	-	-
myCBR	-	user	-	Yes	Protégé plugin Scriptable similarities in Jython
CBRExpress	-	-	User	-	-
ReMind	Adjustment of difference between new case and retrieved case	-	-	Yes	Text
ESTEEM	Rules by user	-	-	-	Nested Cases Multi case base
CBR-GESCONDA	Null Weighted Mean Custom Formula	user	User Configurable conditions	Easily	Integrated in Data Mining tool Multiformat

Table 5.1: Comparison of CBR Shells

In this part, we are more similar to JColibri2 because we allow to configure the distances used. JColibri2 also offers local weighting and different selection distance depending on the attribute type.

In adaptation, we can see that some systems do not do anything. Others allow the user to insert rules. But without the user help, we can see that everything offered is quite simple as it is a difficult topic. In IUCBRF as well as in our system, a weighted average can be computed. While in IUCBR the weight means the number of times that a case has been used, in our system we can weight by several attributes, including the utility attribute (see 3.9), which can be configured with different criteria.

As we can see in the table 5.1, none shell implements a generic evaluation. In fact, when revising the literature, the systems explain what phases are done, while directly ignoring the rest of phases.

Eventually, the retain task is only covered by IUCBRF, which allows to create triggers and, as a maintenance policy, to delete old cases. Our system faces it through customizable conditions. We offer some of them and a default configuration. Then, the user can modify the configuration and even introduce new conditions.

The most significant difference over other systems is that this CBR shell is integrated within an Intelligent Data Analysis Tool (GESCONDA), easing the data preparation, data filtering, data relevance and data visualization steps. Also, the degree of flexibility to cope with difficult domains, and the degree of customization by the user are tightly outstanding.

5.2 Experimental evaluation

The CBR shell developed has been preliminary tested with some target domains. Some of them coming from the UCI repository, and others coming from real-world data. All functionalities have been evaluated from the retrieve step to the retain step.

The CBR shell has been evaluated according to the competence of its provided solutions.

The CBR shell let the users to model whatever domain mainly characterized by a big number of features and data (i.e., a Data-Intensive Case-Based System). Especially mentionable are the different ways and strategies available in the reuse and retain step. Also, the possibility to use all the available capabilities within the Intelligent Data Analysis tool (GESCONDA) allow making easier the task of data preparing, data filtering, feature weighting, data visualization, etc., which is not commonly found in most of the available CBR shells in the literature.

5.2.1 Testing the Iris dataset

We start using Iris Data from the UCI repository[1]. This data is composed by 150 instances classified in 3 classes (iris-setosa, iris-virgilica and iris-versicolor). The data do not have missing values and one class is linearly separable from the other two; the latter are not linearly separable from each other.

[1] `http://archive.ics.uci.edu/ml/datasets/Iris`

Distance	Success Frequency	Bad Classified Instances	ReRunned Well Classified
Euclidean	73/75	120, 135	135
Euclidean	73/75	71, 135	135 , 71
Weighted Euclidean	73/75	71, 134	71
Clark	71/75	69, 71, 73, 78	69, 73
Canberra	72/75	107, 120, 134	

Table 5.2: Experimental test of CBR with different distance measures applied to Iris Data

Algorithm	Success Freq per class	Success Frequency
ID3 with C4.5 prunning	50/50, 49/50, 45/50	146/150
Prism(25 rules)	22/22, 25/29, 23/23	70/74
Kmeans	50/50, 48/50, 36/50	134/150
Marata	50/50, 48/50, 36/50	134/150

Table 5.3: Experimental test with different algorithm applied to Iris Data

First, we define the case structure indicating that the solution is the class attribute. After we generated a battery of CBR cycles where we select the half of data as new cases with the random option. And we remove the new cases from the dataset.

Using the euclidean distance without standardization and without weights we obtain that 97.3% of cases has been successfully classified, using 3 cases in retrieval and using the adaption scheme by the mode.

In table 5.2.1, we show the results with different distances. All of them, have been adapted by the mode and the number of retrieved cases is equal to three. In this table, we show the instances that have been bad classified and in the last column, the cases that has been re-runned with all the case base, and has been well classified.

Finally, we see in table 5.2.1 the iris data classified by other methods.

5.2.2 Testing the Abalone dataset

The next dataset used for testing our CBR shell is Abalone. Also, it is from UCI repository[2]. Its objective is predict the age of abalone from physical measurements.

The data contains 4177 instance and 9 attributes. 1 attribute is discrete and the rest are continuous. The age is represented by the rings attribute, since the age is +1.5 of the value of rings. This attribute *rings* has 29 values and can be seen both as a discrete or continuous.

Our first test consists in run a battery of the 10% (approximately 500 cases) of data that has been selected randomly. In table 5.2.2 we can see the main configuration and its results. The discretization used for attribute Rings is the same that has been used in the proposed articles by own repository [Clark et al., 1996]. This discretization groups ring classes into 1-8, 9 and 10, and 11 on. Also,

[2] `http://archive.ics.uci.edu/ml/datasets/Abalone`

Distance	Num. Retrieves	Weigths	Solution	Success (%)	Student's $\mathcal{T}test$	Other
Euclidean	3	-	Sex, Rings(C)	73.9	0.08	
Euclidean	3	-	Sex	77.4		
Euclidean	5	-	Rings(3)	79.3		standardize
Euclidean	5	-	Rings(3)	78		
Euclidean	3	-	Rings(3)	83.7		
Euclidean	3	-	Rings(3)	82.9		
Euclidean	5	-	Rings(3)	63.5*	0.04	
L'Eixample	3	PROJ	Rings(3)	82		α =0.8
L'Eixample	5	PROJ	Rings(3)	80.8		α =1
L'Eixample	3	CVD	Rings(3)	82.6		α =0.8
Canberra	3	-	Rings(3)	80		
Cosinus	3	-	Rings(C)	64.5*	0.42	
Cosinus	5	-	Rings(3)	79.3		

Table 5.4: Results of applying different CBR configurations to Abalone dataset. The column "Solution" remarks if the attribute Rings is continuous with "C" or discrete by the number of classes. "*" These values are assessed with the continuous values, this are encapsulated into the 3 groups(<8; [9,10]; >11) and compared with the real values

there are some reference about studies on this data. In the article [Waugh, 1995] where his results fluctuate around 65%

In addition, to assess the success of the discrete attributes we compare our solutions with the real values. Then we measure the percentage of success. With continuous attributes is more difficult, because is difficult to compare the results. We decide to evaluate using the Student's $\mathcal{T}test$. With the difference among our results and the real ones, we can know the variance and the typical deviation. This test is normally accepted for p-values < 0.5, this means that there is a 95% of possibilities that the distance between both results (predicted and real) is closed to 0.

Chapter 6

Conclusions and Future Work

Along this document we have introduced our CBR approach, integrated in the last version of GESCONDA . Another objective was to achieve an integration that enabled to use all the functionalities of GESCONDA independently of where the data came from.

The advantage to have integrated it in GESCONDA is the possibility of using its functionalities at any moment that is required. Besides, GESCONDA uses the typical data formats in Data Mining. This fact allows to use the available databases directly in the CBR system without the need to create a new case base.

This integration has been one of the most challenging and time-consuming parts of the work, since the original GESCONDA functionalities were totally bound to its interface (and even interface-driven), and some bugs had to be fixed to allow proper integration.

The main objective was to create a CBR shell that was suitable to be configured by the user, with no fixed parameter or option. We mostly reached this objective with the exception of the evaluation phase, in which the evaluation has to be inserted by the user. In addition, the configuration can be done through an interface or by changing a file or some files. The first configuration, with an interface, is easy to do without the need of an expert user. In the moment the CBR shell needs some data, it informs the user and for each step it shows the pertinent dialog with the default configuration, which corresponds to that in the configuration corresponding file.

To make easy to work with the system, a configuration file exists where all the parameters independently from the data are possible to change.

Besides, the system keeps creating a file with what it has assessed. For each phase, it writes what has been used and the results.

Eventually, the system can be run through GESCONDA interface or by command line. If the user just wants to use CBR and knows what is the configuration he/she wants, then he can specify the files containing the data and the new cases to be calculated.

From both the theoretical assessment against other available CBR shells and from the experimental work done, it can be concluded, that CBR-GESCONDA is a very promising tool as a framework to develop CBR systems for many domains In next section we explain the future work to be done, because this is a first version and it can be extended in many aspects to improve its functionality

providing more options to the user.

6.1 Future Work

Along the documentation, we have mentioned several times that some functions or options are not yet implemented. These will be the first ones to be implemented in the next future. First, we want to extend the memory organization as we introduced in the description of Case Library(section 3.2). Probably, it will be our next step.

6.1.1 Memory Organization

About the topic of the organization of the case base, we want to expand the types of indexation with Self Organizing Maps(SOM) and Hierarchical Structures. We have already considered them both and know what the algorithms are, although we have not yet implemented them. Of course, more structures or algorithms related to memory organization can be easily integrated when needed.

Currently, only the flat memory is implemented because no changes had to be done, since instances in GESCONDA II are stored in a list. Nevertheless, we ensure that the most similar case is selected. This similarity is based on distances and no other calculations which could find other relations between cases are taken into account.

The implementation is already prepared for these new library types, and just by implementing the algorithms and adding the subsequent options into the pertinent dialog to show the user will be enough to integrate them.

6.1.2 Evaluation

In the system that we have developed, the evaluation or revise task does not offer any automatic option to be calculated. So, an important future work is to research in this subject, and develop a final system that offers functionalities to each phase without user intervention.

6.1.3 Case Base Maintenance

Besides, the matter of Case Base Maintenance is mentioned in the Retain or Learning phase. But, despite it is an important issue we have not covered it in depth in this work. Fernando Orduña[1], a PHD student at the Knowledge Engineering and Machine Learning KEMLg group, is centering his work in case maintenance, the different techniques and when are they more useful to be used. Hence, one of our future objectives is adding his work in the system.

6.1.4 Local Weighting

Another topic is about using distance measures that make use of weighting schemes.

When we talked about weighting attributes, we defined it as a technique to rank the relevance of the attributes. We talked about global weights, that is, a

[1] http://www.lsi.upc.edu/~forduna/

weight per attribute. Nevertheless, it is also possible to talk about local weights. Local weights are not focused on the whole attribute, but in the particular values of this attribute.

In the literature, there is a lot of information about this topic. We found some articles that show how CBR retrieval improves with local weighting [Núñez et al., 2003] [Park et al., 2004]. Besides, diverse techniques can be applied depending on the type of data. When the data can be guided by its class (supervised) then methods as Value Difference Metric (VDM) [Stanfill & D.Waltz, 1986] or EBL [Núñez et al., 2003] can be used. For non-parametric data, the use of kernels [Ferraty & Vieu, 2006] is very popular.

We would like to extend the functionality of weighting by introducing this local aspect. For doing it, it is necessary to change some parts of GESCONDA II. At present, GESCONDA II model is prepared to handle these local weights, but no algorithm is implemented to use them.

6.1.5 Selection of Distance Measures

Another important topic is the configuration of distance measures. The current implementation of GESCONDA II just supports the definition of one distance measure for all the instances. So, we would like to change that part, to be more flexible and allow the user to select a different distance measure for each attribute.

Since each attribute is different from the others, might be worthy to configure each one separately. In related work (section 2.3) we can see that JColibri2 (subsection 2.3.2) implements this functionality.

Finally, we want our calculation of a weight distance to be more general:

$$distance(C_i, C_j) = \sum_{k=1}^{n} w_k \cdot attr\_distance(C_{ik}, C_{jk}) \tag{6.1}$$

Where *attr_distance* could be different for each attribute. For its assessment, other weights depending on the values could be also used.

6.1.6 General Lines

Another topic for further research is the interpretation of the results. Thus, we want to integrate skills that make possible the monitoring of the system and the capability of generating some graphical charts about several important parameters of the system (case library size, retrieval time, etc.). These capabilities will be available in next releases of the tool.

Eventually, we have created an extended system. So, we keep searching about CBR and techniques to improve it, and we will integrate them as well. All of the methods that will be developed in GESCONDA will also be used in CBR if they are suitable. A CBR system which has been designed to be extended is never likely to be closed. As research keeps generating new possibilities, they will be welcome to be considered for our system.

References

[com, 1992] (1992). Compaq installs cbr help desk. *Intelligent Systems Report.* 1.1

[3cb, 1992] (1992). Three cases: all cbr, but worlds apart - case-based reasoning program development tools are described: Remind, esteem and cbr express; use of cbr express by american airlines and a typical case are discussed. 2.3.7, 2.3.8, 2.3.9

[cas, 1995a] (1995a). Casl description document v1.3. 2.3.1

[cas, 1995b] (1995b). *An Introductory Guide to Caspian.* 2.3.1

[giz, 1996] (1996). Broderbund's new online technical support receives two prestigious awards; "gizmotapper" technical support on the internet endorsed as premier online support system. 1.1

[Aamodt & Plaza, 1994] Aamodt, A. & Plaza, E. (1994). Case-based reasoning:foundational issues, methodological varations and system approaches. *AI Communications.IOS Press, Vol. 7: 1,pp,39-59.* 2.1, 2.1, 3.2.2, 3.6, 3.7

[Althoff, 1999] Althoff, K.-D. (1999). *Case Base Reasoning Research and Development.* Springer Verlag. 2.1

[Althoff et al., 1995] Althoff, K. D., Auriol, E., Barletta, R., & Manago, M. (1995). A review of industrial case-based reasoning tools. 2.3.7

[Baccigalupo & Plaza, 2007] Baccigalupo, C. & Plaza, E. (2007). A case-based song scheduler for group customised radio. *International Conference on Case Based Reasoning(ICCBR'07).* 3.7

[Bergmann, 2001] Bergmann, R. (2001). *Highlights of the European INRECA Projects*, (pp. 1–15). Springer Berlin/ heidelberg. 2.3.6, 2.3.6

[Bergmann & Stahl, 1998] Bergmann, R. & Stahl, A. (1998). : (pp. 25–36).: Springer. 2.3.6

[Bogaerts & Leake, 2005] Bogaerts, S. & Leake, D. (2005). *A Framework for Rapid and Modular Case-Based Reasoning System Development.* Technical report. 2.3.3

[Clark et al., 1996] Clark, D., Schreter, Z., & Adams, A. (1996). A quantitative comparison of dystal and backpropagation. *Australian Conference on Neural Networks (ACNN'96).* 5.2.2

[Corchado et al., 2004] Corchado, E. S., Corchado, J. M., & Aiken, J. (2004). Ibr retrieval method based on topology preserving mappings. In *Journal of Experimental & Theoretical Artificial Intelligence, Volume 16, Number 3*: Taylor and Francis Ltd. 2.1

[Corchado & Lees, 2001] Corchado, J. & Lees, B. (2001). A hybrid case-based model for forecasting. *Applied Artificial Intelligence, Volume 15, number 2*, (pp. 105–127). 3.6

[de la Rosa et al., 2007] de la Rosa, T., Olaya, A. G., & Borrajo, D. (2007). Using cases utility for heuristic planning improvement. In *Case-Baed Reasoning Research and Development*, volume Volume 4626/2007 (pp. 137–148).: Springer Berlin / Heidelberg. 3.9.1

[de Mantaras et al., 2006] de Mantaras, R. L., McSherryA, D., Bridge, D., Leake, D., Smith, B., Craw, S., Faltings, B., Maher, M. L., Michael T, C., Forbus, K., Keane, M., Aadmodt, A., & Watson, I. (2006). Retrieval, reuse, revision, and retention in cbr. *The Knowledge Engineering Review*, 20(3), 215–240. 2.1, 2.1

[Duda et al., 2004] Duda, R. O., Hart, P. E., & Stork, D. G. (2004). *Pattern Classification*. A Wiley-Interscience Publication. A.2.1, A.3

[Eidenberger, 2003] Eidenberger, H. (2003). Distance measures for mpeg-7-based retrieval. *International Multimedia Conference archive Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*. A.4.1, A.4.2

[E.R.Bareiss, 1988] E.R.Bareiss (1988). *PROTOS: A Unified Approach to Concept Representation, Classification, and learning*. Technical report, Dept. of Computer Science, Vanderbilt University, Nashville, TN. 1.1

[Fdez-Riverola & Corchado, 2000] Fdez-Riverola, F. & Corchado, J. (2000). Sistemas híbridos neuro-simbólicos:una revisión. *Inteligencia Artificial, Revista Iberoamericana de IA*, 4(11), 12–26. 3.6

[Ferraty & Vieu, 2006] Ferraty, F. & Vieu, P. (2006). Local weighting of functional variables. In *Nonparametric Functional Data Analysis* (pp. 37–44). 6.1.4

[Fodor, 2002] Fodor, I. K. (2002). A survey of dimension reduction techniques. *UCRL*. 3.2.3

[Francis & Ram, 1994] Francis, A. G. & Ram, A. (1994). *A comparative Utility Analysis of Case-Based Reasoning and Control-Rule Learning Systems*. AAAI Techincal Report WS-94-01. 3.9

[Gibert & Perez-Bonilla, 2006] Gibert, K. & Perez-Bonilla, A. (2006). *Revised boxplot based discretization as the kernel of automatic interpretation of classes using numerical variables*, (pp. 229–237). Batagelj, V.and Bock, H.-H. and Ferligoj, A.and Ziberna, A. (Eds.). 2.2.2

[Gibert et al., 2006] Gibert, K., Sánchez-Marré, M., & Rodríguez-Roda, I. (2006). Gesconda: An intelligent data analysis system for knowledge discovery and management in environmental data bases. *Environmental Modelling & Software 21(1):116-121.* 2.2, 2.2.2

[Hammond & Head, 1990] Hammond, K. & Head, R. (1990). Case-based planning: A framework for planning from experience. *Cognitive Science*, 14, 385–443. 1.1

[Helen et al., 2003] Helen, C., Causton, C., & ans Alvis Brazma, J. Q. (2003). *Microarray gene expression data analysis.* Wiley-Blackwell. A.2.1

[Iglezakis et al., 2004] Iglezakis, I., Reinartz, T., & Roth-Berghofer, T. (2004). Maintenance memories: Beyond concepts and techniques for case base maintenance. In P. Funk & P. A. G. Calero (Eds.), *Advances in Case-Based Reasoning* (pp. 227–241).: Springer-Verlag. 3.8

[Kolodner, 1993] Kolodner, J. (1993). *Case-Based Reasoning (Morgan Kaufmann Series in Representation & Reasoning).* Morgan Kaufmann. 2.1

[Koton, 1989] Koton, P. (1989). *Using experience in learning and problem solving.* Technical report, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science. 1.1

[Lance & Williams, 1967] Lance, G. N. & Williams, W. T. (1967). Mixed-data classificatory programs i - agglomerative systems. *Australian Computer Journal*, 1(1), 15–20. A.4, A.4.1

[Leake & Wilson, 2000] Leake, D. B. & Wilson, D. C. (2000). Remembering why to remember: Performance-guided case-base maintenance. In *Fifth European Workshop on Case-Based Reasoning* (pp. 161–172).: Springer Verlag. 3.8

[Lieber, 2007] Lieber, J. (2007). Application of the revision theory to adaptation in case-based reasoning: the conservative adaptation. *7th International Conference on Case-Based Reasoning - ICCBR'07.* 3.6

[Lim et al., 1991] Lim, J., Lui, H., & Tan, A. (1991). Inside: a connectionist case-based diagnostic expert system that learns incrementally. *IEEE International Joint Conference on Neural Networks.* 3.8.3

[Margarit, 2007] Margarit, E. O. (2007). Gesconda ii: Integració de components, redisseny, reimplementació i ampliació de funcionalitat. Master's thesis, Universitat Politécnica de Catalunya. 2.2

[Michael et al., 2001] Michael, R. B., Richter, M. M., Schmitt, S., Stahl, A., & Vollrath, I. (2001). Utility-oriented matching: A new research direction for case-based reasoning. In *In Professionelles Wissensmanagement: Erfahrungen und Visionen. Proceedings of the 1st Conference on Professional Knowledge Management. Shaker* (pp. 264–274). 3.7

[Minton, 1990] Minton (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42.* 3.8, 3.9

[nez et al., 2003] nez, H. N., Sánchez-Marré, M., & Cortés, U. (2003). Similarity measures in instance-based reasoning. 2.2.2, A.1, A.4

[Núñez & Sánchez-Marré, 2004] Núñez, H. & Sánchez-Marré, M. (2004). Instance-based learning techniques of unsupervised feature weighting do not perform so badly! *ECAI: 16th European COnference on Artificial Intelligence.* 2.2.2

[Núñez et al., 2003] Núñez, H., Sánchez-Marré, M., & Cortés, U. (2003). Improving similarity assessment with entropy-based local weighting. *ICCBR:international conference on case-based reasoning.* 2.2.2, 6.1.4

[Orduña-Cabrera & Sánchez-Marré, 2008] Orduña-Cabrera, F. & Sánchez-Marré, M. (2008). Case base maintenance: Terms and directions. 2.1, 3.9

[Orduña-Cabrera & Sánchez-Marré, 2009] Orduña-Cabrera, F. & Sánchez-Marré, M. (2009). The dynamic adaptative case-based library for continuous domains. *CCIA.* 3.9

[Park et al., 2004] Park, J. H., Im, K. H., Shin, C.-K., & Park, S. C. (2004). Mbnr: Case-based reasoning with local feature weighting by neural network. *Applied Intelligence*, 21, 265–276. 6.1.4

[Recío-García et al., 2008] Recío-García, J., Díaz-Agudo, B., & González-Calero, P. (2008). *jCOLIBRI2 Tutorial.* Technical report, Department of Software Engineering and Artificial Intelligence. University Complutense of Madrid. 2.3.2

[Riesbeck & Schank, 1989] Riesbeck, C. K. & Schank, R. C. (1989). *Inside case-based reasoning.* Lawrence Erlbaum Associates, Pubs., Hillsdale, N.J. 2.1, 3.5

[Romdhane & Lamontagne, 2008] Romdhane, H. & Lamontagne, L. (2008). : (pp. 474–486).: Springer Berlin/Heidelberg. 3.9

[Ross, 1989] Ross, B. H. (1989). Some psychological results on case-based reasoning. *Case-based Reasoning Workshop, DARPA.* 2.1

[Sánchez-Marré et al., 1999] Sánchez-Marré, M., Cortés, U., Roda, I. R., & Poch, M. (1999). Sustainable case learning for continuous domains. In *Environmental Modelling and Software Volume 14, Issue 5* (pp. 349–357). A.1

[Sánchez-Marré et al., 2004] Sánchez-Marré, M., Gibert, K., & Rodríguez-Roda, I. (2004). Gesconda: A tool for knowledge discovery and data mining in environmental databases. In *In e-Environment: Progress and Challenge (Eds. P. Prastacos, U. Cortés, J.L. Díaz de León y M. Murillo). In the series Research on Computing Science, Vol. 11, CIC, Mexico.* (pp. 348–364). 2.2

[Sánchez-Marré et al., 1998] Sánchez-Marré, M., R.-Roda, I., & Comas, Q. (1998). L'eixample distance: a new similarity measure for case retrieval. *1st Catalan Conference on Artificial Intelligence(CCIA'98).* A.1

[Schank, 1982] Schank, R. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press. 2.1

[Schank & Abelson, 1977] Schank, R. & Abelson, R. (1977). *Scripts, plans, goals and understanding.* Lawrence Erlbaum. 2.1

[Schulz, 1999] Schulz, S. (1999). Cbr-works - a state-of-the-art shell for case-based application building. In *Proceedings of the 7th German Workshop on Case-Based Reasoning, GWCBR'99, Wrzburg* (pp. 3–5).: Springer-Verlag. 2.3.6

[Schumacher, 2002] Schumacher, J. (2002). Empolis orenge - an open platform for knowledge management applications. In *Proceedings of the 1st German Workshop on on Experience Management* (pp. 61–62).: GI. 2.3.6

[Slade, 1991] Slade, S. (1991). Case-based reasoning: A research paradigm. *AI magazine Volume 12 Number 1.* 3.2.2

[Sovat et al., 2001] Sovat, R. B., Aluísio, S. M., & de Carvalho, A. (2001). Rabeca: A hybrid case-based reasoning development environment. In *Tools with Artificial Intelligence, Proceedings of the 13th International Conference* (pp. 61–68). 2.1

[Stahl & R.Roth-Berghofer, 2008] Stahl, A. & R.Roth-Berghofer, T. (2008). *Rapid Prototyping of CBR Applications with the Open Source Tool myCBR.* Technical report, German Research Center for Artificial Intelligence(DFKI)GmbH Image Understanding and Pattern Recognition Department(IUPR). 2.3.5

[Stanfill & D.Waltz, 1986] Stanfill, C. & D.Waltz (1986). Toward memory-based reasoning. *Communications of the ACM.* 6.1.4

[Veloso & Carbonell, 1993] Veloso, M. M. & Carbonell, J. G. (1993). Derivational analogy in prodigy: automating case acquisition, storage and utilization. (pp. 249–278). 3.2.2

[Watson & Marir, 1994] Watson, I. & Marir, F. (1994). Case-based reasoning: A review. *AI-CBR, Dept.of Computer Science, University of Auckland, New Zealand.* 1, 2.3.7, 2.3.8

[Waugh, 1995] Waugh, S. (1995). *Extending and benchmarking Cascade-Correlation.* PhD thesis, Computer Science Department, University of Tasmania. 5.2.2

[Zilles, 2009] Zilles, L. (2009). *myCBR Tutorial.* 2.3.5

Appendix A

Similarity Measures

We've decided to cover this topic in a different section because it's very important, in general, for several algorithms in CBR systems. Most of them use a similarity measure to assess its results, being so depended on how is this similarity function.

To understand the concept of "Similarity", the different definitions follow:

> *"Similarity is some degree of symmetry in either analogy and resemblance between two or more concepts or objects. The notion of similarity rests either on exact or approximate repetitions of patterns in the compared items" by Wikipedia*[1]
>
> *"Similar: of the same kind in appearance, character, or quantity, without being identical" by Oxford dictionary*[2]
>
> *"Similar: having characteristics in common : strictly comparable" by Merriam-Webster dictionary*[3]
>
> *"Distance: the degree or amount of separation between two points, lines, surfaces, or objects" by Merriam-Webster dictionary*[4]

So, a similarity metric is merely a function that gives a generalized scalar distance between two arguments - patterns, vectors or instances.

The objects that are used in GESCONDA II and in CBR are vectors of values, where each vector has the same definition for each position (attribute). So, the vectors could be seen as a points in a space where the coordinates are the attributes. The coordinates are not necessary orthogonal, depending on the correlation among the attributes.

Calculating the similarity or distances between two instances depends on the chosen distance (in the next subsection we define the distances supported by GESCONDA II) and, also, on the weights defined for each attribute. We want to remark that feature selection and weighting are an important part of the analysis the data because those techniques help to clean up the noise, irrelevant

[1] `http://en.wikipedia.org/wiki/Similarity`
[2] `http://www.askoxford.com/concise_oed/similar?view=uk`
[3] `http://www.merriam-webster.com/dictionary/similar`
[4] `http://www.merriam-webster.com/dictionary/distance`

or redundant data from the data. To describe this preprocesses is out of our scope, but we emphasize the weights that result from this algorithm because they are used in the retrieval task of the CBR cycle.

GESCONDA offers some automatic algorithm for weighting the attributes , such as the gradient (see GESCONDA functionalities2.2.2). These algorithms can also be specified by the end user. In fact, it does not matter where the weights come, but it is important to use them to assess the distance by giving more importance to relevant attributes.

This technique has two aspects. The first one weights whole attributes, that is, every attributes has a unique weight for all its values. The second one weights the different values into an attribute. To use this last aspect, continuous attributes should be discretized in order to weight each interval. Unfortunalety, GESCONDA does not implement the local weighting, although it's a future issue(see 6.1).

A.1 L'Eixample

An exponential weighting transformation would lead a better attribute relevance characterization, when the number of attributes is very high [nez et al., 2003]. L'Eixample [Sánchez-Marré et al., 1998] [Sánchez-Marré et al., 1999] is a normalized weight-sensitive distance function. It takes into account the different nature of the quantitative or qualitative values of the continuous attributes depending on their relevance.

L'Eixample distance is sensitive to weights. For the most important continuous attributes - that is, weight $> \alpha$ - the distance is computed based on their qualitative values(a previous discretization is done for quantitative attributes). This implies that relevant attributes having the same qualitative value are equals, and having different qualitative values are very different, even when a continuous measure would be very small. And for those less relevant ones - that is, weight $< \alpha$ - the distance is computed based on their quantitative values. This implies that non-relevant attributes having the same qualitative value are not equals, and having different qualitative values, are more similar. L'Eixample measure is defined as:

$$d(C_i, C_j) = \frac{\sum_{k=1}^{n} e^{w_i} \times d(A_{ki}, A_{kj})}{\sum_{k=1}^{n} e^{w_i}} \tag{A.1}$$

$$d(A_{ki}, A_{kj}) = \begin{cases} \frac{|qtv(A_{ki}) - qtv(A_{kj})|}{upperval(A_k) - lowerval(A_k))} & \text{if } A_k \text{ is a continuous attribute and } w_k < \alpha \\ \frac{|qlv(A_{ki}) - qlv(A_{kj})|}{\#mod(A_k) - 1} & \text{if } A_k \text{ is a continuous attribute and } w_k > \alpha \\ & \text{or } A_i \text{ is an ordered discrete attribute} \\ 1 - \delta_{qlv(A_{ki}), qlv(A_{kj})} & \text{if } A_k \text{ is non a ordered discrete attribute} \end{cases}$$

where:

C_i : the case i
A_k : attribute k
W_k : weight of attribute k
A_{ki} : value of the attribute k in the case i
$qtv(A_{ki})$: quantitative value of A_{ki}
$upperval(A_k)$: upper quantitative value of A_k
$lowerval(A_k)$: lower quantitative value of A_k
α : cut point on the weight of the attributes
$qlv(A_{ki})$: qualitative value of A_{ki}
$\#mod(A_k)$: of modalities (categories) of A_k
δ : delta of Kronecker

A.2 Minkowski

The Minkowski distance is a metric on Euclidean space which can be considered as a generalization of both the Euclidean distance and the Manhattan distance[5]. It uses the Chebyshev distance when the order tends to infinity, that corresponds between two points to the maximum of projected distances.

The variable called *order*(r in equation A.2) indicates the distances type: the higher the value of r, the more significant is the contribution of the largest components$(x_{ik} - x_{jk})$. If $r = 1$ Manhattan distance is obtained and for $r = 2$, Euclidean distance.

The next equation A.2 is the formalization of this measure used in GESCONDA:

$$d(x_i, x_j) = \left(\frac{\sum_{k=1}^{K} w_k * dif_k(x_i, x_j)^r}{WeightSum}\right)^{\frac{1}{r}}, r \geq 1 \qquad \text{(A.2)}$$

[5] http://en.wikipedia.org/wiki/Minkowski_distance

where:

$k : number\ of\ input\ attributes$

r : order

$x_i : case\ i$

$$w_k : \begin{cases} weight(attribute\ k) & \text{use weights is activated} \\ 1 & \text{use weights is not activated} \end{cases}$$

$$WeightSum : \begin{cases} \sum_{k=0}^{K} weight(attrib_k) & \text{if use weights is activated} \\ 1 & \text{if use weights in not activated} \end{cases}$$

$$dif_k(x_i, x_j) : \begin{cases} |normalized(x_{ik}) - normalized(x_{jk})| & \text{if attribute k is continuous} \\ & normalize = true \\ |x_{ik} - x_{jk}| & \text{if attribute k is continuous} \\ & normalize = false \\ \frac{|x_{ik} - x_{jk}|}{NumberOfModalities(k)-1} & \text{if attribute k is ordered discrete} \\ 0 & \text{if attribute k is discrete \&} \\ & value_k(x_i) == value_k(x_j) \\ constant & \text{if attribute k is discrete \&} \\ & value_k(x_i) \neq value_k(x_j) \end{cases}$$

A.2.1 Euclidean

Euclidean distance is the most common and intuitive distance measure. Euclidean distance between two points in two dimensions can be expressed using Pythagora's theorem [Helen et al., 2003].

The Euclidean distance will be invariant to translations or rotations in feature space. However, it will not be invariant to linear transformations in general. That's one reason why the data should be normalized [Duda et al., 2004].

A.2.2 Manhattan

Manhattan distance, also called Taxicab geometry or Rectilinear distance, considered by Hermann Minkowski in the 19th century, is a form of geometry in which the usual metric of Euclidean geometry is replaced by a new metric in which the distance between two points is the sum of the (absolute) differences of their coordinates. The Manhattan name alludes to the grid layout of most streets on the island of Manhattan, which causes the shortest path a car could take between two points in the city to have length equal to the points' distance in taxicab geometry. Formally, the distance between two vectors in an n-dimensional real vector space with fixed Cartesian coordinate system is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes.

A.3 Cosinus Distance

Cosine similarity is a measure of similarity between two vectors by finding the cosine of the angle between them. This measure is invariant to rotation and

dilation, though it is not invariant to translation and general linear transformation [Duda et al., 2004].

In GESCONDA the Cosinus Distance is implemented following the next equation A.3:

$$d(X_i, X_j) = 1 - \cos(\theta) = 1 - \frac{X_i \cdot X_j}{\|X_i\| \, \|X_j\|} = 1 - \frac{\sum_{k=0}^{K} x_{ik} * x_{jk}}{\sqrt{\sum_{k=0}^{K} (x_{jk})^2 * \sum_{k=0}^{K} (x_{jk})^2}} \tag{A.3}$$

where:

X_i : *case i*

k : number of input attributes

$$x_{ik} : \begin{cases} normalized(value_k(x_i)) * weight(k) & \text{if attribute k is continuous} \\ unified(value_k(x_i)) * weight(k) & \text{if attribute k is ordered discrete} \\ 0 & \text{if attribute k is discrete \&} \\ & value_k(x_i) \neq value_k(x_j) \\ weight(k) & \text{if attribute k is discrete \&} \\ & value_k(x_i) == value_k(x_j) \end{cases}$$

A.4 Unweighted Similarity Measures

These similarity metrics, defined in Lance and Williams (1966) [Lance & Williams, 1967], are very sensitive to small changes close to $x_{ik} = 0 = x_{jk}$, and can be less reliable if the (x_{ik}) are sample estimates of some quantities. An advantage of these metrics is that they do not need a previous normalization [nez et al., 2003]. And as the name indicates, these measures ignore the weights attributes.

A.4.1 Canberra

This similarity ignores the attributes's weights. The Canberra distance is a metric function often used for data scattered around an origin[6]. It was introduced in 1966 ([Lance & Williams, 1967]). Canberra metric can be seen as a normalized form of Manhatthan distance [Eidenberger, 2003].

The equation A.4 is used in GESCONDA:

$$d(x_i, x_j) = \sum_{k=1}^{k} dif_i(x_{ik}, x_{jk}) \tag{A.4}$$

[6] http://www.code10.info/index.php?option=com_content&view=article&id=49:article_canberra-distance&catid=38:cat_coding_algorithms_data-similarity&Itemid=57

$$dif_i(x_{ik}, x_{jk}) = \begin{cases} \frac{|x_{ik}-x_{jk}|}{|x_{ik}+x_{jk}|} & \text{if attribute k is continuous} \\ \frac{|x_{ik}-x_{jk}|}{|x_{ik}+x_{jk}+2|} & \text{if attribute k is ordered discrete} \\ 0 & \text{if attribute k is discrete \&} \\ & value_k(x_i) \neq value_k(x_j) \\ constant & \text{if attribute k is discrete \&} \\ & value_k(x_i) = value_k(x_j) \end{cases}$$

A.4.2 Clark

Like the previous distance Canberra, this distance does not take in account the attributes's weights. The formula A.5 is similar to the Canberra formula A.4

In the same way as Canberra is a normalization of Manhattan distance, Clark's divergence coefficient is a normalized version of Euclidean measure [Eidenberger, 2003].

$$d(x_i, x_j) = \sum_{k=1}^{k} dif_i(x_{ik}, x_{jk}) \tag{A.5}$$

$$dif_i(x_{ik}, x_{jk}) = \begin{cases} \frac{|x_{ik}-x_{jk}|^2}{|x_{ik}+x_{jk}|^2} & \text{if attribute k is continuous} \\ \frac{|x_{ik}-x_{jk}|^2}{|x_{ik}+x_{jk}+2|^2} & \text{if attribute k is ordered discrete} \\ 0 & \text{if attribute k is discrete \&} \\ & value_k(x_i) \neq value_k(x_j) \\ constant & \text{if attribute k is discrete \&} \\ & value_k(x_i) = value_k(x_j) \end{cases}$$

Appendix B

Global configuration of the CBR Shell

```
 ########################################################################
 # Cofiguration File:                                                   #
 # This file contains all the variables than affect the global CBR      #
 # cycle                                                                #
 ########################################################################

######################## Case Base Organization ########################

## index indicates what are the indexing that is currently used, if
## the organization needs any parameter this is specify below to that,
## only the corresponding parameter will be read
## values = {PLAIN, HIERARCHICAL, SOM}

CaseBase.index = PLAIN


######################## New Case ######################################

## When a case is used from the library exist choices:##
## useCopy indicates that if the case is from the case base, then
## the values are copy in a new case and will be used as a new one.
## If is false then the case will be updated with the new proposed
##   solution
##   values = {true, false}

NewCase.useCopy = true;


##   In useCopy == true and newcase is from the casebase, then is
##   possible to configure that in the retrieve step, this new case
##   will be compared with itself or not
##   values = {true, false}
```

```
NewCase.useSameCaseInRetrieve = false


######################### Retrieve ###################################

##   The retrieve task calls the library with the new case and it
##   returns a set of cases depending on its indextation. So, this
##   task is centered in compare the newcase with all the cases that
##   have been returned by the library

##   numRetrievedCases indicates the number of the cases which will be
##   returned by this retrieve algorithm and lately used by adaptation.
##   The most similar numRetrievedCases will be returned if these
##   exist. If the next case after the numRetrievedCases has the same
##   distance, it will be returned too, in fact will be returned all
##   that have the same distance than the numRetrievedCases case.
##   values = [0..inf]

Retrieve.numRetrievedCases = 3

##   useOnlyDescriptions indicates if the comparation of the cases will
##   be among the description attributes or can be used the solution
##   ones and evaluation if exist
##   values = {true, false}

Retrieve.useOnlyDescriptions = true

######################### Retrieve- Distance ##########################

##   To compare the cases can be selected the distances used. For each
##   one, some parameters can be variables too.
##   Distance.type = {EUCLIDEA, MANHATTAN, MINKOWSKI, EIXAMPLE,
##                    CLARK, COSINUS, CANBERRA}

Retrieve.Distance.type = EUCLIDEA

##  DistanceBetweenTwoNonOrderedDiscretValues is common to all distance
##  types, and marks how to penalized when are different
##  values = [-inf, inf] ##default = 1

Retrieve.Distance.DistanceBtwNonOrderDiscreValues = 1

##   r is a variable power to specify the distance type.
##   Power = 1 Manhattan distance, and power=2 is the Euclidean
##   distance, which are the most commonly used.
##   values = [1 .. inf]

Retrieve.Distance.r = 2
```

```
##   In MINKOWSKY, EUCLIDEAN, MANHATTAN is possible to configure if
##   to use the weights previously set up
##   values = {true, false}

Retrieve.Distance.useWeights = true

##   In MINKOWSKY, EUCLIDEAN, MANHATTAN sets whether the attributes
##   must be normalized or not
##   values = {true, false}

Retrieve.Distance.normalize = true

##   In L'EIXAMPLE alpha is a threshold that defines the boundary for
##   using quantitative or qualitative values for continuous
##   attributes
##   values =  default = 8.f

Retrieve.Distance.alpha = 8.f


######################### Reuse(Adaptation) ############################

##   Adaptation phase can be performanced by different methods. By now
##   just is possible to define as MEAN, but by the interface by
##   FORMULA, then each attribute that belongs to the solution has the
##   option to be adapted by a formula manually introduced by the user,
##   null or mean.
##   values = {NULL, MEAN, FORMULA, WEIGHTEDMEAN} ##default = MEAN

Reuse.adaptMode = MEAN

##   If adapMode = WEIGHTEDMEAN, the weighting can be done by the
##   distance results or by an attribute with the specified index

Reuse.adaptByDistance = false
Reuse.adaptByAttribute = -1

######################### Evaluation(Revise) ###########################

##   The attribute Evaluation is optional, therefore this phase can
##   be skipped.
##
##   When is defined the case structure, one continuous attribute can
##   be mark as evaluation, or create a new one if this
##   values = {true, false}

Evaluation.createEvaluation = false

##   Default Value, is prepare to be an percentage of how good is it
##   values = [0..100] ##default = 100 (all is well-done !!)
```

```
Evaluation.maxValue = 100
Evaluation.minValue = 0
Evaluation.evaluationDefaultValue = 100


##   One option to make this process automatic, is to supose that
##   a case come from the library, then is possible to compare the
##   solution, and evaluate if has been well-done
##   Or, a function than provides an average of the evaluations
##   from the cases that has been used to adapt this new one

Evaluation.compareSolutions = true

######################### Retain(Learn) ################################

##   This phase is encharge to store the case that has been generate
##   in the CBR cycle or to reject it.
##   values = {LEARN, NOTLEARN, CONDITION}

Retain.learnMode = LEARN

##   if learnMode = CONDITION then has to exist a file called
##   conditions.xml that contains the conditions that have to
##   satisfy the new case to be store.
##   The xml file describes all the conditions and indicates for each
##   one what are the java class that implements that have to be
##   launched and this class implements java Condition interaface.

Retain.conditionsFile = /gesconda/resources/xml/conditions.xml


######################### Utility ######################################

##   To force to create a utility attribute
##   values = {true, false}

Utility.createUtility = false

## utilityMode= {FREQ, CUSTOM} if is custom won't be updated

Utility.utilityMode = FREQ

## if is a FREQ , Utility = numerator/denominator

## values = {NUM_TOTAL_RETRIEVES, ## number of retrieves in the system
##          NUM_RETRIEVES, ## Number of times that a case has been
##                           retrieved
##          NUM_SELECTED, ## number of times that a case has been
##                           selected to adapt another case
```

```
##          NUM_OK ## number of times that has been used to create the
##                    solution of other case, and this last has been
##                    evaluated positive

Utility.Freq.numerator = NUM_SELECTED
Utility.Freq.denominator = NUM_TOTAL_RETRIEVES

## if utility  exists and is FREQ mode , this attribute has to
## be created to updated the utility when evaluation is involved

Utility.createSourceCase = false;

######################### Battery #######################################

##   This function will be executed the CBR cycles as many times as
##   many new cases there are. The options or parameters to use are
##   defined above. The unique to define is the source of the cases
##   Those might be from the existing case base or load in a new file.
##   If there're from the library, there must specify if those has to
##   be deleted from the case base or leave there.
##
## source determines whether is from library or file
## values = {FIRST, RANDOM, FILE}

Battery.source = RANDOM

## if the source is the library has to specify the % of the cases to
## use, numberOfCases = "size of the case base * percent / 100 "
## if FIRST => the numberOfCases first ones cases
## else if RANDOM => numberOfCases cases ara selected randomly
## value = [0..100]

Battery.percent = 10

## if is from library, the new cases can be erased from the library
## values = {true, false}

Battery.deleteFromLibrary = false

## if source == FILE then the file name . Remember that the cases
## must be structured like the existing in the case base##

Battery.CasesSourceName = "resources/newcases.xml"
```