"Gheorghe Asachi" Technical University of Iași, Romania
Faculty of Automatic Control and Computer Engineering
Major: Systems Engineering
Specialization: Systems and Control

Master's Thesis

A Case-Based Reasoning Solution for Personalization of Cooking Recipes

Advisor
Assoc. Prof. Letiția Mirea

Author
Alexandru Cohal

Iași
June 2017

DECLARAŢIE DE ASUMARE A AUTENTICITĂŢII LUCRĂRII DE DISERTAȚIE

Subsemnatul **ALEXANDRU COHAL**, legitimat cu **CI** seria **MX** nr. **962636**, CNP **1930204226728**, autorul lucrării:

A CASE-BASED REASONING SOLUTION FOR PERSONALIZATION OF COOKING RECIPES

elaborată în vederea susţinerii examenului de finalizare a studiilor de **MASTERAT** organizat de către Facultatea de Automatică şi Calculatoare din cadrul Universităţii Tehnice „Gheorghe Asachi" din Iaşi, sesiunea **IUNIE** a anului universitar **2016-2017**, luând în considerare conţinutul Art. 34 din Codul de etică universitară al Universităţii Tehnice „Gheorghe Asachi" din Iaşi (Manualul Procedurilor, UTI.POM.02 - Funcţionarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române (legea 8/1996) şi a convenţiilor internaţionale privind drepturile de autor.

Data Semnătura

Table of Contents

Abstract

The aim of this thesis is to present a solution for the one of the challenges proposed within the *Computer Cooking Contest* of the *International Conference on Case-Based Reasoning* held in the year 2017. The challenge chosen is entitled '*The salad challenge*' and implies to suggest a salad recipe based on lists of desired and unwanted ingredients selected from a limited set. Moreover, the quantities of the used ingredients have to be managed in order to produce a balanced and tasty recipe. This recipe suggestion has to be made based on the provided base of 68 salad recipes from *WikiTaaable* and on the *WikiTaaable* food ontology.

The proposed solution is based on a Case-Based Reasoning approach, which represents an important field of Artificial Intelligence. The four stages of a classic Cased-Based Reasoning solution are used (Retrieval, Reusage, Revision and Retaining) in order to produce the best results both in present and in future. Details about the developed solution and its implementation in `Python` programming language are given in this thesis together with results and discussions about their correctness and ways to improve them.

This recipe recommendation system can be used for both reducing the food waste and for contributing to the increase of computational creativity. According to various statistics, almost one third of the food produced is wasted every year, most of it due to neglecting (e.g. out of date, storage in inappropriate conditions). The consumer's behaviour can be changed in many ways, one of them being the creation of a system which can recommend salad recipes using already bought ingredients. At the same time, important steps were performed in the last years in the direction of improving the creativity of computers in domains historically associated with creative people like cooking. Thus, by improving the creativity of computers in all directions, the human behaviour will be understood, modeled and simulated better and sooner.

Section 1. Introduction

In the last years, Case-Based Reasoning (CBR) has become an important Artificial Intelligence field which aims to solve new problems by reusing similar previously solved problems and adapt them to the current case. It is more than a single algorithm or a single method, it is a paradigm which is fundamentally different from any other artificial intelligence approaches (Aamodt 1994). Moreover, CBR is based on an incremental approach, due to the fact that a new successful experience obtained after reusing and adapting an old one, is stored in the memory in order to be further used in the future.

In fact, Case-Based Reasoning methodology is based on the human behavior: reasoning by reusing (Leake 1996). The most similar past experience is searched and selected (*Retrieve stage*), its solution is then adapted to fit the current problem (*Reuse stage*) and then it is applied and evaluated (*Revise stage*), following that the newly formed problem – solution group to be stored in the memory for future further usage (*Retain stage*). This intuitive and efficient way of solving problems has its origins at the intersection of several disciplines like cognitive science, computer science, information science and engineering (Richter 2013).

The *Computer Cooking Contest* is a workshop of the *International Conference on Case-Based Reasoning*, held annually. The first edition took place in 2008 and it is a competition open for anyone who is interested in artificial intelligence topics like case-based reasoning, semantic technologies and information extraction.

The challenge of this event is to develop a software which proposes cooking recipes according to a set of given specifications (e.g. ingredients list). The motivations behind this competition can be considered as being the desires to demonstrate the creativity of the computers, to propose healthy meals necessary for a healthy diet and to improve the current situation of food waste.

Computational creativity is a subject taken into discussion more and more in the last years, being considered one of the subfields of artificial intelligence which will have a huge impact in passing the *Turing test* (Turing 1950) (a test which evaluates a machine's ability to behave indistinguishable than a human, from the intelligence point of view).

In (Boden 1998) it is stated that whereas creativity is a fundamental characteristic for the human intelligence, for artificial intelligence this is a real challenge. The reason for this is given in the same paper by the definition of a creative idea which has to be '*novel, surprising and valuable (interesting, useful, beautiful…)*'. However, the term *novel* is a relative one because it might be considered with respect to the whole history or with respect to the individual in discussion.

In (Colton and Wiggins 2012), the computational creativity is defined as '*The philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviours that unbiased observers would deem to be creative*'. However, the problem of judging whether a result obtained is a creative one or not has to be taken into account carefully.

Graeme Ritchie states that being creative implies low typicality but high quality and proposes in (Ritchie 2001) a formal definition of creative behavior of a computer program.

Thus, important steps were performed in the last years in the direction of improving the creativity of computers in domains historically associated with creative people like poetry, musical compositions, paintings and even culinary, even though doubts whether the program or the programmer is creative were addressed.

Food waste represents an important issue in the present days due to the high level of raw materials, effort and money losses and due to the hunger present in many countries around the world. According to a study done by *The Food and Agriculture Organization of the United Nations* in 2009 (Lipinski 2013), almost one third of the food produced was lost or wasted. With respect to calories, one out of four calories produced was not consumed. This high level of food waste has many negative effects, both economic and environmental: wasted investment for producers and buyers, unnecessary pollution for fabrication, water and raw materials waste, increased garbage level.

In an analysis done by the *World Resource Institute* (WRI), the biggest food losses by weight were in fruits and vegetables, roots and tubers and cereals categories, whereas by calories, the biggest were the cereals with more than 50% (*Fig. 1*). The difference in these two categories is given by the water content: fruits and vegetables contain more water than the cereals. However, fruits and vegetables contain a high level of vitamins and minerals which are required for a healthy diet. Thus, even though the levels of waste in each category are different, each of them has various benefits and their waste is equally important. Actions for reducing the loss in each of these categories has to be taken as fast as possible because the high increase of the population level will lead to the impossibility to feed all the persons.

The results of a research shown in *Fig. 2* explain the main reasons of food being wasted. The most important factors belong to the same group - the neglect of the bought food (e.g. out of date, in fridge for too long time, mouldy) and not to the mistakes made during cooking.

One of the most important solutions proposed in (Lipinski 2013) for reducing the food waste level is changing the consumer behavior. Aside from delivering useful information on various ways (e.g. newspapers, advertisements and social media) about the current situation and overcoming solutions of this problem, one useful idea could be the development of computer or mobile phone software for suggesting various usages of the food owned by a person. This is the case of the proposed software in this thesis: based on a list of available ingredients (which can be some leftovers from previous days), a salad recipe is proposed. Thus, those ingredients are prevented from being thrown away due to expiration date passing and used in a smart and healthy way for both environment and owner.

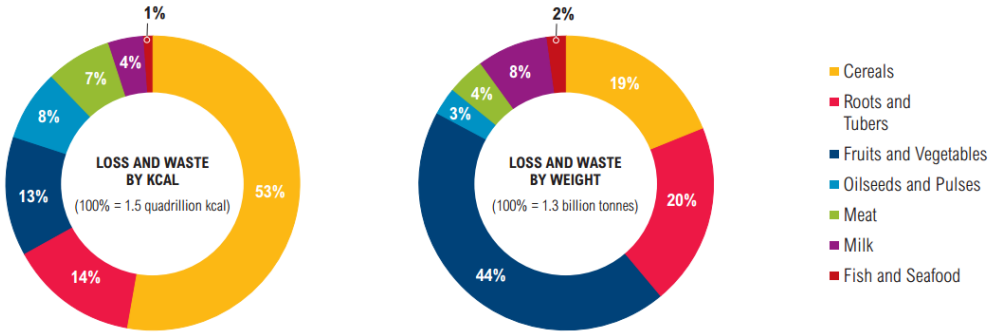


Fig. 1. Breakdown of the food loss and waste in 2009 (Lipinski 2013)

The aim of this thesis is to present a solution for the one of the challenges proposed within the *Computer Cooking Contest* of the *International Conference on Case-Based Reasoning* held in the year 2017. The challenge chosen is entitled '*The salad challenge*' and implies to suggest a salad recipe based on lists of desired and unwanted ingredients selected from a limited set. Moreover, the quantities of the used ingredients have to be managed. This suggestion has to be made based on the provided base of 68 salad recipes from *WikiTaaable* and on the *WikiTaaable* food ontology. The proposed solution is based on a Case-Based Reasoning approach, it is implemented in `Python` programming language, and it is developed in order to fulfil the requirements imposed by the participation at *Computer Cooking Contest 2017*.

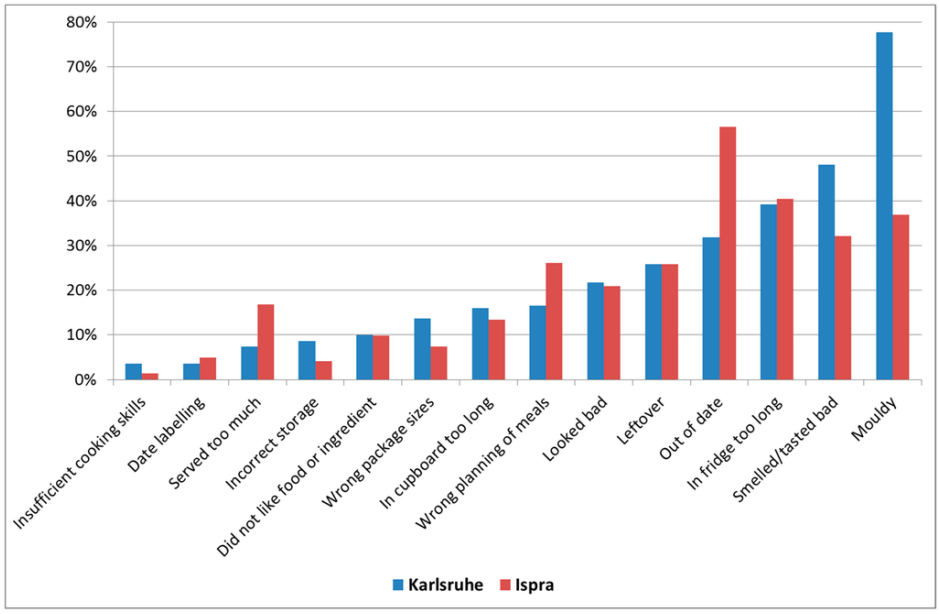


Fig. 2. The main reasons that lead to food being wasted (results of an online survey among two European research centers in Italy (JRC/Ispra) and Germany (KIT/Karlsruhe)) (Jörissen 2015)

This thesis is organized as follows: Section 2 describes the background knowledge required for solving the chosen challenge of the *Computer Cooking Contest*. In Section 3 are presented the basic knowledge of Case-Based Reasoning. The solution for the considered challenge and the implementation details are described together in Section 4. Section 5 contains the results obtained which are discussed in the same chapter. In the end, the conclusions of this thesis are shown in Section 6.

Section 2. Background knowledge

2.1. *Ontologies*

As defined in (Guarino 2009), an ontology is a formal way to model a system by defining the types, properties and interrelations of involved entities in a particular universe of discourse. These entities are divided into concepts and relations which are represented as unary and binary predicates. The hierarchy of concepts are organized using the generalization – specialization approach, thus taxonomies being obtained. According to the ideas presented in (Pănescu 2016), the purpose of an ontology is to create a common standard vocabulary of a certain topic such that future interpretations to have the same basic understanding.

One of the most used examples of ontologies is presented in (Noy 2001) which focuses on wines (*Fig. 1*). Classes are the main focus in an ontology because are used to describe concepts. In the given example, a class of wines represents all the existent wines, whereas instances of this class (e.g. `Bordeaux`) represent specific types of wines. Slots describe properties of classes and instances. In the considered example, `Chateau Lafite Rothschild Pauillac` type of wine has the slot maker which outlines its maker, `Chateau Lafite Rothschild` winery.

Four steps are described in (Noy 2001) for developing an ontology: definition of the classes, arrangement of these into a subclass – superclass hierarchy (taxonomy), definition of slots and their permitted values and in the end, instantiation of defined classes.

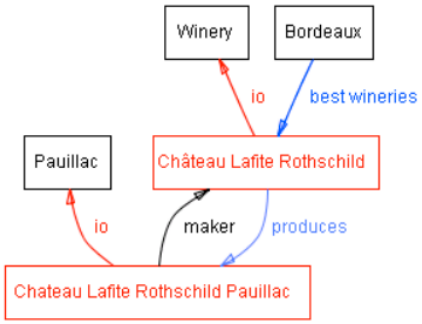


Fig. 1. A part of an ontology which models the wine domain (Noy 2001). Classes are marked with black color and instances with red. The direct links between these represent slots and internal links.

2.2. *Wikis*

2.2.1. Wiki

The concept of *wiki* was firstly introduced in 1995 by Ward Cunningham and it was latterly presented by him in (Leuf 2001). A *wiki* is defined as being a collection of interconnected Web pages which can be modified by any user, using no more than a regular Web browser. One major advantage of *wikis*, specified in (Leuf 2001), is the ability of creating a collaborative knowledge base for both academic and corporate purposes. Nowadays, the most popular *wiki* is the online encyclopedia *Wikipedia*.

The main difference between a regular Web page and a *wiki* is the opening attribute given by the possibility of any user to edit (modify, add or delete) any aspect of a *wiki* page. In addition

to this property, other ten characterizing principles of a *wiki* are defined in (Wagner 2004). Among these are the incremental (pages can contain citations to other pages which exist or do not exist already), the organic (the content and the structure of a page can evolve at any time, by being modified by any user) and the universal (a contributing user is responsible for the content and for the formatting and organization of the *wiki* page at the same time) principles.

The advantage of collective collaboration of a *wiki* comes at the same time with a downside: false or biased information can be added intentionally or not by the users. Thus, the notion of trust becomes an important element when a *wiki* is used. Due to the fact that some *wikis* evolve very rapidly (e.g. *Wikipedia* has 800 new articles per day according to their statistics), review processes were added, in which articles and contributors are rated in order to maintain a high level of informational trust.

Behind any *wiki* there is a *wiki engine* (*wiki clone*), a software which runs on a server and has a suite of databases behind. If a user wants to create its own *wiki*, one option is to configure and install a *wiki engine* on its own server. Many *wiki engines* can be used freely, as they are open source (e.g. *MediaWiki*, *PmWiki*). Another option is to use a *wiki host* (*wiki farm*) (e.g. *Seedwiki*, *PBWiki*), which runs a *wiki engine* on their servers as a service (Raman 2006) (Kille 2006).

2.2.2. MediaWiki

MediaWiki is a free open source *wiki engine* software which powers more than 2000 *wikis*, including *Wikipedia* (Barrett 2008). It allows users to create and maintain collaborative articles (*wiki pages*) under the same basic format and components. The content of these pages are written in *wikitext*, a markup language (HTML and LaTeX for example are also markup languages).

2.2.3. Semantic MediaWiki

Wiki pages contain plain text which has value only for humans. In order to be possible that the information contained to be also used by computers, machine-readable annotations have to be added.

Semantic MediaWiki is a *wiki engine* enhanced with the ability to store annotations of the content added by the users. The problems which represented the motivation of developing this extension are outlined in (Krötzsch, Vrandečić et al. 2006): content consistency (multiple pages can contain the same information), knowledge accessibility (finding the key points of a page in a faster way than reading it), and knowledge reuse (create connections between articles in order to create a broad perspective for the users). The benefits of annotating the *wikis* can be also used by processing algorithms which can solve various tasks (e.g. web crawlers browse web pages for indexing in order to improve the results returned by web engines). *Semantic MediaWiki* contributes to the *Semantic Web* trend which intends to extend the classic Web by giving explicit meanings to data.

Most of the annotations used in *Semantic MediaWiki* are based on the *ABox statements* in OWL language (Krötzsch, Vrandečić et al. 2006). *ABox statements* describe individuals by specifying their classifications, their attributes or the relations between them (e.g. Rabbit is a mammal).

2.3. Taaable and WikiTaaable

2.3.1. Taaable

Taaable is the name of a team who participated multiple times at the *Computer Cooking Contest*. More than proposing solutions to the competition's challenges, this team improved the data set provided as base by reorganizing into an ontology which can be continuously updated with new information.

2.3.2. WikiTaaable

Further on, the *Taaable* team has transferred their created ontologies into a *Semantic MediaWiki* called *WikiTaaable*, firstly introduced in (Cordier 2009). From the initial *Computer Cooking Contest*'s recipe base, six hierarchies were created and annotated: *dish types*, *dish roles*, *origins*, *diets*, *culinary actions* and *food*, describing ingredients and the process of preparing recipes.

The *food* ontology is the biggest among these, containing 2165 ingredients. The root of this ontology is the Food class. This has 19 subcategories (e.g. dairy, egg, fruit, meat), each of them having their own subcategories (e.g. dairy subcategory has 5 subcategories on its own: cheese, cream, cultured milk product, milk and nondairy topping) and so on. A complete description of the *food* ontology provided in `RDF` format is given in *Appendix 4*.

WikiTaaable also contains a large recipes base, each being composed of a list of ingredients annotated with amounts and units, preparation steps and diet specifications.

All these information were structured in a graph of semantic *wiki* pages. These pages form the knowledge based firstly used only by the *Taaable* team for their solutions, nowadays used as a default basis by all the participant at the *Computer Cooking Contest*. It is said in (Cordier 2009) that this *Semantic MediaWiki* works like a '*blackboard*' for the general users and for the Case-Based Reasoning engines due to the fact that all the trivial information can be found there.

Another feature of *WikiTaaable* is the *MediaWiki Web User Interface* which can be used by users to add new recipes by defining their ingredients, types and origins of the dishes. When a new recipe is added, a *Recipe Indexing Bot* crawls its page, extracts ingredients information and updates the recipe ontology with semantic indexing and recipe categorization.

WikiTaaable's *Semantic MediaWiki* which stores all the ingredients and recipes information can be downloaded as an `RDF` or `XML` dump from its website[1] (Blansché, Cojan et al. 2012). A dump contains the main information of a *wiki* (not including images, edit logs nor contributing user's accounts) structured in a specific format (e.g. `XML`, `RDF`).

2.4. The Semantic Web

The *Semantic Web* (named also *Web 3.0*) is an extension of the well know Web through the standards introduced by the *World Wide Web Consortium* (`W3C`) in which all the information are written in a common semantic annotated format which can be interpreted by computers (DuCharme 2011). The *Semantic Web* concept was firstly introduced in (Berners-Lee 2001) and it described the evolution from the unstructured stage of the current Web which contains only human readable articles into a *'Web of data'* where computer algorithms could understand

[1] http://wikitaaable.loria.fr/index.php/Main_Page

and manipulate data (W3C 2015). This change will also create links between information which will enhance the sharing and the reusing capabilities of the knowledge.

The architecture of the *Semantic Web* is illustrated by the *Semantic Web Stack* (*Fig. 2*) which shows how different technologies and programming languages are organized in order to make the concept of *Semantic Web* possible (Obitko 2007). This structure was firstly presented by Tim-Berners Lee, the author of *Semantic Web* concept.

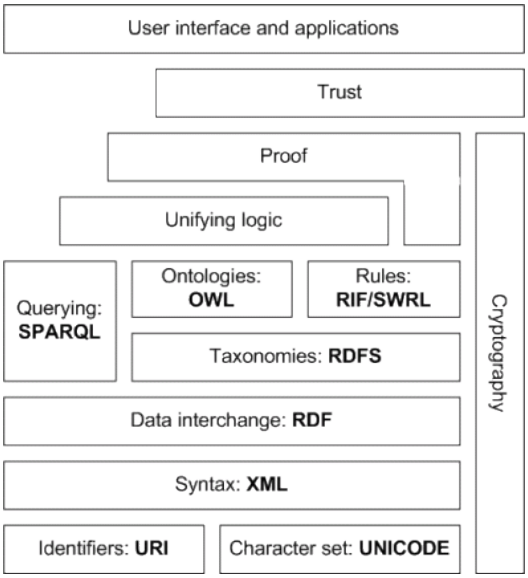


Fig. 2. The Semantic Web Stacks – the architecture of the Semantic Web (Obitko 2007)

The first layer contains the Unicode encoding standard and the Uniform Resource Identifier (URI). Unicode standard is used for encoding all the international characters which makes possible that all the human languages to be written and read on the Web. Uniform Resource Identifier (URI) is defined as a string of characters used for identifying a resource for various interactions over a network (e.g. tel:+1-862-555-539 identifies a resource by its telephone number). The most common URI is the Uniform Resource Locator (URL), also known as Web address, specifying both its primary location and the access mechanism (e.g. http://www.example.org/ specifies HTTP (Hypertext Transfer Protocol) as the access mechanism). Uniform Resource Name (URN) is another subset of URI which provides a mechanism for identifying resources in particular namespaces (e.g. urn:isbn:0-486-27557-4 is the URN for the book having the ISBN number 0-486-27557-4).

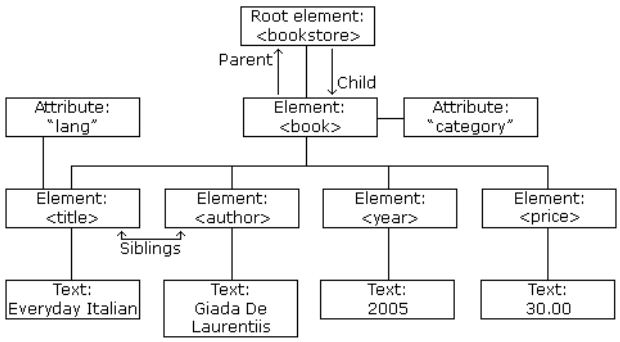
The second layer is based on the Extensible Markup Language (XML) which is a general purpose markup language intended to be used for structuring information in order to be readable by both humans and machines. Although it is similar to HTML (HyperText Markup Language), XML was designed to carry data, not to define a way of displaying it, and it does not use predefined tags. Any XML document is composed only from four basic elements: declarations, tags, elements and attributes.

- A declaration can be used at the beginning of an XML document in order to specify information like the version number and encoding scheme used (e.g. <?xml version="1.0" encoding="UTF-8"?>)
- A tag is a markup construct which begins with the character < and ends with the character > (e.g. <name> is a start tag, </name> is an end tag, and <name /> is an empty element tag).

- The composition of a start tag, an end tag and a content in between is called element (e.g. `<name> Alan Turing </name>` is an element and Alan Turing is the element's content). The content of an element can be another element, thus creating a nesting element.
- A start tag or an empty element tag can contain name-value pairs called attributes (e.g. `<name maxLength = "20">`).

The content of an XML document (the sequence of elements and their content and attributes) can be structured as a tree. This tree has a root element which is the element which contains inside all the other elements. An example of an XML document and its corresponding tree representation is showed in *Table. 1*.

Table 1. An example of an XML document and its corresponding tree structure (W3Schools)

XML document	***XML tree structure***
<?xml version="1.0" encoding="UTF-8"?> <bookstore> <book category="cooking"> <title lang="en">Everyday Italian</title> <author>Giada De Laurentiis</author> <year>2005</year> <price>30.00</price> </book> <book category="children"> <title lang="en">Harry Potter</title> <author>J K. Rowling</author> <year>2005</year> <price>29.99</price> </book> <book category="web"> <title lang="en">Learning XML</title> <author>Erik T. Ray</author> <year>2003</year> <price>39.95</price> </book> </bookstore>	

In addition of respecting the syntax rules provided in the technical specifications, an XML document has to be also valid by respecting the corresponding Document Type Definition (DTD) or XML Schema if exists. In both DTD an XML Schema documents are defined the elements and the attributes which can be used, as well as the grammar rules which have to be used (the way of using the elements and their attributes). The XML Schema document is written using the XML syntax, whereas DTD uses a separate syntax. Moreover, XML Schemas permit more detailed constraints than DTDs. An example of usage of a DTD and a XML Schema is shown in *Table 2*.

RDF (Resource Description Framework) is the primary model of representing the data from *Semantic Web*. RDF structures the information in a labeled directed graph where nodes are resources and edges are the predicates based on *subject – predicate – object* triples (e.g. Edison – invented – the bulb) (DuCharme 2011). RDF documents are written in RDF/XML language which is based on XML. An example of an RDF document is shown in *Table 3*.

RDF Schemas (RDFS) represent an extension of the RDF vocabularies in order to describe taxonomies of classes and properties. These are also extending some of the RDF elements (e.g. sets the domain and range for properties used).

Table 2. An example of an using a DTD and a XML Schema with the same XML document (W3Schools)

DTD Example	***XML Schema Example***
note.xml	**note.xml**

`<?xml version="1.0"?>`
 	
`<!DOCTYPE note SYSTEM`
`"https://www.w3schools.com/xml/note.dtd">`
 	
`<note>`
`  <to>Tove</to>`
`  <from>Jani</from>`
`  <heading>Reminder</heading>`
`  <body>Don't forget me this weekend!</body>`
`</note>`
	`<?xml version="1.0"?>`
	
	`<note`
	`xmlns="https://www.w3schools.com"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="https://www.w3schools.com`
	`note.xsd">`
	`  <to>Tove</to>`
	`  <from>Jani</from>`
	`  <heading>Reminder</heading>`
	`  <body>Don't forget me this weekend!</body>`
	`</note>`

note.dtd	**note.xsd**

`<!ELEMENT note (to, from, heading, body)>`
`<!ELEMENT to (#PCDATA)>`
`<!ELEMENT from (#PCDATA)>`
`<!ELEMENT heading (#PCDATA)>`
`<!ELEMENT body (#PCDATA)>`
	`<?xml version="1.0"?>`
	`<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema`
	`"`
	`targetNamespace="https://www.w3schools.com"`
	`xmlns="https://www.w3schools.com"`
	`elementFormDefault="qualified">`
	
	`<xs:element name="note">`
	`  <xs:complexType>`
	`  <xs:sequence>`
	`  <xs:element name="to" type="xs:string"/>`
	`  <xs:element name="from" type="xs:string"/>`
	`  <xs:element name="heading" type="xs:string"/>`
	`  <xs:element name="body" type="xs:string"/>`
	`  </xs:sequence>`
	`  </xs:complexType>`
	`</xs:element>`
	
	`</xs:schema>`

Table 3. An example of an RDF document. The subject of the statement is an object with the unique ID http://www.linkeddatatools.com/clothes#t-shirt, the predicate is 'has a property with name feature:Size', and the object is the value 12 (LinkedDataTools)

tshirt.rdf

`<rdf:RDF`
`  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"`
`  xmlns:feature="http://www.linkeddatatools.com/clothing-features#">`
 	
`  <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">`
`  <feature:size>12</feature:size>`
`  </rdf:Description>`
 	
`</rdf:RDF>`

OWL (Web Ontology Language) is a family of semantic web markup languages which are designed to represent knowledge and relationship between information in order to create ontologies which can be processed by computers. These languages are based on RDF, at which more facilities for expressing the semantics were added.

SPARQL (SPARQL Protocol and RDF Query Language) language is used to query RDF data and RDFS and OWL ontologies. It is based on RDF triples for retrieving and for receiving the answers. It is similar with the SQL language specific for data bases management (Obitko 2007). An example of a SPARQL query is shown in *Table 5*.

Table 5. An example of SPARQL query. The ontology is queried in order to find out all the country capitals in Africa (Wikipedia)

```
PREFIX ex: <http://example.com/exampleOntology#>
SELECT ?capital
       ?country
WHERE
  {
    ?x  ex:cityname      ?capital   ;
        ex:isCapitalOf   ?y         .
    ?y  ex:countryname   ?country   ;
        ex:isInContinent ex:Africa  .
  }
```

The following layers are not yet standardized and have to be implemented by the users without any rules to follow (Obitko 2007). The trust of the derived statements (*Trust layer*) can be achieved on one hand by using digital signatures for the RDF statements in order to ensure safe information transferred (*Cryptography layer*) and on the other hand by verifying the content obtained by formal proving the results obtained (*Proof layer*) which were combined together according to specified rules (*Unifying logic layer*).

2.5. Computer Cooking Contest 2017

Fig. 3. The logo of the Computer Cooking Contest

Computer Cooking Contest (CCC 2017) is a workshop of the *International Conference on Case-Based Reasoning* and it has been held annually since 2008. It is an open competition at which anyone interested in the Artificial Intelligence field can participate by solving at least one of the proposed challenges using any technology and any software tools.

The edition held in 2017 has four challenges:

- *The salad challenge*
  - A salad recipe has to be suggested based on a list with desired ingredients and a list with unwanted ingredients. The quantities used of each ingredient has to be specified, otherwise the entire original quantity will be considered.
  - The choice has to be made starting from a provided base of 68 salad recipes from *WikiTaaable* and from the *WikiTaaable* ontology.
  - The evaluation criteria for this challenge are the scientific quality of the solution and the culinary quality (public vote after tasting the recipes proposed and prepared by each participant for the same query).

- *The easy steps challenge*
  - A cold sandwich recipe has to be chosen based on a list with desired ingredients and a list with unwanted ingredients. The preparation steps should be adapted in order to minimize the effort of the recipe which has to be estimated in the end between levels 0 (easy) and 5 (hard). There is no restriction on how the effort should be computed.
  - The choice has to be made starting from a provided base of 21 salad recipes from *WikiTaaable*, from the *WikiTaaable* ontology and from an SQL database containing sandwich recipes crawled from the web.
  - The evaluation criteria for this challenge are the scientific quality of the solution and the culinary quality (public vote after evaluating the difficulty level of the recipes proposed and prepared by each participant for the same query).

- *The mixology challenge*
  - A cocktail recipe has to be chosen based on a list with desired ingredients and a list with unwanted ingredients. Based on a limited set of ingredients, the system should adapt the ingredients used.
  - The choice has to be made starting from a provided base of 109 cocktail recipes from *WikiTaaable* and from the *WikiTaaable* ontology.
  - The evaluation criteria for this challenge are the scientific quality of the solution and the culinary quality (public vote after tasting the recipes proposed and prepared by each participant for the same query).

- *The open challenge*
  - Any ideas can be proposed connected with the retrieval, the adaptation or with the creativity of cooking recipes (e.g. workflow adaptation, text adaptation, similarity computation).
  - Any resources can be used for this challenge.
  - The evaluation criteria for this challenge are the scientific quality and the originality of the given solution. For this challenge, a running system is optional.

A technical paper with a maximal length of 8 pages has to be written according to the *Springer* template in which the solution to a challenge has to be presented. If more than one challenge are solved, then all of them can be described in the same paper. For the first three challenges, a live demonstration of the running system has to be done during the paper presentation. Moreover, the running system has to be available online for the evaluation, two weeks before the conference.

The evaluation of each competition entry is based on:

- The technical paper describing the solution proposed. A jury composed of members from the conference's committee will award for each paper up to 7 points for scientific significance, up to 2 points for the presentation of the paper and up to 1 point for the results presented.
- The evaluation of the system by a jury for the *salad*, *easy steps* and *mixology* challenges. The culinary quality of each solution will be evaluated by the jury members after applying the same queries. A number of points between -2 and 2 will be awarded, where -2 means horrible, -1 not good, 0 correct, 1 good and 2 very good.
- The evaluation of the system by the public for the *salad*, *easy steps* and *mixology* challenges. For the *salad*, and *mixology* challenges, each team will have to prepare the recipes obtained as solutions after running their systems on a query chosen by the jury. Each person will vote for the best salad and for the best cocktail after tasting a small portion of the prepared

recipes. For the easy steps challenge, the audience will have to vote the difficulty level of the recipes obtained as solutions after running the systems on a query chosen by the jury.

The first three places for each available evaluation on each challenge (scientific review of the papers, jury vote and public vote) will be awarded.

For the online evaluation of the systems (two weeks before the conference) participating at the *salad*, *easy steps* or *mixology* challenges, the input of the system should have the following format:

systemURL?**d**=desired_ingredients&**u**=undesired_ingredients

where:

- **systemURL** is the URL on which the system is available and can accept and solve queries. If multiple challenges are solved by a participant, then each solution has to have a different URL
- **d** is the list of recipe's desired ingredients (if more than one ingredient is included in this list, then the vertical bar symbol '|' will be used for concatenation)
- **u** is the list of recipe's unwanted ingredients (if more than one ingredient is included in this list, then the vertical bar symbol '|' will be used for concatenation)

For example, for the system input

http://getCccCocktail.net/?d=Strawberry%20syrup|Apple%20juice&u=Mint

the system is available at the web address http://getCccCocktail.net/, and the query contains a list of two desired ingredients (strawberry syrup and apple juice) and a single unwanted ingredient (mint).

For the live demonstration of the running systems during the paper presentations (for the *salad*, *easy steps* and *mixology* challenges), the system's input can be delivered in any way (e.g. a graphical user interface can be used).

Based on the input query, the system should return only one recipe for the *salad*, *easy steps* and *mixology* challenges, which satisfies the given restrictions and it is considered to be the best fit.

For the online evaluation, the recipe which represents the solution has to be delivered under the XML format provided in the XML schema CccSystemOutput.xsd (*Appendix 1*) which includes the XML schema of a Computer Cooking Contest recipe CccRecipe.xsd (*Appendix 2*). An example of a valid recipe is presented and discussed in *Appendix 3*.

For the live demonstration of the running systems during the paper presentations (for the *salad*, *easy steps* and *mixology* challenges), the system's output can be delivered in any way (e.g. a graphical user interface can be used).

In this thesis, a solution based on a Case-Based Reasoning approach for the *Salad challenge* which respects all the previously mentioned rules is presented.

2.5.1. The salad challenge

The task of the *Computer Cooking Contest's Salad challenge* is to suggest a salad recipe based on a list with desired ingredients and a list with unwanted ingredients. For each ingredient, the original quantities will be used unless specified otherwise. Starting from a selected base salad recipes, one of them has to be chosen according to the best similarity level (which can be computed using any method) and adapted in order to satisfy the given requirements.

The selected salad recipes base contains 68 recipes taken from *WikiTaaable* saved in the XML file `ccc_salad.xml`[2]. The tree structure of this XML document is shown in *Fig. 4*. `<recipes>` element is the root element which has 68 `<recipe>` elements as children. Each `<recipe>` element has five children elements: `<recipeid>` (the identification number of the current recipe), `<title>` (the name of the recipe), `<ingredients>` (ingredients list), `<preparation>` (preparation steps) and `<diet>` (diet specifications). `<ingredients>` element has multiple `<ingredient>` children, one for each ingredient used in the current recipe. Each `<ingredient>` element has 4 attributes: `"ingredient"`, `"quantity"`, `"unit"` and `"qualifiers"` and contains inside a text describing that ingredient. `<preparation>` element has multiple `<step>` elements, each corresponding to a preparation step. The description of each preparation step is given as text of the `<step>` element. The `<diet>` element can contain multiple `<exclude-for-diet>` elements, each corresponding to a type of diet for which the current recipe is not recommended. The diet name is given as text inside the `<exclude-for-diet>` element. One of these salad recipes is shown as an example in *Fig. 5*.

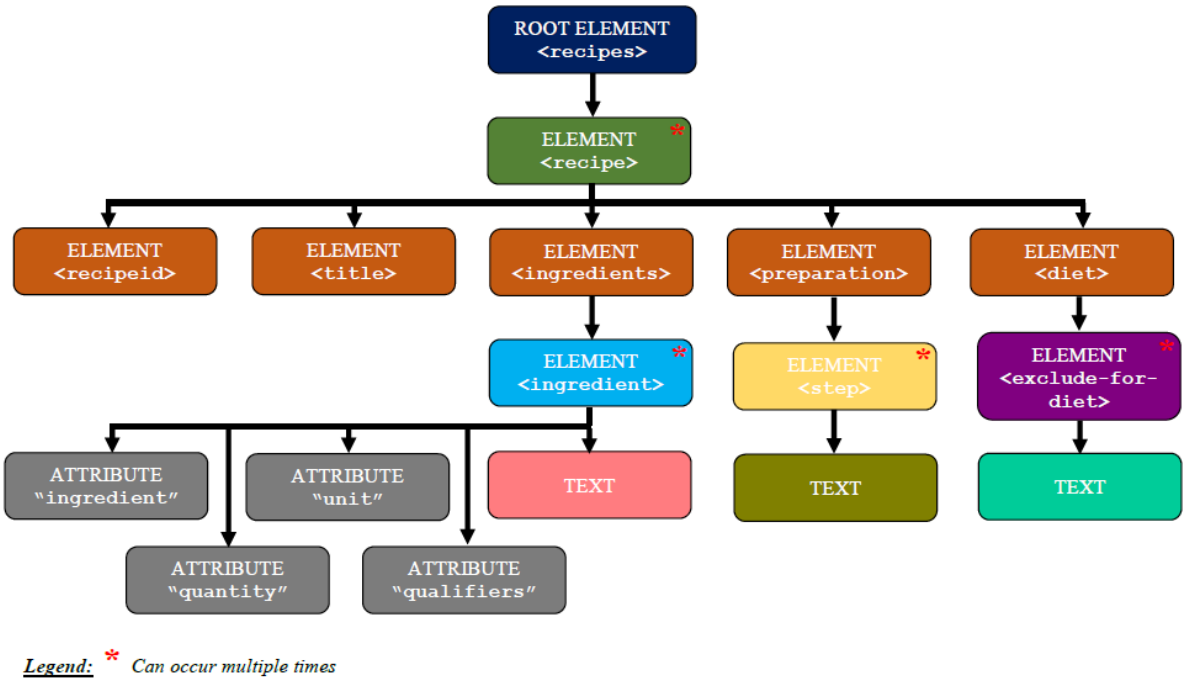


Fig. 4. The tree structure of the XML document `ccc_salad.xml` *which contains the provided salad recipes taken from WikiTaaable*

The evaluation criteria for this challenge are the scientific quality of the solution and the culinary quality. The jury selects a query which is given to all the participants. Based on the recipes suggested, each team has to prepare small portions for the public. After tasting, the public votes the level of the culinary quality. The same criteria is evaluated also by the jury.

The input and the output of a system participating at the *Salad challenge* has to fulfill the general specifications presented previously.

[2] The `ccc_salad.xml` file can be viewed and downloaded from the following address: https://github.com/nanajjar/Computer-Cooking-Contest/blob/master/ccc_salad.xml

```
    <recipe>
        <recipeid>66</recipeid>
        <title>Watergate salad</title>
        <ingredients>
            <ingredient ingredient="walnut" quantity="0.5" unit="c"
qualifiers="chopped">1/2 c Chopped walnuts</ingredient>
            <ingredient ingredient="cool_whip" quantity="8" unit="oz"
qualifiers="thawed">8 oz Cool Whip, thawed</ingredient>
            <ingredient ingredient="pudding_mix" quantity="1" unit="pk"
qualifiers="nstant">1 pk (4 oz) pistachio Instant Pudding Mix</ingredient>
            <ingredient ingredient="pineapple" quantity="1" unit="can"
qualifiers="crushed">1 cn (20 oz) crushed pineapple, Do Not Drain</ingredient>
            <ingredient ingredient="marshmallow" quantity="1.5" unit="c"
qualifiers="miniature">1 1/2 c Mini marshmallows</ingredient>
            <ingredient ingredient="maraschino_cherries" quantity="" unit=""
qualifiers="or garnish">Maraschino cherries for garnish, (optional)</ingredient>
        </ingredients>
        <preparation>
            <step>Blend dry pudding mix into Cool Whip and mix well</step>
            <step>Add crushed pineapple, marshmallows and walnuts</step>
            <step>Refrigerate about 2 hours</step>
        </preparation>
        <diet>
            <exlcude-for-diet>Cholesterol diet</exlcude-for-diet>
            <exlcude-for-diet>Nut free</exlcude-for-diet>
            <exlcude-for-diet>Veganism</exlcude-for-diet>
        </diet>
    </recipe>
```

Fig. 5. An example of salad recipe contained in the recipe base taken from WikiTaaable and provided as starting material for the «Salad challenge». The recipe is called «`Watergate`» salad and has the id `66`. This salad can be made using 6 ingredients (walnuts, cool whip, pudding mix, pineapple, marshmallows and maraschino cherries) and following 3 preparation steps. It is excluded for «Cholesterol», «Nut free» and «Veganism» diets.

Section 3. Case – Based Reasoning

In the last three decades, Case-Based Reasoning (CBR) has gain more interest due to the fact that more researches and practical applications are based on this field of Artificial Intelligence. This domain can be shortly described as being an *experience-based problem solving* (Bergmann 2009).

The ideas of using the human reasoning and the human memory organization models in reasoning problems, presented by Roger Schank in (Schank 1983), are considered to be the origins of Case-Based Reasoning. He introduced the notion of *Memory Organization Packets* (*MOPs*) by associating them with the way humans store information about the world in memory through certain valuable episodes encountered during the life. Moreover, these entities does not exist isolated, but there are interconnected in the same way our memories are. These *MOPs* are reused later, when a new situation is encountered: if a similar situation was met before and if it was a successful one, that experience is recollected and the same (or slightly adapted) steps are followed in order to achieve the success again.

This approach was a novel one, due to the fact that until then, reasoning was modeled as chaining together generalized rules in order to draw conclusions (Leake 1996). Thus, the primary knowledge became the stored experiences encountered previously. So, it can be said that in Case-Based Reasoning, *"reasoning is based on remembering"* (Leake 1996).

Case-Based Reasoning solves new problems by reusing and adapting similar previous met problems and their solutions, which are forming together the *cases*, stored in a *case-base*. It is more than a single algorithm or a single method, it is a paradigm which is fundamentally different from any other artificial intelligence approaches (Aamodt 1994).

This intuitive and efficient way of solving problems has its origins at the intersection of several disciplines like cognitive science, machine learning, computer science, information science and engineering. Though, among these, cognitive science had the biggest impact. From this field, the notions of *experience*, *memory* and *analogy*, which are related with human behaviour were taken and used (Richter 2006). Nowadays, some of the most challenging problems like recommender systems and data mining are studied using the Case-Based Reasoning technique (Richter 2013).

The entire success of Case-Based Reasoning lays on two general beliefs. The first one says that the world is regular: similar problems have similar solutions and thus, solutions for similar problems met before can be reused and adapted in order to solve new problems. The second one says that the problems met by a certain person or entity have the tendency to repeat. Thus, the probability that a similar problem was met before is very high (Leake 1996).

Moreover, Case-Based Reasoning is based on an incremental approach, due to the fact that a new (successful or not) experience obtained after reusing and adapting an old one, is stored in the memory in order to be further used in the future. Thus, it can be said that a case-based reasoner uses its previous experiences in order to achieve the success and minimize the failure (Leake 1996).

As explained in (Leake 1996), Case-Based Reasoning uses both successful and unsuccessful experiences in order to improve the learning process. When a solution is declared as being successful, the entire case (problem and solution) is stored in the case-based for future

direct usage, avoiding the need to apply again the same derivation steps. On the other hand, failures are also used for improvement. Two types of failures can be met:

- *Task failures* – The provided solution is unsuccessful;
- *Expectation failures* – The obtained outcome is different than the expected one. Thus, the system is not capable of anticipate the impact of the solution. This type of failure can appear even though there is also a *task failure* or not (e.g. a generated plan is successful beyond the expectations, a generated plan fails even though it was expected to be successful).

When a failure occurs it can be either repaired then stored in the case base or it can be stored as raw information for future analysis and comparisons with uncertain situations.

According to (Leake 1996), Case-Based Reasoning can be divided into:

- *Interpretive Case-Based Reasoning* – the task is to characterize a new situation or to classify it by finding the similarities and the differences with previously characterized or classified cases (e.g. medical diagnosis, American law system based on precedence);
- *Problem-solving Case-Based Reasoning* – the task is to generate the solution of a new problem by using previously applied solutions to similar problems (e.g. explanation systems). The difference between this category and the previous one is represented by the usage of old case adaptation in order to fit the new case.

Another classification of Case-Based Reasoning is given in (Bergmann 2009), where three classes are suggested:

- *Structural Case-Based Reasoning* – the information contained in cases is expressed using a common structural vocabulary (e.g. an ontology);
- *Textual Case-Based Reasoning* – the information contained in cases is given as free text, without any structure which has to be respected;
- *Conversational Case-Based Reasoning* – the information contained in cases is given as a list of questions (which can differ from one case to another) and their answers.

A hybrid type has also been used, by bringing the features of textual and conversational types to the structural one (e.g. by using information extraction techniques, textual cases can be structured to respect a specific case representation characteristic for structural CBRs).

Since its introduction, Case-Based Reasoning has represented the basis of numerous applications from various areas like diagnosis, customer service, recommender systems, knowledge management, medicine (e.g. self-healing domain), law, designing, computer games, music and image processing. Two of the most representative applications from the engineering diagnosis field are *Cassiopee*, the troubleshooter of the *CFM56-3* engines used for the *Boeing 737* airplane, and *ICARUS*, the fault diagnosis for the locomotives of *GE Transportation Systems* (Bergmann 2009). The development of the current Web into the *Semantic Web* gives the opportunity for developing Case-Based Reasoning applications based on the semantic descriptions implemented using the knowledge representation language OWL.

3.1. Case – Based Reasoning Cycle

The main stages of a Case-Based Reasoning system (*retrieve*, *reuse*, *revise* and *retain*), their inputs, outputs and relationship with the case base were graphically explained very suggestive in (Aamodt 1994) (*Fig. 1*). Due to the cyclicality idea given by the repetition of the same steps every time a new problem has to be solved and by the concept of reasoning by remembering, the four stages of a CBR system are visually composing a circle. Moreover, the

case base is placed in the middle of the circle due to the fact that the knowledge composed of previous experiences represents the core of the system.

When a new problem is given, a query is created. One or more similar cases are collected from the case based using a similarity metric (the *retrieve* stage). The selected experiences are then used in the context of the new problem. Though, in some situations, these cannot be used directly for the current problem, so an adaptation has to be performed firstly. Thus, a suggested solution is obtained (the *reuse* stage). After applying the solution, the result is verified (e.g. by a domain expert) in the real world or in a simulation and corrected if needed. The confirmed solution is obtained (the *revise* stage). Finally, the case based is updated with the new case (which contains the given query and the confirmed solution) for future usage (the *retain* stage).

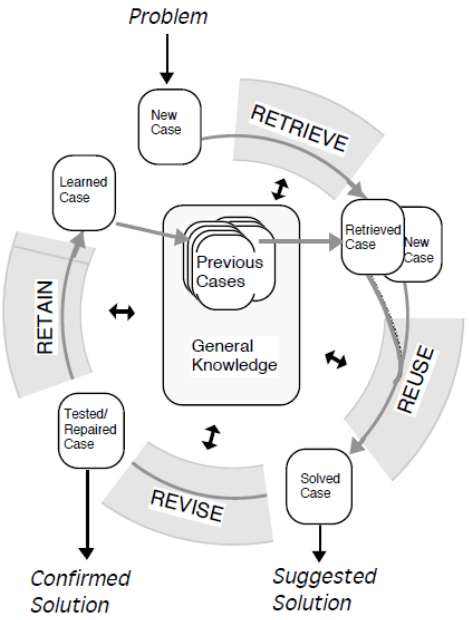


Fig.1. The four stages of a Case-Based Reasoning system (Retrieve, Reuse, Revise and Retain), their inputs, outputs and relationships (Aamodt 1994)

3.1.1. Retrieve

The retrieve phase of the Case-Based Reasoning cycle takes as inputs a query (i.e. a partial problem description) and outputs the best matching case taken from the case base. According to the task intended to be solved, only the best one or multiple best cases are retrieved (i.e. *k-nearest neighbours*). These returned cases have to be so much alike to the description given in the query such that their solution can be reused directly or with minimal adaptation. Basically, the retrieval stage should provide the answer to the question *'What case in the memory has the most suitable solution I can reuse to solve my new problem?'* (Richter 2013). This closeness relationship is called *similarity*.

In order to start the retrieval stage, a new query has to be available. In (Richter 2013) are described different ways of generating a query:

- *Automatic generation*;
- *Interviewing an user*
  - *Form-based interview* – the user receives a form in which its request is transposed in the universe of discourse defined by the used attributes from the case base;

- *Dialogue-based interview* – CBR systems which interact with the user in order to capture the problem with all the details in a conversational way;
- *Query-by-example* – instead of describing the problem, an example is provided.

The efficiency of the retrieval stage is strongly connected with both the size of the case base (more candidates have to be taken into account if the case base is larger) and with the structure of the cases. Thus, *the similarity function* used to determine the closeness between two cases (or between a query and a case) becomes very important. It should be precise enough in order to establish an order between candidates but at the same time it should not require too many computations because it will increase the execution time and the memory used. Thus, a compromise is needed.

When the closeness level between two cases is determined, two similarity types have to be taken into account (Richter 2013):

- *The similarity between the attributes of the cases* – The best way of comparing two cases is attribute by attribute. Thus, for different types of attributes, different similarity measures have to be defined according to their meaning. Each of these similarity measures is defined as a function which has as inputs the two values of the same attribute from the compared cases and one numerical output: 0 if the attributes are totally different, 1 if the values are identical or a real value between them;
- *The relative relevance of each attribute to the overall similarity meaning* – In practical cases, each attribute has a different relevance. This importance is highlighted through a number (*weight*) (can be either real or integer according to the considered structure) – large numbers show a more important attribute.

Thus, based on these two types of similarities, *the Local - Global principle* can be formulated: The similarity level between the query and a case from the case base, called *global similarity*, is the result of the combination of attribute specific similarities, called *local similarities*, by using an *amalgamation function* (Richter 2006). An amalgamation function is a function which combines together the values obtained for the similarity levels of the attributes, by considering their relevance for the overall similarity meaning. More details about the choices of these similarities functions can be read in *Section 3.4*.

Within the retrieval process, some errors may occur. As stated in (Richter 2013), these can be classified in two categories (*Fig. 2*):

- *α-error* – a case which is similar enough to the given query is not selected due to the result of the similarity function used. In this case, good candidates are eliminated, leading to a wrong result or to a greater computation effort for adapting the case to the given query;
- *β-error* – a case which is not similar enough to the given query is selected due to the result of the similarity function used. In this case, too many candidates are obtained, leading to greater computational efforts for processing all the retrieved cases.

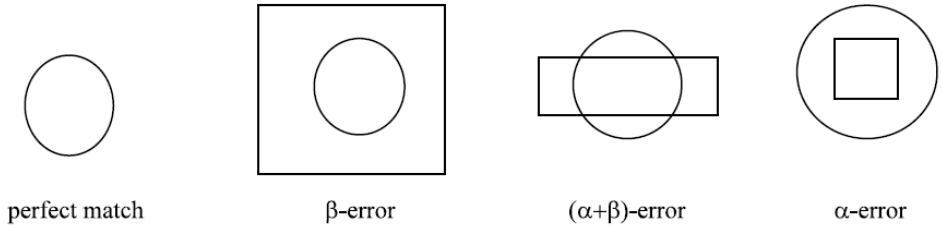


Fig. 2. Graphical representation of the possible retrieval errors (Richter 2013)

In (Richter 2013) are described some of the most used retrieval techniques:

- *Sequential retrieval* – it is the brute force method for determining the *k-nearest neighbours*. All the cases are iteratively tested by computing the entire similarity function for each. The similarity values are stored and in the end, the best *k* are chosen. It is a simple to implement method which is suited for all types of similarity measures but it implies high execution time with large case bases, no matter how the query looks like (i.e. how many restrictions are implied);
- *Two-level retrieval* – It is based on the *Many are called; Few are chosen* (*MAC/FAC*) principle composed of two steps: in the first step some candidates from the case base are selected and in the second case, the *k* nearest neighbours are searched only within these cases. The first step performs a relational retrieval because it uses a binary predicate instead of the similarity function for the pre-selection. It is not easy to define a binary predicate which excludes the cases very different to the query, without eliminating the ones which are similar to the query. The performance of this method is dependent on the chosen predicate: the efficiency is improved if the similarity function has to be applied only to a few cases but a too restrictive predicate may lead to a result affected by *α-error*;
- *Geometric retrieval* – For these methods, the cases are considered to be points in the n-dimensional real space, and the similarity is defined as a distance metric between these points and a point defined by a query.
    - *Voronoi Diagrams* – are constructed by partitioning the space into regions based on the minimal distance to the considered points. Thus, around each point are defined cells (*Voronoi regions*) determined by multiple edges (*Voronoi edges*) in which that point is the nearest point around (*Fig. 3*). These *Voronoi edges* and *regions* are constructed in a pre-processing stage. The closest neighbour to the point assigned to the given query is the point which defines the *Voronoi region* in which is it placed. For finding more closest neighbours (*k nearest neighbours*), the adjacent regions of the one in which the query is placed have to be analysed;

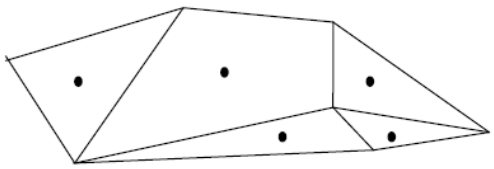


Fig. 3. An example of a 2D Voronoi diagram (Richter 2013)

    - *Geometric Approximation* – Various geometric shapes (e.g. circle, squares, rectangles, rhombus) can be used to define a border of those cases which are considered to be similar enough to a given query (i.e. the most similar cases are inside the shape, the less similar cases are outside). According to the similarity level desired to be retrieved, the size of the approximation shape can be lowered (if a higher similarity is desired) or increased (if more nearest neighbours are desired) (*Fig. 4*);
- *Index-based retrieval* – For the attributes of each case from the case base are generated indexes in a pre-processing stage which will be used later, to guide the search for the most similar cases. Some attributes are indexed and some are not. Those who are, will be used for retrieval. Those who are not, will not be used for retrieval but can be used for the following steps (e.g. adaptation).
    - *k-d trees* (*k-dimensional trees*) – The case base is decomposed iteratively into smaller groups and arranged in a binary tree. The *root* of the tree represents the entire case base, each *inner node* splits the case base with respect to an attribute value from the *k* available

and a *leaf node* represents a group of cases which are very similar and should not be partitioned further. In order to obtain the best efficiency (i.e. fast retrieval assured by short paths from the root to the leaves), a balanced tree should be created and those cases which are very different should be separated as close to the root node as possible. An example of a *k-d tree* partioning is shown in *Fig. 5*. Even though the indexing in this case is computational intensive and some problems may arise when some cases do not have all the attribute's values specified, this method leads to a faster retrieval than the other methods;

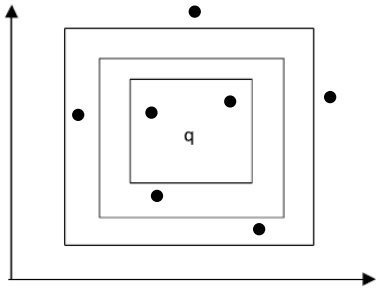


Fig. 4. The geometric approximation using rectangles (points represent cases from the case base, q is the given query, the rectangles represent possible approximations). If a higher similarity level is desired then the size of the rectangle should be decreased. If more nearest neighbors to the given query (q) are desired then the size of the rectangle should be increased (Richter 2013)

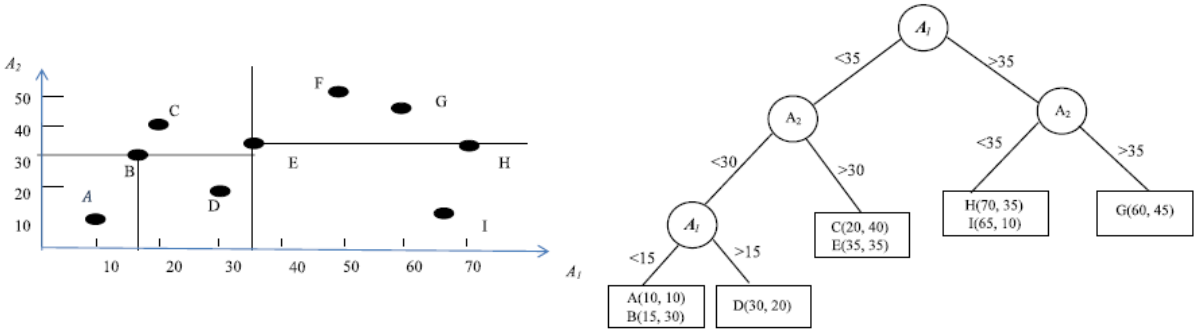


Fig. 5. The kd-Tree partitioning (right) of the case based composed of multiple 2D points characterized by the real-valued attributes A_1 and A_2 (left) (Richter 2013)

- *Retrieval network* – In a pre-processing stage, the information contained in cases is separated into information entities (e.g. attribute-value pairs) which defines the nodes of the network. Each case from the case base has its own node in the network. According to the similarity level between these entities, weighted connections are established between their nodes. In a retrieval request, the information entities from the query are activated in the network, triggering further on the connections between their corresponding nodes. In the end, the case nodes are activated with different values representing the similarity to the given query (Aamodt 2017). An example is shown in *Fig. 6.*

In conclusion, the retrieval stage has an important impact on the entire CBR cycle because it has to select from the case base one or more cases which are close enough to the given query based on similarity measures, such that their solutions to be useful enough for adaptation and reuse in order to solve the problem.

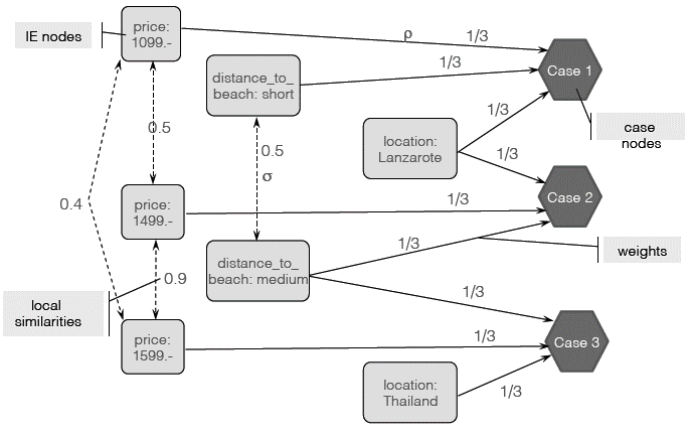


Fig. 6. An example of a retrieval network. The considered case base contains information about houses (price, location and distance to beach) (Aamodt 2017)

3.1.2. Reuse

After retrieving one or more similar cases to the current problem, their solutions are reused in order to solve the given task. If the new problem is exactly like the one met before then the reuse step is straight forward – apply the old solution directly. Though, this situation is not so often. In most cases, the differences between the previous problems and the current one are too big. In this case, an adaptation process of the old solution in order to fit the new one has to be performed (*Fig. 7*). The adaptation can be performed by a human expert or automatically, by the CBR system. Further on, the second case will be considered.

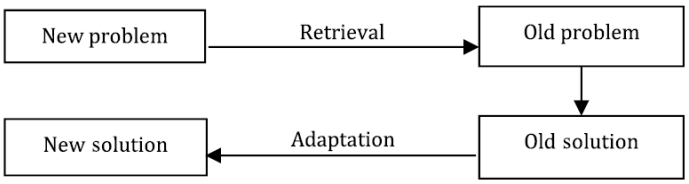


Fig. 7. The Reuse principle (Richter 2013)

The adaptations can be divided in three categories (Richter 2013):

- *Null adaptations* – for those cases which does not have to be adapted (can be reused as they are) or for those which cannot be adapted;
- *Transformation-based adaptations* – are performed by using rules and can be divided in two categories:
  - *Substitutions* – some parts of the retrieved solutions are replaced with other ones according to some previously known rules (e.g. values of attributes);
  - *Structural transformations* – the structure of the solution is changed by applying some modifications not only on the value level like in the previous case (e.g. insertion or deletion of complete attributes);
- *Generative adaptations* – it assumes that the cases contain not only the problem and its solution but also the inference steps of the solution. These steps are then replayed for the new problem and a solution is obtained. However, sometimes only by applying these steps it is not enough due to the difference between the problems. In this case, the *'gaps'* have to be solved by a generative problem solver. This solver can be activated:
  - After the complete replay of the inference steps (*one shot replay*)
  - Alternately with the inference steps (*interleaved replay*).

Even though the generative methods have to include some more information in the cases (the inference procedure), they guarantee the correctness of the solution whenever the problem solving knowledge is correct. On the other hand, the transformational methods does not ensure for all the possible cases the correctness of the obtained solution due to the fact that the embedded adaptation knowledge in the shape of rules can sometimes not cover all the situations *(Aamodt 2017)*.

3.1.3. Revise

In the revision stage of the Case-Based Reasoning cycle, a feedback about the solution obtained is created and specific measures are taken. A CBR solution may have some weaknesses or defects which lead to incorrect solutions. In this case, the response has to be corrected and the system should be improved.

As described in (Richter 2013), the revise phase can be decomposed in three steps:

- *Evaluation* – the purpose is to determine the quality of the solution by applying different tests. There are three types of evaluation methods:
  - *Evaluation by a human expert*;
  - *Evaluation in the real world* – based on the feedback from reality (e.g. sensors);
  - *Evaluation in the model* – can be performed by comparing the outputs of several example tests with their expected results (statistical evaluation);
- *Revision* – has a local effect by correcting a single case. After revision, the old case can be discarded or it can be kept for statistical evaluations. The revision can be realized in two ways:
  - *Self-repair* – the system can repair the solutions by itself based on a certain provided knowledge;
  - *User-repair* – an expert is notified that the case is not good enough and it has to be improved;
- *Learning* – has a global effect by improving the general weaknesses of several cases. Though, when a concept is learned, the problems of *over-fitting* (learning not only the useful information but also the noise and the errors) or *under-fitting* (learning without 'understanding' something – usage of a wrong model) may occur. The similarity functions may be refined by applying machine learning techniques (e.g. regression learning).

3.1.4. Retain

In the retain phase, the Case-based Reasoning system usually adds the new case to the case base. Although, storing too many cases will decrease the performance of the entire system because the retaining phase will need more time for processing. Selective retention models were developed in order to choose what case represents a group of new cases and deserves to be stored for future reference in the case base (Bergmann 2009).

3.2. Knowledge Containers

Excepting *The 4 R Model* of a Case-Based Reasoning system introduced in (Aamodt 1994) and described in *Section 3.1*, another model exists – *The Knowledge Containers Model,* introduced in (Richter 1995). This model is based on the idea that a CBR system is a knowledge-based system, a class of intelligent systems which are characterized by a knowledge base as an independent and central module (Richter 2013). In this case, the knowledge based is divided into four containers, each of them containing different information which combined together help to solve the given problem (*Fig. 8*). These containers are:

- *The Case Base Container* – contains all the previous experiences, stored as cases, which represent the main knowledge source of a CBR system. It can be seen as a particular case of a data base. The stored cases can be resulted from past situations, adapted from existing cases or created artificially. The way these cases are stored is explained in *Section 3.3*;
- *The Similarity Container* – contains those information which are needed to determine the closeness level between two cases. This measure is important for finding which cases should be retrieved from the case base in order to reuse them for solving a new problem. More details about how the similarity level can be determined are given in *Section 3.4*;
- *The Adaptation Container* – contains the knowledge used to adapt those cases which does not fit perfectly the given query in order to obtain a useful solution. Rules represent the most common type of adaptation knowledge (e.g. *If* X is not available *Then* replace it with Y);
- *The Vocabulary Container* – represents the basis for any knowledge-based system and in particular for any Case-Based Reasoning system. It contains the knowledge needed to describe and interpret the elements used in the CBR system (cases, similarity measures and adaptation knowledge): classes, relations, attributes, data types, terms, attributes and concepts.

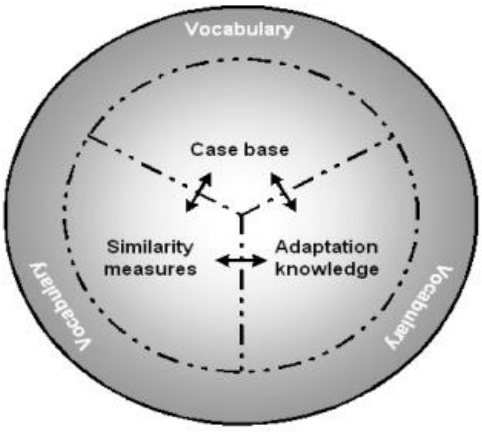


Fig. 8. The Knowledge Containers of a Case-Based Reasoning system and the interaction between them (Richter 2003)

3.3. Cases Representation and Organization

A case represents the basic element of a case base, containing knowledge about a previous experience: the problem encountered and the solution used. These cases have to be represented and organized according to the type of tasks intended to be solved, such that when a new problem in met and a similar experience is searched, the results to be obtained as accurate and as fast as possible.

Cases store the information received from humans (in most of the cases) in a formalized way in order to be processed by the Case-Based Reasoning system. This representation can be done by using: attribute-value pairs, text, images, speech or conversational representation. Depending on the task solved, the variant which has the higher benefits should be selected. Though, the attribute-value option is the most common because it is the easiest one to be understood by the humans. However, in order to be also understood by the computers, some additional information (knowledge about interpretation and usage) has to be added (Richter 2013).

The major representations for the attribute-value method of encoding the cases' information presented in (Richter 2013) are:

- *Flat attribute-value representation* – each case is represented as an attribute-value list. The selection of the attributes used for describing the cases have an important impact on the functionality of the CBR system. These have to describe independent features and have to ensure the completeness and the minimality of the description (should include only the relevant attributes). Moreover, in order to enhance the adaptation results, each feature of the solution which is relevant for the adaptation should be represented as an individual attribute. Also, each attribute has to have a type associated. The basic types are: `Integer`, `Real` (used for representing numerical information; can be both adapted and used in the similarity measurement very easy), `String` (used for less structured information; both adaptation and similarity measurement is very difficult), `Symbols` (unordered or ordered) (used for small number of fixed attribute values (e.g. `{small, big, large}`); can be both adapted and used in the similarity measurement very easy). If an attribute can accept more than one value, then a *set* is defined for it;
- *Object-oriented representation* – uses also the attribute-value representation but those attributes which are describing the same concept are aggregated together in an *object*. An *object class* describes the structure of an object (attributes and types), whereas an *object instance* assigns values to its attributes in order to describe an actual element of that class. This method has the advantage of being able to represent the relations between attributes, the most common being:
  - *Taxonomic relation* (`is-a` relation) – expresses generalizations and specializations relations (e.g. triangle `is-a` polygon) and leads to inheritance hierarchies;
  - *Compositional relation* (`is-part-of` relation) – expresses the composition of objects from sub-objects (e.g. room `is-part-of` house) and leads to decomposition hierarchies;
- *Trees and graphs* – are used in those cases where more complex relations between entities have to be modelled (e.g. molecular structures, architectural planes).

A collection of cases compose a case base which stores them by using a specific organization. The main types of organization are: flat (list), structured (e.g. hierarchies, networks) and unstructured (text, images) (Richter 2013) (*Fig. 9*).

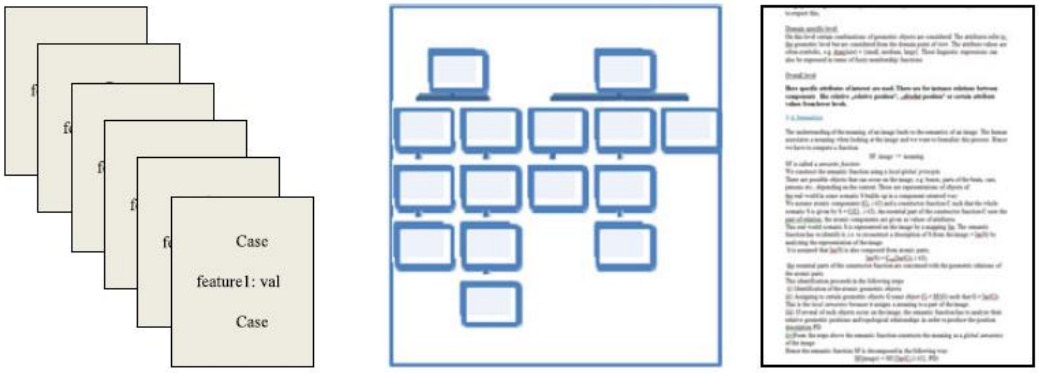


Fig. 9. Cases organization within a case base: flat (left), structured (middle) and unstructured (right) (Richter 2013)

3.4. Cases Similarity

As explained in *Section 3.1.1*, the concept of similarity plays an important role in CBR systems due to the main principle of Case-Based Reasoning – *'similar problems have similar solutions'* (Leake 1996). Thus, the closeness level between the cases from the case base and the given query is determined using this concept.

Similarity is a notion characterized by relativity (depends on the universe of discourse and on certain aspect), non-transitivity and non-symmetry (usually). Its purpose is to approximate as best as possible and as less expensive as possible the utility of usage of a certain case in solving a new problem. Thus, it can be said that the similarity function has to be chosen by considering a trade-off between the quality of the result obtained (how well approximates the utility), the modelling effort (the amount of domain knowledge used) and the computational complexity (Aamodt 2017).

Usually, similarity is formulated as a function $sim: ExE \rightarrow [0,1]$ which compares two elements from a known domain $E$and returns a real value between 0 (the elements are totally different) and 1 (the elements are identical). Thus, having a query q and two cases c_1 and c_2, the first case is preffered over the second one if $sim(q, c_1) > sim(q, c_2)$ (Richter 2006).

There is not a universal similarity measure. An appropriate type has to be selected according to the type of information contained by the cases or by the elements. These types are enumerated in (Richter 2013):

- *Counting similarities* – certain occurrences in the representation are counted. *Hamming measure*, used for comparing `Boolean` attributes by comparing the number of different bits, is part of this category;
- *Metric similarities* – used for attributes with numerical values. *Euclidean distance*, *City block metric* and *Maximum norm* are members of this category (relations *(1)*, *(2)* and *(3)* respectively);

$$sim(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}, where\ x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n), and\ x_i, y_i \in R \quad (1)$$

$$sim(x,y) = \sum_{i=1}^{n} |x_i - y_i| \,, where\ x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n), and\ x_i, y_i \in R \qquad (2)$$

$$sim(x,y) = max_{i=1}^{n} |x_i - y_i|, where\ x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n), and\ x_i, y_i \in R \qquad (3)$$

- *Transformation similarities* – it is usually used for `string` type attributes. *Levenshtein distance*, which counts how many *insert*, *delete* and *modify* operations are needed to transform a string into another one, is an example from this category;
- *Structured similarities* – used for those elements which are part of an organized group (e.g. symbols, taxonomies):
  - For the *ordered symbols* case, an integer value can be associated to each element such that the order is preserved and then the numerical values are used for comparing and adaption (e.g. small (1), medium (2), large (3));
  - For the *unordered symbols* case, a similarity table showing the similarity value between each possible combination of elements has to be defined;
  - For *taxonomies*, a similarity value can be assigned to each inner node such that the values for child nodes become higher. The similarity level between two leaf nodes is equal with the value associated to their closest common parent node. Another solution is to apply the *Wu-Palmer metric* described in (Richter 2013) which is based on the depth of the two compared concepts in the taxonomy (relation *(4)*).

$$sim_{WuPalmer}(c_1, c_2) = \frac{2 \cdot depth(DeepestCommonParrent(c_1, c_2))}{depth(c_1) + depth(c_2)} \qquad (4)$$

- *Information-oriented similarities* – the comparison is based on the information and knowledge contained in the attributes / cases (semantic not syntactic). It is used usually for texts and images.

As previously stated (*Section 3.1.1*), the similarity function can be computed using *the Local - Global principle* (the similarity level between the query and a case from the case base, called *global similarity*, is the result of the combination of attribute specific similarities, called *local similarities*, by using an *amalgamation function* (Richter 2006)). Some examples of amalgamation functions are: *weighted average*, *maximum value* and *minimum value* (relations *(5)*, *(6)* and *(7)* respectively). According to the problem intended to be solved by the CBR system, the structure of cases, the case base size and the efficiency needed, the appropriate similarity measure has to be selected.

$$F(s_1, s_2, \dots, s_n) = \sum_{i=1}^{n} w_i \cdot s_i \,, with \sum_{i=1}^{n} w_i = 1\ and\ s_i (i = 1 \dots n)\ are\ local\ similarities \qquad (5)$$

$$F(s_1, s_2, \dots, s_n) = \max\{s_1, s_2, \dots, s_n\}, where\ s_i (i = 1 \dots n)\ are\ local\ similarities \qquad (6)$$

$$F(s_1, s_2, \dots, s_n) = \min\{s_1, s_2, \dots, s_n\}, where\ s_i (i = 1 \dots n)\ are\ local\ similarities \qquad (7)$$

Section 4. Solution

4.1. *The Parser of the RDF file which contains the Food ontology (food.rdf)*

The parsing of the *Food* ontology RDF file is an operation which has to be performed only once. The required information are extracted from this file and stored under a convenient format.

The implementation of the parser can be seen in *Appendix 5* and it is explained broadly further on. These explanations are based on the structure of the *Food* ontology RDF file detailed in *Appendix 4*.

- The input of the parser is the RDF file food.rdf containing the *WikiTaaable*'s *Food* ontology;
- The output of the parser (the returned items) is represented by three lists:
  - The first one contains the name of all the classes of the ontology, given as strings (i.e. all the name of the ingredients);
  - The second one contains the generalization relationships between ontology's classes (i.e. ingredients) (e.g. the *Vegetable* class generalizes the *Onion* class with the cost of 0.2278820375), given as instances of the GeneralizationRelation class;
  - The third one contains for each ingredient a list of possible weight conversions between different measurement units (e.g. half of salmon fillet unit is similar with 159 *grams* of salmon fillet);
- The class GeneralizationRelation defines the generalization relationships and has 3 members: parent (string), child (string) and generalizationCost (float);
- The entire parser is included in the function parseRDFFoodOntology;
- The xml.etree.ElementTree Python library for parsing XML files is used;
- Firstly, the RDF file is read and it is parsed (using the function parse), resulting a tree (ElementTree instance);
- For each child of the root element, which defines a new ingredient (<owl:Class> element), the following steps are executed:
  - The value of the attribute rdf:about of the main element (owl:Class) is taken. A string is obtained which represents the URI of that ingredient. The common part of the URI (the prefix) is deleted (http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3A) in order to keep only the ingredient's name (e.g. from http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3AOnion remains only Onion). This value is added in a list which stores all the ingredient names (ingredientList);
  - The value of the attribute rdf:resource of the <rdfs:subClassOf> element is taken. A string is obtained which represents the URI of the parent of that ingredient. Similarly, the common part of the URI (the prefix) is deleted. Due to the fact that an ingredient can have multiple parents (multiple generalization relationships defined) (e.g. Onion is a sub class of both Vegetable and Flavoring classes), multiple <rdfs:subClassOf> elements can exist for the same ingredient. Thus, all parents of the current ingredient are added in a list (ingredientParents);

- The generalization cost between the current ingredient and its parent is taken from the content of the element <property:Generalisation_cost>, nested on the second level inside the <property:Has_generalisation_cost> element. Also, the name of the super class is taken from the URI given as value of the attribute rdf:Resource of the <property:Generalisation_class> element. Due to the fact that an ingredient can have multiple parents, multiple <property:Has_generalisation_cost> elements can exist for the same ingredient. Thus, all the generalization super classes and all the generalization costs are added in a dictionary (ingredientGeneralizations) (where the parent represents the *key* and the cost is the *value*);
- For each generalization relationship defined, an instance of the class GeneralizationRelation is created (containing the current ingredient, its parent (one of them) and its generalization cost), based on the list of parents ingredientParents and the generalization relationships dictionary ingredientGeneralizations. All the instances are added to a list (generalizationList);
- The weight conversion transformations are taken from the content of the element <property:Weight_conversion>. This element contains inside three children elements: <property:Original_quantity>, <property:Original_unit> and <property:Final_quantity> (the final unit is always grams). Additionaly, an extra children can exist, <property:Qualifiers> which provides a detail about the original quantity of ingredient (e.g. in the conversion of salmon from 0.5 unit, fillet to 159 grams, $fillet$ represents the qualifier for the original quantity). All these information are added in a tuple (in the following order: original quantity, original unit, original qualifier, final quantity and final unit (always set to the string g for *grams*)). Due to the fact that an ingredient can have defined multiple weight conversions, multiple <property:Weight_conversion> elements can exist. Thus, multiple weight conversion tuples are created which are added to the corresponding list of the current ingredient from the list ingredientWeightConversion.

- The lists ingredientList, generalizationList and ingredientWeightConversion are returned after all the ontology classes are processed.

Thus, a list containing the generalization relations, a list containg the name of all the ingredients and a list containing the possible weight conversions for each ingredient are created (*Table 1*) for the implementation of the next steps.

After parsing the RDF file which contains the *Food* ontology, it was observed that there are defined 2164 ingredients and 2324 generalization relationships. Also, the number of parents of an ingredient varied between 0 (the root node TopPrimitiveClass) and 4 (the ingredient Fusilli is the only one with 4 parents (Salad_pasta, Heavy_sauces_pasta, Baked_casserole_pasta and Pasta_shape)). Though, most of the ingredients have only one parent. Also, some mistakes were observed:

- 21 generalization relationship costs were not defined (e.g. the generalization relationship between Lemon and Citrus_fruit classes was specified in the <rdfs:subClassOf> element but there was no <property: Has_generalisation_cost> correspondent element);
- The super classes defined in 2 generalization relationships were not consistent with the super classes defined in the <rdfs:subClassOf> element;
- 202 generalization costs were equal with 0;
- 1 generalization cost was negative;
- 1 class was wrongly enclosed in a super class (Bell_pepper had Pepper as super class).

Table 1. Parts of the lists containing: the generalization relations between the classes of the Food ontology and their costs, the name of all the classes (ingredients) of the Food ontology, and the weight conversions for each ingredient, resulted after parsing the `food.rdf` file

Partial content of the list containing the generalization relationships between the classes of the *Food* ontology *(*ingredients*)*, stored as instances of the `GeneralizationRelation` class (for printing the content of an instance, the function `str` was overloaded)

```
...
Onion generalizes Pearl_onion with cost 0.34785522788204
Pearl_onion generalizes White_pearl_onion with cost 0.0026809651474531
Onion generalizes Red_onion with cost 0.33646112600536
Red_onion generalizes Red_onion_ring with cost 0.013404825737265
Onion generalizes Green_onion with cost 0.27479892761394
Green_onion generalizes Scallion_greens with cost 0.074396782841823
Onion generalizes White_onion with cost 0.34584450402145
Onion generalizes Yellow_onion with cost 0.34383378016086
Onion generalizes Leek with cost 0.33981233243968
Onion generalizes Sweet_onion with cost 0.34785522788204
Sweet_onion generalizes Vidalia with cost 0.0026809651474531
Onion generalizes Dry_onion with cost 0.34182305630027
Dry_onion generalizes Onion_flake with cost 0.0067024128686327
...
```

Partial content of the list containing the name of the ingredients, stored as `strings`

```
...
'Onion', 'Pearl_onion', 'White_pearl_onion', 'Red_onion', 'Red_onion_ring',
'Green_onion', 'Scallion_greens', 'White_onion', 'Yellow_onion', 'Leek',
'Sweet_onion', 'Vidalia', 'Dry_onion', 'Onion_flake', 'Onion_powder',
...
```

Partial content of the list containing the weight conversions for each ingredient

```
...
For the ingredient Salmon: [(0.5, 'unit', 'fillet', 159.0, 'g'), (3.0, 'oz',
'', 85.0, 'g')]
For the ingredient Gorgonzola: [(1.0, 'oz', '', 28.35, 'g'), (1.0, 'unit',
'', 17.0, 'g'), (1.0, 'c', 'crumbled.packed', 135.0, 'g')]
...
```

However, not all these mistakes were affecting the ingredients which could appear in the recipe of a salad. Though, it was considered better to keep the entire ontology as correct as possible, without making any assumptions regarding which ingredients are used for the considered task (the *Salad challenge*). In order to 'repair' some of these mistakes, the following actions were taken:

- 2 classes were deleted because were redundant (`Shiitake` class was deleted because `Shiitake_mushroom` class is referring to the same ingredient) or useless (`TestFood` class, sub class of `Food` class, had no specialization class defined and had no gain for the ontology);
- The generalization classes of 2 classes were changed (`Grand_marnier` and `Cointreau` classes had their generalization classes changed from `Orange_liqueur` to `Triple_sec`);
- The negative generalization cost -0.10255 between `Lemon_partof` and `Lemon` was changed to 0.00201, the same cost used for the generalization between `Gapefruit_partof` and `Grapefruit`;

- 1 super class was changed (the super class of `Bell_pepper` was changed from `Pepper` to `Fruit_Vegetable`, and its generalization cost was set the same as the one between `Sweet_pepper` and `Fruit_Vegetable`)
- 18 generalization relationship costs were defined based on the values of similar relationships (e.g. the generalization cost between `Red_bell_pepper` and `Bell_pepper` was defined the same as the one between `Sweet_red_pepper` and `Sweet_pepper`);
- The generalization costs equal with 0 were left unmodified due to the difficulty of choosing the right values.

4.2. The Parser of the XML file which contains the Salad recipes (ccc_salad.xml)

The parsing of the salad recipes `XML` file is an operation which has to be performed only once. The required information are extracted from this file and stored under a convenient format.

The implementation of the parser can be seen in *Appendix 6* and it is explained broadly further on. These explanations are based on the `XML Schema` of a recipe detailed in *Appendix 2*.

- The input of the parser is the `XML` file `ccc_salad.xml` containing the provided list of salad recipes for the *Computer Cooking Contest* (*Salad challenge*);
- The output of the parser (the returned item) is a single list which contains the recipes, stored as instances of the `Recipe` class;
- The class `Recipe` defines the content of a recipe and has 5 members: `id` (`string`), `name` (`string`), `ingredients` (list of `Ingredient` class instances), `steps` (list of `string` elements) and `excludedDiets` (list of `string` elements);
- The class `Ingredient` defines an ingredient used in a recipe and has 4 members: `name` (`string`), `quantity` (`float`), `unit` (`string`) and `qualifier` (`string`);
- The entire parser is included in the function `parseXMLSaladRecipes`;
- The `xml.etree.ElementTree Python` library for parsing `XML` files is used;
- Firstly, the `XML` file is read and it is parsed (using the function `parse`), resulting a tree (`ElementTree` instance);
- For each child (`<recipe>` element) of the root element (`<recipes>`), which defines a new recipe, the following steps are executed:
    - The content of the element `<recipeid>` is read and stored;
    - The content of the element `<title>` is read and stored;
    - For each occurrence of the element `<ingredient>` inside the element `<ingredients>`, an instance of the `Ingredient` class is created based on the values of the attributes `ingredient`, `quantity`, `unit` and `qualifiers`. All instances are then added in a list (`recipeIngredients`);
    - For each occurrence of the element `<step>` inside the element `<preparation>`, the content is read and stored as a `string` and then added in a list (`recipeSteps`);
    - For each occurrence of the element `<exclude-for-diet>` inside the element `<diet>`, the content is read and stored as a `string` and then added in a list (`recipeDiets`);

- An instance of the `Recipe` class is created with the previous acquired information (ID, title, ingredients list, steps list and diets list) and added in a list (`recipeList`).

Thus, a list containing the given salad recipes is created for the implementation of the next steps.

4.3. The Parser of the TXT file which contains the given query (query.txt)

It is considered that the query is given in a text file (`query.txt`). This file contains two lines:

- On the first line are enumerated all the desired ingredients, separated between them with single spaces. These are written in the format used in the *food* ontology file (`food.rdf`) for the values of the `about` attribute of the `<owl:Class>` element (i.e. if an ingredient name is composed of multiple words, these are separated with underscores, not with blank spaces (e.g. `White_salsify`));
- On the second line are enumerated all the undesired ingredients, separated between them with single spaces. These ingredients are also written as described above.

The parsing of the query `TXT` file is an operation which has to be performed only once. The required information are extracted from this file and stored under a convenient format.

The implementation of the parser can be seen in *Appendix 7* and it is described further on:

- The input of the parser is the `TXT` file `query.txt` containing the desired and undesired ingredients list;
- The output of the parser (the returned items) are two lists: one for the desired ingredients and the second one for the undesired ingredients;
- The entire parser is included in the function `parseTXTQuery`;
- The first line of the file is read. The ingredients are separated based on the blank space separator between them (`split` function). The `strings` resulted are added in a list (`desiredIngredients`);
- Similar for the second line. The `strings` resulted are added in another list (`undesiredIngredients`);
- These two lists are returned.

Thus, the lists containing the desired and the undesired ingredients for a salad recipes are created for the implementation of the next steps.

4.4. The solution of the 'Salad challenge'

The solution proposed in this thesis for the *Salad challenge* of the *Computer Cooking Contest* was implemented in `Python` and follows the four steps of a Case-Based Reasoning problem: *retrieval*, *reusage*, *revision* and *retaining*. This implementation is focused on solving the given problem and not on ensuring a pleasant experience for the user through a good user-interface. Though, a simple text-based interaction with the user was used. An improved user-

interface can be added as a separate layer on top of the actual implementation, without affecting the solving mechanism of the considered problem.

The main file of the solution is **`main.py`** in which the previously enumareted steps are executed:

- The retrieving stage is performed by calling the `retreveRecipe` function from `retrieveRecipe.py` file
- The reusage stage is performed by using the ingredient replacement suggetions offered by the retrieval stage and by calling the `quantityAdaptation` and `stepsAdaptation` functions from `adaptation.py` file
- The revision stage is performed by a human expert and consists only of one question with a `Yes / No` answer
- The reusage stage is performed by calling the `addToCaseBase` function from `reusage.py` file

The way of solving these steps through the given functions will be thoroughly explained further on.

4.4.1. The Retrieval stage

The retrieval stage aims to find a recipe from the recipe case-base which fits as best as possible the given query – to try to contain all the desired ingredients and to not contain any unwanted ingredients.

The given recipes are tested one by one for the fulfillment of these two conditions. When a recipe which respects these ingredient restrictions is found, the retrieval stage ends and the solution is returned.

If none of the given recipes fulfills both conditions at the same time, the second level of the retrieval stage is started. The objective is to choose a recipe which is as close as possible to the ideal one (the one which contains all the desired ingredients and which does not contain any unwanted ingredient). This recipe represents the solution of the retrieval stage due to the main principle of Case-Based Reasoning: *'similar problems have similar solution'*. Thus, the similarities between the given query and each recipe from the case-base have to be computed. Based on these values, the most similar recipe represents the solution.

The given recipes are tested one by one to find those which does not contain any of the unwanted ingredients. Between the ones which remained, the most similar one with the given query has to be selected. The similarity between a recipe from the recipe case-base and the given query is determined as following:

- The similarity between each ingredient of the recipe and each ingredient of the desired ingredients list of the query is determined (*Section 4.4.1.1*);
- For each desired ingredient, one different ingredient from the recipe's ingredients is chosen such that the sum of the similarity levels of these conversions for all the desired ingredients to be the maximum one (*Section 4.4.1.2*);
- The maximum determined sum of conversion similarities represents the general similarity level between the given query and the recipe case-base.

This way of calculating the similarity level is helpful for the adaptation phase of the reusage stage, where it will be already known which ingredients from the retrieved recipe have to be replaced in order to use all the desired ingredients.

The retrieval stage is implemented in the **`retrieveRecipe.py`** file. This file contains two functions:

- `main()` – includes the calls of the parsing functions for the query text file and salad recipes base `XML` file (`parseTXTQuery` function from `parseTXTQuery.py` file and `parseXMLSaladRecipes` from `parseXMLSaladRecipes.py` file, respectively). The returned lists (with desired ingredients, unwanted ingredients and with recipes) are stored in global variables;
- `retrieveRecipe()` – performs the actual retrieval stage, following the steps previously described. If a perfect recipe match is found (i.e. one which has all the desired ingredients and does not have any of the unwanted ingredients), then a message is printed on the screen (`'Perfect match:'` followed by the name of the recipe). Otherwise, a message containg the number of candidates remained after eiliminating those which contain unwanted ingredients is printed (`'No perfect match.'` followed by the number of candidates) This function returns three elements:
    - the recipe retrieved as a `Recipe` class intance;
    - the list with replaced ingredients;
    - the list with replacement ingredients (the ingredients from the desired ingredients list). The order of the ingredients from this lists correspons with the order of the ingredients from the replaced ingredients list (i.e. an ingredient with index j in the replaced ingredients list is replaced by the ingredient with index j in the replacement ingredients list).

4.4.1.1. The similarity level between two ingredients

The similarity level between two ingredients is determined based on the provided food ontology. This ontology cannot be interpreted as having a *Tree* structure because one ingredient can have multiple parents (e.g. `Onion` has both `Vegetable` and `Flavoring` as parents). Thus, the ontology can be transposed into an *Oriented Graph* where the nodes represent the ingredients and the arcs represent the generalization relationship (*'parent-of'* type relationships). Each arc has an associated cost which represents the generalization cost (given in the food ontology `RDF` file) (*Fig. 1*). The arcs are stored in separated lists for each starting node, based on the generalization relations list obtained after parsing the food ontology file.

The similarity level between two nodes of this graph can be calculated as being the minimum sum of the paths' costs from the nodes to a common parent node (*Fig. 1*). This idea is similar to the concept of *Lowest Common Ancestor* applicable for *Tree* structures, only that now two ingredients can have multiple common parents. In order to do this, the boundaries of the two ingredients based on the generalization relationship are increased alternatively until it is not useful anymore (until the addition of any parent node to the path leads to a sum of paths' costs bigger than the current minimum). The elements which are added to a boundary are then verified if exist in the other boundary, thus obtaining a common parent node.

The steps for applying this idea in order to determine the similarity level between two ingredients are the following ones:

- Each of the two analyzed ingredients has attached a list which represents the current boundary (CB_{list}) and a list which represent the queue of the ingredients from the boundary which have to have their parents analyzed (Q). The lists are composed of tuples with two elements, where the first element is the index of an ingredient in the ingredient name list (returned by the food ontology parser) and the second component is the cost of the path from one of the starting ingredient to the current ingredient. Initially each boundary and each queue contains only the starting ingredients;

- The *Breadth First Search* (*BFS*) traversal graph algorithm is used. The top element from the queue of an ingredient is extracted. Each parent element of this element is added to CB_{list} and to Q of the ingredient which started that path. Then it is checked to see if it is in the boundary of the other ingredient. If it is, it means it is a common parent of the two ingredients and the sum of the two paths is compared to the minimum sum obtained until now. The minimum sum is updated if it is the case. Even if a node represents a parent node of the two considered starting ingredients, the traversal will not stop when that node is reached because another minimum cost path can be obtained for a parent node which is the parent of the currently obtained parent node (i.e. if a path it has the shortest length it does not mean it has also the shortest cost);
- The top element from the queue of the other ingredient is extracted. The similar operations are done as for the other ingredient.
- The last two steps are executed alternatively until both queues become empty.

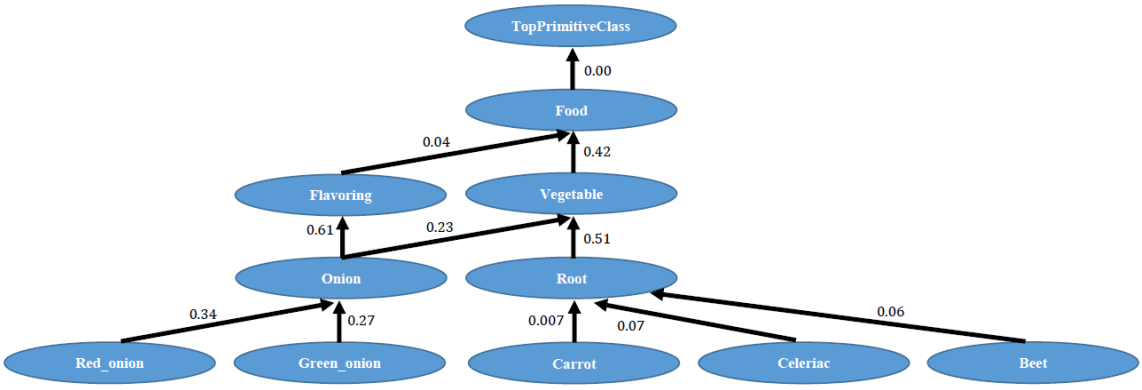


Fig. 1. A part of the representation of the food ontology as an Oriented Graph. The nodes represent the ingredients, the arcs represent the generalization relationships, and the costs represents the generalization costs. The common parent concept can be defined for this graph (e.g. `Vegetable`, `Food` *and* `TopPrimitveClass` *are common parents of* `Red_onion` *and* `Carrot`*). The similarity level between two ingredients can be calculated as the minimum sum of the paths' costs from the nodes to a common parent node (e.g. the similarity level for* `Red_onion` *and* `Carrot` *is* 1.087 *because is the sum of the paths between* `Red_onion` *and* `Vegetable` *and* `Carrot` *and* `Vegetable`*. If the similarity level is mapped in the interval [0,1] from the interval [0,2.1], the similarity value obtained is* 0.48*)*

This method of determining the similarity level between two ingredients has the advantage that it considers all the existent parent nodes, not only the shortest from the number of arcs point of view. Even though the theoretical time complexity of this solution is very high ($O(|N| + |A|)$, where $|N|$ is the number of nodes and $|A|$ is the number of arcs), in practice it is obtained a better performance. The reason for this is that only the arcs which are defining a generalization relationship are used for traversal (if we would imagine the food ontology as being a *Tree*, it would mean that the traversal can go only upwards, towards the root, not downwards, towards the leaves) and because in the graph of the longest path from the root node to a leaf node is equal with 7 (so, it can be said that the graph of the food ontology is rather *shallow* than *deep*).

Though, the disadvantage of this method is given by the used food ontology which, as stated in *Section 4.1*, it does not have defined the costs for all the generalization relationships. When such an arc is encountered, the final result will be altered because it is similar with a closer common parent, which is not true. A solution of determining the similarity level based on the depth of the common parent node (e.g. the *Wu-Palmer metric* described in *Section 3.4*)

cannot be applied because the ontology is not structured as a tree, it is structured as an oriented graph, and so, the depth cannot be defined.

The presented method of similarity calculation between two ingredients outputs a value which is not between 0 and 1, as it was explained in *Section 3* it should be. Though, when two ingredients are closer in the food ontology graph, the returned value is smaller, which is according to the similarity function rules (high similarity leads to high values). By calculating the costs of the longest paths (between leaves nodes and root node), the maximum cost of a path was determined to not be higher than 2.1. A mapping has to be done in this case from the interval [0, 2.1] to the interval [0, 1]. A simple formula is applied in order to perform this (relation *(1)*).

$$sim_{[0,1]Interval} = 1 - \frac{similarity_{[0,2.1]Interval}}{2.1} \qquad (1)$$

The similarity calculation between two ingredients is performed in two files:

- **`similarityCalculator.py`**
  - The function `similarityIngredients(ingred1, ingred 2)` receives as input two ingredients as `strings`, calls the function `lowestCostCommonAncestor()` which calculates the similarity between the two ingredients, and then converts the similarity value from the interval $[0, 2.1]$ to the interval $[0, 1]$;
- **`graphFoodOntology.py`**
  - The function `createGraphFoodOntology()` creates the oriented graph associated to the food ontology based on the generalization relations and cost. The nodes of the graph contain the index of the ingredients in the ingredients list, not the exact name of the ingredient, due to the fact that `integer` comparisons are faster than `string` comparisons;
  - The function `lowestCostCommonAncestor(indexIngred1, indexIngred 2)` calculates the similarity level between the two given ingredients using the method described previously.

4.4.1.2. The similarity level between the given query and a recipe

If a selected recipe from the recipe case-base has `N` ingredients and the desired ingredients list contains `M` ingredients, a table of size `N` by `M` can be created in which the similarities between an ingredient from the recipe and an ingredient from the desired list are stored.

Then, in order to determine the similarity level between the given query and the selected recipe, `M` substitutions have to be defined, such that the sum of the similarities of the ingredients replaced to be maximum. This is performed using a backtracking algorithm. For each desired ingredient is chosen an ingredient from the recipe ingredients such that to not have any repetition (a vector of size `N` contains the value `-1` if the corresponding recipe ingredient was used in a replacement or the index of the desired ingredient, otherwise). When a substitution was defined for each desired ingredient, the similarities sum obtained is compared with the best one achieved so far. Then, another set of substitution is generated by moving backward one step and changing the previous choice made.

Thus, the similarity level is determined and returned together with a list containing the ingredients from the recipe list which were replaced in order to achieve that similarity level.

This method of calculating the similarity level between the given query and a recipe was implemented in the file **`similarityCalculator.py`**, in the functions:

- `similarityQueryRecipe(desiredIngredientsList, recipe)` – receives as input the desired ingredients list and a recipe (instance of `Recipe` class). First it

creates the similarity table between all the desired ingredients and recipe ingredients. Then it starts the backtracking algorithm for choosing the best substitutions. In the end, it returns the similarity value obtained and the list with the replaced ingredients;

- `chooseBestReplacement(indexColumn)` – represents the backtracking algorithm for choosing the substitutions of some ingredients from the recipe's ingredients with the desired ingredients in order to obtain the maximum sum of similarities between the converted ingredients. The parameter represents the index of the desired ingredient for which a replacement has to be found at the current step.

4.4.2. The Reusage stage

After the retrieval stage, when a recipe which fully respects the given query (contains all the desired ingredients and does not contain the unwanted ingredients) or which is similar to the given query if a perfect match does not exist, is selected, it has to be reused.

In the case of a perfect match, the retrieved recipe can be used without any modifications. Otherwise, if it is not a perfect match, the retrieved recipe has to be adapted because it cannot be used as it is. For the considered problem (the *Salad challenge*), three types of adaptations are performed:

- Ingredient adaptation;
- Ingredient quantity adaptation;
- Preparation steps adaptation.

4.4.2.1. Ingredient adaptation

This adaptation is required in order to ensure that the retrieved recipe does not contain any unwanted ingredient and contains all the desired ingredients.

The unwanted ingredients are avoided through the retrieval stage in which all the recipes which contain one of these ingredients are not taken into consideration.

The desired ingredients which does not appear already in the retrieved recipe, have to replace other ingredients. The selection of the replaced ingredients is done based on the computed similarity values between ingredients: an ingredient from the desired list replaces the ingredient from the retrieved recipe which is the most similar with. Due to the fact that the similarity level between an ingredient and itself has the highest value possible (i.e. has the value 1 from the possible range of values $[0, 1]$), if a desired ingredient already appears in the retrieved recipe, it will not replace other ingredient, leading to a doubling of that ingredient. This selection is done when the similarity levels between the given query and the recipes from the case-base are computed (*Section 4.4.1*).

In order to ensure a better binding of the ingredients after all the replacements were done, some extra ingredients can be added or some extra ingredients can be replaced (e.g. if in a recipe lamb is replaced with beef, it will fit better to add bay leaves instead of rosemary (Terbach 2013)). This can be performed by using a knowledge base with recommended ingredient combinations either provided by a human expert or resulted after a learning phase using various recipes (not only the ones from the case base, some extra recipes crawled from the web for example can be used). This step was not implemented in the presented solution but in theory it provides better results not for the computational level but for the taste level.

4.4.2.2. Ingredient quantity adaptation

However, only a replacement of the ingredients name is not enough. Also the quantities have to be changed because not all the ingredients are using the same measurement unit (e.g. lemon juice can be measured in cups, whereas salmon cannot) and not all the ingredients have the same consistency such that to be able to use the same quantity for all of them (e.g. one cup of lemon juice weights $244\ grams$, whereas one cup of crumbled gorgonzola weights $135\ grams$).

In the provided food ontology, each ingredient contains also information about the conversion from some measurement units to another ones (e.g. $1\ oz$ of gorgonzola is equivalent with $28.35\ grams$ and $1\ cup$ of crumbled gorgonzola is equivalent with $135\ grams$). These conversions in most of the cases specify a transformation from the specific unit of an ingredient into grams. This general rule can be used to convert the replaced quantity into grams by performing a simple search of the used unit into the provided conversions list.

However, for some ingredients, a weight conversion list is not provided. Also, the case where the list does not contain a conversion to $grams$ can occur. For these ingredients, the quantity unit used is a unit similar to grams (e.g. oz ($ounce$) is a British mass unit). In order to convert the quantities in these cases, a measurement unit graph can be created (*Fig. 2*). In this graph, the nodes represent the measurement units and the arcs have costs associated with the proportionality factor between the connected nodes (e.g. $1\ oz$ is approximately equal with $28\ grams$, thus, the arc from the node oz to the node g will have the cost 28). The conversion from a measurement unit to $grams$ is performed by a *Breadth First Search* traversal of the graph until the $grams$ node is reached, and multiplying the costs of the path's arcs.

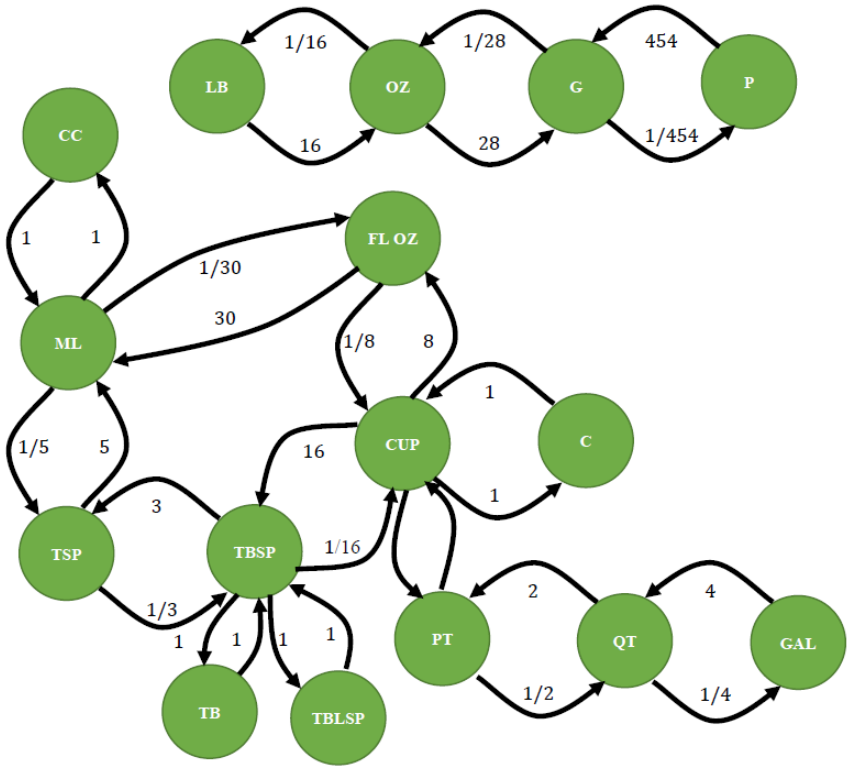


Fig. 2. The measurement units graph used in the presented solution. Each node represents a measurement unit used in cooking (not all of them are different, some of them represent the same unit but with different names) and the arcs and their costs represent the conversion and the proportionality factor, respectively, between the units

After calculating the quantity of a replaced ingredient into $grams$, the similar idea is applied in order to transform the quantity from $grams$ into a unit specific for the replacement ingredient. The same table of quantities conversions provided in the food ontology file is used but this time in the reversed direction. Due to the fact that for an ingredient can exist multiple suited quantities, the one which leads to the highest replaced value is chosen. This choice is made in order to avoid those cases when a value obtained is correct from the theoretical point of view but unusable in practice (e.g. converting $1\ gram$ of an ingredient is equivalent to $0.035\ ounces$, values which cannot be measured when cooking).

Another possibility for the choice in case of multiple possible units is to choose the one which is the closest to an integer value or to a half of integer value (because it is easier to use these values in practice). However, when a close value to an integer is searched, a very close value to 0 has to be avoided (e.g. in the previous example, $1\ gram$ of an ingredient can be converted to $0.035\ ounces$, values which is very close to the integer value 0 but it cannot be used in practice).

Moreover, a quantity adaptation could be done by considering some detailed information about the ingredients like the energy provided or the nutritional values. These values are also delivered in the food ontology file but not for all ingredients, thus making this solution harder to use.

This adaptation is performed in the files:

- **`adaptation.py`**
  - the function `quantityAdaptation (recipeRetrieved, replacedIngredRecipeRetrieved, replacementsIngredRecipeRetrieved)` performs the quantity adaptation described above for all the ingredients changed, returing two lists containing the replaced ingredient quantities and the replacement ingredient quantities (the lists are composed of tuples containing the quantity and the weight unit used). It uses the weight conversion tables given in the food ontology file and the `convertQuantity` function from `convertUnits.py` file;
- **`convertUnits.py`**
  - the function `createGraphUnits()` creates the measurement units graph (described above) based on the relations between measurement units given in the text file `units.txt` (in the format: original quantity, original unit, final quantity, final unit);
  - the function `convertQuantity(quantity1, units1, units2)` determines the quantity in `units2` corresponding to the quantity `quantity1` given in `units1` measurement unit based on the traversal of the measurement units graph method previously described.

4.4.2.3. Preparation steps adaptation

Not always two replaced ingredients have the same usage method. If the ingredients are very similar (for example, they are the children of the same parent) then the preparation steps can be kept (e.g. sweet green pepper and sweet red pepper are children of sweet_pepper). However, in most cases, this is not applicable (e.g. salmon has a different preparation way than eggs) and the preparation steps have to be adapted.

Solving this adaptation issue is the most difficult among all. This is because the adaptation has to be done now at semantic level, not syntactic as before (numeric values for quantities or symbol list for ingredients name).

The solution which was used in this thesis was to simply replace each occurrence of the replaced ingredient in the preparation step with the name of the replacement ingredient.

However, this approach does not always lead to very good results due to wrong obtained steps from the cooking point of view (e.g. if in a recipe *shrimp* is replaced with *lemon juice*, then the preparation step *'Bring the water to a boil; add the lemon juice and cook 3 to 5 minutes'* it is not the best one because it makes no sense to do this) or due to the fact that the preparation steps may contain the plural or a shorter version of the ingredient name (e.g. mayonnaise instead of light mayonnaise) which does not lead to a replacement.

This method was implemented in the function `stepsAdaptation(recipeRetrieved, replacedIngredRecipeRetrieved, replacementsIngredRecipeRetrieved)` from the **`adaptation.py file`**. In this function, all the preparation steps from the retrieved recipe are taken one by one and any occurence of the ingredients name from replaced ingredients list is replaced with its corresponding ingredient name from replacement ingredients list.

A better solution would be to have a knowledge base of adequate preparation steps for each ingredient, either from a human expert or from a learning phase. Moreover, the moment of using an ingredient in a recipe (e.g. when to add a spice) can be learned, also from a human expert or after a learning phase.

4.4.3. The Revision and Retaining stages

In the case when a perfect match of the query with a recipe from the case-base does not exist, a new recipe is obtained by adapting the most suited recipe in order to obtain a logical recipe from both theoretical and practical point of views. This new recipe can be evaluated by a human expert (e.g. a cook) either by simply analyzing it or by actually perform it and observe the results. If the opinion is that the obtained recipe is a valuable one, it can be added in the case-base for future usage. This is implemented in the presented solution by a textual question with a `Yes / No` answer. If the recipe is decided to be added, the `XML` file with salads is updated with the new member. Though, the recipe has to be added in the format provided in the `XML schema`. This is done by implementing a simple 'de-parser'.

The retaining is performed in the **`retain.py`** file:

- `createXMLNewRecipe(recipeRetrieved, id, replacedIngrName, replacementIngrName, replacementIngrQuantity, stepsAdapted)` – creates an `XML` file (`newRecipe.xml`) containing the retrieved and adapted recipe according to the given query;
- `updateCaseBase()` – adds to the recipe case-base file (`ccc_salad.xml`) the new recipe. This function is called only if the retaining is desired by answering `'Yes'` to the question `'Update the recipe case-base with the new recipe?'` asked in the main function of `main.py` file.

Moreover, the human expert can suggest some improvements of the obtained recipe in order to increase the quality of the resulted salad (e.g. adjust some quantities, add some binding ingredients).

A rating system can be also used, such that to give for each resulted recipe (either an old recipe or a new one) a mark which reflects the quality of the resulted salad. Thus, the retrieving phase can consider also this rating when selects a recipe from the case-base (e.g. when the similarity level between two recipes is close enough, the rating can be used to choose between them).

To sum up, in this section was presented a possible way of providing a recipe according to a given set of desired ingredients and a set of unwanted ingredients which is built according to the Case-based reasoning concepts (i.e. similarity concept, *The 4R Cycle* (Retrieve-Reuse-Revise-Retain)). The connections between the stages and the files of the solution are shown in *Fig. 3*. Moreover, improving ideas for each stage were described.

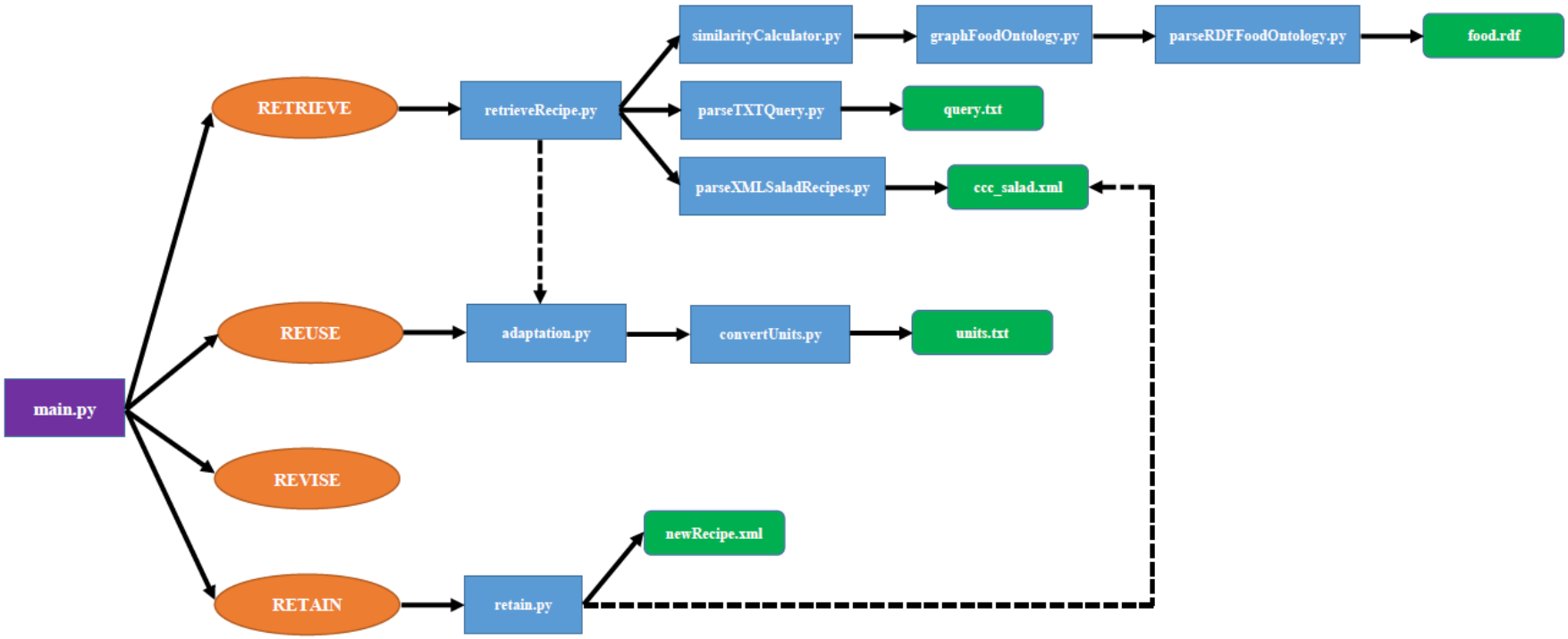


Fig. 3. The architecture of the presented solution for the 'Salad challenge'. The connections between Case-Based Reasoning stages, `Python` code files and text files are shown

Section 5. Results and Discussions

Due to the fact that the results of the salad recipes recommender based on Case-Based Reasoning can be analysed only from the qualitative point of view and not also quantitative (i.e. it is not a classic machine learning system for which a training error and a testing error can be computed and analysed), the recipes resulted have to be inspected one by one in order to verify the correctness and to find improvements. In this section, the results of three different queries will be showed and discussed. At the same time, possible ways of improving the outcome will be detailed.

5.1. Query 1: Perfect match

The query contains:

- Desired ingredients: `onion`, `shrimp`, `pea`.
- Unwanted ingredients: `oil`, `bacon`.

For this query, a perfect recipe match is found: `Macaroni-shrimp salad`. This recipe contains all the desired ingredients and does not contain any of the unwanted ones. Thus, no adaptation is required and the retrieved recipe can be reused as it is. Due to the fact that a new case (recipe) was not generated, the case-based does not have to be updated. The retrieved recipe is written without any modifications in the output `XML` file `newRecipe.xml`. The content of this file is shown below.

```
<recipe>
    <recipeid>36</recipeid>
    <title>Macaroni-shrimp salad Adapted</title>
    <ingredients>
        <ingredient ingredient="salt" qualifiers="" quantity="0.5"
unit="sp">0.5 sp  salt</ingredient>
        <ingredient ingredient="pepper" qualifiers="" quantity="0.13"
unit="sp">0.13 sp  pepper</ingredient>
        <ingredient ingredient="" qualifiers="hard-boiled,chopped"
quantity="2.0" unit="">2.0  hard-boiled,chopped </ingredient>
        <ingredient ingredient="lettuce" qualifiers="" quantity="0"
unit="">0   lettuce</ingredient>
        <ingredient ingredient="water" qualifiers="" quantity="4.5"
unit="c">4.5 c  water</ingredient>
        <ingredient ingredient="shrimp"
qualifiers="npeeled,medium,fresh" quantity="1.0" unit="lb">1.0 lb
npeeled,medium,fresh shrimp</ingredient>
        <ingredient ingredient="lbow_macaroni" qualifiers="cooked"
quantity="1.5" unit="c">1.5 c cooked lbow_macaroni</ingredient>
        <ingredient ingredient="pea" qualifiers="ozen,thawed"
quantity="1.0" unit="c">1.0 c ozen,thawed pea</ingredient>
        <ingredient ingredient="sweet_pepper"
qualifiers="medium,chopped" quantity="1.0" unit="">1.0  medium,chopped
sweet_pepper</ingredient>
        <ingredient ingredient="pimento" qualifiers="chopped"
quantity="0.5" unit="c">0.5 c chopped pimento</ingredient>
        <ingredient ingredient="onion" qualifiers="chopped"
quantity="1.0" unit="tblsp">1.0 tblsp chopped onion</ingredient>
```

```
        <ingredient ingredient="sour_cream" qualifiers=""
quantity="0.25" unit="c">0.25 c  sour_cream</ingredient>
        <ingredient ingredient="mayonnaise" qualifiers="" quantity="0.5"
unit="c">0.5 c  mayonnaise</ingredient>
    </ingredients>
    <preapration>
        <step>Bring the water to a boil; add the shrimp and cook 3 to 5
minutes</step>
        <step>Drain well and rinse with cold
water</step><step>Chill</step>
        <step>Peel and devein shrimp</step>
        <step>In a large bowl, combine the shrimp and all ingredients
except the mayonnaise, salt, and pepper</step>
        <step>Toss well</step><step>Whisk together the mayonnaise, salt,
and pepper and pour over shrimp mixture, tossing gently</step>
        <step>Chill</step><step>Spoon the salad onto lettuce leaves and
sprinkle with paprika to serve</step>
        <step>Add thin slices of avocado or wedges of tomato for a
colorful touch</step>
        <step>Do you have a basket of decorated Easter eggs sitting on
your kitchen counter? Why not put these hard-boiled beauties to good use
in one of this week's healthy recipes</step>
    </preapration>
    <diet>
        <exclude-for-diet>Cholesterol diet</exclude-for-diet>
        <exclude-for-diet>Gluten-free diet</exclude-for-diet>
        <exclude-for-diet>Veganism</exclude-for-diet>
        <exclude-for-diet>Vegetarian</exclude-for-diet>
    </diet>
</recipe>
```

In this case, the result obtained is correct: a recipe which fully matches the desires expressed in the query is obtained. Moreover, the retrieval was the best one due to the fact that the recipe which meets the desires without any adaptations was found and outputted. The quantities and the preparation steps did not have to be modified and thus, these are correct (from both theoretical and practical (cooking) point of views). The execution time for the whole program was 1.121 *seconds* on a regular computer (Intel i7 2 GHz CPU, 8 GB RAM).

5.2. Query 2: Not a perfect match but good adaptations

The query contains:

- Desired ingredients: `pineapple`, `onion`
- Unwanted ingredients: `sugar`, `oil`.

For this query, a perfect recipe match cannot be found. Thus the closest one to the given query has to be selected and modified in order to fit the two ingredient lists conditions. After removing those recipes which contain unwanted ingredients, the similarity between the others and the given query is computed. The most similar one is resulted to be: `Lighter chicken waldorf salad`. This recipe does not contain any unwanted ingredients, but `mango` and `celery` have to be added instead of `pineapple` and `onion`, respectively. These changes are feasible because the ingredients are somehow similar (e.g. a fruit is not replaced with a vegetable).

According to this ingredient adaptation, the quantities of the replacement ingredients and the preparation steps have to be adapted also.

The quantity of `pineapple` is set to `3.7 units` after the `1 unit` of `mango` used in the retrieved recipes is converted into `grams` according to the specific weight conversion specified in the food ontology (`1 unit` of `mango` is equal with `207 grams`). Then, the resulted `207 grams` are transformed to `3.7 units` of `pineapple` based on the conversion specific for `pineapple (1 unit` of `pineapple` is equal with `905 grams)`. Similarly, the conversion of `1 unit` of `celery` to `18.3 units` of `onion` is performed. These modifications in quantity are correct and the resulted values can be used in practice (`3.7 units` can be approximated as `3 units and 3 quarters` and `18.3 units` can be approximated to 18 units and a quarter).

Only one preparation step is related with the replaced ingredients `mango` and `celery`: `'Combine with the apple, celery, mango and candied ginger'`. This step is correctly adapted to `'Combine with the apple, onion, pineapple and candied ginger'` from both theoretical and cooking point of views. No problems are encountered on this adaptation phase.

Due to the fact that a new case (recipe) was generated in this case, the case-based can be updated. The adapted retrieved recipe is written in the output `XML` file `newRecipe.xml`. The content of this file is shown below. The recipe case-base stored in the `ccc_salad.xml` file will be updated only if the user wants this. The execution time for the whole program was 1.214 *seconds* (excluding the time for waiting an answer from the user and including the time needed for the case-base update).

```
<recipe>
    <recipeid>173</recipeid>
    <title>Lighter chicken waldorf salad Adapted</title>
    <ingredients>
        <ingredient ingredient="chicken_breast" qualifiers="oasted"
quantity="0.75" unit="lb">0.75 lb oasted chicken_breast</ingredient>
        <ingredient ingredient="anny_smith_apple"
qualifiers="medium,cored" quantity="1.0" unit="">1.0  medium,cored
anny_smith_apple</ingredient>
        <ingredient ingredient="onion" qualifiers="" quantity="18.3"
unit="units">18.3 units onion</ingredient>
        <ingredient ingredient="pineapple" qualifiers="" quantity="3.7"
unit="unit">3.7 unit pineapple</ingredient>
        <ingredient ingredient="crystallized_ginger" qualifiers="minced"
quantity="2.0" unit="blsp">2.0 blsp minced
crystallized_ginger</ingredient>
        <ingredient ingredient="light_mayonnaise" qualifiers=""
quantity="0.33" unit="c">0.33 c  light_mayonnaise</ingredient>
        <ingredient ingredient="fat-free_sour_cream" qualifiers=""
quantity="0.33" unit="c">0.33 c  fat-free_sour_cream</ingredient>
        <ingredient ingredient="lime_juice" qualifiers="" quantity="2.0"
unit="blsp">2.0 blsp  lime_juice</ingredient>
        <ingredient ingredient="mango_chutney" qualifiers=""
quantity="2.0" unit="blsp">2.0 blsp  mango_chutney</ingredient>
        <ingredient ingredient="ainy_mustard" qualifiers=""
quantity="1.0" unit="sp">1.0 sp  ainy_mustard</ingredient>
        <ingredient ingredient="walnut" qualifiers="coarsely chopped"
quantity="3.0" unit="blsp">3.0 blsp coarsely chopped walnut</ingredient>
```

```
        <ingredient ingredient="mint" qualifiers="minced,fresh"
quantity="2.0" unit="blsp">2.0 blsp minced,fresh mint</ingredient>
    </ingredients>
    <preapration>
        <step>1</step>
        <step>Dice the cooked chicken</step>
        <step>Combine with the apple, onion, pineapple and candied
ginger</step>
        <step>In a medium bowl, combine the mayonnaise, sour cream, lime
juice, chutney and mustard; mix well</step>
        <step>Add to the salad, mixing well</step>
        <step>Cover and refrigerate until ready to serve</step>
        <step>Just before serving, stir in the chopped walnuts and
mint</step>
        <step>Note: Waldorf salad is typically chock full o' fat; this
one has some fat, but less fat than the traditional recipe</step>
        <step>Candied or sugared ginger can be found in the Asian
section of major supermarkets</step>
```

5.3. Query 3: Not a perfect match and insufficient adaptations

The query contains:

- Desired ingredients: `salmon`, `lemon juice`, `onion`
- Unwanted ingredients: `oil`, `vanilla`.

For this query, a perfect recipe match cannot be found. Thus the closest one to the given query has to be selected and modified in order to fit the two ingredient lists conditions. After removing those recipes which contain unwanted ingredients, the similarity between the others and the given query is computed. The most similar one is resulted to be: `Crab and pea salad elegante`. This recipe does not contain any unwanted ingredients, contains `lemon juice`, but `salmon` and `onion` have to be added instead of `crab meat` and `lettuce`, respectively.

In this recipe, `onion` is already one of its ingredients, but the name contains an underscore in the beginning (`_onion`). This makes the presented solution to not recognize that it is the same ingredient, thus replacing other ingredient. Now, two ingredients are referring to `onion`, but they are spelled differently. This problem leads to a confusion of what quantity should be used in total. The solution of not encountering this problem anymore is to have a human expert who can check all the recipes' ingredients for any spelling mistakes because the computer cannot recognize when it is such a mistake and when the two ingredients are truly different.

According to this ingredient adaptation, the quantities of the replacement ingredients and the preparation steps have to be adapted also.

The quantity of `salmon` is set to `11.9 oz` after the `12 oz` of `crab meat` used in the retrieved recipes are converted into `grams` according to the standard ratio (`1 oz` is equal with `28 grams`) because the `crab meat` ingredient does not contain any specific weight conversions in the food ontology provided. Then, the resulted `336 grams` are transformed back to `oz` based on the conversion specific for `salmon`: `3 oz` corresponds to `85 grams`. Thus, `11.9 oz` are obtained for `salmon` ingredient.

Due to the fact that the recipe does not specify any quantity nor any unit for the `lettuce` ingredient (being considered probably that `lettuce` leaves are used according to the cook's

taste), the quantity resulted for the replacement ingredient onion is `0 cups` (of `chopped onion`). This issue is due to the inexact description of the given recipe. An initial verification by a human expert of all recipes for such cases and their correction would be recommended in order to avoid these situations.

The steps adaptation also have some problems in this case:

- In the steps `'Combine mayonnaise, mustard, lemon_juice, and curry powder; stir well and add to crab mixture'` and `'Spoon crab mixture into pastry shell'`, due to the fact that the structure `'crab meat'` is searched after, when only the word `'crab'` appears, it is not taken into account and remains unmodified. This can be done by splitting the ingredient name. Though, the splitting does not have to be done word by word because wrong replacements may be performed (e.g. if the ingredient `sweet red pepper` is used in a recipe, it does not mean that any time the words `sweet`, `red` or `pepper` appear in the preparation steps they are referring to the same ingredient (e.g. other types of pepper may be used)). The splitting has to be done in a way which keeps the meaning (e.g. `red sweet pepper` - `sweet pepper`, `red pepper`) by a human expert and then stored in a rule base;
- In the step `'Serve on lettuce leaves, and sprinkle with pecans, if desired'`, when the word `lettuce` is replaced with the word `onion`, the result is theoretically correct but meaningless from the cooking point of view. In this case, the whole step should be eliminated and a utilization specific to `onion` has to be found and used.

Due to the fact that a new case (recipe) was generated in this case, the case-based can be updated. The adapted retrieved recipe is written in the output `XML` file `newRecipe.xml`. The content of this file is shown below. The recipe case-base stored in the `ccc_salad.xml` file will be updated only if the user wants this. The total execution time of the program was 1.603 *seconds* (excluding the time for waiting an answer from the user and including the time needed for the case-base update).

```
<recipe>
    <recipeid>168</recipeid>
    <title>Crab and pea salad elegante Adapted</title>
    <ingredients>
        <ingredient ingredient="curry_powder" qualifiers=""
quantity="0.5" unit="sp">0.5 sp  curry_powder</ingredient>
        <ingredient ingredient="mayonnaise" qualifiers="" quantity="0.5"
unit="c">0.5 c  mayonnaise</ingredient>
        <ingredient ingredient="salmon" qualifiers="" quantity="11.9"
unit="oz">11.9 oz salmon</ingredient>
        <ingredient ingredient="pea" qualifiers="mall,drained"
quantity="15.0" unit="oz can">15.0 oz can mall,drained pea</ingredient>
        <ingredient ingredient="_onion" qualifiers="ced" quantity="0.5"
unit="c">0.5 c ced _onion</ingredient>
        <ingredient ingredient="chestnut" qualifiers="ced,drained"
quantity="8.0" unit="oz can">8.0 oz can ced,drained
chestnut</ingredient>
        <ingredient ingredient="jon_mustard" qualifiers=""
quantity="2.0" unit="sp">2.0 sp  jon_mustard</ingredient>
        <ingredient ingredient="lemon_juice" qualifiers=""
quantity="2.0" unit="sp">2.0 sp lemon_juice</ingredient>
        <ingredient ingredient="onion" qualifiers="" quantity="-0.0"
unit="c">-0.0 c onion</ingredient>
    </ingredients>
```

```
    <preapration>
        <step>Combine first 4 ingredients in a large bowl; toss</step>
        <step>Combine mayonnaise, mustard, lemon_juice, and curry
powder; stir well and add to crab mixture</step>
        <step>Stir gently to combine</step><step>Cover with plastic wrap
and chill 2 hours</step>
        <step>Serve on onion leaves, and sprinkle with pecans, if
desired</step>
        <step>Yield: 4 to 6 servings</step>
        <step>Crab and Pea Quiche: Prepare recipe as directed above,
omitting lemon_juice and adding 3 beaten eggs to mayonnaise
mixture</step>
        <step>Prepare 1 sheet Pillsbury All Ready Pie Crust according to
package directions for an unfilled one-crust pie using a 10-inch tart
pan</step>
        <step>Do not prick crust</step>
        <step>Partially bake pastry shell at 450 degrees for 9 to 11
minutes or until golden brown</step>
        <step>(If crust puffs up, gently press back to bottom and sides
fo pan with back of wooden spoon</step>
        <step>) Spoon crab mixture into pastry shell</step>
        <step>Bake at 375 degrees for 40 to 45 minutes or until
set</step>
        <step>Sprinkle with pecans, if desired</step>
    </preapration>
    <diet>
        <exclude-for-diet>Cholesterol diet</exclude-for-diet>
        <exclude-for-diet>Nut free</exclude-for-diet>
        <exclude-for-diet>Veganism</exclude-for-diet>
        <exclude-for-diet>Vegetarian</exclude-for-diet>
    </diet>
</recipe>
```

To sum up, the improvements which can be done in order to obtain better performances for the *Salad challenge* cooking recipes personalization system are:

- Verification by a human expert of all the ingredients name, quantities and units in order to not have any missing values nor spelling mistakes;
- Creation of a base of similar names for ingredients which can appear in the preparation steps (e.g. crab meat - crab);
- Creation by a human expert (or using a learning phase) of specific usages of ingredients to avoid the cases when a replacement ingredient is prepared incorrectly in the recipe or it is introduced in the wrong moment;
- The quantity adaptation can be performed by using also the energy value and the nutritional values of the ingredients involved in the replacement;
- Some binding ingredients can be added in addition to the desired ingredients in order to obtain a better taste of the final salad;
- Add the generalization costs for each relationship between two ingredients;
- All the ingredient quantities can be adapted such that to obtain only integer values, half of integer values or quarters of integer values for easiness in cooking.

Section 6. Conclusions

The Salad challenge is one of the tasks within the *Computer Cooking Contest* workshop held during the *International Conference on Case-Based Reasoning*. The aim of the challenge is to develop a system which proposes salad recipes according to a list of desired ingredients and a list of unwanted ingredients. The recommendation has to be done based on provided food ontology and salad recipes base.

The development of computational creativity and the need of finding new ways to decrease the food waste due to neglection of bought ingredients are the main reasons for developing a cooking recipe recommendation system based on lists of ingredients already owned and ingredients unwanted.

This thesis proposes a solution for this challenge. Only the core part which provides a solution for the given query was implemented and detailed. The user interface and the online interface are extra layers which do not affect the functioning of the recommendation and personalization system and can be implemented separately.

The proposed solution is based on the Case-Based Reasoning metholdology which aims to solve new problems by adapting similar previous met problems and their solutions. This approach is based on the human way of reasoning.

The main stages of a Case-Based Reasoning Systems are used and applied accordingly in order to solve the considered problem:

- *Retrieval* – selects from the case-based composed of the given salad recipes the one which matches the best the given query. The similarity between each recipe and the query is computed by calculating the similarities between desired and used ingredients. The best replacements are determined in order to introduce in the recipe the ingredients which are not already used
- *Reusage* – the selected recipe is adapted in order to be used: the desired ingredients which are not already in the recipe are introduced through replacement, their quantities is modified and the preparation steps are changed in order to fit the new ingredients
- *Revision* – the new recipe is outputted in order to be analyzed by a human expert which decides whether the result is good enough in order to be added in the case-base for future usage
- *Retaining* – if the obtained recipe is considered to be good enough, it is added in the XML file which contains all the case-base's recipes

Some of the obtained results are shown and discussed and the improvements needed are listed in order to outcome the present problems.

To sum up, Case-Based Reasoning is shown to be capable of provinding the methodology for the cooking recipes recommendation and adaptation task. The results obtained can be used in practice, even though in some points the presented system should be improved. The main solution which could lead to a very performant recipe recommendation system is the introduction of Semantic Web in the area of cooking recipes. Thus, the computer will be able much more easily and efficiently to interpret the information in order to provide correct and tasty recipes.

References

Aamodt, A. B., Kerstin (2017). TDT 4173 Machine Learning and Case-Based Reasoning Course, Norwegian University of Science and Technology.

Aamodt, A. P., Enric (1994). "Case-based reasoning: foundational issues, methodological variations, and system approaches." AI Commun. **7**(1): 39-59.

Barrett, D. J. (2008). MediaWiki, O'Reilly Media, Inc.

Bergmann, R. A., Klaus-Dieter; Minor, Mirjam; Reichle, Meike; Bach, Kerstin (2009). "Case-Based Reasoning: Introduction and Recent Developments." KI - Künstliche Intelligenz, German Journal on Artificial Intelligence - Organ des Fachbereiches "Künstliche Intelligenz" der Gesellschaft für Informatik e.V. (KI) **23**: 5-11.

Berners-Lee, T. H., James; Lassila, Ora (2001). "The Semantic Web." Scientific American: 29-37.

Blansché, A., et al. (2012). "WikiTaaable." Retrieved 7 May, 2017, from http://wikitaaable.loria.fr/index.php/Main_Page.

Boden, M. A. (1998). "Creativity and Artificial Intelligence." Artificial Intelligence **103**(1): 347-356.

CCC (2017). "Computer Cooking Contest 2017." Retrieved 4 May, 2017, from https://computercookingcontest.com/.

Colton, S. and G. A. Wiggins (2012). Computational creativity: the final frontier? Proceedings of the 20th European Conference on Artificial Intelligence. Montpellier, France, IOS Press**:** 21-26.

Cordier, A. D.-L., Valmi; Lieber, Jean; Nauer, Emmanuel; Badra, Fadi; Cojan, Julien; Gaillard, Emmanuelle; Infante-Blanco, Laura; Molli, Pascal; Napoli, Amedeo; Skaf-Molli, Hala (2014). Taaable: a Case-Based System for personalized Cooking. Successful Case-based Reasoning Applications-2. S. J. Montani, Lakhmi C. Berlin Heidelberg, Springer-Verlag**:** 121-162.

Cordier, A. L., Jean; Molli, Pascal; Nauer, Emmanuel; Skaf-Molli, Hala; Toussaint, Yannick (2009). WIKITAAABLE: A semantic wiki as a blackboard for a textual case-based reasoning system. SemWiki 2009 - 4rd Semantic Wiki Workshop at the 6th European Semantic Web Conference - ESWC 2009. Heraklion, Greece.

DuCharme, B. (2011). Learning SPARQL, O'Reilly Media, Inc.

Guarino, N. O., Daniel; Staab, Steffen (2009). What is an Ontology? Handbook on Ontologies. S. S. Staab, Rudi. Heidelberg, Springer Verlag.

Jörissen, J. P., Carmen; Bräutigam, Klaus-Rainer (2015). "Food Waste Generation at Household Level: Results of a Survey among Employees of Two European Research Centers in Italy and Germany." Sustainability **7**(3): 2695-2715.

Kille, A. (2006). "Wikis in the Workplace: How Wikis Can Help Manage Knowledge in Library Reference Services." Library and Information Science Research e-journal (LIBRES) **16**(1).

Krötzsch, M., et al. (2006). Semantic MediaWiki. The Semantic Web - ISWC 2006: 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006. Proceedings. I. Cruz, S. Decker, D. Allemang et al. Berlin, Heidelberg, Springer Berlin Heidelberg**:** 935-942.

Leake, D. B. (1996). Case-Based Reasoning: Experiences, Lessons and Future Directions. Menlo Park, MIT Press.

Leuf, B. C., Ward (2001). The Wiki way: quick collaboration on the Web, Addison-Wesley Longman Publishing Co., Inc.

LinkedDataTools. "LinkedDataTools." Retrieved 8 May, 2017, from http://www.linkeddatatools.com.

Lipinski, B. H., Craig; Waite, Richard; Searchinger, Tim; Lomax, James; Kitinoja, Lisa (2013). Reducing Food Loss and Waste. Creating a Sustainable Food Future. W. R. Institute.

Noy, N. F. M., Deborah L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology.

Obitko, M. (2007). "Semantic Web Architecture." Retrieved 8 May, 2017, from http://obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html.

Pănescu, D. A. (2016). 'Knowledge Representation and Reasoning' Course, 'Gheorghe Asachi' Technical University of Iași, Romania.

Raman, M. (2006). "Wiki Technology as A "Free" Collaborative Tool within an Organizational Setting." Information Systems Management **23**(4): 59-66.

Richter, M. M. (1995). The Knowledge Contained in Similarity Measures. Invited Talk at the First International Conference on Case-Based Reasoning, ICCBR'95, Sesimbra, Portugal.

Richter, M. M. (2003). Knowledge Containers. Readings in Case Based Reasoning, Morgan Kaufmann Publishers.

Richter, M. M. A., Agnar (2006). "Case-based reasoning foundations." The Knowledge Engineering Review **20**(3): 203-207.

Richter, M. M. W., Rosina O. (2013). Case-Based Reasoning: A Textbook. Heidelberg, Springer Publishing Company, Incorporated.

Ritchie, G. (2001). "Assessing Creativity." Proceedings of AISB Symposium on AI and Creativity in Arts and Science.

Schank, R. C. (1983). Dynamic Memory: A Theory of Reminding and Learning in Computers and People, Cambridge University Press.

Shadbolt, N., et al. (2006). "The Semantic Web Revisited." IEEE Intelligent Systems **21**(3): 96-101.

Terbach, R. (2013). Case-based Cooking: A semantic similarity based approach for retrieval and adaptation of recipes for lists of ingredients. Fakultät Wirtschaftsinformatik und Angewandte Informatik, Otto-Friedrich-Universität Bamberg. **Bachelorarbeit:** 64.

Turing, A. M. (1950). "Computing machinery and intelligence." Mind **59**(236): 433-460.

W3C (2015). "Semantic Web." Retrieved 7 May, 2017, from https://www.w3.org/standards/semanticweb/.

W3Schools. "W3Schools." Retrieved 8 May, 2017, from https://www.w3schools.com.

Wagner, C. (2004). "Wiki: A Technology for Conversational Knowledge Management and Group Collaboration." Communications of the Association for Information Systems **13**.

Wikipedia. "Wikipedia." Retrieved 8 May, 2017, from https://en.wikipedia.org/wiki/Main_Page.

Appendices

Appendix 1.

The given XML Schema which defines the format of the XML file of the recipes delivered as solution for the *Computer Cooking Contest* (CccSystemOutput.xsd)

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema">

  <include schemaLocation="CccRecipe.xsd" />

  <element name="cccAnswer">
    <complexType>
      <sequence>
        <element name="system"      minOccurs="1" maxOccurs="1" type="string" />
        <element name="query"       minOccurs="1" maxOccurs="1" type="string" />
        <element name="retrieve"    minOccurs="1" maxOccurs="1" />
        <element name="reuse"       minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>

  <element name="retrieve">
    <complexType>
      <sequence>
        <element name="recipe" minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>

  <element name="reuse">
    <complexType>
      <sequence>
        <element name="adaptation"  minOccurs="1" maxOccurs="1" type="string" />
        <element name="recipe"      minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>

</schema>
```

- The root element has to be <cccAnswer>
- The nested elements inside the root element (which have to occur exactly once) are: <system> (has to contain the URL of the system (as a string) which returned the response), <query> (has to contain the given query (as a string)), <retrieve> (has to contain the recipe retrieved from the recipes case-base), <reuse> (has to the contain the adapted version of the recipe which satisfies the given restrictions)

- Both the `<retrieve>` and the `<reuse>` elements have to contain inside a `<recipe>` element exactly once (its structure is detailed in the `XML Schema CccRecipe.xsd`)
- The `<reuse>` element has to contain in addition the `<adaptation>` element which describes the changes made in the retrieved recipe (as a `string`)

Appendix 2.

The given XML Schema which defines the format of the XML file of a recipe for the *Computer Cooking Contest* (CccRecipe.xsd)

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema">

  <element name="recipe">
    <complexType>
      <sequence>
        <element name="title"       minOccurs="1" maxOccurs="1" type="string" />
        <element name="ingredients" minOccurs="1" maxOccurs="1" />
        <element name="preparation" minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>

  <element name="ingredients">
    <complexType>
      <sequence>
        <element name="ingredient" minOccurs="1" maxOccurs="unbounded" type="string" />
      </sequence>
          <attribute name="food"        type="string"   use="required" />
  <attribute name="quantity"    type="integer" />
  <attribute name="unit"        type="string" />
    </complexType>

    </element>

  <element name="preparation">
    <complexType>
      <sequence>
        <element name="step" minOccurs="1" maxOccurs="unbounded" type="string" />
      </sequence>
    </complexType>
  </element>

</schema>
```

- The root element is `<recipe>`
- The nested elements inside the root element (which have to occur exactly once) are: `<title>` (the title of the recipe), `<ingredients>` (a list of `<ingredient>` elements defining all the ingredients required), `<preparation>` (a list of `<step>` elements defining the preparation steps)
- An `<ingredient>` element can occur multiple times inside an `<ingredients>` element, has a `string` as content and has 3 attributes: `quantity`, `unit` and `food`
- A `<step>` element can occur multiple times inside the `<preparation>` element and has a `string` as content

Appendix 3.

An example of a valid recipe (according to the XML Schema from CccSystemOutput.xsd) provided as output for the *Computer Cooking Contest*

```
<?xml version="1.0" encoding="utf-8" ?>
<cccAnswer>
    <system>http://getCccCocktail.net/</system>
    <query>d=Strawberry%20syrup|Apple%20juice&amp;u=Mint</query>
    <retrieve>
        <recipe>
          <title>Bora bora</title>
          <ingredients>
            <ingredient quantity="10" unit="cl" food="pineapple juice">10 cl pineapple
juice</ingredient>
            <ingredient quantity="6" unit="cl" food="passion fruit juice">6 cl passion
fruit juice</ingredient>
            <ingredient quantity="2" unit="cl" food="grenadine">2 cl grenadine
syrup</ingredient>
            <ingredient quantity="1" unit="cl" food="lemon juice">1 cl lemon
juice</ingredient>
            <ingredient quantity="3" unit="" food="ice cube">3 ice cubes</ingredient>
          </ingredients>
          <preparation>
            <step>Make this recipe in a shaker</step>
            <step>Serve in a glass over ice with a slice of orange</step>
          </preparation>
        </recipe>
    </retrieve>
    <reuse>
        <adaptation>Replace pineapple juice (10 cl) with apple juice (10 cl). Replace
grenadine (2 cl) with strawberry syrup (3cl)</adaptation>
        <recipe>
          <title>Adaptation of Bora bora</title>
          <ingredients>
            <ingredient quantity="10" unit="cl" food="apple juice">10 cl apple
juice</ingredient>
            <ingredient quantity="6" unit="cl" food="passion fruit juice">6 cl passion
fruit juice</ingredient>
            <ingredient quantity="3" unit="cl" food="strawberry syrup">3 cl strawberry
syrup</ingredient>
            <ingredient quantity="1" unit="cl" food="lemon juice">1 cl lemon
juice</ingredient>
            <ingredient quantity="3" unit="" food="ice cube">3 ice cubes</ingredient>
          </ingredients>
          <preparation>
            <step>Make this recipe in a shaker</step>
            <step>Serve in a glass over ice with a slice of orange</step>
          </preparation>
        </recipe>
    </reuse>
</cccAnswer>
```

- The structure of this XML file (which contains the output of a system for a given query) respects the XML Schema defined in CccSystemOutput.xsd file (*Appendix 1*)
- The structure of the recipe included in the answer respects the XML Schema defined in CccRecipe.xsd file (*Appendix 2*)

Appendix 4.
The *WikiTaaable*'s *food* ontology provided in RDF format (food.rdf[3])

- An OWL class is defined for each ingredient of the food ontology. All the information about each ingredient is contained inside an **<owl:Class>** element (this element represents the root element of each class). This element has a single attribute (**rdf:about**) representing the subject of that class given as an URI

```
    <owl:Class
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot">
```

- Some of the most important nested elements inside the root element of each class are:
    - **<rdfs:label>** element has as content a string representing the name of the ingredient

```
    <rdfs:label>Carrot</rdfs:label>
```

    - **<property:Compatible_with_diet>** elements give information about the diets which can use that ingredient. The diets are specified as URIs in the values of the rdf:resource attributes

```
    <property:Compatible_with_diet rdf:resource="&wiki;Category-3AVeganism"/>
```

(where **&wiki;** is a prefix (macro) defined at the begining of the document, used to replace the common part of all the URIs)

    - Information about the energy quantity of the current ingedient is given in the content of the child elements **<property:Energy_quantity>** and **<property:Energy_unit>** of the element **<property:Has_energy>**

```
    <property:Has_energy>
        <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-23_56c99e5d4de4c39edb5eb73b9bd6da09">
            <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
            <property:Energy_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">41</property:Energy_quantity>
            <property:Energy_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">kcal</property:Energy_unit>
        </swivt:Subject>
    </property:Has_energy>
```

[3] The food.rdf file can be downloaded from the following address: http://wikitaaable.loria.fr/rdf/food.rdf

- The external link to the *Wikipedia* page is given as the value of the attribute `rdf:resource` of the element **`<property:Has_external_link>`**

```
    <property:Has_external_link
rdf:resource="http://en.wikipedia.org/wiki/index.html?curid=5985739"/>
```

- **`<property:Has_lexical_variant>`** elements contain the translation of the ingredient's name in different languages

```
    <property:Has_lexical_variant>
                <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_5d99c2df45d287c03d75af50afb34878">
                    <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                    <property:Language
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Français</property:Language>
                    <property:Plural
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">carottes</property:Plural>
                    <property:Singular
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">carotte</property:Singular>
                </swivt:Subject>
     </property:Has_lexical_variant>
```

- The nutritional values are given in the elements **`<property:Has_nutritional_value>`**. Each of these elements contain three nested elements: **`<property:Nutritional_fact>`**, **`<property:Nutritional_quantity>`** and **`<property:Nutritional_unit>`**. Their content defines the name, the quantity and the measurement unit of each nutritional element

```
    <property:Has_nutritional_value>
                <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_3766f5d07432f7cdb5c9f150a895bef1">
                    <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                    <property:Nutritional_fact
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Sugar</property:Nutritional_fact>
                    <property:Nutritional_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">4.74</property:Nutritional_quanti
ty>
                    <property:Nutritional_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">g</property:Nutritional_unit>
                </swivt:Subject>
    </property:Has_nutritional_value>
```

- The parent class of each ingredient is given as `URI` value for the attribute `rdf:resource` of the element **`<rdfs:subClassOf>`**. An ingrendient can have multiple parents (i.e. multiple **`<rdfs:subClassOf>`** elements).

```
    <rdfs:subClassOf rdf:resource="&wiki;Category-3ARoot"/>
```

- Each of these child – parent relations (i.e. generalization relations) has a cost associated, which can be interpreted as the generalization effort or as the similarity level between the two connected classes. This cost is represented as a real value between 0 and 1. It is given as the content of the element **<property:Generalisation_cost>** nested on the second level inside the element **<property:Has_generalisation_cost>**. The URI of the parent class is given as the value of the attribute **rdf:resource** of the element **<property:Generalisation_class>**.

```
    <property:Has_generalisation_cost>
        <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-
3ACarrot-23_abe77c56551f06b607dcd1d019988cc8">
            <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
            <property:Generalisation_class rdf:resource="&wiki;Category-
3ARoot"/>
            <property:Generalisation_cost
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.0067024128686327</prop
erty:Generalisation_cost>
        </swivt:Subject>
    </property:Has_generalisation_cost>
```

- Each possible weight conversion between two different measurement units are given in the element **<property:Weight_conversion>** which has as children the elements **<property:Original_quantity>**, **<property:Original_unit>** and **<property:Final_quantity>** (the final unit is always grams). Additionaly, an extra children can exist for some ingredients **<property:Qualifiers>**. Multiple <property:Weight_conversion> elements can exist for one ingredient if multiple weight conversions possibilities are defined.

```
    <property:Weight_conversion>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Cat
egory-3ACarrot-23_3901281c1199d120435d198c566700ba">
                <swivt:masterPage rdf:resource="&wiki;Category-
3ACarrot"/>
                <property:Final_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">50</property:Fina
l_quantity>
                <property:Original_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">1</property:Origi
nal_quantity>
                <property:Original_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">unit</property:Or
iginal_unit>
                <property:Qualifiers
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">small</property:Q
ualifiers>
            </swivt:Subject>
    </property:Weight_conversion>
```

- Sections of the code describing the ingredient `Carrot` are given below

```
    <owl:Class
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot">
        <rdfs:label>Carrot</rdfs:label>
        <swivt:page
rdf:resource="http://wikitaaable.loria.fr/index.php/Category:Carrot"/>
        <rdfs:isDefinedBy
rdf:resource="http://wikitaaable.loria.fr/index.php/Special:ExportRDF/Category:Carrot"/>
        <swivt:wikiNamespace
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">14</swivt:wikiNamespace>
        <property:Compatible_with_diet rdf:resource="&wiki;Category-3AVeganism"/>
        <property:Compatible_with_diet rdf:resource="&wiki;Category-3AVegetarian"/>
        <property:Compatible_with_diet rdf:resource="&wiki;Category-3AGluten-
2Dfree_diet"/>
        <property:Has_description
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The carrot (Daucus carota subsp.
sativus, Etymology: Middle French carotte, from Late Latin carōta, from Greek καρότον
karōton, originally from the Indo-European root ker- (horn), due to
its...</property:Has_description>
         <property:Has_energy>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_56c99e5d4de4c39edb5eb73b9bd6da09">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Energy_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">41</property:Energy_quantity>
                <property:Energy_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">kcal</property:Energy_unit>
            </swivt:Subject>
        </property:Has_energy>
        <property:Has_external_link
rdf:resource="http://en.wikipedia.org/wiki/index.html?curid=5985739"/>
        <property:Has_generalisation_cost>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_abe77c56551f06b607dcd1d019988cc8">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Generalisation_class rdf:resource="&wiki;Category-3ARoot"/>
                <property:Generalisation_cost
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.0067024128686327</property:Gene
ralisation_cost>
            </swivt:Subject>
        </property:Has_generalisation_cost>
        <property:Has_lexical_variant>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_637bf87a1ccea8b197073f88c416283b">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Language
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">English</property:Language>
                <property:Plural
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Plural not
available</property:Plural>
                <property:Singular
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">carrot</property:Singular>
            </swivt:Subject>
        </property:Has_lexical_variant>
```

```
        <property:Has_lexical_variant>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_5d99c2df45d287c03d75af50afb34878">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Language
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Français</property:Language>
                <property:Plural
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">carottes</property:Plural>
                <property:Singular
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">carotte</property:Singular>
            </swivt:Subject>
        </property:Has_lexical_variant>
```

```
        <property:Has_number_of_cases
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">91</property:Has_number_of_cases>
        <property:Has_nutritional_value>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_bf7a4b36b254ff5ef3e455767678c2b4">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Nutritional_fact
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Carbohydrate</property:Nutritiona
l_fact>
                <property:Nutritional_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">9.58</property:Nutritional_quanti
ty>
                <property:Nutritional_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">g</property:Nutritional_unit>
            </swivt:Subject>
        </property:Has_nutritional_value>
        <property:Has_nutritional_value>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_3766f5d07432f7cdb5c9f150a895bef1">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Nutritional_fact
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Sugar</property:Nutritional_fact>
                <property:Nutritional_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">4.74</property:Nutritional_quanti
ty>
                <property:Nutritional_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">g</property:Nutritional_unit>
            </swivt:Subject>
        </property:Has_nutritional_value>
```

```
        <property:Has_picture>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_6ce3aaa46557e4e01dd4df5afba308d6">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Picture_description
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Harvested
carrots</property:Picture_description>
                <property:Picture_file
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">File:Carrot0.jpg</property:Pictur
e_file>
            </swivt:Subject>
        </property:Has_picture>
```

```
        <property:Weight_conversion>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_46aec50f6c3957c4e61401658878e4f2">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Final_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">128</property:Final_quantity>
                <property:Original_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">1</property:Original_quantity>
                <property:Original_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">c</property:Original_unit>
                <property:Qualifiers
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">chopped</property:Qualifiers>
            </swivt:Subject>
        </property:Weight_conversion>
        <property:Weight_conversion>
            <swivt:Subject
rdf:about="http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3ACarrot-
23_af68bcd613620fe328e395868471918e">
                <swivt:masterPage rdf:resource="&wiki;Category-3ACarrot"/>
                <property:Final_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">110</property:Final_quantity>
                <property:Original_quantity
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">1</property:Original_quantity>
                <property:Original_unit
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">c</property:Original_unit>
                <property:Qualifiers
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">grated</property:Qualifiers>
            </swivt:Subject>
        </property:Weight_conversion>

        <swivt:wikiPageModificationDate
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2012-11-
20T00:34:55Z</swivt:wikiPageModificationDate>
        <swivt:wikiPageSortKey
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Carrot</swivt:wikiPageSortKey>
        <rdfs:subClassOf rdf:resource="&wiki;Category-3ARoot"/>
    </owl:Class>
```

Appendix 5.
The Python implementation of the RDF parser (parseRDFFoodOntology.py) for the food.rdf[4] file containing *WikiTaaable*'s *food* ontology

- The input of the parser is the RDF file food.rdf containing the *WikiTaaable*'s *Food* ontology;
- The output of the parser (the returned items) is represented by three lists:
  - The first one contains the name of all the classes of the ontology, given as strings (i.e. all the name of the ingredients);
  - The second one contains the generalization relationships between ontology's classes (i.e. ingredients) (e.g. the *Vegetable* class generalizes the *Onion* class with the cost of 0.2278820375), given as instances of the GeneralizationRelation class;
  - The third one contains for each ingredient a list of possible weight conversions between different measurement units (e.g. half of salmon fillet unit is similar with 159 *grams* of salmon fillet);
- More details about the implementation are given in *Section 4.1*.

```
# Parse the food ontology RDF file food.rdf. Return a list of
generalization relations (GeneralizationRelation objects).

import xml.etree.ElementTree as ET

URI_OWL = "{http://www.w3.org/2002/07/owl#}"
URI_RDF_SYNTAX = "{http://www.w3.org/1999/02/22-rdf-syntax-ns#}"
URI_RDF_SCHEMA = "{http://www.w3.org/2000/01/rdf-schema#}"
URI_CATEGORY =
"http://wikitaaable.loria.fr/index.php/Special:URIResolver/Category-3A"
URI_PROPERTY =
"{http://wikitaaable.loria.fr/index.php/Special:URIResolver/Property-
3A}"


class GeneralizationRelation:
    def __init__(self, parent, child, generalizationCost):
        self.parent = parent
        self.child = child
        self.generalizationCost = generalizationCost

    def __str__(self):
        return self.parent + " generalizes " + self.child + " with cost
" + str(self.generalizationCost)


def parseRDFFoodOntology():
    print("\n ----- Parsing food ontology RDF file food.rdf ... -----
\n")
```

[4] The food.rdf file can be downloaded from the following address: http://wikitaaable.loria.fr/rdf/food.rdf

```
    tree = ET.parse('food.rdf')
    root = tree.getroot()

    ingredientList = []
    generalizationList = []
    ingredientWeightConversion = []

    for ingredientClass in root:

        if ingredientClass.tag == URI_OWL + 'Class':

            # 'about' attribute
            ingredientAbout = ingredientClass.attrib[URI_RDF_SYNTAX +
'about'].replace(URI_CATEGORY, "")
            ingredientList.append(ingredientAbout)

            # 'label' element
            ingredientLabel = ingredientClass.find(URI_RDF_SCHEMA +
'label')
            if ingredientLabel is not None:
                ingredientLabel = ingredientLabel.text
            else:
                ingredientLabel = ""

            # 'subClassOf' elements
            parentsElements = ingredientClass.findall(URI_RDF_SCHEMA +
'subClassOf')
            ingredientParents = []
            for parent in parentsElements:
                ingredientParents.append(parent.attrib[URI_RDF_SYNTAX +
'resource'].replace(URI_CATEGORY, ""))

            # 'Has_generalisation_cost' elements ('Generalisation_class'
and 'Generalisation_cost' elements)
            generalizationsElements =
ingredientClass.findall(URI_PROPERTY + 'Has_generalisation_cost')
            ingredientGeneralizations = {}
            for generalisation in generalizationsElements:
                generalizationParent =
generalisation[0].find(URI_PROPERTY +
'Generalisation_class').attrib[URI_RDF_SYNTAX +
'resource'].replace(URI_CATEGORY, "")
                generalizationCost = generalisation[0].find(URI_PROPERTY
+ 'Generalisation_cost').text
                ingredientGeneralizations[generalizationParent] =
generalizationCost

            for parent in ingredientParents:
                if parent in ingredientGeneralizations:
                    generalizationObj = GeneralizationRelation(parent,
ingredientAbout, ingredientGeneralizations[parent])
                else:
                    generalizationObj = GeneralizationRelation(parent,
ingredientAbout, 0)
                generalizationList.append(generalizationObj)

    print("\n ----- Parsing food ontology RDF file food.rdf DONE -----
\n")

    return ingredientList, generalizationList
```

```
            # 'WeightConversion' elements
            conversionElements = ingredientClass.findall(URI_PROPERTY +
'Weight_conversion')
            for conversion in conversionElements:
                conversionFinalQuantity =
float(conversion[0].find(URI_PROPERTY + 'Final_quantity').text)
                conversionOriginalQuantity =
float(conversion[0].find(URI_PROPERTY + 'Original_quantity').text)
                conversionFinalUnit = "g"

                try:
                    conversionQualifier =
conversion[0].find(URI_PROPERTY + 'Qualifiers').text.lower()
                except:
                    conversionQualifier = ""

                try:
                    conversionOriginalUnit =
conversion[0].find(URI_PROPERTY + 'Original_unit').text.lower()
                except:
                    conversionOriginalUnit = "unit"

                ingredientWeightConversion[-
1].append((conversionOriginalQuantity, conversionOriginalUnit,
conversionQualifier, conversionFinalQuantity, conversionFinalUnit))

    print("\n ----- Parsing food ontology RDF file food.rdf DONE -----
\n")

    return ingredientList, generalizationList,
ingredientWeightConversion
```

Appendix 6.
The Python implementation of the XML parser (parseXMLSaladRecipes.py) for the ccc_salad.xml[5] file containing the Salad recipes for the *Computer Cooking Contest*

- The input of the parser is the XML file ccc_salad.xml containing the provided list of salad recipes for the *Computer Cooking Contest* (*Salad challenge*);
- The output of the parser (the returned item) is a single list which contains the recipes, stored as instances of the Recipe class;
- More details about the implementation are given in *Section 4.2*.

```
# Parse the salad recipes XML file ccc_salad.xml. Return a list of
recipes (Recipe objects).

import xml.etree.ElementTree as ET


class Ingredient:
    def __init__(self, name, quantity, unit, qualifier):
        self.name = name
        self.quantity = quantity
        self.unit = unit
        self.qualifier = qualifier

    def __str__(self):
        return str(self.quantity) + " " + self.unit + " " + self.name +
" " + self.qualifier


class Recipe:
    def __init__(self, number, name, ingredients, steps, excludedDiets):
        self.id = number
        self.name = name
        self.ingredients = ingredients
        self.steps = steps
        self.excludedDiets = excludedDiets

    def __str__(self):
        return self.name
```

[5] The ccc_salad.xml file can be downloaded from the following address: https://github.com/nanajjar/Computer-Cooking-Contest/blob/master/ccc_salad.xml

```
def parseXMLSaladRecipes():
    print("\n ----- Parsing salad recipes XML file ccc_salad.xml ... ---
-- \n")

    tree = ET.parse('ccc_salad.xml')
    root = tree.getroot()

    recipeList = []
    for recipe in root:

        # 'recipeid' element
        recipeID = recipe[0].text

        # 'title' element
        recipeName = recipe[1].text

        # 'ingredients' element ('ingredient' elements)
        recipeIngredients = []
        for ingredientXmlElem in recipe[2]:
            ingredientObj =
Ingredient(ingredientXmlElem.attrib['ingredient'],

float(ingredientXmlElem.attrib['quantity']) if
ingredientXmlElem.attrib['quantity'] != "" else 0,
                                        ingredientXmlElem.attrib['unit'],

ingredientXmlElem.attrib['qualifiers'])
            recipeIngredients.append(ingredientObj)

        # 'preparation' element ('step' elements)
        recipeSteps = []
        for step in recipe[3]:
            recipeSteps.append(step.text)

        # 'diet' element ('exclude-for-diet' elements)
        recipeDiets = []
        for diet in recipe[4]:
            recipeDiets.append(diet.text)

        recipeObj = Recipe(recipeID, recipeName, recipeIngredients,
recipeSteps, recipeDiets)

        recipeList.append(recipeObj)

    print("\n ----- Parsing salad recipes XML file ccc_salad.xml DONE --
--- \n")

    return recipeList
```

Appendix 7.
The Python implementation of the TXT parser (parseTXTQuery.py) for the query.txt file containing the query for the *Computer Cooking Contest*

- The query is given in a text file (query.txt) and contains two lines:
  - On the first line are enumerated all the desired ingredients, separated between them with single spaces. These are written in the format used in the *food* ontology file (food.rdf) for the values of the about attribute of the <owl:Class> element (i.e. if an ingredient name is composed of multiple words, these are separated with underscores, not with blank spaces (e.g. White_salsify));
  - On the second line are enumerated all the undesired ingredients, separated between them with single spaces. These ingredients are also written as described above;
- The input of the parser is the TXT file query.txt containing the desired and undesired ingredients list;
- The output of the parser (the returned items) are two lists: one for the desired ingredients and the second one for the undesired ingredients;
- More details about the implementation are given in *Section 4.3*.

```
def parseTXTQuery():

    queryFile = open('query.txt', 'r')
    queryFileLines = queryFile.read().splitlines()

    desiredIngredients = queryFileLines[0].split(' ')

    undesiredIngredients = queryFileLines[1].split(' ')

    queryFile.close()

    return desiredIngredients, undesiredIngredients
```

Alexandru Cohal
Iași, Romania
June 2017