Taaable: a Case-Based System for personalized Cooking

Amélie Cordier, Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Fadi Badra, Julien Cojan, Emmanuelle Gaillard, Laura Infante-Blanco, Pascal Molli, Amedeo Napoli, et al.

Chapter 1
TAAABLE: a Case-Based System for personalized Cooking

Amélie Cordier[1], Valmi Dufour-Lussier[2], Jean Lieber[2], Emmanuel Nauer[2], Fadi Badra[4], Julien Cojan[2], Emmanuelle Gaillard[2], Laura Infante-Blanco[2], Pascal Molli[3], Amedeo Napoli[2], Hala Skaf-Molli[3]

Abstract

TAAABLE is a Case-Based Reasoning (CBR) system that uses a recipe book as a case base to answer cooking queries. TAAABLE participates in the Computer Cooking Contest since 2008. Its success is due, in particular, to a smart combination of various methods and techniques from knowledge-based systems: CBR, knowledge representation, knowledge acquisition and discovery, knowledge management, and natural language processing. In this chapter, we describe TAAABLE and its modules. We first present the CBR engine and features such as the retrieval process based on minimal generalization of a query and the different adaptation processes available. Next, we focus on the knowledge containers used by the system. We report on our experiences in building and managing these containers. The TAAABLE system has been operational for several years and is constantly evolving. To conclude, we discuss the future developments: the lessons that we learned and the possible extensions.

1.1 Introduction

TAAABLE is a Case-Based Reasoning system that provides cooking recipes in response to queries from users. When no recipe can be found that satisfies a query, an existing recipe is adapted. TAAABLE was developed to take part in the Computer Cooking Contest (CCC),[1] a competition held every year

[1] LIRIS (CNRS, Université Claude Bernard Lyon 1)
[2] LORIA (CNRS, INRIA, Université de Lorraine)
[3] LINA (CNRS, Université de Nantes)
[4] LIM&BIO (Université Paris 13)

[1] http://computercookingcontest.net

since 2008 during the International Conference on Case-Based Reasoning (ICCBR). The initial goal of the contest was to provide a common ground to allow for different CBR systems could compete and be compared. Therefore, it provided a benchmark for CBR tools. To this end, the contest organisers have made a *recipe book* available to participants and have created various challenges. Every year, each system is tested in front of a live audience at the conference and is evaluated by a panel of experts. The best system in each category is selected. TAAABLE has participated in the CCC since the first edition and has won several awards.

Several French researchers, from different laboratories, worked together to design and implement TAAABLE. They combined their skills and knowledge of various research issues: knowledge representation and knowledge management, case base organisation and representation, development of a similarity measure, adaptation knowledge acquisition, formal representation of preparations, retrieval and adaption strategies in CBR, etc. These different questions, investigated during the development of the TAAABLE project, are addressed in this chapter.

It is important to note that, although TAAABLE was primarily developed to enter the CCC and to address the CCC challenges, it also served as the subject of other research, which is also described in this chapter and published in various papers. TAAABLE has served as an application domain and as an experimentation and testing ground for three PhD students. Furthermore, it was the starting point of a wider research project, funded by the French national agency for research: the Kolflow project.[2]

This chapter is organised as follows. In the rest of the introduction, we provide the reader with background knowledge on the CCC and we give a general overview of the TAAABLE system and architecture. In Sect. 1.2, we describe the inference engine of TAAABLE, which implements the fundamental operations of CBR: case retrieval and adaptation. These two operations are described in details. In Sect. 1.3, we study TAAABLE from the point of view of the knowledge it uses. We report the methods and tools we used in order to acquire, represent and manage this knowledge. We detail the knowledge containers used in the system: domain knowledge (ontologies), cases, and adaptation knowledge. Finally, we discuss the lessons we have learnt during the project and we briefly report our plans for future work.

The Computer Cooking Contest (CCC). The CCC is an annual competition that is organised by the international CBR community. Each year, the organisers provide the participants with a recipe book and define a set of challenges. Participants implement systems addressing these challenges by using the recipe book as a case base. Each participant describes their system in a technical paper and gives a live demonstration during the conference. Systems are evaluated against several *criteria* by a panel of scientists and

[2] http://kolflow.univ-nantes.fr

```
<RECIPE>
<TI>Glutinous Rice with Mangoes</TI>
<IN>3 c Glutinous rice</IN>
<IN>1 1/2 c Coconut cream</IN>
<IN>1/2 c Sugar</IN>
<IN>1 ts Salt</IN>
<IN>1 1/4 c Coconut cream</IN>
<IN>2 tb Sugar</IN>
<IN>1/4 ts Salt</IN>
<IN>6 Ripe mangoes, well chilled</IN>
<IN>2 tb Sesame seeds, toasted</IN>
<PR>SEASONINGS SAUCE GARNISH Soak the rice in cold water for 2 hours.
Drain. Line a steamer with cheesecloth, heat steamer and lay rice on the
cheesecloth. Steam for 30 minutes or until cooked through. The rice will
become glossy. Mix the SEASONINGS ingredients in a large bowl and gently
mix in the hot steamed rice. Cover tightly and let soak for 30 minutes to
absorb the coconut flavour. Blend the SAUCE ingredients in a pot and heat
until it just reaches the boiling point. Let cool. Peel the mangoes, slice
lengthwise and remove the pits. Divide the rice among 6 plates. Place mango
slices on top and cover with the sauce. Sprinkle with the sesame seeds and
serve.
</PR>
</RECIPE>
```

Fig. 1.1 An example of recipe in the CCC recipe book.

cooking experts. The recipe book provided by the organisers is a simple XML file containing recipes. The schema of a recipe consists of one level of tags only, with a title, a list of ingredients and a textual instructions that we call *preparation*. Fig. 1.1 is an example of a recipe.

Using this recipe book is challenging because ingredients and preparations are written in natural language and may contain spelling mistakes. They cannot be used as the input of a computer program in their raw form. Therefore, a pre-processing step is required.

Over the years, the CCC organisers have proposed various challenges that are described hereafter. TAAABLE addressed all but the menu challenge.

The **main challenge.** "Given a query from a user, expressed in natural language, find a recipe satisfying this query. The proposed recipe must necessarily be adapted from a recipe of the recipe book." For example, "I would like a recipe with escarole endive and lemon juice but without onions" could be a query of the main challenge.

The **adaptation challenge.** "Given a recipe (from the recipe book) and a set of constraints, provide a suitable adaptation of this recipe." For example, given the recipe "Baked apple pancake" and the fact that I want to use bananas instead of apples, how do I adapt the recipe?

The **menu challenge.** "Given a query from a user, propose a suitable three-course menu." The user gives a list of ingredients and the goal is to retrieve three recipes using these ingredients.

The **healthy challenge.** This challenge is similar to the main challenge, but it includes additional constraints on special diets such as vegetarianism or a gluten-free diet.

The **open challenge.** This challenge was created in order to allow participants to investigate specific issues and demonstrate their results during the contest.

Major competing systems.

From 2008 to 2010, many systems participated in the CCC. Even if they address the same problem, these systems are different in the way they index, retrieve and adapt cases, and in the type of adaptation they address. The "What's in the fridge?" system focuses on the recipe indexing using an active learning approach. The case retrieval and adaptation is based on a classical information retrieval technique, using Wordnet to compute similarity measures between the query and the set of recipes [24]. ColibriCook uses also a simple information retrieval technique: cases are retrieved using a similarity measure based on whether the elements of the query (ingredients, types, etc.) appear in the recipes [9]. Other systems like JadaCook [10] or CookIIS [15] take advantage of hierarchies to compute similarity. JadaCook uses a hierarchy of classes in order to compute the similarity between the query and the cases. The case retrieval takes into account the distance between the ingredients in the hierarchy. CookIIS also uses the hierarchy of ingredients, but do so in order to compute the similarity between ingredients. These similarities are then used both to compute the similarity between the query and the cases for case retrieval and to compute the adaptation. In addition, CookIIS uses ingredient similarities extracted from cooking websites [15]. CookIIS also addresses the adaptation of the recipe text, by making substitutions at the string level: a string (name of an ingredient) is replaced by another string (name of another ingredient). CookingCakeWF [22] also addresses the adaptation of the preparation, considering the instructions as a workflow. However, this system does not deal with the automatic transformation of the textual preparation into a workflow.

An overview of the Taaable user interface. The user interface of Taaable[3] consists of a query and result interfaces (see Fig. 1.2). The upper part of the figure shows a query input (dessert dish, with rice and figs, no particular diet). The lower part of the figure shows the answer to the query– only one recipe is returned in this example–and the associated adaptation (replace Mango with Fig).

The user can see the original recipe as well as the details of the adaptation by clicking on the available links. Fig. 1.3 shows the result of the adaptation

[3] http://taaable.fr

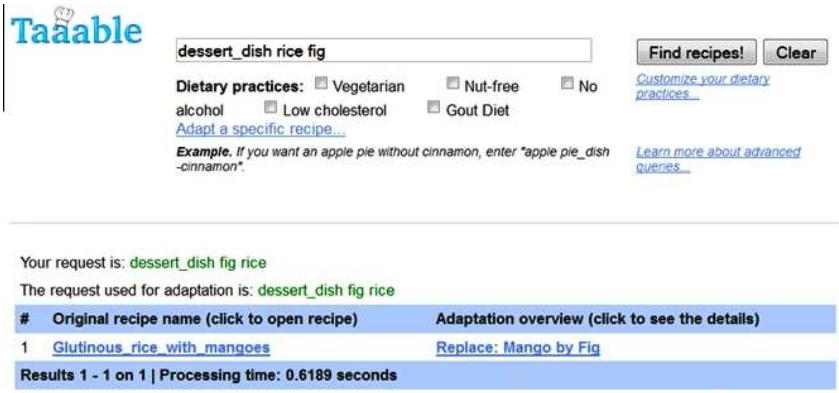

Fig. 1.2 An illustration of the TAAABLE user interface.

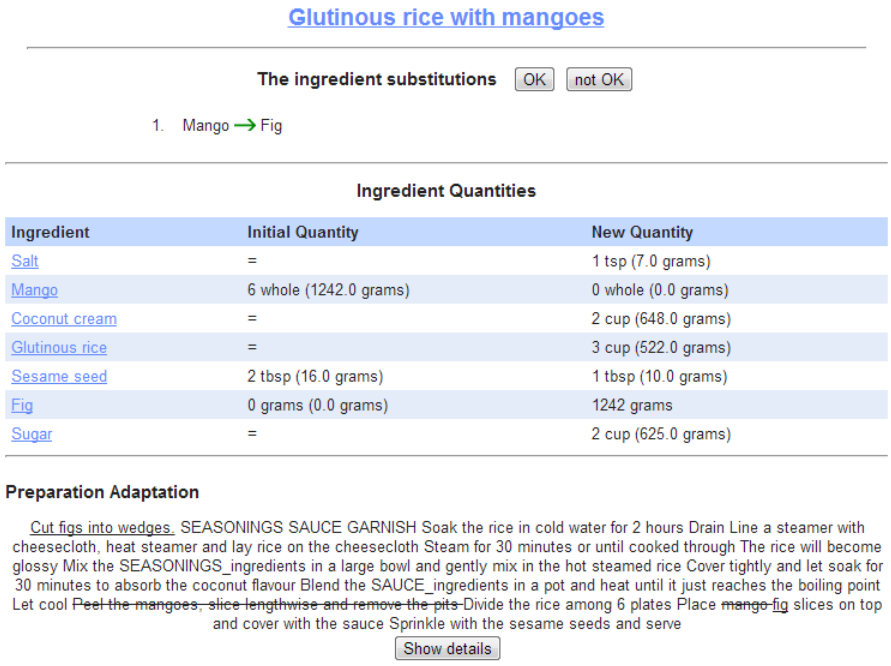

Fig. 1.3 An example of adaptation results.

of the recipe "Glutinous Rice with Mangoes" obtained by replacing mangoes with figs. The adaptation is composed of three parts. First, the ingredient substitutions are mentioned. Second, the ingredient quantities are adapted. Third, the textual preparation of the recipe is adapted. These three types of adaptation and the processes that are involved are detailed in Sect. 1.2.

An overview of the TAAABLE architecture. As a CBR system, TAAABLE takes as input a query from a user and searches for a similar case in the case base, adapts the retrieved case, and presents the result. It uses most notably a case base (the recipe book), a domain knowledge base (about cooking), and an adaptation knowledge base. Early version of the system used knowledge bases stored in XML files and built and managed by hand, but this proved to be impossible to maintain. For this reason, we decided to use a common tool to manage all the knowledge sources: a semantic wiki named WIKITAAABLE.

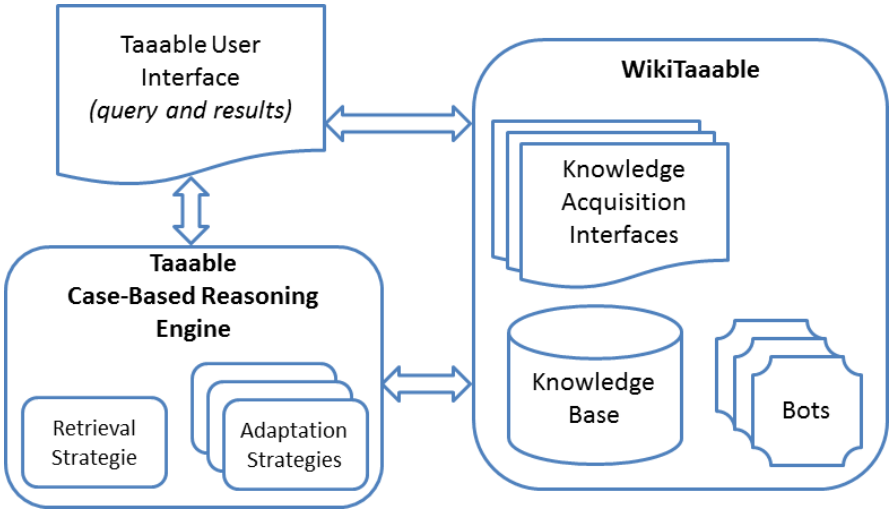

Fig. 1.4 General architecture of the TAAABLE application.

Fig. 1.4 presents the general architecture of TAAABLE. The TAAABLE user interface makes it possible to query the system and to display the results provided by the TAAABLE CBR engine. The CBR engine is connected to WIKITAAABLE, which contains the knowledge base. WIKITAAABLE provides the users the way to browse and edit the knowledge units and to participate in knowledge acquisition tasks. Some specific interfaces for knowledge acquisition (in particular, adaptation knowledge acquisition) are implemented as modules and are integrated within WIKITAAABLE. It must be noted that WIKITAAABLE also embeds bots (i.e. programs that perform automated tasks) that implement specialised maintenance tasks related to the wiki. TAAABLE CBR engine is described in Sect. 1.2. WIKITAAABLE is described in Sect. 1.3.

1.2 TAAABLE inference engines

1.2.1 Case retrieval

In TAAABLE, the retrieval of recipes consists in selecting recipes `R` from the recipe base `Recipes` (used as a case base[4]), where `R` is a best match to the query `Q`. This best match is computed by finding a generalisation $\Gamma(\texttt{Q})$ of `Q` that is minimal (according to a cost function) and such that there is at least one recipe exactly matched by $\Gamma(\texttt{Q})$. This matching takes into account the domain knowledge `DK`.

The domain knowledge `DK` can be considered as a set of axioms $a \Rightarrow b$ in propositional logic, where a and b are propositional variables representing recipe classes. For example, `lemon` (resp., `citrusfruit`) represents the class of recipes having lemons (resp., citrus fruits) as ingredients and the axiom `lemon` $\Rightarrow$ `citrusfruit` states that every recipe with lemon is a recipe with citrus fruit. In fact, each food name x is interpreted as "the class of recipes having x as ingredient". There are other propositional variables, such as `mediterranean` that represents the class of Mediterranean recipes. Therefore, `oliveoil` $\Rightarrow$ `mediterranean` states that every recipe with olive oil is a Mediterranean recipe. From an implementation viewpoint, `DK` is a hierarchy, i.e. a directed acyclic graph whose nodes are propositional variables and edges $a \longrightarrow b$ correspond to axioms $a \Rightarrow b$. $\top$, the root of this hierarchy, denotes the recipe universe.

Sect. 1.3.1 presents the acquisition of a domain ontology $\mathcal{O}$ given by class hierarchies, organised with a "more specific than" relation denoted by $\sqsubseteq$. For instance, there is a food hierarchy in $\mathcal{O}$ stating, in particular, that `Lemon` $\sqsubseteq$ `CitrusFruit`. The domain knowledge `DK` used for case retrieval and case adaptation is based on this ontology with a change of formalism (from the ontology language to propositional logic). In particular, `lemon` represents the recipes with lemon and, since `Lemon` is a subclass of `CitrusFruit`, each recipe with lemon is a recipe with fruit, hence the axiom `lemon` $\Rightarrow$ `citrusfruit` of `DK`.

Fig. 1.5 is an excerpt of the domain knowledge hierarchy (the valuation on the edges represents costs that are explained further).

The recipe base `Recipes` is the case base of the system. These recipes are provided by the CCC organisers (there were about 900 source recipes in the first CCC and about 1500 ones for the further CCCs). A recipe `R` $\in$ `Recipes` is described by a shallow XML document that has to be transformed within

[4] For the adaptation challenge, the recipe base contains a sole case, the recipe that must be adapted.

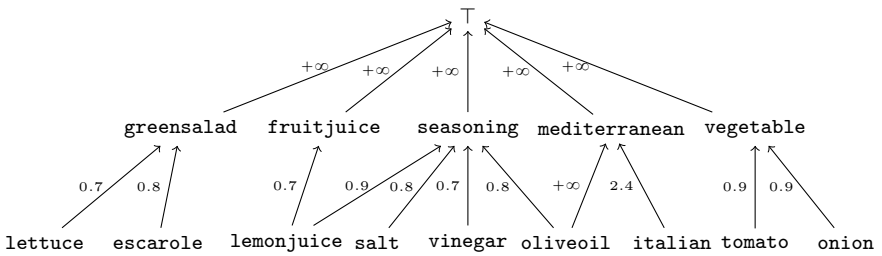

Fig. 1.5 An excerpt of domain knowledge with costs. The edge $a \xrightarrow{c} b$ means that $a \Rightarrow b \in \mathtt{DK}$ and $\mathtt{cost}(a \rightsquigarrow b) = c$.

a representation formalism. For case retrieval, $\mathtt{R}$ is transformed into $idx(\mathtt{R})$ (the index of $\mathtt{R}$ which is a conjunction of literals) thanks to an annotation process presented in Sect. 1.3.4. For example,

$$idx(\mathtt{R}) = \mathtt{lettuce} \wedge \mathtt{vinegar} \wedge \mathtt{oliveoil} \wedge \mathtt{tomato} \wedge \textit{nothing else} \quad (1.1)$$

is a formal and abstracted representation of the recipe $\mathtt{R}$ whose ingredients are a lettuce, vinegar, olive oil, tomatoes, and nothing else. A closed world assumption is associated to $idx(\mathtt{R})$: if a property cannot be deduced from $idx(\mathtt{R})$ and the domain knowledge $\mathtt{DK}$, then it is considered as false. In other words, if $idx(\mathtt{R}) \not\models_{\mathtt{DK}} a$ then the term *nothing else* is a conjunction of literals containing the literal $\neg a$. In the example above, $idx(\mathtt{R}) \models_{\mathtt{DK}} \neg\mathtt{meat} \wedge \neg\mathtt{fish}$. It can be noticed that $idx(\mathtt{R})$ has exactly one model: for each propositional variable a, either $idx(\mathtt{R}) \models_{\mathtt{DK}} a$ or $idx(\mathtt{R}) \models_{\mathtt{DK}} \neg a$.

Query $\mathtt{Q}$ is represented by a conjunction of literals. For example,

$$\mathtt{Q} = \mathtt{escarole} \wedge \mathtt{lemonjuice} \wedge \neg\mathtt{onion} \quad (1.2)$$

represents the query "I would like a recipe with escarole and lemon juice (among other ingredients) but without onions."

Recipe retrieval is described in the algorithm 1 and is an application of smooth classification as described in [19]. It consists in finding a generalisation $\Gamma(\mathtt{Q})$ of $\mathtt{Q}$ that exactly matches at least one recipe of the recipe base (i.e. there exists $\mathtt{R} \in \mathtt{Recipes}$ such that $idx(\mathtt{R}) \models_{\mathtt{DK}} \Gamma(\mathtt{Q})$). Lines 4 and 5 require more explanation.

The generalisation $\Gamma(\mathtt{Q})$ of $\mathtt{Q}$ is searched in a state space where each state is a conjunction of literals, the initial state is $\mathtt{Q}$ and the successors of a state s is the set of the states $\gamma(s)$ where γ is a state transition that is either

Retrieval $(\mathtt{Q}, \mathtt{Recipes}, \mathtt{DK}) : (\{\mathtt{R}_i\}_i, \Gamma)$

Input a query $\mathtt{Q}$, the recipe base $\mathtt{Recipes}$, the domain knowledge $\mathtt{DK}$
Output a set of recipes $\{\mathtt{R}_i\}_i \subseteq \mathtt{Recipes}$ and a generalisation function Γ such that each $\mathtt{R}_i$ exactly matches $\Gamma(\mathtt{Q})$

1: $\Gamma \leftarrow$ identity function
2: $\{\mathtt{R}_i\}_i \leftarrow \varnothing$
3: **while** $\{\mathtt{R}_i\}_i = \varnothing$ **do**
4: $\quad \Gamma \leftarrow$ next generalisation
5: $\quad \{\mathtt{R}_i\}_i \leftarrow \{\mathtt{R}_i \in \mathtt{Recipes} \mid idx(\mathtt{R}_i) \models_{\mathtt{DK}} \Gamma(\mathtt{Q})\}$
6: **end while**

algorithm 1: TAAABLE retrieval algorithm.

- a one step generalisation of a positive literal in s according to the ontology, e.g. if s contains the literal $\mathtt{lemon}$, then $\gamma = \mathtt{lemon} \rightsquigarrow \mathtt{citrusfruit}$ ($x \rightsquigarrow y$ is the substitution of x with y) is one of the γ's if $\mathtt{lemon} \Rightarrow \mathtt{citrusfruit}$ is an axiom of the ontology; or
- a removal of a negative literal of s, e.g. if s contains $\neg\mathtt{garlic}$, then $\gamma = \neg\mathtt{garlic} \rightsquigarrow \top$ is one of the γ's.

Thus, each generalisation of $\mathtt{Q}$ is a $\Gamma(\mathtt{Q})$ where Γ is a composition of γ's. This state space is searched according to an increasing value of $\mathtt{cost}(\Gamma)$ where $\mathtt{cost}$ is an additive cost function ($\mathtt{cost}(\Gamma_2 \circ \Gamma_1) = \mathtt{cost}(\Gamma_1) + \mathtt{cost}(\Gamma_2)$) such that the cost of the identity function is null and $\mathtt{cost}(\gamma) > 0$ for each state transition γ. So, the next generalisation mentioned in line 4 is the generalisation of cost immediately greater than the cost of Γ. Technically, this next generalisation is the first element of a list of generalisations ordered by an increasing cost, as in a classical A* search [23].

Line 5 of the algorithm is implemented in TAAABLE using a hierarchical classification of $\Gamma(\mathtt{Q})$ in a hierarchy containing the recipe indexes $idx(\mathtt{R})$. This mechanism is a classical deductive inference described, e.g. in [1].

The cost function has to be defined for each $a \rightsquigarrow b$ such that $a \Rightarrow b \in \mathtt{DK}$. Indeed, every generalisation Γ is a composition of n substitutions $a_i \rightsquigarrow b_i$, and $\mathtt{cost}(\Gamma) = \sum_{i=1}^{n} \mathtt{cost}(a_i \rightsquigarrow b_i)$ (since $\mathtt{cost}$ is additive). It is assumed that the recipe base, $\mathtt{Recipes}$, constitutes a homogeneous sampling of recipes (in the recipe space) and thus, that $\mu(x) = \dfrac{\mathcal{N}(x)}{\mathcal{N}(\top)}$ is a good approximation of the proportion of recipes of class x, where $\mathcal{N}(x)$ is the number of recipes of the class x:

$$\mathcal{N}(x) = \left|\{\mathtt{R} \in \mathtt{Recipes} \mid idx(\mathtt{R}) \models_{\mathtt{DK}} x\}\right|$$

Thus, if $a \Rightarrow b \in \mathtt{DK}$, $\mu(b) - \mu(a)$ is the proportion of recipes of class b that are not recipes of class a and thus is characteristic of the risk made by the general-

isation $a \rightsquigarrow b$. Therefore, a first idea is to define $\texttt{cost}(a \rightsquigarrow b) = \mu(b) - \mu(a)$. Now, a more accurate definition of cost is based on the adaptation-guided retrieval principle [25] stating that adaptation knowledge should be taken into account during retrieval. In particular, it is worth noticing that if a and b are ingredient-based recipe classes (e.g. $a = \texttt{lemon}$ and $b = \texttt{citrusfruit}$) then the adaptation will be substitution-based whereas if a and b are, for instance, location-based recipe classes (e.g. $a = \texttt{italian}$ and $b = \texttt{mediterranean}$) then the adaptation will be a mere copy of the source recipe, involving more risks. Therefore, the adaptation cost depends on the *type* τ of propositional variables a and b. This type is associated to each propositional variable, e.g. the type of `lemon` and of `citrusfruit` is $\tau =$ ingredient and the type of `italian` and of `mediterranean` is $\tau =$ location. Thus, a coefficient K_τ depending on the type τ of classes a and b is used and the cost is defined by:

$$\texttt{cost}(a \rightsquigarrow b) = K_\tau \frac{\mathcal{N}(b) - \mathcal{N}(a)}{\mathcal{N}(\top)} \quad \text{with } \tau \text{ the type of } a \text{ and } b$$

(if a and b are of different types, the generalisation is forbidden, e.g. $\texttt{cost}(\texttt{oliveoil} \rightsquigarrow \texttt{mediterranean}) = +\infty$). Since the adaptation related to ingredients is assumed to be less risky than the one related to location, $K_{\text{ingredient}}$ is much lower than K_{location}. In practice, the following values give satisfying results: $K_{\text{ingredient}} = 1$ and $K_{\text{location}} = 10$.

For example, let us consider the domain knowledge `DK` with the costs given by Fig. 1.5, the query of equation (1.2) and a case base containing only one case: the recipe `R` indexed by the $idx(\texttt{R})$ of equation (1.1). Then, the retrieval process generates the following generalisation functions Γ_t (the composition operator is denoted by $\circ$):

$\Gamma_0 =$ identity function (cost 0)

$\Gamma_1 = \texttt{lemonjuice} \rightsquigarrow \texttt{fruitjuice}$ (cost 0.7)

$\Gamma_2 = \texttt{escarole} \rightsquigarrow \texttt{greensalad}$ (cost 0.8)

$\Gamma_3 = \texttt{lemonjuice} \rightsquigarrow \texttt{seasoning}$ (cost 0.9)

$\Gamma_4 = \texttt{lemonjuice} \rightsquigarrow \texttt{fruitjuice} \circ \texttt{escarole} \rightsquigarrow \texttt{greensalad}$ (cost 1.5)

$\Gamma_5 = \texttt{lemonjuice} \rightsquigarrow \texttt{seasoning} \circ \texttt{escarole} \rightsquigarrow \texttt{greensalad}$ (cost 1.7)

So, the result of the retrieval process is $(\{\texttt{R}\}, \Gamma)$ with $\Gamma = \Gamma_5$ since $idx(\texttt{R}) \models_{\texttt{DK}} \Gamma_5(\texttt{Q})$ and $idx(\texttt{R}) \not\models_{\texttt{DK}} \Gamma_t(\texttt{Q})$ for $t < 5$.

1.2.2 Case adaptation

Let $\{\mathtt{R}_i\}_i$ be the set of retrieved cases. Given a retrieved recipe $\mathtt{R} \in \{\mathtt{R}_i\}_i$ and the query $\mathtt{Q}$, adaptation aims at pointing out modifications of $\mathtt{R}$ so that it answers the query $\mathtt{Q}$. Several adaptation processes have been implemented in TAAABLE. The first (§1.2.2.1) uses only the result of the retrieval: $\mathtt{R}$ and Γ. The second (§1.2.2.2) uses adaptation knowledge in the form of ingredient adaptation rules. These two adaptation processes just give substitutions of ingredient types, regardless of ingredient quantities. By contrast, the third adaptation process (§1.2.2.3) addresses the issue of ingredient quantities. Finally, Sect. 1.2.2.4 addresses the issue of textual adaptation of the preparation of $\mathtt{R}$.

1.2.2.1 Basic adaptation

The basic adaptation procedure uses only $\mathtt{R}$, Γ and $\mathtt{Q}$ and no other piece of knowledge. It is based on the following sequence of relations that are satisfied since $\mathtt{R} \in \{\mathtt{R}_i\}_i$ and $(\{\mathtt{R}_i\}_i, \Gamma)$ is the result of the retrieval process:

$$idx(\mathtt{R}) \models_{\mathtt{DK}} \Gamma(\mathtt{Q}) \overset{\Gamma}{\longleftarrow} \mathtt{Q} \tag{1.3}$$

This sequence of relations involves a matching between the recipe ingredients and the positive literals of $\mathtt{Q}$. For the current example, this matching is based on the following composition:

$$\begin{aligned} &\mathtt{lettuce} \rightsquigarrow \mathtt{greensalad} \circ \mathtt{vinegar} \rightsquigarrow \mathtt{seasoning} \\ \circ\ &\mathtt{greensalad} \rightsquigarrow \mathtt{escarole} \circ \mathtt{seasoning} \rightsquigarrow \mathtt{lemonjuice} \end{aligned} \tag{1.4}$$

The first line is related to $\models_{\mathtt{DK}}$. The second line is related to $\overset{\Gamma}{\longleftarrow}$, i.e., to $\overset{\Gamma^{-1}}{\longmapsto}$. This composition can be simplified (using the equation $a \rightsquigarrow b \circ b \rightsquigarrow c = a \rightsquigarrow c$ and licit permutations in the composition) into:

$$\mathtt{lettuce} \rightsquigarrow \mathtt{escarole} \circ \mathtt{vinegar} \rightsquigarrow \mathtt{lemonjuice} \tag{1.5}$$

which describes which substitutions can be made on the recipe $\mathtt{R}$ to answer the query $\mathtt{Q}$.

More generally, basic adaptation consists in building a composition of substitutions as in (1.4) and then to simplify it (when it is possible) to obtain a substitution that is applied on the retrieved recipe $\mathtt{R}$. Technically, the elements of the substitutions are pointed out in two steps:

1. Following $idx(\mathtt{R}) \models_{\mathtt{DK}} \Gamma(\mathtt{Q})$, generalisation substitutions are built. Indeed, it can be shown, for φ and χ, conjunctions of literals, that if $\varphi \models_{\mathtt{DK}} \chi$ then there exists a composition G of substitutions of the form $a \rightsquigarrow b$

where $a \Rightarrow b \in \texttt{DK}$ such that $G(\varphi)$ is equivalent to χ (given the domain knowledge DK).

2. Following $\Gamma(\texttt{Q}) \stackrel{\Gamma}{\longleftarrow} \texttt{Q}$, specialisation substitutions γ^{-1} are built: if $\Gamma = \gamma_q \circ \ldots \circ \gamma_1$ then $\Gamma^{-1} = \gamma_1^{-1} \circ \ldots \circ \gamma_q^{-1}$. A generalisation $\gamma = a \rightsquigarrow b$ will lead to the specialisation $\gamma^{-1} = b \rightsquigarrow a$. A generalisation $\gamma = \neg a \rightsquigarrow \top$ will lead to the specialisation $\gamma^{-1} = \top \rightsquigarrow \neg a$, i.e., to the removal of a in the recipe.

1.2.2.2 Rule-based adaptation

The TAAABLE adaptation engine can also take advantage of adaptation knowledge in the form of adaptation rules. Such a rule states that in a given context $\mathcal{C}$, some ingredients $\mathcal{F}$ can be replaced by other ingredients $\mathcal{B}$ ($\mathcal{C}$, $\mathcal{F}$ and $\mathcal{B}$ are the contexts, the "from part" and the "by part" of the adaptation rule). For example, let us consider the following piece of knowledge:

In a recipe with green salad,
vinegar can be replaced by lemon juice and salt.

This piece of knowledge can be represented by an adaptation rule with $\mathcal{C} = \texttt{greensalad}$, $\mathcal{F} = \texttt{vinegar}$ and $\mathcal{B} = \texttt{lemonjuice} \wedge \texttt{salt}$. Such an adaptation rule can be encoded by a substitution $\sigma = \mathcal{C} \wedge \mathcal{F} \rightsquigarrow \mathcal{C} \wedge \mathcal{B}$. In the example:

$$\texttt{greensalad} \wedge \texttt{vinegar} \rightsquigarrow \texttt{greensalad} \wedge \texttt{lemonjuice} \wedge \texttt{salt} \tag{1.6}$$

Let AK denote the adaptation knowledge, i.e., the finite set of substitutions σ representing adaptation rules. A cost associated to each $\sigma \in \texttt{AK}$, $\texttt{cost}(\sigma) > 0$, is also assumed to be known. Given AK, the domain knowledge DK, a query Q, and a retrieved recipe R, rule-based adaptation combines the use of adaptation rules and of the generalisation of the query according to DK. It aims at building a sequence of relations of the form

$$idx(\texttt{R}) \stackrel{\Sigma}{\longmapsto} \Sigma(idx(\texttt{R})) \models_{\texttt{DK}} \Lambda(\texttt{Q}) \stackrel{\Lambda}{\longleftarrow} \texttt{Q} \tag{1.7}$$

where Σ is a (possibly empty) composition of adaptation rules $\sigma \in \texttt{AK}$, Λ is a generalisation function (that may be different from the generalisation function Γ returned by the retrieval process) and $\texttt{cost}(\Sigma) + \texttt{cost}(\Lambda)$ is minimal.

The sequence (1.7) is called an *adaptation path*. The adaptation consists in following this path: first, the composition Σ of adaptation rules is applied on the recipe R, then generalisations corresponding to $\Sigma(idx(\texttt{R})) \models_{\texttt{DK}} \Lambda(\texttt{Q})$ are applied, and finally, specialisations corresponding to $\Lambda(\texttt{Q}) \stackrel{\Lambda}{\longleftarrow} \texttt{Q}$ are applied. It can be noted that the second and third steps correspond to the adaptation path (1.7) when Σ = identity function and $\Lambda = \Gamma$.

Therefore, the main algorithmic difficulty of rule-based adaptation is to build an adaptation path. Once again, the technique used is based on a

best-first search in a state space. For this search, a state is an ordered pair (Σ, Λ), the initial state corresponds to Σ = Λ = identity function, (Σ, Λ) is a final state if (1.7) is satisfied, the cost associated to a state (Σ, Λ) is $\texttt{cost}(\Sigma) + \texttt{cost}(\Lambda)$ and the successors of a state (Σ, Λ) are the states (Σ', Λ) and the states (Σ, Λ') such that

- The Σ' substitutions are such that $\Sigma' = \sigma \circ \Sigma$ with $\sigma \in \texttt{AK}$ and σ is applicable on $\Sigma(idx(\texttt{R}))$;
- The Λ' substitutions are such that $\Lambda' = \gamma \circ \Lambda$ with γ, a generalisation based on DK that is applicable on $\Lambda(\texttt{Q})$ (γ has either the form $a \rightsquigarrow b$ or the form $\neg a \rightsquigarrow \top$).

It can be noticed that the search space contains at least one final state: in particular (Σ, Λ) with Σ = identity function and $\Lambda = \Gamma$ satisfies (1.7), since (1.3) is satisfied (R being a retrieved case with a generalised query $\Gamma(\texttt{Q})$), thus the algorithm terminates. Moreover, this shows that rule-based adaptation amounts to basic adaptation when there is no available adaptation rule ($\texttt{AK} = \varnothing$).

For example, with $idx(\texttt{R})$ and Q defined by (1.1) and (1.2), with the domain knowledge with costs of Fig. 1.5, and with $\texttt{AK} = \{\sigma\}$, σ being the adaptation of (1.6) and $\texttt{cost}(\sigma) = 0.2$, then the adaptation gives $\Sigma = \sigma$, $\Lambda = \texttt{escarole} \rightsquigarrow \texttt{greensalad}$ with a cost $0.2 + 0.8 = 1$ which involves an adaptation of the recipe based on the substitution

$$\texttt{lettuce} \rightsquigarrow \texttt{escarole} \ \circ \ \texttt{vinegar} \rightsquigarrow \texttt{lemonjuice} \wedge \texttt{salt}$$

Taking into account specific adaptation knowledge. For some recipes R, there exist specific adaptation rules, applicable in the context of the recipes ($\mathcal{C} = \texttt{R}$). The set of the specific adaptation rules associated with R is denoted by $\texttt{AK}_\texttt{R}$. The provenance of these rules is detailed in Sect. 1.3.5. This occurs in particular when there are variants in the original text of the recipe; if butter is an ingredient of R that is mentioned to be replaceable by margarine, then $\sigma = \texttt{butter} \rightsquigarrow \texttt{margarine} \in \texttt{AK}_\texttt{R}$ and, since this variant is mentioned in the recipe, the cost of this rule is assumed to be 0.

Taking into account these specific adaptation rules consists simply in considering $\texttt{AK}_\texttt{R}$, where R is the retrieved recipe, in addition to the adaptation rules of AK: adaptation of R is based on $\texttt{AK} \cup \texttt{AK}_\texttt{R}$.

1.2.2.3 Adaptation of the ingredient quantities

The basic adaptation of Sect. 1.2.2.1 points out a composition of ingredient substitutions (like equation (1.5)) and works at the abstract representation level of the recipe index $idx(\texttt{R})$ of the retrieved recipe R. This section and the next one are interested in more concrete elements of R; here, adaptation deals

with the adaptation of quantities and reuse the result of the basic adaptation, i.e., the composition of ingredient substitutions $a_i \rightsquigarrow b_i$.

The ingredient quantities are adapted following the revision-based adaptation principle. This principle is detailed in [5]. Roughly said, a revision-based adaptation consists in modifying the source case minimally so that it is consistent with the query, while taking into account the domain knowledge. Such minimal modification can be performed by a belief revision operator and is frequently modelled thanks to a distance.

This adaptation of ingredient quantities is performed after the retrieval of a source recipe, and a first adaptation, as a conjunction of (Boolean) ingredient substitutions. Each substitution replaces an ingredient a from the source recipe with another ingredient b. So the quantity adaptation takes into account the following inputs:

- The formula $idx(\texttt{R})$ in propositional logic that represents the source recipe (see, e.g., equation (1.1)). Dish types and origins are ignored here. In this section, the following example, corresponding to the recipe "Baked apple pancake", is considered:

  $idx(\texttt{R}) = \texttt{flour} \wedge \texttt{milk} \wedge \texttt{granulatedsugar} \wedge \texttt{egg} \wedge \texttt{apple} \wedge \textit{nothing else}$

- The formula $\texttt{Q}$ in propositional logic that represents the query made by the user. In this section, the following example is considered:

  $$\texttt{Q} = \texttt{banana} \wedge \neg\texttt{chocolate}$$

- A composition of ingredient substitutions $a_i \rightsquigarrow b_i$. In the example of this section, there is only one substitution $a_1 \rightsquigarrow b_1$ with:

  $$a_1 = \texttt{apple} \qquad\qquad b_1 = \texttt{banana}$$

Representation formalism. The formalism used is based on numerical variables and linear constraints. Each variable corresponds to the quantity of an ingredient type. The list of ingredient types is reduced to those appearing in the source recipe, in the query, or in the substitutions. This list of ingredient types is closed by deduction w.r.t. domain knowledge: all the ingredient types that generalise one of the element of this list is also included. The following ingredients are considered in the example: `flour`, `cerealproduct`, `milk`, `dairyproduct`, `egg`, `granulatedsugar`, `chocolate`, `seasoning`, `apple`, `pomefruit`, `banana`, `fruit`, `ingredient`.

The ingredient amounts given in recipes are expressed in different units (grams, millilitres, teaspoons, pieces, ...). According to the unit used, the variables take their value in $\mathbb{R}_+$ (grams, millilitres) or $\mathbb{N}$ (spoons, pieces). To express constraints between the ingredient quantities, the quantities are converted into grams. To do this, we introduce, if needed, an additional vari-

able per ingredient that represents its amount in grams. The quantity of an ingredient can be represented by several variables, conversions between the values of these variables are given by domain knowledge.

In addition to ingredient quantities, nutritional values are also taken into account. The list of these nutritional values is given in the food pages (see Fig. 1.6). A variable is introduced for each nutritional value. The quantities that have a significant impact on the taste and appearance of the final recipe are the amount of sugars, fat and water. They are given more importance in the adaptation calculus than the other variables. The other quantities have, however, a dietary interest and we could consider these values for requests over specific diets.

Some nutritional variables cannot be converted into grams (for example, the energy contained in the ingredients is in calories), so we consider only one variable per nutritional value with the unit in which it is expressed (see Fig. 1.6).

Nutritional values

Nutritional value per 100 g (3.5 oz)	
Energy	52 kcal (220 kJ)
Carbohydrates	13.81 g
Sugars	10.39 g
Dietary fiber	2.4 g
Fat	0.17 g
Protein	0.26 g
Water	85.56 g
Vitamin A (equiv.)	3 µg (0%)
Thiamine (Vit. B1)	0.017 mg (1%)
Riboflavin (Vit. B2)	0.026 mg (2%)
Niacin (Vit. B3)	0.091 mg (1%)
Pantothenic acid (Vit. B5)	0.061 mg (1%)
Vitamin B6	0.041 mg (3%)
Folate (Vit. B9)	3 µg (1%)
Vitamin C	4.6 mg (8%)
Calcium	6 mg (1%)
Iron	0.12 mg (1%)
Magnesium	5 mg (1%)
Phosphorus	11 mg (2%)
Potassium	107 mg (2%)
Sodium	1 mg (0%)
Zinc	0.04 mg (0%)

Percentages are relative to US recommendations for adults.
Source: USDA Nutrient database
Corresponding ingredient: APPLES,RAW,WITH SKIN

Nutritional values

Nutritional value per 100 g (3.5 oz)	
Energy	89 kcal (370 kJ)
Carbohydrates	22.84 g
Sugars	12.23 g
Dietary fiber	2.6 g
Fat	0.33 g
Protein	1.09 g
Water	74.91 g
Vitamin A (equiv.)	3 µg (0%)
Thiamine (Vit. B1)	0.031 mg (2%)
Riboflavin (Vit. B2)	0.073 mg (5%)
Niacin (Vit. B3)	0.665 mg (4%)
Pantothenic acid (Vit. B5)	0.334 mg (7%)
Vitamin B6	0.367 mg (28%)
Folate (Vit. B9)	20 µg (5%)
Vitamin C	8.7 mg (15%)
Calcium	5 mg (1%)
Iron	0.26 mg (2%)
Magnesium	27 mg (7%)
Phosphorus	22 mg (3%)
Potassium	358 mg (8%)
Sodium	1 mg (0%)
Zinc	0.15 mg (2%)

Percentages are relative to US recommendations for adults.
Source: USDA Nutrient database
Corresponding ingredient: BANANAS,RAW

Fig. 1.6 Nutritional values extracted from WikiTaaable pages *Apple* and *Banana*. This data comes from a copyright-free database of the *United States Department of Agriculture* (USDA, http://www.ars.usda.gov/).

Other values could be taken into account for the quantity adaptation, for instance the bitterness of ingredients or values that express their taste strength. But we lack the knowledge to compute them.

The recipes are then represented with variables $x_1, \ldots, x_n$ that respectively take their values in value spaces $\mathcal{U}_1, \ldots, \mathcal{U}_n$ with either $\mathcal{U}_i = \mathbb{R}_+$ or $\mathcal{U}_i = \mathbb{N}$. So a recipe is represented by a value in $\mathcal{U} = \mathcal{U}_1 \times \ldots \times \mathcal{U}_n$. In the example, the same notation for the numerical variables as for the propositional variables is used, with units in underscore. For instance, the variable corresponding to the amount of apples in grams is written $\texttt{apple}_\texttt{g}$, the variable corresponding to the amount of apples in units (pieces) is written $\texttt{apple}_\texttt{u}$. As presented further, the cooking knowledge used can be expressed as linear constraints over variable values.

Representation of recipes. The recipe R is represented by a conjunction $clc(\texttt{R})$ of linear constraints. For each ingredient of the recipe, a linear constraint $(x = v)$ is added to $clc(\texttt{R})$, where x is the variable corresponding to this ingredient and v is its value in the recipe. For any ingredient type p not listed in the recipe, i.e. such that $idx(\texttt{R}) \models_{\texttt{DK}} \neg p$, the formula $(p_\texttt{g} = 0)$ is added to $clc(\texttt{R})$. For example, the recipe "Baked apple pancake" is represented by

$$\begin{aligned} clc(\texttt{R}) = {} & (\texttt{flour}_\texttt{cup} = 1) \wedge (\texttt{milk}_\texttt{cup} = 1) \wedge (\texttt{egg}_\texttt{u} = 4) \\ & \wedge (\texttt{granulatedsugar}_\texttt{cup} = 1) \wedge (\texttt{apple}_\texttt{u} = 2) \wedge (\texttt{banana}_\texttt{g} = 0) \end{aligned}$$

Representation of the domain knowledge. The linear constraints related to the domain knowledge used for this adaptation are denoted by $clc(\texttt{DK})$ and correspond to unit conversions, the food hierarchy given in the domain knowledge DK, and the nutritional data for each food.

Unit conversion. For every ingredient quantity expressed in two units, with a variable $\texttt{food}_\texttt{[unit]}$ that represents its amount in the unit [unit] and a variable $\texttt{food}_\texttt{g}$ that represents its amount in grams, the following formula is added to $clc(\texttt{DK})$:

$$(\texttt{food}_\texttt{g} = \alpha \cdot \texttt{food}_\texttt{[unit]})$$

where α is the mass in grams for 1 [unit] of food.

Remark 1. We use the word "conversion" with a broader meaning than in physics since we also perform the conversion between values expressed in heterogeneous units. For instance, 1 cup of flour weighs 250 g. For ingredients measured in pieces—like eggs, bananas and apples in the example—we consider the typical mass of a piece, these values are also taken from the USDA database (see Fig. 1.6). When the corresponding values are available,

qualifiers given to the ingredients are taken into account. For instance, the recipe "Baked Apple Pancake" contains large apples, that weigh 223 g each according to USDA data, a small apple only weighs 101 g according to the same dataset.

The following unit conversions are used in the example:

$(\texttt{apple}_\texttt{g} = 223 \cdot \texttt{apple}_\texttt{u})$	A (large) apple weighs 223 grams.
$(\texttt{flour}_\texttt{g} = 120 \cdot \texttt{flour}_\texttt{cup})$	A cup of flour weighs 120 grams.

Food hierarchy. The food hierarchy is expressed by linear constraints over variable values of foods related by a generalisation relation. The quantity of a general food in a recipe is equal to the sum of the quantities of the more specific foods used in the recipe.

So, if a food `GenFood` covers the specific foods $\texttt{food1}, \ldots, \texttt{food}k$, the following formula is added to $clc(\texttt{DK})$:

$$(\texttt{GenFood}_\texttt{g} = \texttt{food1}_\texttt{g} + \ldots + \ldots \texttt{food}k_\texttt{g})$$

Remark 2. By "specific foods", we mean ingredient types for which there are no more specific food in the ingredient list. In the example, the value of $\texttt{fruit}_\texttt{g}$ is given by the following equality:

$$\texttt{fruit}_\texttt{g} = \texttt{apple}_\texttt{g} + \texttt{banana}_\texttt{g}$$

The term $\texttt{pomefruit}_\texttt{g}$ does not appear in this sum since we already have $\texttt{pomefruit}_\texttt{g} = \texttt{apple}_\texttt{g}$ as `pomefruit` generalises `apple`: the mass of apples in $\texttt{fruit}_\texttt{g}$ must not be counted twice.

$$\text{We decided to take the equalities } \begin{cases} \texttt{fruit}_\texttt{g} = \texttt{apple}_\texttt{g} + \texttt{banana}_\texttt{g} \\ \texttt{pomefruit}_\texttt{g} = \texttt{apple}_\texttt{g} \end{cases}$$

$$\text{rather than } \begin{cases} \texttt{fruit}_\texttt{g} = \texttt{pomefruit}_\texttt{g} + \texttt{banana}_\texttt{g} \\ \texttt{pomefruit}_\texttt{g} = \texttt{apple}_\texttt{g} \end{cases}$$

to avoid the problems raised by multiple specialisation (recall that the hierarchy is not a tree). For instance, `broccoli` direct superclasses are `inflorescentvegetable` and `cabbage`,[5] so if we write the following equalities:

[5] `cabbage` is not a subclass of `inflorescentvegetable` (drumhead cabbages are not flowers), neither is `inflorescentvegetable` a subclass of `cabbage` (artichokes are inflorescent vegetables).

$$\begin{aligned}
\texttt{vegetable}_\texttt{g} &= \texttt{inflorescentvegetable}_\texttt{g} + \texttt{cabbage}_\texttt{g} \\
\texttt{inflorescentvegetable}_\texttt{g} &= \texttt{broccoli}_\texttt{g} \\
\texttt{cabbage}_\texttt{g} &= \texttt{broccoli}_\texttt{g}
\end{aligned}$$

broccoli mass would be counted twice in the vegetable mass. Instead of the first equality, we write $\texttt{vegetable}_\texttt{g} = \texttt{broccoli}_\texttt{g}$.

Remark 3. An additional problem arises for some recipes when the quantities of foods food1 and food2 are given in the recipe, although food1 is classified as more general than food2. This generates inconsistencies between the constraints. For example, the recipe "*Vegetables in crock pot*" contains 1 cup of broccoli (91 g) and 1 cabbage (1248 g), however, in the hierarchy, broccoli are classified as cabbage. We get the following inconsistent conjunction of constraints:

$$\begin{aligned}
&(\texttt{cabbage}_\texttt{g} = \texttt{broccoli}_\texttt{g}) \\
\wedge\ &(\texttt{cabbage}_\texttt{g} = 1248) \\
\wedge\ &(\texttt{broccoli}_\texttt{g} = 91)
\end{aligned}$$

This problem is solved by introducing an additional variable $\texttt{food1}'_\texttt{g}$ that takes the value given in the recipe for food1 and that is handled as a specific food:

$$\texttt{food1}_\texttt{g} = \texttt{food1}'_\texttt{g} + \texttt{food2}_\texttt{g}$$

if food2 is the only food generalised by food1.

The term $\texttt{food1}'_\texttt{g}$ is also included in the equalities that give the values of $\texttt{food}_\texttt{g}$ for any food that generalises food1.

So, for the recipe "*Vegetables in crock pot*", the previous equalities are replaced by the following ones:

$$\begin{aligned}
\texttt{cabbage}_\texttt{g} &= \texttt{cabbage}'_\texttt{g} + \texttt{broccoli}_\texttt{g} \\
\texttt{cabbage}'_\texttt{g} &= 1248 \\
\texttt{broccoli}_\texttt{g} &= 91
\end{aligned}$$

which are consistent ($\texttt{cabbage}_\texttt{g} = 1248 + 91 = 1339$).

Nutritional values. We have nutritional data for each food. For example, some sugars are contained in the ingredients of the recipe, approximately 0.1 g per gram of apple, 0.12 g per gram of banana, 1 g per gram of granulated sugar, 0.05 g per gram of milk, and 0 g per gram of flour and egg. These quantities add up to give the total amount of sugars in the recipe:

$$\texttt{sugar}_\texttt{g} = 0 \cdot \texttt{flour}_\texttt{g} + 0.05 \cdot \texttt{milk}_\texttt{g} + 0 \cdot \texttt{egg}_\texttt{g} + 1 \cdot \texttt{granulatedsugar}_\texttt{g} + 0.1 \cdot \texttt{apple}_\texttt{g} + 0.12 \cdot \texttt{banana}_\texttt{g}$$

Formally, for a nutritional value represented by the variable `nutVal`, let $\alpha^{\texttt{nutVal}}_{\texttt{food}}$ be the value of `nutVal` contained in 1 g of `food`, the following equality is added to $clc(\texttt{DK})$:

$$\texttt{nutVal} = \alpha^{\texttt{nutVal}}_{\texttt{food1}} \cdot \texttt{food1}_\texttt{g} + \ldots + \alpha^{\texttt{nutVal}}_{\texttt{food}k} \cdot \texttt{food}k_\texttt{g}$$

Like for the constraints that encode the food hierarchy, only the most specific foods are involved in these calculations ($\texttt{food1}, \ldots, \texttt{food}k$ are the specific ingredients of the recipe).

Representation of queries. Let $clc(\texttt{Q})$ be the representation of the query by a conjunction of linear constraints. The query is represented in `Q` by "Boolean" constraints, but we do not know how to express adequately all these constraints as linear constraints in the formalism with numerical variables used here.

A term $\neg\texttt{food}$ in `Q` which stands for the request by the user not to include the food `food` in the recipe, can be expressed by the constraint $\texttt{food}_\texttt{g} = 0$.

By contrast, it is less easy to translate a positive literal `food` of the query `Q`. Indeed, the constraint $\texttt{food}_\texttt{g} > 0$ is not allowed in the language, since strict constraints (involving non closed subsets of $\mathcal{U}$) are not allowed in the language.

To get interesting results, we use the ingredient substitutions obtained from the basic adaptation step. For each substitution $a \rightsquigarrow b$, the following formula is added to $clc(\texttt{DK})$:

$$s = a_\texttt{g} + b_\texttt{g}$$

where s is a new variable. And the following formula is added to $clc(\texttt{Q})$:

$$a_\texttt{g} = 0$$

Assume that we give more importance to the change in the value of s than to the change in the value of $a_\texttt{g}$ and $b_\texttt{g}$. Thus, with no other constraint interfering, the minimal change between a value $u \in \mathcal{U}$ that satisfies $a_\texttt{g} = x$ ($x \in \mathbb{R}$) and $b_\texttt{g} = 0$, and the set of values $v \in \mathcal{U}$ that satisfy $a_\texttt{g} = 0$ is reached for values satisfying $b_\texttt{g} = x$. The expected result of the adaptation results is a "compensating" effect of the adaptation.

Choice of a distance. As mentioned in paragraph "Representation formalism" the recipes are represented in the space $\mathcal{U} = \mathcal{U}_1 \times \ldots \times \mathcal{U}_n$ where each component $\mathcal{U}_i$ is the range of the variable x_i. We consider a distance defined for any $x, y \in \mathcal{U}$ with $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ by:

$$d(x,y) = \sum_{i=1}^{n} w_i |y_i - x_i|$$

where the coefficients w_i are to be chosen.

Some criteria guide the choice of the coefficients w_i:

- Only quantities of ingredients in grams are taken into account, so for any i such that x_i gives the amount of an ingredient in some other unit than grams, $w_i = 0$.
- The coefficient for general foods must be larger than the sums of the specific foods it generalises. Given a food `GenFood` that generalises the foods `food1`$, \ldots,$ `food`k, let x_{i0} = `GenFood`$_{\mathtt{g}}$ be that amount in grams of `GenFood` and $x_{i1}, \ldots, x_{ik}$ the amounts in grams of respectively `food1`$, \ldots,$ `food`k, then:

  $$w_{i0} \geqslant w_{i1} + \ldots + w_{ik}$$

  This condition ensures the "compensation" effect of the adaptation, it makes the substitution of the specific food `food`j by another `food`j' less important than the reduction of the amount of `GenFood`.

In practice, the coefficients of d are calculated thanks to the recipe base, w_i equals to the number of recipes using the food corresponding to x_i: $w_i = \mathcal{N}(x_i)$, with the notation used in Sect. 1.2.1.

Computation. The calculus of the adaptation of $clc(\mathtt{R})$ to solve $clc(\mathtt{Q})$ can be seen as an optimisation problem:

$$\begin{array}{ll} \text{find } y \text{ such that} & x \text{ and } y \text{ satisfy } clc(\mathtt{DK}) \\ & x \text{ satisfies } clc(\mathtt{R}) \\ & y \text{ satisfies } clc(\mathtt{Q}) \\ & d(x,y) \text{ is minimal} \end{array}$$

The constraints that define $clc(\mathtt{DK})$, $clc(\mathtt{R})$ and $clc(\mathtt{Q})$ are linear constraints but the function to be minimised, d, is not linear, so the minimisation of d under these constraints is not a linear optimisation problem. However an equivalent linear optimisation problem can be built. This new problem is defined in the space $\mathcal{U}^3 = \mathcal{U} \times \mathcal{U} \times \mathcal{U}$. Intuitively, for any $(x, y, z) \in \mathcal{U}^3$, x will be an element from $clc(\mathtt{R})$, y an element from $clc(\mathtt{Q})$ and $z = (z_1, \ldots, z_n)$ will be the difference per variable between values of x and y.

In addition to the linear constraint from $clc(\mathtt{R})$ applied on x, from $clc(\mathtt{Q})$ applied on y and $clc(\mathtt{DK})$ applied on both x and y, the following constraints are added for every $1 \leqslant k \leqslant n$

$$z_k \geqslant y_k - x_k \quad \text{and} \quad z_k \geqslant x_k - y_k \qquad (\text{i.e. } z_k \leqslant |y_k - x_k|)$$

The function to minimise is $f : (x, y, z) \mapsto \sum_{i=1}^{n} w_i z_i$, f is a linear function over $\mathcal{U}^3$ to $\mathbb{R}$, so the minimisation of f under the constraints stated above is a linear optimisation problem. It can be noted that as no other constraints are applied to the z_k than the ones stating that $z_k \leqslant |y_k - x_k|$, minimal values of f are reached for (x, y, z) such that $z_k = |y_k - x_k|$ and so $f(x, y, z) = d(x, y)$ when (x, y, z) is an optimum. This shows we have indeed an equivalent linear problem.

This reduction to a linear optimisation problem made it possible to use common linear optimisation tools to perform the quantity adaptation. As variables can either take real or integer values, the resulting problem is a mixed linear optimisation problem, this class of problem is NP-hard (P if all the values are reals) but the problem sizes encountered for recipe adaptation are small and manageable. In practice we use LP_Solve.[6]

1.2.2.4 Textual adaptation of preparations

Not all ingredients are cooked in the same way. Therefore, when an ingredient a is replaced with an ingredient b in a recipe, it may prove necessary to further adapt the preparation part. The preparation text is adapted by replacing the sequence of actions applied to a with a similar sequence of actions that is found in a different recipe where it is applied to b. Using an existing recipe guarantees that the actions used are a correct way of preparing ingredient b. Each different sequence of actions that is found to be applied to b in a recipe is called a "prototype" of b—that is, it defines one of the potentially many ways in which b can be prepared.

Consider the case of a user making the request shown in Fig. 1.2. No dessert recipe with fig and rice exists in the recipe base used by Taaable, but there is a recipe called "Glutinous rice with mangoes". The mango prototype used in this recipe is characterised by the set of actions {`chill`, `peel`, `slice`, `remove-pit`, `place`}. Taaable will suggest replacing mangoes with figs, which seems to be a satisfactory adaptation, except the need for peeling and pitting figs, that makes no sense.

In the recipe base, there are two recipes with figs. Recipe #1163 sports a {`halve`, `sprinkle-over`, `dot-with`, `cook`, `brown`, `place`} fig prototype, and #53 a more modest {`cut`, `combine`}. In order to simplify the processing and avoid discriminating near-synonyms, classes of similar actions can be grouped using an action hierarchy. This could have the effect of grouping, e.g. `peel` with `remove-pit`, `cut` with `slice`, and `cook` with `brown`.

To select the more appropriate fig prototype, the search space is organised as a concept lattice built using formal concept analysis [13] from the fig prototypes. The mango prototype is then merged into it as well. This approach is similar to techniques used in document retrieval by Carpineto and Ro-

[6] http://lpsolve.sourceforge.net/5.5/

mano [4], where a lattice is built according to keywords found in documents and a query is merged in it (see also [20, 21]). The lattice is built from a formal binary context, which maps the set of fig prototypes from recipes #53 and #1163 (the "objects") to the set of culinary actions (the "attributes"), indicating whether a given action occurs in a given prototype. The query, consisting in an additional mango prototype object from the retrieved recipe, is merged into this binary context. The resulting formal binary context is shown in Table 1.1, and the corresponding lattice in Fig. 1.7.

	cut	halve	remove	place	pour	dot	cook	chill	combine
fig_#1163		×		×	×	×	×		
fig_#53	×								×
mango	×		×	×				×	

Table 1.1 Formal binary context of figs and a mango.

All prototypes having actions in common will appear together in the extent of at least one concept. The "lower" this concept is in the lattice, the more attributes the prototypes have in common. A set of candidate fig prototypes is taken from the extent of all concepts "immediately above" the lowest concept with `mango` in its extent (actually called the object concept of `mango` [13]). Whenever there is more than one candidate, the one that minimises the distance between its attributes and the mango's attributes is selected. In the example, `fig_#53` is selected since, while both `fig_#53` and `fig_#1163` appear directly above `mango`, replacing `mango` with `fig_#1163` would require removing three actions and adding four, whereas replacing it with `fig_#53` only requires removing three actions.

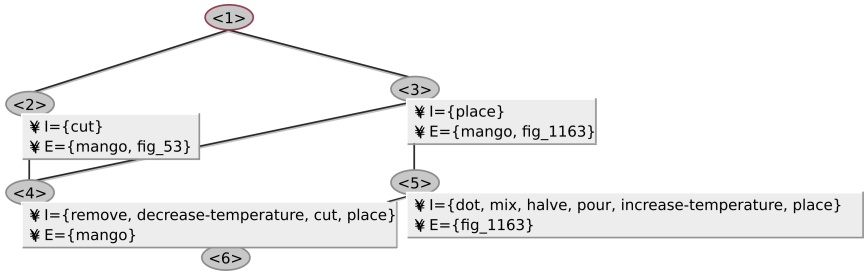

Fig. 1.7 Concept lattice corresponding to the binary context of Table 1.1. Each node corresponds to a formal concept which is composed of an extent, i.e. the objects which are instances of the concept, and an intent, i.e. the attributes composing the description of the concept. The extent is the maximal set of objects sharing the attributes in the intent, and reciprocally.

The process is then completed by replacing the textual parts of the retrieved recipe dealing with mangoes with the parts of recipe #53 dealing with figs:

> [...] Blend the sauce ingredients in a pot and heat until it just reaches the boiling point. Let cool. ~~Peel the mangoes, slice lengthwise and remove the pits.~~ Cut figs into wedges. Divide the rice mixture among 6 plates. [...]

1.3 Managing the TAAABLE knowledge containers

1.3.1 An ontology of the cooking domain

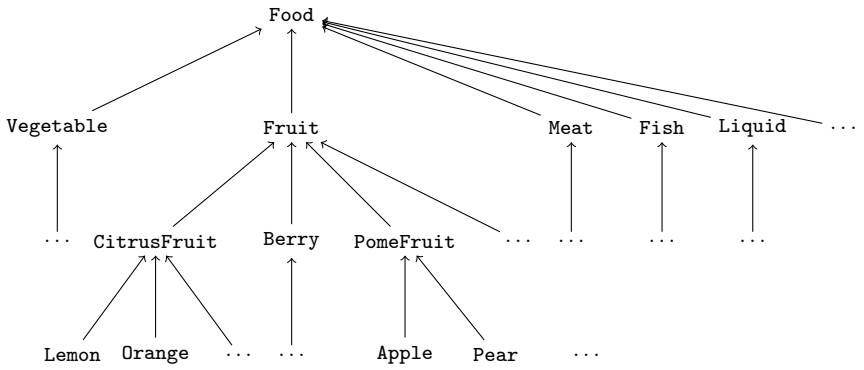

Fig. 1.8 A part of the food hierarchy.

The "cooking ontology" $\mathcal{O}$ defines the main classes and relations relevant to cooking. $\mathcal{O}$ is composed of 6 hierarchies:

- a *food* hierarchy, related to ingredients used in recipes, e.g. `Vegetable`, `Fruit`, `Berry`, `Meat`, etc. (a part of this hierarchy is given in Fig. 1.8)
- a *dish type* hierarchy, related to the types of dish, e.g. `PieDish`, `Salad`, `Soup`, `BakedGood`, etc.
- a *dish moment* hierarchy, related to the time for eating a dish, e.g. `Snack`, `Starter`, `Dessert`, etc.
- a *location* hierarchy, related to the origins of recipes, e.g. `FrenchLocation`, `AsianLocation`, `MediterraneanLocation`, etc.
- a *diet* hierarchy, related to food allowed or not for a specific diet, e.g `Vegetarian`, `NutFree`, etc.

- and an *action* hierarchy, related to cooking actions used for preparing ingredients, `Cut`, `Peel`, etc.

Given two classes `B` and `A` of this ontology, `A` is more specific than `B`, denoted by "`A` $\sqsubseteq$ `B`", iff the set of instances of `A` is included in the set of instances of `B`. For instance, "`CitrusFruit` (referring to citrus fruits) is more specific than `Fruit` (referring to fruit)" means that every citrus fruit is a fruit.

1.3.1.1 Ontology engineering

The ontology $\mathcal{O}$ was built in order to help the design of the retrieval and adaptation processes of the TAAABLE system. Therefore, the conceptual choice for the ontology development was strongly driven by the goal of this particular CBR system. The reuse of existing ontologies was carefully examined but no more considered as they did not cover what was intended to be reached in this project. So, after identification of the main classes during the elaboration of the cooking conceptual model, a fine-grained structuration of these classes has been carried out according to existing terminological resources and manual expertise, as it is done in object-oriented programming when the hierarchy of classes is designed.

Food hierarchy. The first version of `Food` hierarchy was built manually starting from several web resources such as the Cook's Thesaurus,[7] a cooking encyclopedia that covers thousands of ingredients, and Wikipedia. The main task was to select the set of relevant classes and to organise them according to the relation $\sqsubseteq$. At the same time, a first version of the terminological database was built in order to associate to each class (e.g., `BokChoy`) a linguistically preferred form (e.g., *bok choy*) as well as a set of lexical variants which can be a morphological variants or synonyms (e.g., *pak choy*, *Chinese cabbage*, *Chinese mustard cabbage*, etc.).

The first version of the food hierarchy and of the terminological database have then been enriched by adding iteratively new classes that occur in the recipe book but were missing in the hierarchy. For this, a semi-automatic process has been designed. This process retrieves the ingredient lines of the recipe book that cannot be linked to food classes by the annotation process. A manual expertise is required in order to determine the reason of these failures and to correct what is required. Three cases have been identified:

- the food class is missing in the hierarchy: the expert has to create it and to attach it to the most specific class(es) subsuming it.
- the food class exists in the hierarchy but the lexical form that appears in the recipe is not in the terminological database: the expert has to add this new lexical form as a lexical variant attached to the class.

[7] http://www.foodsubs.com

- the ingredient line contains an error (e.g. no food is mentioned, a mispelling error occurs, etc.): the expert has to correct the ingredient line.

This approach has been integrated in the wiki: when a new recipe is added, the annotation process directly gives feedback to the user, so that she can correct and complete the knowledge.

Finally, some information and knowledge about ingredients is automatically collected using `Freebase`,[8] a RDF database. This database is queried for collecting a short description, a link to the wikipedia page, some lexical variants in several languages, the compatibility/incompatibility with some diets, and possible images.

The food hierarchy is the largest hiearchy. Its current version contains about 3000 classes, organised around 9 levels. The terminological base related to food contains about 4200 English lexical forms, without taking into account singular/plurial variations.

Dish type and dish origin hierarchies. Starting from the organisation of dish types and dish origins in the Recipe Source database,[9] a list of general dish types and dish origins has been collected, and hierarchically organised following the subsumption relation.

The dish origin hierarchy contains 41 classes organised around 2 levels. The first level classifies the origin following their regions, such as `AfricanLocation`, `AsianLocation`, `EuropeanLocation`, etc. Each first level class is specialised, at the second level, by the country, origin of the dishes. For example, `FrenchLocation`, `GermanLocation`, `SpanishLocation`, etc. are subclasses of `EuropeanLocation`.

The dish type hierarchy, containing 69 classes, is organised around 3 levels. At the first level, there are classes like `BakedGood`, `Burger`, `Dessert`, `MainDish`, etc. The second and third levels introduce, if necessary, more specialized classes. For example, `BakedGood` is specialised into `Bagel`, `Biscuit`, `Bread`, `Muffin`, `Cookie`, etc. However, these classes are no more deeply detailed even if more specific categories exist in Recipe Source.

Dish moment and diet hierarchies These two hierarchies have only 1 level, each concept being directly under the root (i.e. the most general concept) of the hierarchy. There are currently 8 dish roles (e.g. `Snack`, `Starter`, etc.) and 7 types of diet (e.g. `Vegetarian`, `NutFree`, etc.) in $\mathcal{O}$.

Action hierarchy. The Action hierarchy was obtained by organising a list of verbs automatically extracted in recipe preparation texts. The ac-

[8] http://www.freebase.com

[9] http://www.recipesource.com

tion hierarchy contains 449 classes organised around 5 levels. Each class is described by syntactic and semantic properties, which make the automatic case acquisition process described hereafter possible. For example, `ToCut` is a subclass of `ToSplit` and is described by the linguistic property `isADirectTransitiveVerb`, meaning that a direct object has to be searched in the text by the preparation annotation process. `ToSplit` is also described by the functional property `isAnUnionVerb`, meaning that this action produces more than one output.

1.3.1.2 Nutritional data and weight equivalence acquisition

Using the USDA Nutrient database,[10] food classes are linked to their nutritional values (sugar, fat, protein, vitamins, etc.) and some weight conversion equivalences, two types of knowledge required for the adaptation of ingredient quantities (cf. Sect. 1.2.2.3). For linking nutritional values and weight conversions to their corresponding ontology concepts, a semi-automatic alignment was processed. This alignment makes the link between the food label in the USDA Nutrient database and its label in the wiki as a category. As the mapping is incomplete, some of the ingredients in the wiki do not possess such information.

Fig. 1.6 shows an example of nutritional data related to apples and bananas.

1.3.2 Adaptation knowledge

Taaable uses a particular form of adaptation knowledge (AK): adaptation rules about the substitution of some ingredients by others (e.g. in *"My Strawberry Pie"* recipe, `Strawberry` could be replaced with `Raspberry`). Formally, an adaptation knowledge unit is a 4-tuple $(context, replace, with, provenance)$, where:

- *context* represents the recipe or the class of recipes on which the substitution can be applied. An AK unit is specific if its *context* is a single recipe and generic if its *context* is a class of recipes (a specific type of dish, for example).
- *replace* and *with* are respectively the set of ingredients that must be replaced and the set of replacing ingredients.
- *provenance* is the source the AK unit comes from. Currently, four sources are identified:
  1. Taaable, when AK results from a proposition of adaptation given by the reasoning process of Taaable.

[10] http://www.nal.usda.gov

2. *AK extractor* (resp. *Generic AK extractor*), when AK results from the specific (resp. generic) knowledge discovery system integrated in WikiTaaable (see Sect. 1.3.5.2).
3. *user*, when AK is given by a user editing the wiki, as it is usually done in cooking web site, when users add comments about ingredient substitution in a recipe. See, for example, http://en.wikibooks.org/wiki/Cookbook:substitutions.
4. *recipe*, when the AK is directly given by the original recipe when a choice between ingredients is mentioned (e.g. "100g butter or margarine"). This particular substitutions are taken into account by a wiki bot which runs through the wiki for automatically extracting them.

According to this definition, (*"My Strawberry Pie"*, *Strawberry*, *Raspberry*, Taaable), is an AK unit obtained from Taaable, meaning that strawberries can be replaced with raspberries in the *"My Strawberry Pie"* recipe. In WikiTaaable, each substitution is encoded as a wiki page like the one given in Fig. 1.11. A semantic query is used to feed automatically the *Substitutions* section of a recipe page, as visible in Fig. 1.9.

1.3.3 WikiTaaable*: a semantic wiki for* Taaable

WikiTaaable[11] is a semantic wiki which is used in order to represent, edit and maintain knowledge used by Taaable [7]. To develop WikiTaaable, we relied on an existing tool: Semantic MediaWiki [17].[12] Semantic MediaWiki is an extension of MediaWiki (a well-known wiki engine, used among others by Wikipedia) enabling users to embed semantic in their wiki pages. We decided to use a wiki to benefit from the online and collaborative edition facilities it provides. Semantic MediaWiki was then a suitable solution to enable the introduction of knowledge units in the wiki.

Using WikiTaaable, users can browse, query, and edit the knowledge base through a user-friendly interface. The CBR engine is connected to WikiTaaable and uses the knowledge base during reasoning process. WikiTaaable embeds bots (small programs) which are in charge of performing automated tasks such as annotation of pages, tests or maintenance operations. Finally, additional interfaces are implemented within WikiTaaable so that users have only one tool to master when they manage knowledge. In summary, WikiTaaable acts as a collaborative and shared space between humans and agents.

WikiTaaable is composed of 4 main components:

Semantic MediaWiki. Semantic MediaWiki is the base of the architecture of WikiTaaable. Users access the system through the web interface. The

[11] http://wikitaaable.loria.fr/

[12] http://semantic-mediawiki.org

CBR engine and bots are connected to the wiki through a set of predefined semantic queries. Semantic MediaWiki encodes the knowledge base as a set of semantic wiki pages. The indexed recipes, the ontology classes, and the adaptation knowledge are encoded in semantic wiki pages. See Fig. 1.9, Fig. 1.10, and Fig. 1.11 for examples of wiki pages encoding respectively a recipe, an ontology class, and an adaptation.

Import bots. Scripts import TAAABLE knowledge base, and especially the recipes provided by the CCC into Semantic Media Wiki. These bots are only run once to bootstrap the wiki by creating a minimal set of knowledge units and recipes.

Recipe Annotation Bot. The recipe annotation bot parses the recipe pages, extracts ingredient information, and updates recipe pages with semantic annotation and categorisation of recipes. The parsing and update of recipes is done using the mediawiki API, accessing the knowledge base is done using predefined semantic queries. This bot is triggered each time a new recipe is added into the wiki in order to build its semantic annotation, and each time a new ingredient is added into the wiki in order to fill the missing annotations in.

Knowledge Acquisition Interfaces. WIKITAAABLE implements several knowledge acquisition interfaces. See Sect. 1.3.5.2 for illustration and details.

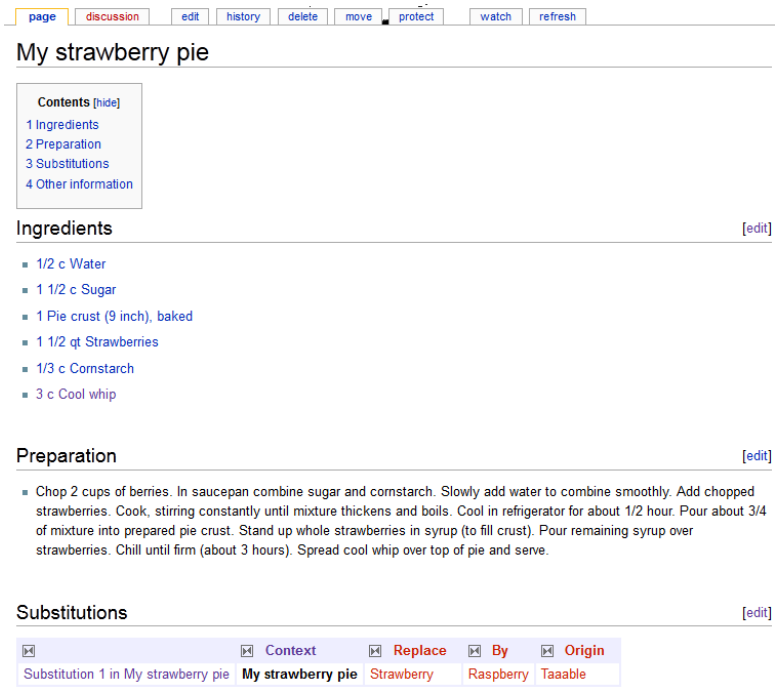

Fig. 1.9 An example of a WIKITAAABLE recipe: "My strawberry pie".

Fig. 1.10 An example of an ontology class: the `Berry` food class.

Fig. 1.11 An example of an adaptation rule page where `Strawberry` is replaced with `Raspberry` in the "My Strawberry Pie" recipe.

1.3.4 Case acquisition from texts

1.3.4.1 Ingredient annotation

The case base engine requires a formal representation of a recipe. The annotation process aims at formally encoding using semantic wiki properties and classes the content of a recipe as well as meta-properties like, for example, its origin and the type of dish the recipe will produce. This process is in between *controlled indexing* [16] where terms come from a predefined terminology and *semantic annotation* [27] where terms (named entities, sequences of words) are explicitly associated with the respective and most specific classes in the ontology. The result of the annotation of the ingredient part of a recipe is a set of food classes linked each to a recipe with the `hasIngredient` semantic wiki property. For that, the list of ingredients is parsed. First, each ingredient entry in the recipe is parsed and split into the following 4-tuple `(<quantity>,<unit>,<foodcomponent>,<modifiers>)`. For example, the entry "`1/3 cup milk`" is parsed as `(1/3,cup,milk,_)`. The terminological database guides the parsing process: the lexical variants are used conjointly to regular expressions for searching in the text an instance of a class of $\mathcal{O}$. For example, as "*pak choy*" is a lexical form associated to the food class `BokChoy` in the food hierarchy, the entry "*1 kg sliced pak choi*" is parsed as `(1, kg, BokChoi, sliced)` and the recipe containing this ingredient line is linked to the `BokChoy` food class using the `hasIngredient` property.

For example, the recipe entitled "*My srawberry pie*", illustrated in Fig. 1.9, is indexed by the conjunction of the ingredients `Water`, `Sugar`, `PiePastry`, `Strawberry`, `Cornstarch`, and `CoolWhip`.

1.3.4.2 Annotation of the dish types and dish origins

Another annotation process indexes recipes w.r.t. its origin (e.g., `AsianLocation`) and w.r.t. the type(s) of dish produced (e.g., `MainDish`, `Dessert`). As there is no indication about this in the recipe book, Recipe Source is used again in order to build a corpus where recipes are assigned to their origin and dish type(s). The process to determine the origin and the dish type of a recipe is based on 3 steps. For a given recipe:

- If there exists a recipe in Recipe Source with the same title, then the origin and the dish type(s) of the recipe in Recipe Source are assigned to the Recipe Book recipe;
- If the title of the recipe (e.g., "*Chinese Vegetable Soup*") contains keywords corresponding to subclasses of `DishType` (e.g., `soup`) or `DishOrigin` (e.g., "*Chinese*") then these origin and dish type(s) are assigned to the recipe;

- A set of association rules has also been extracted from the Recipe Source corpora, using the data-mining toolkit Coron [26]. According to exact associations rules (with 100% confidence) of the form <set of ingredients> $\longrightarrow$ <origin or dish type> (e.g., vanilla bean, banana, chocolate $\longrightarrow$ dessert), assignations of origin and dish type(s) can be done as follows: if part of the recipe matches the left-hand side of the rule, then the origin and/or the dish type(s) in the right-hand side is assigned to the recipe.

For example, the recipe entitled "My strawberry pie" is indexed by some ingredients, as presented below, but also by the following dish types: `Dessert` and `PieDish`.

As the semantic representation of a recipe is stored into a wiki, users are involved for correcting manually the annotation. Users may enter new ingredients, new lexical variants or also choose the dish types, the dish origins, the dish moments associated to recipes.

1.3.4.3 Preparation annotation

To make possible the textual adaptation task described in Sect. 1.2.2.4, a more complete formal case representation, including the way the ingredients are prepared, is required. Because it consists, in a nutshell, in combining a set of ingredients in specific ways until an edible dish is obtained, it is natural to represent a recipe as a tree, as shown in Fig. 1.12. Each node represents the state of a food component (an ingredient or a mix of ingredients) at a given time, and each edge $\langle a, b \rangle$ represents an action applied on a food component a that yields a new food component b. A set of edges sharing the same head (b) represents a "combining action", i.e. an action applied to many food components at once that yield out one new food component.

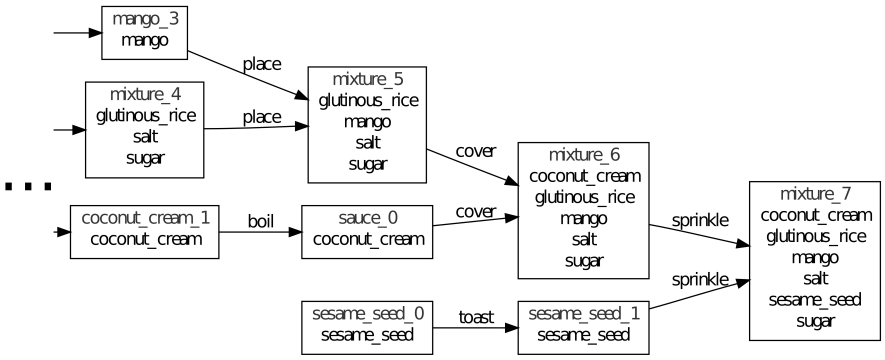

Fig. 1.12 Excerpt of the formal representation of "Glutinous rice with mangoes".

This representation is built iteratively, verb by verb, following the order of the text. Some sentences are difficult to analyse, and the partially built representation can be helpful. For example, in a sentence such as "Peel the mangoes, slice [the mangoes] lengthwise and remove the pits [from the mangoes]", it is easy for humans to understand that mangoes are being sliced and pitted, but some heuristics are needed for making this understandable to a computer. Each action is assigned an arity, making it possible to detect the absence of an argument. Whenever this happens, it is assumed that the missing argument corresponds to the last node that was added to the tree. This is one way of dealing with anaphoras, i.e. the phenomenon wherein a different word–or no word at all as it might be–is used to represent an object.

Other types of anaphora appear as well. Still in "Glutinous rice with mangoes", the expression "seasonings ingredients" is found, referring to some set of food components. The ingredient hierarchy is used to find all the nodes of the tree that fit under the "seasonings" category. A phrase such as "cover with sauce" is trickier because there is no obvious clue in the text or in the ontology about which food component this "sauce" may be. We built, from the analysis of thousands of recipes, "target sets" of ingredients that usually appear in the food components being referred to by word such as "sauce" or, say, "batter". Because a quantity of recipes include coconut cream-based sauces (and none contain, say, rice-based sauces), it makes sense to assume that, if a food component in a given recipe contains coconut cream, this is the one indicated by the use of the word "sauce".

Once the tree representation is built, it is straightforward to identify the sequence of actions being applied to a given ingredient in a recipe, in order to replace it as described in Sect. 1.2.2.4. Because we defined the prototype of an ingredient as the sequence of actions applied to this ingredient, up to and including the first combining action, in practice, only the leftmost part of the tree is replaced or used as a replacement. This may mean missing relevant actions, but also getting rid of actions that, while being applicable to certain mixtures containing an ingredient, may not make sense when applied directly to this ingredient. In the mango rice, for example, the *chill, peel, slice lengthwise, remove pits,* and *place on top* actions are kept, whereas *cover with, sprinkle with,* and *serve* are lost. It would seem that those last three actions are indeed too generic to be relevant in the present application. As an added benefit, because language processing errors accumulate, the leftmost parts of the tree, corresponding to the beginning of the recipe, are built with a higher reliability than the rightmost parts at this time.

1.3.5 Adaptation knowledge acquisition (AKA)

This section presents the various AKA strategies implemented in Taaable. When Taaable adapts a recipe to match a user query, it produces an ingre-

dient substitution. If users are satisfied with the adapted recipe, e.g. if they click on the "OK" button to validate the adaptation, the ingredient substitution is stored in WikiTaaable as an AK unit for future reuse. This AK acquisition strategy is straightforward but proves to be efficient.

If users are not satisfied with the adapted recipe, they can click on the "NOT OK" button. A process aiming at repairing the failed adaptation is then triggered. If this process is successful, additional AK units are acquired. This is what we call failure-driven AKA. This process is described hereafter.

We have also implemented other mechanisms to support the process of acquiring AK independently of a specific query. For that, we have tuned well-known knowledge discovery (KD) techniques. These mechanisms are described in the second part of this section.

1.3.5.1 Failure-driven adaptation knowledge acquisition

The Taaable system has been extended in 2009 to support online AKA [3]. The knowledge acquisition process complies with the FIKA principles of knowledge acquisition in case-based reasoning [6], in which the knowledge is acquired:

- **online**: in a particular problem-solving context,
- **interactive**: knowledge is acquired through interaction with users, and
- **opportunistic**: knowledge acquisition is triggered in response to a reasoning failure, i.e. when the user is not satisfied with the proposed solution.

Two strategies have been implemented to repair an adaptation failure, which differ by the source of AK. In the first strategy, AK is acquired only from the user and from the domain knowledge. In the second strategy, a KD step is added to the knowledge acquisition process. When the KD process, called CabamakA [2, 8], is triggered, it turns the case base into an additional source of AK.

The different steps involved in the overall AKA process are summarised in Fig. 1.13. The knowledge acquisition process starts at the solution test phase of the CBR cycle, when the user is not satisfied with the proposed solution. Repairing a failed adaptation is a two-step process. First, an explanation of the failure is identified through interactions with the user and a predefined explanation pattern is selected and instantiated. Then, the user is asked to choose among as set of repair strategies that are associated to this failure explanation pattern. This approach is inspired from CHEF's critic-based adaptation [14]. CHEF's approach to adaptation is to run a simulation and use a causal model to detect and explain potential problems in the generated solution. To solve these problems, CHEF makes use of a set of critics. A critic identifies a set of potential problems that can occur in the adapted solution and associates to them a repair strategy.

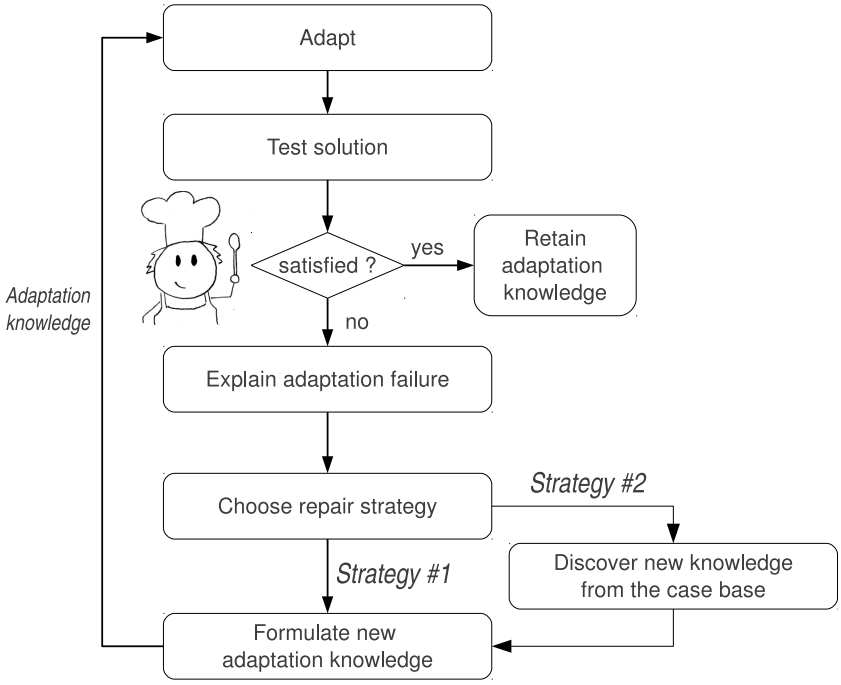

Fig. 1.13 The different steps of the interactive knowledge acquisition process.

The knowledge discovery process is optionally triggered when a repair strategy has been chosen. Its goal is to search the case base for information needed to instantiate this repair strategy, thereby implementing a case-based substitution approach in the spirit of DIAL's *adaptation=transformation+memory search* approach [18]. In the DIAL system, rule-based adaptation is performed by selecting an abstract rule and by searching for information needed to instantiate the rule. In our approach, a repair strategy defines a set of constraints on the adaptation rule to apply. These constraints are used to restrict both the training set and the hypothesis space of the learning process. CABAMAKA extracts from the case base a set of adaptation rules that satisfy these constraints. AK is dynamically generated from the case base and transferred to the adaptation knowledge base. The KD process is designed to only generate adaptation rules that are useful for solving a particular adaptation problem. Since the extracted adaptation rules are in a format directly usable by the system, the system can use them to re-run the adaptation step and propose a new solution to the user. The validation step is performed online by the system user, which is presented with the repaired solution together with the AK that was used to generate the solution.

Once the user is satisfied with the proposed solution, the discovered AK is retained in the AK base for future reuse.

In the remaining of this section, the two AKA strategies are illustrated. In the solution test phase, the retrieved recipe $\texttt{Sol}(\texttt{srce})$ is presented to the user together with the adaptation path $\texttt{AP}$ that was used to generate the candidate solution which is denoted $\widetilde{\texttt{Sol}}(\texttt{tgt})$. In this example, the recipe *Apple Pancakes from the Townships* is presented to the user in the solution test phase, together with the adaptation path $\texttt{AP} = \sigma$ (with $\sigma = \texttt{apple} \rightsquigarrow \texttt{pear}$), that was used to generate the solution $\widetilde{\texttt{Sol}}(\texttt{tgt})$ (Fig. 1.14).

In the failure explanation step, the user is encouraged to formulate an explanation of the adaptation failure. The failure explanation step is achieved in three substeps:

- *Substitution Selection.* The user selects a substitution $\sigma = \texttt{A} \rightsquigarrow \texttt{B}$ of $\texttt{AP}$ which is problematic, where $\texttt{A}$ and $\texttt{B}$ are conjunctions of literals. In the example, the user selects the substitution $\sigma = \texttt{apple} \rightsquigarrow \texttt{pear}$, so $\texttt{A} = \texttt{apple}$ and $\texttt{B} = \texttt{pear}$.
- *Explanation pattern selection.* The user selects an explanation pattern. Each explanation pattern explains the failure of a single substitution σ of the adaptation step, in which the solution $\texttt{Sol}(\texttt{pb})$ of a problem $\texttt{pb}$ is transformed in a solution $\sigma(\texttt{Sol}(\texttt{pb}))$. So far, three explanation patterns have been considered in the TAAABLE system: (1) an ingredient $\texttt{x}$ of $\texttt{B}$ requires an ingredient $\texttt{y}$ which is not in $\sigma(\texttt{Sol}(\texttt{pb}))$, (2) an ingredient $\texttt{x}$ of $\texttt{Sol}(\texttt{pb})$ requires an ingredient $\texttt{y}$ of $\texttt{A}$ which has just been removed, and (3) an ingredient $\texttt{x}$ of $\texttt{B}$ is not compatible with an ingredient $\texttt{y}$ of $\sigma(\texttt{Sol}(\texttt{pb}))$. Each one expresses a dependence between ingredients that was violated in the proposed recipe $\sigma(\texttt{Sol}(\texttt{pb}))$. In this example, the explanation pattern selected by the user expresses the fact that an ingredient x was added to the recipe after applying the substitution σ is incompatible with one of the ingredients of the recipe $\sigma(\texttt{Sol}(\texttt{pb}))$.

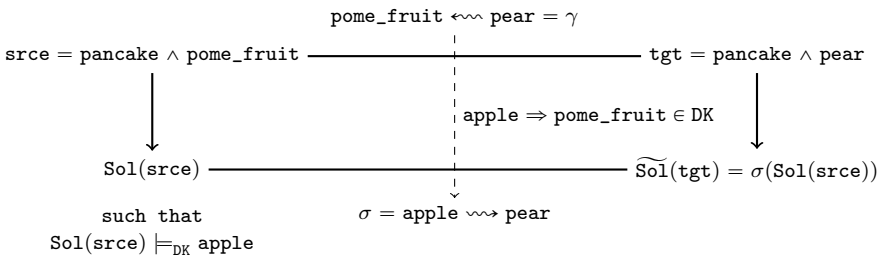

Fig. 1.14 An adaptation proposed by the TAAABLE system to answer the query "I want a pear pancake". A solution $\widetilde{\texttt{Sol}}(\texttt{tgt})$ for the target problem $\texttt{tgt}$ is constructed from the representation $\texttt{Sol}(\texttt{srce})$ of the retrieved recipe *Apple pancakes from the townships*. The substitution $\sigma = \texttt{apple} \rightsquigarrow \texttt{pear}$ is automatically generated from the substitution by generalisation $\gamma = \texttt{apple} \rightsquigarrow \texttt{pome\_fruit}$ using the axiom $\texttt{apple} \Rightarrow \texttt{pome\_fruit}$ of $\texttt{DK}$.

Failure Explanation	*Associated Repair Strategies*
An ingredient x of `B` is incompatible with an ingredient y of the adapted recipe.	–if `tgt` $\not\models_{\texttt{DK}} x$, remove x –if `tgt` $\not\models_{\texttt{DK}} x$, find a substitute for x –if `tgt` $\not\models_{\texttt{DK}} y$, remove y –if `tgt` $\not\models_{\texttt{DK}} y$, find a substitute for y
$x \leftarrow$ `pear` $y \leftarrow$ `cinnamon` (`pear` is incompatible with `cinnamon` in the recipe "*Apple pancakes from the townships*")	–remove `cinnamon` –find a substitute for `cinnamon`

Table 1.2 An example of failure explanation pattern and the associated repair strategies that are used to repair an ingredient substitution rule $\sigma = \texttt{A} \rightsquigarrow \texttt{B}$ in the cooking domain.

- *Explanation Pattern Instantiation.* The user selects the ingredients x and y in a list of propositions. In the example, the user selects the ingredients $x =$ `pear` and $y =$ `cinnamon`, in order to express the fact that keeping the cinnamon in the retrieved recipe is not satisfactory when pears are added.

To each explanation pattern is associated a set of repair strategies. A repair strategy is selected by the user among the ones that are applicable[13]. In the example, four repair strategies correspond to the selected explanation pattern but only the last two ones are applicable since `tgt` $\models_{\texttt{DK}}$ `pear` (Table 1.2). The two repair strategies proposed to the user are:

Strategy #1: remove `cinnamon`
Strategy #2: find a substitute for `cinnamon`

Two types of repair strategies are considered: the strategies that consist in adding or removing an ingredient, and the strategies that consist in finding a substitute for an ingredient.

Strategies that consist in adding or removing an ingredient are easy to instantiate. To add (resp. remove) an ingredient, the ingredient has to be selected on a list for addition (resp. removal). A repaired substitution σ' is generated from the substitution $\sigma = \texttt{A} \rightsquigarrow \texttt{B}$, in which an ingredient has been added or removed. In the example, the substitution σ' generated to repair σ in Strategy #1 is:

$$\sigma' = \texttt{apple} \wedge \texttt{cinnamon} \rightsquigarrow \texttt{pear}$$

[13] It can be noticed that at least one repair strategy is always applicable for a selected explanation pattern. Indeed, if the explanation pattern corresponds to a dependence of the form "x requires y", then either `tgt` $\models_{\texttt{DK}} y$ or `tgt` $\not\models_{\texttt{DK}} y$ holds. If the explanation pattern corresponds to a dependence of the form "x and y are incompatible", then `tgt` $\models_{\texttt{DK}} x$ and `tgt` $\models_{\texttt{DK}} y$ holding simultaneously would mean that `tgt` contains two incompatible ingredients.

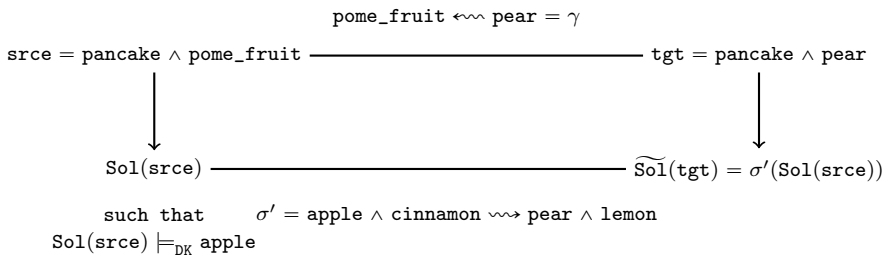

Fig. 1.15 A repaired adaptation proposed by the TAAABLE system to answer the query "I want a pear pancake". In this new adaptation, apples and cinnamon are substituted in the retrieved recipe by pear and lemon.

To instantiate the strategies that consist in finding a substitute for an ingredient, the CABAMAKA KD process is run in an opportunistic manner. Its goal is to learn a substitution σ' from the comparison of two recipe sets. In the example, the substitution σ' is of the form:

$$\sigma' = \texttt{apple} \wedge \texttt{cinnamon} \wedge \mathit{something}_1 \rightsquigarrow \texttt{pear} \wedge \mathit{something}_2$$

in which $\mathit{something}_1$ and $\mathit{something}_2$ are conjunctions of literals that need to be determined.

The form of the σ' substitution is used to tune the KD process. In the data preparation step, σ' is used to restrict the training set. The result set is also filtered in order to retain only the adaptation rules that have the form of σ' and that are applicable to solve the adaptation problem at hand [2].

In the example, the user selects the substitution:

$$\sigma' = \texttt{apple} \wedge \texttt{cinnamon} \rightsquigarrow \texttt{pear} \wedge \texttt{lemon}$$

The substitution σ' is used to re-run the adaptation step and generate a new solution $\widetilde{\texttt{Sol}}(\texttt{tgt})$. The repaired adaptation is summarised by Fig. 1.15.

During the solution test phase, the retrieved recipe $\texttt{Sol}(\texttt{srce})$ is presented to the user together with the new adaptation path $\texttt{AP}' = \sigma'$ that is applied to $\texttt{Sol}(\texttt{srce})$ to produce the solution $\widetilde{\texttt{Sol}}(\texttt{tgt})$.

1.3.5.2 Knowledge discovery for adaptation knowledge acquisition

Two interactive and opportunistic online systems have also been implemented to acquire AK independently of any specific query. These systems are also based on KD processes. In AK EXTRACTOR and GENERIC AK EXTRACTOR systems, users query the system, and validate some extraction results, eventually with some corrections, as AK unit. Therefore, users validate knowledge on the fly, in context (i.e. for a specific recipe), which is more convenient than

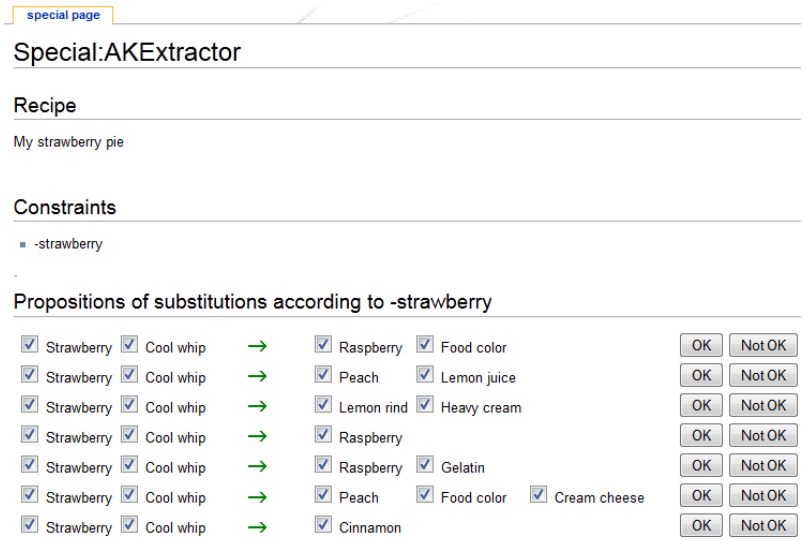

Fig. 1.16 Validation interface of the AK Extractor system for the query: adapt the "My Strawberry Pie" without strawberry.

dealing with a huge amount of candidate knowledge (information units) out of any context (i.e. for a KD process applied to a large set of recipes).

AK Extractor was built in order to acquire specific AK units, i.e. AK units that applies only to one recipe. AK Extractor takes into account compatibility and incompatibly of ingredients, overcoming the weakness of the adaptation process of Taaable. AK Extractor is based on a comparison of ingredients between the recipe to adapt, and a set of similar recipes. The system selects first recipes which have a minimal number of ingredients in common and a minimal number of different ingredients with the recipe that has to be adapted. Closed itemsets (CIs) [13] are extracted from variations of ingredients between recipes, starting from variations between each selected recipe of the previous step and the recipe to adapt. The CIs are then filtered and ranked with specific rules (which are detailed in [11]). The system displays the propositions for ingredient substitutions coming from the first five better ranked CIs. Fig. 1.16 shows the validation interface of the AK Extractor system coming from user constraints: "adapt the recipe *"My Strawberry Pie"* without strawberry". The first proposition of substitution means that `Strawberry` and `CoolWhip` could be substituted with `Raspberry` and `FoodColor`. Finally, user may give feedback for (fully or partially) validating or not some of these substitution propositions.

Generic AK Extractor was built in order to acquire generic AK units. Generic AK units provide rules that could be applied in a larger number of situations, because the context of an AK unit will be a set of recipes (e.g. for cakes), instead being only one recipe. Generic AK Extractor de-

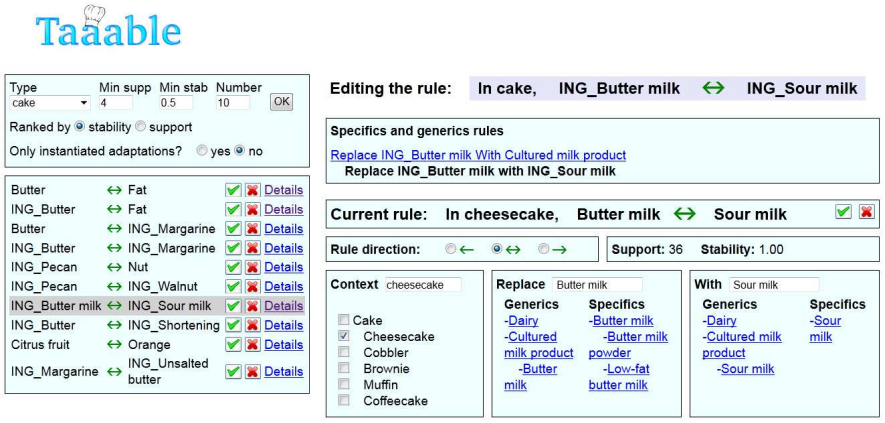

Fig. 1.17 Interface of the GENERIC AK EXTRACTOR system.

fends an approach based on CIs for extracting generic substitutions (generic AK units) starting from specific ones (specific AK units) [12]. Thanks to a dedicated interface, users can trigger the extraction of CIs from the specific AK units applicable to a set of recipes. CIs extracted are then filtered and ranked with specific rules (based on two measures commonly used in KD processes based on CIs: support and stability) before being presented to the user. The system displays the propositions of ingredient substitutions. The user can validate or repair a proposition of AK in order to generalise or specialise the substituted and substituting ingredients, change the direction of the rule and/or the class of recipes for which the substitution held. For example, in Fig. 1.17, the system is queried in order to acquire adaptation rules for cake dishes. 10 rules are proposed by the system (ranked by stability with a minimal support of 4 and a minimal stability of 10). Before validating a proposition of adaptation rule, the user can change the dish type on which the rule applies, as well as ingredients involved in the rule.

For the two systems, once validated, the adaptation rule is stored as an AK unit in WIKITAAABLE in order to be used by TAAABLE for the future adaptations.

1.4 Conclusion and ongoing work

In this chapter, the TAAABLE Case-Based Reasoning system is presented. TAAABLE consists of a user interface, an *ad-hoc* CBR engine and an extensive knowledge base stored in a semantic wiki. The knowledge base consists of the recipes, the domain ontology, and a set of AK. A semantic wiki is used

in order to facilitate knowledge enrichment tasks for TAAABLE users and contributors.

TAAABLE is an active research project. Improvements and new developments are made on a regular basis. Ongoing work includes the following tasks. First, TAAABLE's user interface has to be improved in order to facilitate the expression of complex queries for inexperienced users. Next, the knowledge acquisition modules have to be better integrated into the main application. This is important to encourage users to experiment with these modules. Indeed, these modules have mainly been used by TAAABLE developers, aware of how the system works; if these modules were efficiently used by more users, including, ideally, cooking experts with little skill in computer science, the TAAABLE knowledge containers would be improved. For this reason, we need to build a community of TAAABLE users in order to get better feedback on the application and to help us with the knowledge evolution tasks. Our work on TAAABLE also raised other issues that are discussed below.

TAAABLE participates in the Computer Cooking Contest (CCC) since 2008. The system was declared vice-champion in 2008 and 2009, adaptation-challenge champion in 2009 and world champion in 2010. Competing in the CCC is a way of evaluating CBR cooking systems such as TAAABLE. The CCC allows testing the capabilities of cooking systems to solve requests created by external examiners (and not by system's developers). It also makes it possible for us to compare our results with those obtained by other systems, which gives a first overview of the benefits and the drawbacks of each system. That being said, for many reasons, the CCC is not a sufficient form of evaluation for a system such as TAAABLE. Firstly, the CCC does not provide any Gold Standard or evaluation protocol. The evaluation is a comparative evaluation between systems only. Therefore, it is not possible to quantitatively assess the quality of the results obtained by a system. The CCC does not comply with what we usually expect in a benchmark. Second, the tasks given in the CCC are very general and, consequently, the evaluation is also very general. Applications are not evaluated on specific points such as quality of the knowledge acquisition, or adaptation of preparations, for example. Third, the CCC gathers only a small number of participants and experiments with systems during a very short amount of time, which does not provide good coverage of all the possible outcomes of the competing systems. We believe that elaborating evaluation strategies for the different components of TAAABLE is an important challenge. It is a complex task for two reasons. Firstly, there are a lot of components to evaluate (quality of the knowledge base, quality of the retrieval process, quality of the adaptation process, ergonomics of the user interface, etc.). Second, in the cooking domain, evaluation has a necessary degree of subjectivity. Indeed, the "quality" of a recipe produced in response to a query is very user-dependent. Therefore, we need to find a way to take into account the subjective dimension in the evaluation process. Implementing interfaces allowing users to provide feedback and the results, and

being able to take into account this feedback to improve TAAABLE is a major challenge.

During the development of TAAABLE, we have extended it with many components to improve the reasoning process, the user interface, the integration of external sources of knowledge, and the interactive acquisition of knowledge. Among these improvements is WIKITAAABLE, a semantic wiki that acts as a unique tool to manage the whole knowledge base of the system. WIKITAAABLE was designed to solve the knowledge maintenance problems we had during the first year of the project. During this year, we built, mostly manually, the knowledge sources used by TAAABLE. To that end, we had to integrate external knowledge sources and to annotate the recipe book. This manual process led to many errors that had to be corrected. However, because several people were involved in this process, versioning issues soon arose. This is how we realised we needed a collaborative tool in order to manage the knowledge base of the system. We chose Semantic MediaWiki because it seemed to be a smooth solution to support collaborative editing of knowledge bases, because it was a web-based solution, and because it provided us with connectors in order to easily plug in the CBR engine of TAAABLE. But, with the ease of use, risks appeared. Indeed, with WIKITAAABLE, it is now very easy to make a "mistake" while editing the knowledge base and thus to jeopardise TAAABLE results. For that reason, we need to setup testing and validation mechanisms (especially regression tests) to ensure that ontology modifications do not degrade the performances of the system. This is an active topic of research for TAAABLE, within the framework of the Kolflow project.

Another topic we wish to investigate is the consideration of different viewpoints in a single system. For example, in Brazil, avocados are often eaten as a dessert, mixed with milk and sugar, while in France, they are eaten as a starter, with shrimps and mayonnaise. Given the way the knowledge base is organised in TAAABLE, it is currently impossible to represent these two points of view. It is always possible to define AK to solve this issue, but this solution does not provide the flexibility we expect. We believe that viewpoints must be taken into account in the design of the knowledge base, and we investigate solutions making it possible for users to define their own knowledge bases while collaborating with others users. This raises a lot of questions regarding knowledge representation, knowledge sharing, consistency of knowledge bases and conflict management, but it is a very challenging and promising topic.

Last but not least, making the CBR inference engine evolve appears to be an important issue. For the moment, the inference engine of TAAABLE performs query reformulations (generalisations) in order to find satisfactory recipes. Recipes are adapted through a specialisation process in order to match the components of the initial query. Therefore, the engine is based on a generalisation/specialisation mechanism. The engine only processes knowledge expressed in propositional logic. This representation formalism does not allow representation of attributes (such as the quantities of ingredients or their

properties). As a consequence, to process such attributes, additional adaptation mechanisms have to be developed on top of the main engine. Moreover, these attributes are not taken into account during the retrieval process (for example, it is not possible to ask TAAABLE for a "low sugar recipe" at the moment). Retrieval only relies on the knowledge represented in the ingredient ontology. Additional AK, such as adaptation rules, is not taken into account during this step, but it should. We should therefore move towards a more expressive representation language while making sure that computation time remains reasonable. We need to make the inference engine evolve accordingly. Indeed, a joint and coordinated development of the knowledge bases and the reasoning engine is required. Such a refactoring would also give us the opportunity to develop a more generic inference engine and to experiment with it in other application domains.

References

1. Baader, F., Hollunder, B., Nebel, B., Profitlich, H.J.: An Empirical Analysis of Optimization Techniques for Terminological Representation Systems. In: Proceedings of the third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Cambridge, Massachussetts, pp. 270–281 (1992)
2. Badra, F.: Extraction de connaissances d'adaptation en raisonnement à partir de cas. Ph.D. thesis, Université Henri Poincaré - Nancy I (2009)
3. Badra, F., Cordier, A., Lieber, J.: Opportunistic adaptation knowledge discovery. In: Case-Based Reasoning Research and Development, 8th International Conference on Case-Based Reasoning, ICCBR 2009, Seattle, WA, USA, July 20-23, 2009, Proceedings, pp. 60–74 (2009)
4. Carpineto, C., Romamo, G.: Order-Theoretical Ranking. Journal of the American Society for Information Science **51**(7) (2000)
5. Cojan, J., Lieber, J.: Belief revision-based case-based reasoning. In: G. Richard (ed.) Proceedings of the ECAI-2012 Workshop SAMAI: Similarity and Analogy-based Methods in AI, pp. 33–39 (2012)
6. Cordier, A.: Interactive knowledge acquisition in case based reasoning. Ph.D. thesis, Université Claude Bernard Lyon 1, France (2008)
7. Cordier, A., Lieber, J., Molli, P., Nauer, E., Skaf-Molli, H., Toussaint, Y.: WIKITAAABLE: A semantic wiki as a blackboard for a textual case-based reasoning system. In: SemWiki 2009 - 4rd Semantic Wiki Workshop at the 6th European Semantic Web Conference - ESWC 2009. Heraklion, Grèce (2009). URL http://hal.inria.fr/inria-00432353
8. d'Aquin, M., Badra, F., Lafrogne, S., Lieber, J., Napoli, A., Szathmary, L.: Case base mining for adaptation knowledge acquisition. In: Proceedings of the International Conference on Artificial Intelligence, IJCAI'07, pp. 750–756 (2007)
9. DeMiguel, J., Plaza, L., Díaz-Agudo, B.: ColibriCook: A CBR System for Ontology-Based Recipe Retrieval and Adaptation. In: M. Schaaf (ed.) Workshop Proceedings of the 9th European Conference on Case-Based Reasoning, pp. 199–208. Trier (2008)
10. Developed, J.J.A., Knowledge, C.O.O.: P. J. Herrera and P. Iglesias and D. Romero and I. Rubio and B. Díaz-Agudo. In: M. Schaaf (ed.) Workshop Proceedings of the 9th European Conference on Case-Based Reasoning, pp. 209–218. Trier (2008)

11. Gaillard, E., Lieber, J., Nauer, E.: Adaptation knowledge discovery for cooking using closed itemset extraction. In: The Eighth International Conference on Concept Lattices and their Applications - CLA 2011, pp. 87–99. Nancy, France (2011). URL http://hal.inria.fr/hal-00646732
12. Gaillard, E., Nauer, E., Lefevre, M., Cordier, A.: Extracting Generic Cooking Adaptation Knowledge for the TAAABLE Case-Based Reasoning System. In: Cooking with Computers workshop @ ECAI 2012. Montpellier, France (2012). URL http://hal.inria.fr/hal-00720481
13. Ganter, B., Wille, R.: Formal Concept Analysis. Springer, Heidelberg (1999)
14. Hammond, K.J.: Case-Based Planning: Viewing Planning as a Memory Task. Academic Press, San Diego (1989)
15. Hanft, A., Newo, R., Bach, K., Ihle, N., Althoff, K.D.: Cookiis - a successful recipe advisor and menu advisor. In: S. Montani, L. Jain (eds.) Successful Case-based Reasoning applications, pp. 187–222. Springer
16. Jo, T.C., Seo, J.H., Hyeon, K.: Topic Spotting on News Articles with Topic Repository by Controlled Indexing. In: S.B.. Heidelberg (ed.) Intelligent Data Engineering and Automated Learning — IDEAL 2000. Data Mining, Financial Engineering, and Intelligent Agents, *Lecture Notes in Computer Science*, vol. 1983, pp. 89–99 (2008)
17. Krötzsch, M., Schaffert, S., Vrandecic, D.: Reasoning in semantic wikis. In: G. Antoniou, U. Aßmann, C. Baroglio, S. Decker, N. Henze, P.L. Patranjan, R. Tolksdorf (eds.) Reasoning Web, *Lecture Notes in Computer Science*, vol. 4636, pp. 310–329. Springer (2007)
18. Leake, D.B., Kinley, A., Wilson, D.: Acquiring Case Adaptation Knowledge: A Hybrid Approach. In: AAAI/IAAI, vol. 1, pp. 684–689 (1996)
19. Lieber, J.: Strong, Fuzzy and Smooth Hierarchical Classification for Case-Based Problem Solving. In: F. van Harmelen (ed.) Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02), Lyon, France, pp. 81–85. IOS Press, Amsterdam (2002)
20. Messai, N., Devignes, M.D., Napoli, A., Smaïl-Tabbone, M.: Querying a bioinformatic data sources registry with concept lattices. In: F. Dau, M.L. Mugnier, G. Stumme (eds.) ICCS, *Lecture Notes in Computer Science*, vol. 3596, pp. 323–336. Springer (2005)
21. Messai, N., Devignes, M.D., Napoli, A., Smaïl-Tabbone, M.: Many-valued concept lattices for conceptual clustering and information retrieval. In: M. Ghallab, C.D. Spyropoulos, N. Fakotakis, N.M. Avouris (eds.) ECAI, *Frontiers in Artificial Intelligence and Applications*, vol. 178, pp. 127–131. IOS Press (2008)
22. Minor, M., Bergmann, R., Görg, S., Walter, K.: Adaptation of cooking instructions following the workflow paradigm. In: C. Marling (ed.) ICCBR 2010 Workshop Proceedings, pp. 199—208 (2010)
23. Pearl, J.: Heuristics – Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Publishing Co., Reading, MA (1984)
24. Q. Zhang R. Hu, B.M.N., Delany, S.J.: Back to the Future: Knowledge Light Case Base Cookery. In: M. Schaaf (ed.) Workshop Proceedings of the 9th European Conference on Case-Based Reasoning, pp. 239–248. Trier (2008)
25. Smyth, B., Keane, M.T.: Using adaptation knowledge to retrieve and adapt design cases. Knowledge-Based Systems **9**(2), 127–135 (1996)
26. Szathmary, L., Napoli, A.: Coron: A framework for levelwise itemset mining algorithms. In: B. Ganter, R. Godin, E.M. Nguifo (eds.) Third International Conference on Formal Concept Analysis (ICFCA'05), Lens, France, Supplementary Proceedings, pp. 110–113 (2005). Supplementary Proceedings
27. Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F.: Semantic annotation for knowledge management: Requirements and a survey of the state of the art. Journal of Web Semantics: Science, Services and Agents on the World Wide Web **4**(1), 14–28 (2006)

Resumes of the authors

Amélie Cordier is an assistant professor at the University of Lyon 1. She does her research at the LIRIS Laboratory. She got her PhD from Lyon 1 University. Her main research field is dynamic knowledge engineering. She works with case-based reasoning and trace-based reasoning. She lead the TAAABLE project it 2008 and 2009 and she organized the Computer Cooking Contest in 2010 and 2011.

Jean Lieber is an assistant professor of Lorraine Université with a PhD and a habilitation degree in computer science, doing his research at LORIA. His main research field is CBR, with an emphasis on knowledge representation for CBR and adaptation in CBR. He has participated to the TAAABLE project since the first Computer Cooking Contest (2008) and was the TAAABLE project leader in 2011.

Emmanuel Nauer is an assistant professor of Lorraine Université and member of the Orpailleur team, at LORIA. Emmanuel Nauer is currently the leader of the TAAABLE project, on which he has participated since its beginning. In the TAAABLE project, he has been in charge of the aquisition of the ontology, of the annotation process of recipes, and of knowledge discovery for improving the results of the CBR system.

Fadi Badra received a Phd degree in computer science from the University of Nancy in 2009. He is an assistant professor at the University of Paris 13 Bobigny, where he joined the Biomedical Informatics research group (LIM&BIO). Fadi's research contributions to TAAABLE concerned techniques to acquire adaptation knowledge, either from the end user, or from the case base by the means of knowledge discovery.

Julien Cojan is currently an INRIA engineer in the team Wimmics. He works on data extraction from semi-structured textual resources for the semantic web. He has a PhD in computer science from Nancy University in 2011, on the application of belief revision to CBR, that is used in the TAAABLE system for adapting ingredient quantities.

Valmi Dufour-Lussier graduated from Montréal University and Nancy 2 University and is currently a PhD candidate in Computer Science at Lorraine University. His area of research is located at the interface between textual CBR and spatio-temporal reasoning. He has been involved in TAAABLE since 2009, and has led the research on recipe text adaptation.

Emmanuelle Gaillard graduated from Nancy 2 University, and is currently a PhD student in Computer Science at Lorraine University. Her thesis focus on acquisition and management of meta-knowledge to improve a case-based reasoning system. She works also on adaptation knowledge discovery and applies this work to the TAAABLE system.

Laura Infante-Blanco obtained her computing engineer degree in Universidad de Valladolid, Spain, in 2011. She is currently working as an INRIA engineer in the orpailleur team developping a generic ontology guided CBR

system. She has been involved in the development of WikiTaaable and she is currently in charge of the wiki management.

Pascal Molli is full professor at University of Nantes and is head of the GDD Team in LINA research center. He has published more than 80 papers in software engineering, information systems, distributed systems, and computer supported cooperative work (CSCW). He mainly works on collaborative distributed systems and focuses on algorithms for distributed collaborative systems, distributed collaborative systems, privacy and security, and collaborative distributed systems for the Semantic Web.

Amedeo Napoli is a CNRS senior scientist and has a doctoral degree in Mathematics and an habilitation degree in computer science. He is the scientific leader of the Orpailleur research team at the LORIA laboratory in Nancy. He works in knowledge discovery, knowledge representation, reasoning, and Semantic Web.

Hala Skaf-Molli received a PhD in computer science from Nancy University in 1997. From 1998 to September 2010, she was an associate professor at Nancy University, LORIA. Since October 2010, she is an associate professor at Nantes University, LINA. She has mainly worked on distributed collaborative systems and social semantic web.