Improving case retrieval by enrichment of the domain ontology

Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Yannick Toussaint

Improving case retrieval by enrichment of the domain ontology

Valmi Dufour-Lussier[1,2], Jean Lieber[1,3],
Emmanuel Nauer[1,4], and Yannick Toussaint[1,5]

[1] LORIA – UMR 7503. B.P. 239, F-54506 Vandœuvre-lès-Nancy CEDEX, France
[2] Université Nancy 2 [3] Université Henri Poincaré, Nancy 1 [4] Université Paul Verlaine, Metz
[5] INRIA – Nancy Grand-Est
{firstname.lastname}@loria.fr

Abstract. One way of processing case retrieval in a case-based reasoning (CBR) system is using an ontology in order to generalise the target problem in a progressive way, then adapting the source cases corresponding to the generalised target problem. This paper shows how enriching this ontology improves the retrieval and final results of the CBR system. An existing ontology is enriched by automatically adding new classes that will refine the initial organisation of classes. The new classes come from a data mining process using formal concept analysis. Additional data about ontology classes are collected explicitly for this data mining process. The formal concepts generated by the process are introduced into the ontology as new classes. The new ontology, which is better structured, enables a more fine-grained generalisation of the target problem than the initial ontology. These principles are tested out within TAAABLE,[1] a CBR system that searches cooking recipes satisfying constraints given by a user, or adapts recipes by substituting certain ingredients for others. The ingredient ontology of TAAABLE has been enriched thanks to ingredient properties extracted from recipe texts.
Keywords: ontology refinement, formal concept analysis, progressive retrieval

1 Introduction

This paper studies the effect of the domain ontology on a case-based reasoning (CBR) system and the improvement of result quality using a more fine-grained ontology.

TAAABLE [2] is a system as a candidate of the *Computer Cooking Contest*. It is also used as a brain teaser for research in knowledge based systems, including CBR and ontology engineering. Like many CBR systems, TAAABLE uses an ontology to retrieve the source cases that are the most similar to a target case. TAAABLE retrieves and creates cooking recipes by adaptation. According to constraints given by the user, such as inclusion or rejection of ingredients, the type or the origin of the dish, the compatibility with some diets (vegetarian, nut-free, etc.), the system looks up, in the recipe base (which is a case base), whether some recipes satisfy these constraints. Recipes, if they exist, are returned to the user; otherwise the system is able to retrieve similar recipes (i.e. recipes that match partially the target query) and adapts these recipes, creating new ones. Searching similar recipes is guided by several ontologies, i.e. hierarchies of classes (ingredient hierarchy, dish type hierarchy and dish origin hierarchy),

[1] http://taaable.fr

in order to relax constraints by generalising the user query. Relaxation is iterative and progressive. The goal is to find the most specific generalisation of the target case (the one with the minimal cost) for which recipes exist in the case base. Adaptation consists of substituting some ingredients of the source cases by the ones required by the user.

Many CBR systems use class or index hierarchies (taxonomies, ontologies, object-based hierarchies [5], etc.) for retrieval, especially to compute similarity between cases. Therefore, the ontology structure impacts retrieval, raising the issue of how to refine this ontology for improving this process. This paper proposes a novel approach to automatise this refinement.

The main idea for realising this goal is as follows. In a given ontology, the siblings of a class C may be either close to C of far from it, depending on the level of details of this part of the ontology. For instance, if an ingredient ontology is very detailed for berries (i.e. with a deep hierarchy below the class `Berry`) and very coarse for vegetables (i.e. each vegetable class is a direct subclass of the class `Vegetable`) then the similarity between a blackberry and a blueberry would counter-intuitively be much lower than the similarity between a pumpkin and a parsnip. Generalisation of the query is the basis for case retrieval and it is computed using a similarity measure defined on ontologies. Introducing new classes into the ontology provides a more fine-grained generalisation. Going up in a more structured hierarchy will return less and more similar cases than going up in a less structured hierarchy.

A data mining process based on formal concept analysis (FCA) extracts new classes and enriches the initial ontology by structuring it better. Additional sources are collected and used to introduce new properties about classes of the initial ontology. This work helps avoid adaptation failures, i.e. creating a recipe that is not *cookable* because some actions that were applied to the substituted ingredient are not applicable to the substitution ingredient: for instance, when substituting *mascarpone* (a creamy cheese) for *mozzarella* (a semi-firm cheese), the *slice* action will not be applicable anymore.

The paper is organised as follows: Section 2 presents TAAABLE inference engine principles, Section 3 describes related work about existing approaches for improving retrieval in CBR and ontology refinement using FCA, Section 4 details the ontology refinement process, and Section 5 shows on an example the effect of the refined ontology in the TAAABLE system.

2 Reasoning principles in TAAABLE

TAAABLE retrieves cases using query generalisation, then adapts them by substitution. This section gives a simplified description of the TAAABLE system. For more details about the TAAABLE inference engine, see e.g. [2].

Case retrieval.

Domain ontology. An ontology $\mathcal{O}$ defines the main classes and relations relevant to cooking, and guides retrieval. $\mathcal{O}$ is a set of atomic classes organised into several hierarchies (ingredient, dish type, dish origin). Given two classes `B` and `A` of this ontology, `A` is subsumed by `B`, denoted by "`B` $\sqsupseteq$ `A`", iff the set of instances of `A` is included in

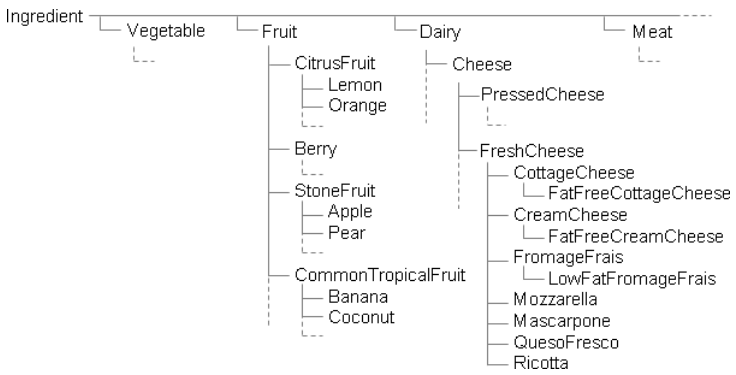

Fig. 1. A part of the ingredient hierarchy, including the details of the `FreshCheese` class.

the set of instances of `B`. For instance, "`Riccota` (referring to the Italian fresh cheese) is subsumed by `FreshCheese`" means that all instances of `Riccota` are instances of `FreshCheese`. For the sake of simplicity, only the ingredient hierarchy is considered in the remainder of this paper. A part of this hierarchy is given in Fig. 1.

Source cases. For the inference engine, recipes are represented by a conjunction of the ingredients they call for, and is encoded in the same language as the one of the queries. Thus, a recipe with ricotta, tomato, olive oil, and salt, is represented by:

$$R = \exists \texttt{ing.Ricotta} \sqcap \exists \texttt{ing.Tomato} \sqcap \exists \texttt{ing.OliveOil} \sqcap \exists \texttt{ing.Salt} \quad (1)$$

Target query. In the TAAABLE system, a query Q is a conjunction of terms $\exists \texttt{ing}.I$, where I is a class in the ingredient hierarchy, meaning that the recipe should contain the ingredient I. For example, searching a recipe with tomato and fresh cheese corresponds to $Q = \exists \texttt{ing.Tomato} \sqcap \exists \texttt{ing.FreshCheese}$. If `C` and `D` are two conjunctions (for example, two queries), $\texttt{C} = \bigsqcap_i \exists \texttt{ing.A}_i$ is subsumed by $\texttt{D} = \bigsqcap_j \exists \texttt{ing.B}_j$ if, for each j, there exists at least one i such that $\texttt{A}_i$ is subsumed by $\texttt{B}_j$ in $\mathcal{O}$.

Given a recipe R and a query Q, R solves Q if the set of recipes represented by R is included in the set of recipes represented by Q. Thus, the recipe represented by (1) solves the query $Q = \exists \texttt{ing.FreshCheese} \sqcap \exists \texttt{ing.Tomato}$. This can be computed simply by testing if R is more specific than Q. It is important to notice that R solves Q does not mean that it is a "good" solution, even if the hypothesis that recipes of the case base are "good" is made (this is a working hypothesis: as no information about the quality of the recipes in the case base is available, it is simplest to consider the recipes as being good or, at least, equally good).

Algorithm. The algorithm consits in an iterated generalisation of the target query minimising cost, until at least one recipe of the case base solves the modified query. The query that has been modified at iteration t is noted $\Gamma_t(Q)$, and Γ_0 is the identity. The generalisation cost of Γ_t, denoted by $\texttt{cost}(\Gamma_t)$, grows with t (see below for the `cost`

function definition). Each generalisation Γ_t is written as a composition of substitutions $\mathtt{A} \rightsquigarrow \mathtt{B}$ such that $(\mathtt{A}, \mathtt{B})$ is a specialisation link between $\mathtt{A}$ and $\mathtt{B}$ in the hierarchy representing the ontology. Furthermore, this composition must be applied to Q: if $\Gamma = (\mathtt{A}_p \rightsquigarrow \mathtt{B}_p) \circ \ldots \circ (\mathtt{A}_2 \rightsquigarrow \mathtt{B}_2) \circ (\mathtt{A}_1 \rightsquigarrow \mathtt{B}_1)$ then $\mathtt{A}_1$ is an element of the conjunction Q, $\mathtt{A}_2$ is an element of the conjunction $(\mathtt{A}_1 \rightsquigarrow \mathtt{B}_1)(Q)$, etc. The algorithm running through the generalisation space of Q is an A* algorithm which takes as parameter the cost function. It stops when at least one recipe R such that R solves $\Gamma_t(Q)$ is found.

The cost function. Let Γ be a composition of substitutions. Function $\mathtt{cost}$ is additive for the composition: $\mathtt{cost}(\sigma_2 \circ \sigma_1) = \mathtt{cost}(\sigma_1) + \mathtt{cost}(\sigma_2)$. For a basic substitution $\sigma = \mathtt{A} \rightsquigarrow \mathtt{B}$ (where $\mathtt{A}$ is subsumed by $\mathtt{B}$), the cost is computed as follows:

$$\mathtt{cost}(\mathtt{A} \rightsquigarrow \mathtt{B}) = \mu(\mathtt{B}) - \mu(\mathtt{A}) \quad \text{with} \quad \mu(\mathtt{X}) = \frac{\text{number of recipes with } \mathtt{X}}{\text{total number of recipes}}$$

Therefore, $0 \leq \mathtt{cost}(\mathtt{A} \rightsquigarrow \mathtt{B}) \leq 1$.

Function $\mathtt{cost}$ has the following property: if Γ has a cost computed with respect to a given ontology and a new node is added into this ontology (without changing the recipe base), then the cost of Γ will not change. For instance, if $\Gamma = \mathtt{Ricotta} \rightsquigarrow \mathtt{FreshCheese}$ and the $\mathtt{ItalianFreshCheese}$ class is "added" into the ontology "between" $\mathtt{Ricotta}$ and $\mathtt{FreshCheese}$, the cost of Γ remains unchanged. So, there is an essential difference between this cost function and one based on the number of edges separating two nodes (number that grows by adding an intermediate node). This property ensures stability of the cost regardless of the introduction of intermediate classes.

Adaptation. Let Q be a query, Γ a generalisation function of this query, and R a recipe solving $\Gamma(Q)$. Let $\mathtt{T}_i$, $\mathtt{B}_j$ and $\mathtt{S}_k$ be classes such as $Q = \bigsqcap_i \exists \mathtt{ing}.\mathtt{T}_i$ ($\mathtt{T}$ as target), $\Gamma(Q) = \bigsqcap_j \exists \mathtt{ing}.\mathtt{B}_j$ and $R = \bigsqcap_k \exists \mathtt{ing}.\mathtt{S}_k$ ($\mathtt{S}$ as source). $\Gamma(Q)$ is more general than both R and Q, thus for all j, there exists k and i such that $\mathtt{S}_k$ and $\mathtt{T}_i$ are subsumed by $\mathtt{B}_j$. Then the adaptation consists of substituting $\mathtt{T}_i$ for $\mathtt{S}_k$ for all $\mathtt{B}_j$, such that $\mathtt{B}_j \neq \mathtt{T}_i$. If several $\mathtt{T}_i$ or $\mathtt{S}_k$ correspond to one $\mathtt{B}_j$, several adaptations are possible and are returned to the user. These substitutions, along with the preparation instructions, give the modifications that must be applied to the recipe in order to address the query.

Example. Let $Q = \exists \mathtt{ing}.\mathtt{Tomato} \sqcap \exists \mathtt{ing}.\mathtt{Mascarpone} \sqcap \exists \mathtt{ing}.\mathtt{Oil}$. Among the generalisations of this query, say that $\Gamma = \mathtt{Mascarpone} \rightsquigarrow \mathtt{FreshCheese}$ is the substitution with the minimal cost (apart from identity). As $\mathtt{Ricotta}$ is subsumed by $\mathtt{FreshCheese}$ in the ontology, TAAABLE retrieves the recipe R given by (1). The adaptation will consist in substituting $\mathtt{S}_k = \mathtt{Ricotta}$ by $\mathtt{T}_i = \mathtt{Mascarpone}$ (generalisation-specialisation through $\mathtt{B}_j = \mathtt{FreshCheese}$).

However, a non-realistic adaptation can be returned, in which some actions have to be applied to objects on which they cannot be applied to. We will address this problem after introducing our approach to ontology refinement.

	FreshCheese	sliceAble	beatAble	cutAble	mashAble
Mascarpone	×		×		
Ricotta	×		×	×	×
Mozzarella	×	×		×	
FromageFrais	×		×		
CottageCheese	×		×		×
QuesoFresco	×		×	×	×

Table 1. A binary context with 6 objects (cheeses from the FreshCheese category) described by 5 properties (the FreshCheese category and 4 cooking actions which can be applied to these cheeses).

3 Formal concept analysis for ontology refinement

The goal of the refinement process is to add intermediate classes into the initial hierarchy of the system. These additional classes will enable for a better distinction of the similarity between classes immediately subsumed initially by a same class. Fig. 2 illustrates how B, C, and D which are indistinguishably similar on the left-hand side are distinguished better after introducing an intermediate class (in this example, C and D more similar to each other than to B).

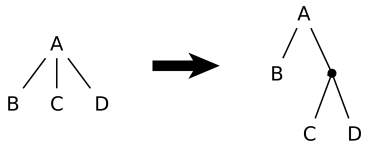

Fig. 2. Refining the ontology by introducing a new class.

3.1 Formal concept analysis

FCA is a classification method based on lattice theory. A *formal context* is a triple $\mathcal{K} = (G, M, I)$, where G is a set of individuals (called *objects*), M a set of properties (called *attributes*) and I the relation on $G \times M$ stating that an object is described by a property [9]. Table 1 shows an example context: G is a set of 6 objects (which are fresh cheese subclasses), M is the set of 5 properties applicable to those cheeses. Among theses properties, one (FreshCheese) comes from the initial hierarchy. FreshCheese is a class which is more general than the 6 cheese classes that have to be classified (in other words, FreshCheese denotes both the class of fresh cheese in the ontology and the property of being a fresh cheese). The 4 other properties are cooking actions that can be applied: sliceAble (can be sliced), beatAble (can be beaten), cutAble (can be cut), and mashAble (can be mashed).

The approach for merging properties coming from several contexts, for a given set of objects, is called *context apposition*. Formally, if $\mathcal{K}_1 = (G, M_1, I_1)$ and $\mathcal{K}_2 =$

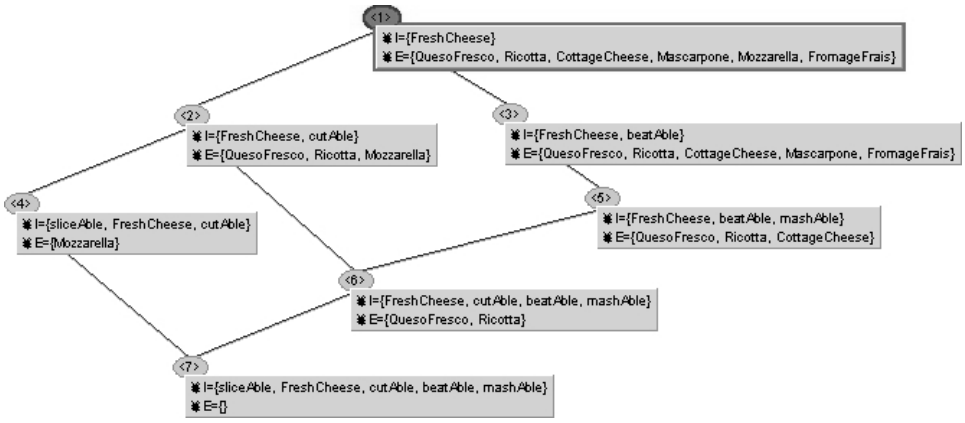

Fig. 3. The lattice corresponding to the binary context given in Table 1.

(G, M_2, I_2) are two binary contexts about the same set of objects G, a binary context, written $\mathcal{K}_1|\mathcal{K}_2$ merging $\mathcal{K}_1$ and $\mathcal{K}_2$ can be built by: $\mathcal{K}_1|\mathcal{K}_2 = (G, M_1 \cup M_2, I_1 \cup I_2)$ [4].

A *formal concept* is a pair (P, O) where P is a set of properties (the *intent* of the formal concept), O is a set of objects (the *extent* of the formal concept), such that (1) P is the set of all properties shared by objects in O and (2) O is the set of all objects sharing properties P. For example, $(\{\texttt{FreshCheese}, \texttt{beatAble}, \texttt{mashAble}\}, \{\texttt{QuesoFresco}, \texttt{Ricotta}, \texttt{CottageCheese}\})$ is a formal concept (node number 5 in Fig. 3).

The set $\mathcal{C}_\mathcal{K}$ of all formal concepts of the context $\mathcal{K} = (G, M, I)$ is partially ordered by extent inclusion, also called the *specialisation* (denoted by $\leq_\mathcal{K}$), between concepts. $\mathcal{L} = \langle \mathcal{C}_\mathcal{K}, \leq_\mathcal{K} \rangle$ is a complete lattice, called the *concept lattice*. The lattice $\mathcal{L}$ can be drawn as a Hasse diagram where nodes are concepts, and edges are specialisation links. Fig. 3 illustrates the lattice corresponding to the binary context given in Table 1. The top concept (with number 1, in the figure) contains all the objects. In our example, its intent is `FreshCheese`, a property shared by all the objects. By contrast, the bottom concept is defined by the set of all properties. In our example, its extent is empty as none of the objects is described by all the properties.

A number of algorithms have been proposed for the construction of concept lattices (see [9]). For our application, we used CORON, a software platform implementing a rich set of algorithmic methods for symbolic data mining, including concept lattice construction algorithms [11]. Fig. 3 was generated by GALICIA,[2] another FCA tool, which provides visualisation functionalities.

3.2 Ontology building using FCA

Many works propose using FCA to build ontologies. Cimiano [6] proposes an approach to build a concept hierarchy from texts in the tourism domain. The objects (hotels, cars, bicycles, tours, etc.) are characterised by the actions that can be applied to them (can

[2] `http://www.iro.umontreal.ca/~galicia`

be reserved, can be driven, can be rented, etc.). In order to reduce the lattice size, a numerical classification is used as a first step of the binary context building.

Stumme *et al.* [10] propose an approach to merging two ontologies. The ontology classes are identified in the texts and a binary context is built for each of the two ontologies, in which classes are characterised by the documents in which they appear. Context apposition is used for aggregating the classes of the two ontologies according to their presence in the texts.

[4] shows how FCA can be used as a unified framework for ontology building and refinement. It points out that the information that may be extracted from the texts depends on the textual resources itself. Thus, from a scientific paper, objects can be extracted and properties can be linked to objects as it is done in [6]. But an ontology organises classes of a given domain into a hierarchy. This information about the specific/generic organisation is seldom present in scientific papers. In Bendaoud *et al.* [4], these two types of information are managed separately and are then combined with context apposition to build a single lattice.

We use this type of approach to build the ingredient ontology in the cooking domain. Recipe texts are used in order to characterise the ingredients according to how they are cooked, and the initial ontology is used in order to get the specific/generic relations.

4 Ontology refinement process

FCA has been succesfully used in CBR: in [1] for building cases from texts (objects correspond to texts and properties to relevant terms), in [7] for organising a case base for the purpose of retrieval. In the present work, FCA is used for ontology refinement.

The goal of the refinement process is to add intermediate classes into the initial hierarchy of the system. To do that, the classes of the initial ontology are characterised with additional properties. These properties enable the creation of a binary context that will be exploited for building a concept lattice with FCA. The formal concepts of the lattice are inserted into the initial hierarchy in order to enrich its organisation. The following four steps describe the ontology building and refinement process. We detail those steps with examples about an ingredient hierarchy refinement.

4.1 Ingredient hierarchy overview

The initial ingredient hierarchy has been semi-automatically created from common sense classifications (e.g. a *citrus fruit* is a *fruit*). All ingredients used in the recipe base have been introduced. The ingredient hierarchy currently contains 1500 classes. An excerpt is shown in Fig. 1. The ontology (and the recipes of the case base) are managed in a semantic wiki [3]. The *high* level of the hierarchy contains general classes: `Vegetable`, `Fruit` , `Dairy`, `Meat`, etc. The *low* level of the hierarchy contains classes that are ingredients actually used in the recipes.

From the sole hierarchy, the *similarity* between two *sibling ingredients* (i.e. immediate subclasses of a single class) cannot be ranked. For example, mascarpone is as close to mozzarella as it is to ricotta. Consequently, when a query is generalised for case retrieval, if `Mascarpone` is generalised to `FreshCheese`, the set of remaining cases will

contain recipes using mozzarella as well as recipes using mascarpone. This is why we want to refine the hierarchy organisation for gathering initial classes that are similar (i.e. classes that share at least one property). Generalising a class C into a parent concept will ensure that sibling classes of C own at least the same properties as C. The generalisation will be more fine-grained.

4.2 Local refinement

The action of refining a limited part of the ontology is called *local refinement*. For instance, we could refine locally the sub-part about `Nut` (walnut, pecan nut, etc.), the sub-part about `Berry` (raspberry, blueberry, etc.), the sub-part about `FreshCheese`, etc. This approach has been preferred to global refinement for several reasons:

- Focusing on a part of the hierarchy (i.e. a subset of classes) is a way to limit the size of the binary context by reducing the number of objects and the number of associated properties. A reduced context will produce a lattice with a smaller size, which can be viewed and interpreted more easily.
- We do not want to build classes gathering ingredients that do not belong in the same part of the hierarchy. For instance, we do not want to group kiwis and onions merely because it is possible to apply a given action, such as *to peel*, to both.
- For each part of the hierarchy, the properties taken into account for the binary context may be different: *to toast* makes sense for nuts but not for liquids.

Therefore we ought to focus on the refinement of *(bottom)* parts of the hierarchy, while maintaining the *top* structure. The local refinements will be plugged into the initial ontology (see hereafter).

Formally, R refers to the class which is the root of the part of the hierarchy that has to be refined. The context $\mathcal{K}_R = (G_R, M_R, I_R)$ is the binary context composed of:

- a set of objects: in this work, objects used for FCA are classes of the ingredient hierarchy, e.g. `Orange`, more specific than R: $G_R = \{x \mid R \sqsupseteq x\}$,
- a set of properties (cooking actions) M_R selected to characterise and discriminate the objects of G_R,
- the relation I_R such that I_R relates $i \in G_R$ to $a \in M_R$ if the action a can be applied to the ingredient i.

The next two sections illustrate the refinement process for $R =$ `FreshCheese`.

4.3 Building the binary context for the refinement

The ontology refinement aims at modifying an existing ontology. Properties of the initial ontology must be retained, and new properties introduced to organise the classes (ingredients) better against each other. Aggregating ricotta with mascarpone and distinguishing these two fresh cheeses from mozzarella is an expected outcome. Two binary contexts must be merged:

- A binary context capturing the initial ontology structure by associating to each ingredient class their superclasses, e.g. the object `Ricotta` has the properties `Ricotta` and `FreshCheese`.

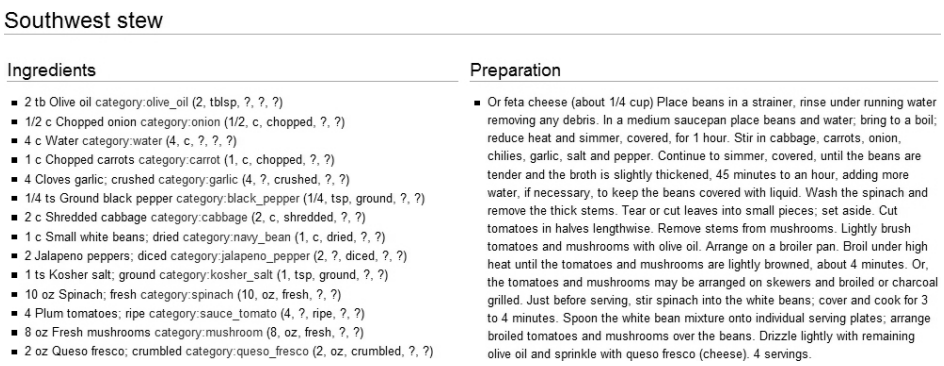

Fig. 4. Example of a recipe. The recipe contains a textual list of ingredients and preparation steps. Each ingredient line is parsed in order to extract structured data: quantity, unit, ingredient and additional pieces of information such as an action already applied or some precision about the ingredient (e.g. *fresh*).

– A binary context describing ingredients through the culinary actions that can be performed on them. For instance, ricotta and mascarpone can be beaten but mozzarella cannot, though mozzarella can be sliced, which is not the case for creamy cheeses.

Ingredient properties are extracted from 73795 recipes taken from Recipe Source.[3] Ingredient properties are extracted both from the ingredient list, which includes some cooking actions, and from the textual preparation, from which the actions applied to the ingredients are extracted using NLP tools [8]. The latter approach uses both classical morpho-syntactic analysis methods and a discourse representation specifically developed for procedural texts (instruction texts, including recipes). The discourse representation is well adapted for solving some complex anaphoric phenomena appearing in those types of texts. An example of a recipe used for property extraction is given in Fig. 4.

Extracting the properties of the ingredient list is done through an automatic annotation process, because the recipes are initially in a textual form. This annotation process was designed in order to obtain a formal representation of the recipes that can be handled by the CBR engine. The annotation process uses a terminological base for indexing the recipes by classes of the ingredient hierarchy. The quantities, the units, and the ingredient modifiers such as *chopped*, *diced*, *crumbled*, etc. (cf. Fig. 4) are also extracted. These modifiers are mostly cooking actions applied to the ingredients. Pairs (ingredient, modifier) and pairs (ingredient,applied action) extracted from the textual preparation are used for building the binary context. Some examples can be seen in the annotated recipe example given in Fig. 4. The set of properties M_R of $\mathcal{K}_R$ contains the cooking actions required for refining the ingredients of G_R. For example, let $R =$

[3] `http://www.recipesource.com/`

	FreshCheese	CottageCheese	FatFreeCottageCheese	CreamCheese	FatFreeCreamCheese	FromageFrais	LowFatFromageFrais	Mozzarella	Mascarpone	QuesoFresco	Ricotta	mashAble	meltAble	beatAble	mixAble	sliceAble	crumbleAble	cutAble
CottageCheese	×	×										×	×					
FatFreeCottageCheese	×	×	×									×	×					
CreamCheese	×			×								×	×	×	×			
FatFreeCreamCheese	×			×	×							×	×	×	×			
FromageFrais	×					×							×	×	×			
LowFatFromageFrais	×					×	×						×	×	×			
Mozzarella	×							×								×		×
Mascarpone	×								×				×	×	×			
QuesoFresco	×									×							×	×
Ricotta	×										×		×	×	×			

Table 2. A binary context for cheeses from the FreshCheese category, described by generic/specific properties and cooking actions that can be applied to the cheeses.

FreshCheese and $M_R = \{$mashAble, meltAble, beatAble, mixAble, sliceAble, cutAble, crumbleAble$\}$. Then, table 2 presents the binary context $\mathcal{K}_{\texttt{FreshCheese}}$.

4.4 Merging the local lattice and the initial hierarchy

Given a class R of the initial hierarchy and the lattice $\mathcal{L}_R$ built from the context $\mathcal{K}_R$, the initial ontology is refined by adding ontology classes corresponding to some of the concepts of $\mathcal{L}_R$. The choice of these concepts in $\mathcal{L}_R$ is based on the notion of *reduced extent* of a concept C of $\mathcal{L}_R$. Following the terminology of object-oriented languages, the reduced extent $E_r(C)$ of C is the set of *direct* instances of C. In other words, an object x belongs to the reduced extent of C if it belongs to the extent of C but does not belong to the extent of any direct descendant D of C in the lattice:

$$E_r(C) = E(C) \setminus \bigcup \{E(D) \mid D\text{: direct descendant of } C \text{ in the lattice}\}$$

It can be proven that $x \in E_r(C)$ iff the properties of C are exactly the ones of $I(C)$, the intent of C (whereas an $x \in E(C)$ may have additional properties, outside of $I(C)$). Moreover, for a context $\mathcal{K}_R$ built as explained in the previous section, for each concept C of $\mathcal{L}_R$, either $E_r(C) = \emptyset$ or $E_r(C)$ contains exactly one element.

Fig. 5 is the lattice $\mathcal{L}_{\texttt{FreshCheese}}$ (with the reduced extents instead of the extents). The grey concepts C are the ones that are chosen in order to be added to the initial ontology. The concept with an empty extent (concept 16 in Fig. 5) covers no instance and thus is not added to the ontology. Some concepts correspond to classes that are already in the ontology and need not be added. This is the case for the root concept, (in the example, FreshCheese, concept 1), and for the concepts C having exactly one

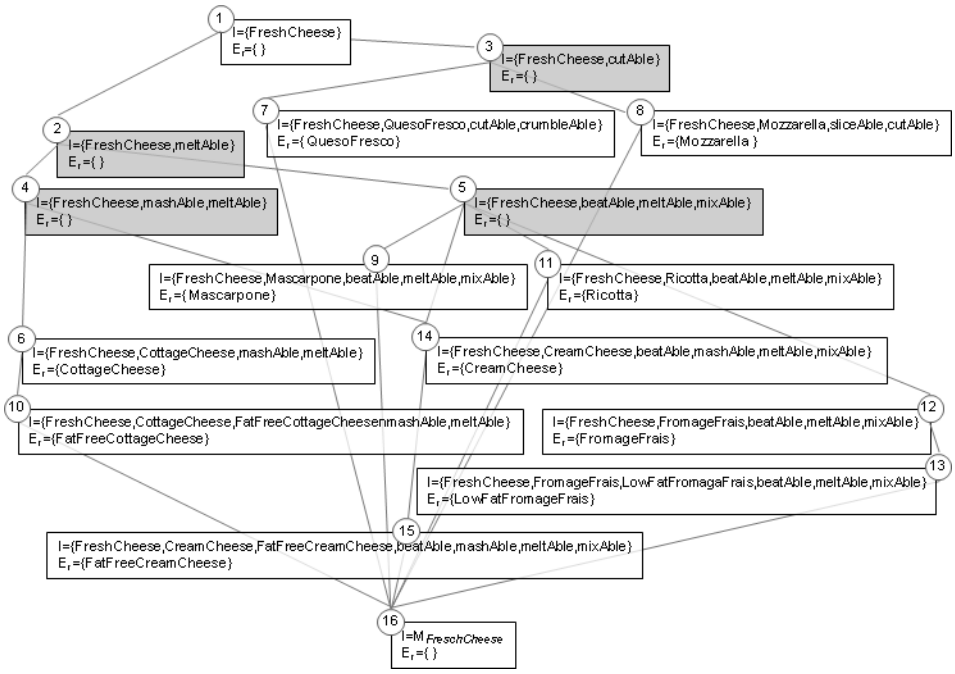

Fig. 5. Concept lattice restructuring the fresh cheeses.

element in their reduced extent: $E_r(C) = \{P\}$ (concepts 6 to 15). Such a FCA concept corresponds to the class P of the initial ontology. For instance, if C is the class P of the ontology, $E_r(C) = \{\texttt{Mozzarella}\}$ and the concept C corresponds to the ontology class `Mozzarella`.

The remaining FCA concepts are structuring concepts that are added to the ontology (concepts 2 to 5, in grey in the Fig. 5). Such a node C has an empty reduced extent, thus its extent equals the union of the extents of its direct descendants in $\mathcal{L}_R$. Its intent factorises properties of its subconcepts. Adding such a concept to the ontology, means reifying a new ontology class with a label obtained by concatenation of the intent properties. For example, the concept 5 leads to the class in the ontology whose label is `FreshCheese_beatAble_meltable_mixAble`. This class refers to the fresh cheeses that can be beaten, melted, and mixed. The part of the refined ontology regarding fresh cheeses is given on the right hand-side of Fig. 6.

5 Effect of the ontology refinement on the CBR process: an example

This section shows how the ontology refinement affects the CBR system. An example is given of TAAABLE's performance *before* and *after* the refinement in order to show

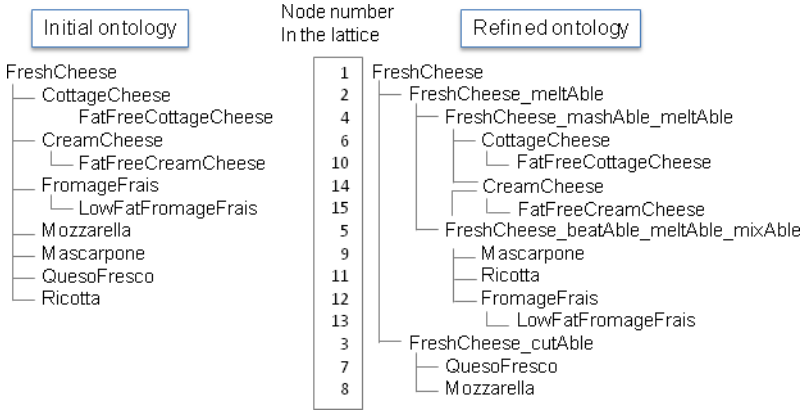

Fig. 6. Initial (left) and refined (right) ontology from the `FreshCheese` class down.

Fig. 7. TAAABLE$_0$ and TAAABLE$_1$'s answers to the query $\exists$`ing.Tomato` $\sqcap$ $\exists$`ing.Mascarpone`.

how retrieval is affected. The two TAAABLE systems use the same case base: the recipe book provided by the *Computer Cooking Contest,* containing 1488 recipes.

In order to evaluate the effect of the refinement, two TAAABLE systems were instantiated: TAAABLE$_0$ using the initial ontology and TAAABLE$_1$ using the refined ontology. The two systems were given the same query to answer. Fig. 7 shows the answer of the two systems to a query asking for a recipe with `tomato` and `mascarpone`. Since no recipe contains both those ingredient in the case base, the two systems search for similar cases.

For TAAABLE$_0$, the retrieval stops after the generalisation `Mascarpone` $\rightsquigarrow$ `FreshCheese`. At this step, 8 recipes satisfying the query $\exists$`ing.Tomato` $\sqcap$ $\exists$`ing.FreshCheese` are found and adapted by substituting `Mascarpone` for `FreshCheese`. Some of the recipes contain several `FreshCheeses`. In this case, the system proposes to substitute one or more ingredients (this is the case for the recipes numbered 1, 3 and 5). For TAAABLE$_1$, the retrieval is more fine-grained: the generalisation stops on the `FreshCheese_beatAble_meltable_mixAble` class, which is more specific than `Fre-`

	Substituted ingredient	Applied actions	Failure
Eggplant parmigiana	Ricotta	add	
	Mozzarella	cut	×
Pasta gratin with etc.	Mozzarella	grate	×
Pasta garden pie	Mozzarella	shred, sprinkle	×
	Ricotta	stir	
Tex mex lasagna	NonFatCottageCheese	mix	
No-fuss lasagna	Mozzarella	shred	×
	Ricotta	mix	
Lasagne a la baroque	Mozzarella	slice, add	×
3-step veggie pizza	CreamCheese	sprinkle	
Southwest stew	QuesoFresco	crumble	×

Table 3. List of actions applied to the substituted ingredients of recipes returned by TAAABLE$_0$ with tomato and mascarpone. A cross in the failure column indicates adaptation failure because an action cannot be applied upon the substitution ingredient.

`shCheese`. This class represents the fresh cheeses that can be beaten, melted, and mixed, actions that are applicable to mascarpone.

The analysis of the results presented in Fig. 7 shows that:

- TAAABLE$_1$ returns only 4 answers, while TAAABLE$_0$ returns 8 answers (including the 4 answers of TAAABLE$_1$ with different substitutions). The generalisation is less sharp in the case of the refined ontology than with the initial one.
- The substitutions proposed by TAAABLE$_0$ are more variable than the ones proposed by TAAABLE$_1$. With TAAABLE$_1$, mascarpone is substituted for ricotta or cream cheese, while TAAABLE$_0$ additionally proposes to substitute mascarpone for mozzarella, non-fat cottage cheese or *queso fresco*. And the larger the set of substituting ingredients, the more likely the adaptation is to fail, because there is less similarity between the ingredients.

Examining the proposed substitutions recipe by recipe allows to evaluate the adaptation success or failure. If the adaptation fails, the reason for failure is identified. Table 3 gives, for each recipes of TAAABLE$_0$, the actions that were applied to the substituted ingredient. If the action cannot be applied to the substituting ingredient — in our case, the mascarpone — the adaptation fails. These failures are marked with a cross in the *Failure* column. There are 3 clear cases of failure, for the following recipes:

- *Pasta gratin with etc.* in which the mascarpone should be grated;
- *Lasagne a la baroque* in which the mascarpone should be sliced;
- *Southwest stew* in which the mascarpone should be crumbled.

There are also three *half*-failures for answers in which TAAABLE$_0$ proposes to choose the ingredient that has to be substituted. So, adapting the *Pasta garden* recipe may be considered as a success if the mascarpone replaces the ricotta, but as a failure if the mascarpone replaces the mozzarella and that it must be shredded. It is the same for the *No-fuss lasagna* recipe, where substituting ricotta succeeds while substituting mozzarella

	Substituted ingredient	Applied actions	Failure
Pasta garden pie	Ricotta	mix	
No-fuss lasagna	Ricotta	mix	
3-step veggie pizza	CreamCheese	sprinkle	
Eggplant parmigiana	Ricotta	add	

Table 4. List of actions applied to the substituted ingredients of recipes returned by TAAABLE$_1$ with tomato and mascarpone. A cross in the failure column indicates adaptation failure because an action cannot be applied upon the substitution ingredient. For TAAABLE$_1$, there is in fact no such failure.

fails, and for the *Eggplant parmigiana* recipe where substituting ricotta succeeds while substituting mozzarella fails. To conclude, 6 of the 11 substitutions fail with TAAABLE$_0$ on this example.

Table 4 gives, for each recipe of TAAABLE$_1$, the actions that must be applied to the substituted ingredient, a failure being marked with a cross. We see that the refined ontology improves the final result as none of the failed adaptations proposed by TAAABLE$_0$ are proposed by TAAABLE$_1$: there is no case of either clear nor half-failure. However, one recipe of TAAABLE$_0$ (*Tex-mex lasagna*) for which the adaptation succeeds is not proposed by TAAABLE$_1$ because of the increased distance between `NonFatCottageCheese` and `Mascarpone` in TAAABLE$_1$'s ontology. This results from the stop condition of the algorithm of TAAABLE which runs through the generalisation space. This problem can be solved by asking, in the system interface, to further search similar recipes beyond the current generalisation, at the risk of introducing again failed adaptations.

6 Conclusion

This paper shows how a domain ontology refinement immediately affects the retrieval process of a CBR system and how it improves the final result of the TAAABLE system. Our approach enables a more progressive retrieval of source cases and, within the framework of TAAABLE, some adaptation failures are eliminated. Beyond the results of our approach, illustrated by a concrete example, this work raises some issues that must be addressed.

A first issue is the supervision of the set of properties taken into account for the refinement. In this work, this set has been limited manually because there are potentially more than 250 cooking actions. Selecting a priori some properties (cooking actions) that discriminate the ingredients limits unwanted aggregations. However, as the use context of an ingredient may change, it is not impossible, that given a set of actions A_1, an ingredient i be closer to an ingredient i_1 and, given another set of actions $A_2 \neq A_1$, i be closer to $i_2 \neq i_1$. So, for one context, ricotta could be closer to mascarpone and, in another context, closer to cottage cheese.

Another issue is case indexation. Each (recipe) case is actually indexed by its ingredients, regardless of how they are used. Improving the case representation by indexing

them by the ingredients jointly with their transformations is another way to eliminate some of the adaptation failures of this type.

Finally, another point planned in order to improve the adaptation in TAAABLE is to manage action sequences and guarantee that the actions are applicable after the adaptation, by using pre-conditions based on domain knowledge: ingredient consistency (*solid*, *liquid*, *soft...*), ingredient sensory properties (*sweet*, *salted...*), etc. This knowledge makes it possible to check whether, after the adaptation, the recipe is *cookable*.

Acknowledgements. The authors wish to thank the reviewers for their helpful comments and suggestions for future work.

References

1. S. Asiimwe, S. Craw, N. Wiratunga, et B. Taylor. Automatically Acquiring Structured Case Representations: The SMART Way. In *Applications and Innovations in Intelligent Systems XV*. Springer, 2007.
2. F. Badra, R. Bendaoud, R. Bentebitel, P.-A. Champin, J. Cojan, A. Cordier, S. Després, S. Jean-Daubias, J. Lieber, T. Meilender, A. Mille, E. Nauer, A. Napoli, et Y. Toussaint. Taaable: Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In *ECCBR Workshops, Workshop of the First Computer Cooking Contest*, pages 219–228, 2008.
3. F. Badra, J. Cojan, A. Cordier, J. Lieber, T. Meilender, A. Mille, P. Molli, E. Nauer, A. Napoli, H. Skaf-Molli, et Y. Toussaint. Knowledge Acquisition and Discovery for the Textual Case-Based Cooking system WikiTaaable. In *Workshops of the 8th International Conference on Case-Based Reasoning*, 2009.
4. R. Bendaoud, A. Napoli, et Y. Toussaint. Formal Concept Analysis: A unified framework for building and refining ontologies. In Aldo Gangemi et Jérôme Euzenat, editors, *16th International Conference on Knowledge Engineering and Knowledge Management - EKAW 2008*, volume 5268, pages 156–171, Acitrezza, Catania Italie, 2008. Springer Berlin / Heidelberg.
5. R. Bergmann et A. Stahl. Similarity measures for object-oriented case representations. In B. Smyth et P. Cunningham, editors, *Fourth European Workshop on Case-Based Reasoning, EWCBR-98*, Lecture Notes in Artificial Intelligence 1488, pages 25–36. Springer, 1998.
6. P. Cimiano, A. Hotho, et S. Staab. Learning concept hierarchies from text corpora using formal concept analysis. In *Journal of Artificial Intelligence Research (JAIR'05)*, volume Volume 24, pages 305–339. AAAI Press, 2005.
7. B. Díaz-Agudo et P. A. González-Calero. Classification Based Retrieval Using Formal Concept Analysis. In D. W. Aha et I. Watson, editors, *Case-Based Reasoning Research and Development — Fourth International Conference on Case-Based Reasoning (ICCBR-01)*, LNAI 2080, pages 173–188, 2001.
8. V. Dufour-Lussier, J. Lieber, E. Nauer, et Y. Toussaint. Text adaptation using formal concept analysis. In *Case-Based Reasoning Research and Development (proceedings of ICCBR-2010)*, volume 6176 of *Lecture Notes in Artificial Intelligence*, pages 96–110. Springer, 2010.
9. B. Ganter et R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1999.
10. G. Stumme et A. Maedche. FCA-MERGE: Bottom-Up Merging of Ontologies. In *17th International Joint Conferences on Artificial Intelligence (IJCAI'01)*, pages 225–234, San Francisco, CA, 2001. Morgan Kaufmann Publishers, Inc.
11. L. Szathmary et A. Napoli. CORON: A Framework for Levelwise Itemset Mining Algorithms. *Supplementary Proc. of The Third International Conference on Formal Concept Analysis (ICFCA '05), Lens, France*, pages 110–113, 2005.