Case-Based Cooking with Generic Computer Utensils: Taaable Next Generation

Emmanuelle Gaillard, Jean Lieber, Emmanuel Nauer

Case-Based Cooking with Generic Computer Utensils: Taaable Next Generation

Emmanuelle Gaillard, Jean Lieber, and Emmanuel Nauer

Université de Lorraine, LORIA — 54506 Vandœuvre-lès-Nancy, France
CNRS — 54506 Vandœuvre-lès-Nancy, France
Inria — 54602 Villers-lès-Nancy, France
firstname.lastname@loria.fr

Abstract. This paper presents the participation of the Taaable team in the 2014 Computer Cooking Contest. The three challenges proposed this year are addressed. The *basic challenge* is addressed with a new version of the Taaable system, built on Tuuurbine, a generic case-based reasoning system over RDFS. The *mixology challenge* which requires building recipes only by using a set of available foods is also directly addressed using Tuuurbine conjointly with Revisor, an adaptation engine implementing various revision operators. For the mixology challenge, Revisor is used to compute ingredient substitutions and to adjust the ingredient quantities. The taste score is evaluated as the probability that the adapted cocktail is tasty, depending on the probabilities that the retrieved recipe and the adaptation performed are good. The text of the preparation is adapted using textual substitutions. Finally, the *originality challenge* addresses reasoning on knowledge built collaboratively by an e-community, taking into account the reliability of knowledge units.
Keywords: case-based reasoning, belief revision, adaptation of ingredient quantities, adaptation of recipe preparations, knowledge reliability

1 Introduction

This paper presents the participation of the Taaable team in the three challenges of the 2014 Computer Cooking Contest (CCC): the *basic challenge*, the *mixology challenge*, and the *originality challenge*.

The Taaable team has successfully participated in the previous CCCs, winning in 2010 the main challenge and the adaptation challenge. The systems developed by the Taaable team in the framework of the CCCs are based on many methods and techniques in the area of knowledge representation (RDFS, belief revision, linear algebra), knowledge management (data mining) and natural language processing (text adaptation). All these techniques are detailed in [**?**] and have led to the creation of two generic tools: Tuuurbine[1], a generic case-based reasoning (CBR) system over RDFS [**?**] and Revisor[2], an adaptation engine based on the use of revision operators [**?**]. A third tool, MKM (for

[1] http://tuuurbine.loria.fr
[2] http://revisor.loria.fr

meta-knowledge model) [**?**] is under study, to manage the reliability of knowledge coming from an e-community and to reason on it.

For this edition of the CCC, three systems have been developed on the basis of TUUURBINE with some interactions with REVISOR for particular adaptations and MKM for managing knowledge reliability. This paper details these systems: Sections 2, 3 and 4 explain respectively how the basic, the mixology, and the originality challenges are addressed.

2 Basic challenge

The basic challenge consists in proposing, according to a set of initial recipes, one or more recipes matching a user query composed of a set of wanted ingredients (at least one) and (optionally) a set of unwanted ingredients. This challenge is the core challenge of the CCC since its first edition.

This year, the TAAABLE system was totally redesigned, as an instantiation of the generic CBR TUUURBINE system [**?**]. TUUURBINE implements a generic case-based inference mechanism in which adaptation consists in retrieving similar cases and in replacing some features of these cases in order to adapt them as a solution to a query. Searching similar cases is based on a generalization/specialization process taking into account generalization costs, and adaptation rules. The whole knowledge (cases, domain knowledge, costs, adaptation rules) is stored in a RDF store.

2.1 Tuuurbine founding principles

TUUURBINE is a generic CBR system over RDFS.[3] The domain knowledge is represented by an RDFS base `DK` consisting of a set of triples of the form $\langle C \ \texttt{subClassOf} \ D\rangle$ where C and D are classes which belong to a same hierarchy (e.g, food hierachy). Fig. 1 represents the domain knowledge for the running examples by a hierarchy whose edges $C \xrightarrow{x} D$ represent the triples $\langle C \ \texttt{subClassOf} \ D\rangle$ The retrieval knowledge is encoded by a cost function, associating to a triple $\langle C \ \texttt{subClassOf} \ D\rangle \in \texttt{DK}$ a positive real number $\texttt{cost}(\langle C \ \texttt{subClassOf} \ D\rangle)$. If $C \xrightarrow{x} D$ is an edge of the figure 1 hierarchy then $\texttt{cost}(\langle C \ \texttt{subClassOf} \ D\rangle) = x$. This cost can be understood intuitively as the measure of "the generalization effort" from C to D.

A TUUURBINE case `case` is described by a set of triples of the form $\langle URI_{\texttt{case}} \ \texttt{prop} \ \texttt{val}\rangle$, where $URI_{\texttt{case}}$ is the URI of `case`, `val` is either a resource

[3] RDF and RDFS are W3C recommendations. RDF (*Resource Description Framework*) represents data about resources by the way of triples of resources $\langle$*subject predicate object*$\rangle$, where the resource *predicate* is a property. A URI (*Unique Resource Identifier*) is an identifier of a resource. RDFS gives some semantics—and thus, inference possibilities—to RDF by the mean of inference rules associated to some resources. For example, the property `subClassOf` is related to the inference rule $\dfrac{\langle C \ \texttt{subClassOf} \ D\rangle \quad \langle D \ \texttt{subClassOf} \ E\rangle}{\langle C \ \texttt{subClassOf} \ E\rangle}$ i.e., if C is a subclass of D which is a subclass of E then C is a subclass of E.

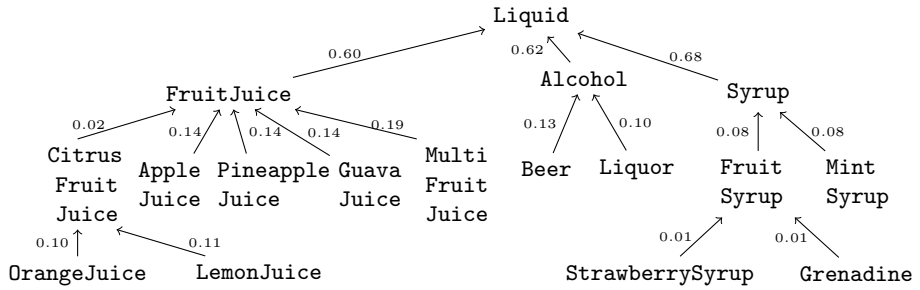

Fig. 1. The hierarchy forming the domain knowledge used in the running example with the generalization costs used as retrieval knowledge.

representing a class of the ontology or a value and `prop` is an RDF property linking `case` to a hierarchy class or to the value. For simplification, in this paper, we represent a case by a conjunction of expressions only of the form `prop : val`. For example, the "Cocktail mexicanos" recipe, is represented by the following index `R`, which means that "Cocktail mexicanos" is a cocktail recipe made from beer, strawberry syrup and pineapple juice (`ing` stands for *ingredient*).

$$\begin{aligned} \texttt{R} = \ & \texttt{dishType} : \texttt{CocktailDish} \\ & \wedge \texttt{ing} : \texttt{Beer} \wedge \texttt{ing} : \texttt{StrawberrySyrup} \wedge \texttt{ing} : \texttt{PineappleJuice} \end{aligned} \tag{1}$$

For instance, the first conjunct of this expression means that the triple $\langle URI_{\texttt{case}}$ `dishType CocktailDish`$\rangle$ belongs to the knowledge base.

2.2 Tuuurbine query

A TUUURBINE query is a conjunction of expressions of the form `sign prop : val` where `sign` $\in \{\epsilon, +, !, -\}$, `val` is a resource representing a class of the ontology and `prop` is an RDF property belonging to the set of properties used to represent cases. For example,

$$\begin{aligned} \texttt{Q} = \ & + \texttt{dishType} : \texttt{CocktailDish} \\ & \wedge \texttt{ing} : \texttt{Beer} \wedge \texttt{ing} : \texttt{PineappleJuice} \wedge \ ! \texttt{ing} : \texttt{Liquor} \end{aligned} \tag{2}$$

is a query to search "a cocktail with beer, pineapple juice but without liquor".

The signs ϵ (*empty sign*) and $+$ are "positive signs": they prefix features that the requested case must have. $+$ indicates that this feature must also occur in the source case whereas ϵ indicates that the source case may not have this feature, thus the adaptation phase has to make it appear in the final case.

The signs ! and $-$ are "negative signs": they prefix features that the requested case must not have. $-$ indicates that this feature must not occur in the source case whereas ! indicates that the source case may have this feature, and, if so, that the adaptation phase has to remove it.

2.3 Tuuurbine retrieval process

The retrieval process consists in searching for cases that best match the query. If an exact match exists, the corresponding cases are returned. For the query `Q` given in (2), the "Cocktail mexicanos" recipe is retrieved without adaptation. Otherwise, the query is relaxed using a generalization function composed of one-step generalizations, which transforms `Q` (with a minimal cost) until at least one recipe of the case base matches $\Gamma(Q)$.

A one step-generalization is denoted by $\gamma = \texttt{prop} : \texttt{A} \rightsquigarrow \texttt{prop} : \texttt{B}$, where `A` and `B` are classes belonging to the same hierarchy with $\texttt{B} \sqsupseteq \texttt{A}$, and `prop` is a property used in the case definition. This one step-generalization can be applied only if `A` is prefixed by ϵ or ! in `Q`. If `A` is prefixed by !, thus `B` is necessarily the top class of the hierarchy. For example, the generalization of `!ing : Rum` is `ϵing : Food`, meaning that if rum is not wanted, it has to be replaced by another food. Classes of the query prefixed by + and − cannot be generalized.

Each one-step generalization is associated with a cost denoted by $cost(A \rightsquigarrow B)$. The generalization Γ of `Q` is a composition of one-step generalizations γ_1, $\ldots \gamma_n$: $\Gamma = \gamma_n \circ \ldots \circ \gamma_1$, with $cost(\Gamma) = \sum_{i=1}^{n} cost(\gamma_i)$. For example, for :

$$\begin{aligned} \texttt{Q} = {} & +\texttt{dishType}:\texttt{CocktailDish} \\ & \wedge \texttt{ing}:\texttt{Beer} \wedge \texttt{ing}:\texttt{Grenadine} \wedge \texttt{ing}:\texttt{PineappleJuice} \wedge !\texttt{ing}:\texttt{Liquor} \end{aligned} \tag{3}$$

`Grenadine` is relaxed to `FruitSyrup` according to the domain knowledge of Fig. 1 . At this first step of generalization, $\Gamma(\texttt{Q}) = \texttt{dishType}:\texttt{CocktailDish} \wedge \texttt{ing}:\texttt{Beer} \wedge \texttt{ing}:\texttt{FruitSyrup} \wedge \texttt{ing}:\texttt{PineappleJuice} \wedge !\texttt{ing}:\texttt{Liquor}$, which matches the "Cocktails mexicanos" recipe described in (1).

2.4 Tuuurbine adaptation process

When the initial query does not match existing cases, the cases retrieved after generalization have to be adapted. The adaptation consists of a specialization of the generalized query produced by the retrieval step. According to $\Gamma(\texttt{Q})$, to `R`, and to `DK`, the ingredient `StrawberrySyrup` is replaced with the ingredient `Grenadine` in `R` because `FruitSyrup` of $\Gamma(\texttt{Q})$ subsumes both `StrawberrySyrup` and `Grenadine`. TUUURBINE implements also an adaptation based on rules where some ingredients are replaced with others in a given context [**?**]. For example, in cocktail recipes, replacing `PineappleJuice` and `StrawberrySyrup` with `OrangeJuice` and `Grenadine` is an example of an adaptation rule. This rule based adaptation is directly integrated in the retrieval process by searching cases indexed by the substituted ingredients for a query about the replacing ingredients, for example by searching recipes containing `PineappleJuice` and `StrawberrySyrup` for a query about `OrangeJuice` and `Grenadine`.

Fig. 2 presents the TAAABLE interface running on query (3). The cocktail recipe "Cocktail mexicanos" which contains `StrawberrySyrup`, `Beer`, `PineappleJuice` and not `Liquor` is retrieved. TUUURBINE suggests to replace `StrawberrySyrup` with `Grenadine`.

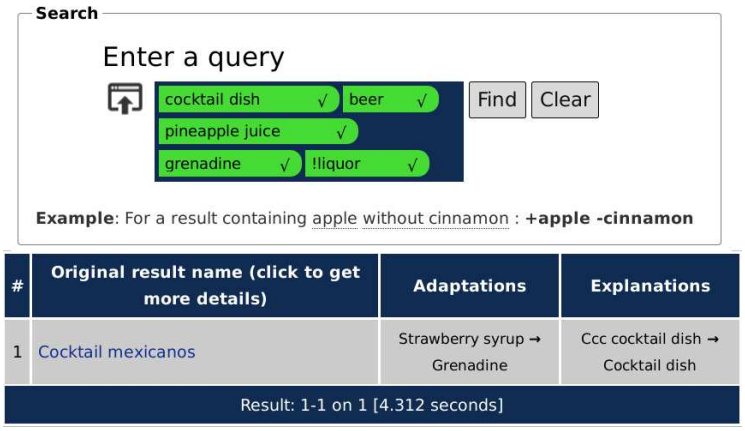

Fig. 2. Example of adaptation in the TAAABLE interface.

2.5 Using Tuuurbine to build the new Taaable system

The TAAABLE knowledge base is WIKITAAABLE[4], the knowledge base made available for this CCC edition. WIKITAAABLE is composed of the four classical knowledge containers: (1) the domain knowledge contains an ontology of the cooking domain which includes several hierarchies (about food, dish types, etc.), (2) the case base contains recipes described by their titles, the dish type they produce, the ingredients that are required, the preparation steps, etc., (3) the adaptation knowledge takes the form of adaptation rules as introduced previously, and (4) the retrieval knowledge, which is stored as cost values on subclass-of relations and adaptation rules.

In WIKITAAABLE, all the knowledge (cases, domain knowledge, costs, adaptation rules) is encoded in a triple store, because WIKITAAABLE uses Semantic Media Wiki, where semantic data is stored into a triple store. So, plugging TUUURBINE over the WIKITAAABLE triple store is quite easy because it requires only to configure TUUURBINE by giving the case base root URI, the ontology root URI and the set of properties on which reasoning may be applied.

3 Mixology challenge

The mixology challenge consists in retrieving a tasty cocktail that matches a user query according to a set of available foods.

First, the set of available foods must be managed. A specific interface, called "My fridge", allows to store the set of available foods in a database. Fig. 3 shows that a user may indicate the foods she has, by selecting the available foods in a list or by entering them in an input zone. The food list on the left-hand side contains the initial list of available foods proposed by the CCC organizers.

[4] `http://wikitaaable.loria.fr/`

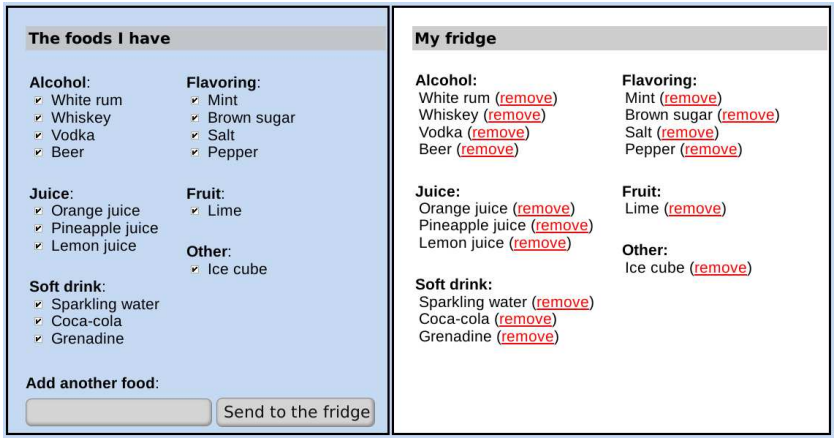

Fig. 3. The "My fridge" interface allowing the user to select the set of available foods.

Second, the system must provide recipes using only the available foods. The expressiveness of TUUURBINE queries can express this kind of request using the ϵ and ! prefixes. Section 3.1 explains how the user query is transformed to take into account only the available foods. For the mixology challenge, TUUURBINE is only used to retrieve cases. The retrieved cases are then adapted using REVISOR/PL (see Section 3.2) to compute the ingredient substitutions, and using REVISOR/CLC (see Section 3.3) to adapt quantities. The adaptation of the text of the preparation is explained in Section 3.4 and the automatic evaluation of the taste is described in Section 3.5.

3.1 Query building

For the mixology challenge, where an answer must only contain the available foods, the query may be built by adding to the initial user query the minimal set of classes of the food hierarchy that subsume the set of foods which are not available, each class being negatively prefixed by !. For example, assume that `AppleJuice` and `PineappleJuice` are the only available fruit juices, that `Beer` is the only available alcohol, that `MintSyrup` and `Grenadine` are the only available syrups, and that the user wants a cocktail recipe with `Beer` but without `SugarCane`. The initial user query will be $\texttt{Q} = +\texttt{dishType}:\texttt{CocktailDish} \wedge \epsilon\texttt{ing}:\texttt{Beer} \wedge !\texttt{ing}:\texttt{SugarCane}$. According to Fig. 1, `CitrusFruitJuice`, `GuavaJuice`, `StrawberrySyrup` and `Liquor` will be added to this initial query with a ! for expressing that the result cannot contain one of these non available classes of food, which includes their descendant classes. The extended query `EQ` submitted to TUUURBINE will be:

$$\texttt{EQ} = \texttt{Q} \wedge !\texttt{ing}:\texttt{CitrusFruitJuice} \wedge !\texttt{ing}:\texttt{GuavaJuice} \wedge !\texttt{ing}:\texttt{StrawberrySyrup} \wedge !\texttt{ing}:\texttt{Liquor}$$

For this example, TUUURBINE retrieves the "Cocktail mexicanos" recipe with the adaptation "replace `StrawberrySyrup` with `Food`", due to `!ing:StrawberrySyrup`.

In order to replace `StrawberrySyrup` by something more specific than `Food`, REVISOR/PL is used.

3.2 Revisor/PL

REVISOR/PL is a belief revision engine in propositional logic. It can be used to adapt a source case to answer a query according to the revision-based adaptation principle [**?**]. Let $\dot{+}$ be a revision operator. The $\dot{+}$-adaptation is defined as follows:

$$\texttt{CompletedTarget} = (\texttt{RevisorDK} \wedge \texttt{RevisorCase}) \dot{+} (\texttt{RevisorDK} \wedge \texttt{RevisorQuery})$$

Intuitively, the source case is modified minimally so that it satisfies the query. The source case, the query and the domain knowledge are represented in propositional logic. Replacing strawberry syrup in the "Cocktail mexicanos" recipe is formalized for REVISOR/PL as follows:

$$\begin{aligned}
\texttt{RevisorCase} &= \texttt{StrawberrySyrup} \\
\texttt{RevisorQuery} &= \neg\texttt{StrawberrySyrup} \\
\texttt{RevisorDK} &= (\texttt{Liquid} \Leftrightarrow \texttt{Alcohol} \vee \texttt{Syrup} \vee \texttt{Juice}) \\
&\quad \wedge (\texttt{FruitSyrup} \Leftrightarrow \texttt{StrawberrySyrup} \vee \texttt{Grenadine}) \\
&\quad \wedge (\texttt{Syrup} \Leftrightarrow \texttt{FruitSyrup} \vee \texttt{MintSyrup}) \\
&\quad \wedge (\texttt{Alcohol} \Leftrightarrow \texttt{Beer}) \\
&\quad \wedge (\texttt{Juice} \Leftrightarrow \texttt{AppleJuice} \vee \texttt{PineappleJuice})
\end{aligned}$$

`RevisorCase` is a conjunction of propositional variables, each of them corresponding to the classes of `EQ` which have been generalized in `Food` by TUUURBINE. `RevisorQuery` is the intersection between classes generalized in `Food` by TUUURBINE and all the classes that do not include, according to the specialization relation, a class representing a food available in the fridge. Classes are prefixed with a positive sign in `RevisorCase` and prefixed with a negative sign in `RevisorQuery`.

Only the part of the domain knowledge used during the reasoning is translated into propositional logic. So, building `RevisorDK` depends on the classes which are in the source case, in the query and in the set of available foods. Although `MintSyrup`, `Grenadine` and `StrawberrySyrup` are not the only syrup in the ontology, in the formalization they are the only ones: `StrawberrySyrup` is the only syrup in `RevisorCase` and `RevisorQuery`, and `Grenadine` and `MintSyrup` are the only syrups available in the food list.

`RevisorDK` is a conjunction of terms $\texttt{ParentClass} \Leftrightarrow \bigvee_i \texttt{DirectSubClass}_i$ meaning that a `ParentClass` implies to have one of its $\texttt{DirectSubClass}_i$ and, conversely, a $\texttt{DirectSubClass}_i$ implies to have its `ParentClass`. For example, the first line of `RevisorDK` consists in asserting that `Alcohol`, `Syrup` or `Juice`

are `Liquid` and, conversely, the only available `Liquid` are `Alcohol`, `Syrup` or `Juice`.

The result returned by REVISOR/PL is:

$$(\texttt{RevisorDK} \wedge \texttt{RevisorCase}) \dot{+} (\texttt{RevisorDK} \wedge \texttt{RevisorQuery})$$
$$\equiv \texttt{RevisorDK} \wedge \texttt{RevisorQuery} \wedge \texttt{Grenadine}$$

The substituting classes of `EQ` being generalized in `Food` by TUUURBINE are classes in REVISOR/PL that do not have descendants in the ontology; and which are also in the set of available foods. In the example, the substituting class for `StrawberrySyrup` is `Grenadine`.

3.3 Revisor/CLC

The result of REVISOR/PL can be read as ingredient type substitutions, but does not manage the quantities of the ingredients of the resulting recipe. This is performed by REVISOR/CLC, another adaptation by revision engine of the REVISOR library, in the form of conjunction of linear constraints (on real numbers and integers). The same engine (with no name at that time) was used for the same purpose in the TAAABLE system presented in the 2010's CCC, so, since this paper concentrates on new aspects of TAAABLE, only a short presentation of this quantity adaptation is given, through an example.

The recipes are formalized as conjunctions of equality constraints, e.g.:

$$\texttt{RevisorCase} = (\texttt{Beer}_g = 12) \wedge (\texttt{Champagne}_g = 0)$$
$$\wedge\, (\texttt{PineappleJuice}_g = 12) \wedge (\texttt{StrawberrySyrup}_g = 3)$$

where i_g is the mass in grams of ingredient i. Since REVISOR/PL has indicated the substitution from beer to champagne, beer has to be removed, indicated by

$$\texttt{RevisorQuery} = (\texttt{Beer}_g = 0)$$

Finally, the domain knowledge is expressed as a conjunction of constraints of the form $i_g \geq 0$ (i.e., the mass of i is nonnegative) and of constraints indicating how to compute the total amount of sugar, the volume, the alcohol quantity, etc. For the latter, the constraint is:

$$\texttt{totalAlcohol}_g = 0.055 \times \texttt{Beer}_g + 0.12 \times \texttt{Champagne}_g$$

(other ingredients involved in this adaptation process do not have alcohol).

Let `RevisorDK` be the conjunction of all these constraints. Then, REVISOR/CLC revises $\texttt{RevisorDK} \wedge \texttt{RevisorCase}$ by $\texttt{RevisorDK} \wedge \texttt{RevisorQuery}$, providing a new conjunction of constraints making `RevisorQuery` precise and tending to the preservation of sugar, volume and alcohol. In this example, this adaptation amounts to the following substitutions (making more precise the result of REVISOR/PL), converted in volumes:

$$12\,\text{cl}\ \texttt{Beer} \rightsquigarrow 5.5\,\text{cl}\ \texttt{Champagne}$$
$$12\,\text{cl}\ \texttt{PineappleJuice} \rightsquigarrow 19.6\,\text{cl}\ \texttt{PineappleJuice}$$
$$3\,\text{cl}\ \texttt{StrawberrySyrup} \rightsquigarrow 1.9\,\text{cl}\ \texttt{StrawberrySyrup}$$

Note that the quantities of pineapple juice and strawberry syrup have been altered by REVISOR/CLC in order to compensate the change in volume and sugar when replacing beer with champagne.

3.4 Adaptation of the textual preparation

Replaced ingredients in the ingredient list of a recipe have also to be replaced in the textual preparation of the recipe. Text occurrences of the replaced ingredients are substituted with the replacing ingredients. A set of rules allows to identify plurals of the ingredient in the text, and replace with the plural form of the replacing ingredients. For example, the textual preparation of "Cocktail mexicanos": "In a *beer* glass, add the strawberry syrup. Complete with *beer*, ..." is adapted to "In a *champagne* glass, add the strawberry syrup. Complete with *champagne*, ...", according to the substitution of `Beer` to `Champagne`.

3.5 Automatic taste evaluation

Our taste score relies on the probability that the adapted cocktail is tasty. This probability depends on the probability that the retrieved recipe is good and on the probability that the adaptation is good. Without additional information about how the recipes of the CCC base are good, we consider them all as being good and thus, the probability of being a good recipe is always 1. Therefore, the probability that the adapted cocktail is tasty depends only on the adaptation. The probability that an adaptation is good relies on the adaptation cost. As this adaptation cost varies on the interval $[0, +\infty[$, the probability of Γ to be a good adaptation is computed as $P(\Gamma \text{ is good}) = e^{-\texttt{cost}(\Gamma)}$. This probability is finally multiplied by 5 and rounded to get a tasty score on a $0-5$ scale.

For example, for $\texttt{Q} = +\texttt{dishType:CocktailDish} \wedge \texttt{ing:Vodka} \wedge \texttt{ing:SparklingWater} \wedge \texttt{ing:BrownSugar} \wedge !\texttt{ing:Mint}$, TUUURBINE suggests the "Gin fizz easy" cocktail (containing `SparklingWater`, `Gin`, `SugarCane`, `Lime` and not `Mint`) and to replace `Gin` with `Vodka` and `SugarCane` with `BrownSugar`. On the retrieval step, `Vodka` has been generalized to `Liquor` with $\texttt{cost}(\texttt{Vodka} \rightsquigarrow \texttt{Liquor}) = 0.061$ and `BrownSugar` has been generalized to `Sugar` with $\texttt{cost}(\texttt{BrownSugar} \rightsquigarrow \texttt{Sugar}) = 0.011$. The taste score is $round(e^{-(0.061+0.011)} \times 5) = 5$.

4 Originality challenge

For the originality challenge, we propose to improve the CBR system TAAABLE by taking into account knowledge reliability. MKM (meta-knowledge model) is a model for managing reliability of the knowledge units that are used in the reasoning process. For this, MKM uses meta-knowledge such as belief, trust and reputation, about knowledge units used by TAAABLE and about the e-community users of TAAABLE. This set or meta-knowledge can be used to compute a reliability score for each knowledge unit (KU). The reliability score in MKM is used

both to select relevant knowledge to conduct the reasoning process, and to rank results provided by the CBR engine.

Let `reliability` be the function which returns the reliability score of a KU *ku* for the e-community, computed from quality, trust, and reputation scores.

4.1 Plugging the MKM on Tuuurbine

As proposed in [**?**], MKM may be used to extend an existing CBR system by adding a filtering process and a ranking process.

Filtering is used to select the most *reliable* set of knowledge according to the query. All the KUs with a reliability score higher than a given threshold are selected to be used by the CBR engine (the KUs with a reliability score lower than the threshold are not used by the CBR engine as if there were removed from the knowledge base).

Ranking computation is used to order the set of results according to the meta-knowledge associated to the KUs involved in the computation of the results. For each result `R`, an *inferred reliability*, denoted by `inferred_reliability(R)` is computed. The inferred reliability of a result is the probability that the result will be satisfactory (e.g., for a cooking application, the probability that this is the recipe of a tasty dish); this probability depends on the probability that the retrieved case is satisfactory, that each KU used in the adaptation is satisfactory, and we assume that these probabilities are independent one from another. Each probability is equivalent to the reliability score of the KU. Computation of the reliability score is detailed in [**?**].

4.2 Model implementation

The MKM has been integrated with aTaaable, a French version of Taaable, on which an e-community has been constituted. aTaaable contains knowledge units, created and evaluated on the Web, by the aTaaable users. With MKM, each knowledge unit is associated to a reliability score. Because the set of CCC recipes and the domain ontology are equivalent in the French version and the English version, reliability scores of knowledge units in the French Taaable is used by the English Taaable system.

4.3 Example of results using Taaable with and without the MKM

Let $\mathrm{CBR_s}$ (*s* for standard) be the system Taaable without the MKM and $\mathrm{CBR_r}$ (*r* for reliability) be the system Taaable with the MKM. Let `Q` = `Grenadine` $\wedge$ `GuavaJuice` be the query. Table 1 presents the 3 first recipes returned by $\mathrm{CBR_s}$ and $\mathrm{CBR_r}$ with their original ingredients, their reliability scores and their adaptation ids which corresponds to an adaptation id of Table 2, that presents adaptations involved in the retrieved recipes.

In this example, adaptations have been computed by the generalization-specialization process for the query. Please refer to [**?**] for details about generalization-specialization and similarity measures between classes.

The first three results of $\mathrm{CBR_s}$ are, in this order:

id	name	$idx(R_i)$	system	reliability	adaptation
$\mathtt{R}_1$	Bora bora	`AppleJuice` $\wedge$ `PineappleJuice` $\wedge$ `LemonJuice` $\wedge$ `Grenadine`	CBR_r + CBR_s	0.76	A_2
$\mathtt{R}_2$	Tequila sunrise	`Tequila` $\wedge$ `OrangeJuice` $\wedge$ `Grenadine`	CBR_r	0.73	A_3
$\mathtt{R}_3$	Bacardi cocktail	`Rum` $\wedge$ `Grenadine` $\wedge$ `LemonJuice`	CBR_r	0.72	A_4
$\mathtt{R}_4$	Spice shoot	`Grenadine` $\wedge$ `Tabasco` $\wedge$ `Vodka`	CBR_s	0.73	A_5
$\mathtt{R}_5$	MTS cocktail	`Martini` $\wedge$ `MultifruitJuice` $\wedge$ `TripleSec` $\wedge$ `CaneSugarSyrup`	CBR_s	0.40	A_1

Table 1. The first three recipes returned by CBR_s and CBR_r for $\mathtt{Q}$ = `Grenadine` $\wedge$ `GuavaJuice`, with their reliability scores and their adaptation ids in Table 2.

id	adaptation	reliability
$\mathtt{A}_1$	`MultifruitJuice` $\rightsquigarrow$ `GuavaJuice`	0.52
$\mathtt{A}_2$	`PineappleJuice` $\rightsquigarrow$ `GuavaJuice`	0.38
$\mathtt{A}_3$	`OrangeJuice` $\rightsquigarrow$ `GuavaJuice`	0.35
$\mathtt{A}_4$	`LemonJuice` $\rightsquigarrow$ `GuavaJuice`	0.33
$\mathtt{A}_5$	`Vodka` $\rightsquigarrow$ `GuavaJuice`	0.11 (filtered)

Table 2. Adaptations of the recipes of Table 1, with their reliability score.

1. $\text{S}_{\text{s}1}$: Spice shoot with adaptation `Vodka` $\rightsquigarrow$ `GuavaJuice`;
2. $\text{S}_{\text{s}2}$: Bora bora with adaptation `PineappleJuice` $\rightsquigarrow$ `GuavaJuice`;
3. $\text{S}_{\text{s}3}$: MTS cocktail with adaptation `MultifruitJuice` $\rightsquigarrow$ `GuavaJuice`.

The first three results of CBR_r are, in this order:

1. $\text{R}_{\text{s}1}$: Bora bora with adaptation `PineappleJuice` $\rightsquigarrow$ `GuavaJuice`;
2. $\text{R}_{\text{s}2}$: Tequila sunrise with adaptation `OrangeJuice` $\rightsquigarrow$ `GuavaJuice`;
3. $\text{R}_{\text{s}1}$: Bacardi cocktail with adaptation `LemonJuice` $\rightsquigarrow$ `GuavaJuice`.

$\text{S}_{\text{s}1}$ is not returned by CBR_r because even if the case entitled "Spice shoot" has a reliability score of 0.73, the adaptation knowledge `Vodka` $\rightsquigarrow$ `GuavaJuice` has been filtered because its low reliability score.

The first result returned by CBR_r is $\text{R}_{\text{s}1}$ which is only the second result returned by CBR_s. The inferred reliability of $\text{R}_{\text{s}1}$ is 0.29 corresponding to the products of the reliability score of its case and its adaptation. The inferred reliability of $\text{S}_{\text{s}2}$ is 0.26 and the inferred reliability of $\text{S}_{\text{s}3}$ is 0.24.

This example illustrates that KUs which are not reliable (e.g., $\langle$`FruitJuice subClassOf Vodka`$\rangle \in \mathtt{DK}$) are not used by CBR_r and that a case with a *high* reliability adapted thanks to a long generalization/specialization

path[5] can be better ranked than a case with a *low* reliability adapted by a shorter generalization/specialization path (e.g., S_{s3} vs. R_{s3}).

5 Conclusion

This paper has presented the three systems developed by the TAAABLE team for its participation in the three challenges of the 2014 Computer Cooking Contest. The three systems are based on many new tools developed by the TAAABLE team based on the experience acquired in the previous contests: a generic CBR system over RDFS called TUUURBINE, an adaptation engine implementing various revision operators called REVISOR, and a meta-knowledge model called MKM that manage the reliability of knowledge acquired on the Web by an e-community.

References

1. A. Cordier, V. Dufour-Lussier, J. Lieber, E. Nauer, F. Badra, J. Cojan, E. Gaillard, L. Infante-Blanco, P. Molli, A. Napoli, and H. Skaf-Molli. Taaable: a Case-Based System for personalized Cooking. In Stefania Montani and Lakhmi C. Jain, editors, *Successful Case-based Reasoning Applications-2*, volume 494 of *Studies in Computational Intelligence*, pages 121–162. Springer, January 2014.
2. E. Gaillard, J. Lieber, E. Nauer, and A. Cordier. How case-based reasoning on e-community knowledge can be improved thanks to knowledge reliability. In *International Conference on Case-Based Reasoning*, 2014. In press.
3. J. Cojan and J. Lieber. Applying belief revision to case-based reasoning. In Henri Prade and Gilles Richard, editors, *Computational Approaches to Analogical Reasoning: Current Trends*, pages 133–162. Springer, 2014.
4. E. Gaillard, J. Lieber, Y Naudet, and E. Nauer. Case-Based Reasoning on E-Community Knowledge. In *21st International Conference on Case-Based Reasoning*, volume 7969, pages 104–118. Springer, 2013.
5. E. Gaillard, L. Infante-Blanco, J. Lieber, and E. Nauer. Tuuurbine: A Generic CBR Engine Based On RDFS. In *International Conference on Case-Based Reasoning*, 2014. In press.
6. E. Gaillard, J. Lieber, and E. Nauer. Adaptation knowledge discovery for cooking using closed itemset extraction. In *The Eighth International Conference on Concept Lattices and their Applications - CLA 2011*, pages 87–99, 2011.
7. E. Gaillard, J. Lieber, Y. Naudet, and E. Nauer. Case-based reasoning on e-community knowledge. In *Case-Based Reasoning Research and Development*, pages 104–118. Springer Berlin Heidelberg, 2013.

[5] the generalization/specialization path between two concepts C and D is the minimal set of edges linking C to D.