Chapter 9

CookIIS – A Successful Recipe Advisor and Menu Creator

Alexandre Hanft, Régis Newo, Kerstin Bach,
Norman Ihle, and Klaus-Dieter Althoff

Intelligent Information Systems Lab
University of Hildesheim, Germany
surname@iis.uni-hildesheim.de

Abstract. CookIIS is a successful Case-Based Reasoning web application that recommends and adapts recipes or creates a complete menu regarding to the user's preferences like explicitly excluded ingredients or previously defined diets. The freely available application CookIIS won the 2nd Computer Cooking Contest (CCC) in 2009 after winning the Menu Challenge at the 1st Computer Cooking Contest in 2008. The chapter explains the realisation of CookIIS starting with the requirements of the first CCC until the final CCC'09 version. CookIIS uses a an industrial strength CBR tool, the empolis Information Access Suite (e:IAS). However, it goes beyond the standard way of building a CBR application based on e:IAS. This chapter will describe the CookIIS system in detail, especially the knowledge modelling, case representation and adaptation processes.

1 Introduction

The cooking domain, especially finding the right recipes for entertaining guests or just preparing dinner for someone's family, is very common, so everybody experiences those problems once in a while. CookIIS is an example that explains how Artificial Intelligence (AI) methods can be applied to enrich everybody's life. It is easy to understand the knowledge model because everyone knows most of the ingredients from her/his own kitchen and many people have an (at least basic) understanding of the cooking domain. Nevertheless only a few are gifted chefs. Most of us sometimes need some help and suggestions. Everybody is interested in new recipes when it comes to cooking something with given ingredients.

Our project CookIIS that is described here is a CBR-based recipe search engine. Roughly spoken, it searches in a case base for suitable recipes with possible ingredients that are given by a user. According to this information CookIIS also considers ingredients the user dislikes or belong to a certain diet. If recipes with unwanted ingredients are retrieved, CookIIS offers adaptation suggestions to replace these ingredients.

S. Montani & L.C. Jain (Eds.): Successful Case-Based Reasoning Appl., SCI 305, pp. 187–222.
springerlink.com

The Computer Cooking Contest is an annual competition with the aim of comparing different technologies and teams facing the same challenge and attracting more people to AI with an easy-to-understand example. At this competition computer programs have to find and adapt recipes according to users wishes. The Computer Cooking Contest offers a set of recipes that we use for testing, evaluating and further developing our system. In 2008 and 2009 the tasks went from retrieval of recipes, a negation challenge and adaptation challenges to the creation of a complete three course menu from given ingredients.

CookIIS uses a very detailed domain model, that is described first in [1]. It was created using the empolis:Information Access Suite (e:IAS) [2]. E:IAS provides a rule engine for two kinds of rules that are heavily used by CookIIS. The first, *completion rules*, enhance cases and queries as well as they restrict the result before the retrieval. The second, *adaptation rules*, modify cases after the retrieval.

The second part of this chapter describes the Computer Cooking Contest, the motivation behind it and the results of the two competitions that took place until 2009. Section 3 gives an overview of e:IAS and its possibilities as well as an outline of the common architecture of applications built with e:IAS. Next, we describe in section 4 the requirements regarding the Computer Cooking Contest and how we meet them in CookIIS with the knowledge modelling, the case representation, the similarity measure definition, the determination of recipe's origin and the handling of diets as well as the menu creator. We proceed in section 5 with the explanation of three different adaptation approaches used in CookIIS. Afterwards, in section 6, we depict the design and technique of CookIIS's web-based GUI and its feedback component. We finish with a conclusion, discussing related work and give an outlook on future work.

2 Computer Cooking Contest

This section describes the aim, progress and outcome of the Computer Cooking Contest. Furthermore, we present the performance of *CookIIS* during the first competition in 2008 and second competition in 2009.

The Computer Cooking Contest(CCC) was announced in autumn 2007 and took place the first time at the ECCBR 2008 in Trier. The motivation behind this competition is the comparison between different tools, technologies and teams facing the same challenge, to foster new developments in Artificial Intelligence (AI), to attract more students to AI, especially CBR, and to have an attractive and easy-to-understand example for people from outside AI. The variation or adaptation of recipes according to some restrictions as a kind of human creativity is essential for good cooking [3] and it is a challenging issue for AI, especially CBR, research. The CCC is proposed as an open competition, not restricted to any technology. The only restriction is the given recipe base.

After a successful qualification, the software systems met in a real competition and were evaluated with respect to scientific/technical quality by researchers and with respect to the culinary quality by a real chef. The scientific and technical quality subsumes the technical originality of the approach, the usability of the software, as well as their maintainability and scalability. The chef reviewed the offered recipes whether they are appropriate to the query, tasty, cookable and creative[1].

Each year the Computer Cooking Contest consists of the main discipline, *Compulsory Task*, and at least two additional challenges. In 2008 a *Negation Challenge* was offered which fosters the handling of unwanted ingredients. This was replaced by the *Adaptation Challenge* since 2009. Until 2009 the second challenge was the *Menu Challenge*. In 2010 this challenge were replaced by an *open challenge* that focuses on various challenges arising from CBR research[2]. These problems are: working with numeric quantities, considering adaptation as a planning task or discovering adaptation knowledge (from the web).

2.1 First Computer Cooking Contest in 2008

Six teams attended the finale of the first CCC in September 2008. In an one-and-half-hour show all teams had to demonstrate live in public how their systems performed with the contest queries. It was accompanied by a workshop where each team presented the technical and scientific details of their project.

The system "What's in the fridge?" awarded in the Compulsory Task as *European Champion* on Computer Cooking. They used an active learning technique to grab the necessary meta data [4]. Team "TAAABLE" [5] was the biggest team from three French universities and got the European Vice Champion title. The Team of "ColibriCook" [6] captured a victory in the *Negation Challenge* with a suggestion of grapefruit and white wine instead of the forbidden lemon in the original recipe with poached fish. CookIIS (former CCCIIS) achieved the third place in the *Compulsory Task* and attained the *Menu Challenge*. They won with a Chinese menu composed of a Ginger Apple Salad, Chinese Steamed Pork Dumplings (Shiu Mai) and (not perfect) a cereal bar as a dessert.

2.2 Second Computer Cooking Contest in 2009

For 2009 a different and larger recipe base was provided. However, the structure and contained information are comparable to recipe base of 2008. According to the good results of the participants in 2008 the handling of unwanted ingredients was now assumed as a usual feature in the Compulsory

[1] http://www.wi2.uni-trier.de/eccbr08/index.php?task=ccc, last visited 2010-03-18.

[2] http://vm.liris.cnrs.fr/ccc2010/doku.php?id=rules, last visited 2010-03-18.

Task. Instead of the negation challenge the *adaptation challenge* was proposed in 2009. For it a different and smaller set of pasta recipes was provided, where the preparation was divided into numbered steps.

The finale of the 2^{nd} CCC took place jointly with the ICCBR in July 2009 in Seattle, WA, where five teams competed for the awards. *CookIIS* awarded the World Champion on Computer Cooking 2009 (Compulsory Task) with a pizza with leek. Of course, there was no pizza with leek in the recipe base, but CookIIS retrieved a "No Meat Bean Burn Pizza" and suggested to replace the contained onions with the missed leek. The Team from France with their system "WikiTAAABLE" [7] defended the Vice Champion title. The team "CookingCAKE"[8] succeed in the Menu Challenge[3]. Three teams continued their engagement and attended on two finals: (Wiki)Taaable[7], JadaCook(2) [9], and CookIIS. Next CCC is planned during the ICCBR 2010 in Alexandria and the CookIIS team is looking forward to attend the finale as well.

3 Information Access Suite

This section introduces the basics of the empolis Information Access Suite (e:IAS), that is used to build CookIIS. E:IAS [2] is a successful industrial strength software of empolis, an attensity group company (former a Bertelsmann subsidiary). E:IAS is now branded as *empolis research & discovery suite* (EDR) [10]. The remaining part explains in detail the system architecture, the knowledge modelling, the workflow organisation, the similarity measure definition and possibility for retrieval and the rule engine of e:IAS.

The Information Access Suite is a framework to build and configure knowledge-based applications. They are completely JAVA-based (web)client-server applications and is built on top of third-party software like Tomcat. Furthermore, the communication between the components of e:IAS is done via RMI. For the web client Java Server Pages (JSP) is the most powerful technique for accessing the functionality of e:IAS. Nevertheless other techniques like HTML plus CGI for simpler web interfaces are feasible. We use JSP for our web client to realise the web-based GUI of CookIIS that is described more detailed in section 6.

E:IAS is accompanied with a knowledge modelling tool called *Creator* that is based on Eclipse. *Creator* is a GUI tool for the configuration of an e:IAS-based application, which itself is created as an e:IAS project based on one of the delivered project templates. The whole configuration of an e:IAS project is defined in a set of XML-based languages and can be done either by using the Creator or by editing the XML files itself. The most used components of the Creator are the *Model Manager*, the *Data Manager* and the *Pipe Manager*. Others are the Indexer, the Index Spy, the Query Tester, the TextMiner Tester, the Ontology Builder, the DocMarker and the Snippet Editor. The

[3] http://www.wi2.uni-trier.de/ccc09/index.php?task=results, last visited 2009-12-21.

architecture of an e:IAS-based application is explained in the next section as well as the Model Manager and the Pipe Manager in the following sections.

3.1 System Architecture

Figure 1 shows the architecture of a web-based application that has been built using e:IAS. An e:IAS-based application consists of several components and is designed in a 3-tier architecture. *Data sources*, *connector* and *iterator* represent the data layer at the bottom and the *JSP Client* at the top stands for the presentation layer. The remaining components in the middle of the figure belong to the business layer.

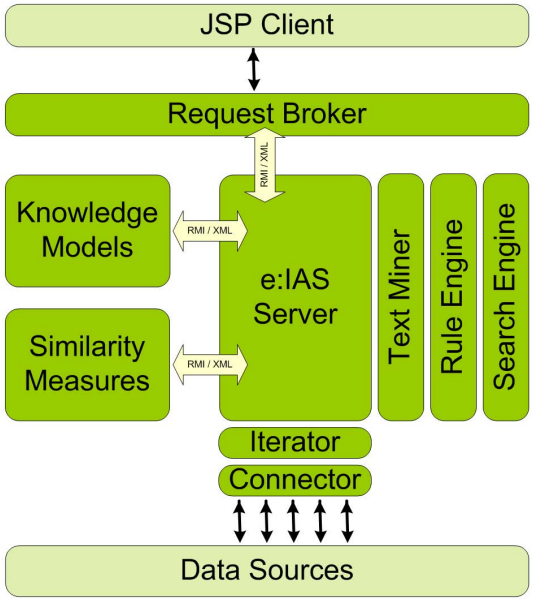

Fig. 1. The architecture of e:IAS used for CookIIS

Connectors are used to connect to the *Data Sources*. With the aid of *Iterators* the data is split into useful parts (e.g. cases). This data flow is controlled by the *e:IAS Server*. *Knowledge Models* are built to define the objectives and concepts the e:IAS project has to deal with. Knowledge Models correspond to the knowledge container vocabulary [11]. The *TextMiner* takes the imported data and extracts modelled concepts.

Furthermore, a *Rule Engine* can transform the data during indexing as well as during retrieval. *Similarity Measures* should be defined to determine the similarity globally between a query and a case as well as locally between attribute values (see section 3.4). The *Search Engine* can be realised through

several kinds of different retrieval techniques (called *Knowledge Server*). This is for instance the *KS/Index* realising a Case Retrieval Net (CRN)[12].

The *Request Broker* accepts requests from a JSP Client and forwards them to the Server, which delegates the search for appropriate cases to the *Search Engine*. Afterwards the Request Broker sends the response back to the client.

3.2 Model Manager

The Model Manager is one of the main components of the *Creator*. It is used to create types and attributes of a case, to configure the analysis mapping and to edit the rules for the completion of meta data and adaptation. The types of attributes are modelled as classes, which themselves are based on predefined types or even text. To allow more than one value in an attribute of a case, sets can be built based on these classes.

Especially for text types all possible terms for concepts in different languages can be added. These terms are considered by the *TextMiner* during stemming the words of free text (in a case or query) to their principal form in order to recognise concepts. For numeric values patterns can be defined to extract numbers from free text.

Additionally, the concepts can be organised in one or more taxonomies. Furthermore, (at least) one special class, an *aggregate*, has to be defined that represents the complete case (case format) with all contained attributes belonging to predefined types or user-defined classes. The query is also an instance of the same aggregate. The definition of concepts, types and aggregate classes belongs to the Knowledge Models. All concepts correspond to the (case) vocabulary in common CBR terminology [11]. Attributes can be assigned to certain views to appear only during the tasks in which they are relevant and necessary. These views can be set in JSP code or in the pipelets (see next section) to transfer only parts of the attributes from a case or query between the e:IAS Server and the JSP Client.

3.3 Workflow Organisation

Two major tasks have to be accomplished by an e:IAS project. The first is extracting data and make it available for searching and the second is searching for solutions according to user requirements. For the first major task the e:IAS project has to tap the data sources, segment the data, recognise the catchwords in the data as well as build up an index structure for the retrieval. The second major task should at least recognise the catchwords in the query and find similar cases. Furthermore, the query could be enhanced with data and the solution might be adapted to fit the query.

All tasks that should be performed by e:IAS are organised with the aid of a workflow model. A workflow here is called *pipeline*. Single steps in such a pipeline are called *pipelets* representing components mentioned in section 3.1 as well as all other executable services. A pipeline establishes a blackboard, where each pipelet can read and write down the objects on it. To run

an e:IAS project a *Process Manager* is started to work off the configured pipelines. More precisely, the e:IAS Server (from figure 1) is an instance of the ProcessManager.

Many predefined pipelets representing distinct services are available. Additionally, user-defined pipelets can be implemented as it is described in section 5.5. Pipelines and both kinds of pipelets can be configured within the Creator. The Creator supports several project templates with preconfigured pipelines. An e:IAS project has at least three pipelines. All pipelines are configured in the *Pipe Manager* component, except the special *Conncect* Pipeline, which is configured in the *Data Manager* component.

The first pipeline writes the data in the retrieval structure and is performed once for each new or changed item of the data sources. Usually this is done for the whole case base while the applications starts. Given by the project template this pipeline is denoted as *InsertCaseProvider* and can be seen on the left hand side in figure 2. It performs (at least) the following tasks each represented by a pipelet:

1. It *connect*s to the data sources, splits them into pieces,
2. It analyses the data (text) to recognise the concepts with the help of the *TextMiner*,
3. It adds, if necessary, additional information to each case with *Completion Rules*, and
4. It *insert*s the cases into the retrieval structure (Knowledge Server with CRN).

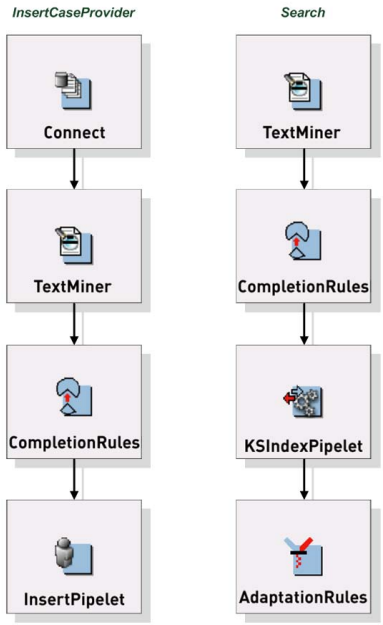

Fig. 2. The two pipelines in an e:IAS project

Connect refers to the special pipeline modelled in the Data Manager. Together with the InsertCaseProvider pipeline they fulfil the first major task.

The third pipeline (standard name is *Search_Pipeline*) is processed for each user request (in CookIIS: each search or recipes) and performs the second major task mentioned at the beginning of this section. This pipeline can be found on the right hand side in figure 2. It

1. Analyses the free text of the query with the help of the *TextMiner*,
2. Adds, if necessary, additional information to the query with *Completion Rules*,
3. Determines the most similar cases from the retrieval structure (*KSIndex-Pipelet*), and
4. Performs adaptation on the retrieved cases with *Adaptation Rules*.

Before the result is sent back to the requesting client. According to other necessary tasks more pipelines can be created and configured.

3.4 Similarity Assessment

The similarity determines appropriate or most fitting cases according to the query. The global similarity between two cases or the query and each case follows the local-global principle [11]. Hence, the similarity is calculated in two steps.

For each class a similarity measure can be defined, which determines (except for aggregates) the local similarity. For numeric types different mathematical functions can be parameterised to define the similarity between two values. For non-numeric types the following symbolic similarity measures are available: table, a (non-configurable) text similarity as well as measures based on a taxonomy and taxonomy-path similarity. Additionally, a combination taking the maximum or minimum value of the aforementioned measures can be used. The *table similarity measure* uses explicitly defined values for pairs of certain values (concepts) and can be initialised by copying the values from other measures.

For the *taxonomy measure* a weight is assigned to each node in the taxonomy with its relative depth within the whole taxonomy (from 0 for the root to 1) and the similarity between two nodes equals the weight of the deepest common parent node.

The *taxonomy-path similarity* also bases on an order and determines the similarity according to the shortest path from the query value to the case value. Therefore it calculates the similarity as the product of a way-up-factor for each step from the query value to the most specific common concept and a way-down-factor for each step down to the case value.

First, the local similarity between values of the same attributes are calculated according to their defined similarity measure. Within an aggregate class

all attributes are assigned to a weight (even 0), which represents the relative (to the weight of other attributes) importance of an attribute for the whole case. Second, the global similarity is subsequently determined as weighted sum of these (local) attribute similarities and their assigned weights.

3.5 Retrieval

E:IAS offers several retrieval engines determining the similar cases according to the above defined similarity: Linear retrieval, index retrieval, full text retrieval, and others that work with SQL databases, XPath or LDAP. *KnowledgeServer/Linear* does not need a retrieval structure but calculates the similarity for each case one after another during each retrieval.

KnowledgeServer/Index is more efficient and appropriate for most purposes. It realises a Case Retrieval Net (CRN)[12] that performs well with large cases and a high amount of cases and can be used for Structured as well as Textual CBR [13].

A Case Retrieval Net [12] is a graph and consists of two types of nodes: *Information Entity* (IE) and *case descriptor*. An IE represents a concept or a particular attribute-value pair. Furthermore, weighted *similarity edges* are in the CRN between similar IE nodes, and *relevance edges* between a case descriptor and all IE nodes contained in this case.

The retrieval starts with the activation of all IE nodes corresponding to IEs found in the query. Second, in the propagation step all IE nodes, that are similar to activated IE nodes, are also activated over the similarity edges with a strength according to the similarity weight. Third, the activation of all IE nodes is accumulated for each case using a relevance function (following the relevance edges). Consequentially, the case (descriptor) with the highest activation represents the most similar case according to the query.

The functionality to insert new cases into the CRN and to retrieve similar cases from them is implemented by two distinct services and represented by two pipelets *KnowledgeServer/Index InsertPipelet* and *KSIndexPipelet*.

3.6 Rule Engine

Rules can be used for automatic modification of the query as well as the cases. Two kinds of rules are available in e:IAS: *completion rules* and *adaptation rules*. *Completion rules* are executed before building the case index or before the retrieval if its corresponding *CompletionRulesPipelet* is inserted into a pipeline (see figure 2). Their purpose is to enhance cases or queries with meta-information or modify attributes. *Adaptation rules* are executed after the retrieval to modify retrieved cases, if their corresponding *AdaptationRulesPipelet* is inserted into a pipeline.

Like in rule-based systems e:IAS rules follow the classic IF ... THEN ... structure. The Creator offers a rule editor with auto-completion support, however it does not detect all syntax errors like misspelled variable names. Both kinds of rule types use the same e:IAS specific syntax, which is stored after the compilation in the format of the Orenge Rule Markup Language (ORML), an XML language. The only difference in syntax between both kinds of rules remains in using a prefix for accessing the query or case.

Completion rules can only access one kind of objects depending on the pipeline: cases during the insertion process and the queries during the search before the retrieval (KnowledgeServer pipelet). Hence, a prefix is neither necessary nor possible. Adaptation rules have, apart from a read access to the query object (with prefix @query), write access to the retrieved cases (prefix @case) since they are executed after the retrieval.

A large amount of predefined functions helps to manipulate single values as well as value sets. Nevertheless, these functions have some limitations which will be explained in more detail using examples in section 5.2.

To control the order of the execution of rules they can be prioritised. To execute different subsets of rules in different pipelines or places in the same pipeline each rule can be assigned to a certain context.

4 The CookIIS Project

CookIIS[4] was initiated following the call for the first Computer Cooking Contest (CCC)[5] in autumn 2007. Although the CCC does not prescribe a certain technique, it seemed that according to the requirements and the context of the participants using case-based reasoning as underlying technique would be (among others) a promising approach.

This section describes how CookIIS meets the requirements of the Computer Cooking Contest by applying its knowledge modelling, the case representation, the similarity measures, the determination of recipe's origin, the handling dietary practices as well as the creation of a three-course menu.

4.1 Requirements of the Computer Cooking Contest

Each year the Computer Cooking Contest was divided into a main discipline *Compulsory Task* and two additional challenges. The requirements on CookIIS arise from the different challenges and the general rules of the CCC as well as the provided data. The Computer Cooking Contest is described in section 2.

The data basis was given as an XML file containing a fixed set of recipes with only three attributes: a title, a list of ingredients as text and preparation instructions as text.

According to this simple XML structure the participants were faced with the tasks of recognising the concepts from pure text and adding all

[4] but called CCCIIS until the first CCC.
[5] http://www.wi2.uni-trier.de/eccbr08/index.php?task=ccc

additional and required information (automatically). Summing up, we collected the following requirements:

- provide an easy-to-use (web-based) user interface,
- suggest at least 5 recipes which fit the user's wishes as best as possible,
- recognise the concepts for food, preparation, cuisine et cetera from a recipe,
- add additional meta information automatically or by hand to the given recipes for the type of meal and cuisine,
- handle unwanted ingredients,
- consider certain diets like low-cholesterol, gout, non-alcoholic, nut-free or vegetarian,
- handle other restrictions like the usage of a seasonal calendar,
- adapt the complete recipe with its preparation steps with a separate set of recipes (*Adaptation Challenge* in 2009), and
- create a three-course menu

We met these requirements with certain parts of e:IAS. When the user provides possible ingredients, CookIIS searches for suitable recipes in a case base. Thereby it considers ingredients the user does not want or those excluded according to certain requested diet. If recipes with unwanted ingredients are retrieved, CookIIS offers adaptation suggestions to replace these ingredients. Besides the retrieval and adaptation of recipes it also provides recipes for a complete three course menu from the given ingredients (and maybe some additional ones).

4.2 Case Representation

Although free text is used as input, the case representation is based on Structured CBR, because this allows a more detailed adaptation than using Textual CBR (TCBR)[14]. We modelled eleven classes for the different kinds of ingredients (basic ingredients, fish, fruit, drinks, meat, milk products, minor ingredients, oil and fat, spices and herbs, supplements and vegetables). Moreover, some classes for additional information (e.g. type of meal, type of cuisine, tools, season et cetera) were created. For these classes we modelled about 2000 concepts (which are possible instances for the ingredient classes) of the cooking domain. Most of the refinements of the modelling for the 2^{nd} CCC in 2009 were based on [15]. Figure 4 exemplifies a part of the taxonomy of the class fruit concerning the modelled nuts. It should be noticed that we do not always follow the botanic point of view, but focus on the cooking domain.

Example: From the botanic point of view almonds and coconuts are not, in contrast to walnuts, real nuts. They are drupes. Nevertheless, users assign almonds and coconuts as nuts. Hence we modelled them as sub-concepts of nuts.

Most concepts of the classes are structured in specific taxonomies to model a hierarchy of (almost) all ingredients. For each ingredient class we built set types to allow multiple values for each ingredient class of the recipe. The attribute names for the ingredients start with Att_ingredients_.

General | Analysis | Dialog | Completionrules | Adaptationrules

Attribute	Type	Weight	View_Input	View_Index	View_Result
Att_File_ID	Text		☐	☐	☐
Att_Text_In	Text		☐	☐	☐
Att_FullText	SetOfFullText	1	☐	☐	☐
Att_File_Type	Text	0	☐	☑	☐
Att_Recipe_Complete	Text		☐	☐	☐
Att_Recipe_Title	SetOfText		☐	☑	☑
Att_Recipe_Ingredients	SetOfText		☐	☑	☑
Att_Recipe_Processing	Text		☐	☑	☑
Att_QueryIn	Text		☑	☑	☑
Att_Forbidden_In	Text	0	☑	☑	☐
Att_Extra_Diet	SetOfTxt_Diet	1	☑	☑	☐
Att_MainDishIn	Text	0	☑	☑	☐
Att_StarterIn	Text	0	☑	☑	☐
Att_DessertIn	Text	0	☑	☑	☐
Att_MainDish_CuisineIn	Text	0	☑	☐	☐
Att_Starter_CuisineIn	Text	0	☑	☐	☐
Att_Dessert_CuisineIn	Text	0	☑	☐	☐
Att_Language	Txt_Language	0	☐	☐	☐
Att_Ingredient_Basic	SetOfTxt_Basic	2	☐	☑	☐
Att_Ingredient_Fish	SetOfTxt_Fish	6	☐	☑	☐
Att_Ingredient_Fruit	SetOfTxt_Fruit	4	☐	☑	☐
Att_Ingredient_Drinks	SetOfTxt_Drinks	1	☐	☑	☐
Att_Ingredient_Meat	SetOfTxt_Meat	6	☐	☑	☐
Att_Ingredient_Milk	SetOfTxt_Milk	2	☐	☑	☐
Att_Ingredient_Minor	SetOfTxt_MinorIngredient	1	☐	☑	☐
Att_Ingredient_OilAndFat	SetOfTxt_OilAndFat	1	☐	☑	☐
Att_Ingredient_SpiceAndHerb	SetOfTxt_SpiceAndHerb	1	☐	☑	☐
Att_Ingredient_Supplement	SetOfTxt_Supplement	3	☐	☑	☐
Att_Ingredient_Vegetable	SetOfTxt_Vegetable	5	☐	☑	☐
Att_Extra_DishCategory	SetOfTxt_DishCategory	4	☐	☑	☐
Att_Extra_MethodOfPreparation	SetOfTxt_Method	1	☐	☑	☑
Att_Extra_Specie	SetOfTxt_Specie	6	☐	☑	☐
Att_Extra_Tool	SetOfTxt_Tool	1	☐	☑	☐
Att_Extra_TypeOfMeal	SetOfTxt_TypeOfMeal	4	☐	☑	☐
Att_Extra_TypeOfMeal_Help_AllIngredients	Text	0	☐	☑	☐
Att_Extra_TypeOfCuisine	SetOfTxt_TypeOfCuisine	3	☐	☑	☐
Att_Extra_Month	Txt_Month	1	☑	☑	☑
Att_Extra_Season	Comp_Season	1	☐	☑	☐
Att_Extra_Season_Info	Text	1	☐	☑	☑
Att_Extra_Diet_AllIngredients	SetOfAtomic		☐	☑	☑
Att_Forbidden_Count	Integer	0	☐	☑	☑
Att_Forbidden_Ingredient_Basic	SetOfTxt_Basic	0	☐	☑	☐
Att_Forbidden_Ingredient_Fish	SetOfTxt_Fish	0	☐	☑	☐
Att_Forbidden_Ingredient_Fruit	SetOfTxt_Fruit	0	☐	☑	☐
Att_Forbidden_Ingredient_Drinks	SetOfTxt_Drinks	0	☐	☑	☐
Att_Forbidden_Ingredient_Meat	SetOfTxt_Meat	0	☐	☑	☐
Att_Forbidden_Ingredient_Milk	SetOfTxt_Milk	0	☐	☑	☐
Att_Forbidden_Ingredient_Minor	SetOfTxt_MinorIngredient	0	☐	☑	☐
Att_Forbidden_Ingredient_OilAndFat	SetOfTxt_OilAndFat	0	☐	☑	☐
Att_Forbidden_Ingredient_SpiceAndHerb	SetOfTxt_SpiceAndHerb	0	☐	☑	☐
Att_Forbidden_Ingredient_Supplement	SetOfTxt_Supplement	0	☐	☑	☐
Att_Forbidden_Ingredient_Vegetable	SetOfTxt_Vegetable	0	☐	☑	☐
Att_Forbidden_TypeOfMeal	SetOfTxt_TypeOfMeal	1	☐	☑	☐
Att_Extra_Exchanges_Text	SetOfText	0	☐	☑	☑

Fig. 3. The most important attributes of the aggregate for a recipe or query together with their type, weight and views

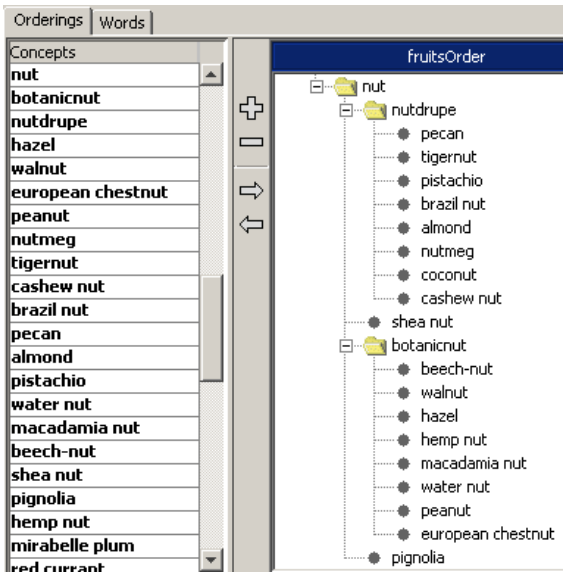

Fig. 4. This is a part of the fruit taxonomy considering the modelling of nuts. Almonds and coconuts are nuts from the user's perspective.

Figure 3 shows the most important (regarding the modelling) attributes of the aggregate for a recipe or query along with their type (second column), weight (third column) and views (last three columns).

Example: The attribute Att_Extra_Exchanges_Text collects all the adaptation advice generated during the adaptation (see section 5), but it is irrelevant for the similarity (weight is zero).

The attributes Att_Query_In as well as Att_Forbidden_In appear in the view "View_Input" and store the user input during a query. The view "View_Result" in the right column in figure 3 marks only those attributes as visible which are used to represent the retrieved recipe to the user, for instance the attribute Att_Recipe_Processing storing the preparation instructions.

Although we had to model a different aggregate class for the *Adaptation Challenge*, we used the same modelled concepts and classes. Indeed, it had one additional, more specific attribute for Pasta, because the focus of this challenge lay on pasta recipes.

4.3 Type of Meal and Type of Cuisine

The determination of the type of meal and cuisine for a recipe was another requirement of the CCC. Therefore we built separate types with corresponding taxonomies to model the different types of meal (e.g. starter, main course, dessert as well as soup, ice cream) and cuisines (e.g. Italian, Chinese,

Mediterranean). For both we used sets of completion rules to add this meta information to the recipes. The type of meal is determined by two main aspects: indicative keywords in the recipe title and indicative (combination of) ingredients [16]. The assignment of the type of cuisine to a recipe follows three aspects (more details in [16]):

1. identification of the recipes origin in the recipe title,
2. identification of characteristic strings or meals in the recipe title, and
3. identification of ingredients (mainly spices and herbs) that are characteristic for a type of cuisine.

4.4 Similarity Assessment

As usual in CBR we defined at first a local similarity measure for each type to define the similarity between two attribute values. Second, we set a global similarity measure to compare the query with a case [11]. For each ingredient class (except *Minor ingredients*) at least one taxonomy was built. Based on those a *taxonomy-based similarity measure* calculates the similarity value between two concepts as the product along the path from one node to the other in the taxonomy. We chose a factor of 0.75 for a generalisation step (upwards) and a slightly higher factor of 0.8 for each specialisation step (downwards). Therefore each parent concept has a similarity value of 0.75, each child concept one of 0.8 and each sibling concept a value of 0.75 * 0.8 = 0.6. A nephew concept has in the same manner a similarity of 0.75 * 0.8 * 0.8 = 0.48 (one step upwards and two steps downwards). The advantage of this approach is that we have to set only two factors, but accordingly each pair of concept and parent concept respectively child concept has the same similarity value.

For pairs that should have an individual similarity value, we built a custom *table similarity measure* to define the similarity between certain pairs explicitly. Afterwards we configured a *combined similarity measure* taking the maximum value of both measures as the local similarity measure for each attribute.

Example: Figure 5 shows the similarity values for all concepts in relation to the concept "nectarine". Following the path from nectarine one step upwards (0.75 for generalisation) and one step downwards (0.8 for specialisation) "peach" has a similarity of 0.75 * 0.8 = 0.6 according to the *taxonomy path similarity measure*. However, in the *table similarity measure* a similarity of 0.9 between "nectarine" and "peach" is defined, hence the *combined similarity* results in 0.9.

For the calculation of the global similarity we assigned a weight to each attribute of the aggregate (or even zero if this attribute is irrelevant for the retrieval). In CookIIS, important attributes in terms of similarity are meat, fish, vegetable and the species of a recipe, which can be seen their weight ≥ 4 in the third column of figure 3.

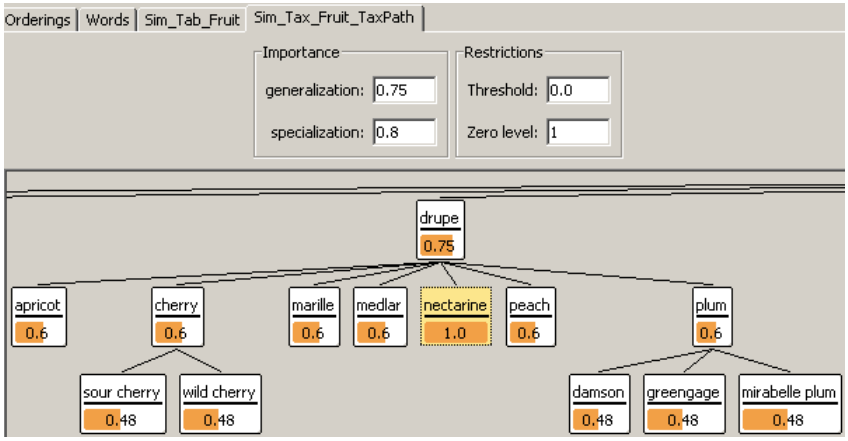

Fig. 5. Part of the taxonomy for fruit showing drupe concepts. This part shows the similarity values based on a *taxonomy path similarity measure* for all concepts with respect to the concept "nectarine". For instance, following the path one step upwards (0.75 for generalisation) and one step downwards (0.8 for specialisation) "peach" has a similarity of 0.8 * 0.75 = 0.6.

Accordingly the global similarity between the query and a case is set as weighted sum of the local similarities as follows:

$$
\begin{aligned}
sim_{global} = \frac{1}{\sum weight_{local\ sim}} \times \\
\Big(6 \times (sim_{meat} + sim_{specie}) + 5 \times sim_{vegetable} \\
+ 4 \times (sim_{fruit} + sim_{typeOfMeal} + sim_{dishCategory}) \\
+ 3 \times (sim_{supplement} + sim_{typeOfCuisine}) \\
+ 2 \times (sim_{fish} + sim_{basicIngr.} + sim_{milk}) + 1 \times \Big(\sum sim_{others}\Big)\Big)
\end{aligned}
$$

It is calculated automatically during the execution of the *KSIndexPipelet*.

4.5 Modelling Dietary Practices

CookIIS allows the user to request recipes according to certain dietary practices, which are currently *vegetarian*, *nut-free*, *non-alcoholic*, *gout*, *low-cholesterol* and *seasonal vegetable*. The first three came from the the first CCC, but we kept them for the second CCC, where the last three were required for the second CCC.

Furthermore, the user has the possibility to exclude one or more ingredients explicitly. We consider both as the same kind of restriction and we denote

such ingredients as *forbidden*. In general we propose (at least) five methods to handle them (based on [17]) guiding our retrieval and adaptation afterwards:

1. Ignore all recipes containing forbidden ingredients
2. Remove the forbidden ingredients from the recipe
3. Replace the forbidden ingredients with other ingredients, according to the following principles:
   a. Replace with similar ingredients
   b. Replace with ingredients according to experience (e.g. from community)
   c. Replace with ingredients according to certain explicit modelled replacements
4. Modify the similarity measure to decrease the similarity of recipes containing forbidden ingredients
5. Inform (the user) that a certain constraint is not met

We chose different ways to handle these restrictions, first depending on the kind of restriction (diet) and second depending on the ingredient category. For method 1 we applied completion rules which extend the query before the retrieval is executed with some additional attribute-values and filters.

Example: If the user chooses a *nut-free* diet, all cases containing the concept "nut" or a child concept should be ignored (see figure 4). For that we set a filter to the concept "nut" in the fruit taxonomy using a completion rule.

For the *non-alcoholic* diet a filter to the concept "alcohol" in the "drink" taxonomy was set to exclude all recipes containing alcoholic ingredients (method 1). For the *vegetarian* diet all cases containing a concept from the taxonomies for meat and fish were excluded, because we cannot offer a good replacement for these main ingredients. Additionally, we marked *minor ingredients* from animal origin as *forbidden* with the aim of replacing them through adequate ones during the adaptation process.

If users choose to use *seasonal* ingredients and certain ingredients of a recipe are not available, we inform them that these ingredients are only available from storage (method 5) [16]. Otherwise, we advice a replacement with another given ingredient. The information, in which months which ingredients are freshly available, are in storage or have to be replaced with valid replacements, was given as a table and transformed into a set of adaptation rules.

Example: If the user asks for a recipe with asparagus and demands the seasonal restriction for August to be considered, a retrieved recipe with asparagus will be enhanced with the hint that fresh asparagus is not available in August (only fresh in June and July). Otherwise, in October and November a replacement with fresh salsify is advised.

A list of recommended and not recommended ingredients is given for the *gout* and the *low cholesterol diet*. A set of completion rules is built that includes all recipes with preferable ingredients and excludes all recipes with non-preferable ingredients (method 1) [16]. Here we could not use our usual similarity-based replacement, because the applicability in case of gout or the

amount of cholesterol in an ingredient is not modelled in the taxonomies. *Example:* Milk that is not recommended for cholesterol diet in contrast to low-fat milk which is a child concept of milk.

4.6 Three-Course Menu Creator

With regard to the Menu Challenge CookIIS is capable to create a three-course menu with a starter, a main course and a dessert. The underlying assumption is that all three dishes shall belong to the same origin. The user can state desired and unwanted ingredients for starter, main course and dessert separately. At first the best-matching main course is retrieved to determine the type of cuisine that is used to restrict the subsequent requests for the starter and the dessert. Some recipes can be equally offered as starter and as main course. However, CookIIS ensures that not the same dish is offered as starter and main dish. If necessary, it offers the second-best instead of the best-matching recipe.

5 Adaptation in CookIIS

Adaptation of solutions from retrieved cases is a central part of case-based reasoning [18]. The importance of a good adaptation increases if the case base is restricted to a small number of cases or if the variation of problems to be solved is very high. Nevertheless, adaptation has not been the important topic in recent years of CBR research [19]. Often adaptation means justifying values in a bounded range [20] and is done via rules created and maintained by a domain knowledge expert or system developer [21]. Compared to the whole variety of favoured ingredients, the provided recipe base is too small. We assume that even a recipe base of a hundred thousand recipes does not cover all possible combinations of wanted and undesirable ingredients. Hence, we need to adapt the given recipes.

This section describes three different kind of adaptations we use in CookIIS. We start with the *model-based* adaptation and afterwards we analyse its shortcomings. According to the shortcomings we investigate by the model-based adaptation we describe the idea, implementation and evaluation of the *community-based* adaptation, which collects adaptation advice from cooking communities on the web. A least we explain how the *in-place* adaptation transfers the adaptation advice from the conceptual level to the text of the recipes and how this is integrated into e:IAS.

We denote ingredients that are excluded by a diet or explicitly by the user as *forbidden* ingredients. If a *forbidden* ingredient occurs in a retrieved recipe we consider it as *critical* (for this recipe) because it has to be omitted or replaced.

Due to the amount of ingredients we cannot model a replacement advice for each single ingredient by hand or by interviewing experts. Our adaptation approaches share a certain generality in the used methods to be able to transfer these methods to other domains. Nevertheless, to assure certain a

level of applicability we restrict the replacements to the same class as the replaced ingredient. Of course, replacements from another ingredient class are imaginable, but in this case we had to spend much effort to gather these pairs of original and replaced ingredient by hand or use other techniques to collect this information from communities.

5.1 Three Kinds of Adaptation

CookIIS features several kinds of adaptation methods. Our standard is the *model-based adaptation* working for all concepts with a similarity measure modelled in CookIIS. According to some shortcomings of this approach [17] we enhanced it with a second approach: the *community-based adaptation*. With this approach we gather adaptation knowledge from internet communities using the domain model that usually exists in structured CBR applications [14]. The third approach, the *in-place adaptation*, is dedicated especially to the adaptation challenge. It changes the ingredient list as well as the preparation instructions directly in the pasta recipes considering also parent concepts and plural or abbreviated forms, whereas both preceding approaches consider only the concept names.

While executing a query unwanted (*forbidden*) ingredients are collected in extra attributes of the case (starting with $Att\_Forbidden\_Ingredient\_$, see figure 3). The components of all three approaches access these attributes to determine which concepts of the recipe are *critical* and should be replaced.

Example: If a user asks CookIIS for tomato, eggplant and chicken but wants neither mushrooms nor beans, then the attribute Att_Ingredient_Meat contains chicken and Att_Ingredient_Vegetable owns tomato and eggplant. The attribute Att_Forbidden_Ingredient_Vegetable is of the same type as the last one and contains mushroom and bean to indicate forbidden vegetables.

Integration of the three Approaches. All three adaptation approaches are executed sequentially and each approach uses the results of the preceding one. First, the community-based adaptation is performed. For those forbidden ingredients where no community-based replacements can be offered, the standard model-based adaptation is carried out secondly. Both approaches only add adaptation advice in an extra text attribute of retrieved recipes but leave all attributes for ingredients and preparation unchanged. In contrast, the third approach changes the ingredient list and preparation text traceably according to the advice of both preceding approaches.

The integration of the three approaches is realised with three different pipelets. A user-defined pipelet for the community adaptation is inserted into the given Search_pipeline before the AdaptationRulesPipelet implementing the model-based adaptation (compare figure 2). The in-place adaptation as third approach is realised as another user-defined pipelet and integrated after the AdaptationRulesPipelet in the same pipeline. All three approaches are described in the next sections.

5.2 Model-based Adaptation

The model-based adaptation is realised by a set of adaptation rules executed by the rule engine of e:IAS after the retrieval of similar cases. Especially for the handling of sets of attributes (multiple values) a couple of set-oriented functions are used. The general scheme of this adaptation approach is to determine *critical* ingredients (according to a certain diet or explicitly unwanted) and replace them with some similar ingredients of the same class. This approach works on the defined similarity measure (and the underlying taxonomies) for each ingredient class. Accordingly, we have the same set of six adaptation rules for each ingredient class.

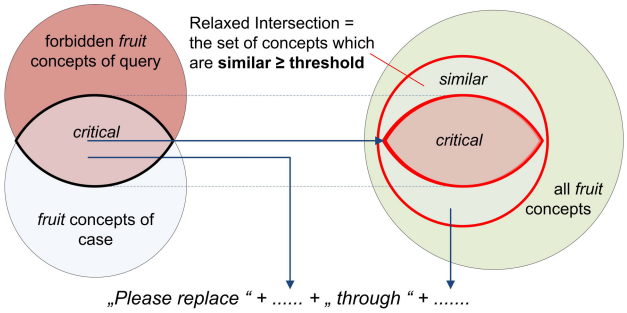

Fig. 6. Scheme for the Model-based Adaptation: Exchange critical ingredients with similar ones (from [22])

We explain the adaptation rules for the ingredient category fruit in figure 6 using set theory. First, the intersection of the set of all *forbidden* ingredients and the set of all fruit ingredients in the retrieved recipe is calculated to determine which ingredients are *critical* and have to be replaced in this recipe (illustrated on the left side). The second step uses a relaxed intersection function that determines for the first set all similar concepts (in virtue of the defined similarity measures) out of a second set above a given threshold. We chose 0.48 as the threshold, which corresponds to the similarity of a nephew concept (e.g. "mirabelle plum" for the "nectarine" in figure 5). Applying this function to the *critical* ingredients and to all (modelled) fruit concepts, we receive all fruits that are similar to the critical ones and can be used as replacement (marked bold in figure 6). This approach follows our adaptation advice described in method 3a (in section 4.5). At the end a new replacement advice is generated.

Example: Having only the *taxonomy-based similarity measure* and looking for similar concepts to replace the *critical* "nectarine", twelve concepts will be returned (all concepts shown in figure 5). Considering that plum is also forbidden, "plum" and its sub-concepts will be excluded.

If the forbidden ingredient cannot be substituted (no similar ingredient found), the system recommends to omit it (method 2 in section 4.5).

Shortcomings of the Existing Model-based Adaptation Approach. Since the used adaptation approach makes use of the modelled taxonomies, the results sometimes lack refinement (first in [17]). For instance the relaxed intersection function returns all sibling concepts to the *critical* concepts as well as parent and child concepts, whereas only the siblings are most appropriate. In most situations, if siblings exist, parents are too unspecific as a replacement suggestion. Moreover, for an unwanted ingredient one or two ingredients as replacement suggestion would be sufficient and preferable compared to all siblings.

Example: If "cucurbit(a)" is undesired, all sub-concepts of them like "butternut squash" or "pumpkin" are not adequate as well as the parent concept "fruit vegetable".

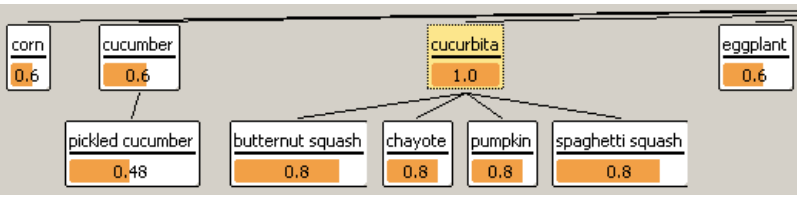

Fig. 7. Sub-concepts of fruit vegetables in the Vegetable taxonomy. The values show the similarity to "cucurbita".

Indeed, the intersection function neither delivers the similarity value of the returned concepts to pick the most similar ones nor does there exist a function to determine the relation of concepts inside taxonomies. Furthermore, the Rule Engine does not provide a function to iterate over sets.

Improvements of the Model-based Adaptation. To overcome the aforementioned drawbacks we added rules to prune parent and child concepts from the result and removed replacement candidates which are forbidden. For that we reviewed the values in all similarity tables and ensured that they are different from taxonomy-based similarity values by increasing them. We determined the child concepts using only the default combined similarity measure by calling the relaxed intersection function twice. First, we called the intersection function with a threshold equal to the similarity value for a child and second, with a threshold slightly above the last threshold. Determining the difference set between these two calls we got the replacement candidates without child concepts. We did the same for grandchildren. Moreover, to avoid suggestions for very common concepts like "fruit vegetable" or "meat" all the time, we cut them out of the set of "all" ingredients during the compilation of the adaptation rules.

Afterwards, ingredients for which replacement candidates are found, are saved in an extra attribute for later processing as well as the ones where no similar replacement candidates are found.

To reduce the amount of offered replacement candidates we could increase the value for some pairs of concepts explicitly (table similarity) and increase the threshold for the minimal similarity accordingly. Thus for each concept only one or two ingredients would have had a similarity above the threshold and would have been recommended. But acquiring these pairs is a time-consuming and expensive task. Facing this effort we had the idea to gather the necessary information from communities as described in the next section.

Additionally, we added rules that look for desired ingredients which are not in the recipe and added another adaptation advice to replace existing ones with desired, similar ingredients. For that they use same intersection function as above to determine whether desired ingredients have similar ingredients from within the recipe.

As a consequence we had six rules for each ingredient class in both aggregates. To realise a data and control flow where some rules depend on the results of preliminary executed rules, we added only for this purpose a couple of auxiliary variables into the case format (aggregate class) and prioritised the rules to control the right order of execution.

A workflow system would be more appropriate than a classic rule set or a combination of both [23], because a rule set with dependent variables is difficult to maintain and more complex than a nested if-then-else control structure with local variables as in usual procedural languages.

5.3 Community-Based Adaptation

The acquisition of knowledge for adaptation (Adaptation Knowledge acquisition: AKA) is an exhausting task because it is highly domain dependent and the required experts are rarely available for acquiring and maintaining the necessary knowledge. Some research on the automatic adaptation knowledge acquisition has been done to tackle this challenge, but it mainly focused on the automatic AKA from cases in the case-base [24, 25, 26].

The WWW, especially the Web 2.0 with its user-generated content, is (beside the case base) a large source of any kind of knowledge and experience. Following the Web 2.0 paradigm of user-interaction people upload their experience, opinions and advice on any kind of topics. Plaza [27] proposes to gather the experience from communities as the main source for future CBR applications.

Although people are not necessarily experts in a domain, the assumption is that the mass of users will correct most of the mistakes as practised for example in the Wikipedia project. Each single post has only a small importance. But if a lot of users submit the same proposal (e.g. suggest to apply the same ingredients replacements), this gives weight to this proposal.

As a consequence, we investigate communities for dedicated replacements for each ingredient without modelling these replacements all by hand (first published [28]). Fortunately, there exist a lot of cooking communities where people post comments to provided recipes expressing (among others) adaptation suggestions of these recipes.

Idea behind the Approach. We used comments that people posted in reply to provided recipes. In these comments users express their opinion on the recipe, before as well as after cooking it. They write about their experience with the preparation process and also tell what they changed while preparing the recipe. With that they express their personal adaptation of the recipe and frequently give reasons for this.

Since this is written down in natural language text, often using informal language, we had the idea to avoid semantic analysis of written sentences and to just find the occurrences of ingredients in the comment texts. For this purpose we used our existing knowledge model from the CookIIS application and the TextMiner provided by e:IAS. Afterwards we compared them to the ingredients mentioned in the actual recipe. We classified them into three classes, depending on whether the ingredients mentioned in a comment appeared in the recipe or not. Thus we use only the class that contains old and new ingredients in a comment as replacement suggestions.

Analysis of Example Cooking Communities. In Germany, *chefkoch.de*[6] is a well known cooking community with a large number of active users. Over 157,000 recipes have been provided so far by the users with an even larger amount of comments on them. The users also have the possibility to vote on the recipes, send them to a friend via email or even add pictures of their preparation. Besides the recipes, *chefkoch.de* features an open discussion board for all kinds of topics on cooking with more than 8.8 million contributions. Their English partner site *cooksunited.co.uk*[7] is unfortunately much smaller with only about 5,000 recipes and 4,400 posts.

But with *allrecipes.com*[8] a comprehensive platform with nearly 44,000 recipes and over 2.4 millions reviews is available in English.

It has comprehensive localizations for the United States, Canada, the United Kingdom, Germany, France and others countries. *Allrecipes.com* explicitly provides variants of an existing recipe. Hence, it also seems to be also a good source candidate. Another large German cooking community is *kochbar.de*[9] with over 261,000 recipes and 176,000 user posts. Besides these large communities a number of smaller communities exist in the Web with more or less similar content. For our approach we decided to use a large German community since the recipes and the corresponding comments are pre-

[6] http://www.chefkoch.de, last visited 2010-03-18.
[7] http://www.cooksunited.co.uk, last visited 2010-03-18.
[8] http://allrecipes.com, last visited 2010-03-18.
[9] http://www.kochbar.de, last visited 2010-03-18.

sented on one page with a standardized HTML-code template, which makes it easier to crawl the site and extract relevant information items. Besides the technical issues other problems could prevent the gathering of experience. Obstacles could be that the owner of the community site prohibits the automatic grabbing of their website or that problems occur if users upload copyright-protected content (e.g. pictures of recipes).

Extraction of Information Items from a Cooking Community. We crawled about 70'000 recipes with more than 280'000 comments from a large German community. We saved the HTML source-code of each web page containing a recipe together with the corresponding comments. From this HTML code we extracted the relevant information snippets using the customized HTML Parser tool[10]. For the recipes these entities were primarily the recipe title, ingredients and the preparation instructions, but also some additional information on the preparation of the recipe (e.g. estimated time for the preparation, difficulty of the preparation, et cetera) and some usage statistics (e.g. a user rating, number of times the recipe has been viewed, stored or printed). We extracted the text of the comments, checked whether the comment was an answer to another comment and whether the comment had been marked as helpful or not. The recipe ID of the related recipe was also saved. We stored all this information in a database for an efficient access.

In the next step, we created another e:IAS-based application and indexed all recipes and comments into two different case bases using a slightly extended CookIIS knowledge model. For each recipe and each comment we extracted the mentioned ingredients and stored them in the case using our knowledge model and the e:IAS TextMiner during the indexing process. Since our knowledge model is bilingual (English and German) we were also able to translate the original German ingredient names from the comment text into English terms during this process and in this way had the same terms in the case bases that we use in our CookIIS application.

Classification of Ingredients. Having built the two case bases, we first retrieved a recipe and then all of the comments belonging to the recipe and compared secondly the ingredients of the recipe with those in the comments. Afterwards, we classified the ingredients mentioned in the comments to the following three categories:

- *New*: ingredients mentioned in the comment, but not in the recipe
- *Old*: ingredients mentioned in the comment as well as in the recipe
- *OldAndNew*: two or more ingredients of one ingredient class, of which at least one was mentioned in the recipe and in the comment and at least one other one was only mentioned in the comment

[10] http://htmlparser.sourceforge.net, last visited 2010-01-10.

We interpreted the classification as follows:

- *New*: New ingredients are a refinement or variation of the recipe. A new ingredient (for example a spice or an herb) somehow changes the recipe in taste or is a try of something different or new.
- *Old*: If an ingredient of a recipe is mentioned in the comment it means that this ingredient is especially liked or disliked (for example the taste of it), that a bigger or smaller amount of this ingredient has been used (or even left out), or there is a question about this ingredient.
- *OldAndNew*: This is either an adaptation (e.g. instead of milk I took cream) or an explanation/specialization (e.g. Gouda is a semi-firm cheese).

The last class is the interesting for the adaptation. For each ingredient classified as *OldAndNew* we also stored whether it was the new or the old one. We distinguished between adaptation and specialization by looking for hints in the original comment text and using the taxonomies of our knowledge model. For that we tried to find terms in the comment during the text-mining process that confirmed if it was an adaptation (e.g. terms like: instead of, alternative, replaced with, ...) and stored those terms in the corresponding case. Additionally, we looked in the taxonomy of the ingredient class whether the one ingredient is a child of the other (or the other way around). We interpreted this as specialization or explanation, because one ingredient is a more general concept than the other. Indeed, we do not consider specializations as adaptation suggestions, because we decline child concepts and parent concepts as replacements. This way we could avoid adaptations like: "Instead of semi-firm cheese take Gouda.".

After assigning the score we aggregated our classification results. We did this in two steps: First we aggregated all classified ingredients of all comments belonging to one recipe. Thereby we counted the number of the same classifications in different comments and added up the score of the same classifications. More details to calculate the score are given in [29]. For instance a specific recipe has 12 comments in which 3 of them mention milk and cream.

Second, we aggregated all classifications without regarding the recipe they belong to. In our dataset we found comments with milk and cream belonging to 128 different recipes. This way we could select the most common classifications out of all classifications. Since we are using a CBR tool and have cases, we also investigate if similar recipes have the same ingredients with the same classification mentioned in the comments. We did this for each recipe first with a similarity of at least 0.9, then with a similarity of 0.8. If many of the same classified ingredients exist in similar recipes, this supports our results.

Usage as Adaptation Knowledge. We use *OldAndNew*-classified ingredients (except specialisations) to generate adaptation suggestions. This can be done in two different ways: independent from the recipe or with regard to the recipe. Considering the first way, we look in the database table for the ingredient that needs to be replaced and use the result where this ingredient is categorized as old and appears in the most recipes or has the highest

	id integer	ingr_class text	oldingr1 text	oldingr2 text	newingr1 text	newingr2 text	score double precis	specification text	card_recipes integer
1	52	Comment_Ingredient_Milk	cream		milk		79.9583333333	adaptation	128
2	63	Comment_Ingredient_Milk	milk		cream		48.2	adaptation	78
3	254	Comment_Ingredient_Supplement	potatoes		sauce		36.1333333333	adaptation	61
4	238	Comment_Ingredient_Supplement	potatoes		broth		31.025	adaptation	54
5	386	Comment_Ingredient_OilAndfat	margarine		butter		29.4166666666	adaptation	46
6	128	Comment_Ingredient_Meat	bacon		ham		28.175	adaptation	45
7	20	Comment_Ingredient_Supplement	noodle		sauce		27.8166666666	adaptation	48
8	127	Comment_Ingredient_OilAndfat	butter		olive oil		27.0333333333	adaptation	43
9	393	Comment_Ingredient_OilAndfat	butter		margarine		25.55	adaptation	41
10	39	Comment_Ingredient_Vegetable	onion		green onion		21.55	adaptation	34
11	230	Comment_Ingredient_Milk	cream		yogurt		20.35	adaptation	31
12	332	Comment_Ingredient_Milk	creme fraiche		smetana		20.15	adaptation	33
13	1072	Comment_Ingredient_SpiceAndHerb	salt		pepper		18.55	adaptation	32
14	160	Comment_Ingredient_Supplement	rice		broth		17.025	adaptation	29
15	21	Comment_Ingredient_Milk	smetana		sour cream		16	adaptation	25
16	785	Comment_Ingredient_Vegetable	onion		paprika pepper		14.725	adaptation	26
17	57	Comment_Ingredient_Milk	cheese		cream		14.525	adaptation	25
18	60	Comment_Ingredient_Vegetable	onion		leek		14.25	adaptation	23
19	64	Comment_Ingredient_Milk	cream		coconut milk		14.1916666666	adaptation	22
20	73	Comment_Ingredient_Drinks	rum		amaretto		13.9	adaptation	22
21	660	Comment_Ingredient_Milk	cream		cheese		13.85	adaptation	24
22	98	Comment_Ingredient_Meat	ham		bacon		13.85	adaptation	22
23	424	Comment_Ingredient_Milk	yogurt		cream		13.775	adaptation	23
24	249	Comment_Ingredient_Meat	ham		salami		13.65	adaptation	21
25	385	Comment_Ingredient_Minor	honey		maple syrup		13.125	adaptation	21

Fig. 8. The most frequent of 6200 suggestions for adaptation gathered from 280.000 comments

score. It is possible to retrieve two or more adaptation suggestions to be more manifold. Using this approach we got more than 6200 different adaptation suggestions of which we only used the most common (regarding the number of appearances in the comments and the score) per ingredient. Figure 8 shows the most frequent of these suggestions. For instance, in the first line a suggestion to replace cream with milk appears in comments to 128 different recipes.

5.4 Evaluation of the Results of the Community-Based Suggestions

Only the ingredient class “supplement” reveals problems that stem from the fact that too many different ingredients are integrated into this class. This can be changed by further improving the knowledge model.

The evaluation can be divided into two different parts. First, we checked if our classification and the interpretation correspond to the intentions written in the original comments. This was done manually by comparing the classification results and their interpretation to the original comments: it matches in most of over 400 tests the classification.

The second evaluation was done on the results of the overall aggregated adaptation suggestions. We examined whether the adaptation suggestions with a high score are good adaptation suggestions for any kind of recipe. We took a representative number of recipes and presented them with adaptation suggestions to chefs. These chefs then rate the adaptation suggestions.

If the adaptation suggestion was applicable, the chefs should rate it as very good, good and practicable. Here again the dependent suggestions perform

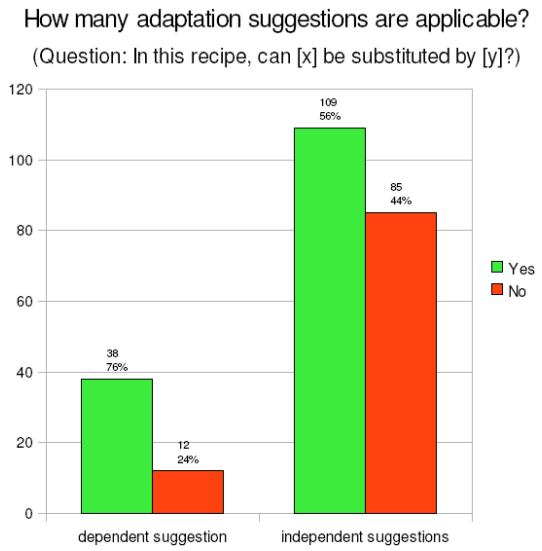

Fig. 9. Applicability of dependent and overall independent suggestion rated by chefs

better, see figure 11. 22% of the dependent suggestions are very good comparing with 18% of the first independent suggestions and 49% compared to 43% are rated as good.

We designed a questionnaire by choosing randomly a recipe and add one adaptation suggestion extracted from comments belonging to that recipe (*dependent*) and secondly add two adaptation suggestions without regard of the recipe (*independent*) each with two ingredients as replacement suggestion. At the end we present the 50 questionnaires with 50 *dependent* pairs and 100 pairs of *independent* ingredients to different chefs, because each chef may have a different opinion. 76% (38 of 50) of the *dependent* and 56% (109 of 194) of the *independent* adaptation suggestions were confirmed as applicable by the chefs (see figure 9). Differentiating the first and second independent suggestion it could be observed that the first one is noteworthy better (see figure 10). Summing up it follows that only by 11 of the 100 independent adaptation suggestions no ingredient can be used as substitution.

After gathering possible replacement candidates we proceed with a way these advice could be better integrated into the retrieved recipes.

5.5 In-Place Adaptation on Recipes (for Adaptation Challenge)

The third approach was especially settled for the pasta adaptation challenge. It changes the retrieved recipes according to the prior advice. Thereby it has to tackle two problems: Simple string matching for the replacement often does not succeed and the original ingredient should be visible if one of the

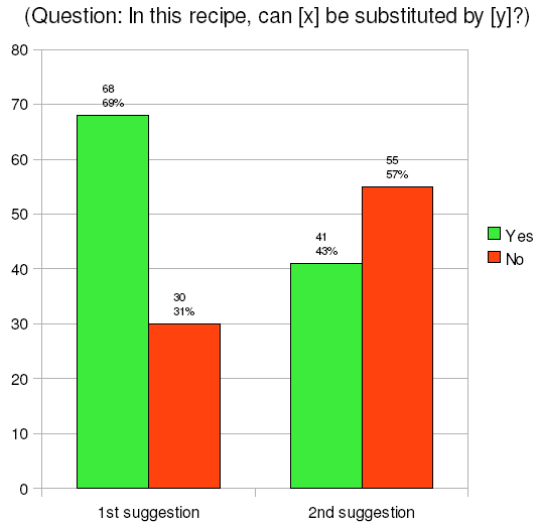

Fig. 10. Applicability of the first and second independent suggestion

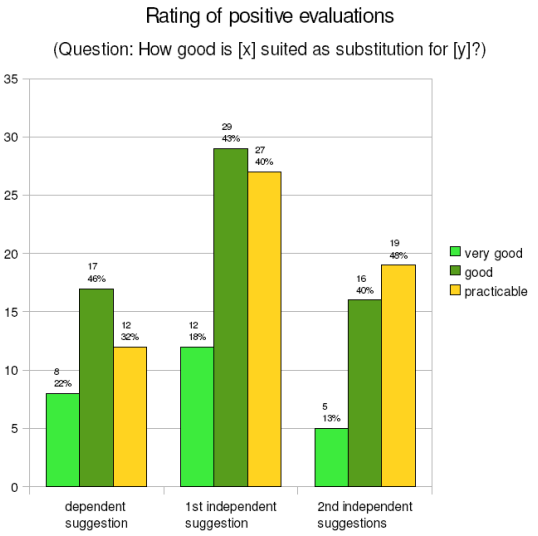

Fig. 11. Ratings of applicable adaptation suggestions

replacement suggestions is not accepted by the user. This is done for each critical concept in two areas of the recipe: in the ingredient list as well as in the preparation description. The replaced and the new text are marked up with HTML tags in order to make changes on the original text of the case traceable. Therefore, if a suggestion is not good as expected, the user can

notice the original ingredient. One possible realisation we implemented is to strike out all occurrences of the forbidden ingredients and insert the first of the replacement candidates in italics behind it.

Although e:IAS is accompanied with a MarkerPipelet, we cannot meet these requirements with the predefined functionality of e:IAS. Hence, we built a customized pipelet and add them as last element to our search pipeline. The outcome and approach is described as follows and afterwards we highlight some details of building customized pipelets.

By investigating the preparation instructions we detect the following difficulties that prevent using a simple string matching: ingredients are mentioned in their (irregular) plural form, refered in an abbreviated form or more general term or described by their synonyms. To identify critical concepts in the given text four consecutive possibilities are tested until one is successful:

1. find the concept name in the text
   a. search and replace plural form (considering -es, -ies, -ves and -oes) and
   b. search and replace singular form,
2. consider common synonyms belonging to certain concepts,
3. if it is a 2-word-concept, replace only the last word, otherwise
4. look for the class name of this concept and replace if exists.

Let us motivate and illustrate this approach with examples.
1. Concepts appear in singular and plural form. At the moment we consider singularia tantum (singular equals plural form) for some words, but no irregular plural forms.
2. Some ingredients are often mentioned as their synonyms (e.g. stock, broth and bouillon). In case of "chicken stock" we also look for "chicken broth" (also together with other kind of meat). However, at the moment we do not look for all modelled synonyms for each concept, but only for the most common ones.
3. We often discovered in the preparation of the given recipes that only the more common parent concept is used. For example, if "olive oil" should be replaced and appears in the ingredient list, but it is possibly referred in the preparation only as "oil". Hence we looked additionally for "oil" after the unsuccessful search for "olive oil" in the preparation.
4. If a recipe lists tomato, paprika and aubergine (eggplant) in the ingredients, they are sometimes mentioned in the preparation as vegetable.

An improvement of step 4 would be to look for the most special occurring parent concept. Figure 12 shows an example where "tomato*es*" (1.a) and "parsley" (1.b) are replaced by "onion*s*" and "basil" respectively. In the preparation only "oil" (3.) occurred and is replaced by "sunflower oil", which is the first of the two replacement candidates for "olive oil" from the community.

Create User-defined Pipelets to implement new Behaviour. To build an own pipelet, a JAVA class must be exported into a package and announced

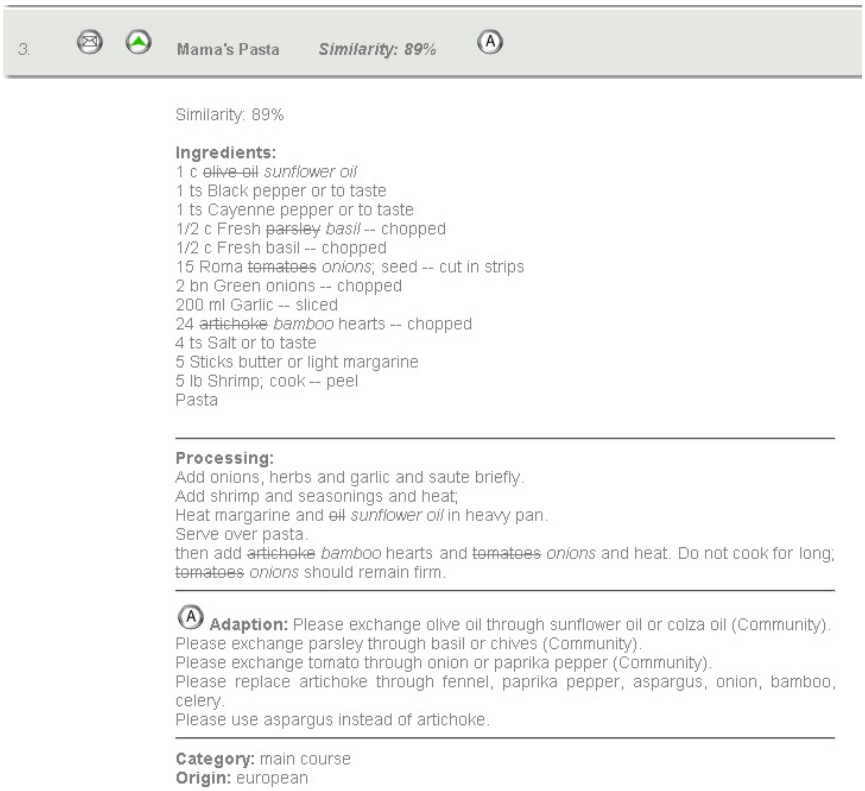

Fig. 12. Replacement of forbidden concepts (crossed out) with new replacements (italicised) in the ingredient list and preparation: sunflower oil replaces olive oil in the ingredient list and only oil in the preparation because olive oil does not exist there.

to the Creator. A pipelet always has to implement a kind of the Service class. To preserve a high flexibility for the functionality and assure that the pipelet can be configured within the Creator as well as executed from the Process Manager the *Adapter Pattern* [30] has to be used. The aim of using an Adapter Pattern is to enable a client (here: the Process Manager) to use an incompatible interface (from our new service). Following to this pattern each pipelet consist of (at least) two classes: adapter (for the configuration) and adaptee (for the functionality). The Pipelet itself has via adapters and iterators access to the case objects (an aggregate) on the blackboard and can read and write almost every attribute.

6 User Interface and Feedback

A well-designed and easy-to use interface is essential for a successful application because that is the only way how the application can be presented to the user. We decided to have a web-based application that is realised with JSP and TagLib libraries. The website does not only present recipes, it includes also a feedback component, which enables interaction with the user.

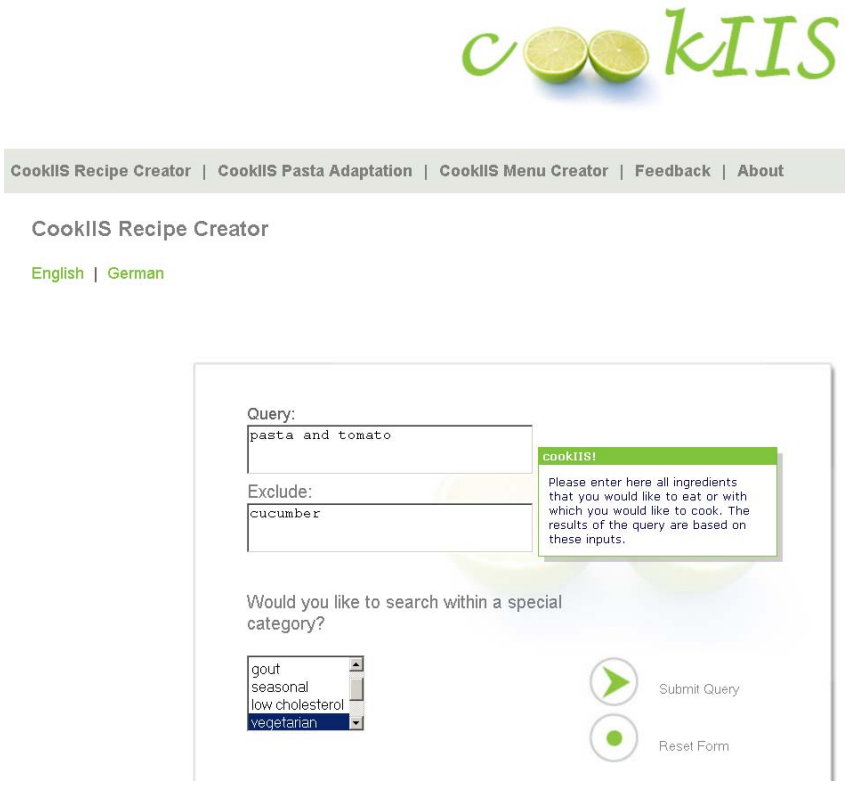

Fig. 13. Screenshot of the CookIIS Recipe Creator

After the 1^{st} Computer Cooking Contest the web-based GUI was completely redesigned to become more user-friendly. The Web Accessibility Initiative (WAI) proposes guidelines for web content accessibility[11]. In similar sense there are of course other guidelines that conduct not only to make websites more accessible but also to attract user (for instance [31]). For our project we took the most important ones for realisation:

[11] http://www.w3.org/TR/WCAG/, Version 2.0 released 11th December 2008.

– clear and easy to understand structure of the site,
– corporate identity with a distinctive design,
– clear design (in colour and layout),
– visualisation with graphics instead of text, and
– possibility to give feedback.

According to the first point the menu structure of the website consist only five items: one for each challenges according to the CCC plus two additional items for feedback and about.

Due to the second point of our guidelines we designed a new logo and complete design with a new colour scheme for CookIIS. The GUI has been designed using only the colours green, light grey and white for a clear and light-weight appearance. Popup boxes explain each element of the GUI. Figure 13 shows the new web GUI of CookIIS.

The website is offered in English and German, therefore all texts of the site are accessed via the standard fmt JSP Tag Library (JSTL). Therefore the JSP Code contains Tags, that are replaced by English or German strings from configured languages files depending on the actually selected language. More details are given in [32].

To meet the last guideline we offer the user a possibility to give feedback in general as well as on a concrete recipe. We did not premise a configured email client but offer an email form to make it as easy as possible for the user. This form is shown in figure 14 and checks the user inputs and the given email address. The functionality for checking and sending email is implemented as a JAVA Bean that can be easily reused for other projects.

The special e:IAS Tag Library allows accessing the functionality of the e:IAS server (also in [1]). To send a query with the user input and present the results, the JSP page has to perform five steps:

1. initialise the RequestManager bean and create a request,
2. set the query, transfer user input into parameters,
3. execute the query (request),
4. fetch the results in local variables, and
5. iterate over the results and display them

The JSP page is reloaded after clicking on the submit button and filled with data starting from the second step. The result of each pipelet including the sent query can be accessed separately to display the results after the retrieval or after the adaptation. According to the CCC the five best-matching recipes are displayed at once and can be expanded on demand. Buttons are offered to navigate to the next or previous five recipes. Each recipe is presented with their original attributes and enhanced with adaptation suggestions if adaptations for the retrieved recipes are necessary. Furthermore, only on the page for the adaptation challenge the original ingredient list and preparation are modified directly (see figure 12).

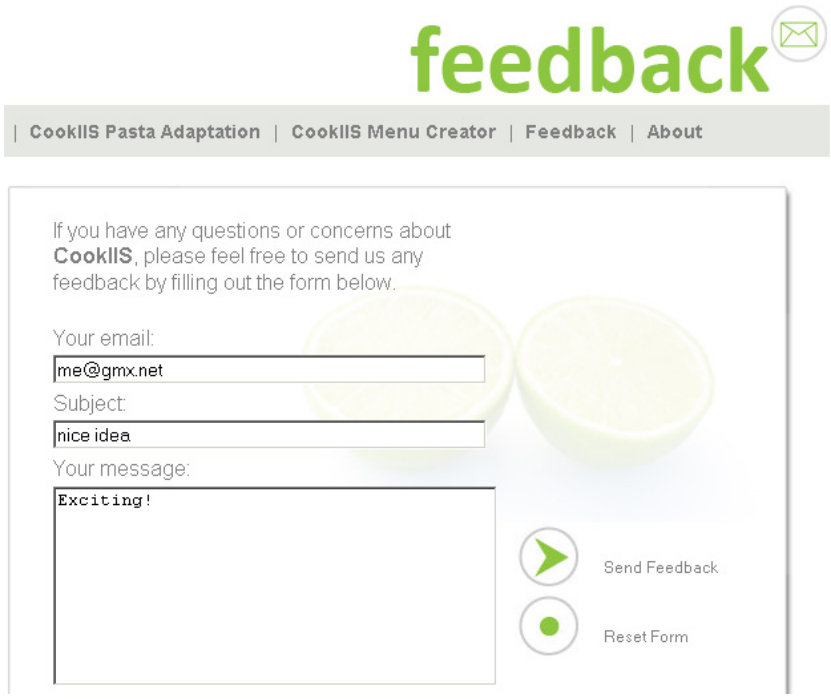

Fig. 14. Screenshot of the CookIIS Feedback form

7 Conclusion, Related Work and Outlook

In this chapter we presented the successful CBR application CookIIS that provides adapted recipes according to the users wishes. It is a web-based client-server software using JAVA technologies and is realised with the empolis Information Access Suite (e:IAS), a framework for building CBR applications. The CookIIS system is equipped with a detailed domain modelling containing over 2000 concepts, a fine-grained rule set to determine the type of cuisine and the type of meal of given recipes.

Following the tasks of the Computer Cooking Contest it was described how CookIIS masters the Compulsory Task, the Adaptation Challenge as well as the Menu Challenge. For each of the CCC challenges CookIIS presents an extra web page. We faced the Adaptation Challenge with modelling an extra aggregate and the ingredient class pasta as well as our in-place adaptation to make the changes directly visible in the recipe.

Furthermore, CookIIS is capable to handle six different kinds of diets and any other ingredient the user explicitly mentions as undesired. This is implemented with a couple of filters and a complex adaptation process. The filters, realised by completion rules, exclude recipes completely from the

result. In contrast, the implemented adaptation process exchanges forbidden ingredients through other adequate ingredients.

CookIIS uses three different kinds of adaptations: a *community-based*, a *model-based*, and an *in-place*. The *community-based approach* collects concrete pairs of ingredients as adaptation advice from comments inside cooking communities on the web. This represents a new approach for validating adaptation knowledge acquired from communities. The appropriateness of this approach was shown with an evaluation by chefs.

For forbidden ingredients, where no replacements at the first step are found, the subsequent *model-based adaptation* is executed and replaces them with similar ingredients. It requires only a designed knowledge model and similarity measures. The last, *in-place adaptation*, applies strategies to find forbidden ingredients that go behind a simple string matching and modify the original text of the recipes traceably according to the replacement suggestions. Whereas the both preceeding approaches only work on the concept level and generate the text of their adaptation advice with a simple sentence template, the in-place works directly on the original free text.

Related Work. Early CBR systems that suggested preparation advice for meals are JULIA [33] and CHEF [34]. CHEF was a planning application that builds new recipes in the domain of Szechwan cooking. JULIA integrated CBR and constraints for menu design tasks. It uses a large taxonomy of concepts and problem decomposition with fixed decomposition plans. Unlike our approach their knowledge was built by experts and was not captured from communities.

The community-based as well as the model-based adaptation approaches presented here belong to the structural adaptation [14]. The idea presented with the community-based adaptation closely relates to the research of Plaza [27]. However, they focus more on gathering cases from web experience instead of adaptation knowledge. Furthermore, acquisition of adaptation knowledge from cases was done by by [25] or with the CABAMAKA System by [26].

The integration of subsequent adaptation approaches, precisely the procedure of looking at first for concrete adaptation suggestions and apply afterwards, if the first step yields no results, more general rules, was done also by [21] with DIAL, which attempts to retrieve adaptation cases first. CookIIS has a couple of competitors (e.g. [4, 5, 6]) facing the same challenges of the Computer Cooking Contest, for instance appropriate similarity measures, recognising ingredient concepts, determining the type of cuisine, handling unwanted ingredients and others.

Outlook. For the near future we will work on the tighter integration of community advice, so that we wo be able to look for adaptation suggestions in recipes that are similar to the one that needs to be adapted. We plan to improve and integrate the *in-place adaptation* within the Compulsory Task in the way that it can be used for all retrieved recipes. Following the SEASALT architecture [35] we also want to realise a multi-agent system that

continuously monitors the community for new experiences with the recipes and enhance our adaptation knowledge if necessary. The CookIIS team has prepared for the third CCC and is looking forward to attend the finale in July 2010 in Alexandria.

Acknowledgements. We would like to thank Franziska Öllerer for her great work on the new design and the better usability as well as the feedback component of the CookIIS GUI.

References

1. Hanft, A., Ihle, N., Bach, K., Newo, R., Mänz, J.: Realising a cbr-based approach for computer cooking contest with e:IAS. In: [36], pp. 249–258
2. empolis GmbH: Technical white paper e:information access suite. Technical report, empolis GmbH (January 2008), `http://www.empolis.com/downloads/download-english/article/white-paper-empolisinformation-access-suite.html` (last verified 2009-11-22)
3. Barham, P.: The Science of Cooking. Springer, Heidelberg (2001)
4. Zhang, Q., Hu, R., Namee, B.M., Delany, S.J.: Back to the future: Knowledge light case base cookery. In: [36], pp. 239–248
5. Badra, F., Bendaoud, R., Bentebitel, R., Champin, P.-A., Cojan, J., Cordier, A., Desprès, S., Jean-Daubias, S., Lieber, J., Meilender, T., Mille, A., Nauer, E., Napoli, A., Toussaint, Y.: Taaable: Text mining, ontology engineering, and hierarchical classification for textual case-based cooking. In: [36], pp. 219–228
6. DeMiguel, J., Plaza, L., Díaz-Agudo, B.: Colibricook: A cbr system for ontology-based recipe retrieval and adaptation. In: [36], pp. 199–208
7. Badra, F., Cojan, J., Cordier, A., Lieber, J., Meilender, T., Mille, A., Molli, P., Nauer, E., Napoli, A., Skaf-Molli, H., Toussaint, Y.: Knowledge acquisition and discovery for the textual case-based reasoning system wikitaaable. In: [37], pp. 259–268
8. Fuchs, C., Gimmler, C., Günther, S., Holthof, L., Bergmann, R.: Cooking CAKE. In: [37], pp. 259–268
9. Herrera, P.J., Iglesias, P., Sánchez, A.M.G., Díaz-Agudo, B.: JaDaCook2: Cooking over Ontological Knowledge. In: [37], pp. 279–288
10. empolis GmbH: Empolis research & discovery, `http://www.empolis.com/applications-services/applications/research-discovery.html` (last verified 2009-11-22)
11. Richter, M.M.: Introduction. In: [38], pp. 1–15
12. Lenz, M.: Case Retrieval Nets as a Model for Building Flexible Information Systems. Dissertation, Mathematisch-Naturwissenschaftliche Fakultät II der Humboldt-Universität zu Berlin (1999)
13. Lenz, M., Hübner, A., Kunze, M.: Textual CBR. In: [38], pp. 115–138
14. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications. LNCS (LNAI), vol. 2432. Springer, Heidelberg (2002)

15. Löbbert, R., Dietlind Hanrieder, U.B., Beck, J.: Lebensmittel: Waren, Lebensmittel, Trends. Verlag Europa-Lehrmittel, Haan-Gruiten (2001)
16. Ihle, N., Newo, R., Hanft, A., Bach, K., Reichle, M.: Cookiis - A Case-Based Recipe Advisor. In: [37], pp. 269–278
17. Hanft, A., Ihle, N., Newo, R.: Refinements for retrieval and adaptation of the CookIIS application. In: Hinkelmann, K., Wache, H. (eds.) GI-TCS 1983. LNI, vol. 145, pp. 139–148 (2009)
18. Kolodner, J.L.: Case-Based Reasoning. Morgan Kaufmann, San Mateo (1993)
19. Greene, D., Freyne, J., Smyth, B., Cunningham, P.: An Analysis of Research Themes in the CBR Conference Literature. In: [39], pp. 18–43
20. Cojan, J., Lieber, J.: Conservative adaptation in metric spaces. In: [39], pp. 135–149
21. Leake, D.B., Kinley, A., Wilson, D.C.: Learning to improve case adaptation by introspective reasoning and cbr. In: Veloso, M.M., Aamodt, A. (eds.) ICCBR 1995. LNCS, vol. 1010, pp. 229–240. Springer, Heidelberg (1995)
22. Hanft, A., Ihle, N., Bach, K., Newo, R.: CookIIS – competing in the first computer cooking contest. Künstliche Intelligenz 23(1), 30–33 (2009)
23. Bali, M.: Drools JBoss Rules 5.0 Developer's Guide. Packt Publishing, Birmingham (2009)
24. Wilke, W., Vollrath, I., Althoff, K.D., Bergmann, R.: A framework for learning adaptation knowledge based on knowledge light approaches. In: 5th German Workshop on CBR, pp. 235–242 (1996)
25. Hanney, K., Keane, M.T.: The adaptation knowledge bottleneck: How to ease it by learning from cases. In: Leake, D.B., Plaza, E. (eds.) ICCBR 1997. LNCS, vol. 1266, pp. 359–370. Springer, Heidelberg (1997)
26. d'Aquin, M., Badra, F., Lafrogne, S., Lieber, J., Napoli, A., Szathmary, L.: Case base mining for adaptation knowledge acquisition. In: Veloso, M.M. (ed.) IJCAI, pp. 750–755. Morgan Kaufmann, San Francisco (2007)
27. Plaza, E.: Semantics and experience in the future web. In: [39], pp. 44–58 (invited talk)
28. Ihle, N., Hanft, A., Althoff, K.-D.: Extraction of adaptation knowledge from internet communities. In: [37], pp. 35–44
29. Ihle, N.: Modellbasierte Wissensextraktion aus Internet-Communities. Master's thesis, University of Hildesheim (2009)
30. Freeman, E., Freeman, E., Bates, B., Sierra, K.: Head First Design Patterns. O'Reilly, Sebastopol (2004)
31. Scott, B., Neil, T.: Designing Web Interfaces: Principles and Patterns for Rich Interactions. O'Reilly Media, Sebastopol (2009)
32. Öllerer, F.: Redesign und Programmierung einer intuitiven Weboberfläche für das Projekt CookIIS, project thesis. Technical report, University of Hildesheim (2009)
33. Hinrichs, T.R.: Problem solving in open worlds. Lawrence Erlbaum, Mahwah (1992)
34. Hammond, K.J.: Chef: A model of case-based planning. In: American Association for Artificial Intelligence, AAAI 1986, Philadelphia, pp. 267–271 (1986) http://www.aaai.org/Papers/AAAI/1986/AAAI86-044.pdf
35. Bach, K., Reichle, M., Althoff, K.D.: A domain independent system architecture for sharing experience. In: Hinneburg, A. (ed.) Proceedings of LWA 2007, Workshop Wissens- und Erfahrungsmanagement, September 2007, pp. 296–303 (2007)

36. Schaaf, M. (ed.): ECCBR 2008, Workshop Proceedings, Trier, Germany, September 1-4. Tharax Verlag, Hildesheim (2008)
37. Delany, S.J. (ed.): Workshop Proceedings of the 8th International Conference on Case-Based Reasoning, Seattle, WA, USA (July 2009)
38. Lenz, M., Bartsch-Spörl, B., Burkhard, H.D., Wess, S. (eds.): Case-Based Reasoning Technology. LNCS, vol. 1400. Springer, Heidelberg (1998)
39. Althoff, K.D., Bergmann, R., Minor, M., Hanft, A. (eds.): ECCBR 2008. LNCS (LNAI), vol. 5239. Springer, Heidelberg (2008)