KE2012 Project

Recipe adjustment for diabetic and hypoglycemic patients

Alex Olieman
alex.olieman@student.uva.nl

Julien Lehuen
julien@lehuen.net

Michael Wolbert
michael.wolbert@gmail.com

January 28th, 2013

Contents

1. Context analysis

1.1 Worksheet OM-1: problems & solutions

Organization Model	**Problems and Opportunities Worksheet OM-1**
PROBLEMS AND OPPORTUNITIES	● Medical condition ● Need to watch levels of sugar/carbs/etc. ● Very few available recipes ● Possibility to adapt existing recipes ● Ingredients substitutions, omissions
ORGANIZATIONAL CONTEXT	Mission, vision, goals of the organization: The diet-advising organization wants to provide suitable recipes for people with glycemic conditions. Nowadays, possible substitutes exist for almost all ingredients (e.g. lactose intolerant people have no trouble replacing all problematic ingredients with soy-based ingredients). The goal of the organization is to propose the easiest way to proceed such recipe adaptation, for blood glucose-related conditions.
	Important external factors the organization has to deal with: People have different tastes, and the substitutions might not please everyone. Religion and allergies could also be an issue. Lastly, people with a condition often don't wish to eat alone, and the final recipe should feel "normal" to others.
	Strategy of the organization: Capitalizing a large knowledge base about ingredients and substitutions, to be able to process any recipe. The panel of recipes could then be "infinite", without the need for extra work from a diet advisor.
	Its value chain and the major value drivers: Diet-advising organisations are publicly funded on a national level. Our knowledge-based system adds value by applying the knowledge from guidelines which were already produced in the past into an automated process. The diet-advisor keeps his or her role of intermediate between the needs of a customer and the general diet guidelines, but this link is made stronger through a system which assists the user in everyday life.
SOLUTIONS	● Semi-automatic system ● Knowledge about recipes and ingredients ● Check existing recipes ● Replace, adjust quantities, and omit ingredients

1.2 Worksheet OM-2: process & people involved

Organization Model	**Variant Aspects Worksheet OM-2**
STRUCTURE	See Figure 1. Use cases and Figure 2. Organigram

PROCESS	See Figure 3. Activity diagram
PEOPLE	● Organisation ○ Management ○ Diet advisor ● Customer
RESOURCES	● Information systems ○ Recipes database ○ Ingredients database ● Technical equipment ○ Computers for the diet advisor / at home ○ Mobile devices (phones/tablets) for use while shopping
KNOWLEDGE	● Ingredients properties ● Glycemic constraints ● Possible substitutions/omissions ● Recipes
CULTURE AND POWER	<u>Power:</u> Patients hire the diet adviser for a session in which evaluation of the patients current eating habits and conditions is performed. The diet advice which results from this session are not enforced on the patient, hence its an advice. Therefore the power relation is rather flexible between the two parties.
	<u>Culture:</u> The consultation between a patient and a diet adviser is informal given the personal advice. The sessions are open for discussion and no one enforces power over the other.

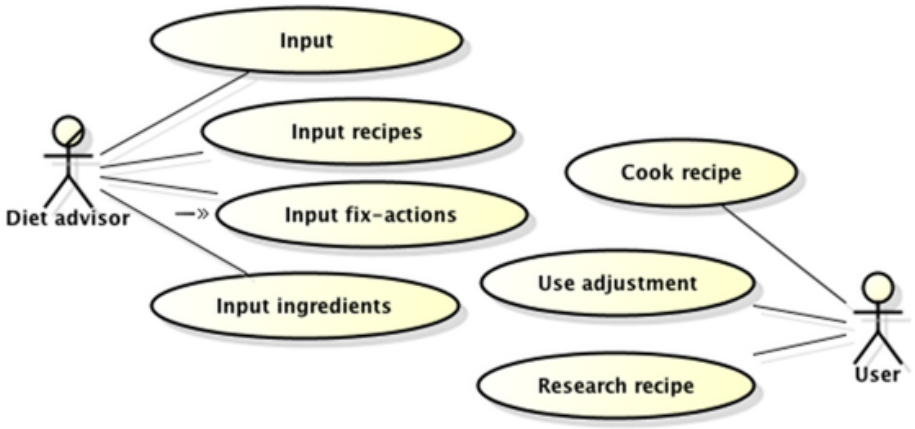


Figure 1. Use cases

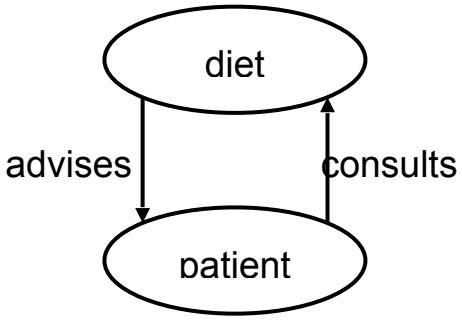


Figure 2. Organigram

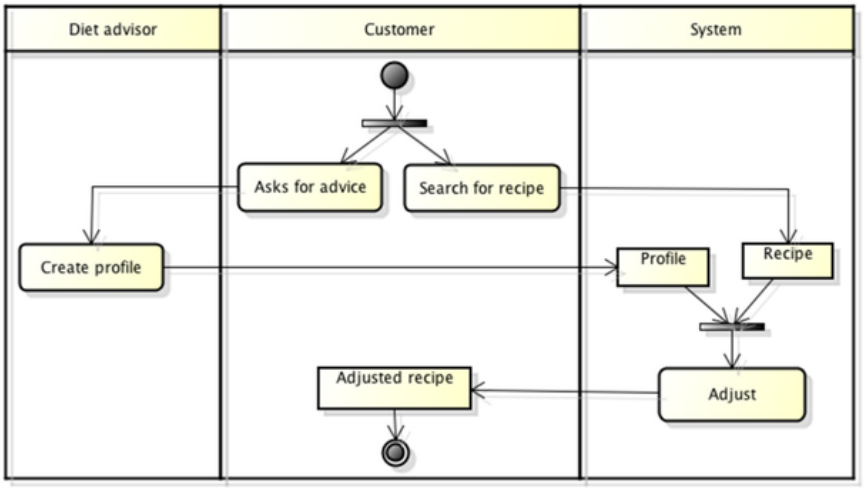


Figure 3. Activity diagram

1.3 Worksheet OM-5: technical and business feasibility

Organization Model	Checklist for Feasibility Decision Document: Worksheet OM-5
BUSINESS FEASIBILITY	1. Benefits The automation of recipe adaptation will provide the organization with a very large panel of recipes for people with a glucose-related condition. Such people are many and the quick and automatic generation of suitable recipes is a true benefit. 2. Added value Diet advisor can focus on specific tasks, without having to do this repetitive systematic job. Customers will get more ideas instead of limiting themselves to the few recipes they are used to. The life of both the advisor and the customer will be made easier this way. 3. Costs Consequent developing costs to create the system and gather all the required knowledge. The costs of gathering knowledge should be kept low by reusing existing knowledge of recipes and nutritional values of ingredients. Very low running costs to keep the equipment running and let the system work. 4. Alternatives Giving personalized advice on the recipe level manually will give a lot of work to the diet advisors. 5. Organizational changes Diet advisors will have a more specific role, and people should directly be given access to the system's outputs in some way. 6. Risks and uncertainties People could dislike the impersonal aspects of such system. Because no culinary knowledge is represented in the system, some adapted recipes might not taste well or be difficult to prepare.
TECHNICAL FEASIBILITY	1. Task complexity The task itself is very simple. The required knowledge is a - as complete as possible - listing of ingredients with their properties, associated to substitution/omission knowledge and a recipes database. The analysis and the modification of a given recipe is basically a calculation on the recipe ingredients properties, and eventually some substitutions and/or omissions until thresholds are met. 2. Critical aspects The most critical aspect of the recipe adaptation is probably the prioritization of the possible recipe modification. The effect on the glycemic-related values should be maximized and the taste changes should be minimized. 3. Success measures The system outputs new recipes, which should then be "tried" (cooked and tasted) by test-users. Patients using the system should also report whether their symptoms are alleviated when following the adjusted recipes. This evaluation would state if the system is a success. 4. User interaction complexity The user interaction is simple, since it only consists in the choice of a recipe, maybe some personalisation and the output of an adapted recipe. 5. Other systems interaction complexity The system could benefit from the interaction with existing ingredients and/or recipe knowledge systems, which would add complexity to the project. 6. Risks and uncertainties The gathering of a consequent knowledge base still feels very uncertain. Diet advisors might have to contribute manually, since an automatic crawling could give vague/incomplete results.

PROJECT FEASIBILITY	1. Commitment The diet advisor is committed to work closely with the development team, to share knowledge, help build the models and perhaps fill the knowledge bases. 2. Resources Personal hardware is enough to set the first prototypes up. 3. Knowledge The data about recipes and ingredients should be gathered automatically into a DataBase Management System. Diet advisor could complete the base with constraints and calculations. 4. Realistic The final system is rather ambitious and some issues could be quite complex. Building a prototype with the simplest kinds of rules is still realistic. 5. Organization The diet advice organization has a hosted website, which could welcome a recipe adaptation system. 6. Risks and uncertainties No specific risks are to be expected during the prototyping phase.
PROPOSED ACTIONS	1. Focus The team will focus on the building of a simplified prototype. This is interesting and feasible, while aiming at a complete system is too ambitious. 2. Target solution Working prototype and demo with two or three recipes. 3. Results and benefits A working prototype would show the possible use of a knowledge-related system, and suggest the further development of a final system. 4. Project actions Create models and designs. Spiral development of the prototype until January 25th. 5. Change conditions RAS

1.4 Worksheet TM-1: task analysis

***Note**: the worksheet is only filled for the most knowledge intensive task.*

Task Model	**Task Analysis Worksheet TM-1**
TASK	04 - Recipe adaptation
ORGANIZATION	Diet advice (Sèfi Willemsen)
GOAL AND VALUE	Modify an original recipe until it meets a set of constraints
DEPENDENCY AND FLOW	Preceding Tasks: 01 - User connection 02 - Recipe selection 03 - Constraints listing
	Follow-up Tasks: 05 - Recipe checking 06 - Recipe output
OBJECTS HANDLED	Input Objects: ● Original recipe ● Constraints

	Output Objects: ● Modified recipe
	Internal Objects: ● Ingredients ● Fix actions
TIME AND CONTROL	Frequency, Duration: The user input should be kept simple, and would never exceed 2 to 5 minutes. The computation itself should be as quick as possible (1 to 5 seconds).
	Control: The system should constantly check for constraints and recipe coherence.
	Constraints & Conditions: ● Preconditions: the recipe is coherent ● Postconditions: the recipe is coherent and respects the glucose-related constraints ● Constraints ○ maximum glycemic load ○ calories, fiber, and protein constraints
AGENTS	Running on a server Use of databases
KNOWLEDGE AND COMPETENCE	The output of the system could be saved as an history, so that the recipe is re-used later without going through the whole process again. This idea would then make the task described in TM-2 a generator of knowledge for the organization...
RESOURCES	1 (web)server 4 knowledge bases: recipes, ingredients, constraints & fix actions
QUALITY AND PERFORMANCE	Test-users have to cook and taste the output recipes, and fill evaluation forms

1.5 Worksheets TM-2: knowledge used in the selected task

***Note**: This worksheet is filled four times for the four kinds of knowledge items: recipe, ingredient, fix action, constraints.*

Task Model	**Knowledge Item Worksheet TM-2**
Name: Possessed by: Used in: Domain:	Recipe “people” in general 02 - Recipe selection ; 04 - Recipe adaptation Cooking (Health & care)
NATURE OF THE KNOWLEDGE	
Formal, rigorous	
Empirical, quantitative	✓
Heuristic, rules of thumb	
Highly specialized, don specific	
Experience-based	✓ (bottleneck)

Action-based	✓
Incomplete	✓ (to be improved)
Uncertain, may be incorrect	
Quickly changing	
Hard to verify	✓ (to be improved)
Tacit, hard to transfer	✓ (bottleneck)
FORM OF THE KNOWLEDGE	
Mind	✓ (bottleneck)
Paper	✓ (to be improved)
Electronic	✓
Action skill	
Other	
AVAILABILITY OF THE KNOWLEDGE	
Limitations in time	
Limitations in space	
Limitations in access	✓
Limitations in quality	✓ (to be improved)
Limitations in form	✓ (bottleneck)

Task Model	**Knowledge Item Worksheet TM-2**
Name: Possessed by: Used in: Domain:	Ingredient properties Databases, indexes 04 - Recipe adaptation ; 05 - Recipe checking Cooking (Health & care)
NATURE OF THE KNOWLEDGE	
Formal, rigorous	
Empirical, quantitative	✓
Heuristic, rules of thumb	
Highly specialized, don specific	
Experience-based	✓ (bottleneck)
Action-based	✓
Incomplete	✓ (to be improved)

Uncertain, may be incorrect	
Quickly changing	
Hard to verify	✓ (to be improved)
Tacit, hard to transfer	✓ (bottleneck)
FORM OF THE KNOWLEDGE	
Mind	
Paper	✓ (to be improved)
Electronic	✓
Action skill	
Other	
AVAILABILITY OF THE KNOWLEDGE	
Limitations in time	
Limitations in space	✓
Limitations in access	✓
Limitations in quality	✓
Limitations in form	✓ (to be improved)

Task Model	**Knowledge Item Worksheet TM-2**
Name: Possessed by: Used in: Domain:	Fix actions System 04 - Recipe adaptation Cooking (Health & care)
NATURE OF THE KNOWLEDGE	
Formal, rigorous	
Empirical, quantitative	✓
Heuristic, rules of thumb	✓
Highly specialized, don specific	
Experience-based	✓ (bottleneck)
Action-based	
Incomplete	✓ (to be improved)
Uncertain, may be incorrect	
Quickly changing	

Hard to verify	✓ (to be improved)
Tacit, hard to transfer	
FORM OF THE KNOWLEDGE	
Mind	✓ (to be improved)
Paper	
Electronic	✓
Action skill	
Other	
AVAILABILITY OF THE KNOWLEDGE	
Limitations in time	
Limitations in space	
Limitations in access	✓
Limitations in quality	✓ (to be improved)
Limitations in form	✓ (bottleneck)

Task Model	**Knowledge Item Worksheet TM-2**
Name: Possessed by: Used in: Domain:	Constraints User (“extracted” by the diet advisor) 04 - Recipe adaptation Health, condition
NATURE OF THE KNOWLEDGE	
Formal, rigorous	✓
Empirical, quantitative	✓
Heuristic, rules of thumb	✓
Highly specialized, don specific	✓
Experience-based	
Action-based	
Incomplete	
Uncertain, may be incorrect	
Quickly changing	✓ (bottleneck)
Hard to verify	
Tacit, hard to transfer	

FORM OF THE KNOWLEDGE	
Mind	
Paper	✓ (to be improved)
Electronic	✓
Action skill	
Other	
AVAILABILITY OF THE KNOWLEDGE	
Limitations in time	✓
Limitations in space	
Limitations in access	✓ (to be improved)
Limitations in quality	
Limitations in form	

1.6 Worksheets AM-1: humans: expected capabilities and skills

***Note**: The AM-1 worksheet is filled here for two agents: the diet advisor and the user*

Agent Model	**Agent Worksheet AM-1**
NAME	Diet advisor
ORGANIZATION	Works for the organization; Meets users (clients).
INVOLVED IN	01 - User connection 06 - Recipe output
COMMUNICATES WITH	Users
KNOWLEDGE	Recipes Ingredient properties
OTHER COMPETENCIES	Knowledge of good practices Social, able to work with the user
RESPONSIBILITIES AND CONSTRAINTS	User's health and diet Client relation

Agent Model	**Agent Worksheet AM-1**
NAME	User (client)
ORGANIZATION	Is a client of the organization Meets the diets advisor Interacts with the system

INVOLVED IN	01 - User connection 02 - Recipe selection 03 - Constraints listing 06 - Recipe output
COMMUNICATES WITH	Diet advisors
KNOWLEDGE	Personal taste Religion Allergies
OTHER COMPETENCIES	Use a computer and access the web Understand a recipe and cook
RESPONSIBILITIES AND CONSTRAINTS	Own health and diet Medical condition

2. Knowledge model

2.1 Task knowledge

2.1.1 Task template

The template inference structure which matches our system best is the configuration design template. Especially, the propose-critique-modify (PCM) method corresponds a lot to the way our system could proceed (Chandrasekaran, 1990). Section 2.2 presents an inference structure that is based on this template, annotated with specificities of our system. The main modifications of the template inference structure are the replacement of specify with search, and the omission of soft requirements. Furthermore, we expect recipes to violate multiple constraints, where the template assumes that only one constraint will be violated at a time. We also modified the original inference, so that any new version of the recipe goes through the verification step before being modified again.

2.1.2 Task decomposition diagram

The tasks involved for the goal of our system, adjusting meal recipes, is twofold (see Figure 4).

First, the user searches for recipe instances in the knowledge base by entering a concept query (Van Harmelen et al. 2009). An example of such a concept query could be ‘Italian spaghetti’. This results in a set of recipe instances which are of the type ‘Italian spaghetti’ (or any of its subclasses). The user then selects one recipe from the result set.

Second, the PCM task method takes as input the received recipe from the search task method via the obtain transfer function. The PCM task method is used to configure the recipe based on constraints and preferences. The control regimen of the inference steps in the PCM task method may be looped over several times. This is specified further by the following subsection.

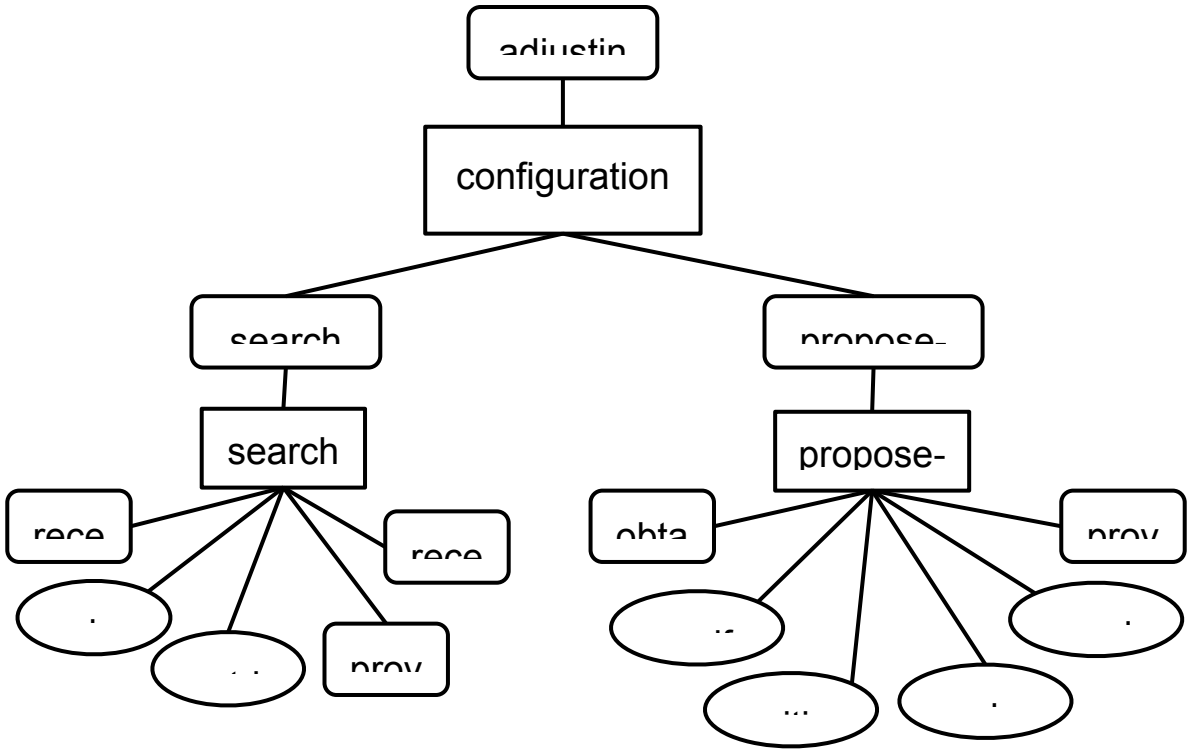


Figure 4. Task knowledge model for recipe search and propose-critique-modify.

2.1.3 Task and Task method description

TASK configuration-design;
  ROLES:
    INPUT:
      requirements: “personal dietary guidelines, as provided by the diet advisor”;
      query: “search terms, including the (partial) name and/or attribute values of a recipe”;
    OUTPUT:
      design: “the reconfigured recipe”;
END TASK configuration-design;

TASK METHOD propose-critique-modify;
  REALIZES: configuration-design;
  DECOMPOSITION:
    INFERENCES: operationalize, search, propose, verify, critique,
                select, modify;
  ROLES:
    INTERMEDIATE:
      hard-requirements: “requirements to be used as hard constraints”;
      skeletal-design: “the initial recipe and its ingredients”;
      extension: “modified (adjusted quantity, addition, omission, substitution) ingredient for recipe”;
      violations: “constraint(s) violated by the current recipe”;
      truth-value: “Boolean indicating result of the verification”;
      action-list: “ordered list of possible recipe fix actions (adjust ingredient quantity,

```
        ingredient addition, ingredient omission, ingredient substitution)";
        action: " a single recipe fix action";
    CONTROL-STRUCTURE:
      operationalize (requirements -> hard-requirements);
      search (query -> skeletal-design);
      WHILE NEW-SOLUTION propose(skeletal-design + design => extension)  DO
        design := extension ADD design;
        verify(design + hard-requirements -> truth-value + violations);
        IF truth-value == false
        THEN
          critique(violations + design -> action-list);
          REPEAT
            select(action-list -> action);
            modify(design + action -> design);
            verify(design + hard-requirements -> truth-value + violations);
          UNTIL truth-value == true;
          END REPEAT
        END IF
      END WHILE
END TASK-METHOD propose-critique-modify;
```

2.2 Inference knowledge

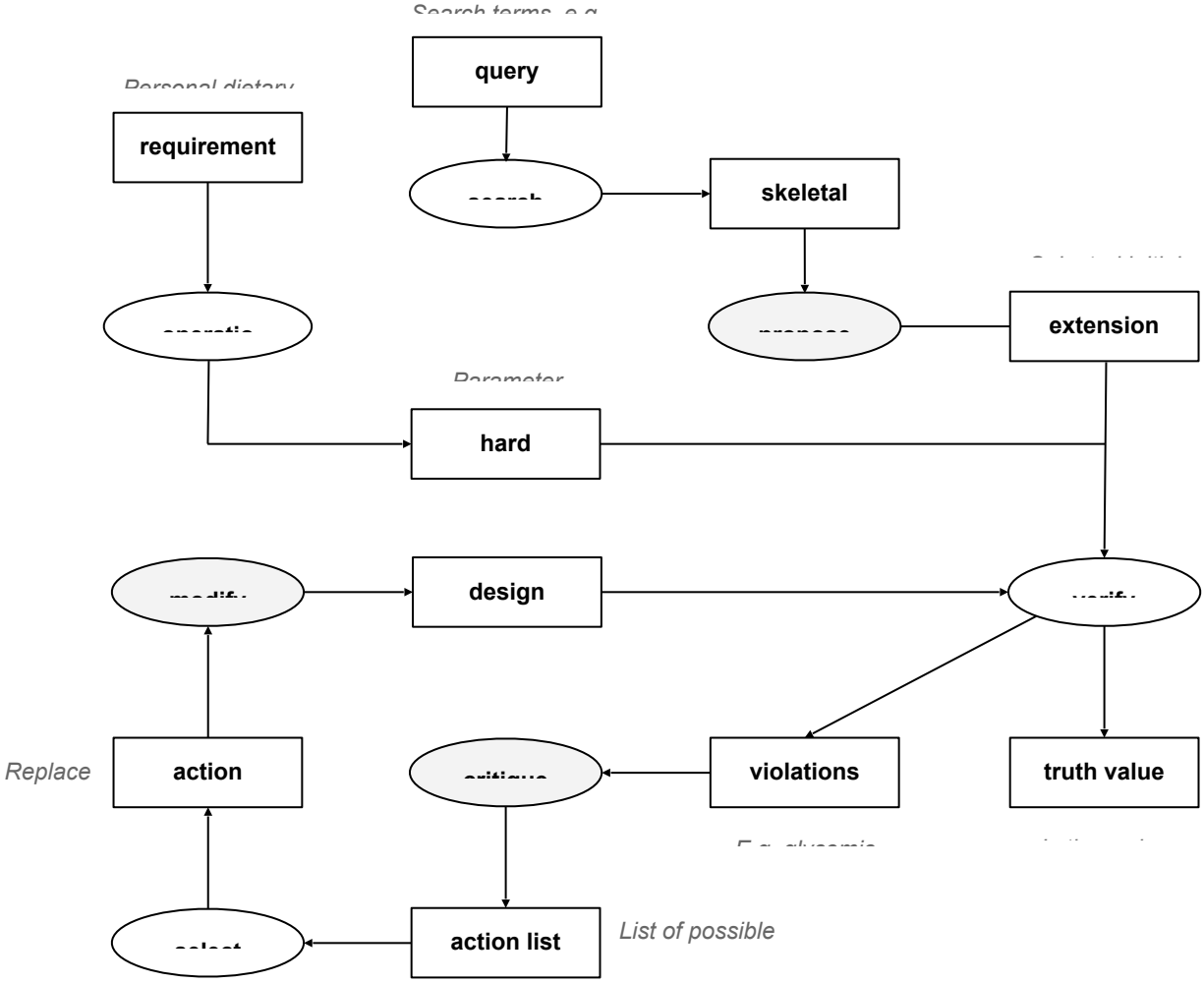


Figure 5. Annotated inference structure for modified configuration task.

Knowledge role	Type	Domain schema
hard requirements	dynamic	recipe-constraint
skeletal-design	static	recipe
violations	dynamic	recipe-constraint
action	dynamic	recipe-fix-action
design	dynamic	recipe

2.3 Domain knowledge

2.3.1 Domain schema

The domain schema for our system is presented in Figure 6. Recipes and their ingredients are associated through the association relation has-ingredient which holds the quantity of an ingredient of the given recipe. Recipe-fix-action consist of ingredient quantity adjustment, ingredient omission, ingredient addition and ingredient substitution. For the latter two, an association is needed to a single ingredient as additive and substitute respectively. All recipe-fix-actions target a has-ingredient instance (e.g. quantity adjustment modifies the has-ingredient quantity attribute). If a recipe-constraint (e.g. the glycemic load of a recipe is too high) is violated, the violation attribute holds 'true'. Therefore each recipe-constraint should have at least one recipe-fix-action associated to propose a possible fix for the recipe. An example for this is that the veganism-constraint has associations with an omission and substitution fix action to remove or replace any animal ingredients from the design.

We have chosen to leave the parameter concept out of the domain schema for sake of clarity. We argue that predefined values are not necessary at this moment to be stored in a separate parameter concept, but may instead be represented as attributes of recipes and ingredients. Also, the computed values can be stored in attributes of the recipe concept instead of in the parameter concept. It is important to note that the recipe attributes glycemic-load, calories, saturated-fats, proteins and fibers are the computed values for one serving of the given recipe. These attributes are consequents of the recipe-calculation-rules as presented in sections 2.3.2 and 2.3.3.

Furthermore, we have introduced some (not all!) subclasses of the ingredient concept to demonstrate the integration of a food related ontology. These concepts can be used to evaluate and find suitable fix-actions (e.g. substitution), but may also be used in the future to write rules regarding culinary knowledge.

Two domain-specific concepts may require an additional explanation. The glycemic index (GI) of an ingredient gives an indication of the relative blood glucose-response that is to be expected after eating the respective ingredient. Values for the GI are obtained by giving healthy participants a portion of the food containing a fixed amount of carbohydrates, and measuring their blood glucose levels at fixed intervals. This result is normalized by comparing the found blood glucose-response to the known response for pure glucose, which has GI = 100.

While the GI allows the comparison of ingredients in an abstract sense, there is a need to estimate the actual blood glucose-response that will occur after consuming a food. To this end, the concept of glycemic load (GL) is used. The glycemic load takes into account both the amount of carbohydrates in the portion of food and the respective GI. It can be calculated for any quantity of an ingredient, but can also be summed for an entire recipe. This allows the creation of constraints on dishes that are meaningful to diabetic and hypoglycemic patients.

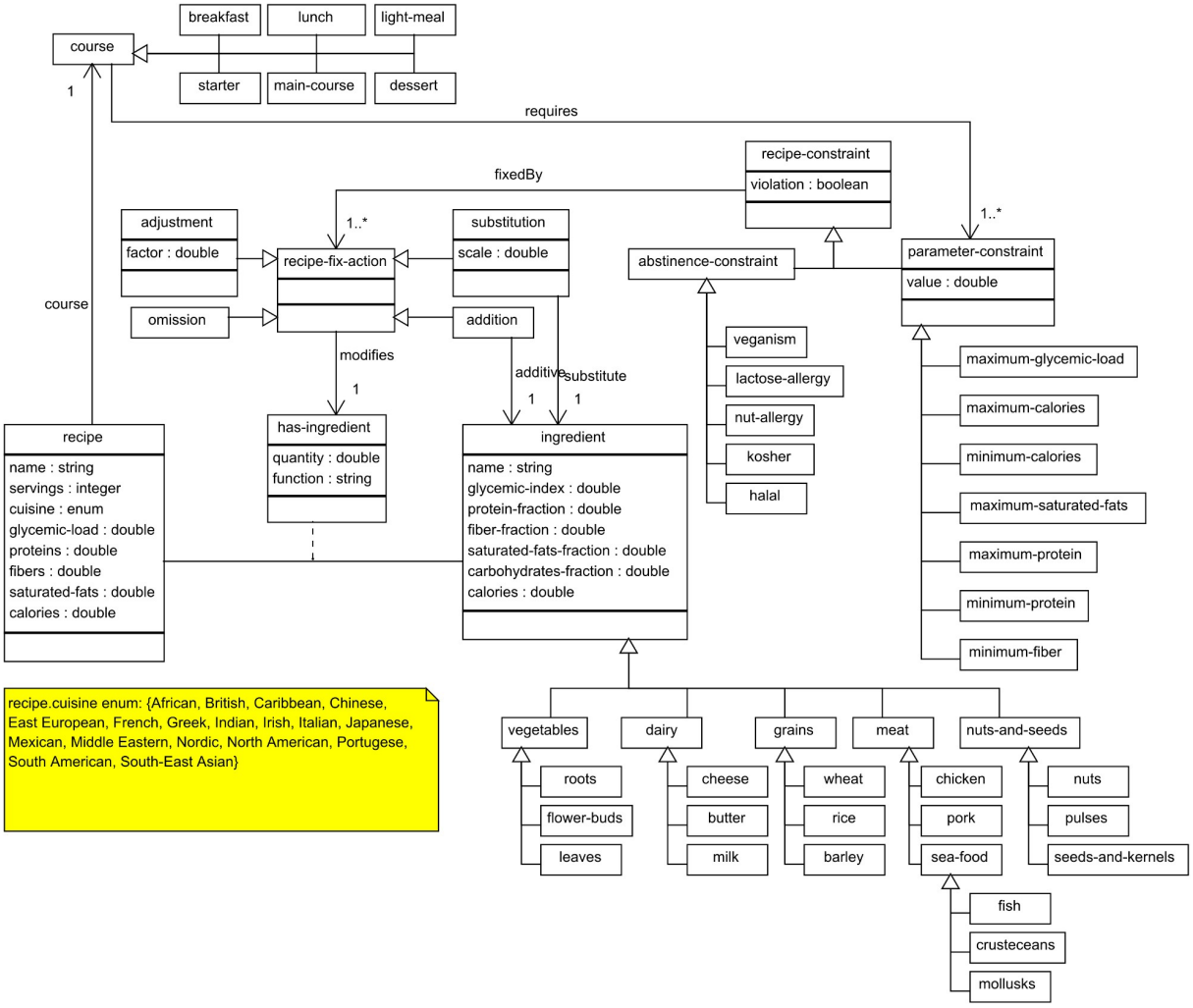


Figure 6. Domain schema for the recipe reconfiguration domain.

2.3.2 Rule types

The design element from the domain schema template of configure design, map to recipe and ingredient concepts in our domain schema. The rule types that are found in configuration design are presented below, modified slightly to match the domain schema. Preference expressions are not used because we only consider hard-requirements.

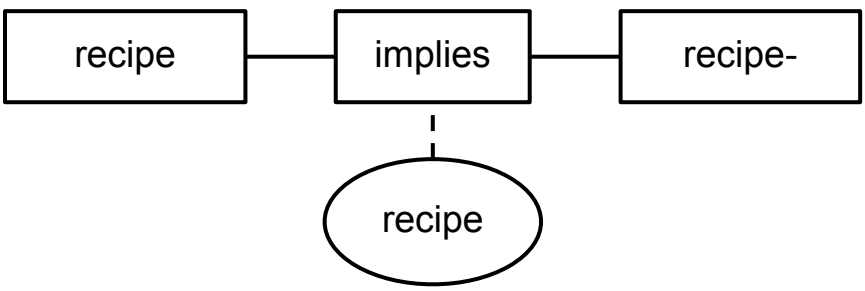


Figure 7. recipe-constraint-rule type

```
RULE-TYPE recipe-constraint-rule
 ANTECEDENT: recipe
  CARDINALITY: 1+;
 CONSEQUENT: recipe-constraint
  CARDINALITY: 1;
 CONNECTION-SYMBOL:
  implies;
END RULE-TYPE recipe-constraint-rule
```

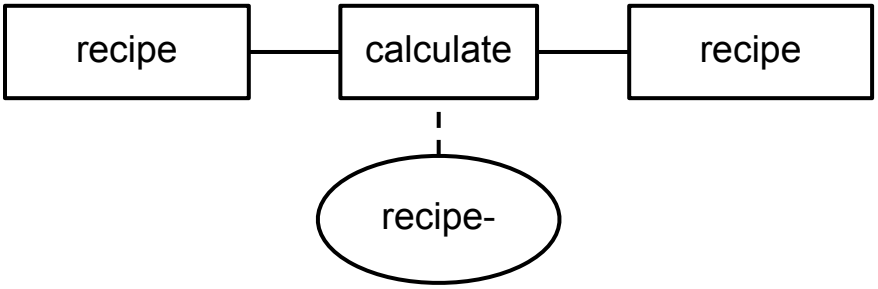


Figure 8. recipe-calculation-rule type

```
RULE-TYPE recipe-calculation-rule
 ANTECEDENT: recipe
  CARDINALITY: 1+;
 CONSEQUENT: recipe
  CARDINALITY: 1;
 CONNECTION-SYMBOL:
  calculates;
END RULE-TYPE recipe-calculation-rule
```

2.3.3 Knowledge base

In this knowledge base we include instances of constraint rules, calculation rules and fix-actions. Instances of recipes and ingredients are described in section 2.4 Scenarios.

```
KNOWLEDGE-BASE recipe-reconfiguration;
 USES:
  recipe-constraint-rule FROM recipe-reconfiguration-schema,
  recipe-calculation-rule FROM recipe-reconfiguration-schema;
 EXPRESSIONS:
  # Glycemic load rule instance
  recipe.glycemic-load > recipe.course.maximum-glycemic-load.value
     IMPLIES
  maximum-glycemic-load.violation = true;

  # Saturated fat rule instance
  recipe.saturated-fats > recipe.course.maximum-saturated-fats.value
```

```
   IMPLIES
maximum-saturated-fats.violation = true;

# Calorie rule instances
recipe.calories > recipe.course.maximum-calories.value
   IMPLIES
maximum-calories.violation = true;

recipe.calories < recipe.course.minimum-calories.value
   IMPLIES
minimum-calories.violation = true;

# Fiber rule instance
recipe.fibers < recipe.course.minimum-fibers.value
   IMPLIES
minimum-fibers-constraint.violation = true;

# Protein rule instances
recipe.proteins > recipe.course.maximum-proteins.value
   IMPLIES
maximum-proteins.violation = true;

recipe.proteins < recipe.course.minimum-proteins.value
   IMPLIES
minimum-proteins.violation = true;

# Abstinence rule instances
recipe.has-ingredient.dairy OR recipe.has-ingredient.meat
   IMPLIES
veganism.violation = true;

recipe.has-ingredient.dairy
   IMPLIES
lactose-allergy.violation = true;

recipe.has-ingredient.nuts OR recipe.has-ingredient.pulses
   IMPLIES
nut-allergy.violation = true;

recipe.has-ingredient.pork OR recipe.has-ingredient.mollusks
OR recipe.has-ingredient.crustaceans
   IMPLIES
kosher.violation = true;

recipe.has-ingredient.milk AND recipe.has-ingredient.meat
   IMPLIES
kosher.violation = true;
```

```
recipe.has-ingredient.pork OR recipe.has-ingredient.alcohol
   IMPLIES
halal.violation = true;

# Calculation rule instances
FORALL x:recipe.has-ingredient
        sum(x.ingredient.glycemic-index *
        x.ingredient.carbohydrates-fraction * (x.quantity /
        recipe.servings))
          CALCULATES
        recipe.glycemic-load;

FORALL x:recipe.has-ingredient
        sum(x.ingredient.saturated-fats-fraction * (x.quantity /
        recipe.servings))
          CALCULATES
        recipe.saturated-fats;

FORALL x:recipe.has-ingredient
        sum(x.ingredient.calories * (x.quantity / recipe.servings))
          CALCULATES
        recipe.calories;

FORALL x:recipe.has-ingredient
        sum(x.ingredient.fiber-fraction * (x.quantity /
        recipe.servings))
          CALCULATES
        recipe.fibers;

FORALL x:recipe.has-ingredient
        sum(x.ingredient.protein-fraction * (x.quantity /
        recipe.servings))
          CALCULATES
        recipe.proteins;

# Fix-action instances
fix-action( action(adjustment), recipe.has-ingredient.quantity );

VAR x, y: ingredient;
fix-action( action(substitution), recipe.has-ingredient.x,
             recipe.has-ingredient.y );

fix-action( action(addition), recipe.has-ingredient.ingredient );

fix-action( action(omission), recipe.has-ingredient.ingredient );
```

```
END KNOWLEDGE-BASE recipe-reconfiguration;
```

2.4 Scenarios

To illustrate the objectives of our system, we asked Z. van Nahuijs to tell us what adjustments she would usually do on ten example recipes. These are the kind of adjustments we would want our system to make on any recipe that would be included in the knowledge base. Two of these recipes and their respective adjustments are presented below. Following the two scenarios, we describe how the knowledge base is filled with instances of recipes and ingredients.

2.4.1 Fast lasagne with pork and spinach

Ingredients for the ragu and spinach sauce:

- 1 tbsp oil
- 450g pork sausage meat
- 1 red chilli, deseeded and finely chopped
- 2 fat cloves garlic, crushed
- 150g button mushrooms, sliced
- 200 ml full-fat crème fraiche
- 100g baby spinach, roughly chopped
- salt and freshly ground black pepper

Ingredients for the tomato and thyme sauce:

- 400g can chopped tomatoes
- 2 tbsp sun-dried tomato paste
- 1 tbsp demerara sugar
- 1 tbsp fresh thyme leaves

Ingredients for the lasagne:

- 6 sheets, about 75g quick-cook white lasagne or fresh lasagne
- 200g strong cheddar cheese, grated

Recipe adjustments
Most of the adjustments are ingredient-related. Some ingredients have to be replaced, some others can simply be used in lower doses or omitted. Ingredients can also be added.

Replacements:

- Beef meat instead of pork meat *(=> less saturated fat)*
- Whole wheat lasagna instead of white *(=> more fiber)*
- Goat's cheese instead of cheddar *(=> less lactose and saturated fat)*
- Baking soda instead of sugar *(=> lower glycemic load, still reduces acidity)*

Omission

- Crème fraiche *(=> too much saturated fat, lactose and additives)*

Additions

- More vegetables *(=> more fiber)*

2.4.2 Beetroot chocolate cake

Ingredients:

- 200g butter, plus extra for greasing
- 250g cooked and peeled beetroot
- 200g dark chocolate (70% cocoa solids)
- 4 tbsp hot espresso
- 135g plain flour
- 1 heaped tsp baking powder
- 3 tbsp cocoa powder
- 5 free-range eggs, separated
- 190 g golden caster sugar
- crème fraîche or double cream, to serve

Recipe adjustments
Here again, most of the adjustments are ingredient-related. Some ingredients have to be replaced, some others can simply be used in lower doses or omitted.

Replacements:

- Oil instead of butter *(=> less saturated fat)*
- Whole wheat flour instead of white *(=> more fiber)*

Omission

- Crème fraiche *(=> too much saturated fat, lactose and additives)*
- Sugar *(=> too high glycemic load)*

Addition
More baking powder *(=> because whole wheat flour has no additive)*

2.4.3 Instances of recipes and ingredients

For the users of this system it is desirable to be able to choose from the greatest possible diversity of recipes. Therefore, we give a brief overview of the possibility to gather instances of recipes and ingredients automatically. A great amount of recipes could be gathered from the Web by crawling methods. This is aided by the existence of the existence of the hRecipe microformat[1], which is already in use by several search engines. However, for the scenarios and testing we found it sufficient to add five recipes to the knowledge base manually. We used "beetroot chocolate cake", "cheeseburger with skinny fries", "delicious fried rice", "fast lasagne with pork and spinach", and "gingerbread pancakes" from the BBC recipe site[2].

When looking for nutritional facts of ingredients on the Web, much conflicting information can

---

[1] http://microformats.org/wiki/hrecipe

[2] http://www.bbc.co.uk/food/recipes/

be found. It is therefore preferable to use reliable data sources in the definition of ingredient instances. Wolfram|Alpha[3] is an example of a provider which allows programmatic access to its databases, which have a broad coverage of curated data for the food and nutrition domain. To define ingredient instances for the scenarios, we automatically retrieved nutritional data from Freebase[4]. For some ingredients no data could be found on Freebase, so this was manually added from Wolfram|Alpha. Both sources, however, do not offer any glycemic index values for ingredients. We used the overview compiled by Foster-Powell, Holt, & Brand-Miller (2002).
The following syntax gives an example of the instances that represent recipes and ingredients.

```
INSTANCE beetroot-cake;
 INSTANCE-OF recipe;
 ATTRIBUTES:
   name: 'Beetroot chocolate
         cake';
  servings: 6;
  cuisine: Irish;
END INSTANCE

TUPLE
 INSTANCE-OF: course;
 ARGUMENT-1: cake;
 ARGUMENT-2: dessert;
END TUPLE

TUPLE
 INSTANCE-OF: has-ingredient;
 ARGUMENT-1: cake;
 ARGUMENT-2: sugar;
 ATTRIBUTES:
  quantity: 200.0;
  function: 'sweetener',
            'texturizer';
END TUPLE

INSTANCE sugar;
 INSTANCE-OF sweetener;
 ATTRIBUTES:
  name: 'Sugar';
  glycemic-index: 68;
  protein-fraction: 0.0;
  fiber-fraction: 0.0;
  saturated-fats-fraction: 0.0;
  carbohydrates-fraction: 0.9998;
```

```
INSTANCE fast-lasagne;
 INSTANCE-OF recipe;
 ATTRIBUTES:
  name: 'Fast lasagne with pork
        and spinach';
  servings: 6;
  cuisine: Italian;
END INSTANCE

TUPLE
 INSTANCE-OF: course;
 ARGUMENT-1: fast-lasagne;
 ARGUMENT-2: starter;
END TUPLE

TUPLE
 INSTANCE-OF: has-ingredient;
 ARGUMENT-1: fast-lasagne;
 ARGUMENT-2: cheddar;
 ATTRIBUTES:
  quantity: 200.0;
  function: 'topping';
END TUPLE

INSTANCE cheddar;
 INSTANCE-OF cheese;
 ATTRIBUTES:
  name: 'Cheddar';
  glycemic-index: 0;
  protein-fraction: 0.25;
  fiber-fraction: 0.0;
  saturated-fats-fraction: 0.21;
  carbohydrates-fraction: 0.01;
  calories: 4.03;
```

[3] http://www.wolframalpha.com/examples/FoodAndNutrition.html

[4] http://www.freebase.com/schema/food/nutrition_fact

3. Communication model

3.1 Communication plan

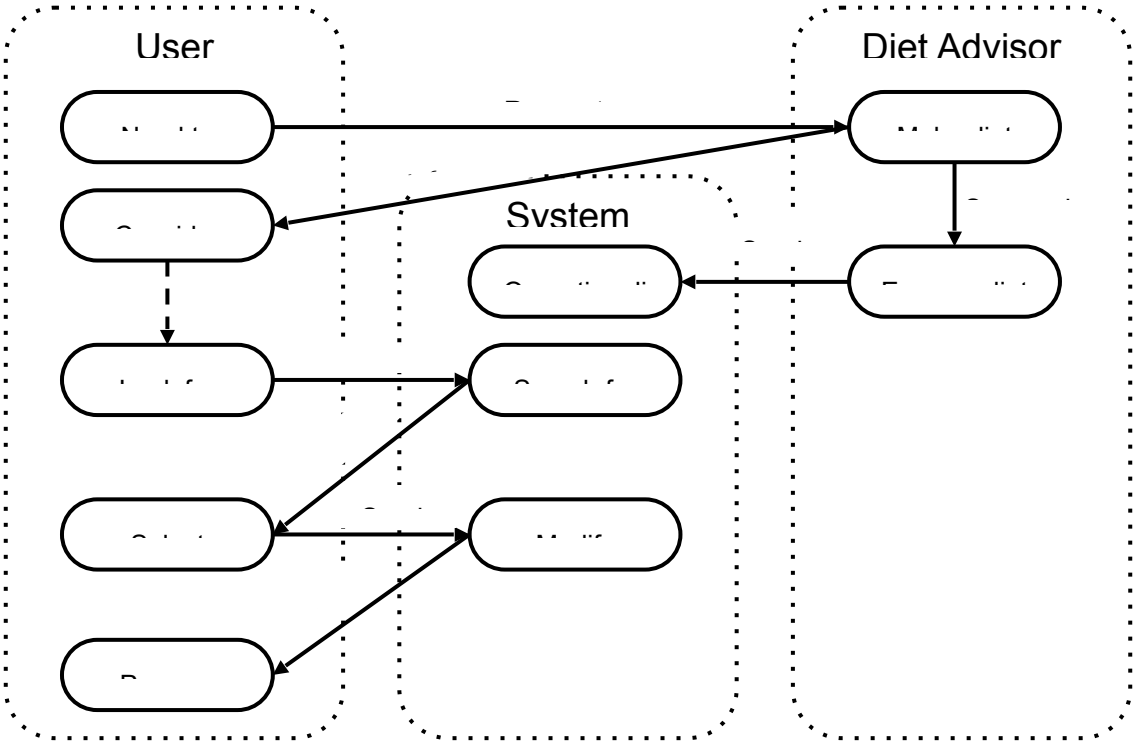


Figure 9. Communication plan

3.2 Worksheets CM-1: Transactions description

Communication Model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER	Request advice
INFORMATION OBJECT	Medical diagnoses, the general health condition, and the eating habits of the user.
AGENTS INVOLVED	User, Diet Advisor
CONSTRAINTS	Pre-condition: Medical diagnoses have been made. Post-condition: The diet advisor has enough information to make diet guidelines.

Communication Model	**Transaction Description Worksheet CM-1**

TRANSACTION IDENTIFIER	Inform user
INFORMATION OBJECT	Diet guidelines (meal schedule, ingredients to avoid/ include, portion sizes)
AGENTS INVOLVED	Diet Advisor, Patient
CONSTRAINTS	Pre-condition: Personal diet guidelines have been made. Post-condition:

Communication Model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER	Summarize
INFORMATION OBJECT	The diet guidelines are quantified and summarized into constraints on meal parameters.
AGENTS INVOLVED	Diet Advisor
CONSTRAINTS	Pre-condition: Post-condition:

Communication Model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER	Send constraints
INFORMATION OBJECT	Constraints on meal parameters
AGENTS INVOLVED	Diet Advisor, System
CONSTRAINTS	Pre-condition: The constraints could be expressed as lower and upper limits on parameters of single dishes. Post-condition:

Communication Model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER	Query
INFORMATION OBJECT	A query, consisting of the (partial) name of a dish, a course, and/or cuisine.
AGENTS INVOLVED	User, System
CONSTRAINTS	Pre-condition: The patient is planning a meal. Post-condition:

Communication Model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER	Send results
INFORMATION OBJECT	A ranked set of search results (recipes)
AGENTS INVOLVED	System, Patient

CONSTRAINTS	Pre-condition: The query returned at least one result. Post-condition:

Communication Model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER	Send selection
INFORMATION OBJECT	One recipe from the search results
AGENTS INVOLVED	User, System
CONSTRAINTS	Pre-condition: Post-condition:

Communication Model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER	Send recipe
INFORMATION OBJECT	The modified recipe
AGENTS INVOLVED	System, Patient
CONSTRAINTS	Pre-condition: Post-condition:

3.3 Worksheets CM-2: Information exchange specifications

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Request advice
AGENTS INVOLVED	Sender: User Receiver: Diet Advisor
INFORMATION ITEMS	Medical diagnoses, the general health condition, and the eating habits of the user. Role: core Form: structured conversation Medium: speech or writing
MESSAGE SPECIFICATION	Type: REQUEST Content: Reference:
CONTROL OVER MESSAGES	

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Inform patient
AGENTS INVOLVED	Sender: Diet Advisor Receiver: Patient
INFORMATION ITEMS	Diet guidelines (meal schedule, ingredients to avoid/ include, portion sizes) Role: core Form: temporal overview, lists Medium: writing Explanation of diet guidelines Role: supporting Form: remarks / annotations Medium: speech or writing
MESSAGE SPECIFICATION	Type: PROPOSE Content: Reference:
CONTROL OVER MESSAGES	

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Summarize
AGENTS INVOLVED	Sender: Diet Advisor Receiver: Diet Advisor

INFORMATION ITEMS	Constraints on meal parameters Role: core Form: upper and lower limits Medium: fields on patient record (written or digital)
MESSAGE SPECIFICATION	Type: REPORT Content: limits for glycemic load, saturated fat, dietary fibers, protein, and calories Reference: domain knowledge of the diet advisor
CONTROL OVER MESSAGES	

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Send constraints
AGENTS INVOLVED	Sender: Diet Advisor Receiver: System
INFORMATION ITEMS	Role: core Form: natural numbers Medium: HTML form input fields
MESSAGE SPECIFICATION	Type: ORDER Content: limits for glycemic load, saturated fat, dietary fibers, protein, and calories Reference:
CONTROL OVER MESSAGES	

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Query
AGENTS INVOLVED	Sender: User Receiver: System
INFORMATION ITEMS	Role: core Form: data string (containing keywords and concepts) Medium: HTML form input fields
MESSAGE SPECIFICATION	Type: ASK Content: query terms Reference:
CONTROL OVER MESSAGES	

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Send results
AGENTS INVOLVED	Sender: System Receiver: Patient

INFORMATION ITEMS	Role: core Form: a ranked set of recipes Medium: HTML ordered list, with hrefs to recipes
MESSAGE SPECIFICATION	Type: REPLY Content: the ranked set of recipes Reference:
CONTROL OVER MESSAGES	

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Send selection
AGENTS INVOLVED	Sender: User Receiver: System
INFORMATION ITEMS	Role: core Form: recipe identifier Medium: HTML button
MESSAGE SPECIFICATION	Type: ORDER Content: the identifier of the selected recipe Reference:
CONTROL OVER MESSAGES	

Communication Model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	Send recipe
AGENTS INVOLVED	Sender: System Receiver: Patient
INFORMATION ITEMS	The modified recipe Role: core Form: table presentation of ingredient names, quantities, and preparation methods Medium: HTML element Modified recipe parameter values Role: support Form: table presentation of parameter values & compliance with the active constraints Medium: HTML element
MESSAGE SPECIFICATION	Type: REPORT Content: the modified recipe, parameter values, and compliance with the constraints Reference:
CONTROL OVER MESSAGES	

4. Design model

4.1 Worksheet DM-1: System architecture

Architecture Decision	**System Architecture worksheet DM-1**
Subsystem architecture	Model View Controller (MVC) principles
Control model	There should be centralized control for the component (controller) which handles input.
Subsystem decomposition	The model is decomposed using OO principles.

As stated above, the subsystem of our system should be implemented with MVC principles. MVC is concerned with the separation of distinct aspects of an application. We argue that, given our incremental approach, we should first focus on implementation of the controller (control flow, input, output) and application model (tasks, inferences, domain schema and data). For the first iteration we implement a simple console which receives input and displays output to external agents. The next development iteration may feature the implementation of a more appealing graphical interface as a view component.

4.2 Worksheet DM-2: Target implementation platform

Design Model	**Target Implementation Platform worksheet DM-2**
Software package	Java EE
Potential hardware	Server: Hardware which supports an OS to run a Java Virtual Machine (JVM). The following listing presents the minimum resource requirements of the hardware in order to provide a system which is able to run our application while still being responsive; - OS: Mac OS, Windows or Unix - 1x 2.0 GHz core - 512MB memory - 2GB free space on hard disk Client: Hardware which supports web-browsing.
Target hardware	Any Java Server in a DataCenter would be suitable to host the main server of the application. Clients can be any personal device with a web access (desktop computer, laptop, smartphone, tablet).
Visualization library	Web standards (HTML/CSS/JS) connected to J2EE web-services.
Language typing	Strongly typed and OO

Knowledge representation	Procedural
Interaction protocols	- J2EE web-services - HTML forms
Control flow	-

5. The prototype

5.1 Description

During the short period we had to build a prototype, we decided to develop only the core idea of the future hypothetical system. We worked specifically on the knowledge-intensive task: modification of a recipe while respecting constraints.

We first started to develop the prototype using Pyke[5], a knowledge-based inference engine which works with Python[6], but is inspired by Prolog. We expected to be able to simply specify our domain model and different sets of rules to the engine, and to let it work for us on the modification through backward-chaining. However, Pyke considers assertions to be immutable; i.e. doesn't allow for facts to be modified or retracted during runtime. This did not suit the Propose-Critique-Modify method well, because a new (identifiable) recipe had to be created for every iteration. Unfortunately, because of our lack of experience in logic programming, the use of this engine became too complicated and time consuming for us.

Implementing the adjusting program ourselves, without relying on an existing inference engine then felt like a better solution. We decided to use Java[7] to represent our domain model in classes and to write a control regimen which executes the necessary inferences and provides a basic input and output mechanism.

5.2 How it works

5.2.1 Heuristic evaluation version

This heuristic evaluation version was our first approach. We then evaluated the recipe, by calculating how much difference there was with the different calculated properties of the recipe and their corresponding constraints. To adjust the recipe at each iteration, we considered every possible substitution and adjustment: specific substitutions, same-type substitutions and quantity adjustments. We then evaluated each of these possible new recipes, kept the best one, and re-iterated until the recipe would have an evaluation of 0 (no difference between properties and constraint meaning that the recipe is satisfying all constraints).

As shown in Figure 10, this was a kind of heuristic Hill Climbing algorithm[8], where we have an original state (the recipe), a way to create successors of this state (adjustments), and an

---

[5] Pyke: http://pyke.sourceforge.net/

[6] Python: http://www.python.org/

[7] Java: http://www.oracle.com/technetwork/java/

[8] Hill Climbing algorithm: http://en.wikipedia.org/wiki/Hill_climbing

evaluation function. As such, it is subject to the issues of local optima, and could therefore be improved with the use of meta-heuristic (i.e. Simulated Annealing).

```
while recipe.is_not_ok :
        best_recipe = recipe
        // try specific substitutions
        foreach ingredient of recipe :
                foreach specific_substitution of ingredient :
                        new_recipe = specific_substitution.apply(recipe)
                        if  new_recipe.is_better(best_recipe) :
                                best_recipe = new_recipe
        if best_recipe == recipe :
                // try same-type substitutions
                foreach ingredient of recipe :
                        foreach same_type_substitution of ingredient :
                                new_recipe = same_type_substitution.apply(recipe)
                                if  new_recipe.is_better(best_recipe) :
                                        best_recipe = new_recipe
        if best_recipe == recipe :
                // try quantity adjustments
                foreach ingredient of recipe :
                        new_recipe = ingredient.increase(recipe)
                        if  new_recipe.is_better(best_recipe) :
                                best_recipe = new_recipe

                        new_recipe = ingredient.decrease(recipe)
                        if  new_recipe.is_better(best_recipe) :
                                best_recipe = new_recipe
        if best_recipe == recipe :
                // impossible to adjust more
                return best_recipe
        else :
                // next step
                recipe = best_recipe
return recipe
```

Figure 10. Adapted Hill Climbing algorithm for recipe adjustment

This version worked quite well and was the one demonstrated to our teachers and classmates on Friday, January 25th. It corresponds to the commit 132ca6119a of our Github repository.

When finalising this document, we realised that our algorithm had gone in a different direction than what was planned. We tried to check the mapping between the inference structure steps, and the steps of this adjusting algorithm (see Figure 10), and realised that the mapping was far from perfect: we had an evaluation instead of a verification and the selection step relied on this evaluation instead of using the constraints.

5.2.2 Verify-Critique-Modify version

This second approach was built on top of the previous one, after a major modification of the main loop. It corresponds to the commit 822d1760e3 of our Github repository. Constraints classes were added to the code and the algorithm written as shown in Figure 11. To understand the pseudocode below, it is important to know that:

- the `recipe.verify()` method returns the constraint with the highest violated recipe value (e.g. recipe.glycemic-load) relative to the constraint value (e.g. maximum-

glycemic-load.value) ;

- the `critique(recipe)` method simply makes the list of all possible fix-actions for a given recipe ;
- the `select(fixActions, violation, recipe)` method keeps the fix-action which is the best at fixing the violated constraint.

```
while violation = recipe.verify() is not null :
        fixActions = critique(recipe)
        fixAction = select(fixActions, violation, recipe)
        fixAction.modify(recipe)
```

Figure 11. Verify-Critique-Modify algorithm for recipe adjustment

The mapping between this version and our inference structure (see Figure 12) then seems completely obvious, since each step of the inference structure now has a method with the same name, and the code is also more simple and clear.

Inference	**Hill Climbing version**	**V-C-M version**
Verify	Test: "`recipe.is_not_ok`"	`recipe.verify() is not null`
Critique	Possible fix-actions: "`foreach`" loops	`critique(recipe)`
Select	Test: "`recipe.is_better(other)`"	`select(fixActions, violation, recipe)`
Modify	Application: "`recipe = best_recipe`"	`fixAction.modify(recipe)`

Figure 12. Mapping between the inference structure and the algorithms

This version can still be seen as a Hill Climbing algorithm, since it proceeds incrementally, by successive improvements, and therefore is also subject to local optima.

While the second version seems to correspond more tightly to our models, the first version is arguably a better solution. The only difference between the two algorithms is in the way the multiple constraints are combined to select "the best" fix-action at each iteration. In the V-C-M approach, the selection of one fix-action depends only on one violation and the selected fix-action is the one which fixes that violation the most. In the heuristic evaluation, all constraints are combined into the evaluation method, and the selected fix-action is therefore the one which globally improves the recipe the most.

5.3 What is missing

For the algorithm to be complete, it should consider all possible fix actions. Here only substitutions and quantity adjustments are considered. Removals and additions should also be considered. In one way, removal are almost in already because the quantity is free to be adjusted down to zero, but still: both removals and additions should be added to the algorithm as separate fix actions.

For the system to be complete, it misses everything that goes around the core function which was prototyped here: the way diet advisers input constraints, the way users can research a recipe and get the adjusted version of it, and the way to manage and fill the knowledge base. The

gathering of ingredient properties was already partially addressed with a Python script that is able to automatically retrieve these properties through the Freebase API.

5.4 Example scenarios

The traces for the heuristic version are shown in APPENDIX B. The results seem very satisfying, since the recipes were adjusted in a few iterations to a recipe with a correct Glycemic Load. The use of an heuristic evaluation allows the algorithm to never get into an infinite loop, and the evaluation can also be made smarter to stop adjusting before all errors are fixed (which is sometimes impossible). In this version, the adjustments stops if the glycemic load is completely fixed and the remaining error is below a threshold value (20 for the demonstration).

Original	**Heuristic eval adjustment**	**V-C-M adjustment**
Fast lasagne Olive oil has qty=8.00 Ground pork has qty=450.00 Chili pepper has qty=4.50 Garlic has qty=8.00 Edible mushroom has qty=150.00 Creme fraiche has qty=215.00 Spinach has qty=100.00 Tomato has qty=400.00 Tomato paste has qty=20.00 Sugar has qty=1.50 Lasagnette has qty=75.00 Cheddar has qty=200.00	Fast lasagne Sunflower oil has qty=8.00 Ground beef has qty=366.53 Chili pepper has qty=4.50 Garlic has qty=8.00 Edible mushroom has qty=150.00 Sour cream has qty=215.00 Spinach has qty=100.00 Tomato has qty=400.00 Tomato paste has qty=20.00 Maple syrup has qty=1.50 Lasagnette has qty=75.00 Soft goat cheese has qty=200.00	Fast lasagne Sunflower oil has qty=8.00 Ground beef has qty=38.37 Chili pepper has qty=4.50 Garlic has qty=8.00 Edible mushroom has qty=150.00 Sour cream has qty=62.78 Spinach has qty=100.00 Tomato has qty=400.00 Tomato paste has qty=20.00 Sugar has qty=1.50 Lasagnette has qty=75.00 Soft goat cheese has qty=29.98
Beetroot chocolate cake Flour has qty=135.00 Chocolate has qty=200.00 Beet has qty=250.00 Baking powder has qty=1.00 Cocoa has qty=9.00 Sugar has qty=200.00 Egg has qty=280.00 Butter has qty=200.00	Beetroot chocolate cake Whole-grain wheat flour has qty=128.25 Chocolate has qty=200.00 Beet has qty=250.00 Baking powder has qty=1.00 Cocoa has qty=9.00 Maple syrup has qty=200.00 Egg has qty=280.00 Sunflower oil has qty=154.76	The system is not able to satisfy all constraints for this recipe. It keeps cycling between two substitution fix-actions. Without a break an OutOfMemoryError is thrown by the JVM.

Figure 13. Adjustment examples

The traces for the V-C-M version are shown in APPENDIX C. This algorithm was able to adjust the lasagne recipe after a lot more iterations, and could not satisfy all constraints for the adjustment of the chocolate cake. Figure 13 shows the two recipes and their respective adjustments. It is interesting to note that almost all ingredient substitutions were the same in both case, for the lasagne recipe reconfiguration. Only the sugar was replaced in one case by maple syrup and kept in the other case. The main difference is that the quantities of meat, cream and cheese were drastically reduced in the V-C-M case. This is because this approach satisfies all constraints, while the first one simply want to fully satisfy the GL constraint. It would then have to signal the user that the recipe still contains a bit too much fat and calories. The adjustment is therefore less perfect in the first case, but the recipe is kept more consistent.

For the chocolate cake, the second approach does not terminate. Because the algorithm does not stop unless all constraints are fixed, it can reach a situation where there is a cycle of fix actions, where two constraints are conflicting. In such case, the best that could be done would be to detect the cycle, and return one of these sensibly different recipes as the best possible

adjustment. Backtracking (which was one of the supposed advantages of Pyke) could also allow to explore more modification possibilities when cycles would be detected.

The first approach did give better results overall, but the second one was an interesting approach. Not only because it corresponded more to our original models, but also because it left more room for improvement: adding cycle detection, backtracking, and trying to have a “smart” adjustments instead of an “naïve” evaluation could give better results. Satisfying all constraint when possible would also be a better alternative than stopping as soon as the GL constraint is satisfied and the others are “good enough”. With the little time we had, though, the heuristic evaluation was simpler and gave results which seemed better.

One major improvement to both methods would be to consider culinary knowledge. By considering more constraints than the GL calculation, we tried to keep recipes consistent, but making sure that a recipe keep its characteristics and quality would be a real challenge. For instance, replacing the pork with beef in the lasagne completely changes the recipe.

6. Reflection

6.1 Reflection on project approach

Our team had good open communication, mostly via Google Hangout, email and chat. We used Google Documents for the collaboration at creating all reports and slides. The code was shared on github, to allow us to work together on the prototype(s).

We agreed to contact our supervisor only if we thought necessary. This resulted in one physical meeting to discuss our initial knowledge model and some emails.

The approach we took was inspired by Frank's "Spiral approach"; doing small iterations over the models and prototype to get smaller, but more manageable results. This approach proved to be successful for us.

6.2 Reflection on knowledge elicitation

The interview with Sèfi Willemsen was very interesting, but her knowledge was too general for our project. The sorted cards, main outcome of this interview, are presented in APPENDIX A. As a diet advisor, she knew about ingredient properties, and constraints to watch in a recipe to keep a consistent meal. She did not have particular knowledge about the specific issues related to glycemic load. This might be the reason why our system addressed a broader issue than initially planned in the end, which has a positive (achieved more than expected) and a negative (had more work and did not stick to the plans) aspects.

While on the phone with a dietician to make an appointment for an interview, we were directed to online documentation on diabetes diets (Nationale Diabetes Federatie, 2010) without making an appointment because the expert was very busy. A possibility to avoid this is to plan such appointments much earlier in the process. An issue which may have arisen from the obtained documentation, is our own (in)correct interpretation of these documents, which then results in the phenomena of "becoming one's own expert".

As for a potential user of our system, we consulted Zinzi van Nahuijs, who is experienced in modifying recipes for hypoglycemic symptoms. She provided valuable insights in how she would want to use a recipe adjustment system, which gave us the confidence that we were dealing with a real problem that needed to be addressed. Taking the structured elicitation approach of annotating recipes with her was very important to get an idea of the system output we would work towards early in the project. This also made clear that it would be difficult for us to represent the combination of dietary and culinary knowledge that she uses in our system.

6.3 Reflection on knowledge modeling

Initially we envisioned to target our system at diabetic and hypoglycemic patients. Therefore the

glycemic load property of a recipe was most important to be included as a computed value and constraint rules. While we were modeling these concepts, we noticed that we could easily incorporate other recipe properties (such as calories, proteins, fibers, saturated fats) as well. With these properties added we were now able to verify the recipe on these properties too, resulting in recipes which were more balanced.

An issue we faced during the initial modeling phase was that we were too inclined with the CommonKADS methodology when it came to domain schema reuse. We tried to squeeze our domain into the configuration design domain schema template which resulted in some confusion on our end. Eventually, we figured out how *not* to use the domain schema template and ignored aspects of the template which we agreed not a necessity in our domain.

Furthermore, we argue that separation of dietary - and culinary knowledge is very difficult. Currently, when modifications are made to the recipe, we can not guarantee similar characteristics and quality (e.g. texture, taste, color) of the resulting recipe in comparison to the original recipe. Culinary knowledge should i.e. capture an ingredient's function within a recipe or in relation to other ingredients to keep the original recipe's characteristics and quality most similar while it is being modified.

6.4 Reflection on prototyping

We started to explore which tools we could use for building the prototype when we finished our initial knowledge model. As argued in the Prototype section, we started to work with Pyke, but found that we were too inexperienced and that Pyke lacked functionality which was crucial to our envisioned solution. Working with Pyke caused us to lose about two to three days mainly because of its learning curve. Although it was an interesting experience, we argue that we naively followed this route without exploring the use of a general purpose language first.

Eventually, we started to use Java to build our prototype in a custom fashion. Since we were familiar with the language, we were able to prototype fast. As mentioned in the Prototype section, we implemented two approaches to the algorithm of recipe modification, which were both discussed in that previous section.

6.5 Future work

The direction of future work can be divided in knowledge model, prototype and evaluation.

The knowledge model can be further expanded to add more rule instances to the knowledge base for new constraints. The use of preferences (soft requirements), e.g. try to omit cheese, might also be an interesting addition to the model. Furthermore, the knowledge model should have concepts which describe a user profile which captures the diet constraints and preferences of one or multiple users. In the current knowledge model the course concept is used to store diet constraints per course.

Currently, the both prototypes only support substitution and quantity adjustment fix actions. We have modelled omission and addition fix actions but were unable to implement these in the current prototype. These fix actions have a strong dependency on culinary knowledge. Therefore

we argue that culinary knowledge should be further incorporated into the knowledge model before considering implementation of these fix actions.

Evaluation of the knowledge models and prototype results should be done with our expert. This may give valuable insights for discrepancies and next steps in the modeling and development process. Furthermore, real-life diet constraints (e.g. diabetes type 2 diets; high carbohydrates/ low protein - or (very) low calorie diets (Nederlandse Diabetes Federatie, 2010)) for an individual should be entered into the system by a diet adviser to evaluate whether the resulting modified recipes prove to be balanced and sound.

Bibliography

Chandrasekaran, B. (1990). Design problem solving: A task analysis. *AI magazine*, *11*(4), 59.

Foster-Powell, K., Holt, S.H.A., & Brand-Miller, J.C. (2002). International table of glycemic index and glycemic load values: 2002. *American Journal of Clinical Nutrition* 5, pp. 5-56.

van Harmelen, F., ten Teije, A., & Wache, H. (2009). Knowledge engineering rediscovered: towards reasoning patterns for the semantic web. In *K-CAP '09: Proceedings of the 5th international conference on Knowledge capture*, pp. 81-88, New York, NY, USA: ACM.

Nederlandse Diabetes Federatie (2010), Voedingsrichtlijn voor diabetes type 1 en 2., Amersfoort, retrieved from http://www.diabetesfederatie.nl/start/richtlijnen-en-adviezen/ndf-voedingsrichtlijnen-bij-diabetes-2010/download.html on 24/1/2013

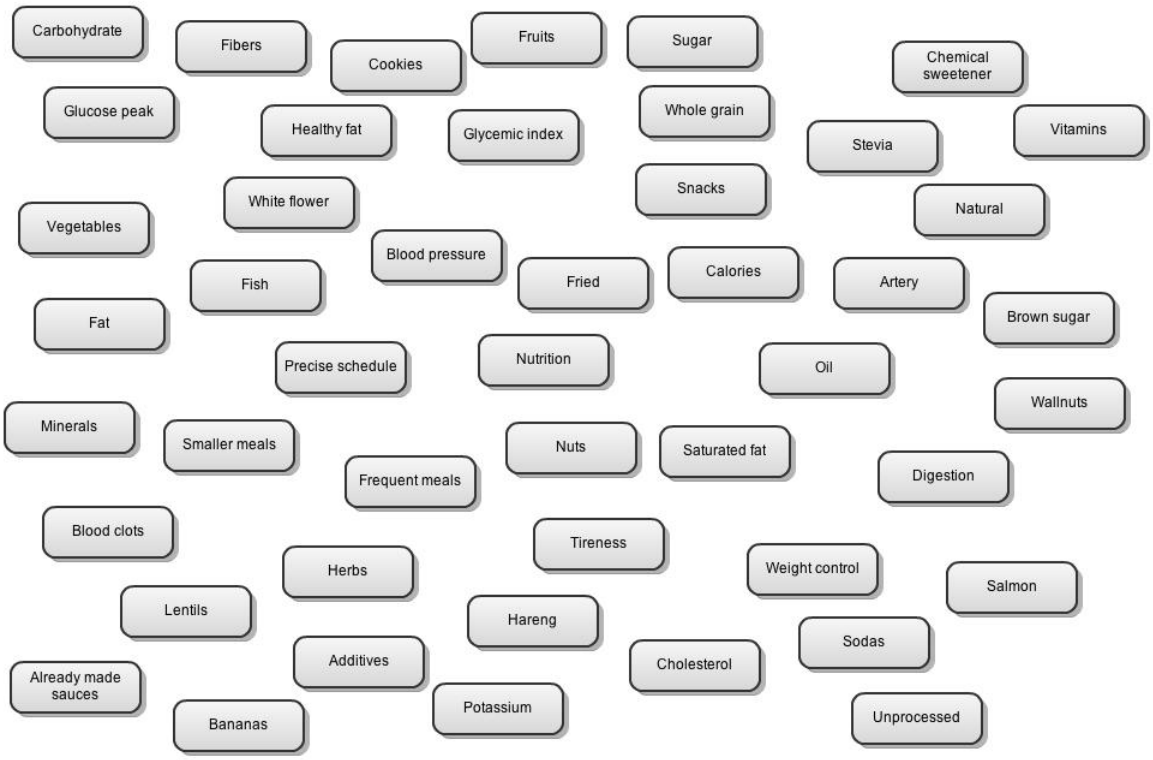
Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., van de Velde, W., & Wielinga, B. (1999). *Knowledge engineering and management: the CommonKADS methodology*. MIT press.

Wielinga, B., & Schreiber, G. (1997). Configuration-design problem solving. *IEEE Expert*, *12*(2), 49-56.

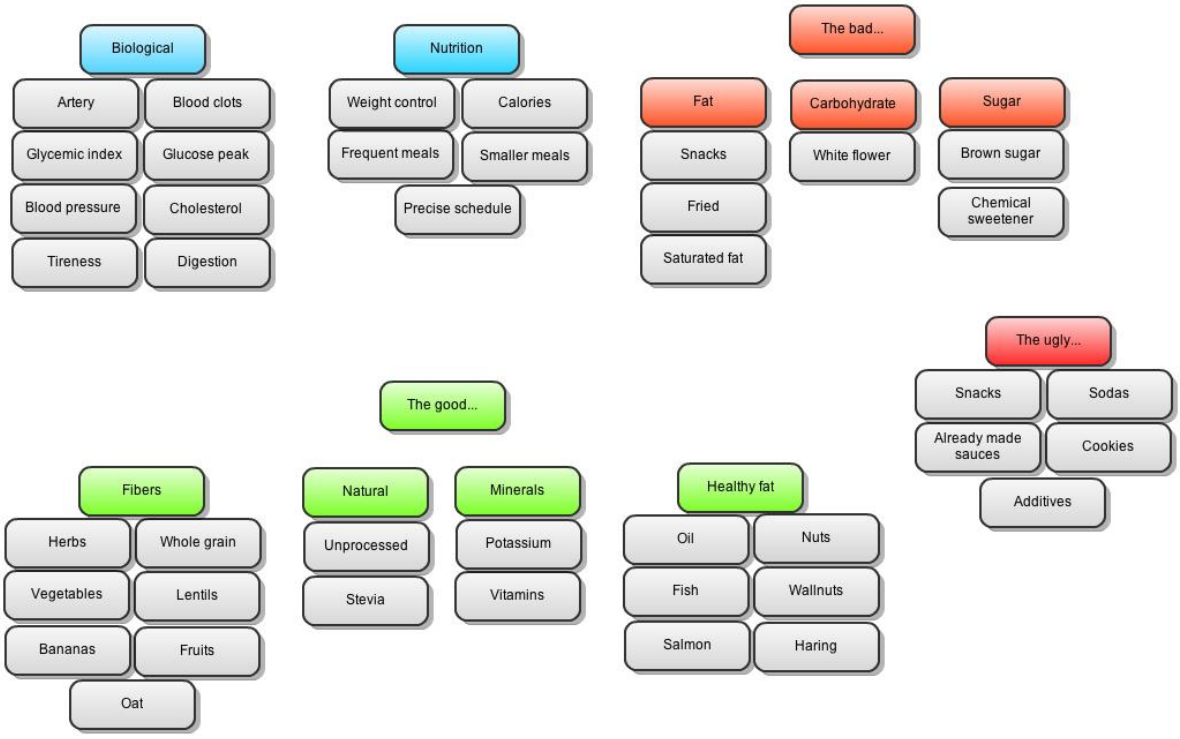
APPENDICES

APPENDIX A - Expert interview: card sorting

During the interview, we asked Sèfi Willemsen to give us any ideas that could come to her mind after we presented her our problem:



We then worked with her at sorting all these concepts:



All these were good research axis for us. In general it was still quite general and required a lot of work on our side to get specific Glycemic Load related knowledge and apply all of the knowledge (from Sèfi and ourselves) to the system.

APPENDIX B - Traces: “heuristic evaluation” version

B.1 Fast lasagne with pork and spinach

```
[LOG] Errors=80.01 (GL=0.00; fats=31.89; proteins=0.00; fibers=0.00; calories=481.12)
[LOG] Replacing Ground pork with Ground beef (scale: 1.00)
[LOG] Errors=44.67 (GL=0.00; fats=21.55; proteins=0.00; fibers=0.00; calories=231.24)
[LOG] Replacing Creme fraiche with Sour cream (scale: 1.00)
[LOG] Errors=32.53 (GL=0.00; fats=15.82; proteins=0.00; fibers=0.00; calories=167.10)
[LOG] Replacing Cheddar with Soft goat cheese (scale: 1.00)
[LOG] Errors=26.03 (GL=0.00; fats=13.82; proteins=0.00; fibers=0.00; calories=122.10)
[LOG] Replacing Olive oil with Sunflower oil (scale: 1.00)
[LOG] Errors=25.96 (GL=0.00; fats=13.75; proteins=0.00; fibers=0.00; calories=122.10)
[LOG] Replacing Sugar with Maple syrup (scale: 1.00)
[LOG] Errors=25.93 (GL=0.00; fats=13.75; proteins=0.00; fibers=0.00; calories=121.79)
[LOG] Adjusting Ground beef quantity from 450.00 to 427.50
[LOG] Errors=24.26 (GL=0.00; fats=13.33; proteins=0.00; fibers=0.00; calories=109.34)
[LOG] Adjusting Ground beef quantity from 427.50 to 406.13
[LOG] Errors=22.68 (GL=0.00; fats=12.93; proteins=0.00; fibers=0.00; calories=97.51)
[LOG] Adjusting Ground beef quantity from 406.13 to 385.82
[LOG] Errors=21.17 (GL=0.00; fats=12.55; proteins=0.00; fibers=0.00; calories=86.28)
[LOG] Adjusting Ground beef quantity from 385.82 to 366.53
[LOG] Errors=19.74 (GL=0.00; fats=12.18; proteins=0.00; fibers=0.00; calories=75.60)

[ORIGINAL] Recipe: Fast lasagne for 6 (Course 'starter')
Olive oil has qty=8.00
Ground pork has qty=450.00
Chili pepper has qty=4.50
Garlic has qty=8.00
Edible mushroom has qty=150.00
Creme fraiche has qty=215.00
Spinach has qty=100.00
Tomato has qty=400.00
Tomato paste has qty=20.00
Sugar has qty=1.50
Lasagnette has qty=75.00
Cheddar has qty=200.00

[ADJUSTED] Recipe: Fast lasagne for 6 (Course 'starter')
Sunflower oil has qty=8.00
Ground beef has qty=366.53
Chili pepper has qty=4.50
Garlic has qty=8.00
Edible mushroom has qty=150.00
Sour cream has qty=215.00
Spinach has qty=100.00
Tomato has qty=400.00
Tomato paste has qty=20.00
Maple syrup has qty=1.50
Lasagnette has qty=75.00
Soft goat cheese has qty=200.00
```

B.2 Beetroot chocolate cake

```
[LOG] Errors=52.06 (GL=11.52; fats=18.91; proteins=0.00; fibers=0.00; calories=216.32)
[LOG] Replacing Sugar with Maple syrup (scale: 1.00)
[LOG] Errors=37.30 (GL=0.94; fats=18.92; proteins=0.00; fibers=0.00; calories=174.41)
[LOG] Replacing Butter with Sunflower oil (scale: 1.00)
[LOG] Errors=28.75 (GL=0.94; fats=4.80; proteins=0.00; fibers=0.00; calories=230.18)
[LOG] Replacing Flour with Whole-grain wheat flour (scale: 1.00)
[LOG] Errors=27.61 (GL=0.00; fats=4.86; proteins=0.00; fibers=0.27; calories=224.78)
[LOG] Adjusting Sunflower oil quantity from 200.00 to 190.00
[LOG] Errors=25.99 (GL=0.00; fats=4.71; proteins=0.00; fibers=0.27; calories=210.05)
[LOG] Adjusting Sunflower oil quantity from 190.00 to 180.50
[LOG] Errors=24.45 (GL=0.00; fats=4.57; proteins=0.00; fibers=0.27; calories=196.05)
[LOG] Adjusting Sunflower oil quantity from 180.50 to 171.48
[LOG] Errors=22.98 (GL=0.00; fats=4.43; proteins=0.00; fibers=0.27; calories=182.75)
[LOG] Adjusting Sunflower oil quantity from 171.48 to 162.90
[LOG] Errors=21.59 (GL=0.00; fats=4.30; proteins=0.00; fibers=0.27; calories=170.12)
[LOG] Adjusting Sunflower oil quantity from 162.90 to 154.76
[LOG] Errors=20.27 (GL=0.00; fats=4.18; proteins=0.00; fibers=0.27; calories=158.12)
[LOG] Adjusting Whole-grain wheat flour quantity from 135.00 to 128.25
[LOG] Errors=19.75 (GL=0.00; fats=4.18; proteins=0.00; fibers=0.15; calories=154.29)

[ORIGINAL] Recipe: Beetroot chocolate cake for 6 (Course 'dessert')
Flour has qty=135.00
Chocolate has qty=200.00
Beet has qty=250.00
Baking powder has qty=1.00
Cocoa has qty=9.00
Sugar has qty=200.00
Egg has qty=280.00
Butter has qty=200.00

[ADJUSTED] Recipe: Beetroot chocolate cake for 6 (Course 'dessert')
Whole-grain wheat flour has qty=128.25
Chocolate has qty=200.00
Beet has qty=250.00
Baking powder has qty=1.00
Cocoa has qty=9.00
Maple syrup has qty=200.00
Egg has qty=280.00
Sunflower oil has qty=154.76
```

APPENDIX C - Traces: “verify-critique-modify” version

C.1 Fast lasagne with pork and spinach

```
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=831.12, Fats=34.89, Proteins=17.66, Fibers=1.69
[LOG] Replacing Ground pork with Ground beef (scale: 1.00)
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=581.24, Fats=24.55, Proteins=24.06, Fibers=1.69
[LOG] Replacing Creme fraiche with Sour cream (scale: 1.00)
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=517.10, Fats=18.82, Proteins=24.42, Fibers=1.69
[LOG] Replacing Cheddar with Soft goat cheese (scale: 1.00)
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=472.10, Fats=16.82, Proteins=22.42, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 450.00 to 427.50
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=459.65, Fats=16.39, Proteins=21.88, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 427.50 to 406.13
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=447.83, Fats=15.99, Proteins=21.37, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 406.13 to 385.82
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=436.59, Fats=15.61, Proteins=20.88, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 385.82 to 366.53
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=425.92, Fats=15.25, Proteins=20.42, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 366.53 to 348.20
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=415.78, Fats=14.90, Proteins=19.98, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 348.20 to 330.79
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=406.14, Fats=14.57, Proteins=19.57, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 330.79 to 314.25
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=396.99, Fats=14.26, Proteins=19.17, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 314.25 to 298.54
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=388.30, Fats=13.97, Proteins=18.80, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 298.54 to 283.61
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=380.04, Fats=13.69, Proteins=18.44, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 283.61 to 269.43
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=372.20, Fats=13.42, Proteins=18.10, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 269.43 to 255.96
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=364.74, Fats=13.17, Proteins=17.78, Fibers=1.69
[LOG] Adjusting Soft goat cheese quantity from 200.00 to 190.00
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=360.27, Fats=12.92, Proteins=17.46, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 255.96 to 243.16
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=353.19, Fats=12.67, Proteins=17.16, Fibers=1.69
[LOG] Adjusting Soft goat cheese quantity from 190.00 to 180.50
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=348.95, Fats=12.44, Proteins=16.85, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 243.16 to 231.00
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=342.22, Fats=12.21, Proteins=16.56, Fibers=1.69
[LOG] Adjusting Soft goat cheese quantity from 180.50 to 171.48
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=338.19, Fats=11.98, Proteins=16.28, Fibers=1.69
[LOG] Adjusting Ground beef quantity from 231.00 to 219.45
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=331.80, Fats=11.77, Proteins=16.00, Fibers=1.69
[LOG] Adjusting Soft goat cheese quantity from 171.48 to 162.90
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(3.0)
[LOG] GL=5.48, Calories=327.97, Fats=11.55, Proteins=15.73, Fibers=1.69
```

C.2 Beetroot chocolate cake

```
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=46.52, Calories=716.32, Fats=24.91, Proteins=11.99, Fibers=3.41
[LOG] Replacing Butter with Sunflower oil (scale: 1.00)
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=46.52, Calories=772.09, Fats=10.79, Proteins=11.71, Fibers=3.41
[LOG] Adjusting Chocolate quantity from 200.00 to 190.00
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=46.08, Calories=763.17, Fats=10.48, Proteins=11.58, Fibers=3.35
[LOG] Adjusting Chocolate quantity from 190.00 to 180.50
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=45.65, Calories=754.69, Fats=10.18, Proteins=11.46, Fibers=3.30
[LOG] Adjusting Chocolate quantity from 180.50 to 171.48
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=45.25, Calories=746.64, Fats=9.91, Proteins=11.35, Fibers=3.24
[LOG] Adjusting Chocolate quantity from 171.48 to 162.90
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=44.87, Calories=739.00, Fats=9.64, Proteins=11.24, Fibers=3.20
[LOG] Adjusting Chocolate quantity from 162.90 to 154.76
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=44.51, Calories=731.73, Fats=9.39, Proteins=11.13, Fibers=3.15
[LOG] Adjusting Chocolate quantity from 154.76 to 147.02
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=44.16, Calories=724.83, Fats=9.15, Proteins=11.03, Fibers=3.11
[LOG] Adjusting Chocolate quantity from 147.02 to 139.67
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=43.84, Calories=718.28, Fats=8.92, Proteins=10.94, Fibers=3.06
[LOG] Adjusting Chocolate quantity from 139.67 to 132.68
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=43.52, Calories=712.05, Fats=8.71, Proteins=10.85, Fibers=3.02
[LOG] Adjusting Chocolate quantity from 132.68 to 126.05
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=43.23, Calories=706.13, Fats=8.50, Proteins=10.77, Fibers=2.99
[LOG] Adjusting Chocolate quantity from 126.05 to 119.75
[LOG] Violation: class keadjustor.constraints.MaximumCalories(500.0)
[LOG] GL=42.95, Calories=700.51, Fats=8.31, Proteins=10.69, Fibers=2.95
[LOG] Replacing Sunflower oil with Butter (scale: 1.00)
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=42.95, Calories=644.74, Fats=22.43, Proteins=10.97, Fibers=2.95
[LOG] Replacing Butter with Sunflower oil (scale: 1.00)
[LOG] Violation: class keadjustor.constraints.MaximumCalories(500.0)
[LOG] GL=42.95, Calories=700.51, Fats=8.31, Proteins=10.69, Fibers=2.95
[LOG] Replacing Sunflower oil with Butter (scale: 1.00)
[LOG] Violation: class keadjustor.constraints.MaximumSaturatedFat(6.0)
[LOG] GL=42.95, Calories=644.74, Fats=22.43, Proteins=10.97, Fibers=2.95
[LOG] Replacing Butter with Sunflower oil (scale: 1.00)

...

Note: replacement of Sunflower oil with Butter and vice versa goes on. Ultimately the system
can not satisfy all constraints.
```