Formal Modelling for Cooking Assistance

Bernd Krieg-Brückner, Serge Autexier, Martin Rink, and Sidoine Ghomsi N.

German Research Center for Artificial Intelligence, DFKI, and
Universität Bremen, Germany
{Bernd.Krieg-Brueckner, Serge.Autexier}@dfki.de,
{mrink, sidoine}@informatik.uni-bremen.de

Abstract. Structured ontologies, with various facets of abstraction, are used to model food, ingredients, recipes, cookware and workflows. They form the uniform knowledge base for modular software assistants. Processes and monitors supervise the cooking process and advise the user.

1 Introduction

The Objective of this Paper is to show the complexity of Formal Modelling for an application domain such as *cooking*, but at the same time to introduce "Formal Methods Light" step by step and to illustrate their added value:

modelling data	in ontologies, analogous to data types, with more semantic rigor than in relational data bases;
flexibility, extendability	of ontologies, easier to maintain than data types;
separation of concerns	by structuring into domain ontologies;
abstraction	in several ways to conquer complexity;
modelling processes	at a high level, in particular for monitoring.

On the side of the application domain, the objective is to propose a *uniform approach* for the *integration* of the many aspects of *cooking*, as a basis for software "assistants", which access the knowledge base and present it appropriately for user interaction, see Sec. 5.1.

How is our protagonist, Ms. W., going to cook when several guests with all sorts of health and other constraints are being invited together (cf. Sec. 7)? To try to solve this and related problems, we shall accompany Ms. W. while she plans a meal, develops recipes, manages ingredients, goes shopping, prepares for cooking, and finally gets the cooking done — supported by DFKI's emerging technology, explained as we go along.

2 Food, Drink and Health

Food and drink are most likely the most important source of a persons well-being. Chinese grandfathers and -mothers, when still living at home in the traditional multi-generation family, supposedly live longer [26], for physical reasons, since they can expect a very diverse diet, but also for psychological reasons, since they

are pampered with a varying and attractive meal three times a day, and live by looking forward to the next. Other great cuisines (such as the Italian or French Cuisine) also thrive on variety, locally grown ingredients — and eating (slowly!) with the family, friends, or at least colleagues, around a big table.

In modern Western (and increasingly other) societies, people suffer from health problems due to stress and hectic eating without the soothing effects of friendly society, but also unhealthy food products provided by an inconsiderate and greedy food industry, marketing dietary dreams that turn out to deteriorate health even further (e.g. "low fat", where fat is substituted by sugar to provide "taste"). One of the most important (health care political) issues is that food producers are still not obliged to (and therefore do not) provide complete details about the composition of their products, and consumers are not sufficiently informed about the effect of these products on their individual health.

Luckily, there are some (government, non-profit, and commercial) organisations, which try to provide the missing information about existing products, and software (apps) to access it [29, 21, 22, 27, 32, 8]. However, information about various aspects and their interrelation is still widely dispersed, often not directly accessible to the layman, and not integrated.

We shall try to delineate an approach in the sequel to unify, integrate and standardise such information to achieve a *personalized* added value for the user.

2.1 Food Classification and Properties

Ontologies. Let us start by classifying food and drink products such that we can then add meaningful properties and relationships to other concepts. A hierarchy of concepts, a taxonomy, becomes an *ontology* when relations or more semantics are added. Every concept X at a lower level is subsumed by the parent *class* C, the concept at the next higher level; we say X *is-a* C (a directed acyclic graph). In Fig. 1, CourgetteVegetable *is-a* ... *is-a* SquashVegetable *is-a* GourdFruitVegetable and eventually ... *is-a* Vegetable, and so on upwards in the hierarchy.

Abstraction of Properties is a central strength of modelling with ontologies. In the hierarchy of concepts, each intermittent *class* concept *abstracts away* from more particular, specialised properties of descendants at lower levels, retaining the properties that hold for all descendants, while some may not hold any more for ancestors or siblings. Indeed, *class*es may be declared to be *disjoint*, such that (sub)*class*es (and associated specialised properties) cannot be shared. Unless stated, hierarchies will be disjoint in the sequel.

Relations. The *relation* *is-a* is the standard *relation* between (sub)*class*es. The interrelation between concepts is the core of semantics. As an example, take *fromPlant* that relates a sub*class* of PlantProduct to the biological sub*class* of Plant of which it is a part, see Sec. 2.2; *isSourceOfPlantProduct* is its *inverse*.

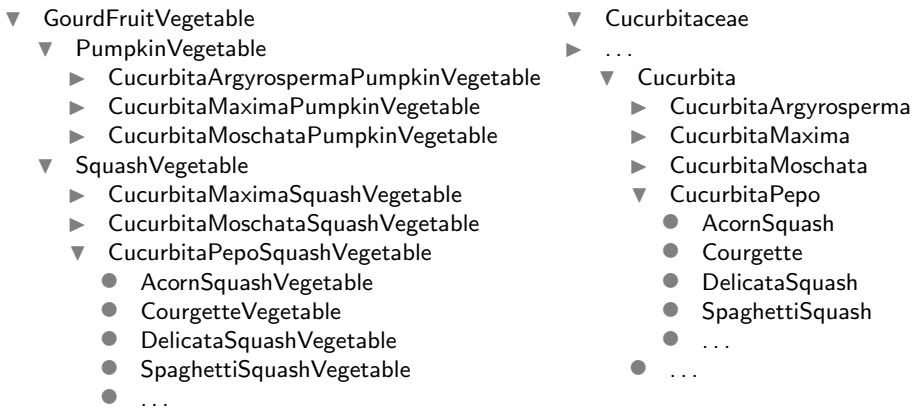

Fig. 1. PlantProduct and Plant Ontologies (Excerpts)

Multi-Lingual Ontologies. Ontologies can be made multi-lingual by attaching names or language terms as *label*s to a *class* name, one term for each desired language (or more if there are synonyms). This helps in a search, but also for automatic translation, for example of recipes. While there may be many *label*s, denoting synonyms in the same language, a *class* name is always unique.

When we want to look up *Zucchini* in German, for example, we find the class CourgetteVegetable, since *Zucchini* in German, Italian, and American English, is called *Courgette* in French, and British English — and we use primarily European terms for *class* names, i.e. British over American English.

We also take care of different traditions, e.g. by modelling German, French and English/American butcher's cuts of meat, relating them appropriately.

2.2 Where the Food Comes From

Biological Source. In fact, resolving such equivalencies between terms in several languages, even synonyms within the same language family such as English, with dictionaries alone may lead to inaccurate results, since "common names" for plants or animals are often overlapping, ambiguous, or misleading.[1]

To be safe, we should resort to relating each *food class* (and the associated linguistic labels) with the proper *biological class* with the relation *fromPlant* or *fromAnimal*, respectively. Biologists have been using taxonomic hierarchies for centuries (since Linné) to uniquely identify animals and plants (in Latin, the common language of scientists of the time), and to group them according to hereditary variations of properties. This way, we also relate breeds of cultivated plants or domestic animals to the respective "wild forms" of their ancestors.

Squash and *pumpkin* are examples of "common names"[2] *distinguishing* and classifying groups of vegetables with certain *culinary* properties, cf. Sec. 2.1. In

[1] *Savoy cabbage* is confusingly called *chou de Milan* in French (*Wirsing* in German); *red cabbage* is regionally called *Rotkohl* or *Blaukraut*, as cooking changes its colour.

[2] Europeans need help with primarily American breeds of *squash*es and *pumpkin*s.

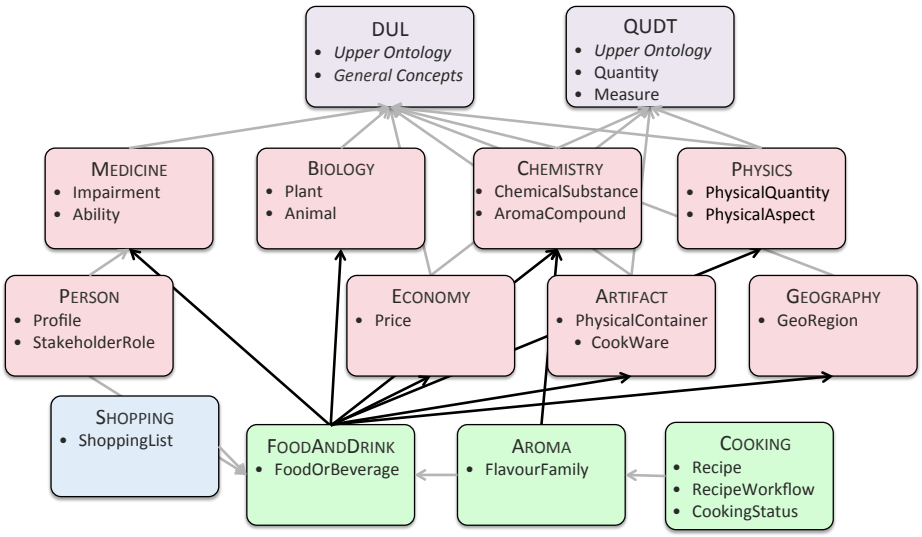

Fig. 2. Domain Ontologies and Import Structure

Fig. 1, SquashVegetable and PumpkinVegetable are *separate class*es from a "vegetable/culinary point of view": SquashVegetables can be cooked and eaten whole, whereas PumpkinVegetables have a hard shell, only the inside of the pumpkin shell is edible. However, they are closely *related as plants*: both are *fromPlant* Cucurbita. For example, some kinds of SquashVegetable and some kinds of PumpkinVegetable *both* are *fromPlant* CucurbitaMoschata, i.e. the *same biological class*.

Constraints on Relations in OWL. The Ontology Web Language, OWL [6], is the standard for the formulation of ontologies. Intricate relationships such as

"all SquashVegetables are related with *fromPlant* to *only* Cucurbita"

may be axiomatised as a *subclass constraint* on the relation *fromPlant*:

SquashVegetable $\sqsubseteq$ *fromPlant* **only** Cucurbita.

Structuring Ontologies. It is a good idea to structure the multitude of ontologies into separate *domain ontologies*, where one ontology *import*s (*class*es, *relation*s, etc.) from other ontologies, cf. Fig. 2, sometimes called "hyper-ontology" [16, 25]. A language for structuring ontologies by imports and morphisms is now proposed as a standard for extending OWL [25, 28]. Ontologies defining very general concepts are called *upper ontologies*; we use DUL [3] (derived from DOLCE [2]) and QUDT [7] (for standardised quantities and measures).

Data Abstraction, Instances. An actual data object, e.g. a particular food product, is modelled as an *instance* of a *class* (i.e. a member of the *class* regarded

as a set), and serves as source or target for the relations contained in the data. Thus the modelling by a ontology abstracts from the particular properties of tens or hundreds of thousands of products contained in data bases, and provides additional information by deductions as an "added value". In fact, with today's technology, these data cannot all be held as *instance*s; instead, data base access from the ontology to several external *data bases* is provided in a hybrid approach, such that only some *instance*s are held as local (copies of) objects.

Integration of Sources for Domain Modelling. Notice the large variety of aspects related to food or beverage products. It is the benefit of our modelling that we integrate and structure this variety inspired by several sources.

The internet portals WikiFood [8], or Barcoo [1] provide a (rather coarse) taxonomy and description of food and beverage products likely to be found in (European, German) food stores. WikiFood is a non-commercial portal focussing on the composition of food regarding nutrition or substances that might lead to incompatibilities; a distinctive feature is the personalized filter for food additives or content substances. WikiFood provides translation into English, German and French. Barcoo maps directly from the barcode to a variety of product information. The up-to-date management of their data bases relies on information from manufacturers, but also strongly on the community of users providing content. Challenging problems are the medical relevance and the quality (in particular the "half-life") of data regarding content substances (cf. Sec. 3.1, [11, 10]). Compare also the overview of food standards in [22, 17], in particular the CEN standard.

While we want to access such portals as data bases for actual food products on the market, we have to do the (integration of the) modelling, and mapping between possibly different models ourselves. The upper part of the FoodOrBeverage taxonomy (not shown here, including PlantProduct in Fig. 1) follows the hierarchy of the European Food Information Resource, EuroFIR [4, 27, 18], which is intended as a standard for organisations, industry, and researchers in Europe.

To enable exchange and comparison of data, an approach to indexing of data bases was established: the multi-lingual Langua aLimentaria Thesaurus, LanguaL [5, 29, 21]. Langual defines some relations to target domains we are modelling, but lacks e.g. information about nutrition impairments (cf. Sec. 3.1).

3 Planning a Meal

3.1 Guests and their Peculiarities

Restricted Diets, Nutrition Impairments. When Ms. W. invites guests for dinner, she may be faced with all sorts of peculiarities: a guest may have a mere preference for a particular diet, such as a NoFlavorEnhancerDiet, or may insist on a meatless diet, such as an OvoLactoPescetarianDiet, a religiously restricted diet, such as a HalalDiet, a culturally restricted diet, such as a NoInnardsDiet, or have a more or less severe NutritionImpairment requiring a medically restricted diet, e.g. a PregnancyDietRestriction with specific *requiresDiet* constraints, cf. the list

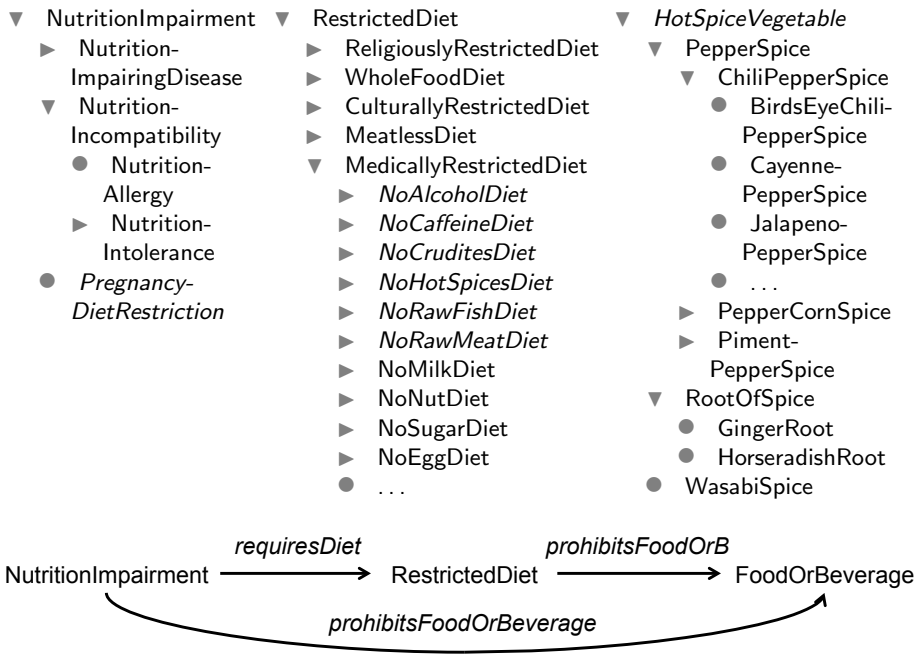

Fig. 3. PregnancyDietRestriction, RestrictedDiets, and Prohibited HotSpiceVegetables

of MedicallyRestrictedDiets marked in italics in Fig. 3. Note that diet restrictions might be elicited anonymously over a form on the Internet; anyway, a guest may state a list of RestrictedDiets individually and need not reveal her pregnancy.

Several Guests, Joining Impairments. When Ms. W. plans a meal for several such guests, she has to join impairments and associated dietary restrictions, thus the allowed foods. Similar considerations apply to a group in a restaurant.

3.2 Relating Impairments to Allowed Foods.

Intermediate Abstraction. When relating PregnancyDietRestriction to allowed foods, cf. Fig. 3, it has been convenient to introduce the extra class hierarchy RestrictedDiet as an *intermediate abstraction*. It allows us to relate NutritionImpairment via *requiresDiet* to RestrictedDiet on the left, and RestrictedDiet via *prohibitsFoodOrB* to FoodOrBeverage on the right; otherwise each relationship between PregnancyDietRestriction and prohibited FoodOrBeverages would have to be defined individually for an overall *relation* *prohibitsFoodOrBeverage*.

On the left, we can focus on all those subclasses of RestrictedDiet that should be related to PregnancyDietRestriction, and define *subclass constraints*, e.g.

PregnancyDietRestriction $\sqsubseteq$ *requiresDiet* **only** NoHotSpicesDiet

(cf. Sec. 2.2) and analogously for NoAlcoholDiet, NoCaffeineDiet, NoRawMeatDiet, NoRawFishDiet, NoCruditesDiet. Other NutritionImpairments are similarly related to *particular* sub*class*es of MedicallyRestrictedDiet.

On the right, we can limit our attention to each sub*class* of RestrictedDiet and its relation to FoodOrBeverage, e.g. for NoHotSpicesDiet to all spicy-hot food

NoHotSpicesDiet $\sqsubseteq$ *prohibitsFoodOrB* **only** HotSpiceVegetable

and similarly for other hot food. Note that it helps considerably to cluster food into *class*es with culinary aspects, but also to define extra (super)*class*es with other properties, such as the spicy-hot aspect; then we need to define *subclass constraints* only for the clustering super*class*es, and they are inherited. In the case of HotSpiceVegetable, the culinary and the special spicy-hot aspects coincide: we distinguish the CayennePepperSpice as a HotSpiceVegetable from the BellPepperVegetable as a bland PepperFruitVegetable (although CayennePepperSpice and BellPepperVegetable are both *fromPlant* CapsicumAnnuum, cf. Sec. 2.2). The *relation* *prohibitsFoodOrBeverage* is defined as a *composite relation*

prohibitsFoodOrBeverage $\sqsubseteq$ *requiresDiet* $\circ$ *prohibitsFoodOrB*;

any HotSpiceVegetable is *deduced* to be prohibited for a PregnancyDietRestriction.

Separation of Concerns. The clou of *intermediate abstraction* is that *requiresDiet* and *prohibitsFoodOrB* can be described *independently*. Perhaps even more importantly, it allows us to define *subclass constraints* (cf. Sec. 2.2) for both relations *separately* at a high level of property abstraction, cf. Sec. 2.1.

Relationships established by *prohibitsFoodOrB* can be *reused* for other diet restrictions, e.g. NoHotSpicesDiet for the NutritionImpairment Gastritis.

3.3 Meals, Courses, Dishes

For the planning of a meal, potentially with a number of courses, dishes, side-dishes, etc. (cf. Sec. 5.3), we have to consider the number of guests and their joint restrictions, choose from a variety of cuisines, and select among the multitude of recipes (or invent a new one). What is the culinary secret for the combination of dishes? for a dish with an accompanying wine? or for the ingredients in a dish?

The secret is the interaction or "interplay" of aromas, their harmony, but also the contrast, coverage and variety of different *flavours* in a dish (or a combination of dishes); moreover, a similar harmony and variety of textures, colours and shapes matters, which we will disregard here.

Flavour Affinities. Why does caviar taste good with white chocolate? or Ms. W.'s heavenly Bavarian cream with raspberry sauce?

There has been considerable research in the analysis of aromas and their chemical composition. *"Food pairing"* relates two ingredients that have one (or more) flavour(s) in common: e.g. for caviar and white chocolate the flavour determining substance trimethylamine. It has become quite popular among food researchers and technologists, star chefs, sommeliers, even perfumers.

"Pairing" refers to a *semantic neighborhood* of a flavour (or aroma) that is shared by two ingredients in harmony.

Caviezel, in a commendably scientific approach, introduces a hierarchy of *flavour levels* in [19], starting with the *taste* level (sweet, sour, salty, bitter, umami, fat), the flavour created in the mouth by taste buds on the tongue, continuing with *aromas* sensed by the nose, ordered in 8 levels according to the volatility of the corresponding molecules. Thus a flavour at a low level is usually more prominent and persistent; some herbs or spices may overpower others (e.g. "spicy hot" from chili). Note that (the stage of) the cooking may significantly influence or even create a flavour, e.g. when roasting meat. In general, an ingredient contains several flavours that are more or less salient, and is thus related "in several directions" to other ingredients. Thus complex and elaborate recipes can be analysed w.r.t. the harmony and intentional contrast in their composition.

The net of [9] shows 381 regularly used ingredients and 1021 aroma substances. To conquer such complexity, we hope to achieve a manageable set of *intermediate flavour abstractions* (perhaps Caviezel's flavour level sets), which allow us to constructively *propose compositions* of ingredients, or *substitutions* of alternative ingredients in existing recipes, for creative cooking.

4 Recipes

4.1 Recipe Structure

Cooking might be defined as the process of performing certain cooking steps on a defined amount of ingredients in a specific order, utilizing cooking utensils, tools, etc. A recipe is then a structured workflow for processing such cooking steps, prescribed by recipe instructions, with corresponding ingredients (cf. Sec. 5.3).

We shall propose a structure for modelling recipes below, which takes care of a variety of "culinary" semantic relationships; for a running example, see Fig. 4 for an Italian *zucchini frittata*, a courgette omelette. The rendering in Fig. 4, ignoring the nested boxes, is similar to what you might expect in a cookbook.

Primary Ingredient(s), Culinary Options. The composition of ingredients is, quite likely, the most characteristic feature of a recipe. Often, a user will search for a recipe with one *primary ingredient*, and choose the others accordingly (cf. pairing in Sect. 3.3). The recipe author should flag, whether an ingredient is optional — an important semantic indication providing freedom for the user:

***essential*:**	not to be omitted
***primary*:**	essential reference ingredient, giving the recipe its name
***optional*:**	dispensable for a restricted diet or by personal preference
***culinary*:**	optional, intended as a special "culinary kick" by the author that would be lost if omitted (or dispensable as a fad of that author)?

In a vegetable omelette, eggs are essential; adding anchovies and capers to bland cauliflowers adds a Mediterranean culinary touch (cf. Sec. 4.2). Deleting an optional Ingredient, e.g. pepper, also deletes the dependent RecipeInstruction(s).

Balancing Amounts, Intervals. Amounts, for example, are likely to be defined in terms of the amount for a *primary ingredient*, in particular, if its quantity cannot be influenced; for example a large rather than a small turkey; for a jam, fruit (as much as could be collected) matched on-to-one by sugar; in baking, just so much yeast per flour quantity. This important dependency should be reflected in the recipe, and tools should calculate dependent measures automatically.

While it is important in such cases to keep amounts and balancing strictly controlled, the precise definition of amounts is often over-specified. The author of a recipe should recommend the interval over which the amount of an ingredient may range based on her/his expertise (and maybe indicate a preference), such that the user may vary according to her/his personal taste or other constraints.

A recommended interval should also make it easier to achieve proper rounding of measures when recomputing for a different number of portions. If, for example, for 4 portions of an omelette, 5-7 (instead of 6) eggs are prescribed, then an omelette for 3 should have 4-5 (and not 4.5) eggs in it.

Measures. There are various different approaches to measure ingredients, depending on the cultural background in different geographic regions. While flour is measured in weight (i.e. mass) in Germany, it is measured in volume in the UK, the US, or Sweden; moreover, measurement units differ. We use the QUDT ontology [7], providing quantities and measurement units, and their relation to each other; so standards can be converted to a style preferred by the user, e.g. 0.23 Liters to a LiquidCupUS; the intervals above help rounding off.

Individual Adaptation of ingredients (adjusting amounts, omission, or substitution) now becomes possible, regarding the variety of dietary constraints, see Sec. 3.1 — and flavour affinities should help find tasty substitutions, see Sec. 3.3.

In view of the abundant minced fish in Denmark, BKB substituted bacon by fish in an *Ærø zucchini frittata*, suitable for an OvoLactoPescetarianDiet.

Recipe Instructions. RecipeInstructions (see Fig. 5) have been modelled to

set up an environment	for cooking, i.e. get the requisite CookWare (see below), add Ingredients, heat the Burner, serve or store away result Ingredients (temporarily or for preservation), clean and restore CookWare for further use;
prepare Ingredients	e.g. cut in a particular way, mix, or whisk;
cook Ingredients	in the present environment, e.g. braise, or fry.

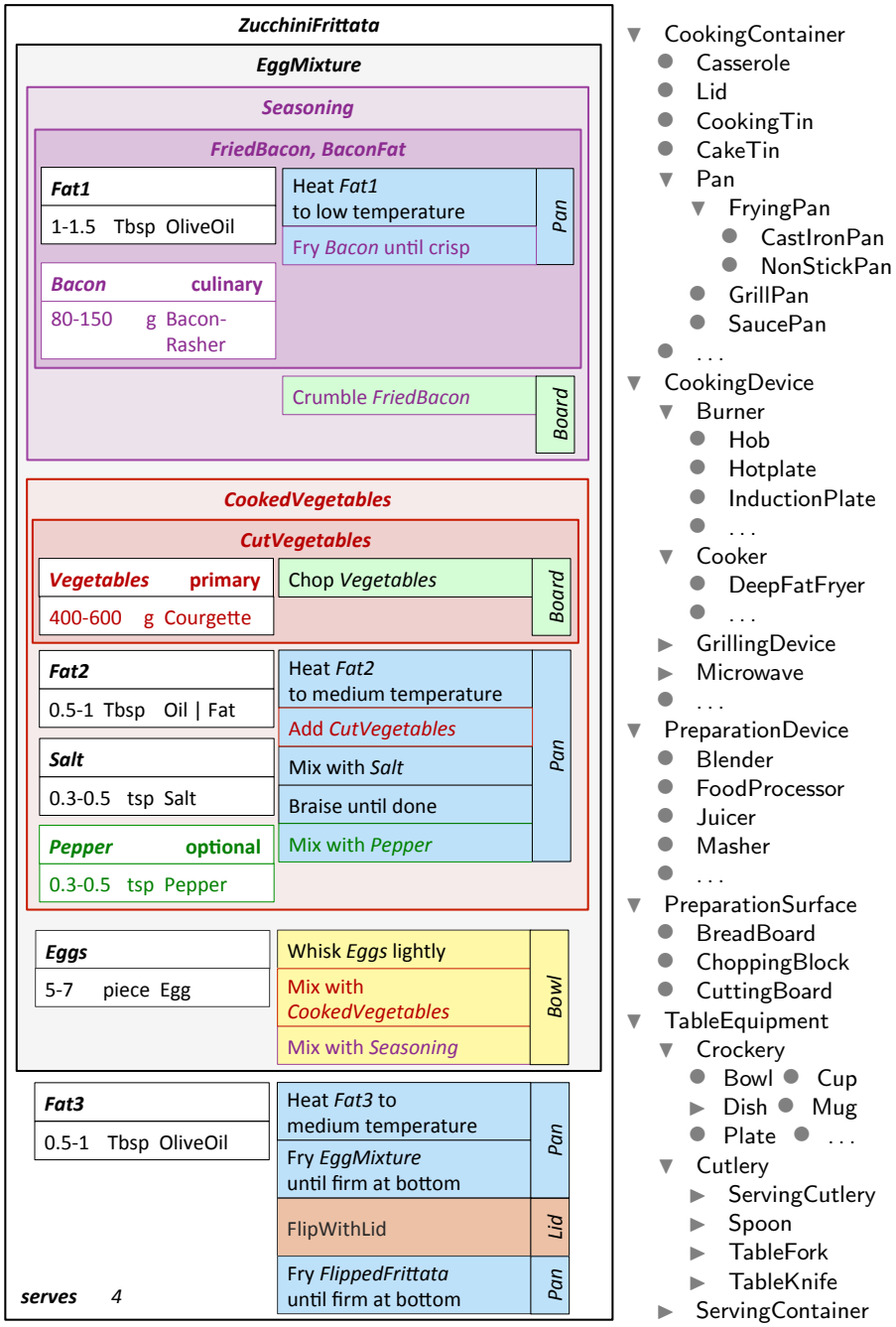

Fig. 4. *Zucchini Frittata* Recipe and CookWare Ontology (Excerpt)

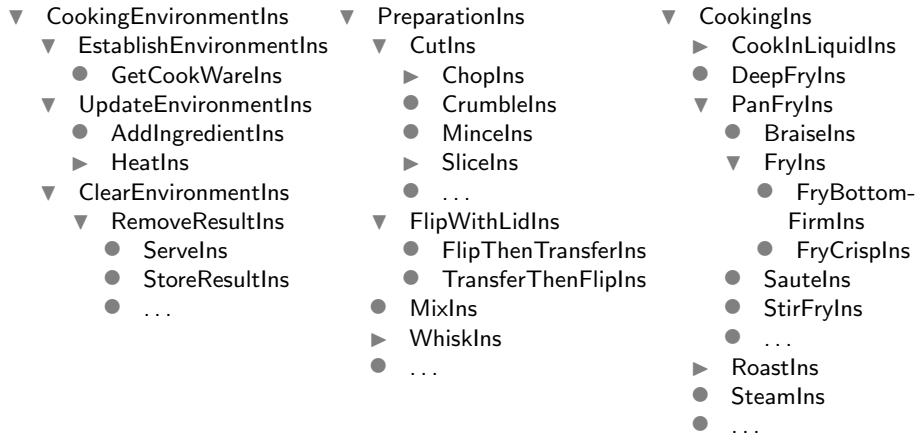

Fig. 5. RecipeInstruction Ontology (Excerpt; "Instruction" abbreviated as "Ins")

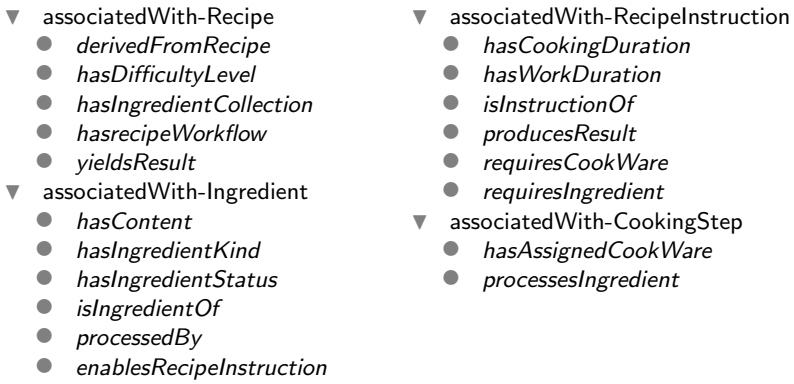

Fig. 6. Recipe Relations in the Ontology (Excerpt)

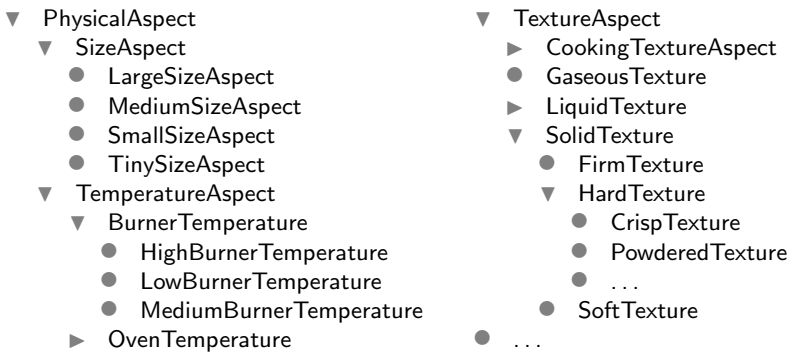

Fig. 7. PhysicalAspect Ontology (Excerpt)

CookWare. The environment contains all CookWare required (see Figs. 4, 5):

PreparationSurface e.g. a CuttingBoard;
PreparationDevice e.g. a FoodProcessor;
CookingDevice e.g. a Cooker with a Burner;
CookingContainer e.g. a Pan to put on the Burner;
CookingUtensil e.g. a KitchenKnife as a FoodCutter, a Whisk;
TableEquipment e.g. a Bowl;

PreparationDevices and CookingDevices have their own power supply and control.

CookWare is modeled as Container since it may contain other CookWare (such as a Pan on a Burner on top of a Cooker), or hold (part of) Ingredients (such as a Bowl, a KitchenKnife, or a CuttingBoard). Note that the required environment of CookWare and Ingredients is modelled with each RecipeInstruction (cf. Sec. 4.3).

BKB hardly uses a FoodProcessor, but cannot do without his ChineseChoppingKnife that doubles as a little plate for small pieces.

Details in a RecipeInstruction. Depending on the abilities of the cook, RecipeInstructions should be more or less detailed. An experienced chef might only need the list of ingredients and apply amounts according to experience, taste and creativity, while a beginner would need to know the exact amount (interval), which tools to use when, etc.

Moreover, RecipeInstructions vary according to the CookWare involved (a RoastInstruction might refine to a GrillInstruction or a RoastInOvenInstruction) or according to the cooking technique used (e.g. a FlipWithLidInstruction to either a TransferThenFlipInstruction or a FlipThenTransferInstruction, cf. Fig. 5), choosing an appropriate one during refinement (cf. Sec. 4.3).

In fact, a RecipeInstruction may be *implementedAs* a Recipe that is more detailed; in particular, it provides an expanded RecipeWorkflow (e.g. for a TransferThenFlipInstruction or FlipThenTransferInstruction). Several implementations may be provided when defining a new RecipeInstruction, giving different amounts of detail for different user profiles (e.g. for a beginner, cf. Sec. 4.3), which may then be used for adapting the interface displayed to the user (cf. Sec. 5.2).

We also expect that modelling a RecipeInstruction explicitly, instead of just having a piece of text, will ease automatic translation of recipes.

Recipe Workflow, Nested Sub-Recipes. A RecipeWorkflow is a sequence of RecipeInstructions, which relate to Ingredients and CookWare, and finally deliver a result that is potentially used as an Ingredient later on. Since a Recipe depends on its Ingredients, and an Ingredient may be the result of another Recipe (e.g. for a seasoning), we are in fact dealing with sub-Recipes inside a Recipe, see Fig. 4. Every sub-Recipe has a name on top referring to its result Ingredient(s), e.g. *FriedBacon, BaconFat*.

A sub-Recipe may be cut out of a Recipe to become an independent, self-contained Recipe, e.g. a Recipe for a seasoning such as *CrumbledBacon*. In the

example recipe, it is purposely left unspecified whether Fat2, i.e. OilOrFat, should contain leftover (flavoring) *BaconFat*; this will only be possible, if it is scheduled to be prepared before CookedVegetables, and is anyway a choice of the cook.

The *relation* Recipe *contains* Recipe is a *partial order*, denoting the dependency of a recipe on another, whose result must be available as an ingredient (cf. also dependent cooking processes in Sec. 5.3). For the *ZucchiniFrittata* to be fried, the *EggMixture* must be ready; for the *EggMixture*, the *CookedVegetables* and the *Seasoning*. The order, in which the *CookedVegetables* and the *Seasoning* have to be prepared, is unspecified (and Mix is commutative); this leaves room for choice in the scheduling of CookingSteps later on.

The overall environment of CookWare involved in a RecipeWorkflow can be deduced from the RecipeInstructions used (cf. Sec. 4.3).

4.2 Generic Recipes, Recipe Development

When trying to find a suitable recipe, the user is faced with an overwhelming number, distributed over many portals, blogs, or web-pages on the internet. Being faced with restricted diets (cf. Sec. 3.1) aggravates the issue.

We hope to eventually provide a uniform (and standardised?) modelling and data base access, not only for information about food (cf. Sec. 2.2), but also recipes. This requires a standard recipe structure and representation (cf. Sec. 4.1) to allow an intelligent search and adaptation in the presence of diet constraints.

We are looking for a way to cluster recipe *variant*s together, encouraging *creativity*. Ms. W. is famous for Apfel-, Topfen- *and* Gemüse-Strudel, cf. Sec. 7.

Variables, Parameter Abstraction. One way to make recipes *generic* (generalised, schematic) is to introduce a kind of *parameter abstraction* (compare CASL generics [12]; not yet available for OWL, cf. Sec. 2.2).

The *primary ingredient* in the *zucchini frittata* (cf. Sec. 4.1), courgette/zucchini, is more generally a vegetable, as seasonally available; but is it really? Can we generalise from CourgetteVegetable to Vegetable, i.e. just navigate upwards in the class hierarchy? No, not just any vegetable, e.g. no cabbage, but perhaps Cauliflower[3]. One proper *culinary* abstraction would be SquashVegetable, serving like a *variable* that can later be substituted by any product in a sub*class*.

To further generalize, an ingredient can be defined as a set of *alternative*s as if an implicit super-*class* was created (cf. Oil | Fat for OilOrFat), e.g.
CourgetteVegetable | FennelVegetable | SpinachVegetable | RadicchioVegetable
for the classic *frittata alla verdura*; even more generally,
SquashVegetable | FlowerVegetable | StalkVegetable | PotatoVegetable | SpinachVegetable | RadicchioVegetable
and so on. StalkVegetable includes fennel; the latter two are special sub*class*es of LeafVegetable, which we want to avoid as it includes CabbageVegetable as well.

This abstraction, allowing seasonal variants and substitutions (cf. Sec. 4.1), and ample room for creativity (cf. Sec. 3.3) with a corresponding abstraction of

[3] *cauliflower*, the German *Blumenkohl*, is actually not a cabbage, but a FlowerVegetable

culinary seasoning, includes some of Ms. W.'s favourites: the Sicilian *frittata di cavolfiore* (CauliflowerVegetable, anchovies and capers) and Umbrian *frittata ai tartufi neri* (PotatoVegetable, black truffles), a sister of the *Spanish omelette*.

4.3 Refinement

Stakeholders, Refinement Stages. When the user of a generic recipe deletes options or provides substitutions for individual adaptation (cf. Sec. 4.1), chooses among alternatives, or navigates down to a particular sub*class*, in fact when being *creative*, s/he becomes an editor of a derived recipe *variant* that is a *refinement* of the original one. Refinement for adaptation will happen in *stages* at various occasions, and the editors will be different *stakeholders* (or assume such roles) with different interests and, more importantly, different *profiles*:

basic author	providing general generic recipes
culinary author	creating recipes with individual culinary kicks
host	gathering and joining the guests' requirements
meal planner	planning recipes for courses and beverages
recipe planner	adapting recipes to the joint guests' requirements
shopper	adapting recipes to (seasonally) available ingredients
kitchen planner	adapting recipes to CookWare available in the kitchen
scheduler	scheduling cooks and RecipeWorkflows
cook	adapting recipes to personal cooking abilities and preferences

Ms. W., as all experienced cooks, will assume all these roles at some time, and change between them. In particular, she prefers to do the shopping herself; she might want to change her mind about a recipe, since today's offer of a fresh seasonal vegetable is so attractive. However, when planning recipes with a derived shopping list for another person as shopper, she will have to be careful to be precise about generalizations and appropriate alternatives for ingredients, keeping the personal shopping profile of the shopper in mind (who might be inclined to choose what he likes, not necessarily in line with her wishes).

Recipe Design. The author of a recipe will be assisted by a special version of a recipe editor (cf. Sec. 5.1), allowing navigation in the *class* hierarchy.

Ms. W. will start with the RecipeInstruction for the Ingredient in focus (cf. Fig. 4). When choosing Bacon as an ingredient, a FryInstruction will be suggested (modelled via *enablesRecipeInstruction*), and Ms. W. will choose the FryCrispInstruction as a refinement. The FryCrispInstruction will be related to the CrispTextureAspect, and, as a FryInstruction, require medium hot OilOrFat in a Pan; this, in turn, will suggest a Burner with a MediumBurnerTemperature, and so on. RecipeInstructions and Ingredients are modelled with corresponding specialised attributes, enabling the Recipe Assistant to suggest appropriate choices.

Similarly, the other stakeholders will be able to navigate in the (generalised) hierarchy of attributes in their refinement process; not only Ingredients, but also RecipeInstructions and CookWare are generalised.

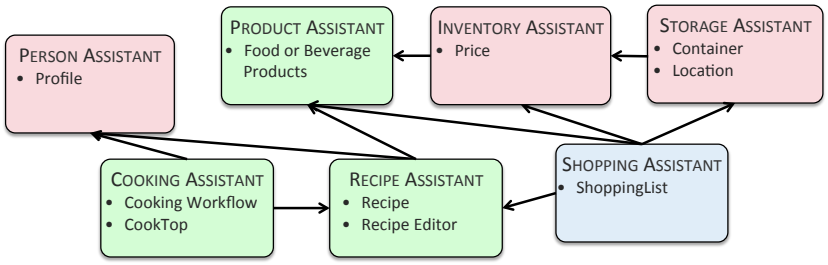

Fig. 8. Software Assistants and Use Relationship

The Recipe Assistant, as a "kitchen planner" prior to the actual cooking, will advise Ms. W. to use a NonStickPan for a Pan, since it knows, which CookWare is preferred and available from her profile as a cook and the profile of the kitchen environment. BKB will get his beloved ChineseChoppingKnife.

Version and Change Management. It is important to record the whole development, a sequence of *refinement*s, for future reference. Thus a new version is placed among a cluster of *variant*s, sharing similar culinary properties.

Ms. W. may wish to revise previous decisions when re-using a recipe next time, omitting a particular dietary constraint, or cooking in a different kitchen. Her recipe variants are kept in a local, private repository.

5 Cooking Assistance

5.1 Software Assistants

Based on the modelling, several modular software assistants are presently under development to help Ms. W. in her tasks, cf. Fig. 8. The Person Assistant manages profiles of stakeholders (cf. Sec. 4.3); the Recipe Assistant helps in the development of recipes, using the Product Assistant and Person Assistant; it generates a shopping list for the Shopping Assistant, which, in turn, uses the Product Assistant for information about food products, the Inventory Assistant about their availability at home or in a shop, and the Storage Assistant about their location.

When Ms. W. goes shopping and changes her mind about a recipe, the Shopping Assistant will be able to trace back to the recipe, Ms. W. can adapt or change it, the shopping list is adjusted accordingly, and the Inventory Assistant bears the availability of food products at home in mind; the Product Assistant will help her choose alternatives or substitutions.

Consistency of Data Updates. The assistants (cf. Fig. 8) correspond to software modules linked to a central controller, which takes care of communication,

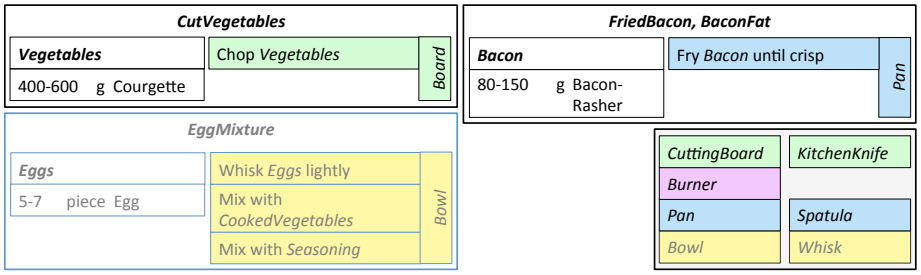

Fig. 9. CookTop View

e.g. interface modules responsible for user interaction, or utility modules for data base access. Assistant modules access data in the ontology (or associated data bases) via the controller; the controller, triggered by an interface module request, distributes the request to appropriate assistants, and forwards answers back to the interface module. The ontology is managed by the SHIP-Tool [13, 15], which, apart from deductions with a standard reasoner, guarantees *consistency of data updates* generated by the processing, a unique feature. The ontology hides and abstracts from associated data bases. Since all the knowledge is represented in the ontology, the assistants only need a minimal data representation internally.

5.2 The CookTop, the Cooking Desktop

The CookTop is the touch-screen via which the cooking assistance processes communicate with the user, see Fig. 9. "Active" (sub)Recipes and CookingSteps, currently being processed, are displayed together with the required Ingredients and CookWare; already completed ones are not displayed any more, neither are those that are not enabled yet, for instance, because the required Ingredients are not yet ready as the result of other processes, or the CookWare is still in use. Once a CookingStep has been completed, the user touches the CookingStep box (or by clicking, voice interaction, etc.). This acknowledgement is recorded by the assistance processes and other possible CookingSteps become enabled. Enabled Recipes (and subsequent CookingSteps in a list) are displayed as gray; they are activated by a user's touch. Depending on the user's abilities and preferences, more or less information (e.g. associated CookWare or durations) is displayed.

5.3 Cooking Workflows, Processes

The RecipeWorkflow of a structured Recipe corresponds to a (partially ordered) tree of sequences of RecipeInstructions for the (sub)Recipes, cf. Sec. 4.1 and Fig. 4. The resulting CookingWorkflows prescribing the order of processing the RecipeInstructions, may be completely sequential, e.g. for an inexperienced cook, to do all preparation work first, and then cook strictly sequentially. However, there is

a potential for parallel work by one cook (or more than one); the scheduling has to take different abilities and resulting prospective durations of workloads, preliminary preparation, actual cooking, settling and cooling phases, into account.

Process Abstractions. The assistance processes control the execution of the CookingWorkflow and can be described as *process*es in the SHIP-Tool at a high level of abstraction. The SHIP-Tool is based on a logical state representation modelling data as well as the state of the real world. States are modelled in Description Logics, which provides the semantic foundation for OWL used to model the recipes. A state consists of the defined *class*es and *relation*s, and *instance*s (individuals) modelling the state. Considering our running example (cf. Fig. 4): each Ingredient, CookWare, etc. is modelled as an *instance* of the respective *class* with relations to other *instance*s as imposed by the class declarations and (constraint) definitions. In SHIP notation, this is expressed as follows

courgette: CourgetteVegetable, (courgette, cquant): hasIngredientQuantity

which represents that the *instance* `courgette` belongs to the class CourgetteVegetable, and `courgette` *hasIngredientQuantity* `cquant`. If relations are *functional* relations, then `courgette`.*hasIngredientQuantity* denotes the associated *instance*.

As modelling discipline we impose that all existential quantifiers have a witnessing *instance* in the ontology. For instance, CookingSteps always have at least one assigned CookWare, which is expressed by the subclass declaration CookingStep $\sqsubseteq \exists$hasAssignedCookware . CookWare; if fry1 is a CookingStep, this imposes that there exists an *instance* in the ontology which is the assigned CookWare (cf. [13]). Available CookWare, the Recipe and instances of the specific Ingredients and quantities (fitting a specific number of persons) are modelled this way.

The assistance processes need to track the status of CookingSteps. To this end, we model the CookingStepStatus as StartedStatus or CompletedStatus. Ingredients necessary for the different CookingSteps and resulting from other CookingSteps create the dependencies between the CookingSteps, cf. Sec. 4.1; active CookingSteps depend on the availability of CookWare; all this information is encoded in an active CookingStep.

Based on the ontological state model, basic computation steps in SHIP are ontology updates which result in a new ontological state. The updates are restricted to *instance*s, the definitions of *class*es and *relation*s cannot be changed. Updates may result from the real world, such as, for instance, the user acknowledging that a specific CookingStep is completed. But updates can also be computing actions of the assistance processes, for instance to enable or initialise a new CookingStep, or to delete a completed cooking step. In SHIP, *action*s can be defined, which have ontological *precondition*s, checked on the current ontological state, and *effect*s describing the update. Based on the actions as basic steps, named, recursive, parallel processes can be defined, used to describe the cooking assistance processes.

Consider the FryBacon step in Fig. 4. The corresponding cooking step assistance process is described in Fig. 10. While the process is presently written manually, we aim at automatic generation from the RecipeWorkflow, cf. Sect. 5.3.

```
1  process fryBacon (fat,bacon) = {
2    init F(fat.currentIngredientStatus:PreparedStatus and
           bacon.currentIngredientStatus:PreparedStatus);
3    fix pan:CookingContainer and pan:UnassignedCookWare and burner:Burner and
               burner:UnassignedCookWare;
4    createHeatupActivity(fat,pan,burner,:lowBurnerTemperature);
5    let d = fat.inv(rawIngredient)
6        prod = d.producedIngredient in
7     init F((fat,pan):at and (fat,:lowBurnerTemperature):hasTemperature and
8       (pan,:lowBurnerTemperature):hasTemperature and
9       prod.currentIngredientStatus:PreparedStatus);
10    closeActivity(d);
11    createFryActivity(bacon,pan,burner,:lowBurnerTemperature,:crisp);
12    let f = bacon.inv(rawIngredient)
13        crispbacon = f.producedIngredient in
14     init F((crispbacon,pan):containedIn and crispbacon.currentIngredientStatus:
            PreparedStatus);
15      closeActivity(f);
16      createRemovalActivity(crispbacon,pan);
17      let doRemove = crispbacon.inv(rawIngredient)
18          storage = doRemove.requiredCookWare in
19       init F((crispbacon,storage):containedIn);
20       closeActivity(doremove) }
```

Fig. 10. Assistance Process for FryBacon

The assistance process is parameterized over the specific ingredients `fat` and `bacon` of the cooking step. It then first waits until these are available, i.e. have PreparedStatus. To this end the SHIP language allows to specify linear temporal logic formulas over ontology expressions, which are monitored over the evolution of the ontological state. We use the standard temporal connectives[4] that allow to start a monitor (line 2) waiting for an ontological state, where both ingredients have PreparedStatus. Once this holds, the process execution continues and we query the current ontological state for unassigned `pan` and `burner` (line 3) and execute the action initialising the first subactivity, i.e. heating up the `fat` in the `pan` on the `burner` (line 4).

Now the information is in the ontology and can be presented to the user on the CookTop interface. We collect the *instance* `d` encoding the activity, but querying the ontological state for the *instance*, of which `fat` is the `rawIngredient` (line 5), as well as the *instance* `prod` introduced to denote the product of the heatup step. Next we wait until the `fat` is in the `pan` and has the right temperature, which the user or some sensing device has indicated, and the product is prepared. The activity is now closed by the action `closeActivity`, which removes the *instance* `d` from the ontological state. Subsequently the next subactivity is started, which

[4] F = Eventually (Future), G = Globally, U = Until

```
1  monitor controlCooking () =
2    G(all s:CookingStep . ((s,r):fromRecipeInstruction and r:CookingInstruction and
3       (s,p):requiresCookWare and p:CookingDevice and
4       (p,ct):currentCookingTemperature and (p,rt):hasCookingTemperature)
5       => (ct =rt U s.yieldsResult.currentIngredientStatus:PreparedStatus))
6
7  process monitorCooking () = {
8    try { init controlCooking }
9    catch {
10     forall s:CookingStep and (s,r):fromRecipeInstruction and r:CookingInstruction and
11       (r,p):requiresCookWare and p:CookingDevice and
12       (p,ct):currentCookingTemperature and (p,rt):hasCookingTemperature and
13       r.yieldsResult.currentIngredientStatus:UnpreparedStatus and
14       ct != rt => if (ct < rt) signalHeatUp(s,p)
15                   else signalCoolDown(s,p);
16                   init F(ct =rt ⟪|⟫ not(!s))
17   }; monitorCooking}
```

Fig. 11. Monitor and Monitor Process

consists of actually frying the `bacon` until it is `crisp` and finally the subactivity to remove it from the `pan`. Again, these subactivities follow the same patterns of (i) initializing the sub-activity possibly preceding a monitor waiting for the availability of Ingredients and CookWare, (ii) a monitor waiting for the user or a sensor in the real world to acknowledge completion of the subactivity, and (iii) closing the subactivity.

For each CookingStep of the Recipe we have respective actions and assistance processes, i.e., `cutVegetables`, `fryBacon` and `eggMixture`. The dependencies between these are managed by the Ingredients and CookWare when they have been produced or become available. Hence the overall assistance process is the parallel composition of these three processes

```
cutVegetables(courgettes) ⟪|⟫ fryBacon(fat,bacon) ⟪|⟫ eggMixture(eggs)
```

The parallel composition is an interleaving of the basic actions of the different processes, as they all operate over the same ontological state.

Monitoring Processes. The SHIP-Tool provides the possibility to define monitors tracking ontological state evolutions, to be used alongside processes to observe the environment and react accordingly. An update violating a running monitor causes a failure in the process semantics, which can be caught like an exception, and processes can be defined to react. Furthermore, it is possible to specify general properties not tied to a specific process, but rather global invariants (in fact, the "common sense of cooking").

As an example consider the monitor controlCooking in Fig. 11. It specifies that in each state, whenever there is an active CookingStep `s` derived from a RecipeInstruction `r` that is a CookingInstruction (in particular, a FryInstruction), then the

required CookingDevice p keeps the required temperature (its current temperature ct is equal to the required temperature rt associated with the CookingDevice in the CookingInstruction) until the resulting Ingredient is prepared.

This monitor can be used in a monitorCooking process, running in parallel to all other assistance processes, that monitors the invariant, signals the respective action to take in case of a violation (heatUp or coolDown) to the user, and, once the invariant is restored, recurses and resumes monitoring.

6 Conclusion

Status of the Modelling and Implementation. Structuring and modelling an intricately interwoven domain such as *Cooking* is indeed a formidable task. Presently, we do not aim for completeness, but for a very substantial coverage that allows the demonstration of nontrivial examples. As the ontology is going to be published in the public domain, we hope for community contributions.

At the same time, we plan to cooperate with other groups. The proper modelling of nutrition impairing diseases or nutrition intolerances (allergies, incompatibilities), cf. Sec. 3.1, requires medical expertise and will be a challenge in itself (see also [31, 10, 11]); we have only made a first attempt so far.

Supporting the CookTop and the actual cooking process by intelligent tools and an intelligent monitoring environment is another direction, where we want to bring in our expertise connected with DFKI's Bremen Ambient Assisted Living Lab, BAALL, and SHIP [14], and combine it with that of the sister Lab at DFKI Saarbrücken, focussed on smart kitchen objects and appliances.

Several Master's and Diploma's theses [20, 23, 24, 30] are under way to complete the modelling, the deduction apparatus, and to develop prototype implementations for the corresponding assistants, to be available as web-apps online.

Cooking with Robots. While the instruction of an experienced cook should be quite terse, a beginner, or an elderly person with slight dementia, needs detailed instruction and detailed sequencing, see Sec. 4.1. It is interesting to note that a cooking robot needs a very similar, if not the same, level of detail to model cooking. We expect to share and combine our modelling with that for robots, e.g. those at Michael Beetz's lab at Universität Bremen.

7 Dedication to Martin Wirsing's Health and Well-Being

How can *Formal Modelling for Cooking Assistance* contribute to Martin Wirsing's health and well-being?

The modelling and methodology described above cite many notions and concepts that have been in the focus of Martin's research on *Formal Methods*: loose (under)specification, abstraction and refinement, processes, temporal logic, etc. He has also always appreciated interesting application domains; now Formal Methods and Cooking come together!

It is, no doubt, primarily his wife Sabine's, i.e. *Ms. W.*'s, *excellent cooking* that is responsible for Martin's good health and well-being. We, as friends, have had the pleasure of sampling it in jolly company; definitely a source of well-being for us, presumably also for Martin, and hopefully for Sabine as well.[5] However, we are getting older and have all sorts of health and other constraints[6] of what we can or wish to eat — so how is Sabine going to cook when a group of us is being invited together?[7]

We hope that Sabine, and others, will eventually get some assistance from the CookTop based on the modelling — and that Martin's good health and well-being will last for many more years to come!

References

1. Barcoo. www.barcoo.com.
2. DOLCE - Descriptive Ontology for Linguistic and Cognitive Engineering. www.loa.istc.cnr.it/old/DOLCE.html.
3. DUL - DOLCE+DnS Ultralite ontology - Ontology Design Patterns (ODP). www.ontologydesignpatterns.org/ont/dul/.
4. EuroFIR AISBL. www.eurofir.org.
5. LanguaL — the International Framework for Food Description. www.langual.org.
6. OWL Web Ontology Language - Use Cases and Requirements - W3C Recommendation 10 February 2004. www.w3.org/TR/2004/REC-webont-req-20040210/.
7. QUDT - Quantities, Units, Dimensions and Data Types Ontologies. www.qudt.org/.
8. WikiFood – Knowing what's inside. www.wikifood.eu/wikifood/struts/welcome.do.
9. Y.-Y. Ahn, S. E. Ahnert, J. P. Bagrow, and A.-L. Barabási. Flavor network and the principles of food pairing. *Scientific reports*, 1:196, Jan. 2011.
10. A. Arens, S. Schnadt, F. Feidert, R. Mösges, N. Roesch, and R. Herbst. Preferences and satisfaction of food allergy sufferers using internet resources. *Clinical and Translational Allergy*, 3:3:126, 2013.
11. A. Arens-Volland, N. Roesch, F. Feidert, P. Harpes, and R. Mösges. Change frequency of ingredient descriptions and free-of labels of food items concern food allergy sufferers. *Allergy (European Journal of Allergy and Clinical Immunology)*, 65:92, 2010.
12. E. Astesiano, M. Bidoit, B. Krieg-Brückner, H. Kirchner, P. D. Mosses, D. Sannella, and A. Tarlecki. CASL - the Common Algebraic Specification Language. *Theoretical Computer Science*, 286:153–196, 2002.
13. S. Autexier and D. Hutter. Constructive DL update and reasoning for modeling and executing the orchestration of heterogenous processes. In T. Eiter, B. Glimm, Y. Kazakov, and M. Krötzsch, editors, *Informal Proceedings of the 26th International Workshop on Description Logics*, volume 1014, pages 501–512, Ulm, Germany, July 2013. Technical University of Aachen (RWTH).

[5] With the generic vegetable omelette abstraction of Sec. 4.2, will Sabine get new ideas for her famous vegetable strudel, so much appreciated by their friends?

[6] Martin, Sabine and family of course excluded

[7] Many of us friends can appreciate such problems as enthusiastic amateur cooks

14. S. Autexier, D. Hutter, C. Mandel, and C. Stahl. SHIP-Tool Live: Orchestrating the Activities in the Bremen Ambient Assisted Living Lab (Demo). In J. C. Augusto and R. Wichert, editors, *Fourth International Joint Conference on Ambient Intelligence*, LNCS, Dublin, Ireland, December 2013. Springer.
15. S. Autexier, D. Hutter, and C. Stahl. An Implementation, Execution and Simulation Platform for Processes in Heterogeneous Smart Environments. In J. C. Augusto and R. Wichert, editors, *Fourth International Joint Conference on Ambient Intelligence*, LNCS, Dublin, Ireland, December 2013. Springer.
16. J. A. Bateman, A. Castro, I. Normann, O. Pera, L. Garcia, and J.-M. Villaveces. OASIS Common hyper-ontological framework (COF). EU FP7 Project OASIS – Open architecture for Accessible Services Integration and Standardization Deliverable D1.2.1, Bremen University, Bremen, Germany, January 2010.
17. W. Becker. Towards a CEN Standard on Food Data. *European journal of clinical nutrition*, 64:S49–S52, 2010.
18. M. Burgos, I. Martínez-Victoria, R. Milá, A. Farrán, R. Farré, G. Ros, M. Yago, N. Audi, C. Santana, L. Millán, et al. Building a unified Spanish food database according to EuroFIR specifications. *Food Chemistry*, 113(3):784–788, 2009.
19. R. Caviezel and T. A. Vilgis. *Foodpairing — Harmonie und Kontrast*. FONA, 2012.
20. S. Ghomsi Nokam. A Food Ontology for the Assistance of Shopping and Cooking. Master's thesis, Universität Bremen, in preparation. (in German).
21. J. Ireland and A. Møller. What's new in LanguaL? *Procedia Food Science*, 2:117–121, 2013.
22. J. D. Ireland and A. Møller. Review of international food classification and description. *Journal of food composition and analysis*, 13(4):529–538, 2000.
23. P. Kolloge. Modelling Dietary Restrictions. Master's thesis, Universität Bremen, in preparation. (in German).
24. D. Kozha. Shopping Assistance from the Kitchen Cabinet to the Supermarket Shelf. Master's thesis, Universität Bremen, in preparation. (in German).
25. O. Kutz, T. Mossakowski, and D. Lücke. Carnap, Goguen, and the Hyperontologies: Logical Pluralism and Heterogeneous Structuring in Ontology Design. *Logica Universalis*, 4(2):255–333, 2010. Special Issue on 'Is Logic Universal?'.
26. K. Lo. *Chinese Cooking and Eating for Health*. Mayflower Granada Publ., 1979.
27. A. Møller, I. Unwin, W. Becker, and J. Ireland. EuroFIR's food databank systems for nutrients and bioactives. *Trends in food science & technology*, 18(8):428–433, 2007.
28. T. Mossakowski, O. Kutz, M. Codescu, and C. Lange. The distributed ontology, modeling and specification language. In C. D. Vescovo, T. Hahmann, D. Pearce, and D. Walther, editors, *WoMo 2013*, volume 1081 of *CEUR-WS online proceedings*, 2013.
29. J. A. Pennington and R. R. Butrum. Food descriptions using taxonomy and the LanguaL system. *Trends in Food Science & Technology*, 2:285–288, 1991.
30. M. Rink. Ontology Based Product Configuration Based on User Requirements. Master's thesis, Universität Bremen, in preparation.
31. N. Roesch, A. Arens, F. Feidert, R. Herbst, and R. Mösges. Computerised identification of allergens in food ingredient descriptions. *Allergy: European Journal of Allergy and Clinical Immunology*, 64:363–364, 2009.
32. C. Snae and M. Bruckner. Foods: a food-oriented ontology-driven system. In *Digital Ecosystems and Technologies, 2008. DEST 2008. 2nd IEEE International Conference on*, pages 168–176. IEEE, 2008.