A Tutorial on Modular Ontology Modeling with Ontology Design Patterns: The Cooking Recipes Ontology

Pascal Hitzler, Wright State University, USA, pascal@pascal-hitzler.de
Adila Krisnadhi, Universitas Indonesia, krisnadhi@gmail.com

August 2018

We provide a detailed example for modular ontology modeling based on ontology design patterns. It is similar to the Chess Ontology tutorial in [6], which we suggest to read first. We will be less verbose in this tutorial; we provide it because additional examples should be helpful for those interested in adopting the modular ontology modeling methodology – see [6] and the book [2] in which it is contained.

We assume that the reader is familiar with the Web Ontology Language OWL [4, 5].

Before we dive into the actual modeling, let us present the general workflow which we recommend for ontology modeling, and which is the same as in [6]. The steps of this workflow are laid out in Figure 1. We will refer to these steps, and explain them in more detail, as we advance through the tutorial.

1 Use Case

Step 1: Define use case or scope of use cases.

Every ontology is designed for a purpose; this purpose may be defined by a use case, or by a set of use cases, or possibly by a set of potential use cases, which may include the future extensions or refinements of the ontology, and future reuse of the ontology by others.

How specific should a use case be? Conventional wisdom may suggest that it is always better to be more specific. However, in the context of ontology modeling the case is not as clear-cut. A very specific use case may give rise to an ontology which is very specialized, i.e. modeling choices (so-called *ontological commitments*) may be made which fit only the very specific and detailed use case. As a consequence, later modifications, e.g. by widening the scope of the application (and therefore of the underlying ontology) become very cumbersome as they may conflict with ontological commitments made earlier.

Let us look at a very simple example of this. Say, our application involves movies and actors from the casts of these movies. It may first seem as if actor

1. Define use case or scope of use cases.
2. Make competency questions while looking at possible data sources and scoping the problem, i.e., decide on what should be modeled now, and what should be left for a possible later extension.
3. Identify key notions from the data and the use case and identify which pattern should be used for each. Many can remain "stubs" if detailed modeling is not yet necessary. Instantiate these key notions from the pattern templates, and adapt/change the result as needed. Add axioms for each module, informed by the pattern axioms. As a result of this step, we arrive at a set of modules for the final ontology.
4. Put the modules together and add axioms which involve several modules. Reflect on all class, property and individual names and possibly improve them. Also check module axioms whether they are still appropriate after putting all modules together.
5. Create OWL files.

Figure 1: Ontology modeling workflow followed.

names could simply be attached to the movies using an OWL datatype property hasActor. E.g., this could be written in RDF Turtle as

```
:myMovie     :hasActor    "JaneSmith"  .
```

This will be sufficient, e.g., if only the name of an actor is relevant for an application.

However, it is conceivable that the application (and thus the ontology) may later on be extended in order to be able to list all movies in which a given actor was a cast member. Since it is likely that there may be different persons with the same name, such as Jane Smith, we would need to be able to identify which name strings identify the same, and which identify different actors, i.e., we have to disambiguate the name strings. Furthermore, Jane Smith may also be listed as actor under a different name, say Jane W. Smith.

Using an OWL datatype property as above, however, was a modeling choice which prevents this. What we would need is URIs for actors. With this case our example would look like the following.

```
:myMovie       :hasActor   :janeSmith1 .
:janeSmith1    :hasName    "JaneSmith" .
```

Further extensions of the application (or attempted reuses of the ontology), however, may pose yet additional problems. E.g., it may be desired to also list the character played by an actor in a specific movie. Given our current modeling choices, however, it seems unclear where to attach this information: If it is

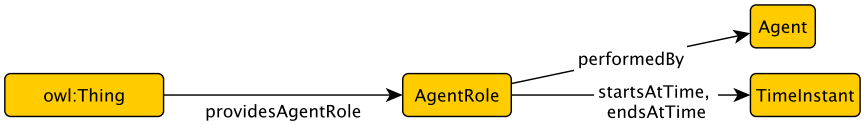

Figure 2: Generic AgentRole pattern

attached to the movie, then we would no longer be able to say which character in the movie was played by which actor. If we attach the information to the person, then we would no longer know which movie the character appeared in. If we attach it to both movie and person, then we would run into difficulties if characters appear in different movies, played by different actors.

The solution in this case is to create another node in the graph, which stands for the *actor role*. Our example would then look as thus.

```
:myMovie          :hasActor    :myMovieMissXRole .
:myMovieMissXRole :assumedBy   :janeSmith1 ;
                  :asCharacter :MissX .
:janeSmith1       :hasName     "JaneSmith" .
```

We understand that we can make modeling choices which make future reuse easier, e.g., by making sure that we include enough nodes in the graph. This, of course, begs the question where to stop? If we follow this principle, then won't we end up with much too many nodes, blowing up the graphs?

This is a valid concern, of course, and there are not straightforward solutions for this issue which work in all circumstances. Generally speaking, we should strive for a balance, i.e., finding a soft spot somewhere between the extremes. Our approach using ontology design patterns addresses the issue as we will be able to reuse patterns which have been created and vetted by the community, and which provide a good trade-off between the extremes in many circumstances.

Returning to the movie example, there are two patterns which would be the standard choices in this situation, the AgentRole and the NameStub pattern. Class diagrams for these two are shown in Figures 2 and 3. We can now take the AgentRole pattern and remove the TimeInstant class, and join it with the NameStub pattern by using Agent instead of owl:Thing, see Figure 4. Finally, we use the resulting class diagram as a template and make renamings of class and property names to fit it to our more specific use case; the result is displayed in Figure 5. This process exemplifies our intended use of ontology design patterns: We use them as templates and specialize and join them in order to obtain a draft of our desired model.

After this discussion, let us now return to the actual task at hand, namely to define a use case scenario for our worked example. The setting we have in mind concerns online cooking recepies, and in particular the task of integrating recipes from different websites in order to enable a fine-grained cross-website search for recipes:

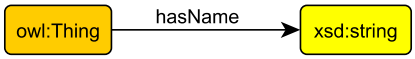

Figure 3: Generic NameStub pattern

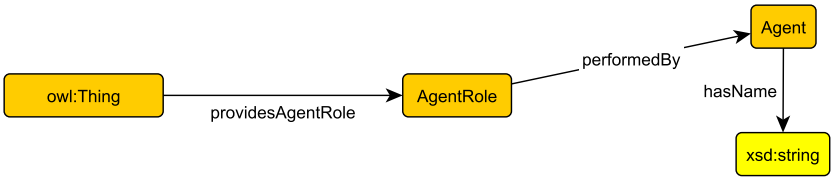

Figure 4: Joining the AgentRole and NameStub patterns

Design an ontology which can be used as part of a "recipe discovery" website. The ontology shall be set up such that content from existing recipe websites can in principle be mapped to it (i.e., the ontology gets populated with data from the recipe websites). On the discovery website, detailed graph-queries (using the ontology) shall produce links to recipes from different recipe websites as results. The ontology should be extendable towards incorporation of additional external data, e.g., nutritional information about ingredients or detailed information about cooking equipment.

Let us make a few remarks about the scenario we have just defined. First of all, we notice that data will come from multiple sources which are not exactly specified. This means that our ontology needs to be general enough to accomodate different conceptual representations on the source side. Second, the ontology shall be extendable towards additional related data, meaning that we have to accomodate such extension capabilities, to a reasonable extent, without knowing what these future extensions would exactly look like, and this again asks for a rather general model. Third, fine-grained search for recipes shall be possible, meaning that our ontology needs to be specific enough to allow these. The scenario thus calls for a reasonable trade-off between specifity and generality, i.e., it is a typical use-case for ontology design pattern based modular ontology modeling.

2 Competency Questions and Data Sources

Step 2: Make competency questions while looking at possible data sources and scoping the problem, i.e., decide on what should be modeled now, and what should be left for a possible later extension.

Competency questions are queries, formulated in natural language, which could potentially used for retrieval of data from the knowledge base. They help

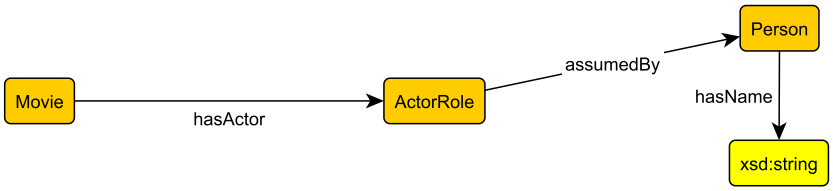

Figure 5: The movie snippet: using Figure 4 as a template

to further specify the use cases, i.e., main classes of potential queries should be represented.

For our scenario, possible competency questions are the following.

1. Gluten-free low-calorie desserts.
2. How do I make a low-carb pot roast?
3. How do I make a Chili without beans?
4. Sweet breakfast under 100 calories.
5. Breakfast dishes which can be prepared quickly with 2 potatoes, an egg, and some flour.
6. How do I prepare Chicken thighs in a slow cooker?
7. A simple recipe with pork shoulder and spring onions.
8. A side prepared using Brussels sprouts, bacon, and chestnuts.

The competency questions already indicate some parameters that will be important, e.g.:

- Retrieval of cooking instructions.
- Search by ingredients.
- Search by properties of the prepared food, e.g. calorie or carb content.
- Search by properties such as cooking time, simplicity.

At this stage, at the latest, it is also necessary to look at possible data sources. A quick Web search provides a significant number of recepie websites, e.g., allrecipes.com, food.com, epicurious.com. Pages commonly list ingredients, cooking instructions, and sometimes other information such as nutritional information. Additional nutritional information is, e.g., available from Google Knowledge Graph nutrition data.[1] Data is usually not available in structured

[1] `https://search.googleblog.com/2013/05/time-to-back-away-from-cookie-jar.html`

form, i.e., for an application it will be necessary to extract content from text-based web pages, which can be a tricky task in itself; however herein we only concern ourselves with producing a suitable underlying ontology.

Looking at the data sources now prompts us to go back to the competency questions, to reevaluate them. Some competency questions may have to be dropped or modified at first, e.g., recipe websites seem to rarely mention equipment, like slow cookers, separately, identify a breakfast as *sweet*, or use classification such as low-carb or gluten-free. We should keep these in mind, though, and make sure that the ontology we produce is extendable towards future inclusion of such aspects. At the same time, inspection of the data may yield further insights regarding data that could now or in the future be included, such as recipe authors, peer recommendations, cooking time, level of difficulty, or category tags such as *dessert* or *side*. These can either be incorporated right away, or alternatively extensibility towards future inclusion can be kept in mind during modeling.

The decision process regarding what to model now versus later is called *scoping*. At the end of this step, we should have arrived at a clear idea concerning the scope of the target ontology.

3 Key Notions to Modules

Step 3: Identify key notions from the data and the use case and identify which pattern should be used for each. Many can remain "stubs" if detailed modeling is not yet necessary. Instantiate these key notions from the pattern templates, and adapt/change the result as needed. Add axioms for each module, informed by the pattern axioms. As a result of this step, we arrive at a set of modules for the final ontology.

Think of the key notions as the main classes of things appearing in the competency questions or which you identify from the data sources. Obvious possible key notions which come to mind are recipe, food, time, equipment, classification of food prepared (e.g., as side), difficulty level, nutritional information. Let's go through these one by one and refine the list while creating corresponding schema diagrams. After that, we will talk about axiomatizing them.

3.1 Class Diagrams

Recipe

Recipe is an obvious candidate for a class, it is central to what we intend to do, and in addition we may want to notice already that the name of the recipe, which is often identical with the food which is going to be prepared, should be recorded. For the latter, we should probably use the NameStub. We now also want to identify a pattern which will be the basis for the core of the recipe modeling.

Of course, we have not yet discussed other patterns than NameStub and AgentRole. One way to approach this is to go through a list of known patterns[2] and to contemplate which may fit best, and sometimes there seem to be more than one candidate. Let us look at three more or less obvious candidates.

Let us first check whether it makes sense to think of recipes as documents. There is certainly a perspective from this this seems valid: in the end, isn't it simply a document which we retrieve from the Web when we download a recipe? However, document seems to be a rather generic notion which does not naturally cater for key aspects of a recipe such as having ingredients, or taking a particular time. We also wouldn't say about a document whether it's low-carb or not. This line of thinking may lead us to the conclusion that a document may *contain* a recipe *description*, but that a recipe as such is a different type of entity.

Since recipes usually contain step-by-step descriptions of the food preparation process, another alternative may be to think of a recipe as a sequence, which is another fundamental ontology design pattern. But then it also seems clear that many of the aspects important for our competency questions are not naturally catered for by the notion of sequence, e.g., what are ingredients in relation to recipe as a sequence? This line of thinking may lead us to the conclusion that some parts of the recipe – the cooking steps – may be representable as a sequence, but the whole of the recipe is much more than that. On the other hand, our competency questions do not indicate that the preparation steps sequence as such is particularly relevant to our task, namely the discovery of recipes.

We could also think of recipes as *processes* which may help us to emphasize input and output aspects. This may indeed be a valuable perspective. However, the notion of process may usually allude to much more rigid and well-defined sequences of actions, so we would have to have a very detailed look at a process ontology design pattern to decide whether it provides the right perspective for our purpose.

The perspective we will actually take here is that a recipe is a type of description. Indeed, the general description pattern [1] has a specialization to plans, and indeed it seems a reasonable perspective to think of recipes as plans (to produce something).

Let us look at a part of the Plan pattern which is depicted in Figure 6. A plan leads from an initial situation to a situation which is understood as the goal of the plan. The initial situation would be one in which required ingredients (and equipment etc) would be available, while in the goal situation the prepared food would be available. In fact, the required ingredients, equipment etc. are necessary for these respective situations, i.e. they are *constituents* for them. Putting these thoughts together, we can arrive at a first piece of the Recipe module, as an instantiation of the Plan template. Its schema diagram is depicted in Figure 7. There is much more to be said about descriptions, plans, situations, etc., but we will not go into detail here. See [1] for a central reference.

[2] `htt://www.ontologydesignpatterns.org` sports many patterns, however they and their documentations are of very differing quality.

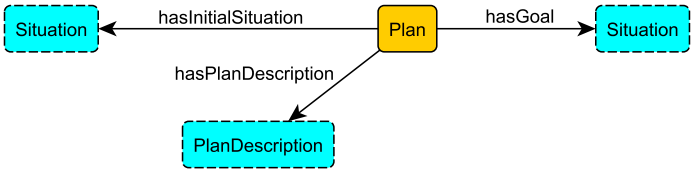

Figure 6: Basic Plan ontology design pattern: schema diagram. The dashed boxes indicate complex notions which would easily merit a pattern description in their own right.

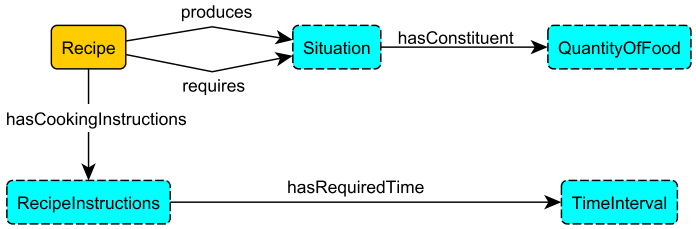

Figure 7: Recipe as plan

Food

Proceeding with our list of potential key notions, the next one is *food*. This seems a little unspecific. What exactly is meant by this? Well, food can be things like cucumbers, potatoes, eggs, lasagna, and Chicken Kiev. Perhaps we should distinguish between ingredients and results, i.e., full dishes? But wait a second, what about, say, Pesto Genovese? It's not a dish by itself, but an ingredient in some recipes; yet there are also recipes how to make Pesto Genovese. Indeed, many cooking ingredients are already processed from even more basic ingredients. So it probably will not make much sense to try to distinguish between ingredients and dishes when talking about recipes.[3]

But, if we say Pesto Genovese, what exactly do we mean? Do we mean Pesto Genovese in general, as such, or do we mean, say, two teaspoons of it, as required by some recipe? Indeed it seems that for recipes the quantity of a required or produced food item is also important.

We're starting to narrow this down. *Quantity* seems like an ontology design pattern which we should make use of for our purposes, and "a quantity of food" seems to be a central concept for modeling recipes as plans, to be used both on the input and on the output side of the recipe as plan.

[3]The type of discussion exemplified in this paragraph is central to coming up with good key notions and modules. It is extremely helpful to have this discussion in a group, as others are often so much better in finding flaws in our ideas than we are ourselves.

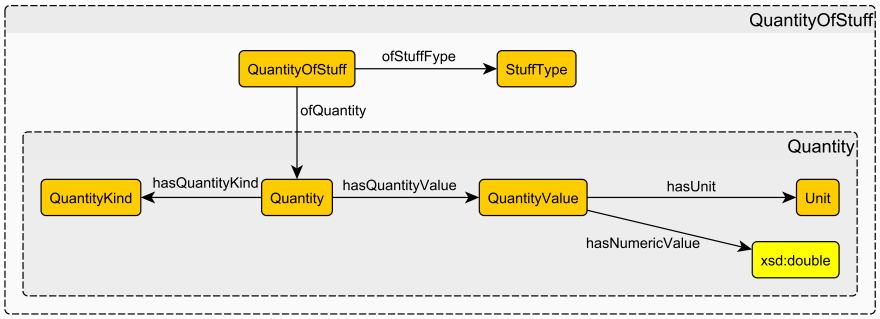

Figure 8: The QuantityOfStuff pattern, the inner box indicates the Quantity pattern. There is much more to be said about quantities, but we will not further dwell on this here

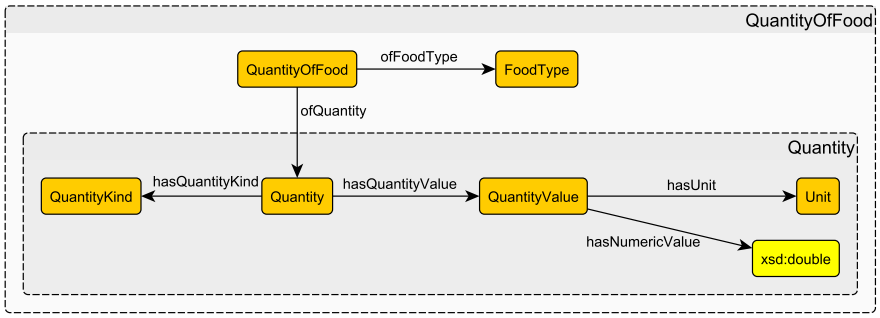

Figure 9: QuantityOfFood module, as an instance of the Quantity pattern from Figure 8.

So we understandt hat there should be a concept of QuantityOfFood (like, 2 tsp of Pesto) which is always of some quantity (like, 2 tsp) and at the same time is of some type of foodstuff (say, Pesto). The foodstuff can thus be understood as a FoodType (like, Pesto, or potato), namely the type of stuff the quantity of food consists of. See Figures 8 and 9 for schema diagrams. Our pattern for Quantity is very much directly derived from QUDT.[4]

Equipment, Classification, Difficulty Level

Next on our list of key notions is *equipment*, such as slow cooker, blender, etc. However, while keeping track of the occasional special equipment may be helpful, our scenario does not call for a detailed modeling of kitchen equipments at this stage. So let us decide to delay such detailed modeling for the moment, i.e., we

[4] `http://qudt.org/`

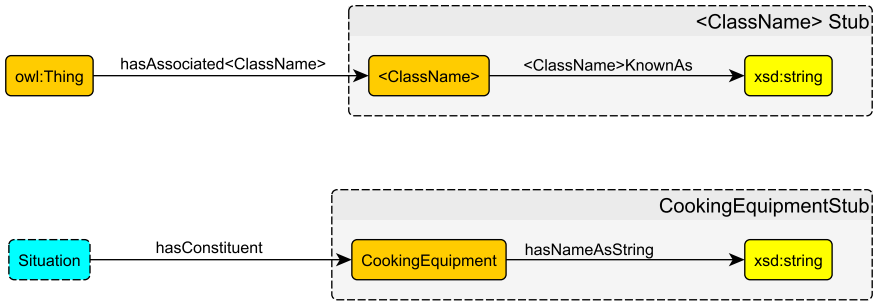

Figure 10: Top, the Stub (meta)pattern. Bottom, its instantiation for equipment.

consciously restrict the scope of our model.

Decisions such as this are very important during the modeling process, as they limit the scope of the ontology. Indeed, it is impossible to always model all details, as we would end up with a model of almost everything in this case. At the same time, however, we would like to keep in mind that our ontology may be reused later and possibly repurposed for a scenario in which detailed modeling of equipment may be more important. This means, that we do not want to simply introduce a datatype property such as `requiresEquipment` with strings – the names of the equipment – as range. We rather want to utilize a slightly more sophisticated approach where we at least have a node as placeholder for the equipment entity.

The corresponding ontology design pattern is called a *Stub* [7], and it is depicted in Figure 10 together with the instantiation for equipment which we will use. It is really essentially the same as the NameStub pattern introduced earlier, the only difference being that the identifying string is not necessarily a name of the thing identified.

Note also that we attach the cooking equipment as constituent to a situation, which seems to be its natural place.

We opt for stubs also for other key notions we have identified, namely for DifficultyLevel and for RecipeClassification (such as low-carb, diabetic, etc.), i.e., for now the ontology will be able to hold only strings for these, but the model remains extendable if so desired in the future. The corresponding schema diagrams can be found in Figure 11.

We will use stubs also in other places, e.g., we have not further talked about FoodType as it appears in Figure 9. As before, it is conceivable that there may be a sophisticated model of different food types, but we will use a stub at this stage.

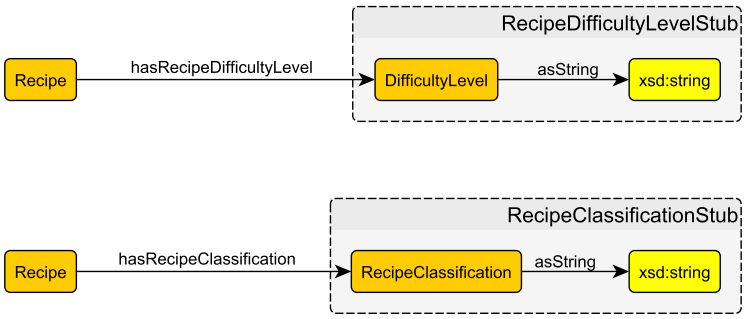

Figure 11: Stubs for DifficultyLevel and RecipeClassification.

Nutritional Information

Next we turn our attention to *nutritional information*, the next keyword on our list, and we opt to model this in somewhat more detail. More precisely, we will model the contents of Nutritional Facts labels as mandated in the U.S.A. for most food products,[5] see Figure 12 While this may seem overly specific, by virtue of our modular modeling approach it would be easy to replace the NutritionalInformation module with one tailored to, e.g., other countries, or other nutritional convictions. In fact, we will highlight this by creating a class `US-Nutrition-Label` as a subclass of the generic NutritionalInformation class.

These Nutritional Facts labels have highly structured content. We will of course not be concerned with layout issues, and it is also not necessary that we model all content. E.g., we will not list "% Daily Value" amounts for fat or sodium. We will list absolute amounts for Fat, Saturated Fat, Trans Fat, Cholesterol, Sodium, Carbs, Dietary Fiber, Sugars, and Protein, and "% Daily Value" amounts for Vitamins A and C, Calcium, and Iron, which we represent as instances of a class `NutritionalContentType`.[6] It seems obvious that we will reuse the *QuantityOfStuff* pattern again, however as a percentage value is not really a quantity, we add an alternative to giving the quantity, which consists simply of a datatype property `isPercentageOfDailyValue` with range `xsd:positiveInteger`. See the right of Figure 13.

Of course we also need to record the serving size to which the nutritional information refers, and this can again be done using the *Quantity* pattern. We also list calorie content and calories-from-fat content as indicated in the figure.

Provenance

We have worked through our list of keywords, but before we move on, let us briefly reflect whether there is anything else that needs modeling, which we

[5] see `https://en.wikipedia.org/wiki/Nutrition_facts_label#United_States`

[6] Essentially, we are creating a small *controlled vocabulary* for substances of nutritional importance.

Nutrition Facts

Serving Size 2/3 cup (55g)
Servings Per Container About 8

Amount Per Serving

Calories 230	Calories from Fat 40
	% Daily Value*
Total Fat 8g	**12%**
Saturated Fat 1g	**5%**
Trans Fat 0g	
Cholesterol 0mg	**0%**
Sodium 160mg	**7%**
Total Carbohydrate 37g	**12%**
Dietary Fiber 4g	**16%**
Sugars 1g	
Protein 3g	
Vitamin A	10%
Vitamin C	8%
Calcium	20%
Iron	45%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily value may be higher or lower depending on your calorie needs.

	Calories:	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Figure 12: Example for a U.S. FDA Nutritional Facts label

can derive from our scenario description. And indeed, our scenario states that queries shall produce links to recipes from different recipe websites – however, or modeling so far did not include anything which would make it possible to track where a recipe came from. We thus need to do some provenance modeling.

The schema diagram of a generic provenance pattern, as derived from PROV-O [9] and mentioned in [13], is provided in Figure 14. The key idea of this is that everything (any `owl:Thing`) for which provenance is important, was generated by some activity (in our case, web retrieval) which used some other thing (in our case, a recipe website). The item under consideration (the recipe) may also be directly related to its origin (the recipe website) using the property `wasDerivedFrom`. In addition, agents may be involved in activities.

For our purpose, it will suffice to reuse a small part of this pattern, namely the `wasDerivedFrom` property. Derivation in this case is from a document which has a URL (i.e., a website), and we can use a Document stub for this. The resulting module is depicted in Figure 15

3.2 Axiomatizations

We have now produced diagrams for all key notions we had identified, and have used schema diagrams of general ontology design patterns to produce them. The list of key notions, together with the used patterns can be found in Figure 16.

We now turn to producing OWL axioms for all modules. We use the earlier

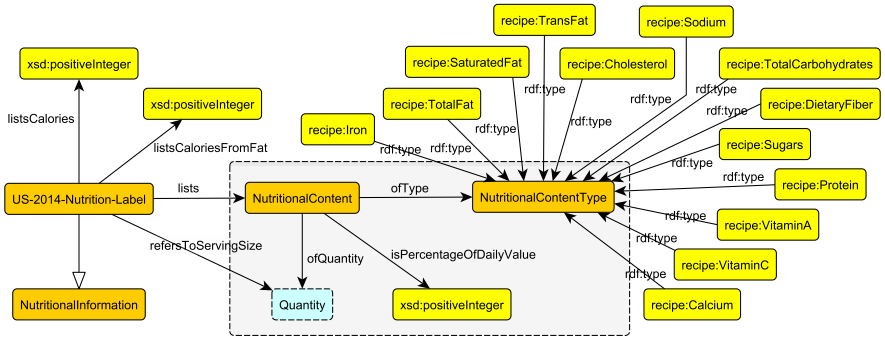

Figure 13: Nutritional Information module. The box indicates a modified instance of the QuantityOfStuff pattern.

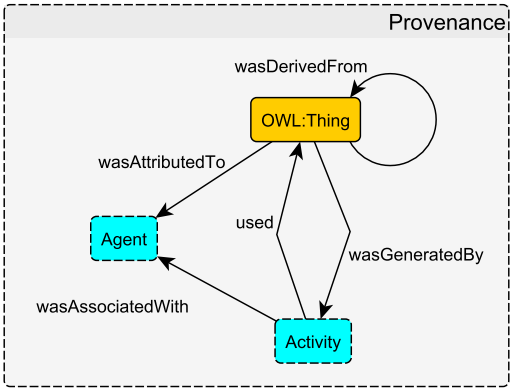

Figure 14: Provenance pattern

schema diagrams as guidance. Usually, axioms would be derived from the axioms provided with the patterns, but we will recreate them from scratch, in order to gain a deeper understanding of them. We will in fact produce a rather exhaustive list of axioms which seem appropriate for our model, while steering away from overly strong ontological commitments.

There is a systematic way to look at each node-edge-node triple in the schema diagram in order to decide which axioms should be added: Given a node-edge-node triple with nodes A and B and edge R from A to B, as depicted in Figure 17, we check all of the following axioms whether they should be included.[7] We list them in natural language, see Figure 18 for the formal versions in description logic notation, and Figure 19 for the same in Manchester syntax, where we also list our names for these axioms.

[7]The OWLAx Protégé plug-in [11] provides a convenient interface for adding these axioms.

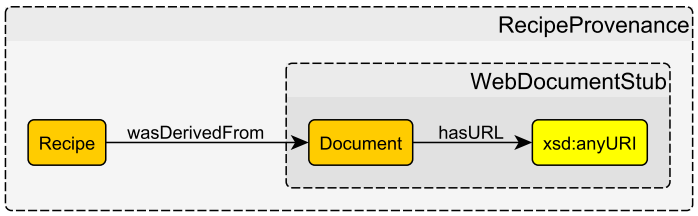

Figure 15: Recipe provenance module

Recipe	Plan
RecipeName	NameStub
RecipeInstructions	Document
TimeInterval	temporal information
QuantityOfFood	QuantityOfStuff
Quantity	Quantity
Equipment	Stub
FoodType	Stub
Difficultylevel	Stub
RecipeClassification	Stub
NutritionalInfo	unspecified pattern using QuantityOfStuff
Source	Provenance

Figure 16: All key notions together with corresponding patterns used

1. A and B are disjoint.
2. The domain of R is A.
3. For every B which has an inverse R-filler, this inverse R-filler is in A. In other words, the domain of R scoped with B is A.
4. The range of R is B.
5. For every A which has an R-filler, this R-filler is in B. In other words, the range of R scoped with A is B.
6. For every A there has to be an R-filler in B.
7. For every B there has to be an inverse R-filler in A.
8. R is functional.
9. R has at most one filler in B.
10. For every A there is at most one R-filler.
11. For every A there is at most one R-filler in B.
12. R is inverse functional.
13. R has at most one inverse filler in A.
14. For every B there is at most one inverse R-filler.
15. For every B there is at most one inverse R-filler in A.

Domain and range axoims are items 2–5 in this list. Items 6 and 7 are

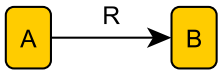

Figure 17: Generic node-edge-node schema diagram for explaining systematic axiomatization

1. $A \sqcap B \sqsubseteq \bot$	6. $A \sqsubseteq R.B$	11. $A \sqsubseteq \leq 1R.B$
2. $\exists R.\top \sqsubseteq A$	7. $B \sqsubseteq R^{-}.A$	12. $\top \sqsubseteq \leq 1R^{-}.\top$
3. $\exists R.B \sqsubseteq A$	8. $\top \sqsubseteq \leq 1R.\top$	13. $\top \sqsubseteq \leq 1R^{-}.A$
4. $\top \sqsubseteq \forall R.B$	9. $\top \sqsubseteq \leq 1R.B$	14. $B \sqsubseteq \leq 1R^{-}.\top$
5. $A \sqsubseteq \forall R.B$	10. $A \sqsubseteq \leq 1R.\top$	15. $B \sqsubseteq \leq 1R^{-}.A$

Figure 18: Most common axioms which could be produced from a single edge R between nodes A and B in a schema diagram: description logic notation.

extistential axioms. Items 8–15 are about variants of functionality and inverse functionality. All axiom types except disjointness and those utilizing inverses also apply to datatype properties.

Recipe as Plan

We now return to our recipe example, more precisely to Figure 7. We will henceforth use Manchester syntax only; the description logic variants can be found in the appendix. The axioms can be found in Figure 20 Note that, generally speaking, we prefer scoped versions of axioms, as they represent the weaker axioms from the perspective of formal semantics.

Items 1 and 2 refer to the `requires` edge and adjacent nodes in Figure 7. Item 1 is a scoped range restriction; note that we do not specify a scoped domain, because we feel that a statement which says that quantities of food can only be ingredients in recipes (and in nothing else) may seem too restrictive. For a similar reason, we specify only one of the standard existential axioms, in Item 2. Items 3 and 4 are analogous, for the `produces` edge and adjacent nodes. Items 5–9 refer to the `hasCookingInstructions` edge and adjacent nodes; in this case we have scoped domain and scoped range expressions, both existentials, and a cardinality expression. The cardinality expression 9 states that every entity in the class RecipeInstructions can be associated as cooking instructions to at most one recipe. Item 10 is a scoped range expression for the `hasRequiredTime` edge and adjacent nodes; note that none of the other axioms seems fully appropriate in this case, e.g., other things can have required times as well. Items 11 and 12 are additional axioms which involve two properties and thus do not come from the list in Figure 19. They state that each recipe requires some `QuantityOfFood` to begin with, and also always produces some `QuantityOfFood`.

In short, we include the following axioms. For `requires`: scoped range, exis-

1. A DisjointWith B (disjointness)
2. R some `owl:Thing` SubClassOf A (domain)
3. R some B SubClassOf A (scoped domain)
4. `owl:Thing` SubClassOf R only B (range)
5. A SubClassOf R only B (scoped range)
6. A SubClassOf R some B (existential)
7. B SubClassOf inverse R some A (inverse existential)
8. `owl:Thing` SubClassOf R max 1 `owl:Thing` (functionality)
9. `owl:Thing` SubClassOf R max 1 B (qualified functionality)
10. A SubClassOf R max 1 `owl:Thing` (scoped functionality)
11. A SubClassOf R max 1 B (qualified scoped functionality)
12. `owl:Thing` SubClassOf inverse R max 1 `owl:Thing` (inverse functionality)
13. `owl:Thing` SubClassOf inverse R max 1 A (inverse qualified functionality)
14. B SubClassOf inverse R max 1 `owl:Thing` (inverse scoped functionality)
15. B SubClassOf inverse R max 1 A (inverse qualified scoped functionality)

Figure 19: Most common axioms which could be produced from a single edge R between nodes A and B in a schema diagram: Manchester syntax.

tential; for `produces`: scoped range, existential; for `hasCookingInstructions`: scoped domain, scoped range, existential, inverse existential, inverse qualified scoped functionality; for `hasRequiredTime`: scoped range. We also have the additional axioms 11 and 12 from Figure 20.

Furthermore, we declare disjointness axioms: `Recipe`, `QuantityOfFood`, `Situation`, `RecipeInstructions`, `TimeInterval` are mutually disjoint.

After going through the standard axiom candidates for each node-edge-node tripel, we also contemplate whether there should be any axioms spanning more nodes or edges. However, none such seem to be appropriate in this case.

QuantityOfFood

We refer to Figure 9. Instead of listing formal axioms, we describe them by using the axiom names we have introduced in Figure 19. We thus have the following standard axioms. For `ofFoodType` and `ofQuantity`: scoped range, existential; for `hasQuantityKind` and `hasQuantityValue`: scoped domain, scoped range, existential, inverse existential, scoped qualified functionality; for `hasUnit`: scoped range, existential, scoped qualified functionality; for `hasNumericValue`: scoped range, existential, functionality.

Furthermore, we declare disjointness axioms: `QuantityOfFood`, `FoodType`, `QuantityKind`, `Quantity`, `QuantityValue`, `Unit` are mutually disjoint. We do not add any other axioms.

There is more to be said about allowed units for each `QuantityKind`, but we will not dive into this here.

1. Recipe SubClassOf requires only Situation
2. Recipe SubClassOf requires some Situation
3. Recipe SubClassOf produces only Situation
4. Recipe SubClassOf produces some Situation
5. hasCookingInstructions some RecipeInstructions SubClassOf Recipe
6. Recipe SubClassOf hasCookingInstructions only RecipeInstructions
7. Recipe SubClassOf hasCookingInstructions some RecipeInstructions
8. RecipeInstructions SubClassOf inverse hasCookingInstructions some Recipe
9. RecipeInstructions SubClassOf inverse hasCookingInstructions max 1 Recipe
10. RecipeInstructions SubClassOf hasRequiredTime only TimeInterval
11. Recipe SubClassOf requires some (hasConstituent some QuantityOfFood)
12. Recipe SubClassOf produces some (hasConstituent some QuantityOfFood)

Figure 20: Axioms for Figure 7

CookingEquipment, RecipeDifficultyLevel, RecipeClassification Stubs

We refer to Figures 10 and 11. The axioms are as follows. For hasConstituent: existential; for hasRecipeDifficultyLevel and hasRecipeClassification: scoped domain, scoped range, existential, inverse existential; for hasNameAsString and asString: scoped range.

Furthermore, we declare disjointness axioms: Recipe, CookingEquipment, DifficultyLevel, RecipeClassification are mutually disjoint. We do not add any other axioms.

NutritionalInformation

We refer to Figure 13. The axioms are as follows. For listsCalories, for listsCaloriesFromFat and for refersToServingSize: scoped range, existential, functional; for lists: scoped domain, scoped range, existential; for ofQuantity, ofType and isPercentageOfDailyValue: scoped range, existential.

Furthermore, we declare disjointness axioms: US-2014-Nutrition-Label, NutritionalContent, Recipe, Quantity, NutritionalContentType are mutually disjoint. We do not add any other axioms.

RecipeProvenance

We refer to Figure 15. The axioms are as follows. For wasDerivedFrom: scoped range, existential; for hasURL: scoped range. Furthermore, we declare disjointness axioms: Recipe and, Document are disjoint. We do not add any other

axioms.

4 Putting Things Together

Step 4: Put the modules together and add axioms which involve several modules. Reflect on all class, property and individual names and possibly improve them. Also check module axioms whether they are still appropriate after putting all modules together.

We put the modules together by first joining the different schema diagram. The result is shown in Figure 21. The diagram also indicates most modules using grey boxes. We have already been very careful with proper naming, so in this case we do not have to make any corresponding improvmements. Also, all axioms remain appropriate even after joining.

We do add one datatype property, though, `hasName` as indicated in the diagram, to give names to recipes. While a name may not appear central at first sight, it should be easily retrievable (and it is often identical with the name of the QuantityOfFood produced by the recipe), and it should be helpful when displaying search results. We only decleare a scoped range for `hasName`. Finally, let us contemplate on additional axioms which we may want to have for the resulting ontology. When inspecting the diagram, additional axioms are sometimes indicated when the schema diagram, understood as an undirected graph, does not have a tree structure, i.e. contains cycles. There are several such cycles in the diagram, which we inspect carefully. However, it turns out that in each case, no additional axioms are warranted. E.g., several classes refer to `Quantity`, but there are no additional relationships between the different quantities to indicate. So we only need to add additional disjointness axioms, and in fact all classes depicted in the diagram are mutually disjoint.

Finally, we go back to the competency questions listed in Section 2. We want to assess to what extent our ontology captures the required information to answer the competency questions. In cases where it does not, or not sufficiently, decisions need to be made whether the ontology should be modified or extended; but we will not go through this additional excercise herein.

The bulk of the competency questions concerns ingredients and equipment, which our ontology models.

For the first question, we notice that desserts (or breakfasts or sides, as in questions 4, 5, 8) are captured in the recipe classification stub, at least in a first, simple fashion. For gluten-free and low-calorie, we carry basic inforamtion in the nutritional information, but do not yet provide corresponding categorizations. These categorizations could be added – the appropriate place would be that they would be part of a refinement of the nutritional information module. The same holds for the notion of low-carb in the second question. Pot roast, as in question 2, and Chili as in question 3 are names of foods which are prepared following a recipe, i.e. it is the recipe name which holds this information. Under 100 calories is captured in nutritional information, though incompletely so, as the nutritional content of the final dish may need to be calculated from the

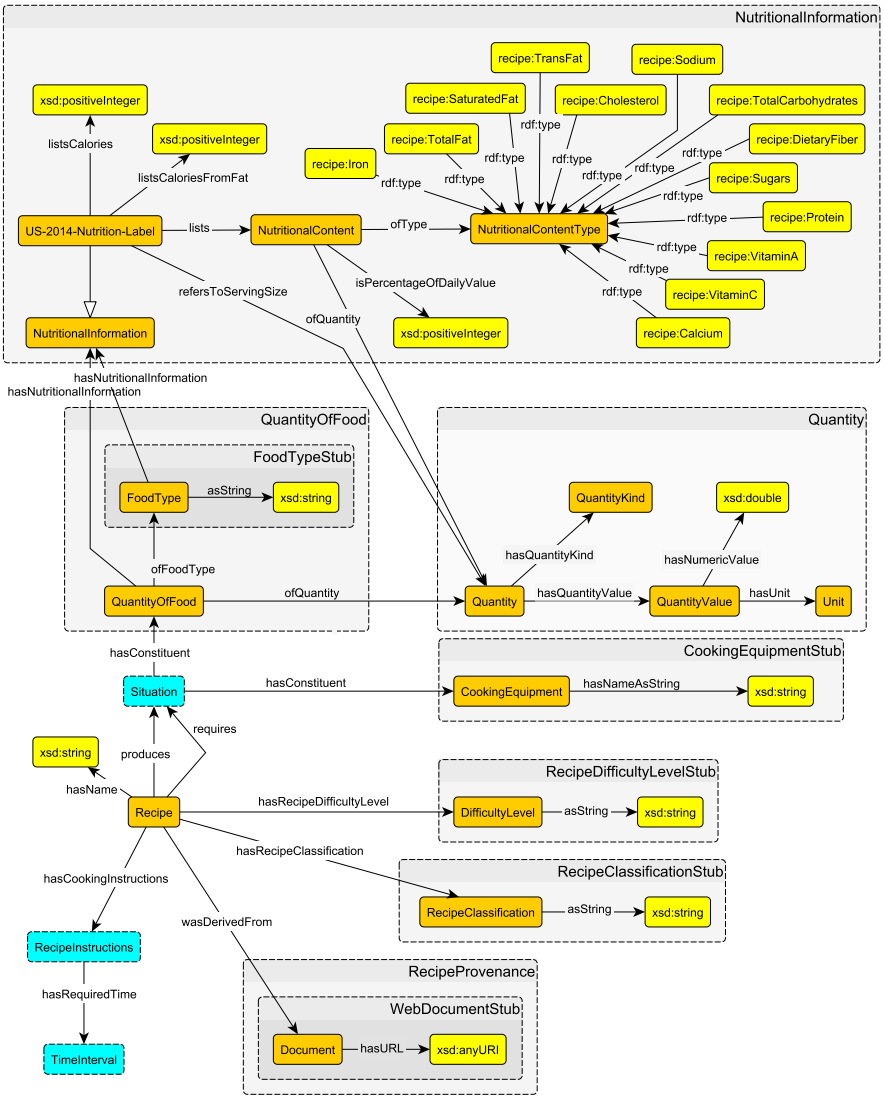

Figure 21: Recipe complete model

ingredients and serving sizes. Currently, our ontology can list this only if the web page from which the recipe originates carries this information. The fact that a breakfast may be "sweet" cannot be captured currently. How to model this would need some contemplation – in the end it is a subjective assessment, in a similar way in which "low-calorie" or "simple" would be a subjective assessment. On the other hand, given the use cases and the fact that recipes are retrieved from Web resources, this may be a case for simply adding some keyword tags obtained from the source.

5 Creating OWL Files

Step 5: Create OWL files.

Modeling up to this stage is usually done using paper, whiteboards, text documents. Only after we have created a solid modular model, we move to creating a data artefact in form of an OWL file which captures our ontology.

One of the problems with using OWL, however, is that it does not natively support modularization in the sense in which we are presenting it. In order to preserve the modularization, one option is to make use of different namespaces for the different related modules, and if a class can be understood as belonging to two different modules, then we recommend to duplicate this class under different namespaces and to set these classes to be equivalent using an `owl:equivalentClass` axiom. A cleaner solution, rather than indicating modules using namespaces, is to make use of the Ontology Design Pattern Representation Language OPLa [3, 12] which is expressed fully in OWL – however we do not go into further detail on this here.

We may choose to include mappings to external entities, e.g., alignments to other ontologies or to external ontology design patterns which were used as templates during the creation of our modules. Indeed, we recommend to keep such external models entirely separate from our own modules, by exclusively using local own, controlled namespaces within the modules, even if pieces from other ontologies are used verbatim. Instead, mappings to such external ontologies should be provided, and they should be provided as separate OWL files. The simple reason for this is that, once merged, it is hard to disentangle internal and external terms. Furthermore, external ontologies may change over time, and their axiomatizations or perspectives may not fit our model completely. By keeping the mappings separate, one can much more easily choose to opt into these mappings, or consult them only if needed.

The completed ontology should, of course, also be documented carfully. Documentation should reflect the modular structure.

Inspiration and some content for this tutorial was taken from [10]. A similar basic introductory example can be found in [6]. A report on a modular application ontology was published in [8].

References

[1] Aldo Gangemi and Peter Mika. Understanding the semantic web through descriptions and situations. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE – OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*, volume 2888 of *Lecture Notes in Computer Science*, pages 689–706. Springer, 2003.

[2] Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, editors. *Ontology Engineering with Ontology Design Patterns – Foundations and Applications*, volume 25 of *Studies on the Semantic Web*. IOS Press, 2016.

[3] Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Alfa Krisnadhi, and Valentina Presutti. Towards a simple but useful ontology design pattern representation language. In Eva Blomqvist, Óscar Corcho, Matthew Horridge, David Carral, and Rinke Hoekstra, editors, *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017.*, volume 2043 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[4] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language Primer (Second Edition*. W3C Recommendation 11 December 2012, 2012. Available from http://ww.w3.org/TR/owl2-primer/.

[5] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC Press, 2010.

[6] Adila Krisnadhi and Pascal Hitzler. Modeling with ontology design patterns: Chess games as a worked example. In Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, editors, *Ontology Engineering with Ontology Design Patterns*, volume 25 of *Studies on the Semantic Web*, pages 3–22. IOS Press/AKA Verlag, 2016.

[7] Adila Krisnadhi and Pascal Hitzler. The Stub Metapattern. In Karl Hammar, Pascal Hitzler, Agnieszka Lawrynowicz, Adila Krisnadhi, Andrea Nuzzolese, and Monika Solanki, editors, *Advances in Ontology Design and Patterns*, volume 32 of *Studies on the Semantic Web*, pages 39–64. IOS Press, Amsterdam, 2017.

[8] Adila Krisnadhi, Yingjie Hu, Krzysztof Janowicz, Pascal Hitzler, Robert A. Arko, Suzanne Carbotte, Cynthia Chandler, Michelle Cheatham, Douglas Fils, Timothy W. Finin, Peng Ji, Matthew B. Jones, Nazifa Karima, Kerstin A. Lehnert, Audrey Mickle, Thomas W. Narock, Margaret O'Brien,

Lisa Raymond, Adam Shepherd, Mark Schildhauer, and Peter Wiebe. The GeoLink Modular Oceanography Ontology. In Marcelo Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul T. Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab, editors, *The Semantic Web – ISWC 2015 – 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 301–309. Springer, 2015.

[9] Timothy Lebo, Satya Sahoo, and Deborah McGuinness, editors. *PROV-O: The PROV Ontology*. W3C Recommendation 30 April 2013, 2013. Available from http://ww.w3.org/TR/prov-o/.

[10] Monica Sam, Adila Krisnadhi, Cong Wang, John C. Gallagher, and Pascal Hitzler. An ontology design pattern for cooking recipes – classroom created. In Victor de Boer, Aldo Gangemi, Krzysztof Janowicz, and Agnieszka Lawrynowicz, editors, *Proceedings of the 5th Workshop on Ontology and Semantic Web Patterns (WOP2014) co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, October 19, 2014.*, volume 1302 of *CEUR Workshop Proceedings*, pages 49–60. CEUR-WS.org, 2014.

[11] Md. Kamruzzaman Sarker, Adila Alfa Krisnadhi, and Pascal Hitzler. OWLAx: A Protégé plugin to support ontology axiomatization through diagramming. In Takahiro Kawamura and Heiko Paulheim, editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016.*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[12] Cogan Shimizu, Quinn Hirt, and Pascal Hitzler. A Protégé plug-in for annotating OWL ontologies with OPLa. In Aldo Gangemi, Anna Lisa Gentile, Andrea Giovanni Nuzzolese, Sebastian Rudolph, Maria Maleshkova, Heiko Paulheim, Jeff Z. Pan, and Mehwish Alam, editors, *The Semantic Web: ESWC 2018 Satellite Events – ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers*, volume 11155 of *Lecture Notes in Computer Science*, pages 23–27. Springer, 2018.

[13] Cogan Shimizu, Pascal Hitzler, and Clare Paul. Ontology design patterns for Winston's taxonomy of part-whole-relationships. In *Proceedings WOP 2018*. To appear.