Modular Ontology Modeling: A Tutorial

Cogan Shimizu[1] [0000−0003−4283−8701] ✉, Pascal Hitzler[1] [0000−0001−8767−4136],, and Adila Krisnadhi[2] [0000−0003−0745−6804]

[1] Data Semantics Lab, Kansas State University, USA,
{coganmshimizu, hitzler}@ksu.edu
[2] Universitas Indonesia, krisnadhi@gmail.com

Abstract. We provide an in-depth example of modular ontology engineering with ontology design patterns. The style and content of this chapter is adapted from previous work and tutorials on Modular Ontology Modeling. It offers expanded steps and updated tool information. The tutorial is largely self-contained, but assumes that the reader is familiar with the Web Ontology Language OWL; however, we do briefly review some foundational concepts. By the end of the tutorial, we expect the reader to have an understanding of the underlying motivation and methodology for producing a modular ontology.

1 Motivation

Knowledge graphs are poised to be significant disruptors in the public and private sectors. They have become a popular paradigm for data synthesis, communication, and visualization. However, as with any sufficiently complex system, they require a considerable commitment of resources (e.g. time and expertise). And, indeed, a commitment to the maintenance and evolution of the knowledge graph. There are a number of ways to mitigate these challenges, but perhaps two of the most important are a) have a schema for the knowledge graph, and b) follow a methodology that promotes known best practices.

The role of a schema is to specify how data is organized and can be possibly interrelated. Moreover, we would want it to encapsulate our human conceptualization. Ontologies, as "explicit specifications of conceptualizations," seem like a natural fit for the role [3]. Ontologies offer a human accessible organization of immense amounts of data and act as a vehicle for the sharing and reuse of knowledge [10].

Unfortunately, many ontologies often do not live up to these promises. Monolithic ontologies, designed with very strong, or very weak, ontological commitments are very difficult to reuse across the same domain, let alone different domains. Strong ontological commitments lead to over-specification, to ontologies essentially only fit to the singular purpose for which they were originally designed. Weak commitments lead to ambiguity in the model, sometimes to the point where it is hard to determine the point of the model. We posit that one effective way to obtain ontologies which are easier to reuse, is to build them in a modular fashion. Ontology modules are instantiated by adapting Ontology Design Patterns to the domain and use-case [4,7]. These modules are then pieced together to create a modular ontology. A sufficiently modularized ontology [11] is designed such that individual users can easily adapt an ontology to their use cases, while maintaining integration and relationships with other versions of the ontology. A modular ontology is constructed by piecing together so-called ontology modules.

The purpose of this chapter is to provide a full execution of the Modular Ontology Modeling methodology, while promoting best practices, which in the end produces a modular ontology to be used as a knowledge graph schema.

The remainder of this chapter is organized as follows. Section 2 introduces preliminary concepts and context for our methodology. This section aims to provide a point of reference for our terminology used within the example. Section 3 provides a thorough description of the modular ontology modeling methodology deeply intertwined with a step-by-step example.

2 Foundations & Conventions

This section briefly reviews the three key concepts for modular ontology modeling. It is not exhaustive, but instead should be used as a point of reference for our terminology and conventions used throughout the chapter.

2.1 Foundations

Ontology Design Patterns (ODP) are the recognition of *conceptual* patterns that occur across domains[2]. They are, in principle, analogous to design patterns in Software Engineering. An ODP can be viewed as a tiny, self-contained ontology that solves a single, invariant problem, leveraging community-identified best practices. See `http://www.ontologydesignpatterns.org` for a list of example ODPs, however the patterns and their documentation are of very differing quality. For a smaller, self-contained, and curated list, see the Modular Ontology Design Library [16].

Modules are ODPs that have been adapted to a particular purpose. To do so, we use a pattern as a template. We therefore reusing the structure of the pattern, but not axiomatically referencing it. Essentially, during the instantiation process the ontology developer will replace the concept and property names to those that are more suited to the domain at hand. This process is outlined in detail, with additional motivation, in [5].

However, in doing so, we lose the connection to the original pattern as OWL does not natively support modularization. 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.

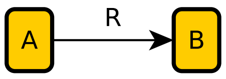


Fig. 1: Generic node-edge-node schema diagram for explaining systematic axiomatization

A cleaner and simpler solution for indicating modules is to make use of the Ontology Design Pattern Representation Language (OPLa) which is a structured set of annotations fully expressed in OWL.[3] OPLa allows us to use a structured set of annotations to connect the provenance of axioms and concepts to different patterns. It further allows us to specify which ontological entities belong to which module, pattern, or even other ontologies. Its original specification and extensions can be found in [8] and [6], respectively.

2.2 Conventions

Axiomatization. The Modular Ontology methodology emphasizes a schema-driven axiomatization. That is, we define and follow a systematic way of axiomatizing a schema diagram by examining each node-edge-node construct in the diagram. Given a node-edge-node construct with nodes A and B and edge R from A to B, as depicted in Figure 1, we check which axioms should be included.[4] Figure 2 provides these axioms in Manchester syntax and our labels for these axioms. All axiom types except disjointness and those utilizing inverses also apply to datatype properties. A structural tautology is a *logically* meaningless axiom; instead it is to convey human meaning. That is, its intent is to indicate the proper or intended usage of two classes.

Visual Syntax for Schema Diagrams. A schema diagram is an informal, but intuitive way for conveying information about the structure and contents of an ontology. We do use, however, a consistent visual syntax for convenience. Orange boxes are classes and indicate that they are central to the diagram. Blue dashed boxes indicate a reference to another diagram, pattern, or module. Gray frames, with a dashed outline, contain modules. Arrows depict relations and open arrows represent subclass relations. Yellow ovals indicate data types (and necessarily, arrows pointing to a datatype are data properties). Finally, purple boxes represent controlled vocabularies. That is, they represent a controlled set of IRIs that are of that type.

[3] The OPLa Annotator plugin for Protégé can be used to easily add OPLa annotations to your modular ontology.

[4] The OWLAx plug-in [14] for Protégé provides a graphical interface for adding these axioms. This behaviour is also supported in a limited fashion by the CoModIDE plugin [15].

1.	A SubClassOf B	(subclass)
2.	A DisjointWith B	(disjointness)
3.	R some `owl:Thing` SubClassOf A	(domain)
4.	R some B SubClassOf A	(scoped domain)
5.	`owl:Thing` SubClassOf R only B	(range)
6.	A SubClassOf R only B	(scoped range)
7.	A SubClassOf R some B	(existential)
8.	B SubClassOf inverse R some A	(inverse existential)
9.	`owl:Thing` SubClassOf R max 1 `owl:Thing`	(functionality)
10.	`owl:Thing` SubClassOf R max 1 B	(qualified functionality)
11.	A SubClassOf R max 1 `owl:Thing`	(scoped functionality)
12.	A SubClassOf R max 1 B	(qualified scoped functionality)
13.	`owl:Thing` SubClassOf inverse R max 1 `owl:Thing`	(inverse functionality)
14.	`owl:Thing` SubClassOf inverse R max 1 A	(inverse qualified functionality)
15.	B SubClassOf inverse R max 1 `owl:Thing`	(inverse scoped functionality)
16.	B SubClassOf inverse R max 1 A	(inverse qualified scoped functionality)
17.	A SubClassOf R min 0 B	(structural tautology)

Fig. 2: Most common axioms, represented in Manchester Syntax, which could be produced from a single edge R between nodes A and B in a schema diagram.

3 Walkthrough

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, such as future extensions, refinements, or evolution of the ontology and reuse of the ontology by others. Conventional wisdom may suggest that it is always better for a use case 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. That is, modeling choices, or ontological commitments, may be made which fit only the very specific and detailed use case. Consequently, later modifications become very cumbersome as they may conflict with those earlier ontological commitments.

We now define a use case scenario for this tutorial. The setting that we have in mind concerns online cooking recipes, and in particular the task of integrating recipes from different websites in order to enable a fine-grained cross-website search for recipes:

Design an ontology that can be used as part of a "recipe discovery" website. The ontology shall be set up such that content from existing recipe websites can be mapped into it (i.e. the ontology will be 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.

First, we notice that data will come from multiple sources that are not well-specified. This means that our ontology needs to be general enough to accommodate different conceptual representations on the source side. Second, the ontology shall be extendable towards additional related data, meaning that we have to accommodate such extension capabilities, to a reasonable extent, without knowing what these future extensions will look like; this again requires a general model. In contrast, 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 specificity and generality, which is a typical use-case for modular ontology modeling.

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, that may be used to retrieve data from the knowledge base. They provide an initial framework for the use cases; that is, the main concepts in each query should be represented in the ontology. In other instances, they may be formulated by the domain experts, expected users of the ontology or knowledge base, and the ontology engineers themselves. It is important for competency questions to fall within the scope of the use-case, as well as to cover a wide range of complexity (e.g. simple vs. complex usages). For our scenario, we have generated some possible competency questions, as follows.

1. What are some 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. What are some sweet breakfast items under 100 calories?
5. Breakfast dishes which can be prepared quickly with two potatoes, an egg, and some flour.
6. How do I prepare chicken thighs in a slow cooker?
7. A simple recipe using pork shoulder and spring onions.
8. A side prepared using Brussels sprouts, bacon, and chestnuts.

It is also important, at this stage, to examine the possible data sources, or in other cases intended or actual data sources. While in this example we did not have data sources at hand, this is not always the case. Indeed, data sources are generally acquired alongside the use-case scenario. Examining the data provides can provide insight as to what can, or cannot, be represented in the ontology, and is highly recommended. Sensor or telemetry limitations, bounded by physical laws or policies like the GDPR, may prevent which data can possibly be collected, therefore providing concrete limitations on what can be stored in the knowledge graph.

Fortunately, for this example, searching online provides a number of cooking websites and recipe repositories, to name a few: allrecipes.com, food.com, epicurious.com. Their recipe pages commonly list ingredients, cooking instructions, and supplementary information such as nutritional information. We note that this chapter is only concerned with producing a suitable underlying ontology.[5]

With a few suitable data sources at hand, we may now iterate over the competency questions for their own suitability. Some competency questions may need to be removed or modified to fit the available data. For example, the recipes rarely mention equipment as explicit requirements or provide up-front, qualitative descriptions (e.g. sweet breakfast item). We should keep these in mind, though, and make sure that the ontology that we produce is extendable towards the future inclusion of such aspects. For example, had this tutorial been authored some years ago, only a few recipes would have been indicated as low-carb or gluten-free. Now, almost every recipe repository has categories that directly cater to these recipe types. It is an obvious reminder that data evolves and we must engineer ontologies that are capable of evolving alongside it.

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 concepts may be be initially incorporated, or alternatively, extensibility towards future inclusion can be kept in mind during modeling. The decision process regarding what to model now, later, or ever is called *scoping*. By the end of this step, we should have a clear understanding of the scope of the target ontology.

Step 3. Identify key notions from the data and the use case and identify which pattern should be used for each (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
Consider the key notions to be the main concepts occurring in the competency questions or those concepts identified as central in the available data sources.

From the competency questions we may find some insight as to what may be important. We arrived at the below generalizing the competency questions into broader usage scenarios. These are helpful for perspective and focusing terms into key notions.

- Retrieval of cooking instructions.
- Search recipes by ingredients.
- Search by properties of the prepared food (e.g. calories or carb content).
- Search by properties of the recipe (e.g. cooking time or degree of complexity).

In our example, possible key notions are recipe, food (or ingredient), time, equipment, classification of food prepared (e.g. *dinner* or *side*), difficulty level, and nutritional information. We arrive at these by inspecting the competency questions, use-case scenario, and available data. This is a subjective process, but can be enhanced with objective measures, such as centrality, term frequency, and so on.

There are also key notions that appear in relation to this first pass such as the name for a recipe, the recipe instructions themselves, and from where the recipe

[5] That is to say, we are only creating the ontology, not the overarching software system that would extract and make use of the data contained in the knowledge base.

Recipe	Plan
RecipeName	NameStub
RecipeInstructions	Document
TimeInterval	TimeInterval
QuantityOfFood	QuantityOfStuff
Quantity	Quantity
Equipment	Stub
FoodType	Stub
Difficultylevel	Stub
RecipeClassification	Stub
NutritionalInfo	QuantityOfStuff
Source	Provenance

Fig. 3: All key notions with their corresponding patterns.

was sourced (provenance). Some of these tangential concepts, while important for context, are not essential (nor central) to the ontology at hand. The modeling for them would need to be robust enough to be helpful, but do not require an initial modeling in depth. One way to acknowledge this is through the use of the Stub pattern, which can be seen in Figure 7.

In the interest of space, we introduce the corresponding patterns alongside their module instantiations in the next step. For some of the key notions, we also discuss different patterns that may be applicable.

Step 4. Instantiate these key notions from the pattern templates (if there is a suitable pattern), and adapt/change the result as needed, arriving at modules. Develop the remaining modules from scratch.
We will go through the key notions identified in the previous step one by one, refine the list, and generate their corresponding schema diagrams.
Recipe is central to what we intend to do. We note that the name of the recipe, which is often identical to the dish which is going to be prepared, should be recorded. To do so, we will use the previously described NameStub pattern.[6] We now want to identify a pattern which will be the basis for the core of the recipe modeling.

However, 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 (such as those mentioned in Section 2.1) and to contemplate which may fit best. Sometimes there may be more than one candidate pattern and it is necessary to evaluate how accurately each models both our data and intent; let us look at three such candidates.[7]

[6] Names are complicated entities and we realize that the name for a particular dish may vary significantly across the world. While this example does not intend to model that upfront, we may wish to do so in the future and use the NameStub pattern to facilitate future extensibility.

[7] In the interest of space, we do not include the diagrams here, but they may be found at `http://ontologydesignpatterns.org/wiki/Submissions:ContentOPs`.

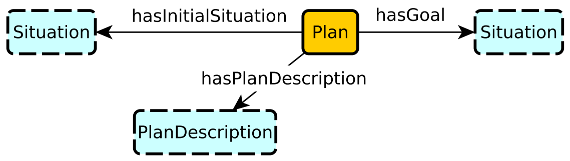


Fig. 4: The schema diagram for the Basic Plan ontology design pattern.

First, we check whether it makes sense to consider a recipe to be a Document. There is certainly a perspective from which this this seems valid: in the end, the recipse itself is simply a document that we download of the internet? However, the Document pattern seems to be a rather generic notion which does not naturally cater to the rest of our key notions. For example, we would not indicate whether a document is low-carb or not. This line of thinking leads 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 generally contain step-by-step descriptions of the food preparation process, another alternative pattern is Sequence, which is a fundamental ontology design pattern. Yet, in the same manner as Document, it seems clear that it does not capture many of the aspects that are important for our competency questions. This line of thinking leads us to the conclusion that some parts of the recipe (e.g. 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 sequence of the preparation steps are particularly relevant to our task, namely the discovery of recipes.

We could also think of recipes as *processes* which may help us to emphasize the input (the ingredients), their transformation (preparation and cooking), and output (the dish) aspects. For some, this may indeed be a valuable perspective. However, as with the others, it does not sufficiently cover our core use-case: that being a resource for recipe discovery.

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 for plans; it seems reasonable to think of recipes as plans to produce something. The Plan pattern is depicted in Figure 4. A plan leads from an initial situation to another situation, which is understood to be the goal of the plan. The initial situation would have availability requirements (e.g. ingredients or equipment), while the goal situation would have the prepared food available. The ingredients and equipment listed in the recipe are necessary for these respective situations; that is, they are *constituents* of the situations. Putting these these together, we arrive at a first piece of the Recipe module, as an instantiation of the Plan template. Its schema diagram is depicted in Figure 5.[8]

[8] See [1] for a deeper discussion on descriptions, plans, situations, etc.

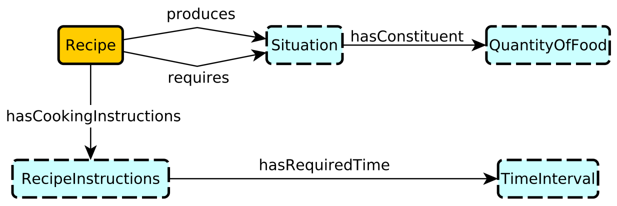


Fig. 5: Recipe as plan

Food is a key notion, but unspecific. Food can be things such as cucumbers, lasagna, or chicken Kiev. Should we distinguish between ingredients and resultant dishes? But what about, say, Pesto Genovese? It is not a dish itself, but an ingredient in some recipes; yet, there are recipes on how to make it. Indeed, many cooking ingredients are made from even more basic ingredients. As such, it does not make much sense to distinguish between ingredients and dishes, when talking about recipes. This type of discussion is central to identifying 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.

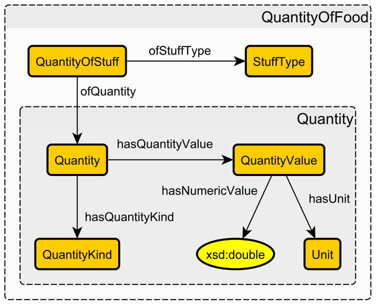
Now, when we say Pesto Genovese, what exactly do we mean? Do we mean Pesto Genovese in general, or do we mean, say, two teaspoons of it required by some recipe? It seems for recipes that the quantity of a required or produced food item is important. Thus, the Quantity pattern seems useful for our purposes, 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.

So we understand that there should be a concept of QuantityOfFood (e.g. 1 tsp. of butter) which is always of some quantity (e.g., 1 tsp.) and at the same time is of some type of foodstuff (e.g. butter). 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 6a and 6b for their schema diagrams. Our pattern for Quantity is directly derived from QUDT.[9]

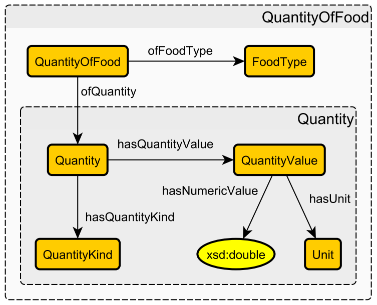
Equipment, Classification, Difficulty Level come next on our list of key notions. Equipment may be a slow cooker or a blender. However, while keeping track of what special equipment is necessary, our scenario does not call for a detailed modeling of kitchen equipment, itself, at this stage. Thus, we delay a detailed modeling and consciously restrict the scope of our model.

Decisions, such as this, are very important during the modeling process. Indeed, it is impossible to always model all details, as we would simply end up with a model of almost everything! At the same time, we would like our ontology capable of later reuse or possibly re-purposed for a scenario in which a detailed modeling of equipment may be more important. This means, that we do not want to merely introduce a data property, such as requiresEquipment with

[9] See `http://qudt.org/`.



(a) The QuantityOfStuff pattern, the inner box indicates the Quantity pattern.



(b) The QuantityOfFood module, instantiated from the Quantity pattern.

Fig. 6: The RecipeProvenance module instantiated from the Provenance pattern.

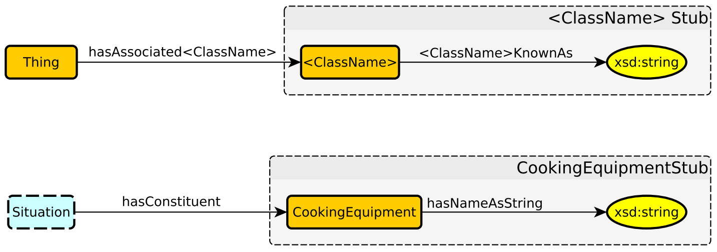


Fig. 7: The Stub pattern (above) and its instantiation for equipment (below).

range xsd:string for the names of the equipment. Instead, we want to utilize a slightly more sophisticated approach where we use a node as placeholder for the equipment entity. The corresponding ontology design pattern is called a Stub [12], and it is depicted in Figure 7 together with the instantiation for equipment which we will use. Note that we attach the CookingEquipment as constituent to a Situation, which seems to be its natural place.

We opt to use the Stub pattern for other for DifficultyLevel and RecipeClassification. Their corresponding schema diagrams are constructed analogously to 7. Stubs are also used in other places that we have not further talked about. For example, FoodType, as it appears in Figure 6b, as it is conceivable that there may be a sophisticated model of different food types.

Nutritional Information is 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.[10] While this may seem overly specific, by virtue of our modular modeling approach

[10] See `https://en.wikipedia.org/wiki/Nutrition_facts_label#United_States`.

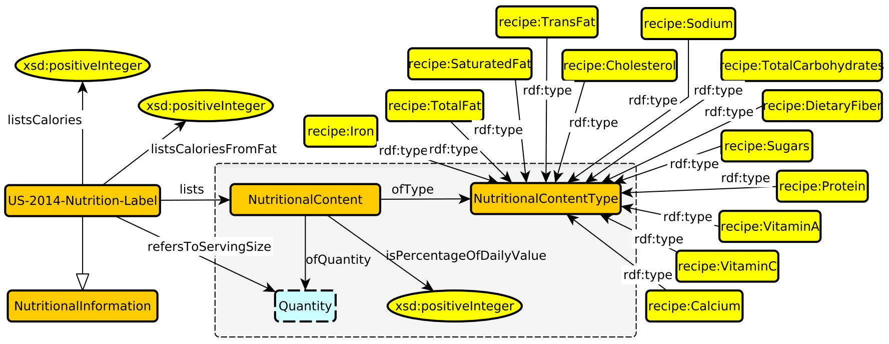


Fig. 8: Nutritional Information module. The box indicates a modified instance of the QuantityOfStuff pattern.

it would be easy to replace the NutritionalInformation module with one tailored to other countries or nutritional convictions. In fact, we will highlight this by creating the class US-Nutrition-Label as a subclass of the NutritionalInformation class.

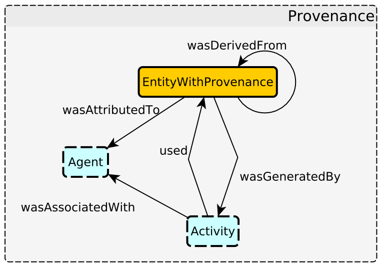
These labels have highly structured content. We will not be concerned with layout issues, nor is it necessary that we model all content, such as the "% Daily Value" amounts for fat or sodium. We will list absolute amounts for fat, saturated fat, trans fat, cholesterol, sodium, carbs, dietary fiber, sugars, and protein, and "% Daily Value" amounts for Vitamins A and C, Calcium, and Iron. We represent these as instances of the class NutritionalContentType.[11] It seems obvious that we will reuse the QuantityOfStuff pattern, 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. We also need to record the serving size to which the nutritional information refers, and this can again be done using the Quantity pattern, as well as the calorie content and calories-from-fat content, but we use data properties here. This module is shown in Figure 8.

Provenance does not initially appear in our list of key notions. Instead, we identify this need not from the competency questions, but from use case. And indeed, our scenario states that queries shall produce links to recipes from different recipe websites. However, our modeling so far has not covered anything that would make it possible to track from where a recipe was obtained. Thus, we need to do model some notion of provenance.

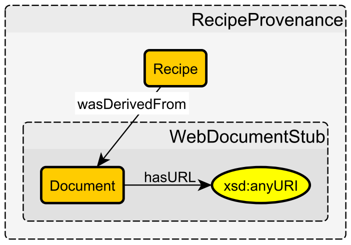
The schema diagram for a generic provenance pattern, presented in in [17], is derived from the W3C recommendation, PROV-O [13], is provided in Figure 9a.

The key idea idea, here, is that any entity for which provenance is important, was both generated by some Activity and derived that entity from some

[11] This is essentially a small controlled vocabulary.



(a) The Provenance pattern.



(b) The RecipeProvenance module.

Fig. 9: The RecipeProvenance module instantiated from the Provenance pattern.

other entity. Analogously, a recipe is downloaded from a website and mapped into the ontology. Recalling our discussion regarding Recipe, we may use the Document pattern, to say that a Recipe wasDerivedFrom some web Document. Our RecipeProvenance module is shown in Figure 9b.

We have now produced diagrams for each of our identified key notions and used the 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 3.

It is not always the case, however, that we will have an ontology design pattern that is applicable to the key notion. In this case, our worked example deals will relatively common concepts and thus by looking through the ontology design patterns portal, existing literature, and the modular ontology design library, we can find such patterns. In the event that a pattern cannot be identified, the developers and domain experts must work closely together to either develop a new pattern that can cover that notion (and then instantiate it) or author a module (that is a tiny ontology that solves exactly this modeling problem) directly.

Step 5. Add axioms for each module, informed by the pattern axioms.
We now produce the OWL axioms for each of the modules using the earlier schema diagrams as guidance. In many cases, the axioms would be derived from those provided with the patterns. Instead we will recreate them from scratch in order to gain a deeper understanding. We will produce an exhaustive list of axioms that seem appropriate for our model, while steering away from overly strong ontological commitments. In our examples, we will use Manchester syntax. This axiomatization is done systematically for the node-edge-node constructs in the diagram, according to the process described in Section 2.2.
Recipe, from Figure 5, is first on our list. The axioms can be found in Figure 10. Note that, generally speaking, we prefer the scoped versions of axioms, as they are weaker axioms from the perspective of formal semantics.

Items 1 and 2 refer to the requires edge and adjacent nodes in Figure 5. 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

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)

Fig. 10: Axioms for Figure 5

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 2. 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, existential; 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 10.

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 6b. Instead of listing formal axioms, we describe them by using the axiom names we have introduced in Figure 2. 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.

CookingEquipment, RecipeDifficultyLevel, RecipeClassification Stubs We refer to Figure 7. 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 8. 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 9b. 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.

Step 6. Put the modules together and revise naming conventions for all class, property, and individual names, as necessary.
To start, we simply join the different schema diagrams together, as shown in Figure 11. Joining, in this case, is not axiomatic. It is simply overlapping the diagrams (and moving things around, as necessary) so that concepts in one schema diagram align where indicated. In the figure we have multiple uses of the Quantity module, so we connect to the same one. We also try to reduce crossing edges as much as possible. In the event that we could not, we would still want to un-clutter the diagram by fully representing modules only once, and using a blue-dashed box to indicate that the concept in question is fully represented elsewhere. 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 improvements. In general, however, these problems may present as mismatch of key terms in different modules, especially if the development of the ontology occurs over a long time.

Step 7. Add axioms which involve several modules and review the entire, connected model for consistency.
Let us contemplate additional axioms that we may want to add to the resulting ontology. First, we add one datatype property, 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 declare a scoped range for hasName.

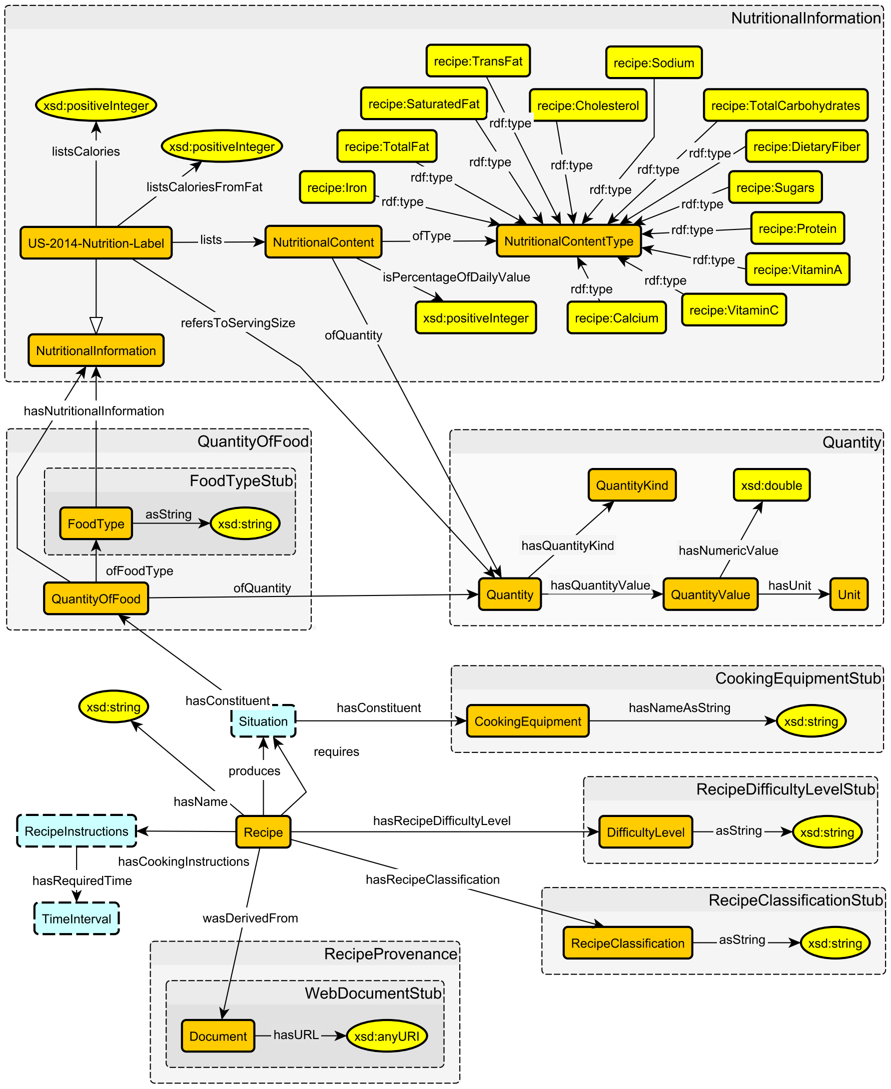


Fig. 11: The schema diagram for the complete Recipe model.

When inspecting the rest of the diagram, additional axioms are sometimes indicated when the schema diagram, understood as an undirected graph, 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. For example, several classes refer to Quantity, but there are no additional relationships between the different quantities to indicate. Thus, we add additional disjointness axioms, and in fact all classes depicted in the diagram are mutually disjoint.

Finally, we revisit the competency questions listed in Step 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.

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, and 8) are captured in the RecipeClassification stub, in a simple fashion. For gluten-free and low-calorie, we carry basic information in the nutritional information, but do not 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. However, as low-calorie or low-carb are relative measures, this may not be a job for the ontology. Pot roast, as in question 2, and chili, as in question 3, are names of foods which are prepared following a recipe. The fact "under 100 calories" is captured in NutritionalInformation, though incompletely so, as the nutritional content of the final dish may need to be calculated from the 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.

This step can be summarized as the final pass over before the ontology is complete. First, the diagram is inspected for any problematic cycles and determine disjointness. Then, we revisit our competency questions, as they inform expectations on how the ontology will be used. If competency questions (that have been determined to be in scope) cannot be answered, we must extend or modify the ontology so that they may.

Step 8. Create OWL files.
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.

For this step, we recommend the use of an ontology editor. In particular, we use Protégé and an assortment of plug-ins that we have developed, such as CoModIDE (the comprehensive modular ontology integrated development environment), which allows us to automatically generate axioms by replicating our schema diagram onto a graphical canvas, and the OPLa Annotator, which allows us to very quickly and easily annotate the ontology with the modular structure.

We also recommend to keep external models, such as the patterns used as templates, 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 as separate OWL files. This allows us to easily disentangle internal and external terms, allows the models to evolve separately, and provides a mechanism for an exchangeable interface with external models, thus improving reuse. Finally, the completed ontology should be documented carefully and clearly reflect the modular structure.

4 Conclusion

We have now produced a modular ontology that captures details about recipes. We have developed a number of modules capturing the key notions, as extracted from the use-case, competency questions, and available data. The patterns upon which each module is based can be found in Figure 3. For each of the modules, we followed our systematic axiomatization process, as outlined in 2.2. After assembling the modular ontology, we briefly reviewed the axioms for consistency, clarity, purpose, and accuracy. For a more in depth look into these processes, we direct the reader to [11] and [9].

4.1 Advancing the Methodology

As we mentioned at the beginning of this chapter, knowledge graphs are complex entities. The methodology to produce them is no less so—we still have many considerations to take into account as we further mature the paradigm. In particular, we see the following questions and advances to be made.

1. This methodology, generally, relies on face-to-face communication and ad hoc graphical modeling. How can this methodology be adapted for remote collaborations?
2. Are there additional patterns that we are missing?
3. What are better documentation strategies? And how can existing tools be modified to incorporate modular structure into their documentation style?

Finally, we have plans to continue the development of the CoModIDE [15] platform for graphical modeling. This platform allows for the direct creation of an ontology by drawing a schema diagram on a graphical canvas. It is a plugin for Protégé, which allows a developer to leverage the entire power of an ontology editor, but also the convenience and ease of use a graphical editor. It offers support for patterns and modules by providing a drag-and-drop mechanism from an embedded ODP library.

Moving forward, we will be implementing a tighter integration between it and OPLa, particular in the use of creating your own modules from scratch. We will furthermore be adding the ability to swap out the embedded ODP library for more domain specific collections. For those interested, the plug-in (and its source code) can be found online.[12]

Acknowledgements. The authors acknowledge partial support from the financial assistance award 70NANB19H094 from the U.S. Department of Commerce, National Institute of Standards and Technology and the National Science Foundation under Grant No. 1936677.

References

1. Gangemi, A., Mika, P.: Understanding the semantic web through descriptions and situations. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) 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. Lecture Notes in Computer Science, vol. 2888, pp. 689–706. Springer (2003)
2. Gangemi, A., Presutti, V.: Ontology design patterns. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 221–243. International Handbooks on Information Systems, Springer (2009). https://doi.org/10.1007/978-3-540-92673-3_10, `https://doi.org/10.1007/978-3-540-92673-3_10`
3. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge acquisition **5**(2), 199–220 (1993)
4. Hammar, K., Hitzler, P., Krisnadhi, A., Lawrynowicz, A., Nuzzolese, A.G., Solanki, M. (eds.): Advances in Ontology Design and Patterns [revised and extended versions of the papers presented at the 7th edition of the Workshop on Ontology and Semantic Web Patterns, WOP@ISWC 2016, Kobe, Japan, 18th October 2016], Studies on the Semantic Web, vol. 32. IOS Press (2017)
5. Hammar, K., Presutti, V.: Template-based content ODP instantiation. In: Hammar, K., Hitzler, P., Krisnadhi, A., Lawrynowicz, A., Nuzzolese, A.G., Solanki, M. (eds.) Advances in Ontology Design and Patterns [revised and extended versions of the papers presented at the 7th edition of the Workshop on Ontology and Semantic Web Patterns, WOP@ISWC 2016, Kobe, Japan, 18th October 2016]. Studies on the Semantic Web, vol. 32, pp. 1–13. IOS Press (2016). https://doi.org/10.3233/978-1-61499-826-6-1, `https://doi.org/10.3233/978-1-61499-826-6-1`
6. Hirt, Q., Shimizu, C., Hitzler, P.: Extensions to the ontology design pattern representation language. In: WOP@ISWC. CEUR Workshop Proceedings, vol. 2459, pp. 76–75. CEUR-WS.org (2019)
7. Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.): Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25. IOS Press (2016)
8. Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A.A., Presutti, V.: Towards a simple but useful ontology design pattern representation language. In: Blomqvist, E., Corcho, Ó., Horridge, M., Carral, D., Hoekstra, R. (eds.) Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with

[12] See `https://github.com/comodide/CoModIDE`.

the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017. CEUR Workshop Proceedings, vol. 2043. CEUR-WS.org (2017), `http://ceur-ws.org/Vol-2043/paper-09.pdf`

9. Hitzler, P., Krisnadhi, A.: A tutorial on modular ontology modeling with ontology design patterns: The cooking recipes ontology. CoRR **abs/1808.08433** (2018), `http://arxiv.org/abs/1808.08433`
10. Hitzler, P., Shimizu, C.: Modular ontologies as a bridge between human conceptualization and data. In: Chapman, P., Endres, D., Pernelle, N. (eds.) Graph-Based Representation and Reasoning - 23rd International Conference on Conceptual Structures, ICCS 2018, Edinburgh, UK, June 20-22, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10872, pp. 3–6. Springer (2018). https://doi.org/10.1007/978-3-319-91379-7_1, `https://doi.org/10.1007/978-3-319-91379-7_1`
11. Krisnadhi, A., Hitzler, P.: Modeling with ontology design patterns: Chess games as a worked example. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns, Studies on the Semantic Web, vol. 25, pp. 3–22. IOS Press/AKA Verlag (2016)
12. Krisnadhi, A., Hitzler, P.: The Stub Metapattern. In: Hammar, K., Hitzler, P., Lawrynowicz, A., Krisnadhi, A., Nuzzolese, A., Solanki, M. (eds.) Advances in Ontology Design and Patterns. Studies on the Semantic Web, vol. 32, pp. 39–64. IOS Press, Amsterdam (2017)
13. Lebo, T., Sahoo, S., McGuinness, D. (eds.): PROV-O: The PROV Ontology. W3C Recommendation 30 April 2013 (2013), available from http://ww.w3.org/TR/prov-o/
14. Sarker, M.K., Krisnadhi, A.A., Hitzler, P.: OWLAx: A Protégé plugin to support ontology axiomatization through diagramming. In: Kawamura, T., Paulheim, H. (eds.) Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016. CEUR Workshop Proceedings, vol. 1690. CEUR-WS.org (2016)
15. Shimizu, C., Hirt, Q., Hitzler, P.: A protégé plug-in for annotating OWL ontologies with opla. In: Gangemi, A., Gentile, A.L., Nuzzolese, A.G., Rudolph, S., Maleshkova, M., Paulheim, H., Pan, J.Z., Alam, M. (eds.) The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11155, pp. 23–27. Springer (2018). https://doi.org/10.1007/978-3-319-98192-5_5, `https://doi.org/10.1007/978-3-319-98192-5_5`
16. Shimizu, C., Hirt, Q., Hitzler, P.: MODL: A modular ontology design library. In: WOP@ISWC. CEUR Workshop Proceedings, vol. 2459, pp. 47–58. CEUR-WS.org (2019)
17. Shimizu, C., Hitzler, P., Paul, C.: Ontology design patterns for Winston's taxonomy of part-whole relations. In: Demidova, E., Zaveri, A., Simperl, E. (eds.) Emerging Topics in Semantic Technologies – ISWC 2018 Satellite Events [best papers from 13 of the workshops co-located with the ISWC 2018 conference]. Studies on the Semantic Web, vol. 36, pp. 119–129. IOS Press (2018). https://doi.org/10.3233/978-1-61499-894-5-119, `https://doi.org/10.3233/978-1-61499-894-5-119`