Modular Ontology Modeling

Pascal Hitzler

Data Semantics Laboratory (DaSe Lab)
Center for Artificial Intelligence and Data Science (CAIDS)
Kansas State University
http://www.pascal-hitzler.de

Premise

Many ontologies are hard to understand and to re-use.

Some reasons:

- **Poor (ad-hoc) modeling.**
- **Large, monolithic ontologies.**
- **Different use-case requirements on granularity (some parts too fine-grained, others too coarse).**
- **Different requirements on data representation for parts of the ontology (e.g., how spatial information is encoded).**

Approach: Two main components

1. **Modules**
   - Rather than thinking of an ontology primarily as an enhanced taxonomy, we think of it as a set of interrelated (and possibly overlapping) modules.
   - Each module is essentially a part of an ontology representing a complex concept in a way which “makes sense” for a human expert. E.g., “oceanographic cruise”.
2. **Use of ontology design patterns (ODPs)**
   - An ODP is a solution template for a recurring ontology modeling problem.
   - ODPs are instantiated (and modified) to become modules. E.g., a general “Trajectory” ODP may be used as a template to create an “ocean science cruise trajectory” module.

Modeling Teamwork

The modeling team ideally has:

- domain experts
- data experts
- ontology engineers

Divide and Conquer

- First decide on the set of modules to be modeled, then draft modules one at a time.

Joint modeling

- Work mainly through schema diagrams and natural language with the domain and data experts.
- Ontology engineers spell out model details between meetings, and cycle back to the experts for feedback.

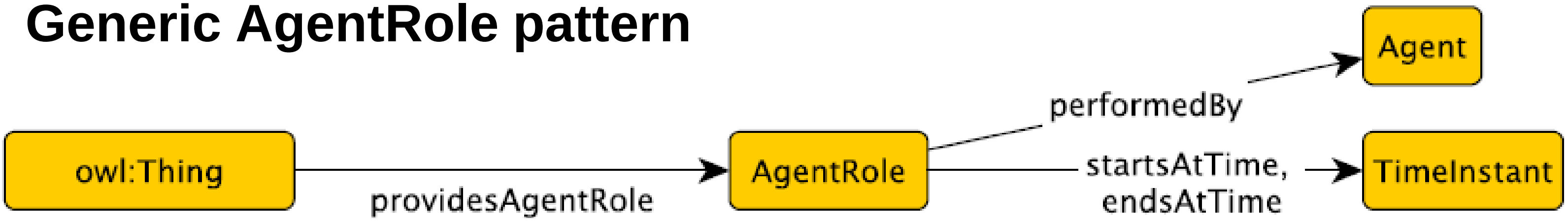
Modeling process – steps

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 (if a suitable pattern is available). Many can remain “stubs” if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

A Few Pattern Examples

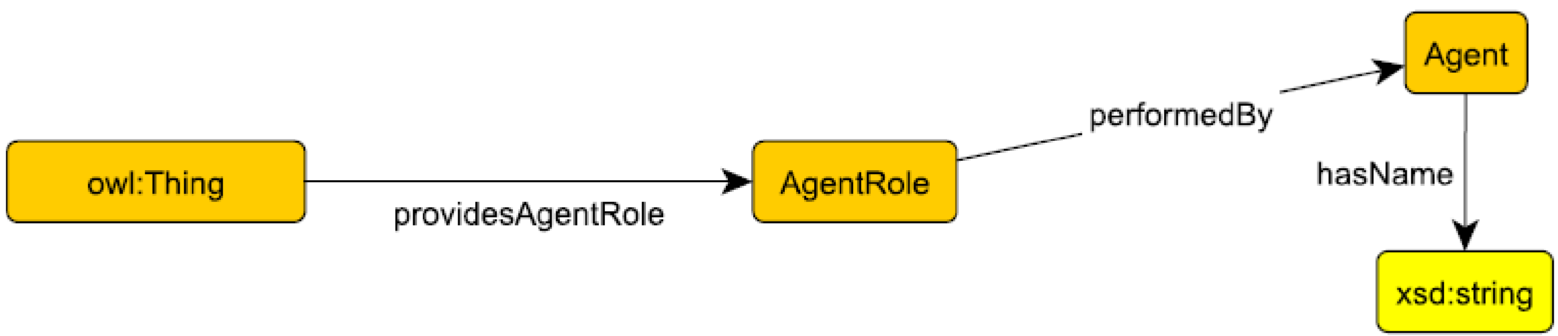
Joining patterns

Generic AgentRole pattern



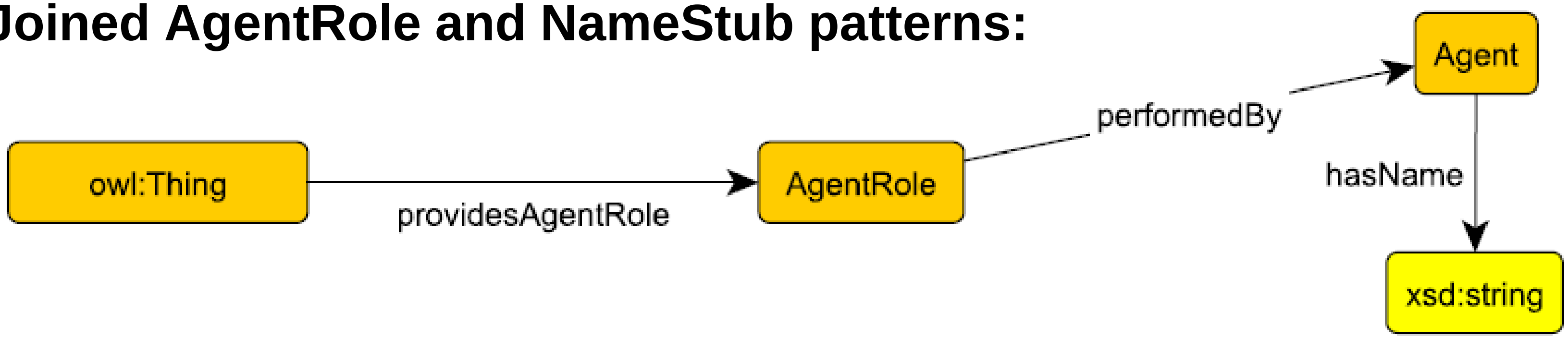
Generic NameStub pattern

Joined:

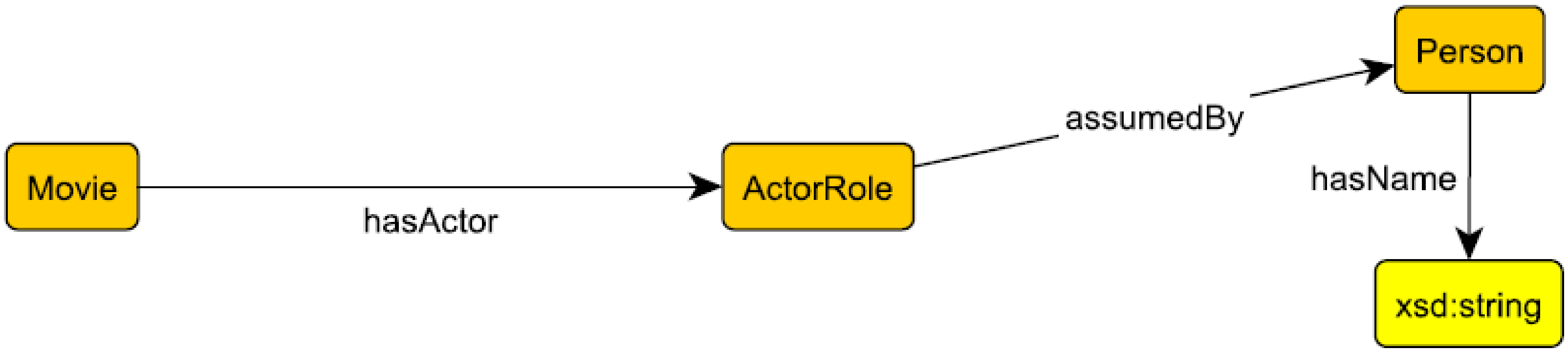


Patterns as templates

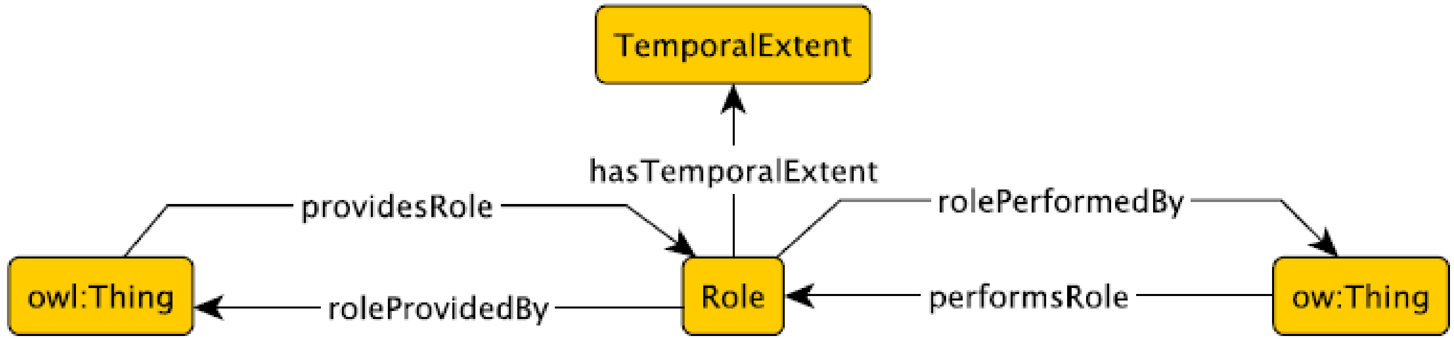
Joined AgentRole and NameStub patterns:



Used as a template for a concrete modeling problem:

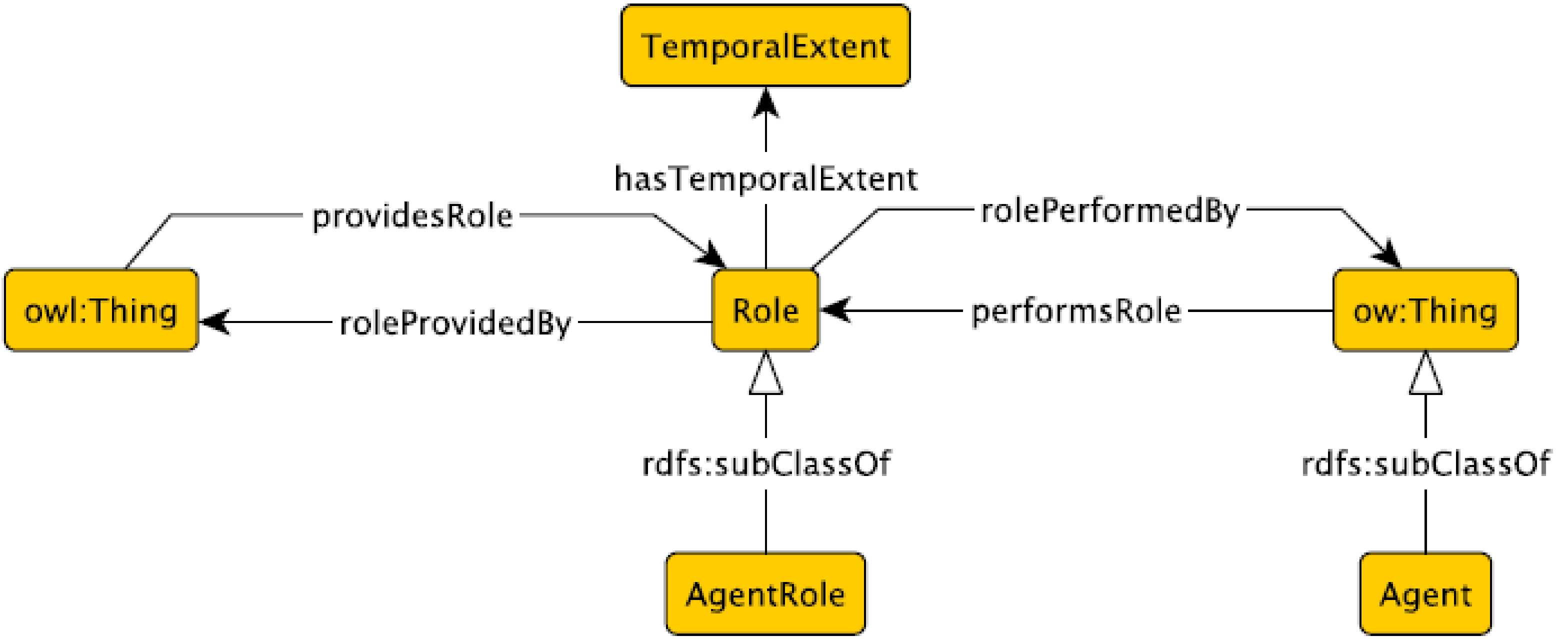


The Role Patterns



$\top \sqsubseteq \forall \mathsf{providesRole}.\mathsf{Role}$	range
$\exists \mathsf{roleProvidedBy}.\top \sqsubseteq \mathsf{Role}$	domain
$\mathsf{providesRole} \equiv \mathsf{roleProvidedBy}^{-}$	inverse
$\top \sqsubseteq \forall \mathsf{performsRole}.\mathsf{Role}$	range
$\exists \mathsf{rolePerformedBy}.\top \sqsubseteq \mathsf{Role}$	domain
$\mathsf{rolePerformedBy} \equiv \mathsf{performsRole}^{-}$	inverse
$\mathsf{Role} \sqsubseteq \exists \mathsf{hasTemporalExtent}.\mathsf{TemporalExtent}$	existential
$\sqcap \forall \mathsf{hasTemporalExtent}.\mathsf{TemporalExtent}$	scoped range
$\sqcap (\leq 1\ \mathsf{roleProvidedBy}.\top)$	range cardinality
$\sqcap (\leq 1\ \mathsf{rolePerformedBy}.\top)$	range cardinality
$\mathsf{Role} \sqsubseteq \exists \mathsf{roleProvidedBy}.\top \sqcap \exists \mathsf{rolePerformedBy}.\top$	existentials
$\mathtt{DisjointClasses}(\mathsf{Role}, \mathsf{TemporalExtent})$	disjointness

The Agent Role Pattern



Axioms: all previous plus the following.

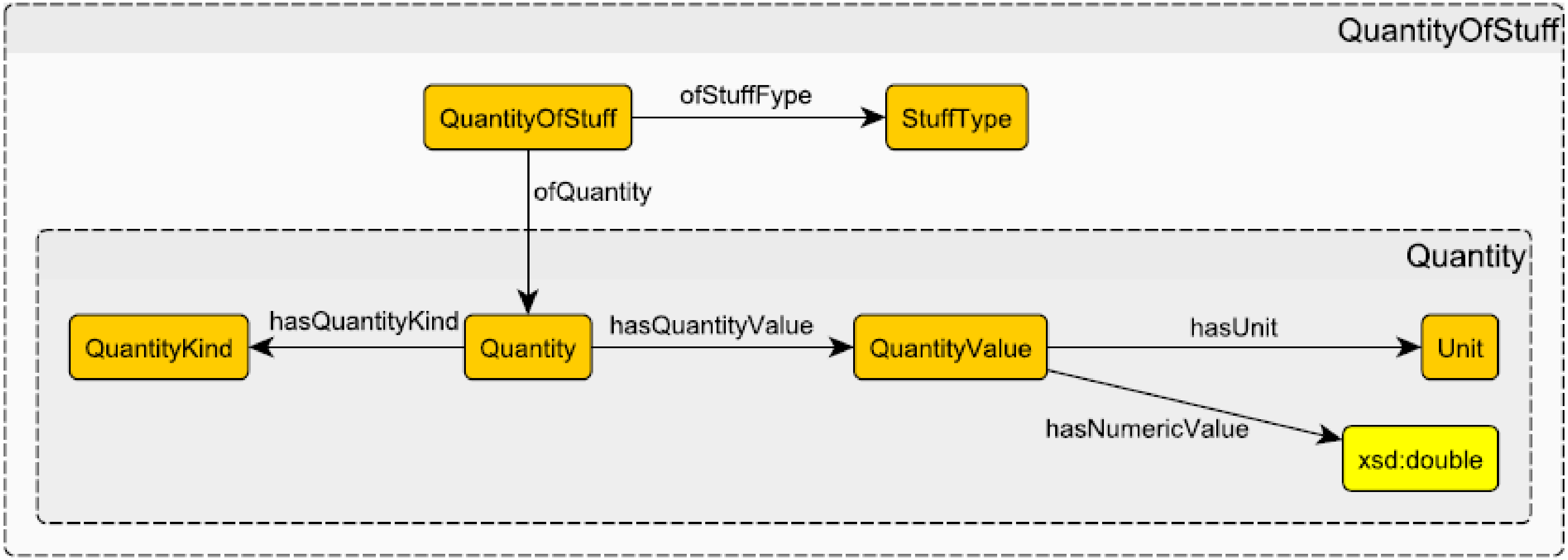
$$\mathsf{AgentRole} \sqsubseteq \mathsf{Role}$$

$$\exists \mathsf{rolePerformedBy}.\mathsf{Agent} \sqsubseteq \mathsf{AgentRole}$$

$$\mathsf{AgentRole} \sqsubseteq \forall \mathsf{rolePerformedBy}.\mathsf{Agent}$$

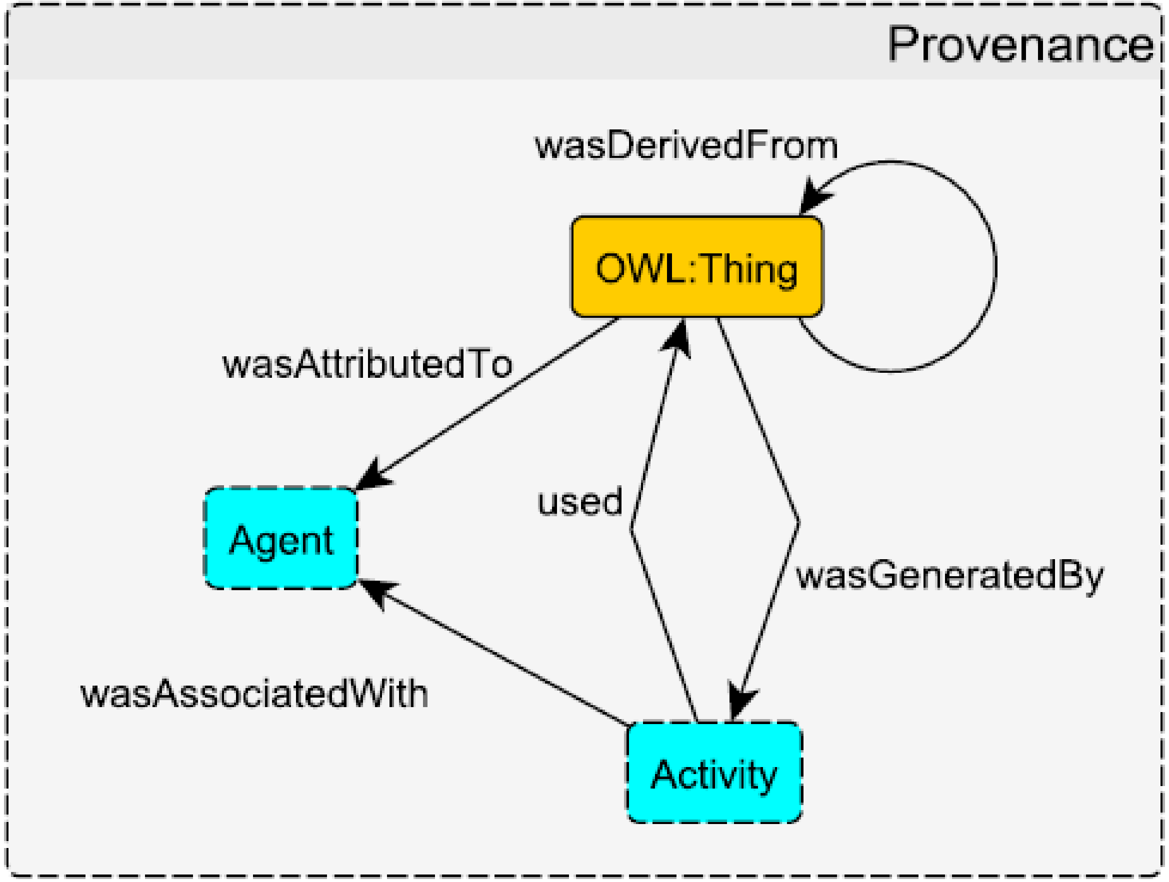
Quantities and Units

Borrowed from the QUDT ontology

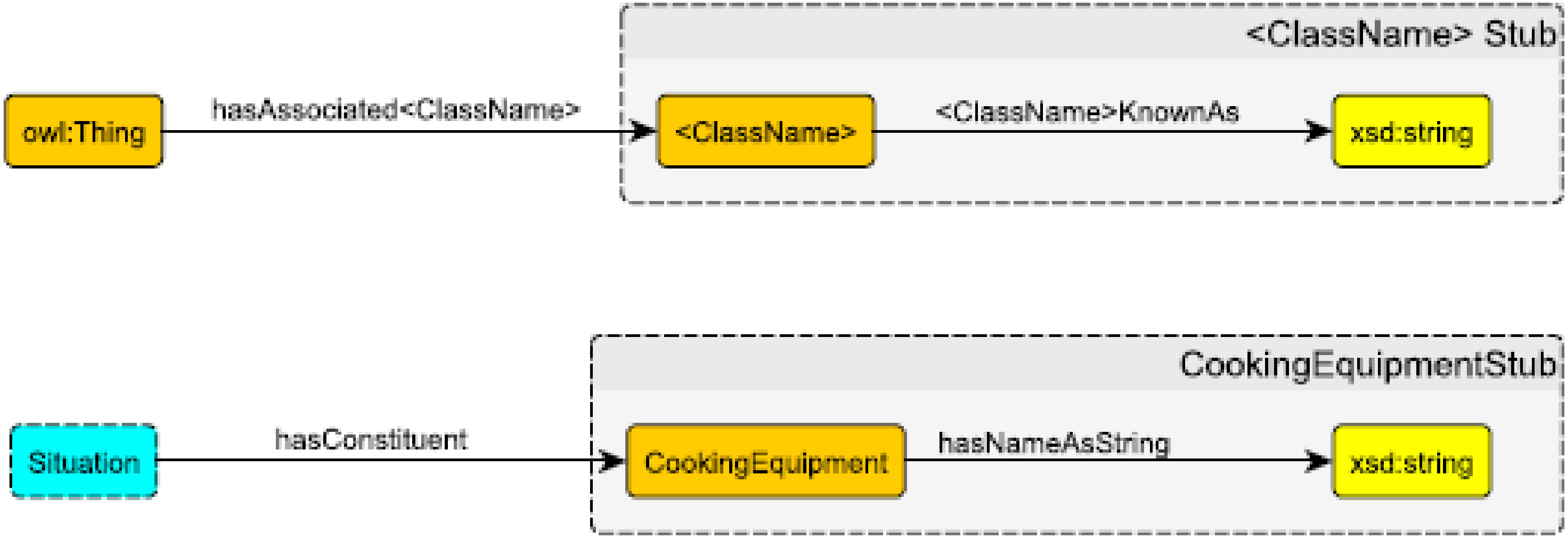


Provenance

Borrowed from PROV-O



The Stub Metapattern



Bottom: The CookingEquipmentStub derived from it.

Recipes Example

Recpies Example

Written version of this part

Pascal Hitzler, Adila Krisnadhi

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

Technical Report, DaSe Lab, Department of Computer Science and Engineering, Wright State University, Dayton, OH, August 2018.

22 pages

http://daselab.cs.wright.edu/pub2/mom-recipes-example.pdf

Modeling process – steps

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 (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Problem setting

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.

Modeling process – steps

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 (if a suitable pattern is available). Many can remain “stubs” if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Competency Questions

- From available data and from application use cases, devise competency questions, i.e. questions which should be convertible into queries, which in turn should be answerable using the data.

Gluten-free low-calorie desserts.

How do I make a low-carb pot roast?

How do I make a Chili without beans?

Sweet breakfast under 100 calories.

Breakfast dishes which can be prepared quickly with 2 potatoes, an egg, and some our.

How do I prepare Chicken thighs in a slow cooker?

A simple recipe with pork shoulder and spring onions.

A side prepared using Brussels sprouts, bacon, and chestnuts.

Modeling process – steps

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 (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Key notions

- Use the competency questions.
- Possibly also query domain experts as to the main notions for the application domain.
- E.g. for the recipes scenario, these would include
  - Recipe
  - Food
  - Time
  - Equipment
  - Classification of food (e.g., as a side)
  - Difficulty level
  - Nutritional information
  - Provenance

Key notions

- **Then prioritize which notions to model first. In this case, e.g.**

  recipe

  food

  equipment

  classification

  difficulty level

  time

  nutritional information

  provenance

Modeling process – steps

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 (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Identifying suitable patterns

- **Understand the nature of the things you are modeling.**

Recipe: Document? Sequence? Process? Plan? Description?
Food: A concrete piece of food? An abstract quantity of food?
Equipment: Do we want a complex model at this stage? No. Stub

Classification: Do we want a complex model at this stage? No. Stub

Difficulty level: Do we want a complex model at this stage? No. Stub

Time: Probably already incorporated in plan?

Nutritional information: model along some existing standard?

Provenance: just that!

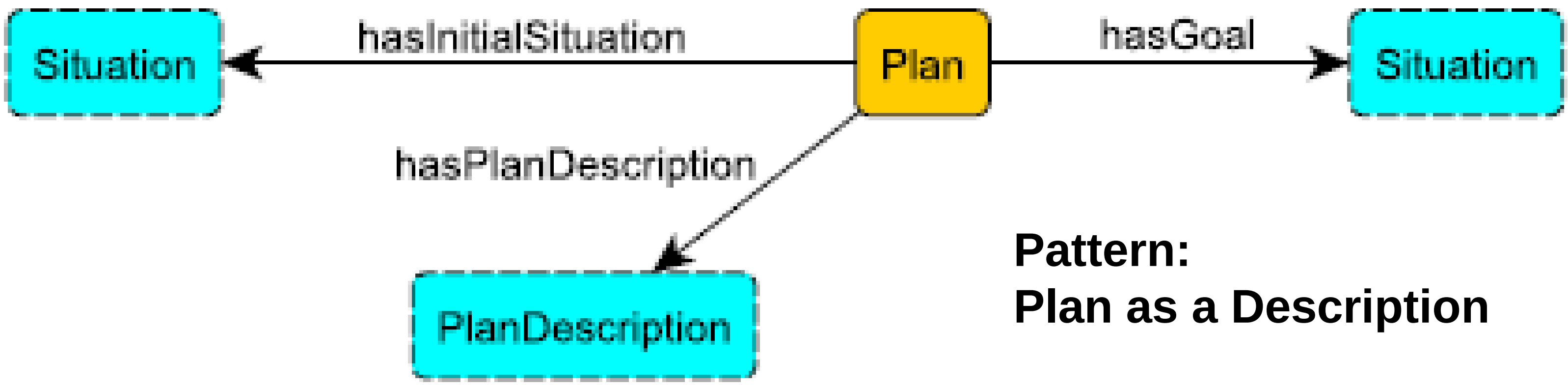
Modeling process – steps

DaSe Lab

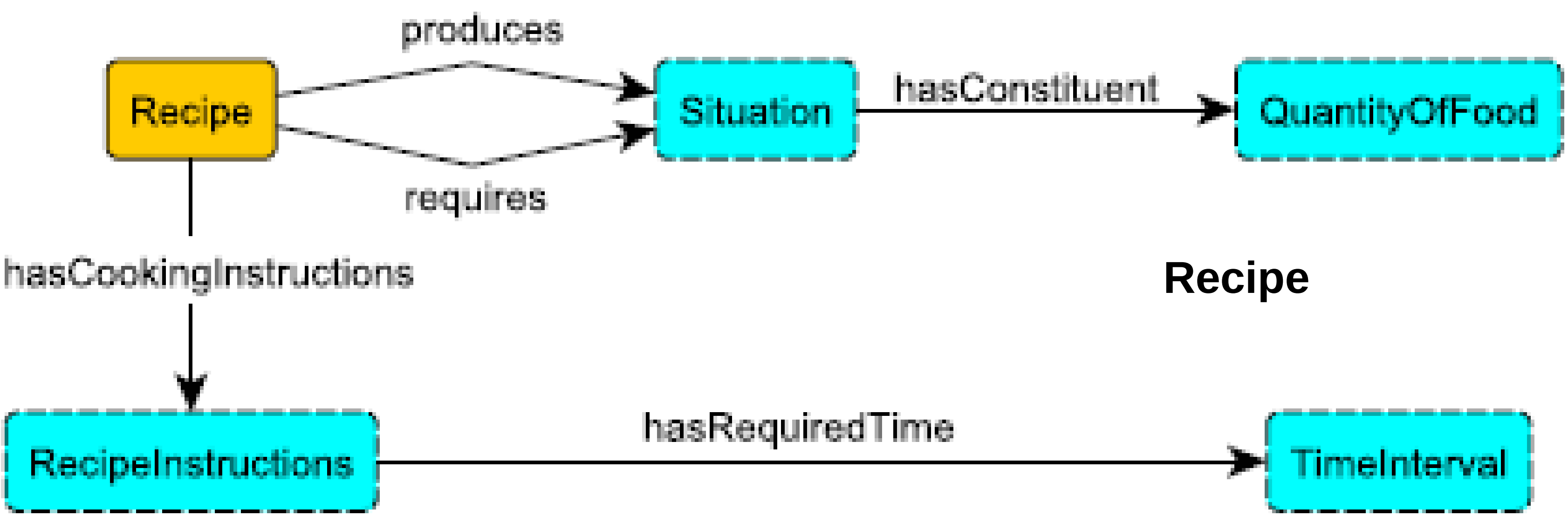
1. Define use case or scope of use cases
2. Make competency questions while looking at possible data sources and scoping the problem, i.e., decide on what should be modeled now, and what should be left for a possible later extension.
3. Identify key notions from the data and the use case and identify which pattern should be used for each (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Recipe

A **plan**, **a** **description**.



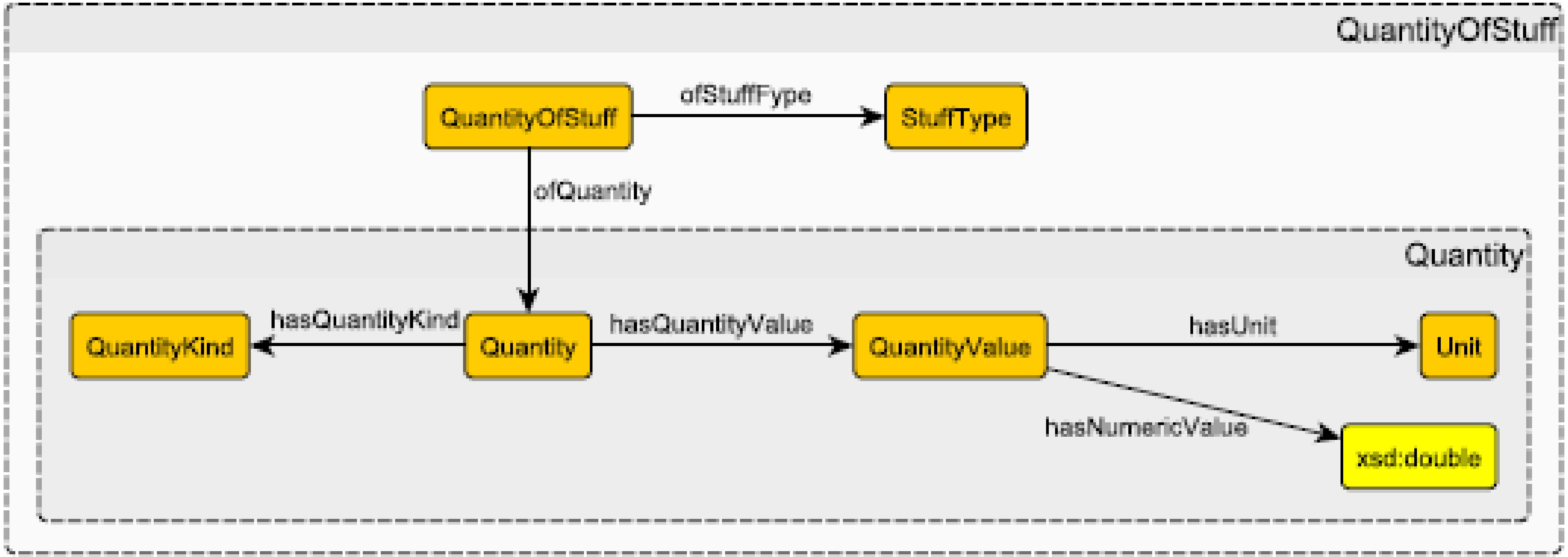
Pattern:
Plan as a Description



Recipe

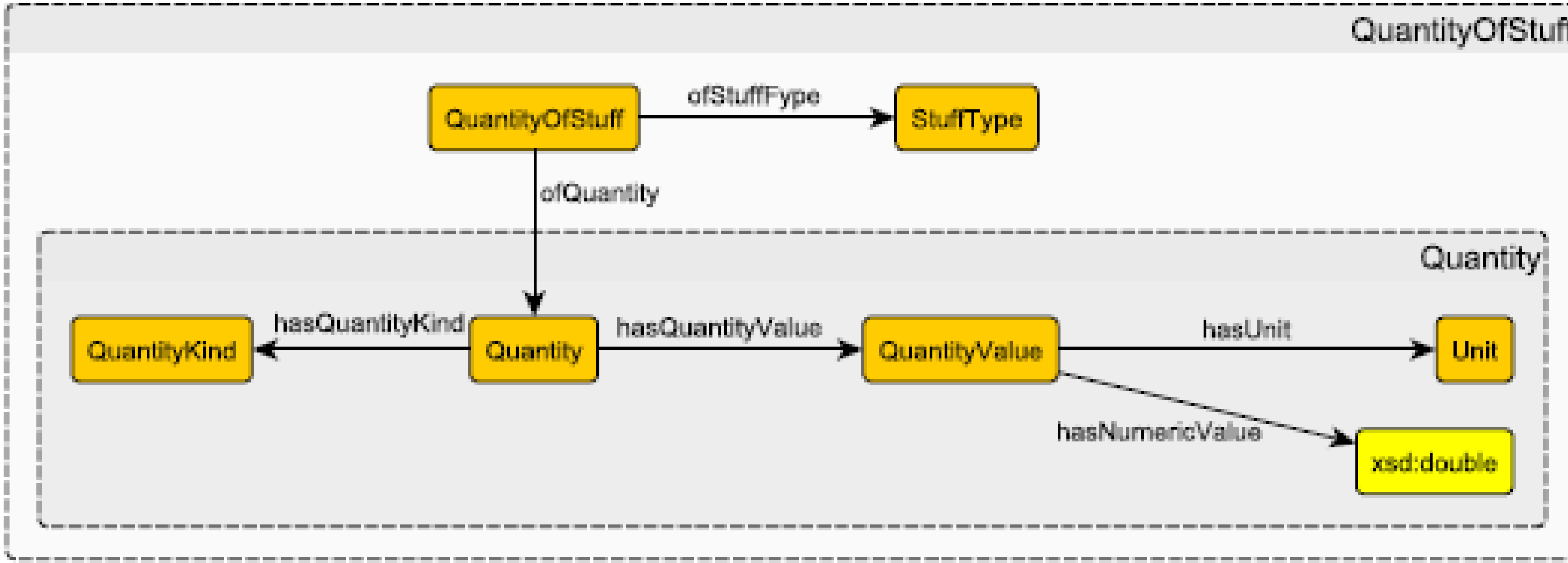
Food

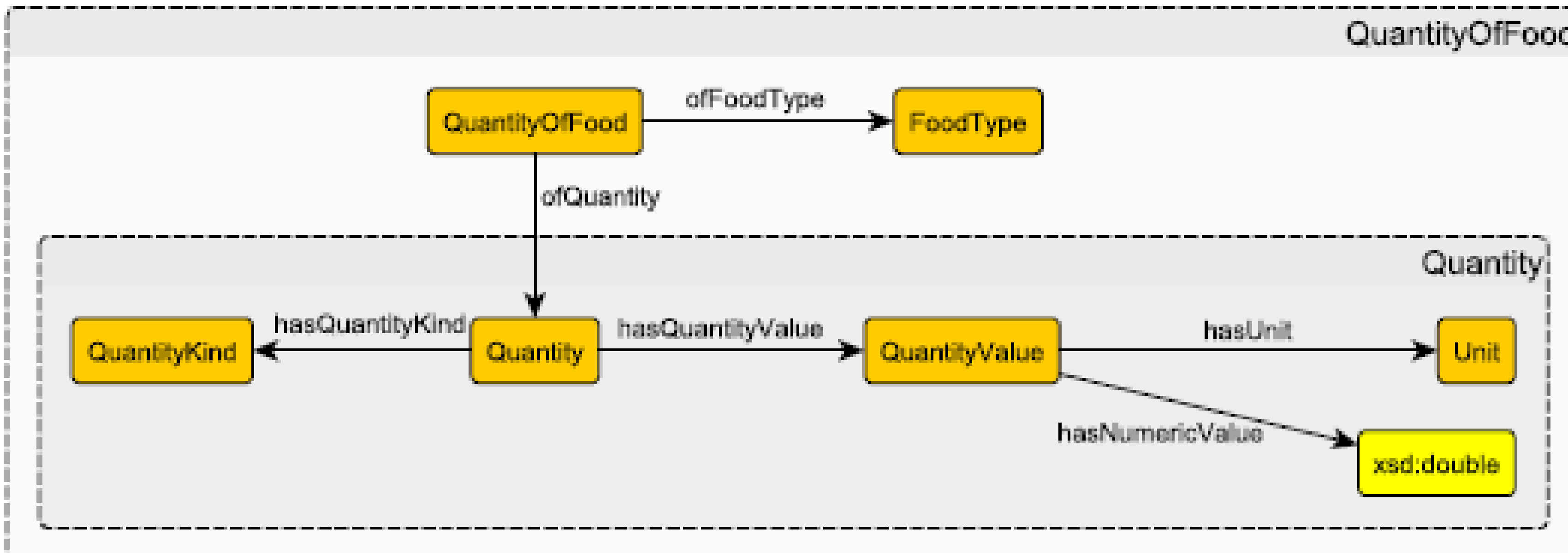
An abstract quantity of food.



Pattern:
QuantityOfStuff (with Quantity sub-pattern)

(derived from QUDT)


QuantityOfStuff
QuantityOfStuff
ofStuffType
StuffType
ofQuantity
Quantity
QuantityKind
hasQuantityKind
Quantity
hasQuantityValue
QuantityValue
hasUnit
Unit
hasNumericValue
xsd:double


QuantityOfFood
QuantityOfFood
ofFoodType
FoodType
ofQuantity
Quantity
QuantityKind
hasQuantityKind
Quantity
hasQuantityValue
QuantityValue
hasUnit
Unit
hasNumericValue
xsd:double

Equipment

No complex model desired at this stage. We just want to use strings, i.e., use our stub meta-pattern.

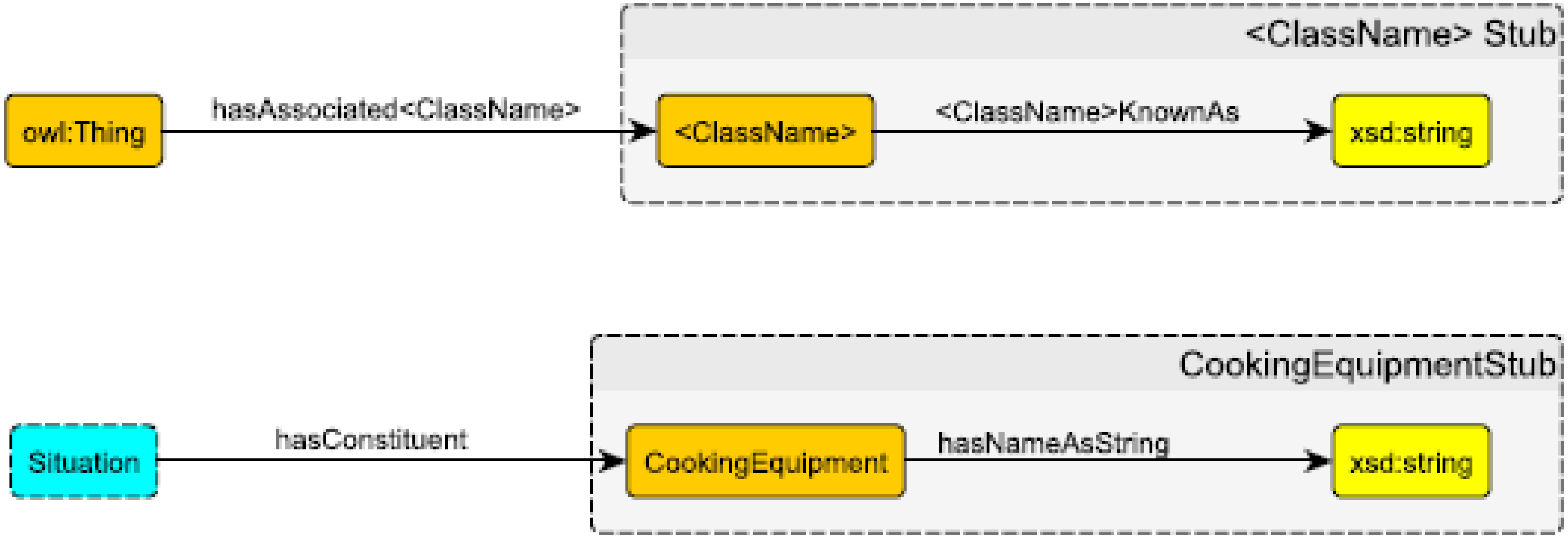
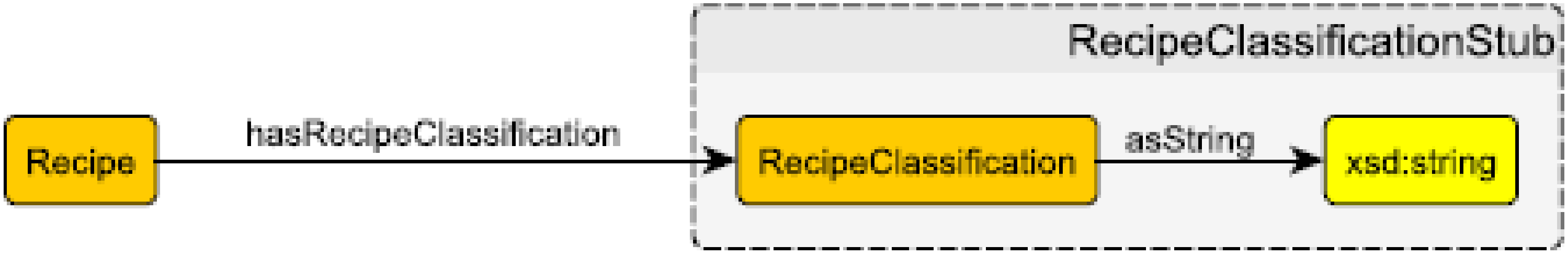


Figure 2.10: Top, the Stub (meta)pattern. Bottom, its instantiation for equipment.

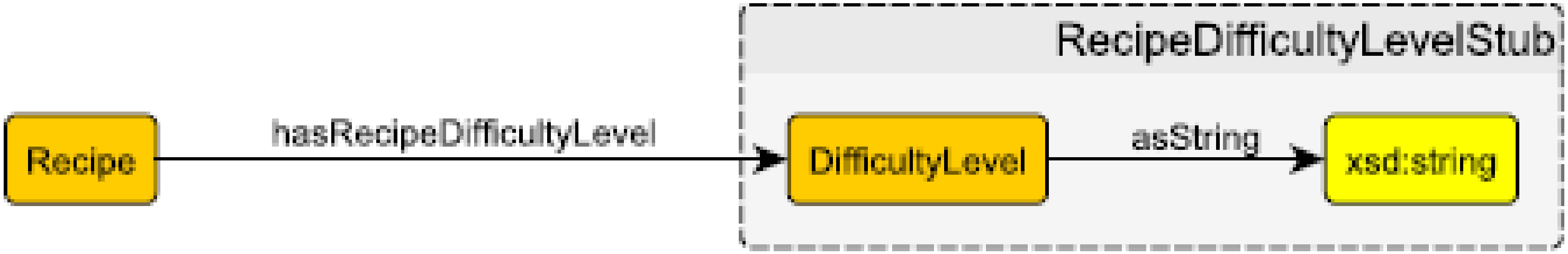
Classification (e.g., entrée)

No complex model desired at this stage. We just want to use strings, i.e., use our stub meta-pattern.



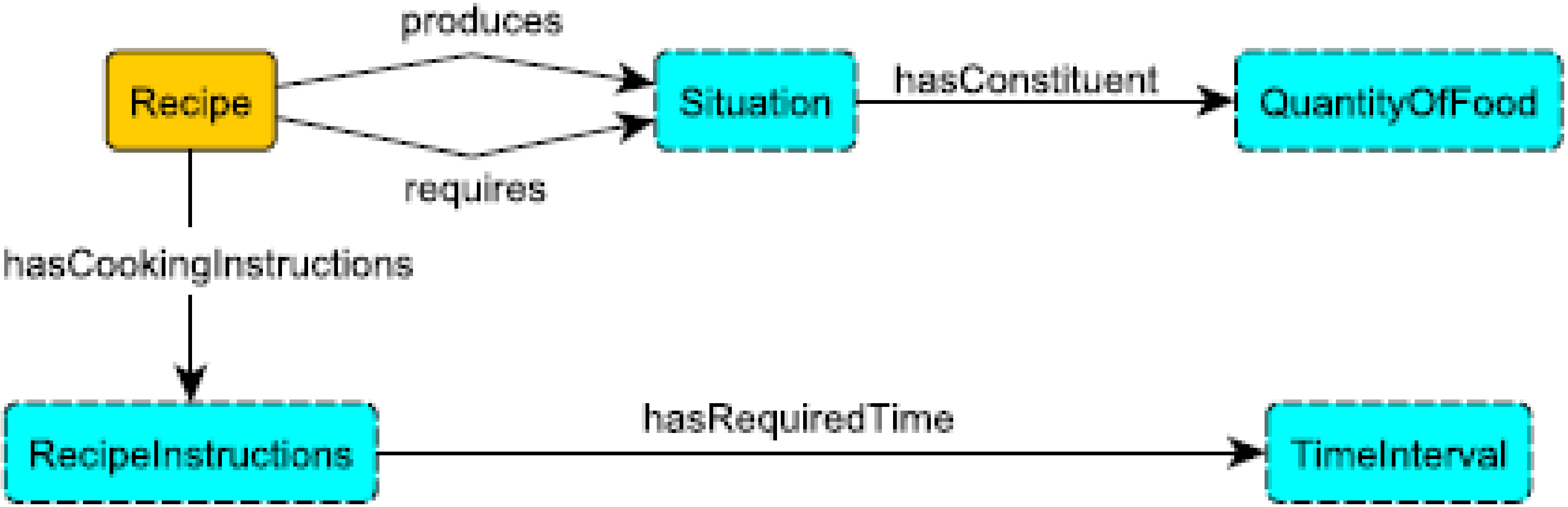
Difficulty level

No complex model desired at this stage. We just want to use strings, i.e., use our stub meta-pattern.



Time

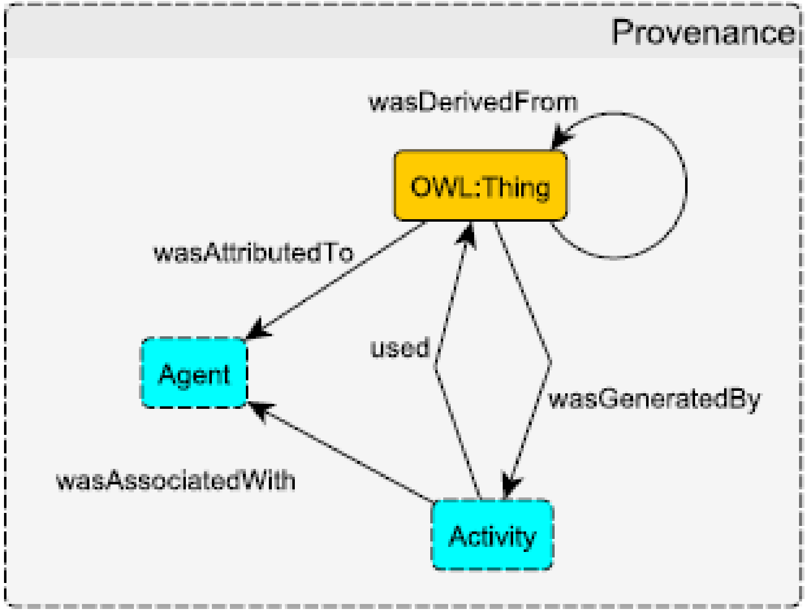
Already incorporated in plan!



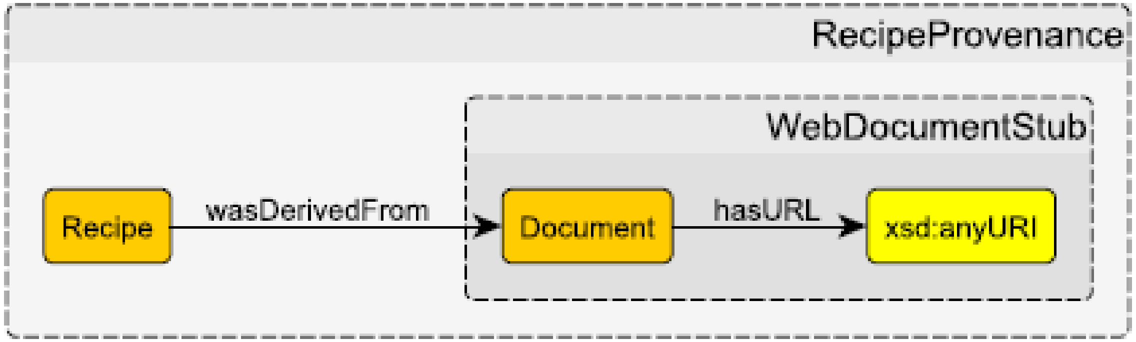
Provenance

Use an ontology design pattern based on PROV-O.

PROV-O derived Provenance pattern:



We'll use only this:



Modeling process – steps

DaSe Lab

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 (if a suitable pattern is available). Many can remain “stubs” if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Nutritional information

Model along some existing standard.

Let's use the U.S. FDA Nutritional Facts label standard.

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

Nutritional information

Model along some existing standard.

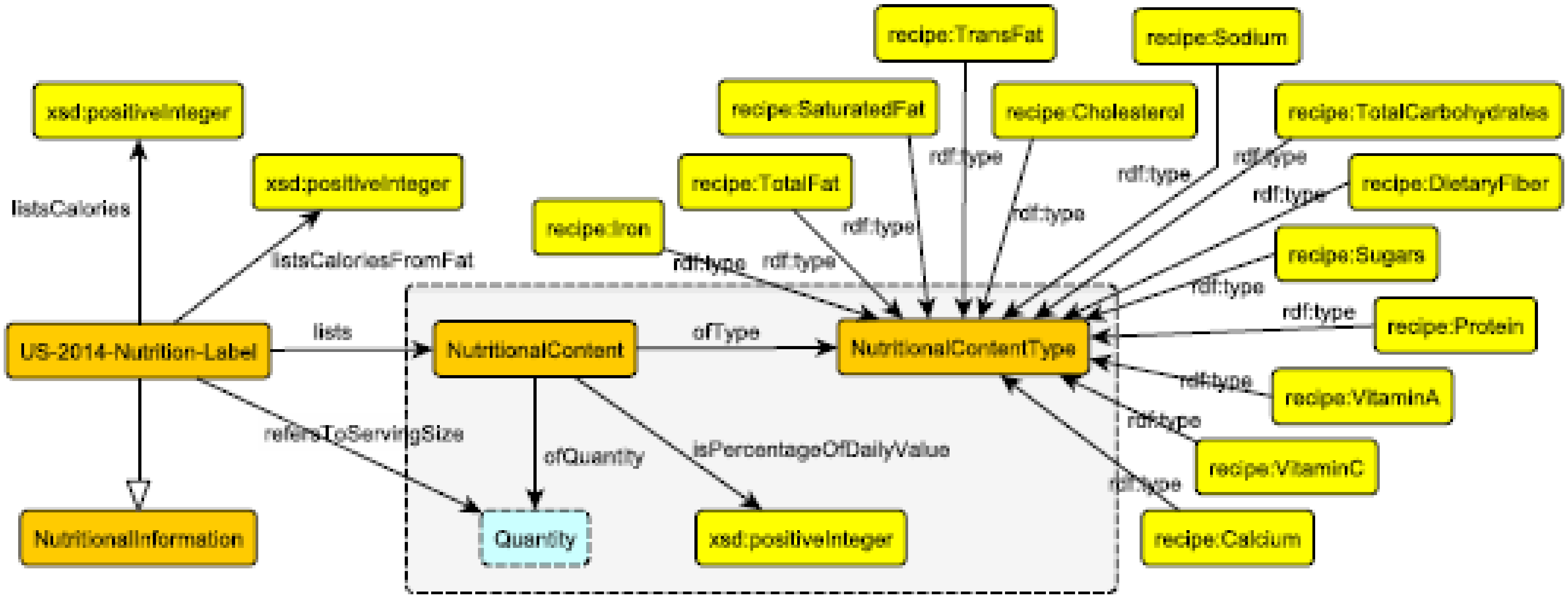


Figure 2.13: Nutritional Information module. The box indicates a modified instance of the QuantityOfStuff pattern.

Adequacy check

- Triplify sample data using the ontology.
  Does it work?

- Check if competency questions can be answered.

- Add axioms as appropriate (the graph is only for intuition, the OWL axioms are the actual ontology).

- (there are more post-hoc details to be taken care of, but let's leave it at that)

Modeling process – steps

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 (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Axiomatization

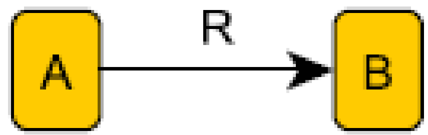
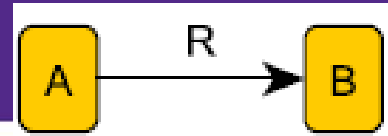


Figure 2.17: Generic node-edge-node schema diagram for explaining systematic axiomatization

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

Figure 2.18: Most common axioms which could be produced from a single edge R between nodes A and B in a schema diagram: description logic notation.

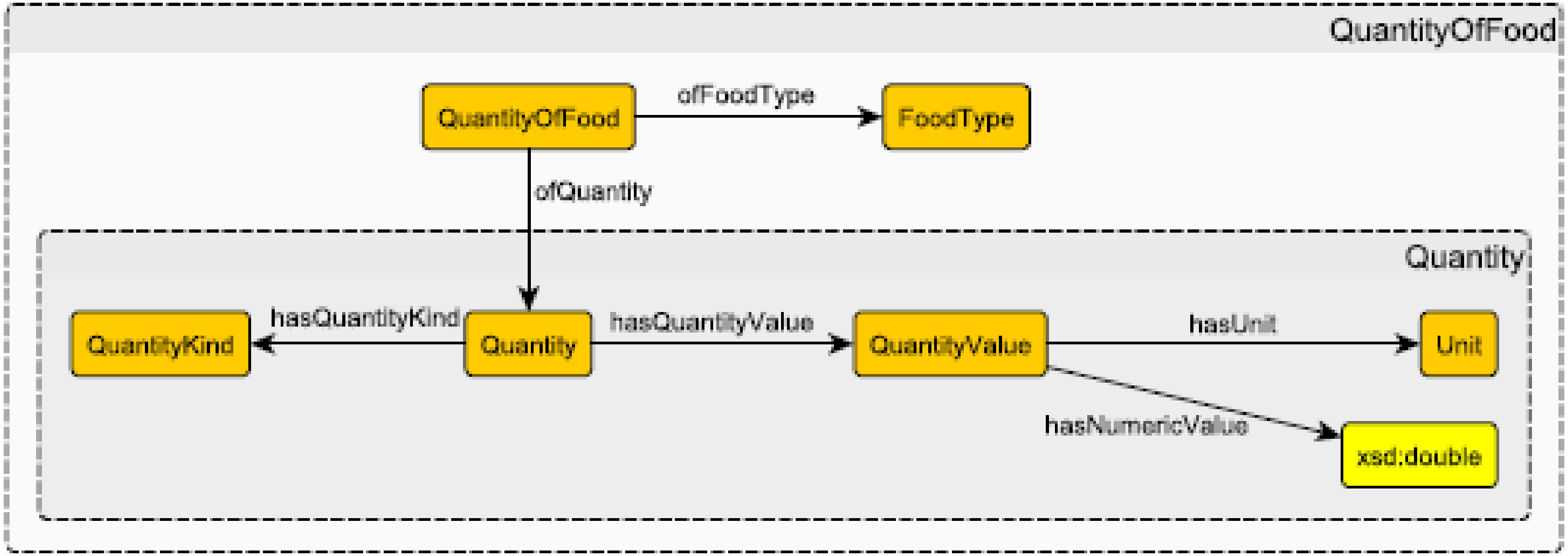
Axiomatization



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

Example Axiomatization



ofFoodType, ofQuantity: scoped range, existential

hasQuantityKind, hasQuantityValue: scoped domain, scoped range, existential, inverse existential, scoped qualified functionality

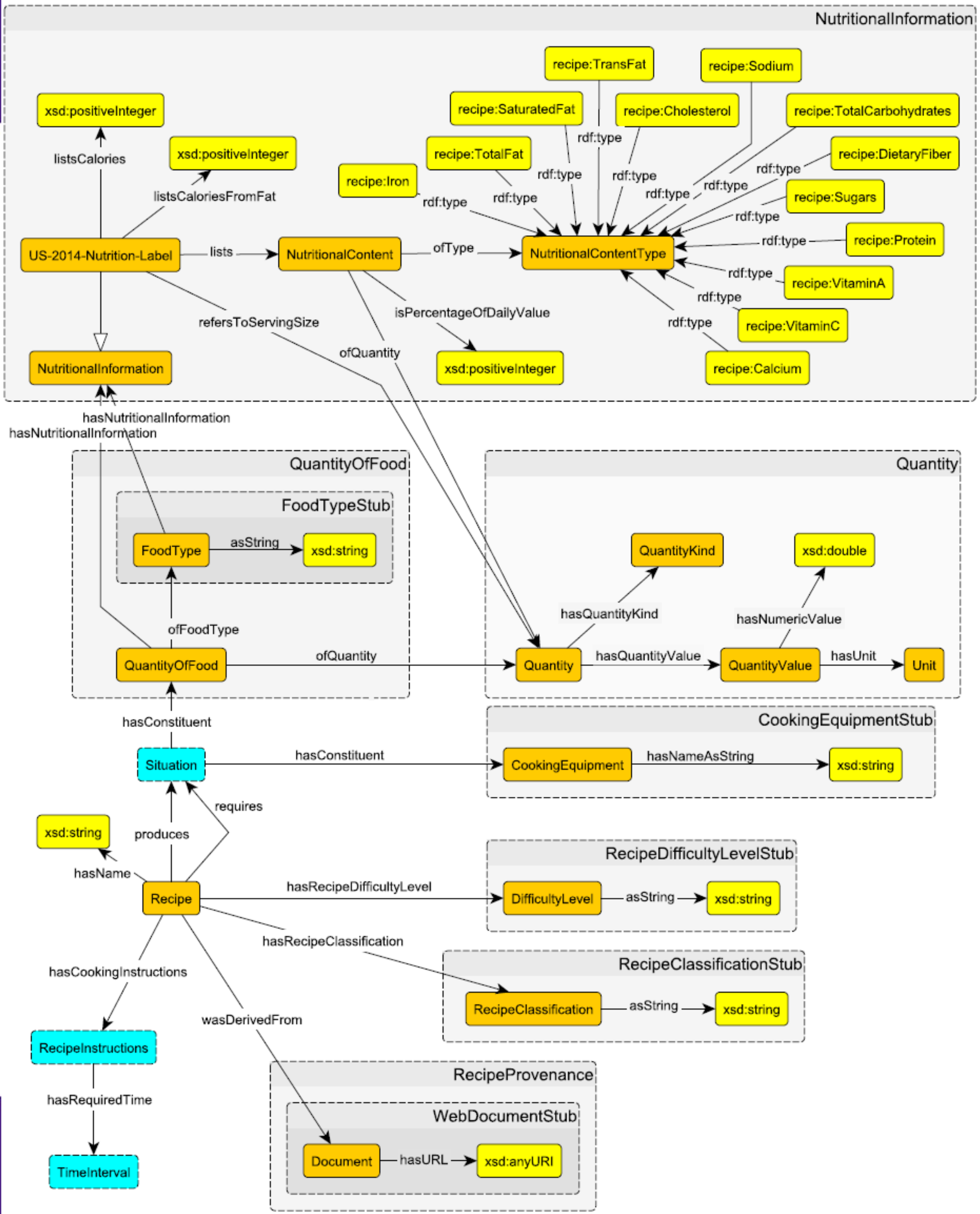
hasUnit: scoped range, existential, scoped qualified functionality

hasNumericValue: scoped range, existential, functionality

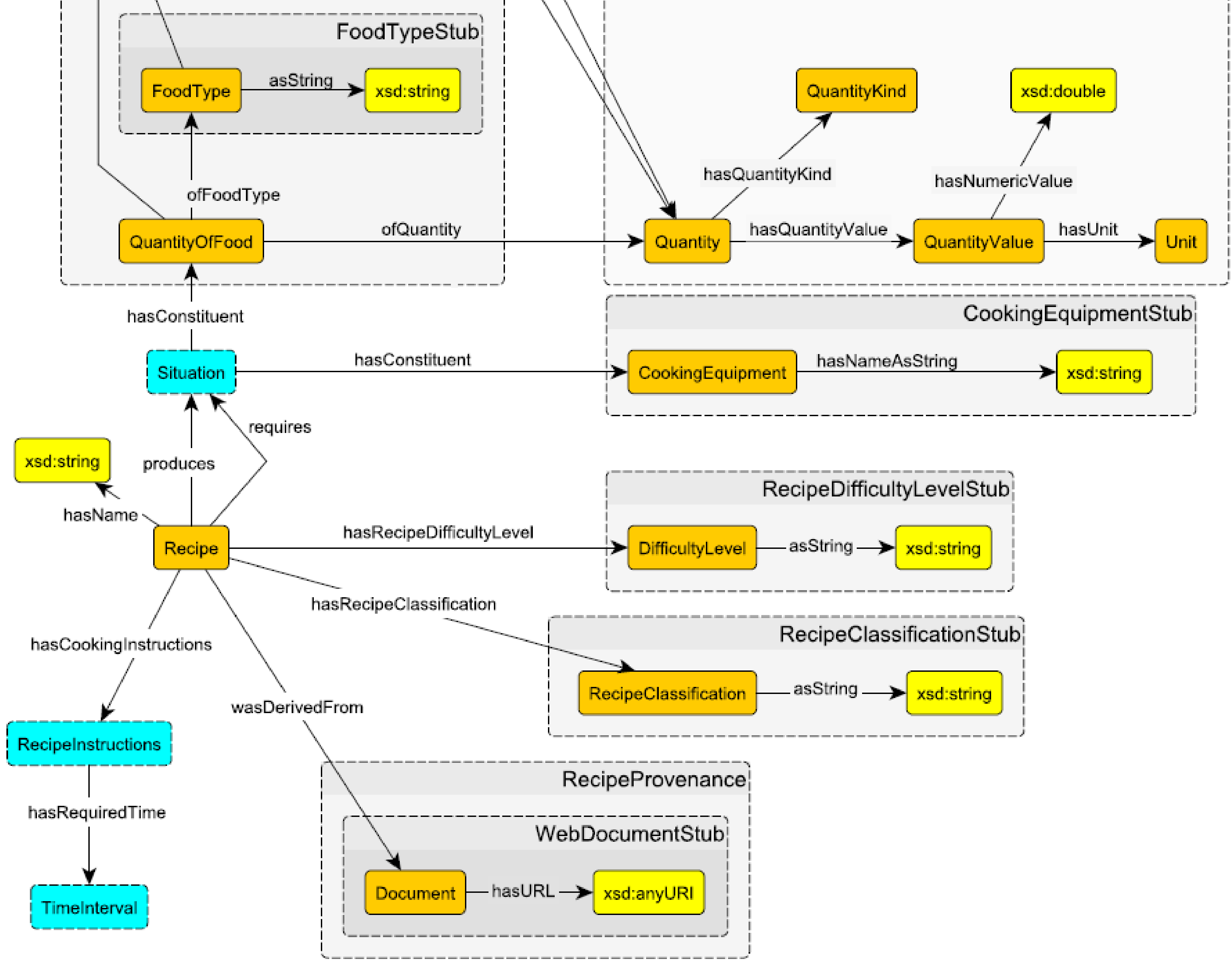
Mutually disjoint: QuantityOfFood, FoodType, QuantityKind, Quantity, QuantityValue, Unit

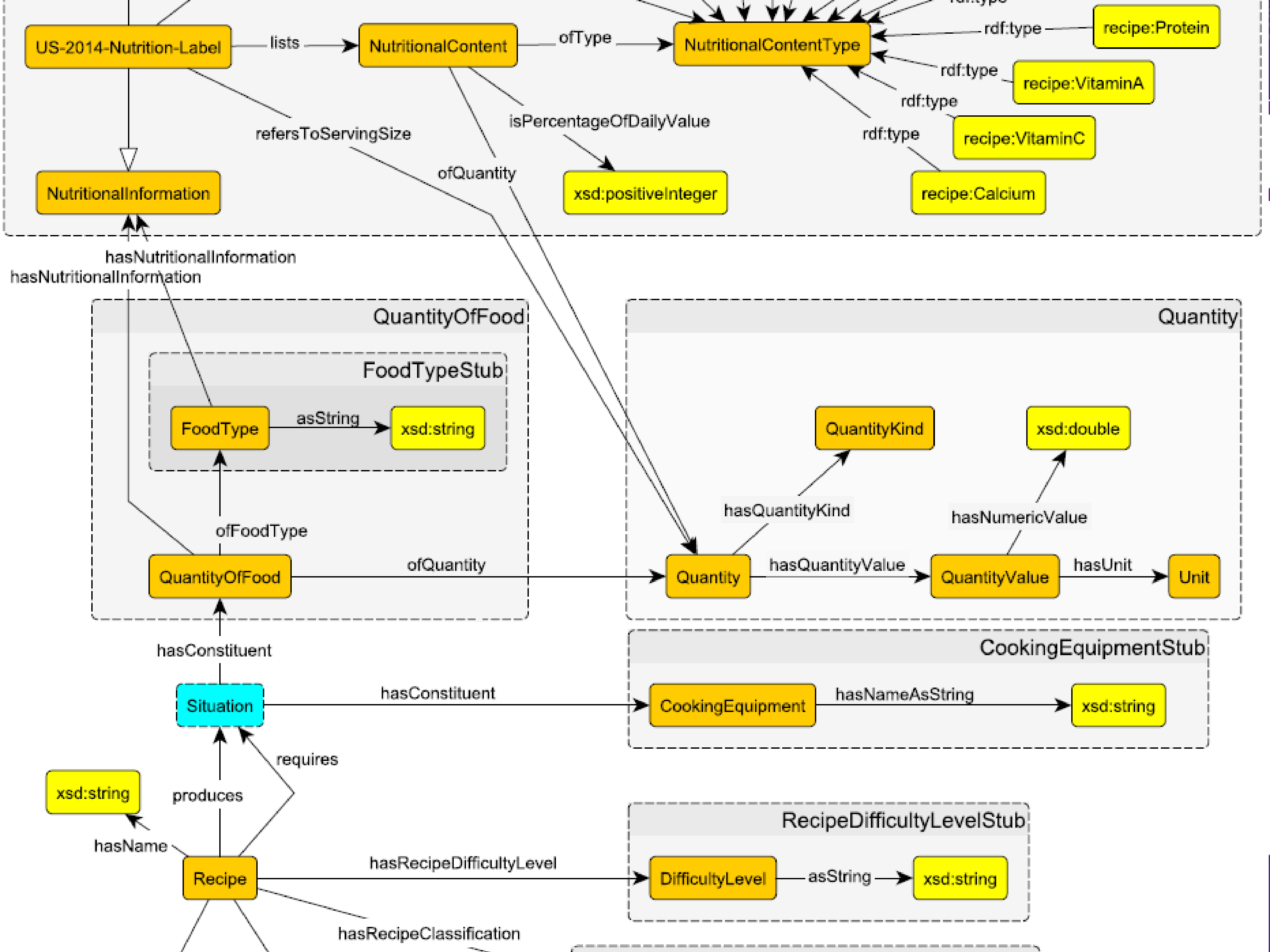
Modeling process – steps

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 (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.



NutritionalInformation
recipe:TransFat
recipe:Sodium
xsd:positiveInteger
recipe:SaturatedFat
recipe:Cholesterol
recipe:TotalCarbohydrates
xsd:positiveInteger
recipe:TotalFat
recipe:DietaryFiber
listsCalories
recipe:Iron
rdf:type
listsCaloriesFromFat
recipe:Sugars
US-2014-Nutrition-Label
lists
NutritionalContent
ofType
NutritionalContentType
recipe:Protein
recipe:VitaminA
refersToServingSize
isPercentageOfDailyValue
recipe:VitaminC
ofQuantity
NutritionalInformation
xsd:positiveInteger
recipe:Calcium
hasNutritionalInformation
hasNutritionalInformation
QuantityOfFood
Quantity
FoodTypeStub
FoodType
asString
xsd:string
QuantityKind
xsd:double
hasQuantityKind
hasNumericValue
ofFoodType
QuantityOfFood
ofQuantity
Quantity
hasQuantityValue
QuantityValue
hasUnit
Unit
hasConstituent
CookingEquipmentStub
Situation
hasConstituent
CookingEquipment
hasNameAsString
xsd:string
requires
xsd:string
produces
RecipeDifficultyLevelStub
hasName
Recipe
hasRecipeDifficultyLevel
DifficultyLevel
asString
xsd:string
hasRecipeClassification
RecipeClassificationStub
hasCookingInstructions
RecipeClassification
asString
xsd:string
wasDerivedFrom
RecipeInstructions
RecipeProvenance
hasRequiredTime
WebDocumentStub
Document
hasURL
xsd:anyURI
TimeInterval


FoodTypeStub
FoodType
asString
xsd:string
ofFoodType
QuantityOfFood
ofQuantity
QuantityKind
xsd:double
hasQuantityKind
hasNumericValue
Quantity
hasQuantityValue
QuantityValue
hasUnit
Unit
hasConstituent
CookingEquipmentStub
Situation
hasConstituent
CookingEquipment
hasNameAsString
xsd:string
requires
xsd:string
produces
hasName
RecipeDifficultyLevelStub
Recipe
hasRecipeDifficultyLevel
DifficultyLevel
asString
xsd:string
hasRecipeClassification
RecipeClassificationStub
hasCookingInstructions
RecipeClassification
asString
xsd:string
wasDerivedFrom
RecipeInstructions
RecipeProvenance
hasRequiredTime
WebDocumentStub
Document
hasURL
xsd:anyURI
TimeInterval



US-2014-Nutrition-Label
lists
NutritionalContent
ofType
NutritionalContentType
rdf:type
recipe:Protein
rdf:type
recipe:VitaminA
rdf:type
recipe:VitaminC
rdf:type
recipe:Calcium
isPercentageOfDailyValue
xsd:positiveInteger
refersToServingSize
ofQuantity
NutritionalInformation
hasNutritionalInformation
hasNutritionalInformation
QuantityOfFood
FoodTypeStub
FoodType
asString
xsd:string
ofFoodType
QuantityOfFood
ofQuantity
Quantity
QuantityKind
xsd:double
hasQuantityKind
hasNumericValue
Quantity
hasQuantityValue
QuantityValue
hasUnit
Unit
hasConstituent
CookingEquipmentStub
Situation
hasConstituent
CookingEquipment
hasNameAsString
xsd:string
requires
xsd:string
produces
hasName
RecipeDifficultyLevelStub
Recipe
hasRecipeDifficultyLevel
DifficultyLevel
asString
xsd:string
hasRecipeClassification

Modeling process – steps

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 (if a suitable pattern is available). Many can remain "stubs" if detailed modeling is not yet necessary.
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.
5. Add axioms for each module, informed by the pattern axioms.
6. Put the modules together and add axioms which involve several modules.
7. 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.
8. Create OWL files.

Tools

- **We are currently developing a set of compatible tools, as Protégé plug-ins.**
  - **See http://comodide.com/**
- **We are also developing ODP libraries.**
  - **See https://daselab.cs.ksu.edu/content/modl-modular-ontology-design-library**

Thanks!

References

- 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.
- 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.
- 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, Oscar 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.

References

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

References

- 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, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul T. Groth, Michel Dumontier, Je Hein, Krishnaprasad Thirunarayan, and Steen Staab, editors, The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, Oc tober 11-15, 2015, Proceedings, Part II, volume 9367 of Lecture Notes in Computer Science, pages 301-309. Springer, 2015.
- 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/.

References

- 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.
- Md. Kamruzzaman Sarker, Adila Alfa Krisnadhi, and Pascal Hitzler. OWLAx: A Protege 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. CEURWS.org, 2016.

References

- Cogan Shimizu, Quinn Hirt, and Pascal Hitzler. A Protege plug-in for annotating OWL ontologies with OPLa. In Aldo Gangemi, Anna Lisa Gentile, Andrea Giovanni Nuzzolese, Sebastian Rudolph, Maria Maleshkova, Heiko Paulheim, Je 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.
- Cogan Shimizu, Pascal Hitzler, and Clare Paul. Ontology design patterns for Winston's taxonomy of part-whole-relationships. In Proceedings WOP 2018.
- C. Shimizu and Hammar, K., “CoModIDE - The Comprehensive Modular Ontology IDE”, in 18th International Semantic Web Conference: Satellite Events, 2019.
- C. Shimizu, Hirt, Q., and Hitzler, P., “MODL: a Modular Ontology Design Library”, Workshop on Ontology Design and Patterns. 2019.