

# Ontology patterns in practice: Reasonable Ontology Templates (OTTR)

Martin G. Skjæveland, Leif Harald Karlsen, Daniel P. Lupp



## Improving the efficiency and quality of ontology engineering

Problem:

- Current ontology engineering methods are too low-level
- Makes ontology construction repetitive and error-prone
- Difficult to engage end-users
- Difficult to generate sustainable large ontologies
- Difficult to efficiently maintain ontologies

## Improving the efficiency and quality of ontology engineering

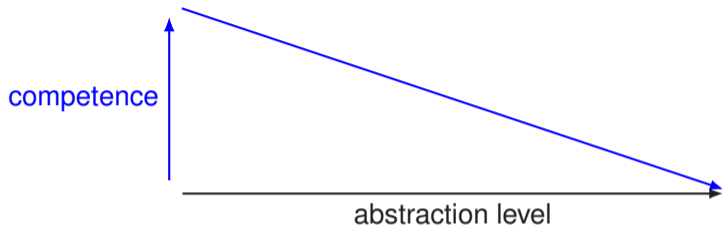
Problem:

- Current ontology engineering methods are too low-level
- Makes ontology construction repetitive and error-prone
- Difficult to engage end-users
- Difficult to generate sustainable large ontologies
- Difficult to efficiently maintain ontologies

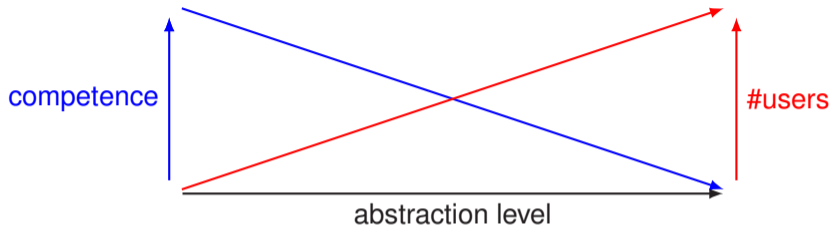
We need:

- Abstractions!
- Capture and instantiate reoccurring modelling patterns
- Formats adapted to domain experts, data managers and ontology experts

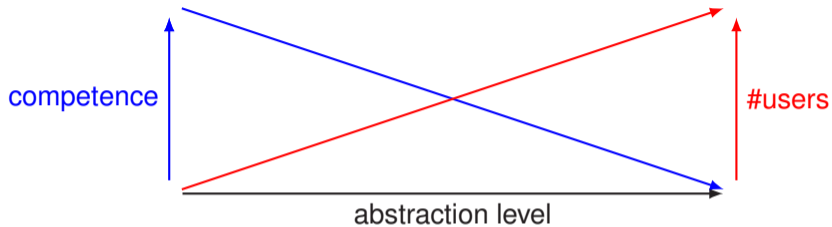
# Abstractions



# Abstractions

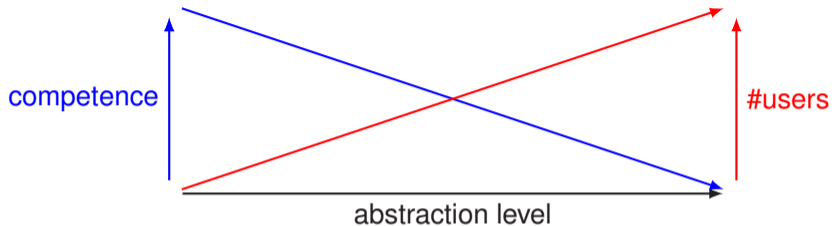


# Abstractions



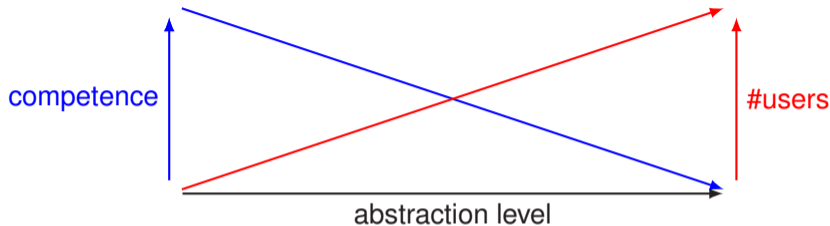
- Programming: assembly vs. high-level languages

# Abstractions



- Programming: assembly vs. high-level languages
- Java: garbage collector — I don't care!

# Abstractions



- Programming: assembly vs. high-level languages
- Java: garbage collector — I don't care!
- Ontology engineering:
  - still low-level OWL “coding”
  - impossible to not care about logic



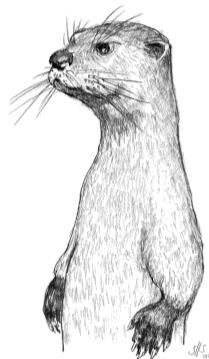
# Pizza Ontology

- Pizza ontology tutorial
- 22 NamedPizza-s
- Difficult to find all instances—  
understanding the ontology
- Error-prone to update pattern

The screenshot displays the Protege ontology editor interface. The main window title is "pizza (http://www.co-ode.org/ontologies/pizza/2.0.0) : [http://protege.stanford.edu/ontologies/pizza/pizza.owl]". The menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. The toolbar shows icons for Active Ontology, Entities, Individuals by class, and DL Query. The left pane shows the "Class hierarchy: Margherita" tree, with "Margherita" selected under "NamedPizza". The right pane shows "Annotations: Margherita" with properties like rdfs:label, skos:prefLabel, and skos:altLabel. Below that, the "Description: Margherita" pane shows class axioms such as "hasTopping only (MozzarellaTopping or TomatoTopping)" and "hasBase some PizzaBase". The bottom status bar indicates "No Reasoner set. Select a reasoner from the Reasoner menu" and "Show Inferences".

# Reasonable Ontology Templates (OTTR)

- Can be viewed as a macro language for knowledge bases
- Abstractions over RDF and OWL
  - hiding complexity and
  - providing friendly interfaces
- Build and interact with ontologies via templates
  - DRY: Do not Repeat Yourself
  - Uniform modelling
  - Completeness of input
- Leverage existing W3C stack and tools



OTTR

## Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

```
p:Hawaii rdf:type owl:Class .  
p:Hawaii rdfs:subClassOf p:Pizza .  
p:Hawaii rdfs:label "Hawaii"@en .
```

```
p:Grandiosa rdf:type owl:Class .  
p:Grandiosa rdfs:subClassOf p:Pizza .  
p:Grandiosa rdfs:label "Grandiosa"@no .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

```
p:Hawaii rdf:type owl:Class .  
p:Hawaii rdfs:subClassOf p:Pizza .  
p:Hawaii rdfs:label "Hawaii"@en .
```

```
p:Grandiosa rdf:type owl:Class .  
p:Grandiosa rdfs:subClassOf p:Pizza .  
p:Grandiosa rdfs:label "Grandiosa"@no .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

```
p:Hawaii rdf:type owl:Class .  
p:Hawaii rdfs:subClassOf p:Pizza .  
p:Hawaii rdfs:label "Hawaii"@en .
```

```
p:Grandiosa rdf:type owl:Class .  
p:Grandiosa rdfs:subClassOf p:Pizza .  
p:Grandiosa rdfs:label "Grandiosa"@no .
```

```
?name rdf:type owl:Class .  
?name rdfs:subClassOf p:Pizza .  
?name rdfs:label ?label .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .
p:Margherita rdfs:subClassOf p:Pizza .
p:Margherita rdfs:label "Margherita"@it .

p:Hawaii rdf:type owl:Class .
p:Hawaii rdfs:subClassOf p:Pizza .
p:Hawaii rdfs:label "Hawaii"@en .

p:Grandiosa rdf:type owl:Class .
p:Grandiosa rdfs:subClassOf p:Pizza .
p:Grandiosa rdfs:label "Grandiosa"@no .
```

```
PIZZA[?name, ?label] :: {
    ?name rdf:type owl:Class .
    ?name rdfs:subClassOf p:Pizza .
    ?name rdfs:label ?label .
} .
```

## Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .
p:Margherita rdfs:subClassOf p:Pizza .
p:Margherita rdfs:label "Margherita"@it .

p:Hawaii rdf:type owl:Class .
p:Hawaii rdfs:subClassOf p:Pizza .
p:Hawaii rdfs:label "Hawaii"@en .

p:Grandiosa rdf:type owl:Class .
p:Grandiosa rdfs:subClassOf p:Pizza .
p:Grandiosa rdfs:label "Grandiosa"@no .
```

```
PIZZA[?name, ?label] :: {
  TRIPLE(?name, rdf:type, owl:Class),
  TRIPLE(?name, rdfs:subClassOf, p:Pizza),
  TRIPLE(?name, rdfs:label, ?label)
} .
```

## Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .
p:Margherita rdfs:subClassOf p:Pizza .
p:Margherita rdfs:label "Margherita"@it .

p:Hawaii rdf:type owl:Class .
p:Hawaii rdfs:subClassOf p:Pizza .
p:Hawaii rdfs:label "Hawaii"@en .

p:Grandiosa rdf:type owl:Class .
p:Grandiosa rdfs:subClassOf p:Pizza .
p:Grandiosa rdfs:label "Grandiosa"@no .
```

```
PIZZA[?name, ?label] :: {
  TRIPLE(?name, rdf:type, owl:Class),
  TRIPLE(?name, rdfs:subClassOf, p:Pizza),
  TRIPLE(?name, rdfs:label, ?label)
} .
PIZZA(p:Margherita, "Margherita"@it) .
PIZZA(p:Hawaii, "Hawaii"@en) .
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```



## Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .
p:Margherita rdfs:subClassOf p:Pizza .
p:Margherita rdfs:label "Margherita"@it .

p:Hawaii rdf:type owl:Class .
p:Hawaii rdfs:subClassOf p:Pizza .
p:Hawaii rdfs:label "Hawaii"@en .

p:Grandiosa rdf:type owl:Class .
p:Grandiosa rdfs:subClassOf p:Pizza .
p:Grandiosa rdfs:label "Grandiosa"@no .
```

```
SUBCLASSOF[?sub, ?super] :: {
  TRIPLE(?sub, rdfs:subClassOf, ?super)
} .

PIZZA[?name, ?label] :: {
  TRIPLE(?name, rdf:type, owl:Class),
  TRIPLE(?name, rdfs:subClassOf, p:Pizza),
  TRIPLE(?name, rdfs:label, ?label)
} .

PIZZA(p:Margherita, "Margherita"@it) .
PIZZA(p:Hawaii, "Hawaii"@en) .
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .
p:Margherita rdfs:subClassOf p:Pizza .
p:Margherita rdfs:label "Margherita"@it .

p:Hawaii rdf:type owl:Class .
p:Hawaii rdfs:subClassOf p:Pizza .
p:Hawaii rdfs:label "Hawaii"@en .

p:Grandiosa rdf:type owl:Class .
p:Grandiosa rdfs:subClassOf p:Pizza .
p:Grandiosa rdfs:label "Grandiosa"@no .
```

```
SUBCLASSOF[?sub, ?super] :: {
  TRIPLE(?sub, rdfs:subClassOf, ?super)
} .

PIZZA[?name, ?label] :: {
  TRIPLE(?name, rdf:type, owl:Class),
  SUBCLASSOF(?name, p:Pizza),
  TRIPLE(?name, rdfs:label, ?label)
} .

PIZZA(p:Margherita, "Margherita"@it) .
PIZZA(p:Hawaii, "Hawaii"@en) .
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?name, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?name, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?name, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?name, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?name, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?name, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```



# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
  
- Substitution
- Macro expansion

```
SUBCLASSOF[?sub, ?super] :: {  
    TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
    TRIPLE(?name, rdf:type, owl:Class),  
    SUBCLASSOF(?name, p:Pizza),  
    TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
  
- Substitution
- Macro expansion

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[?name, ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?name, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
  
- Substitution
- Macro expansion

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[p:Margherita, "Margherita"@it] :: {  
  TRIPLE(p:Margherita, rdf:type, owl:Class),  
  SUBCLASSOF(p:Margherita, p:Pizza),  
  TRIPLE(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
  
- Substitution
- Macro expansion

```
SUBCLASSOF[?sub, ?super] :: {  
  TRIPLE(?sub, rdfs:subClassOf, ?super)  
} .  
PIZZA[p:Margherita, "Margherita"@it] :: {  
  TRIPLE(p:Margherita, rdf:type, owl:Class),  
  SUBCLASSOF(p:Margherita, p:Pizza),  
  TRIPLE(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
  
- Substitution
- Macro expansion

```
SUBCLASSOF[p:Margherita, p:Pizza] :: {  
  TRIPLE(p:Margherita, rdfs:subClassOf, p:Pizza)  
} .  
PIZZA[p:Margherita, "Margherita"@it] :: {  
  TRIPLE(p:Margherita, rdf:type, owl:Class),  
  SUBCLASSOF(p:Margherita, p:Pizza),  
  TRIPLE(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
  
- Substitution
- Macro expansion

```
SUBCLASSOF[p:Margherita, p:Pizza] :: {  
  TRIPLE(p:Margherita, rdfs:subClassOf, p:Pizza)  
} .  
PIZZA[p:Margherita, "Margherita"@it] :: {  
  TRIPLE(p:Margherita, rdf:type, owl:Class),  
  TRIPLE(p:Margherita, rdfs:subClassOf, p:Pizza),  
  TRIPLE(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
PIZZA(p:Margherita, "Margherita"@it) .  
PIZZA(p:Hawaii, "Hawaii"@en) .  
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

# Reasonable Ontology Templates (OTTR): Example

```
p:Margherita rdf:type owl:Class .  
p:Margherita rdfs:subClassOf p:Pizza .  
p:Margherita rdfs:label "Margherita"@it .
```

- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
  
- Substitution
- Macro expansion

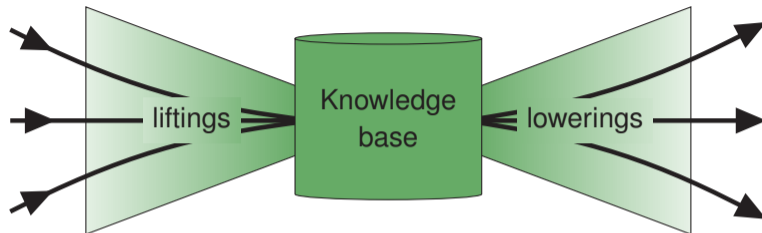
```
TRIPLE(p:Margherita, rdf:type, owl:Class),  
TRIPLE(p:Margherita, rdfs:subClassOf, p:Pizza),  
TRIPLE(p:Margherita, rdfs:label, "Margherita"@it)
```

```
PIZZA(p:Margherita, "Margherita"@it) .
```

```
PIZZA(p:Hawaii, "Hawaii"@en) .
```

```
PIZZA(p:Grandiosa, "Grandiosa"@no) .
```

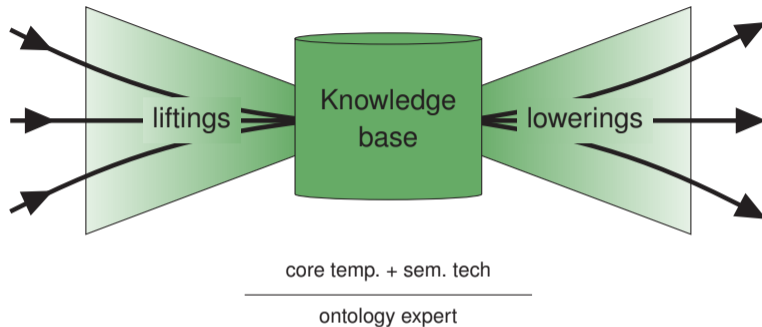
## Vision: Template-driven methodology



- “All” knowledge base interaction via templates
- “API for RDF/OWL”
- Maintain “all” interaction with KB by maintaining templates

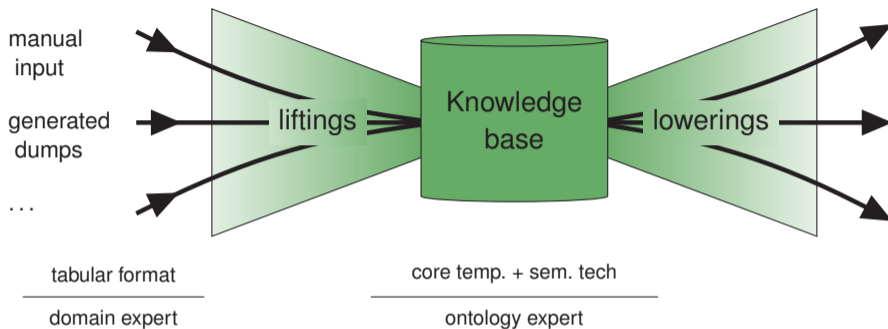


# Vision: Template-driven methodology



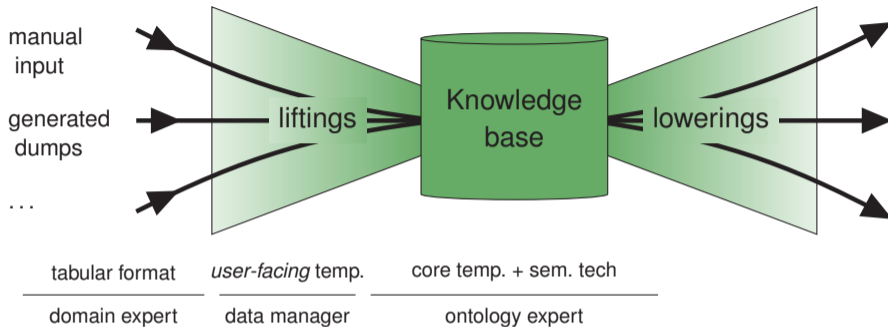
- “All” knowledge base interaction via templates
- “API for RDF/OWL”
- Maintain “all” interaction with KB by maintaining templates

# Vision: Template-driven methodology



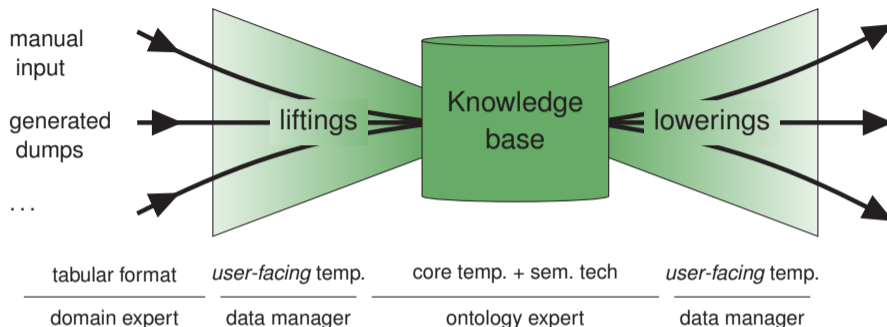
- “All” knowledge base interaction via templates
- “API for RDF/OWL”
- Maintain “all” interaction with KB by maintaining templates

# Vision: Template-driven methodology



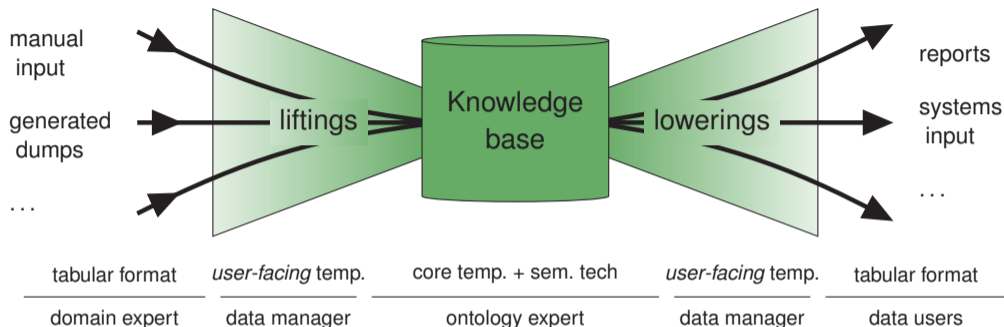
- “All” knowledge base interaction via templates
- “API for RDF/OWL”
- Maintain “all” interaction with KB by maintaining templates

# Vision: Template-driven methodology



- “All” knowledge base interaction via templates
- “API for RDF/OWL”
- Maintain “all” interaction with KB by maintaining templates

# Vision: Template-driven methodology



- "All" knowledge base interaction via templates
- "API for RDF/OWL"
- Maintain "all" interaction with KB by maintaining templates

stOTTR: Easy to read and write

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ] :: {  
  SUBCLASSOF(?Name, :NamedPizza),  
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),  
    OBJECTUNIONOF(_:b1, ?Toppings),  
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++?Toppings)  
} .
```

```
NAMEDPIZZA(Margherita, Italy, (Tomato, Mozzarella)) .
```

```
NAMEDPIZZA(Grandiosa, ottr:none, (Tomato, Jarlsberg, Ham, SweetPepper)) .
```

## wOTTR: OWL vocabulary for semantic web use

```
pizza:NamedPizza a ottr:Template ;
```

```
ottr:parameters (
  [ ottr:type owl:Class; ottr:variable _:pizza ]
  [ ottr:type owl:NamedIndividual; ottr:variable _:country; ottr:modifier ottr:optional ]
  [ ottr:type ( rdf:List owl:Class ); ottr:variable _:toppings; ] ) ;

ottr:pattern
  [ ottr:of ax:SubClassOf ; ottr:values ( _:pizza p:NamedPizza ) ] ,
  [ ottr:of ax:SubObjectHasValue ; ottr:values ( _:pizza p:hasCountryOfOrigin _:country ) ] ,
  [ ottr:of ax:SubObjectAllValuesFrom ; ottr:values ( _:pizza p:hasTopping _:alltoppings ) ] ,
  [ ottr:of rstr:ObjectUnionOf ; ottr:values ( _:alltoppings _:toppings ) ] ,
  [ ottr:of ax:SubObjectSomeValuesFrom ; ottr:modifier ottr:cross ;
    ottr:arguments ( [ ottr:value _:pizza ]
                      [ ottr:value p:hasTopping ]
                      [ ottr:value _:toppings; ottr:modifier ottr:listExpand ] ) ] .
```

# OTTR Serialisations

tabOTTR: tabular format for bulk template instances

9			
10	#OTTR	template	<a href="http://draft.ottr.xyz/pizza/NamedPizza">http://draft.ottr.xyz/pizza/NamedPizza</a>
11	Pizza	Country	Toppings
12	1	2	3
13	iri	iri	iri+
14	p:Veneziana	p:Italy	p:OnionTopping   p:MozzarellaTopping   p:CaperTopping   p:OliveTopping   p:
15	p:American	p:America	p:JalapenoPepperTopping   p:MozzarellaTopping   p:HotGreenPepperTopping
16	p:Margherita		p:MozzarellaTopping   p:TomatoTopping
17	p:FourSeasons		p:TomatoTopping   p:PeperoniSausageTopping   p:CaperTopping   p:Mushroo
18	p:Fiorentina		p:TomatoTopping   p:GarlicTopping   p:ParmesanTopping   p:MozzarellaTopp
19	p:PrinceCarlo		p:LeekTopping   p:RosemaryTopping   p:MozzarellaTopping   p:TomatoToppir
20	p:LaReine		p:MushroomTopping   p:HamTopping   p:MozzarellaTopping   p:OliveTopping
21	p:American	p:America	p:TomatoTopping   p:MozzarellaTopping   p:PeperoniSausageTopping
22	p:Caprina		p:MozzarellaTopping   p:TomatoTopping   p:GoatsCheeseTopping   p:Sundrie
23	p:PolloAdAstra		p:TomatoTopping   p:RedOnionTopping   p:MozzarellaTopping   p:SweetPepp



# OTTR templates

Margherita  $\sqsubseteq$  NamedPizza

Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}

Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)

Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato

Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza  
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}  
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]  
:: {  
  SUBCLASSOF(?Name, :NamedPizza),  
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),  
    OBJECTUNIONOF(_:b1, ?Toppings),  
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)  
}.
```

```
?Name  $\sqsubseteq$  :NamedPizza  
?Name  $\sqsubseteq$   $\exists$  hasC0.{?Country}  
  
?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)  
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin. {Italy}
Margherita  $\sqsubseteq$   $\forall$  hasTopping. (Tomato  $\sqcup$  Mozzarella)
Margherita  $\sqsubseteq$   $\exists$  hasTopping. Tomato
Margherita  $\sqsubseteq$   $\exists$  hasTopping. Mozzarella
```

template name

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]
:: {
  SUBCLASSOF(?Name, :NamedPizza),
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),
  OBJECTUNIONOF(_:b1, ?Toppings),
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)
}.
```

```
?Name  $\sqsubseteq$  :NamedPizza
?Name  $\sqsubseteq$   $\exists$  hasCO. {?Country}

?Name  $\sqsubseteq$   $\forall$  hasT. ( $\sqcup$ ?Toppings)
?Name  $\sqsubseteq$   $\exists$  hasT. ?Toppings ...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza  
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}  
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

template name

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]  
:: {  
  SUBCLASSOF(?Name, :NamedPizza),  
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),  
  OBJECTUNIONOF(_:b1, ?Toppings),  
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)  
}.
```

```
?Name  $\sqsubseteq$  :NamedPizza  
?Name  $\sqsubseteq$   $\exists$  hasC0.{?Country}  
  
?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)  
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

template name      type      parameter name

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]
:: {
  SUBCLASSOF(?Name, :NamedPizza),
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),
  OBJECTUNIONOF(_:b1, ?Toppings),
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)
}.
```

```
?Name  $\sqsubseteq$  :NamedPizza
?Name  $\sqsubseteq$   $\exists$  hasC0.{?Country}

?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza  
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}  
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

template name      type      parameter name      optional input

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]  
:: {  
  SUBCLASSOF(?Name, :NamedPizza),  
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),  
  OBJECTUNIONOF(_:b1, ?Toppings),  
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)  
}.
```

```
?Name  $\sqsubseteq$  :NamedPizza  
?Name  $\sqsubseteq$   $\exists$  hasC0.{?Country}  
  
?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)  
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

template name      type      parameter name      optional input      list input

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]
:: {
  SUBCLASSOF(?Name, :NamedPizza),
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),
  OBJECTUNIONOF(_:b1, ?Toppings),
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)
}.
```

```
?Name  $\sqsubseteq$  :NamedPizza
?Name  $\sqsubseteq$   $\exists$  hasCO.{?Country}

?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

template name    type    parameter name    optional input    list input

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]
:: {
  SUBCLASSOF(?Name, :NamedPizza),
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),
  OBJECTUNIONOF(_:b1, ?Toppings),
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)
}.
```

list expansion

```
?Name  $\sqsubseteq$  :NamedPizza
?Name  $\sqsubseteq$   $\exists$  hasC0.{?Country}

?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings ...
```



# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza  
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}  
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

template name    type    parameter name    optional input    list input

```
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]  
:: {  
  SUBCLASSOF(?Name, :NamedPizza),  
  SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
  SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),  
  OBJECTUNIONOF(_:b1, ?Toppings),  
  cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)  
}.  
list expansion    marked for expansion
```

```
?Name  $\sqsubseteq$  :NamedPizza  
?Name  $\sqsubseteq$   $\exists$  hasC0.{?Country}  
  
?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)  
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings ...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza  
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin. {Italy}  
Margherita  $\sqsubseteq$   $\forall$  hasTopping. (Tomato  $\sqcup$  Mozzarella)  
Margherita  $\sqsubseteq$   $\exists$  hasTopping. Tomato  
Margherita  $\sqsubseteq$   $\exists$  hasTopping. Mozzarella
```

```
NAMEDPIZZA(Margherita, Italy, (Tomato, Mozzarella))
```

```
template name  type  parameter name  optional input  list input  
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]  
:: {  
    SUBCLASSOF(?Name, :NamedPizza),  
    SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
    SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),  
    OBJECTUNIONOF(_:b1, ?Toppings),  
    cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)  
}.  
list expansion  marked for expansion
```

```
?Name  $\sqsubseteq$  :NamedPizza  
?Name  $\sqsubseteq$   $\exists$  hasC0. {?Country}  
  
?Name  $\sqsubseteq$   $\forall$  hasT. ( $\sqcup$ ?Toppings)  
?Name  $\sqsubseteq$   $\exists$  hasT. ?Toppings ...
```

# OTTR templates

```
Margherita  $\sqsubseteq$  NamedPizza  
Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin.{Italy}  
Margherita  $\sqsubseteq$   $\forall$  hasTopping.(Tomato  $\sqcup$  Mozzarella)  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato  
Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella
```

```
NAMEDPIZZA(Margherita, Italy, (Tomato, Mozzarella))  
NAMEDPIZZA(Grandiosa, ottr:none, (Tomato, Jarlsberg, Ham, SweetPepper))
```

```
template name  type  parameter name  optional input  list input  
NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]  
:: {  
    SUBCLASSOF(?Name, :NamedPizza),  
    SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
    SUBOBJECTALLVALUESFROM(?Name, :hasTopping, _:b1),  
    OBJECTUNIONOF(_:b1, ?Toppings),  
    cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)  
}.  
list expansion  marked for expansion
```

```
?Name  $\sqsubseteq$  :NamedPizza  
?Name  $\sqsubseteq$   $\exists$  hasC0.{?Country}  
  
?Name  $\sqsubseteq$   $\forall$  hasT.( $\sqcup$ ?Toppings)  
?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings...
```

# OTTR templates

Margherita  $\sqsubseteq$  NamedPizza

Margherita  $\sqsubseteq$   $\exists$  hasCountryOfOrigin. {Italy}

Margherita  $\sqsubseteq$   $\forall$  hasTopping. (Tomato  $\sqcup$  Mozzarella)

Margherita  $\sqsubseteq$   $\exists$  hasTopping.Tomato

Margherita  $\sqsubseteq$   $\exists$  hasTopping.Mozzarella

Grandiosa  $\sqsubseteq$  NamedPizza

Grandiosa  $\sqsubseteq$   $\forall$  hasTopping. (Tomato  $\sqcup$  Jarlsberg  $\sqcup$  Ham  $\sqcup$  SweetPepper)

Grandiosa  $\sqsubseteq$   $\exists$  hasTopping.Tomato

Grandiosa  $\sqsubseteq$   $\exists$  hasTopping.Jarlsberg

Grandiosa  $\sqsubseteq$   $\exists$  hasTopping.Ham

Grandiosa  $\sqsubseteq$   $\exists$  hasTopping.SweetPepper

NAMEDPIZZA(Margherita, Italy, (Tomato, Mozzarella))

NAMEDPIZZA(Grandiosa, ottr:none, (Tomato, Jarlsberg, Ham, SweetPepper))

template name      type      parameter name  
optional input      list input

NAMEDPIZZA[ owl:Class ?Name, ? owl:Individual ?Country, List<owl:Class> ?Toppings ]  
:: {  
    SUBCLASSOF(?Name, :NamedPizza),  
    SUBOBJECTHASVALUE(?Name, :hasCountryOfOrigin, ?Country),  
    SUBOBJECTALLVALUESFROM(?Name, :hasTopping, \_:b1),  
    OBJECTUNIONOF(\_:b1, ?Toppings),  
    cross | SUBOBJECTSOMEVALUESFROM(?Name, :hasTopping, ++ ?Toppings)  
} .

list expansion      marked for expansion

?Name  $\sqsubseteq$  :NamedPizza

?Name  $\sqsubseteq$   $\exists$  hasC0. {?Country}

?Name  $\sqsubseteq$   $\forall$  hasT. ( $\sqcup$ ?Toppings)

?Name  $\sqsubseteq$   $\exists$  hasT.?Toppings ...

## Template signatures

- A *template signature* gives the *interface* of a template, and contains just its name and parameter declarations, e.g.

NAMEDPIZZA[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings] .

# Template signatures

- A *template signature* gives the *interface* of a template, and contains just its name and parameter declarations, e.g.

NAMEDPIZZA[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings] .

- Giving just a signature, instead of the full definition, is convenient when:
  - you have not yet defined the template
  - to ensure correct usage of a template, but the definition is unavailable
  - for examples or documentation

## Base templates

- A *base template* is a template without definition

## Base templates

- A *base template* is a template without definition
- Currently, the only base template is

```
TRIPLE[ottt:IRI ?subject, ! ottt:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```



## Base templates

- A *base template* is a template without definition
- Currently, the only base template is

```
TRIPLE[ottt:IRI ?subject, ! ottt:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

- A TRIPLE-instance represents an RDF triple, e.g.:

```
TRIPLE(ex:martin, ex:knows, ex:peter)  $\rightsquigarrow$  ex:martin ex:knows ex:peter .
```

## Base templates

- A *base template* is a template without definition
- Currently, the only base template is

```
TRIPLE[ottt:IRI ?subject, ! ottt:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

- A TRIPLE-instance represents an RDF triple, e.g.:

```
TRIPLE(ex:martin, ex:knows, ex:peter)  $\rightsquigarrow$  ex:martin ex:knows ex:peter .
```

- Translation done by implementation

## Base templates

- A *base template* is a template without definition
- Currently, the only base template is

```
TRIPLE[ottt:IRI ?subject, ! ottt:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

- A TRIPLE-instance represents an RDF triple, e.g.:

```
TRIPLE(ex:martin, ex:knows, ex:peter)  $\rightsquigarrow$  ex:martin ex:knows ex:peter .
```

- Translation done by implementation
- Other base templates possible: E.g. QUAD

## Base templates

- A *base template* is a template without definition
- Currently, the only base template is

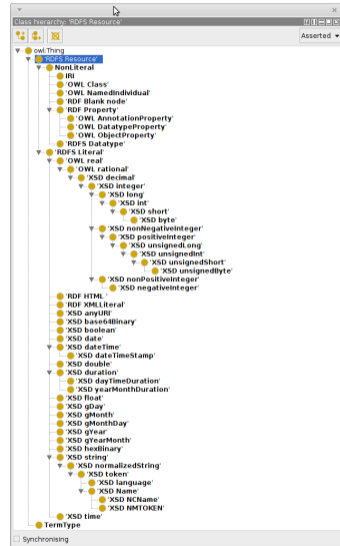
```
TRIPLE[ottt:IRI ?subject, ! ottt:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

- A TRIPLE-instance represents an RDF triple, e.g.:

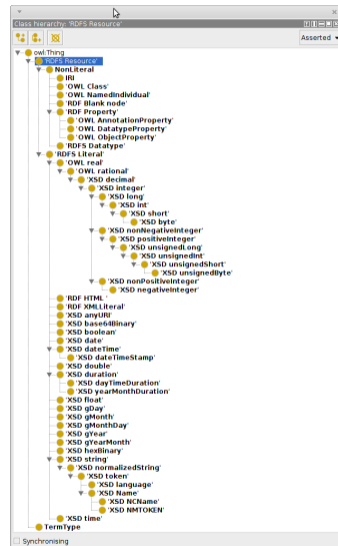
```
TRIPLE(ex:martin, ex:knows, ex:peter) ~→ ex:martin ex:knows ex:peter .
```

- Translation done by implementation
- Other base templates possible: E.g. QUAD
- Note: “Templates all the way down”

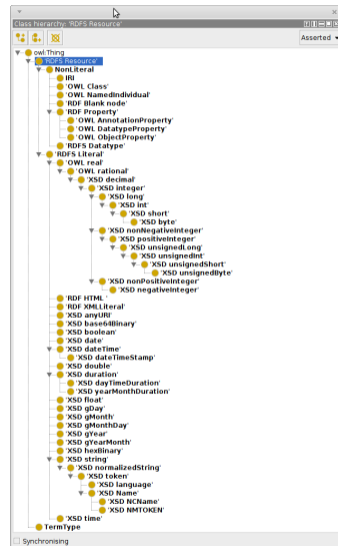
- Specify permissible arguments and consistent use of resources



- Specify permissible arguments and consistent use of resources
- Base-types limited to classes and datatypes defined in XSD, RDF, RDFS, and OWL
  - xsd:int, rdf:Property, rdfs:Resource, ...
  - rdfs:Resource is default and most general



- Specify permissible arguments and consistent use of resources
- Base-types limited to classes and datatypes defined in XSD, RDF, RDFS, and OWL
  - `xsd:int`, `rdf:Property`, `rdfs:Resource`, ...
  - `rdfs:Resource` is default and most general
- Also supports lists and non-empty lists (e.g. `NEList(List(xsd:string))`)



- Similar to types in programming languages (e.g. Java)

---

<sup>1</sup><https://dev.spec.ottr.xyz/r0TTR/develop/types.owl.ttl>

<sup>2</sup><https://dev.spec.ottr.xyz/r0TTR/develop/puntypes.owl.ttl>



- Similar to types in programming languages (e.g. Java)
- But adapted to ontologies

---

<sup>1</sup><https://dev.spec.ottr.xyz/r0TTR/develop/types.owl.ttl>

<sup>2</sup><https://dev.spec.ottr.xyz/r0TTR/develop/puntypes.owl.ttl>

- Similar to types in programming languages (e.g. Java)
- But adapted to ontologies
- Can e.g. allow<sup>1</sup> or disallow<sup>2</sup> punning, depending on chosen hierarchy

---

<sup>1</sup><https://dev.spec.ottr.xyz/r0TTR/develop/types.owl.ttl>

<sup>2</sup><https://dev.spec.ottr.xyz/r0TTR/develop/puntypes.owl.ttl>

- Type checking instances

```
SUBCLASSOF[owl:Class ?sub, owl:Class ?super] .
```

```
SUBCLASSOF(p:Margherita, p:Pizza) .
```

```
SUBCLASSOF("Margherita", p:Pizza) .
```

- Type checking instances

```
SUBCLASSOF[owl:Class ?sub, owl:Class ?super] .
```

```
SUBCLASSOF(p:Margherita, p:Pizza) .      OK
```

```
SUBCLASSOF("Margherita", p:Pizza) .      not OK
```

- Type checking instances

```
SUBCLASSOF[owl:Class ?sub, owl:Class ?super] .
```

```
SUBCLASSOF(p:Margherita, p:Pizza) .    OK
```

```
SUBCLASSOF("Margherita", p:Pizza) .    not OK
```

- Type checking template definitions

```
PIZZA[owl:Class ?name, xsd:string ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?label, p:Pizza),  
  TRIPLE(?name, rdfs:label, ?label)  
} .
```

- Type checking instances

```
SUBCLASSOF[owl:Class ?sub, owl:Class ?super] .
```

```
SUBCLASSOF(p:Margherita, p:Pizza) .    OK
```

```
SUBCLASSOF("Margherita", p:Pizza) .    not OK
```

- Type checking template definitions

```
PIZZA[owl:Class ?name, xsd:string ?label] :: {  
  TRIPLE(?name, rdf:type, owl:Class),  
  SUBCLASSOF(?label, p:Pizza),    not OK  
  TRIPLE(?name, rdfs:label, ?label)  
} .
```

# Types – Consistent use

## Examples

```
INSTANCEOF[owl:Individual ?elem, owl:Class ?class] .
```

```
INSTANCEOF(_:blank, p:Pizza)
```

```
INSTANCEOF(p:mypizza, _:blank)
```

# Types – Consistent use

## Examples

INSTANCEOF[owl:Individual ?elem, owl:Class ?class] .

INSTANCEOF(\_:blank, p:Pizza)

INSTANCEOF(p:mypizza, \_:blank)      OK? (depending on type hierarchy)



# Types – Consistent use

## Examples

INSTANCEOF[`owl:Individual ?elem`, `owl:Class ?class`] .

INSTANCEOF(`_:blank`, `p:Pizza`)

INSTANCEOF(`p:mypizza`, `_:blank`)      **OK?** (depending on type hierarchy)

LABEL[`ottr:IRI ?elem`, `xsd:string ?label`] .

LABEL(`p:mypizza`, "A pizza")

# Types – Consistent use

## Examples

INSTANCEOF[`owl:Individual ?elem`, `owl:Class ?class`] .

INSTANCEOF(`_:blank`, `p:Pizza`)

INSTANCEOF(`p:mypizza`, `_:blank`)      **OK?** (depending on type hierarchy)

LABEL[`ottr:IRI ?elem`, `xsd:string ?label`] .

LABEL(`p:mypizza`, "A pizza")      **OK**

# Types – Consistent use

## Examples

INSTANCEOF[`owl:Individual ?elem`, `owl:Class ?class`] .

INSTANCEOF(`_:blank`, `p:Pizza`)

INSTANCEOF(`p:mypizza`, `_:blank`)      **OK?** (depending on type hierarchy)

LABEL[`ottr:IRI ?elem`, `xsd:string ?label`] .

LABEL(`p:mypizza`, "A pizza")      **OK**

INSTANCEOF(`_:blank2`, `p:Pizza`)

LABEL(`p:mypizza`, `_:blank2`)

# Types – Consistent use

## Examples

INSTANCEOF[`owl:Individual ?elem`, `owl:Class ?class`] .

INSTANCEOF(`_:blank`, `p:Pizza`)

INSTANCEOF(`p:mypizza`, `_:blank`)      **OK?** (depending on type hierarchy)

LABEL[`ottr:IRI ?elem`, `xsd:string ?label`] .

LABEL(`p:mypizza`, "A pizza")      **OK**

INSTANCEOF(`_:blank2`, `p:Pizza`)

LABEL(`p:mypizza`, `_:blank2`)      **not OK**

- The type system ensures that:
  - Templates are used correctly

- The type system ensures that:
  - Templates are used correctly
  - Resources are used consistently

- The type system ensures that:
  - Templates are used correctly
  - Resources are used consistently
  - Any set of valid instances expands into a valid RDF-graph (e.g. no literal in subject position)

- The type system ensures that:
  - Templates are used correctly
  - Resources are used consistently
  - Any set of valid instances expands into a valid RDF-graph (e.g. no literal in subject position)
  - With our templates for OWL-axioms, any set of valid instances expands to a proper OWL-ontology



- The type system ensures that:
  - Templates are used correctly
  - Resources are used consistently
  - Any set of valid instances expands into a valid RDF-graph (e.g. no literal in subject position)
  - With our templates for OWL-axioms, any set of valid instances expands to a proper OWL-ontology
- These invariants also hold for more abstract templates (e.g. domain specific and user-facing templates)

- The type system ensures that:
  - Templates are used correctly
  - Resources are used consistently
  - Any set of valid instances expands into a valid RDF-graph (e.g. no literal in subject position)
  - With our templates for OWL-axioms, any set of valid instances expands to a proper OWL-ontology
- These invariants also hold for more abstract templates (e.g. domain specific and user-facing templates)
- Helps in highlighting the template's intention

## Non-blank

- Non-blank (written !) – requires a node that is not blank as argument

## Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

# Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

```
TRIPLE[ottt:IRI ?subject, ! ottt:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

```
TRIPLE(_:blank, :hasLength, 5)
```

```
TRIPLE(p:mypizza, _:blank2, 5)
```

# Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

```
TRIPLE[ottt:IRI ?subject, ! ottt:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

```
TRIPLE(_:blank, :hasLength, 5)    OK
```

```
TRIPLE(p:mypizza, _:blank2, 5)    not OK
```

# Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

```
TRIPLE[ottr:IRI ?subject, ! ottr:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

```
TRIPLE(_:blank, :hasLength, 5)    OK
```

```
TRIPLE(p:mypizza, _:blank2, 5)    not OK
```

```
RELATEDTOALL[ottr:IRI ?subs, rdf:Property ?relation, rdf:List<ottr:IRI> ?objs] :: {  
  cross | TRIPLE(?sub, ?relation, ++?objs)  
} .
```

# Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

```
TRIPLE[ottr:IRI ?subject, ! ottr:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

```
TRIPLE(_:blank, :hasLength, 5)    OK
```

```
TRIPLE(p:mypizza, _:blank2, 5)    not OK
```

```
RELATEDTOALL[ottr:IRI ?subs, rdf:Property ?relation, rdf:List<ottr:IRI> ?objs] :: { not OK  
  cross | TRIPLE(?sub, ?relation, ++?objs)  
} .
```



# Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

```
TRIPLE[ottr:IRI ?subject, ! ottr:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

```
TRIPLE(_:blank, :hasLength, 5)    OK
```

```
TRIPLE(p:mypizza, _:blank2, 5)    not OK
```

```
RELATEDTOALL[ottr:IRI ?subs, ! rdf:Property ?relation, rdf:List<ottr:IRI> ?objs] :: { OK  
  cross | TRIPLE(?sub, ?relation, ++?objs)  
} .
```

# Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

```
TRIPLE[ottr:IRI ?subject, ! ottr:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

```
TRIPLE(_:blank, :hasLength, 5)    OK
```

```
TRIPLE(p:mypizza, _:blank2, 5)    not OK
```

```
RELATEDTOALL[ottr:IRI ?subs, ! rdf:Property ?relation, rdf:List<ottr:IRI> ?objs] :: { OK  
  cross | TRIPLE(?sub, ?relation, ++?objs)  
} .
```

- With the non-blank flag, a user can be certain that no RDF-predicate is blank

# Non-blank

- Non-blank (written !) – requires a node that is not blank as argument
- Parameters used as arguments to non-blanks must also be non-blank.

```
TRIPLE[ottr:IRI ?subject, ! ottr:IRI ?predicate, rdfs:Resource ?object] :: BASE .
```

```
TRIPLE(_:blank, :hasLength, 5)    OK
```

```
TRIPLE(p:mypizza, _:blank2, 5)    not OK
```

```
RELATEDTOALL[ottr:IRI ?subs, ! rdf:Property ?relation, rdf:List<ottr:IRI> ?objs] :: { OK  
  cross | TRIPLE(?sub, ?relation, ++?objs)  
} .
```

- With the non-blank flag, a user can be certain that no RDF-predicate is blank
- Can also force IRIs of entities

- Instances with lists as arguments can be used together with an *expansion mode*

- Instances with lists as arguments can be used together with an *expansion mode*
- If a mode is present, each argument to an instance is treated as a list (non-list element are treated as lists of one element).

- Instances with lists as arguments can be used together with an *expansion mode*
- If a mode is present, each argument to an instance is treated as a list (non-list element are treated as lists of one element).
- We have three modes:
  - Cross (written `cross`), the template is applied to the *cross product* of the argument lists
  - ZipMax (`zipMax`) the template is applied to the *zip max* of the argument lists
  - ZipMin (`zipMin`) the template is applied to the *zip min* of the argument lists

# Expansion modes

## Example

```
SUBCLASSOF[ owl:Class ?sub, List<owl:Class> ?super ] :: {  
  cross | TRIPLE( ?sub, rdfs:subClassOf, ++?super )  
} .
```

# Expansion modes

## Example

```
SUBCLASSOF[ owl:Class ?sub, List<owl:Class> ?super ] :: {  
    cross | TRIPLE(?sub, rdfs:subClassOf, ++?super)  
} .
```

```
SUBCLASSOF(A, (X, Y, Z)) .
```



# Expansion modes

## Example

```
SUBCLASSOF[ owl:Class ?sub, List<owl:Class> ?super ] :: {  
    cross | TRIPLE(?sub, rdfs:subClassOf, ++?super)  
} .
```

```
SUBCLASSOF(A, (X, Y, Z)) .
```

```
TRIPLE(A, rdfs:subClassOf, X)
```

```
TRIPLE(A, rdfs:subClassOf, Y)
```

```
TRIPLE(A, rdfs:subClassOf, Z)
```

# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    cross | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
}
```

```
SUBCLASSOF((A, B, C), (X, Y, Z)) .
```

# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    cross | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
}
```

```
SUBCLASSOF((A, B, C), (X, Y, Z)) .
```

```
TRIPLE(A, rdfs:subClassOf, X) TRIPLE(B, rdfs:subClassOf, X) TRIPLE(C, rdfs:subClassOf, X)
```

```
TRIPLE(A, rdfs:subClassOf, Y) TRIPLE(B, rdfs:subClassOf, Y) TRIPLE(C, rdfs:subClassOf, Y)
```

```
TRIPLE(A, rdfs:subClassOf, Z) TRIPLE(B, rdfs:subClassOf, Z) TRIPLE(C, rdfs:subClassOf, Z)
```

# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    zipMin | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
}
```

```
SUBCLASSOF((A, B, C), (X, Y, Z)) .
```

# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    zipMin | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
}
```

```
SUBCLASSOF((A, B, C), (X, Y, Z)) .
```

```
TRIPLE(A, rdfs:subClassOf, X)
```

```
TRIPLE(B, rdfs:subClassOf, Y)
```

```
TRIPLE(C, rdfs:subClassOf, Z)
```

# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    zipMin | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
}
```

```
SUBCLASSOF((A, B), (X, Y, Z)) .
```

# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    zipMin | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
}
```

```
SUBCLASSOF((A, B), (X, Y, Z)) .
```

```
TRIPLE(A, rdfs:subClassOf, X)
```

```
TRIPLE(B, rdfs:subClassOf, Y)
```

# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    zipMax | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
}
```

```
SUBCLASSOF((A, B), (X, Y, Z)) .
```



# Expansion modes

## Example

```
SUBCLASSOF[List< owl:Class> ?sub, List<owl:Class> ?super] :: {  
    zipMax | TRIPLE(++?sub, rdfs:subClassOf, ++?super)  
} .
```

```
SUBCLASSOF((A, B), (X, Y, Z)) .
```

```
TRIPLE(A, rdfs:subClassOf, X)
```

```
TRIPLE(B, rdfs:subClassOf, Y)
```

```
TRIPLE(none, rdfs:subClassOf, Z)
```

- Removes repetition, e.g. instead of

`SUBCLASSOF(A, B), SUBCLASSOF(A, C), SUBCLASSOF(A, D)`

## Expansion modes – Motivation

- Removes repetition, e.g. instead of

`SUBCLASSOF(A, B), SUBCLASSOF(A, C), SUBCLASSOF(A, D)`

we can simply write

`cross | SUBCLASSOF(A, ++(B, C, D))`

- DRY Principle (“Do not repeat yourself”) – Easier to maintain

## Expansion modes – Motivation

- Removes repetition, e.g. instead of

`SUBCLASSOF(A, B), SUBCLASSOF(A, C), SUBCLASSOF(A, D)`

we can simply write

`cross | SUBCLASSOF(A, ++(B, C, D))`

- DRY Principle (“Do not repeat yourself”) – Easier to maintain
- Allows a template to take arbitrary number of arguments

## Expansion modes – Motivation

- Removes repetition, e.g. instead of

`SUBCLASSOF(A, B), SUBCLASSOF(A, C), SUBCLASSOF(A, D)`

we can simply write

`cross | SUBCLASSOF(A, ++(B, C, D))`

- DRY Principle (“Do not repeat yourself”) – Easier to maintain
- Allows a template to take arbitrary number of arguments
- Can use both the elements of a list and the list itself as arguments

## Modifiers – Optional

- Optional (?) – permits expansion even if argument is a missing value
- Missing values are denoted by `ottr:none` (and empty cells in `tabOTTR`)
- Instances with missing values passed to parameter without ? are discarded

# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .
```

```
PIZZA(p:WheatEver, none) .
```

# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .    OK
```

```
PIZZA(p:WheatEver, none) .    OK
```



# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .    OK
```

```
PIZZA(p:WheatEver, none) .    OK
```

```
PIZZA(p:WheatEver, none) .
```

# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .    OK
```

```
PIZZA(p:WheatEver, none) .    OK
```

```
PIZZA(p:WheatEver, none) .
```

⋮

```
TRIPLE(p:WheatEver, rdf:type, owl:Class)
```

```
TRIPLE(p:WheatEver, rdfs:subClassOf, p:Pizza)
```

```
TRIPLE(p:WheatEver, rdfs:label, none)
```

# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .    OK
```

```
PIZZA(p:WheatEver, none) .    OK
```

```
PIZZA(p:WheatEver, none) .
```

⋮

```
TRIPLE(p:WheatEver, rdf:type, owl:Class)
```

```
TRIPLE(p:WheatEver, rdfs:subClassOf, p:Pizza)
```

```
TRIPLE(p:WheatEver, rdfs:label, none)
```

# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .    OK
```

```
PIZZA(p:WheatEver, none) .    OK
```

```
PIZZA(none, "No pizza"@en) .
```

# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .    OK
```

```
PIZZA(p:WheatEver, none) .    OK
```

```
PIZZA(none, "No pizza"@en) .    OK
```

# Optional

## Example

```
PIZZA[class ?name, ? xsd:string ?label] .
```

```
PIZZA(p:Margherita, "Margherita"@it) .    OK
```

```
PIZZA(p:WheatEver, none) .    OK
```

```
PIZZA(none, "No pizza"@en) .    OK
```

## Default Values

- We also allow default values for parameters

## Default Values

- We also allow default values for parameters
- Used if the argument to the parameter is `none`



# Default Values

- We also allow default values for parameters
- Used if the argument to the parameter is `none`
- For example:

```
SUBJECTSOMEVALUESFROM[owl:Class ?class, owl:ObjectProperty ?prop, owl:Class ?range = owl:Thing] :: {  
  SUBCLASSOF(?class, _:b),  
  OBJECTSOMEVALUESFROM(_:b, ?prop, ?range)  
} .
```

(i.e.  $?class \sqsubseteq \exists ?prop . ?range$ )

# Default Values

- We also allow default values for parameters
- Used if the argument to the parameter is `none`
- For example:

```
SUBJECTSOMEVALUESFROM[owl:Class ?class, owl:ObjectProperty ?prop, owl:Class ?range = owl:Thing] :: {  
  SUBCLASSOF(?class, _:b),  
  OBJECTSOMEVALUESFROM(_:b, ?prop, ?range)  
} .
```

(i.e.  $?class \sqsubseteq \exists ?prop . ?range$ )

---

```
SUBJECTSOMEVALUESFROM(:Cat, :hasMother, :Cat)  
↪ {SUBCLASSOF(:Cat, _:c), OBJECTSOMEVALUESFROM(_:c, :hasMother, :Cat)}
```

(i.e.  $Cat \sqsubseteq \exists hasMother . Cat$ )

```
SUBJECTSOMEVALUESFROM(:Container, :contains, none)  
↪ {SUBCLASSOF(:Container, _:d), OBJECTSOMEVALUESFROM(_:d, :contains, owl:Thing)}
```

(i.e.  $Container \sqsubseteq \exists contains . \top$ )

- In addition to the core pattern mechanism, OTTR supports:
  - Signatures and base templates
  - Types and non-blank flag, and checks on these
  - Expansion modes and list support
  - Default and optional values
- The features are based on well-known principles in programming language design
- Provides
  - a simple, yet powerful language for expressing ontology patterns
  - that ensures correctness, easy maintenance, and uniform modelling
  - across all levels of abstraction

```
HASCOMPONENT[owl:Class ?whole, owl:Class ?component] :: {  
  SUBOBJECTSOMEVALUESFROM(?whole, ex:hasComponent, ?component)  
} .
```

```
HASCOMPONENT[  
  owl:Class ?whole,  
  owl:Class ?component  
] :: {  
  SUBOBJECTSOMEVALUESFROM  
  (?whole, ex:hasComponent, ?component)  
}.
```

```
HASCOMPONENT[
  owl:Class ?whole,
  owl:Class ?component
] :: {
  SUBOBJECTSOMEVALUESFROM
  (?whole, ex:hasComponent, ?component)
}.
```

```
:HasComponent a ottr:Template ;
  ottr:parameters (
    [ ottr:type owl:Class; ottr:variable :whole ]
    [ ottr:type owl:Class; ottr:variable :component ]
  ) ;
  ottr:pattern
  [ ottr:of ax:SubObjectSomeValuesFrom ;
    ottr:values (:whole ex:hasComponent :component ) ] .
```

```
HASCOMPONENT[  
  owl:Class ?whole,  
  owl:Class ?component  
] :: {  
  SUBOBJECTSOMEVALUESFROM  
  (?whole, ex:hasComponent, ?component)  
}.
```

```
HASCOMPONENTS[  
  owl:Class ?whole,  
  NEList<owl:Class> ?components  
] :: {  
  cross | HASCOMPONENT(  
    ?whole,  
    +?components)  
}.
```

```
:HasComponent a ottr:Template ;  
  ottr:parameters (  
    [ ottr:type owl:Class; ottr:variable :whole ]  
    [ ottr:type owl:Class; ottr:variable :component ]  
  ) ;  
  ottr:pattern  
  [ ottr:of ax:SubObjectSomeValuesFrom ;  
    ottr:values (:whole ex:hasComponent :component ) ] .
```

# wOTTR – Quick overview

```
HASCOMPONENT[
  owl:Class ?whole,
  owl:Class ?component
] :: {
  SUBOBJECTSOMEVALUESFROM
  (?whole, ex:hasComponent, ?component)
}.
```

```
HASCOMPONENTS[
  owl:Class ?whole,
  NELIST<owl:Class> ?components
] :: {
  cross | HASCOMPONENT(
    ?whole,
    +?components)
}.
```

```
:HasComponent a ottr:Template ;
  ottr:parameters (
    [ ottr:type owl:Class; ottr:variable :whole ]
    [ ottr:type owl:Class; ottr:variable :component ]
  ) ;
  ottr:pattern
  [ ottr:of ax:SubObjectSomeValuesFrom ;
    ottr:values (:whole ex:hasComponent :component ) ] .
```

```
:HasComponents a ottr:Template ;
  ottr:parameters (
    [ ottr:type owl:Class; ottr:variable :whole ]
    [ ottr:type (ottr:NEList owl:Class); ottr:variable :components ]
  ) ;
  ottr:pattern
  [ ottr:modifier ottr:cross; ottr:of :HasComponent ;
    ottr:arguments (
      [ ottr:value :whole ]
      [ ottr:modifier ottr:listExpand; ottr:value :components ] ) ] .
```



In wOTTR<sup>3</sup>, templates

- are resources typed with `ottr:Template`
- is `ottr:parameters-related` to an RDF list of resources representing its parameters
- is `ottr:pattern-related` to a set of resources representing its body instances

Each parameter

- is a resource (often a blank node)
- has a type, related to it via `ottr:type`
- has a variable name, related to it via `ottr:variable`

---

<sup>3</sup>See <https://spec.ottr.xyz/wOTTR/0.4/> for the full spec.

### Each instance

- is a resource (often a blank node)
- is `ottr:of`-related to the template it is an instance of
- has a list of arguments, related to it via
  - `ottr:values` if argument values are given directly
  - `ottr:arguments` if some of the arguments have modifiers (e.g. `ottr:listExpand`)

# Making templates and instances

## Tools

- Templates can be built with:
  - Any text editor (wOTTR, soon: stOTTR)
  - Soon: GUI

- Templates can be built with:
  - Any text editor (wOTTR, soon: stOTTR)
  - Soon: GUI
- Template instances can be made using:
  - Any text editor (wOTTR, soon: stOTTR)
  - Excel/LibreOffice etc. (tabOTTR)
  - Work in progress: General mapping language from RDBMSs, CSV, JSON, XML, etc.
  - From ontology with templates as queries (qOTTR – experimental)

- *Lutra*<sup>4</sup> is our open source CLI-tool for working with templates

---

<sup>4</sup><https://gitlab.com/ottr/lutra/lutra/releases>

# Lutra

## CLI tool

- *Lutra*<sup>4</sup> is our open source CLI-tool for working with templates
- Java 8, Streams, Jena

---

<sup>4</sup><https://gitlab.com/ottr/lutra/lutra/releases>

- *Lutra*<sup>4</sup> is our open source CLI-tool for working with templates
- Java 8, Streams, Jena
- Features:
  - Expand template library and instances

---

<sup>4</sup><https://gitlab.com/ottr/lutra/lutra/releases>

- *Lutra*<sup>4</sup> is our open source CLI-tool for working with templates
- Java 8, Streams, Jena
- Features:
  - Expand template library and instances
  - Type-checking, cycle detection, and other sanity checks

---

<sup>4</sup><https://gitlab.com/ottr/lutra/lutra/releases>



- *Lutra*<sup>4</sup> is our open source CLI-tool for working with templates
- Java 8, Streams, Jena
- Features:
  - Expand template library and instances
  - Type-checking, cycle detection, and other sanity checks
  - Input: tabOTTR, wOTTR, legacy wOTTR (others soon to come)

---

<sup>4</sup><https://gitlab.com/ottr/lutra/lutra/releases>

- *Lutra*<sup>4</sup> is our open source CLI-tool for working with templates
- Java 8, Streams, Jena
- Features:
  - Expand template library and instances
  - Type-checking, cycle detection, and other sanity checks
  - Input: tabOTTR, wOTTR, legacy wOTTR (others soon to come)
  - Output: wOTTR (others soon to come)

---

<sup>4</sup><https://gitlab.com/ottr/lutra/lutra/releases>

- *Lutra*<sup>4</sup> is our open source CLI-tool for working with templates
- Java 8, Streams, Jena
- Features:
  - Expand template library and instances
  - Type-checking, cycle detection, and other sanity checks
  - Input: tabOTTR, wOTTR, legacy wOTTR (others soon to come)
  - Output: wOTTR (others soon to come)
  - Also an API for working programmatically with templates
- To run Lutra: `$ java -jar lutra.jar`
- For help: `$ java -jar lutra.jar --help`

<sup>4</sup><https://gitlab.com/ottr/lutra/lutra/releases>

# Example scenario: Device factory



```
DEVICE[owl:Class ?itemClass, ott:NEList<owl:Class> ?components, ? xsd:string ?label] :: {
```

```
DEVICE[owl:Class ?itemClass, ott:NEList<owl:Class> ?components, ? xsd:string ?label] :: {  
  SUBCLASSOF(?itemClass, class:Device),
```

CONSISTSOFF[owl:Class ?whole, NList<owl:Class> ?components] .

“States that ?whole consists of exactly the components in ?components.”

DEVICE[owl:Class ?itemClass, ott:NList<owl:Class> ?components, ? xsd:string ?label] :: {  
 SUBCLASSOF(?itemClass, class:Device),

CONSISTSOFF[owl:Class ?whole, NList<owl:Class> ?components] .

“States that ?whole consists of exactly the components in ?components.”

```
DEVICE[owl:Class ?itemClass, ott:NList<owl:Class> ?components, ? xsd:string ?label] :: {  
  SUBCLASSOF(?itemClass, class:Device),  
  CONSISTSOFF(?itemClass, ?components),
```



CONSISTSOFF[`owl:Class ?whole`, `NEList<owl:Class> ?components`] .

“States that `?whole` consists of exactly the components in `?components`.”

```
DEVICE[owl:Class ?itemClass, ott:NEList<owl:Class> ?components, ? xsd:string ?label] :: {  
  SUBCLASSOF(?itemClass, class:Device),  
  CONSISTSOFF(?itemClass, ?components),  
  TRIPLE(?itemClass, rdfs:label, ?label)  
} .
```

1. Open up `instances.xlsx` containing some instances, and use Lutra to expand these instances into an Ontology via

```
$ java -jar lutra.jar -m expand -I tabottr -l ./templates -o expanded.ttl instances.xlsx
```

2. Open the expanded file in Protégé and examine the result
3. Add instances stating the following:
  - that a `ThinkPad_L420` has components of types `Intel_i5`, `GPU_Nvidia`, `RAM_2GB`, `HDD_1TB`, and has label "ThinkPad L420 Laptop",

```
DEVICEINSTANCE[owl:NamedIndividual ?item, owl:Class ?itemClass, ottt:NEList<owl:NamedIndividual> ?components] :: {
```

```
DEVICEINSTANCE[owl:NamedIndividual ?item, owl:Class ?itemClass, ottt:NEList<owl:NamedIndividual> ?components] :: {  
  TRIPLE(?item, rdf:type, ?itemClass),
```

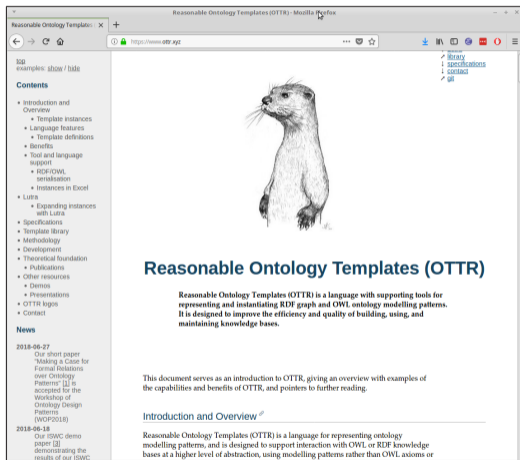
```
DEVICEINSTANCE[owl:NamedIndividual ?item, owl:Class ?itemClass, ottt:NEList<owl:NamedIndividual> ?components] :: {  
  TRIPLE(?item, rdf:type, ?itemClass),  
  cross | TRIPLE(?item, ex:hasComponent, ++?components)  
} .
```

3. Add instances stating the following:
  - that the concrete ThinkPad L420 `pc1` has concrete components `cpu1`, `ram1`, `graphics1`, and `hdd1`.
4. Expand the instances and reopen it in Protégé

```
DEVICE[owl:Class ?itemClass, ottt:NEList<owl:Class> ?components, ? xsd:string ?label] :: {  
  SUBCLASSOF(?itemClass, class:Device),  
  CONSISTSOOF(?itemClass, ?components),  
  TRIPLE(?itemClass, ?label)  
}
```

```
DEVICEINSTANCE[owl:NamedIndividual ?item, owl:Class ?itemClass, ottt:NEList<owl:NamedIndividual> ?components] :: {  
  TRIPLE(?item, rdf:type, ?itemClass),  
  cross | TRIPLE(?item, ex:hasComponent, ++?components)  
}
```

Available at [ottr.xyz](http://ottr.xyz)



Reasonable Ontology Templates (OTTR) - Mozilla Firefox

Reasonable Ontology Templates: x

examples: show / hide


### Contents

- Introduction and Overview
  - Template instances
- Language features
  - Template definitions
- Benefits
  - Tool and language support
    - RDF/OWL serialisation
      - Instances in Excel
- Lura
  - Expanding instances with Lura
- Specifications
- Template library
- Methodology
- Development
- Theoretical foundation
  - Publications
- Other resources
  - Demos
  - Presentations
- OTTR logos
- Contact

### News

2018-06-27  
Our short paper "Making a Case for Formal Relations over Ontology Patterns" [1] is accepted for the Workshop of Ontology Design Patterns (OWDP2018)

2018-06-18  
Our ISWC demo paper [2] demonstrating the results of our ISWC



## Reasonable Ontology Templates (OTTR)

Reasonable Ontology Templates (OTTR) is a language with supporting tools for representing and instantiating RDF graph and OWL ontology modelling patterns. It is designed to improve the efficiency and quality of building, using, and maintaining knowledge bases.

This document serves as an introduction to OTTR, giving an overview with examples of the capabilities and benefits of OTTR, and pointers to further reading.

### Introduction and Overview

Reasonable Ontology Templates (OTTR) is a language for representing ontology modelling patterns, and is designed to support interaction with OWL or RDF knowledge bases at a higher level of abstraction, using modelling patterns rather than OWL axioms or



Available at [ottr.xyz](http://ottr.xyz)

- *Lutra* CLI Java tool

```
File Edit View Search Terminal Help
p.NamedPizza rdf:type owl:Class .

## Generated by the Owl API (version 5.1.0) https://github.com/owlcs/owlapi/
~/temp/ottr: java -jar oslottr.jar -expand -in pizza.ttl

SPrefix : <http://example.com#> .
SPrefix p: <http://example.com/external#> .
SPrefix owl: <http://www.w3.org/2002/07/owl#> .
SPrefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
SPrefix xml: <http://www.w3.org/XML/1998/namespace> .
SPrefix xsd: <http://www.w3.org/2001/XMLSchema#> .
SPrefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
SBase <http://www.w3.org/2002/07/owl#> .

[ rdf:type owl:Ontology
  ] .

#####
# Object Properties
#####

## http://example.com/external#hasTopping
p:hasTopping rdf:type owl:ObjectProperty .

#####
# Classes
#####

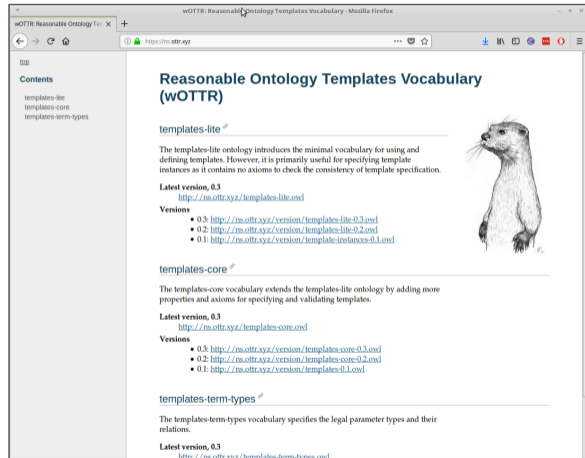
## http://example.com#AnchoviesTopping
:AnchoviesTopping rdf:type owl:Class .

## http://example.com#CaperTopping
:CaperTopping rdf:type owl:Class .

## http://example.com#FourSeasons
:FourSeasons rdf:type owl:Class ;
  rdfs:subClassOf p:NamedPizza ,
  [ rdf:type owl:Restriction ;
    owl:onProperty p:hasTopping ;
    owl:someValuesFrom :AnchoviesTopping
  ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty p:hasTopping ;
    owl:someValuesFrom :CaperTopping
  ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty p:hasTopping ;
    owl:someValuesFrom :MozzerellaTopping
  ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty p:hasTopping ;
    owl:someValuesFrom :MushroomTopping
  ] .
```

Available at [ottr.xyz](https://ottr.xyz)

- *Lutra* CLI Java tool
- Specs: wOTTR, stOTTR, tabOTTR, mOTTR, rOTTR



The screenshot shows a web browser window displaying the 'Reasonable Ontology Templates Vocabulary (wOTTR)' website. The browser's address bar shows the URL <https://ns.ottr.xyz>. The page has a light gray sidebar on the left with a 'Contents' section listing 'templates-lite', 'templates-core', and 'templates-term-types'. The main content area features three sections, each with a heading, a brief description, and a 'Latest version, 0.3' link. The first section is 'templates-lite', which includes a small illustration of a marmot. The second section is 'templates-core', and the third is 'templates-term-types'. Each section also lists previous versions (0.2 and 0.1) with their respective URLs.

## Reasonable Ontology Templates Vocabulary (wOTTR)

### templates-lite

The templates-lite ontology introduces the minimal vocabulary for using and defining templates. However, it is primarily useful for specifying template instances as it contains no axioms to check the consistency of template specification.

**Latest version, 0.3**  
<http://ns.ottr.xyz/templates-lite.owl>

**Versions**

- 0.3: <http://ns.ottr.xyz/version/templates-lite-0.3.owl>
- 0.2: <http://ns.ottr.xyz/version/templates-lite-0.2.owl>
- 0.1: <http://ns.ottr.xyz/version/template-instances-0.1.owl>

### templates-core

The templates-core vocabulary extends the templates-lite ontology by adding more properties and axioms for specifying and validating templates.

**Latest version, 0.3**  
<http://ns.ottr.xyz/templates-core.owl>

**Versions**

- 0.3: <http://ns.ottr.xyz/version/templates-core-0.3.owl>
- 0.2: <http://ns.ottr.xyz/version/templates-core-0.2.owl>
- 0.1: <http://ns.ottr.xyz/version/templates-0.1.owl>

### templates-term-types

The templates-term-types vocabulary specifies the legal parameter types and their relations.

**Latest version, 0.3**  
<http://ns.ottr.xyz/templates-term-types.owl>

Available at [ottr.xyz](http://ottr.xyz)

- Lutra CLI Java tool
- Specs: wOTTR, stOTTR, tabOTTR, mOTTR, rOTTR
- Library of OTTR templates

Reasonable Ontology Templates (OTTR) Library - Mozilla Firefox

Reasonable Ontology Templates: X +

0 <http://draft.ottr.xyz/>

```
http://draft.ottr.xyz/pizza/NamedPizza :pizza :| class, :country : ? individual, <:toppings> : + List |
::
t-owl-axiom:SubClassOf( :pizza, p:NamedPizza )
t-owl-axiom:SubObjectAllValuesFrom( :pizza, p:hasTopping, :_b1 )
t-owl-axiom:SubObjectIntersectionOf( :pizza, p:hasCountryOfOrigin, :country )
X | t-owl-axiom:SubObjectSomeValuesFrom( :pizza, #hasTopping, <:toppings> )
t-owl-rstr:ObjectIntersectionOf( :_b1, <:toppings> ) .
```

Available formats

- Specification
- all
- body
- head
- instance
- Expansion
- all
- body
- head
- Quoties
- select-body
- select-head
- string-construct
- string-update
- lowering-construct
- lowering-update
- FormatSample
- XML-format
- XML-source

Direct dependency templates

Templates instantiated in the body of this template:

- <http://candidate.ottr.xyz/owl/axiom/SubClassOf>
- <http://candidate.ottr.xyz/owl/axiom/SubObjectAllValuesFrom>
- <http://candidate.ottr.xyz/owl/axiom/SubObjectIntersectionOf>
- <http://candidate.ottr.xyz/owl/axiom/SubObjectSomeValuesFrom>
- <http://candidate.ottr.xyz/owl/restriction/ObjectIntersectionOf>

Diagram of pattern

RDF graph visualisation of the expanded body:

Pattern

The pattern the template represents, i.e., the expanded template body.

Prefixes: <http://www.cs-ohs.org/ontologies/pizza/pizza.owl#>

Available at [ottr.xyz](http://ottr.xyz)

- *Lutra* CLI Java tool
- Specs: wOTTR, stOTTR, tabOTTR, mOTTR, rOTTR
- Library of OTTR templates
- Research papers, posters and demos

**Reasonable Ontology Templates**  
Mårin G. Sæviestad, Lutz Haveland, Karsten Daxel, P. Lopp, Department of Informatics, University of Oslo

**Problem**

- Lack of established abstraction mechanisms for R2P-CWL, to capture modelling patterns
- Repetitive and tedious ontology construction
- Difficult to engage domain experts
- Low-level maintenance methods

**Reasonable Ontology Templates (OTTR)**

- Practical mapping language for R2P-CWL
- Formal foundation, built with RISC standards
- Declarative and modular patterns
- Enforce uniform modelling
- Secure completeness of input

- Reusable modelling patterns and bulk content
- Advanced template maintenance
- Available at [ottr.xyz](http://ottr.xyz)
- Open source Java implementation Lutra
- Template views, specifications, demos

**Construct ontologies by instantiating OTTR templates**

**Multiple serialisations: stOTTR (compact), tabOTTR (tabular), wOTTR (RDF), qOTTR (SPARQL)**

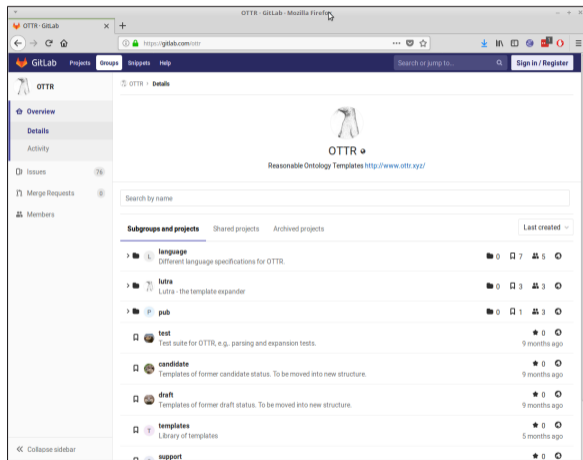
**Ontology engineering methodology: Template abstractions adapted to different user types**

**Maintain ontologies by maintaining templates: Detect and remove redundancies**

UiO University of Oslo

Available at `ottr.xyz`

- *Lutra* CLI Java tool
- Specs: wOTTR, stOTTR, tabOTTR, mOTTR, rOTTR
- Library of OTTR templates
- Research papers, posters and demos
- Git:  
`gitlab.com/ottr/`



- Published after lecture (before 17:00)
- Same procedure as the small obligs (one week, one attempt)
- Can use MrOblig to check your answers
- Can/Should use Lutra, but only need to use

```
$ java -jar lutra.jar -m expand -l <library> -o <expanded> <instances>
```

where

- <library> is the folder containing the templates
- <expanded> is the file to write the expanded instances to
- <instances> is the file containing the instances to expand