

Reasonable Ontology Templates (OTTR)

Martin G. Skjæveland

April 19, 2023



Who am I?

- Martin G. Skjæveland
- University of Oslo, and University of Stavanger
- Background:
 - Computer science,
 - Knowledge representation,
 - Semantic technologies
- Interests:
 - Ontology engineering,
 - Ontology-based data access,
 - Industrial use cases
- Pattern-based ontology engineering
- Reasonable Ontology Templates



Aibel Use Case

- Large-scale ontology (100 000+ classes)
- Representing industry standards and creative designs
- Automating highly complex, but trivial work
- Verification of requirements with reasoning
- Reduced cost and improved quality and accountability

Requirements for building large (industrial) ontologies

- Support large number of classes
 - Some of which might have *very* similar shape
- Support a large number of (industry) standards
 - which might have apparently incompatible ways of defining the world...
- Support *consistent* modelling
 - Semantically consistent (of course!), but also
 - model the same category of thing in the same way, across different standards
- Support collaborative development
 - Contributors may have varying background and competence (also about ontologies)
 - Model consistently between *all* contributors!
- Support (automated) mechanisms for QA and verification



This is a *lot* harder than you may think!

Difficulties with ontology editors

ID	Label	Standard	Size	Pressure class
Flange1	Flange ACME 66 NPS 1 CL150	ACME 66	NPS 1	Class 150
Flange2	Flange ACME 66 NPS 2 CL150	ACME 66	NPS 2	Class 150
Flange3	Flange ACME 66 NPS 1 CL300	ACME 66	NPS 1	Class 300
Flange4	Flange ACME 66 NPS 2 CL300	ACME 66	NPS 2	Class 300
Flange5	Flange ACME 66 NPS 1 CL600	ACME 66	NPS 1	Class 600
Flange6	Flange ACME 66 NPS 2 CL600	ACME 66	NPS 2	Class 600
...	...	...	...	...

Impossible to model consistently

- Humans are *not* good at repetitive tasks!
- Collaboration only makes it worse...
- What if you want to change the way you model flanges...?
- Makes no sense to non-ontologists!

The screenshot shows an ontology editor window titled 'untitled-ontology-100'. The main panel displays a class hierarchy under 'owl:Thing'. The hierarchy is as follows: 'Flange' (a subclass of 'owl:Thing') has two subclasses: 'Flange ACME 66' and 'Nominal Pipe Size'. 'Flange ACME 66' has six subclasses: 'Flange ACME 66 NPS 1 CL150' (highlighted in blue), 'Flange ACME 66 NPS 1 CL300', 'Flange ACME 66 NPS 1 CL600', 'Flange ACME 66 NPS 2 CL150', 'Flange ACME 66 NPS 2 CL300', and 'Flange ACME 66 NPS 2 CL600'. The right panel shows the details for the selected class 'Flange ACME 66 NPS 1 CL150'. It includes a search bar, a description, and a list of annotations. The annotations are: 'rdfs:label' with the value 'Flange ACME 66 NPS 1 CL150'. The description is 'Flange ACME 66 NPS 1 CL150'. The 'Equivalent To' section is empty. The 'SubClass Of' section lists: 'Flange ACME 66', ''has NPS' only 'NPS 1'', and ''has pressure rating' only CL150'. The bottom status bar indicates 'To use the reasoner click Reasoner > Start reasoner' and 'Show Inferences'.

Excel is not a data management tool



- No constraint support => *define as many duplicates as you want!*
- No referential integrity => *introduce ad-hoc classes as you go!*
- Unlimited schema freedom => *inconsistent modelling!*
- What about *type checking, simultaneous edits, edit logs, versioning, and conflict resolution...?*

ID	Label	Standard	Size	Pressure class	Christian's coolness index
Flange1	Flange ACME 66 NPS 1 CL150	ACME 66	NPS 1	Class 150	1
Flange2	Flange ACME 66 NPS 1 CL150	ACME 66	NPS 1	Class 150	1
Flange3	Flange ACME 66 NPS 112 CL150	ACME 66	NPS 112	Class 150	777

Duplicates!

Does this
class exist...?

WAT?!?

aibel®

Improving the efficiency and quality of ontology engineering (for large-scale ontologies)

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 (for large-scale ontologies)

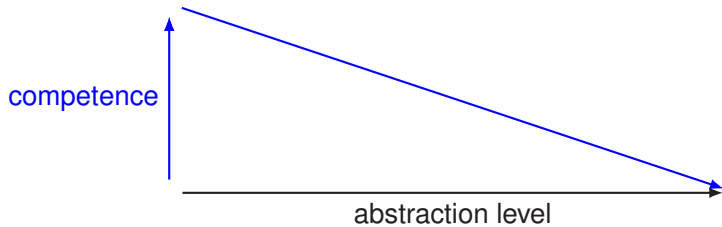
Problem:

- 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

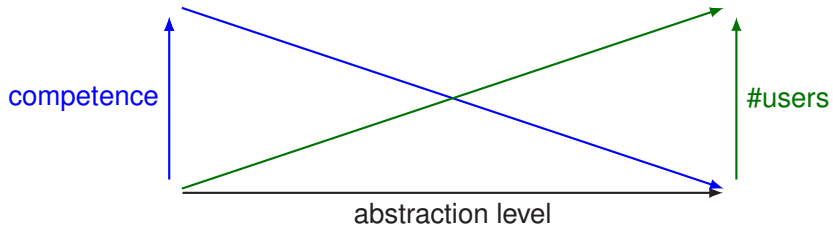
We need:

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

Abstractions



Abstractions



Assembly

```
section      .text
global      _start

_start:

    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80

    mov     eax, 1
    int     0x80

section      .data

msg         db  'Hello, world!', 0xa
len         equ $ - msg
```

Python

```
print("Hello, world!")
```

Assembly

```
section    .text
global    _start

_start:

    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80

    mov     eax, 1
    int     0x80

section    .data

msg        db  'Hello, world!', 0xa
len        equ $ - msg
```

Python

```
print("Hello, world!")
```

- Hide complexity

Assembly

```
section    .text
global    _start

_start:

    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80

    mov     eax, 1
    int     0x80

section    .data

msg        db  'Hello, world!', 0xa
len        equ $ - msg
```

Python

```
print("Hello, world!")
```

- Hide complexity
- Separation of concerns

Assembly

```
section    .text
global    _start

_start:

    mov     edx, len
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     0x80

    mov     eax, 1
    int     0x80

section    .data

msg        db  'Hello, world!', 0xa
len        equ $ - msg
```

Python

```
print("Hello, world!")
```

- Hide complexity
- Separation of concerns
- Support different users

RDF/OWL

?

```
ex:Margherita
  rdfs:subClassOf p:NamedPizza ;
  rdfs:subClassOf [ a      owl:Restriction ;
                   owl:hasValue    ex:Italy ;
                   owl:onProperty  p:hasCountryOfOrigin
                 ] ;
  rdfs:subClassOf [ a      owl:Restriction ;
                   owl:allValuesFrom [ a      owl:Class ;
                                         owl:unionOf ( ex:Mozzarella ex:Tomato )
                                       ] ;
                   owl:onProperty  p:hasTopping
                 ] ;
  rdfs:subClassOf [ a      owl:Restriction ;
                   owl:onProperty  p:hasTopping ;
                   owl:someValuesFrom ex:Tomato
                 ] ;
  rdfs:subClassOf [ a      owl:Restriction ;
                   owl:onProperty  p:hasTopping ;
                   owl:someValuesFrom ex:Mozzarella
                 ] .
```

- Hide complexity
- Separation of concerns
- Support different users

Manchester OWL

?

```
Class: Margherita
SubClassOf:
  NamedPizza,
  hasCountryOfOrigin some { Italy },
  hasTopping some Mozzarella,
  hasTopping some Tomato,
  hasTopping only (Mozzarella or Tomato)
```

- Hide complexity
- Separation of concerns
- Support different users

Manchester OWL

?

```
Class: Margherita
SubClassOf:
  NamedPizza,
  hasCountryOfOrigin some { Italy },
  hasTopping some Mozzarella,
  hasTopping some Tomato,
  hasTopping only (Mozzarella or Tomato)
```

- Hide complexity
- Separation of concerns
- Support different users
- Still low-level OWL “coding”

Manchester OWL

?

```
Class: Margherita
SubClassOf:
  NamedPizza,
  hasCountryOfOrigin some { Italy },
  hasTopping some Mozzarella,
  hasTopping some Tomato,
  hasTopping only (Mozzarella or Tomato)
```

- Hide complexity
- Separation of concerns
- Support different users
- Still low-level OWL “coding”
- Impossible to not care about logic

Manchester OWL

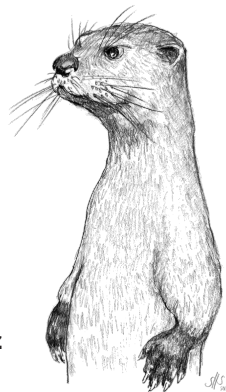
?

```
Class: Margherita
SubClassOf:
  NamedPizza,
  hasCountryOfOrigin some { Italy },
  hasTopping some Mozzarella,
  hasTopping some Tomato,
  hasTopping only (Mozzarella or Tomato)
```

- Hide complexity
- Separation of concerns
- Support different users
- Still low-level OWL “coding”
- Impossible to not care about logic
- No proper abstraction mechanism

Reasonable Ontology Templates (OTTR)

- Macro/Pattern language for knowledge bases
- Abstractions over RDF and OWL
- Represent and instantiate modelling patterns
- Gain well-known benefits from abstractions for creating, maintaining and communicating ontologies and knowledge bases
- Open-source tools and template libraries available at ottr.xyz
- Active community of industrial users



OTTR

OTTR: Example 1

```
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 .
```

OTTR: Example 1

```
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 .
```

OTTR: Example 1

```
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 .
```

OTTR: Example 1

```
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 .
```

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

OTTR: Example 1

```
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 .
```

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

OTTR: Example 1

```
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 .
```

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

OTTR: Example 1

```
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 .
```

```
ax:SubClassOf[?sub, ?super] :: {
    ottr:Triple(?sub, rdfs:subClassOf, ?super)
} .

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

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

OTTR: Example 1

```
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 .
```

```
ax:SubClassOf[?sub, ?super] :: {
    ottr:Triple(?sub, rdfs:subClassOf, ?super)
} .

pz:Pizza[?name, ?label] :: {
    ottr:Triple(?name, rdf:type, owl:Class),
    ax:SubClassOf(?name, p:Pizza),
    ottr:Triple(?name, rdfs:label, ?label)
} .

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

OTTR: Example 1

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

- Abstractions

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

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

- Abstractions
- Don't Repeat Yourself (DRY)

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[?name, ?label] :: {  
    ottr:Triple(?name, rdf:type, owl:Class),  
    ax:SubClassOf(?name, p:Pizza),  
    ottr:Triple(?name, rdfs:label, ?label)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[p:Margherita, "Margherita"@it] :: {  
    ottr:Triple(p:Margherita, rdf:type, owl:Class),  
    ax:SubClassOf(p:Margherita, p:Pizza),  
    ottr:Triple(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[?sub, ?super] :: {  
    ottr:Triple(?sub, rdfs:subClassOf, ?super)  
} .  
  
pz:Pizza[p:Margherita, "Margherita"@it] :: {  
    ottr:Triple(p:Margherita, rdf:type, owl:Class),  
    ax:SubClassOf(p:Margherita, p:Pizza),  
    ottr:Triple(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[p:Margherita, p:Pizza] :: {  
    ottr:Triple(p:Margherita, rdfs:subClassOf, p:Pizza)  
} .  
pz:Pizza[p:Margherita, "Margherita"@it] :: {  
    ottr:Triple(p:Margherita, rdf:type, owl:Class),  
    ax:SubClassOf(p:Margherita, p:Pizza),  
    ottr:Triple(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

```
ax:SubClassOf[p:Margherita, p:Pizza] :: {  
    ottr:Triple(p:Margherita, rdfs:subClassOf, p:Pizza)  
} .  
  
pz:Pizza[p:Margherita, "Margherita"@it] :: {  
    ottr:Triple(p:Margherita, rdf:type, owl:Class),  
    ottr:Triple(p:Margherita, rdfs:subClassOf, p:Pizza),  
    ottr:Triple(p:Margherita, rdfs:label, "Margherita"@it)  
} .  
  
pz:Pizza(p:Margherita, "Margherita"@it) .  
pz:Pizza(p:Hawaii, "Hawaii"@en) .  
pz:Pizza(p:Grandiosa, "Grandiosa"@no) .
```

OTTR: Example 1

```
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

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

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

- Typed parameters: (for type checking and consistent use of IRIs)

`pz:Pizza[owl:Class ?name, xsd:string ?label]`

- Typed parameters: (for type checking and consistent use of IRIs)

`pz:Pizza[owl:Class ?name, xsd:string ?label]`

- List-types and list-expanders: (for multiplying instances, and compact input)

`pz:Pizza[owl:Class ?name, List<xsd:string> ?label]`

- Typed parameters: (for type checking and consistent use of IRIs)

`pz:Pizza[owl:Class ?name, xsd:string ?label]`

- List-types and list-expanders: (for multiplying instances, and compact input)

`pz:Pizza[owl:Class ?name, List<xsd:string> ?label]`

- Optional parameters: (for missing input, more flexible pattern signatures)

`pz:Pizza[owl:Class ?name, ? xsd:string ?label]`

- Typed parameters: (for type checking and consistent use of IRIs)

`pz:Pizza[owl:Class ?name, xsd:string ?label]`

- List-types and list-expanders: (for multiplying instances, and compact input)

`pz:Pizza[owl:Class ?name, List<xsd:string> ?label]`

- Optional parameters: (for missing input, more flexible pattern signatures)

`pz:Pizza[owl:Class ?name, ? xsd:string ?label]`

- Default values: (for simplifying input)

`pz:Pizza[owl:Class ?name, xsd:string ?label = "Grandiosa"]`

- Typed parameters: (for type checking and consistent use of IRIs)

`pz:Pizza[owl:Class ?name, xsd:string ?label]`

- List-types and list-expanders: (for multiplying instances, and compact input)

`pz:Pizza[owl:Class ?name, List<xsd:string> ?label]`

- Optional parameters: (for missing input, more flexible pattern signatures)

`pz:Pizza[owl:Class ?name, ? xsd:string ?label]`

- Default values: (for simplifying input)

`pz:Pizza[owl:Class ?name, xsd:string ?label = "Grandiosa"]`

- Non-blank: (for forcing concrete values)

`pz:Pizza[! owl:Class ?name, ! xsd:string ?label]`

- 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) } .
```

- Type checking instances

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

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

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

```
Pizza(p:Quattro, 4) .
```

- 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) } .
```

- Type checking instances

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

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

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

```
Pizza(p:Quattro, 4) .    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

Examples

- Type checking instances

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

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

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

```
Pizza(p:Quattro, 4) .    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) } .
```

- Type checking consistent use

```
T[?x] :: {  
    Pizza(?x, _:blank) }.
```

- Type checking instances

```
SubClassOf[owl:Class ?sub, owl:Class ?super] .  
SubClassOf(p:Margherita, p:Pizza) .    OK  
SubClassOf("Margherita", p:Pizza) .    not OK  
Pizza(p:Quattro, 4) .    not OK
```

- Type checking template definitions

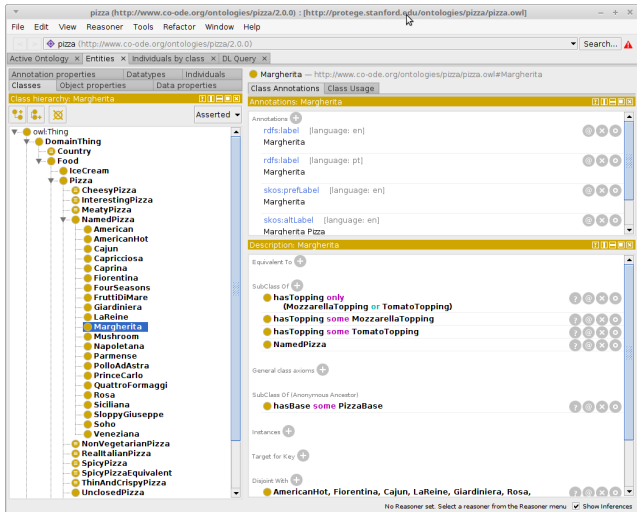
```
Pizza[owl:Class ?name, xsd:string ?label] :: {  
  Triple(?name, rdf:type, owl:Class),  
  SubClassOf(?label, p:Pizza),    not OK  
  Triple(?name, rdfs:label, ?label) } .
```

- Type checking consistent use

```
T[?x] :: { SubClassOf(?x, _:blank),    not OK  
  Pizza(?x, _:blank) } .
```

Pizza Ontology

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



OTTR: Example 2

```
Margherita ⊑ NamedPizza
Margherita ⊑ ∃ hasCountryOfOrigin.{Italy}
Margherita ⊑ ∀ hasTopping.(Tomato ⊔ Mozzarella)
Margherita ⊑ ∃ hasTopping.Tomato
Margherita ⊑ ∃ hasTopping.Mozzarella
```

template name type parameter name optional input list input

```
NamedPizza[ owl:Class ?Name, ? owl:NamedIndividual ?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

OTTR: Example 2

```
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:NamedIndividual ?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

OTTR: Example 2

```
Margherita ⊑ NamedPizza
Margherita ⊑ ∃ hasCountryOfOrigin. {Italy}
Margherita ⊑ ∀ hasTopping. (Tomato ⊔ Mozzarella)
Margherita ⊑ ∃ hasTopping. Tomato
Margherita ⊑ ∃ hasTopping. Mozzarella
```

```
NamedPizza(Margherita, Italy, (Tomato, Mozzarella))
NamedPizza(Grandiosa, none, (Tomato, Jarlsberg, Ham, SweetPepper))
```

template name type parameter name optional input list input

```
NamedPizza[ owl:Class ?Name, ? owl:NamedIndividual ?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

OTTR: Example 2

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, none, (Tomato, Jarlsberg, Ham, SweetPepper))

template name type parameter name optional input list input

NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?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

} .

OTTR: Example 2

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, none, (Tomato, Jarlsberg, Ham, SweetPepper))

template name type parameter name optional input
 NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?Country
 :: {
 SubClassOf(?Name, :NamedPizza),
 SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country),
 SubObjectAllValuesFrom(?Name, :hasTopping,
 ObjectUnionOf(_:b1, ?Toppings),
 cross | SubObjectSomeValuesFrom(?Name, :hasTopping, ?Toppings)
 }
 } .

list expansion marked for expansion

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

	A	B	
Grandiosa	1 #OTTR	prefix	
Grandiosa	2 p	http://example.com#	
Grandiosa	3 #OTTR	end	
Grandiosa	4		
Grandiosa	5 #OTTR	template	http://tpl.ottr.xyz/pizza/0.1/NamedPizza
Grandiosa	6 1	2	3
Grandiosa	7 iri	iri	iri+
Grandiosa	8 Pizza	Country	Toppings
Grandiosa	9 p:Veneziana	p:Italy	p:SultanaTopping p:OnionTopping p:TomatoTopping
Grandiosa	10 p:AmericanHot	p:America	p:HotGreenPepperTopping p:MozzarellaTopping p:TomatoTopping
Grandiosa	11 p:Margherita		p:TomatoTopping p:MozzarellaTopping
Grandiosa	12 p:FourSeasons		p:TomatoTopping p:AnchoviesTopping p:MozzarellaTopping
Grandiosa	13 p:Fiorentina		p:GarlicTopping p:SpinachTopping p:MozzarellaTopping
Grandiosa	14 p:PrinceCarlo		p:MozzarellaTopping p:ParmesanTopping p:LeekTopping
Grandiosa	15 p:LaReine		p:MushroomTopping p:MozzarellaTopping p:HamTopping
Grandiosa	16 p:American	p:America	p:PeperoniSausageTopping p:TomatoTopping p:MozzarellaTopping
Grandiosa	17 p:Caprina		p:SundriedTomatoTopping p:GoatsCheeseTopping p:MozzarellaTopping
Grandiosa	18 p:PolloAdAstra		p:SweetPepperTopping p:CajunSpiceTopping p:MozzarellaTopping
Grandiosa	19 p:Capricciosa		p:TomatoTopping p:PeperonataTopping p:HamTopping
Grandiosa	20 p:Mushroom		p:TomatoTopping p:MushroomTopping p:MozzarellaTopping
Grandiosa	21 p:FruttiDiMare		p:TomatoTopping p:GarlicTopping p:MixedSeafoodTopping
Grandiosa	22 p:SloppyGiuseppe		p:MozzarellaTopping p:GreenPepperTopping p:TomatoTopping
Grandiosa	23 p:Cajun		p:TobascoPepperSauce p:PrawnsTopping p:MozzarellaTopping
Grandiosa	24 p:Napoletana	p:Italy	p:CaperTopping p:TomatoTopping p:OliveTopping

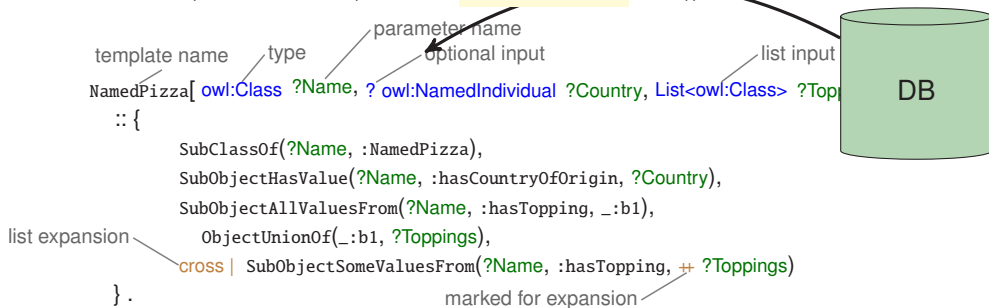
OTTR: Example 2

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, none, (Tomato, Jarlsberg, Ham, SweetPepper))

BOTTR map



Modelling with OTTR: Building template libraries

- Templates are published in template libraries using LOD principles

Contents

[index](#)
[Dependencies](#)
[Modules and packages](#)
[Templates](#)

http://tpl.ottr.xy/p/pizza/0.2/



[0](#)
[1](#)
[2](#)
[3](#)
[4](#)
[5](#)
[6](#)
[7](#)
[8](#)
[9](#)
[A](#)
[B](#)
[C](#)
[D](#)
[E](#)
[F](#)
[G](#)
[H](#)
[I](#)
[J](#)
[K](#)
[L](#)
[M](#)
[N](#)
[O](#)
[P](#)
[Q](#)
[R](#)
[S](#)
[T](#)
[U](#)
[V](#)
[W](#)
[X](#)
[Y](#)
[Z](#)

Contents

[index](#)
[Dependencies](#)
[Modules and packages](#)
[Templates](#)

Dependencies

[0](#)
[1](#)
[2](#)
[3](#)
[4](#)
[5](#)
[6](#)
[7](#)
[8](#)
[9](#)
[A](#)
[B](#)
[C](#)
[D](#)
[E](#)
[F](#)
[G](#)
[H](#)
[I](#)
[J](#)
[K](#)
[L](#)
[M](#)
[N](#)
[O](#)
[P](#)
[Q](#)
[R](#)
[S](#)
[T](#)
[U](#)
[V](#)
[W](#)
[X](#)
[Y](#)
[Z](#)

Modules and packages

[0](#)
[1](#)
[2](#)
[3](#)
[4](#)
[5](#)
[6](#)
[7](#)
[8](#)
[9](#)
[A](#)
[B](#)
[C](#)
[D](#)
[E](#)
[F](#)
[G](#)
[H](#)
[I](#)
[J](#)
[K](#)
[L](#)
[M](#)
[N](#)
[O](#)
[P](#)
[Q](#)
[R](#)
[S](#)
[T](#)
[U](#)
[V](#)
[W](#)
[X](#)
[Y](#)
[Z](#)

Templates

[0](#)
[1](#)
[2](#)
[3](#)
[4](#)
[5](#)
[6](#)
[7](#)
[8](#)
[9](#)
[A](#)
[B](#)
[C](#)
[D](#)
[E](#)
[F](#)
[G](#)
[H](#)
[I](#)
[J](#)
[K](#)
[L](#)
[M](#)
[N](#)
[O](#)
[P](#)
[Q](#)
[R](#)
[S](#)
[T](#)
[U](#)
[V](#)
[W](#)
[X](#)
[Y](#)
[Z](#)

Modelling with OTTR: Building template libraries

- Templates are published in template libraries using LOD principles
- Shared, reusable, (stable) resources

Contents

[Index](#)

© Signatures are expanded according to their complexity. Click arrow to expand class. Click node to display page or right window.

Items in red indicate which object has implemented a signature. A package is in red if one of its members implements for a particular purpose, otherwise it is in grey/blue pages.

Expand all Contract all

- [http://ottr.xyz](#)
 - [core](#)
 - [Resource](#)
 - [Description](#)
 - [Entity](#)
 - [Resource](#)
 - [URI](#)
 - [core](#)
 - [O.V.](#)
 - [Accotation](#)
 - [NameSpace](#)
 - [Partition](#)
 - [PuzzleSolving](#)
 - [uri](#)
 - [O.V.](#)
 - [Statement](#)
 - [StatementNode](#)
 - [uri](#)
 - [O.V.](#)
 - [Type](#)
 - [Label](#)
 - [ResourceDescription](#)
 - [TypedResourceDescription](#)
 - [O.V.](#)
 - [Domain](#)
 - [DomainRange](#)
 - [Label](#)
 - [Range](#)
 - [ResourceDescription](#)
 - [SubClassOf](#)
 - [TypedResourceDescription](#)

http://tpl.ottr.xyz/p/pizza/0.2/



Metrics

- Number of templates: 4
- Template domains: 1
 - [http://ottr.xyz](#)
- Template namespaces: 1
 - [http://ottr.xyz/p/pizza/0.2/](#)
- Root templates: 4
 - [http://ottr.xyz/p/pizza/0.2/DomainList](#)
 - [http://ottr.xyz/p/pizza/0.2/NamesPage](#)
 - [http://ottr.xyz/p/pizza/0.2/Partitions](#)
 - [http://ottr.xyz/p/pizza/0.2/PuzzleSolving](#)

Dependencies

Each graph is visualized using different layouts. Each node is linked to its documentation page. The colour of the node indicates its namespace.

Modules and packages

- Dependent between modules and packages:
- Hierarchical horizontal layout (y&d)

Packages

Expand all Contract all

- [http://ottr.xyz](#)
 - [core](#)
 - [core:resource](#)
 - [core:uri](#)
 - [core:statement](#)

Contents

[Index](#)

© Signatures are expanded according to their complexity. Click arrow to expand class. Click node to display page or right window.

Items in red indicate which object has implemented a signature. A package is in red if one of its members implements for a particular purpose, otherwise it is in grey/blue pages.

Expand all Contract all

- [http://ottr.xyz](#)
 - [core](#)
 - [Resource](#)
 - [Description](#)
 - [Entity](#)
 - [Resource](#)
 - [URI](#)
 - [core](#)
 - [O.V.](#)
 - [Accotation](#)
 - [NameSpace](#)
 - [Partition](#)
 - [PuzzleSolving](#)
 - [uri](#)
 - [O.V.](#)
 - [Statement](#)
 - [StatementNode](#)
 - [uri](#)
 - [O.V.](#)
 - [Type](#)
 - [Label](#)
 - [ResourceDescription](#)
 - [TypedResourceDescription](#)
 - [O.V.](#)
 - [Domain](#)
 - [DomainRange](#)
 - [Label](#)
 - [Range](#)
 - [ResourceDescription](#)
 - [SubClassOf](#)
 - [TypedResourceDescription](#)

http://tpl.ottr.xyz/p/pizza/0.2/



Metrics

- Number of templates: 4
- Template domains: 1
 - [http://ottr.xyz](#)
- Template namespaces: 1
 - [http://ottr.xyz/p/pizza/0.2/](#)
- Root templates: 4
 - [http://ottr.xyz/p/pizza/0.2/DomainList](#)
 - [http://ottr.xyz/p/pizza/0.2/NamesPage](#)
 - [http://ottr.xyz/p/pizza/0.2/Partitions](#)
 - [http://ottr.xyz/p/pizza/0.2/PuzzleSolving](#)

Dependencies

Each graph is visualized using different layouts. Each node is linked to its documentation page. The colour of the node indicates its namespace.

Modules and packages

- Dependent between modules and packages:
- Hierarchical horizontal layout (y&d)

Packages

Expand all Contract all

- [http://ottr.xyz](#)
 - [core](#)
 - [core:resource](#)
 - [core:uri](#)
 - [core:statement](#)

Modelling with OTTR: Building template libraries

- Templates are published in template libraries using LOD principles
- Shared, reusable, (stable) resources
- `tpl.ottr.xyz`: ≈ 200 OWL, RDFS, RDF + example/project templates

[illegible]

Modelling with OTTR: Building template libraries

- Templates are published in template libraries using LOD principles
- Shared, reusable, (stable) resources
- `tpl.ottr.xyz`: ≈ 200 OWL, RDFS, RDF + example/project templates
- Example: `https://tpl.ottr.xyz/p/pizza/0.2/NamedPizza`

[illegible]

Modelling with OTTR: Building template libraries

- Templates are published in template libraries using LOD principles
- Shared, reusable, (stable) resources
- `tpl.ottr.xyz`: ≈ 200 OWL, RDFS, RDF + example/project templates
- Example: `https://tpl.ottr.xyz/p/pizza/0.2/NamedPizza`
- Library management procedures defined (versioning, statuses, publishing)
- Maintenance tools available (e.g., duplication, overlap)

Contents

1. Signatures are organized according to their namespace. Click arrow to expand list. Click text to display page in right window. Items in the list with a colored box represent a namespace. A package is a set of templates constructed for a particular purpose. Click on page at a specific project.

Expanded list: Contents list

- `http://tpl.ottr.xyz/`
 - `owl`
 - `Annotation`
 - `NamedPizza`
 - `Partition`
 - `PizzaTopping`
 - `rdf`
 - `Statement`
 - `StatementTriple`
 - `Type`
 - `rdfs`
 - `Label`
 - `ResourceDescription`
 - `TypeResourceDescription`
 - `uri`
 - `Domain`
 - `DomainRange`
 - `Label`
 - `Range`
 - `ResourceDescription`
 - `SubClassOf`
 - `TypeResourceDescription`
- `packages`
 - Expanded list: Contents list
 - `http://tpl.ottr.xyz/`
 - `owl`
 - `Annotation`
 - `NamedPizza`
 - `Partition`
 - `PizzaTopping`
 - `rdf`
 - `Statement`
 - `StatementTriple`
 - `Type`
 - `rdfs`
 - `Label`
 - `ResourceDescription`
 - `TypeResourceDescription`
 - `uri`
 - `Domain`
 - `DomainRange`
 - `Label`
 - `Range`
 - `ResourceDescription`
 - `SubClassOf`
 - `TypeResourceDescription`

Metrics

<http://tpl.ottr.xyz/p/pizza/0.2/>

- Number of templates: 4
- Template domains: 1
 - `http://tpl.ottr.xyz/`
- Template namespaces: 1
 - `http://tpl.ottr.xyz/p/pizza/0.2/`
- Root templates: 4
 - `http://tpl.ottr.xyz/p/pizza/0.2/NamedPizza`
 - `http://tpl.ottr.xyz/p/pizza/0.2/NamedPizza`
 - `http://tpl.ottr.xyz/p/pizza/0.2/Partition`
 - `http://tpl.ottr.xyz/p/pizza/0.2/PizzaTopping`



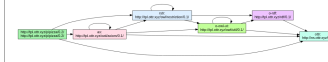
Dependencies

Each graph is visualized using different layouts. Each node is linked to its documentation page. The color of the node indicates its namespace.

Modules and packages

Each graph is visualized using different layouts. Each node is linked to its documentation page. The color of the node indicates its namespace.

- Hierarchical horizontal layout (hdb)



- Hierarchical vertical layout (vdb)
- Spring model layout (msb)
- Spring model layout (hsp)
- Nested layout (hnd)
- Circular layout (cbo)

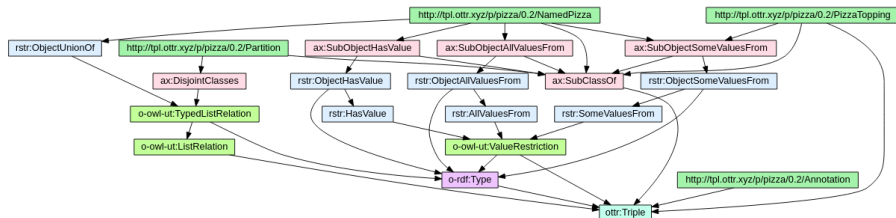
Templates

Each graph is visualized using different layouts. Each node is linked to its documentation page. The color of the node indicates its namespace.

- Hierarchical horizontal layout (hdb)

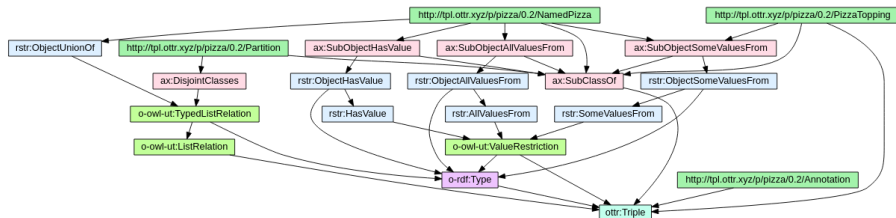
Modelling with OTTR: Top-down methodology

- Top-down modelling with different abstraction levels:
System → Domain → Logical → Utility → Base



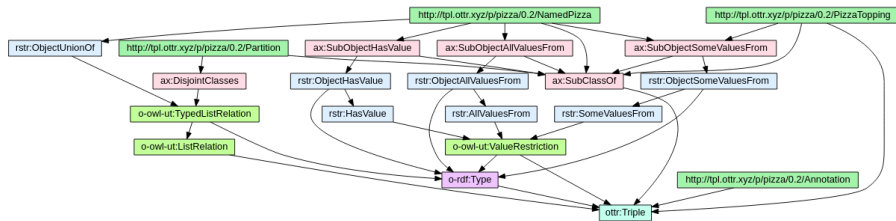
Modelling with OTTR: Top-down methodology

- Top-down modelling with different abstraction levels:
System → Domain → Logical → Utility → Base
- Levels to match competence, responsibility and availability



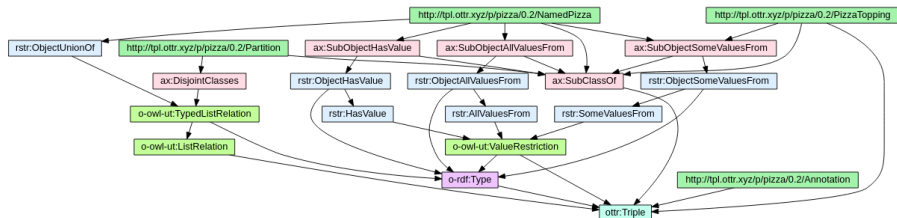
Modelling with OTTR: Top-down methodology

- Top-down modelling with different abstraction levels:
System → Domain → Logical → Utility → Base
- Levels to match competence, responsibility and availability
- Separation of concerns



Modelling with OTTR: Top-down methodology

- Top-down modelling with different abstraction levels:
System → Domain → Logical → Utility → Base
- Levels to match competence, responsibility and availability
- Separation of concerns
- Publish template library with each vocabulary — a vocabulary API!



Summing up: Why use OTTR?

- Represent, share and use your modelling patterns. (Patterns for the win!)



Summing up: Why use OTTR?

- Represent, share and use your modelling patterns. (Patterns for the win!)
- Consistent and correct modelling (signature and type checking)



Summing up: Why use OTTR?

- Represent, share and use your modelling patterns. (Patterns for the win!)
- Consistent and correct modelling (signature and type checking)
- Efficiency



Summing up: Why use OTTR?

- Represent, share and use your modelling patterns. (Patterns for the win!)
- Consistent and correct modelling (signature and type checking)
- Efficiency
- Usability(?)



Summing up: Why use OTTR?

- Represent, share and use your modelling patterns. (Patterns for the win!)
- Consistent and correct modelling (signature and type checking)
- Efficiency
- Usability(?)
- Organise your patterns in a library (modular templates)



Summing up: Why use OTTR?

- Represent, share and use your modelling patterns. (Patterns for the win!)
- Consistent and correct modelling (signature and type checking)
- Efficiency
- Usability(?)
- Organise your patterns in a library (modular templates)
- Visit <http://ottr.xyz> for more information, including primer, interactive examples, tutorials and videos.



Summing up: Why use OTTR?

- Represent, share and use your modelling patterns. (Patterns for the win!)
- Consistent and correct modelling (signature and type checking)
- Efficiency
- Usability(?)
- Organise your patterns in a library (modular templates)
- Visit <http://ottr.xyz> for more information, including primer, interactive examples, tutorials and videos.

Thanks!

