Tutorial: Pattern-based knowledge base construction

Martin G. Skjæveland Leif Harald Karlsen Daniel P. Lupp

We start 13:05 CET.

While you wait, please fill out this form:
`http://tinyurl.com/ottr-iswc2020`

Who are we?

Martin

Leif Harald

Daniel

Martin G. Skjæveland, CS, researcher UiO. Leader of OTTR project. Ontology engineering, Ontology-based systems. Industrial experience from DNV-GL, and from managing industrial research projects.

Leif Harald Karlsen, CS, senior lecturer at UiO. Knowledge representation, semantic technologies, databases and programming languages.

Daniel Lupp, CS, postdoc at UiO. Ontology engineering, ontology-based data integration and logic programming.

Tutorial scope

- Languages and tools for pattern-driven ontology development
- and its benefits: better efficiency, quality for constructing, maintenance and use.
- Our solution and implementation: Reasonable Ontology Templates (OTTR)
- Building and using shared template libraries
- Use cases

Tutorial scope

- Languages and tools for pattern-driven ontology development
- and its benefits: better efficiency, quality for constructing, maintenance and use.
- Our solution and implementation: Reasonable Ontology Templates (OTTR)
- Building and using shared template libraries
- Use cases

Not about:

- Introducing semantic technologies: RDF, OWL, SPARQL, ...
- Concrete ontology modelling problems
- The benefits of ontologies and semantic technologies

Tutorial plan

13:00–13:30	**Introduction**
13:30–13:40	*Break*
13:40–14:25	**OTTR Fundamentals**
14:25–14:35	*Break*
14:35–14:50	**The Core OTTR Template Library**
14:50–15:35	**Modeling in Practice**
15:35–15:45	*Break*
15:45–16:00	**Summary and questions**

Tutorial plan

13:00–13:30	**Introduction**
13:30–13:40	*Break*
13:40–14:25	**OTTR Fundamentals**
14:25–14:35	*Break*
14:35–14:50	**The Core OTTR Template Library**
14:50–15:35	**Modeling in Practice**
15:35–15:45	*Break*
15:45–16:00	**Summary and questions**

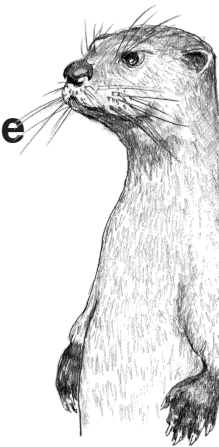
- The schedule is tentative
- Ask questions during the presentations in the chat
- Each section will end with an open discussion

Who are you?

Let’s look at the answers to the form you filled out!

Tutorial: Pattern-based knowledge base construction

Martin G. Skjæveland Leif Harald Karlsen Daniel P. Lupp



Introduction

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

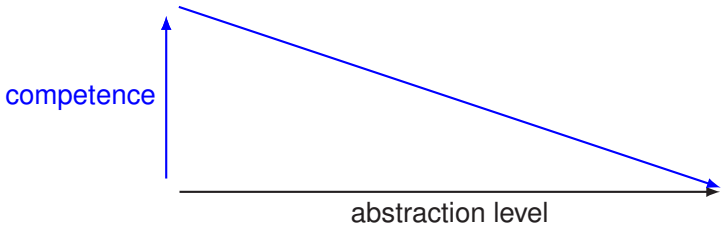
Problem:

- 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

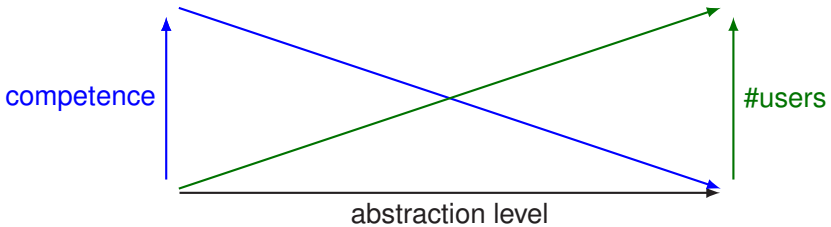
We need:

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

Abstractions



Abstractions



Abstractions

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!")
```

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!")
```

- Hide complexity

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!")
```

- Hide complexity
- Separation of concerns

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!")
```

- Hide complexity
- Separation of concerns
- Support different users

Abstractions

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

Abstractions

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

Abstractions

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”

Abstractions

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

Abstractions

Manchester OWL

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

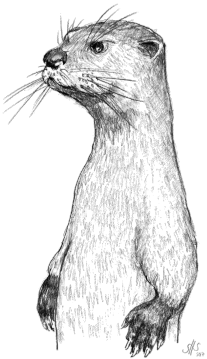
?

- Hide complexity
- Separation of concerns
- Support different users

- Still low-level OWL “coding”
- Impossible to not care about logic
- No concept of pattern

Reasonable Ontology Templates (OTTR)

- Template mechanism for encoding ontology patterns
- 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 .
```

```
pz: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 .
```

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

```
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) .
```

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 .
```

```
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) .
```

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 .
```

```
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) .
```

Reasonable Ontology Templates (OTTR): Example

```
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) .
```

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)

```
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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- 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) .
```

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 encodings
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format

- Substitution
- Macro expansion

$$
\begin{aligned}
&\texttt{ax:SubClassOf}[\texttt{p:Margherita}, \texttt{p:Pizza}] :: \{ \\
&\quad \texttt{ottr:Triple}(\texttt{p:Margherita}, \texttt{rdfs:subClassOf}, \texttt{p:Pizza}) \\
&\} . \\
&\texttt{pz:Pizza}[\texttt{p:Margherita}, \texttt{"Margherita"@it}] :: \{ \\
&\quad \texttt{ottr:Triple}(\texttt{p:Margherita}, \texttt{rdf:type}, \texttt{owl:Class}), \\
&\quad \texttt{ottr:Triple}(\texttt{p:Margherita}, \texttt{rdfs:subClassOf}, \texttt{p:Pizza}), \\
&\quad \texttt{ottr:Triple}(\texttt{p:Margherita}, \texttt{rdfs:label}, \texttt{"Margherita"@it}) \\
&\} . \\
&\texttt{pz:Pizza}(\texttt{p:Margherita}, \texttt{"Margherita"@it}) . \\
&\texttt{pz:Pizza}(\texttt{p:Hawaii}, \texttt{"Hawaii"@en}) . \\
&\texttt{pz:Pizza}(\texttt{p:Grandiosa}, \texttt{"Grandiosa"@no}) .
\end{aligned}
$$

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 encodings
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format

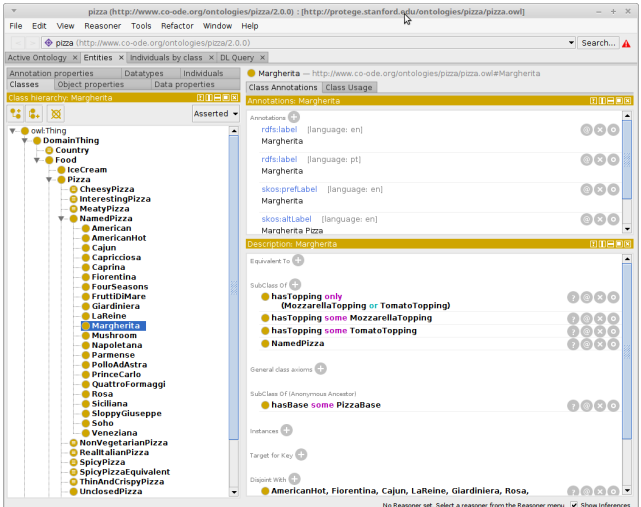
- Substitution
- Macro expansion

```
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) .
```

Pizza Ontology

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



OTTR Serialisations

stOTTR: Easy to read and write

```
pz:NamedPizza[ owl:Class  ?Name, ? owl:NamedIndividual  ?Country, List<owl:Class>  ?Toppings] :: {
        ax:SubClassOf(?Name, p:NamedPizza),
        ax:SubObjectHasValue(?Name, p:hasCountryOfOrigin, ?Country),
        ax:SubObjectAllValuesFrom(?Name, p:hasTopping, _:b1),
          rstr:ObjectUnionOf(_:b1, ?Toppings),
        cross | ax:SubObjectSomeValuesFrom(?Name, p:hasTopping, ++?Toppings)
} .


pz:NamedPizza(p:Margherita, p:Italy, (p:Tomato, p:Mozzarella)) .
pz:NamedPizza(p:Grandiosa, none, (p:Tomato, p:Jarlsberg, p:Ham, p:SweetPepper)) .
```

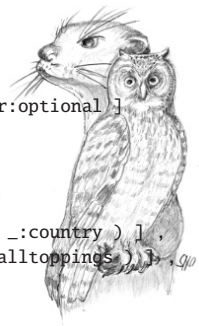
OTTR Serialisations

wOTTR: OWL vocabulary for semantic web use

```
pz: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 ] ) ] .
```



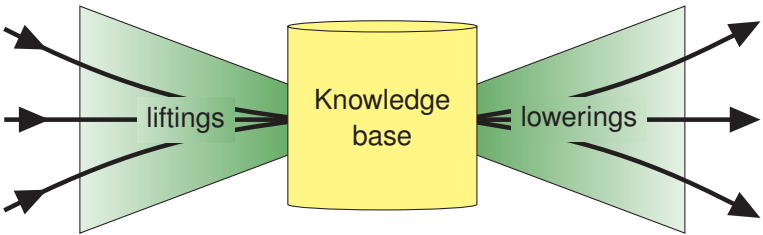
tabOTTR: tabular format for bulk template instances

9			
10	#OTTR	template	http://draft.ottr.xyz/pizza/NamedPizza
11	Pizza	Country	Toppings
12	1	2	3
13	iri	iri	iri+
14	p:Veneziana	p:Italy	p:OnionTopping | p:MozzarellaTopping | p:Caper
15	p:AmericanH	p:America	p:JalapenoPepperTopping | p:MozzarellaTopping
16	p:Margherita		p:MozzarellaTopping | p:TomatoTopping
17	p:FourSeasons		p:TomatoTopping | p:PeperoniSausageTopping |
18	p:Fiorentina		p:TomatoTopping | p:GarlicTopping | p:Parmesa
19	p:PrinceCarlo		p:LeekTopping | p:RosemaryTopping | p:Mozzar
20	p:LaReine		p:MushroomTopping | p:HamTopping | p:Mozzar
21	p:American	p:America	p:TomatoTopping | p:MozzarellaTopping | p:Pep
22	p:Caprina		p:MozzarellaTopping | p:TomatoTopping | p:Goa
23	p:PolloAdAstra		p:TomatoTopping | p:RedOnionTopping | p:Mozz
24	p:Capricciosa		p:HamTopping | p:MozzarellaTopping | p:OliveTo

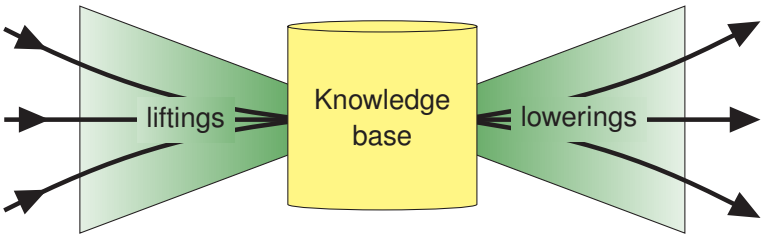
bOTTR: ETL mappings to OTTR instances

```
[] a ottr:InstanceMap ;
    ottr:template ottr:Triple ;
    ottr:query """
        SELECT ?s ?p ?o
        WHERE { ?s ?p ?o }
        LIMIT 5
    """ ;
    ottr:source
[ a ottr:SPARQLEndpointSource ;
  ottr:sourceURL "http://dbpedia.org/sparql" ] .
```

Vision: Template-driven methodology



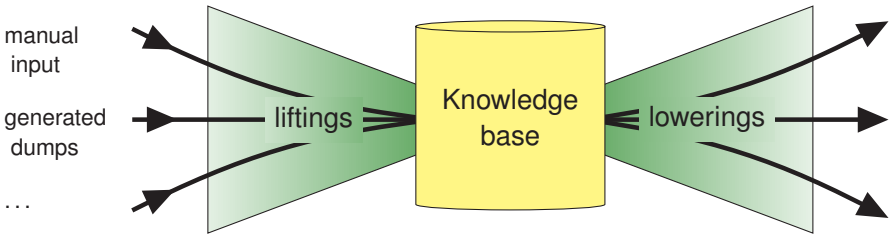
Vision: Template-driven methodology



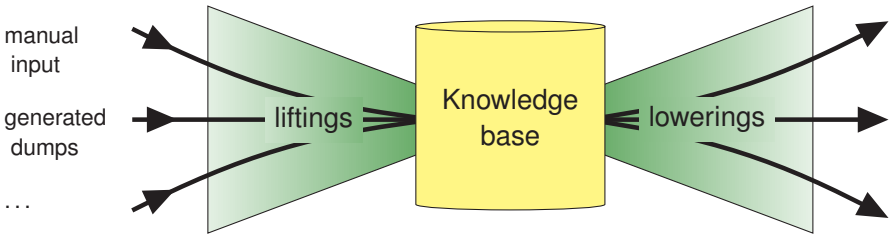
core temp. + sem. tech

ontology expert

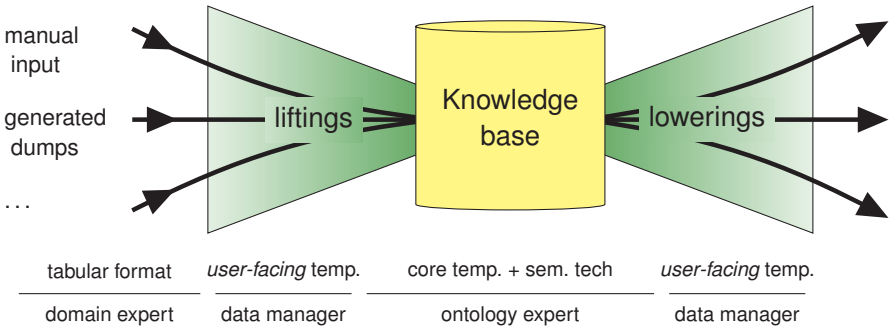
Vision: Template-driven methodology



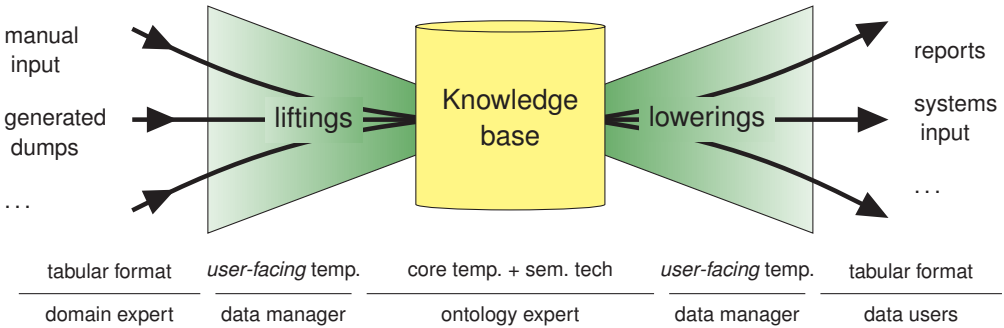
Vision: Template-driven methodology



Vision: Template-driven methodology



Vision: Template-driven methodology



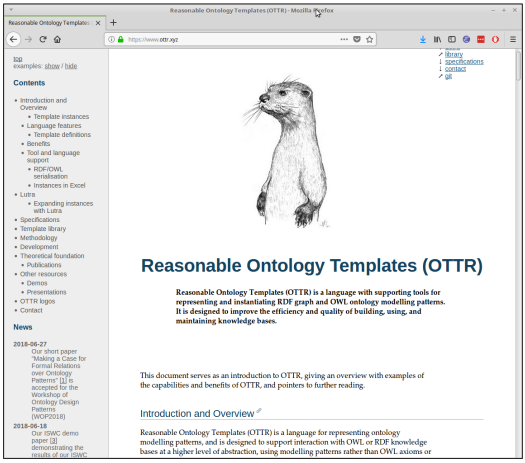
Future work: Templates as queries

Templates as queries, query for instances of pattern

```
SELECT  *
  { ?param1   rdfs:subClassOf     p:NamedPizza ,
      [ owl:onProperty      p:hasTopping ;
        owl:someValuesFrom  param3item ;
        rdf:type            owl:Restriction ] ,
      [ owl:allValuesFrom   [ owl:unionOf  ?param3 ;
                              rdf:type     owl:Class ] ;
        owl:onProperty      p:hasTopping ;
        rdf:type            owl:Restriction ]
  OPTIONAL {
    ?param1   rdfs:subClassOf  [
        owl:hasValue        ?param2 ;
        owl:onProperty      p:hasCountryOfOrigin ;
        rdf:type            owl:Restriction ]
  }
    ?param3  (rdf:rest)*/rdf:first ?param3item
}
```

Resources: `ottr.xyz`

- Documentation
- Specifications
- Primer
- Tools: Lutra and WebLutra
- Template library
- Papers, Demos, Tutorials, Slides
- Git

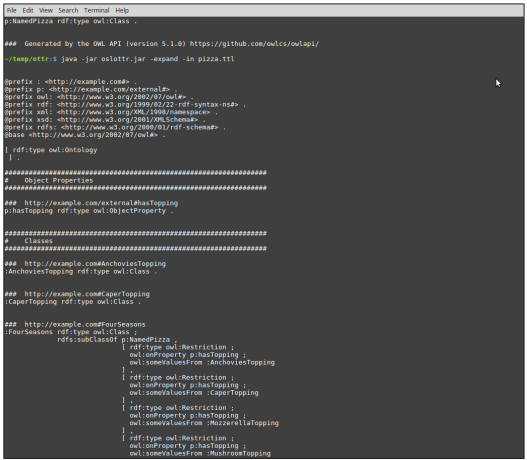


Resources: Lutra CLI

- Java executable CLI
- Open source
- Industry applications

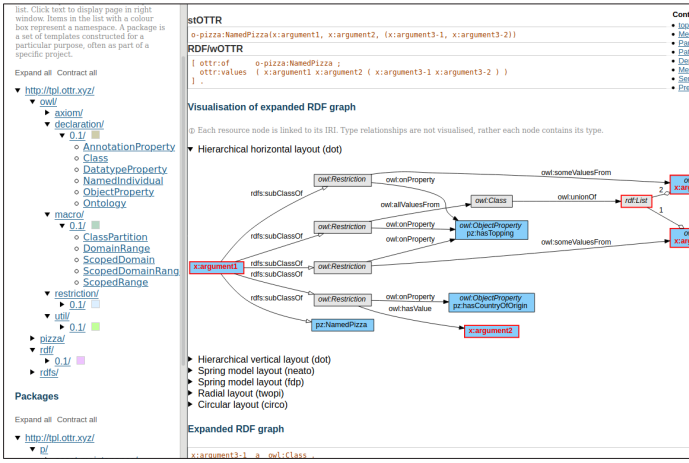
- Expand instances
- Type-checking, cycle detection, ...
- API

- WebLutra: `http://weblutra.ottr.xyz`



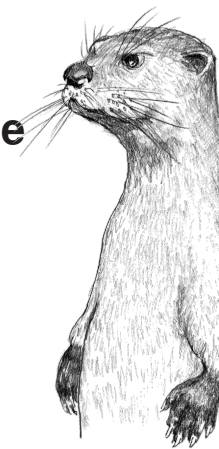
Resources: Core OTTR Template Library

- `http://tpl.ottr.xyz`
- RDF, RDFS, OWL, example
- Git
- open, shared, communtity managed
- Management guidelines



Tutorial: Pattern-based knowledge base construction

Martin G. Skjæveland Leif Harald Karlsen Daniel P. Lupp



OTTR Template basics

Plan

- We will now look at OTTR in more detail
- See some of the language features of OTTR
- Will use the stOTTR serialisation for examples
- These features ensure:
  - Correctness of modelling
  - Uniform modelling
  - Compact definitions
  - Easy maintenance
  - Scalability
  - Control
- Many of the features taken from software engineering

Templates and instances

A template instance consists of:

```
ax:SubClassOf(p:Margherita, p:NamedPizza) .
pz:Pizza(p:Hawaii, "Hawaii"@en) .
pz:Pizza(p:Grandiosa, "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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template

```
ax:SubClassOf(p:Margherita, p:NamedPizza) .
pz:Pizza(p:Hawaii, "Hawaii"@en) .
pz:Pizza(p:Grandiosa, "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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()

```
ax:SubClassOf(p:Margherita, p:NamedPizza) .
pz:Pizza(p:Hawaii, "Hawaii"@en) .
pz:Pizza(p:Grandiosa, "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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()
- Terminated by a dot

```
ax:SubClassOf(p:Margherita, p:NamedPizza) .
pz:Pizza(p:Hawaii, "Hawaii"@en) .
pz:Pizza(p:Grandiosa, "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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()
- Terminated by a dot

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

A template definition consists of:

```
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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()
- Terminated by a dot

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

A template definition consists of:

- A name (QName or IRI)

```
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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()
- Terminated by a dot

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

A template definition consists of:

- A name (QName or IRI)
- A list of parameter declarations surrounded by []

```
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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()
- Terminated by a dot

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

A template definition consists of:

- A name (QName or IRI)
- A list of parameter declarations surrounded by []
- A body containing template instances surrounded by {}

```
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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()
- Terminated by a dot

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

A template definition consists of:

- A name (QName or IRI)
- A list of parameter declarations surrounded by []
- A body containing template instances surrounded by {}
- Terminated by a dot

```
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)
} .
```

Templates and instances

A template instance consists of:

- A reference to a template
- A list of arguments surrounded by ()
- Terminated by a dot

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

A template definition consists of:

- A name (QName or IRI)
- A list of parameter declarations surrounded by []
- A body containing template instances surrounded by {}
- Terminated by a dot

```
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)
} .
```

We'll now look at more features!

Template signatures

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

  `pz:NamedPizza`[?Name, ?Country, ?Toppings] .

Template signatures

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

  ```
  pz:NamedPizza[?Name, ?Country, ?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

```
ottr:Triple[?subject, ?predicate, ?object] :: BASE .
```

Base templates

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

  `ottr:Triple`[?subject, ?predicate, ?object] :: `BASE` .

- A `ottr:Triple`-instance represents an RDF triple, e.g.:

  `ottr:Triple`(`ex:martin`, `ex:knows`, `ex:daniel`) ⇝ `ex:martin ex:knows ex:daniel` .

Base templates

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

  `ottr:Triple`[?subject, ?predicate, ?object] :: `BASE` .

- A `ottr:Triple`-instance represents an RDF triple, e.g.:

  `ottr:Triple`(`ex:martin`, `ex:knows`, `ex:daniel`) ⇝ `ex:martin ex:knows ex:daniel` .

- Translation done by implementation

Base templates

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

  `ottr:Triple`[?subject, ?predicate, ?object] :: `BASE` .

- A `ottr:Triple`-instance represents an RDF triple, e.g.:

  `ottr:Triple`(`ex:martin`, `ex:knows`, `ex:daniel`) ⇝ `ex:martin ex:knows ex:daniel` .

- Translation done by implementation
- Other base templates possible: E.g. `Quad`, OWL axioms, etc.

Base templates

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

  `ottr:Triple`[?subject, ?predicate, ?object] :: `BASE` .

- A `ottr:Triple`-instance represents an RDF triple, e.g.:

  `ottr:Triple`(`ex:martin`, `ex:knows`, `ex:daniel`) ⇝ `ex:martin ex:knows ex:daniel` .

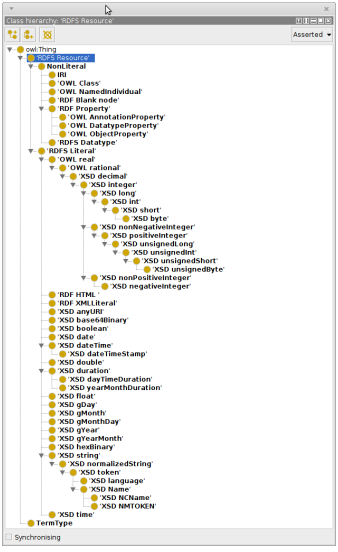
- Translation done by implementation
- Other base templates possible: E.g. `Quad`, OWL axioms, etc.
- Note: “Templates all the way down”

Exercises: Making templates and instance

- We will go to the OTTR Primer at `ottr.xyz`
  - Same as `https://tinyurl.com/ottr-basics`
- Look at some of the exercises in sections 2, 3, and 4

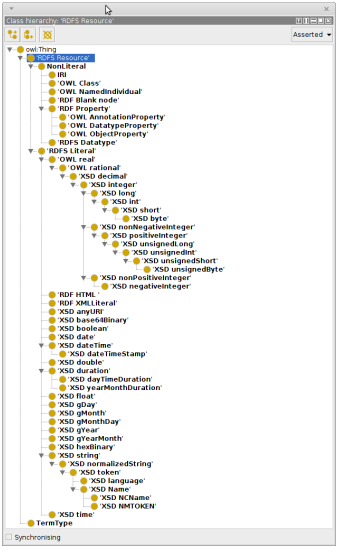
Types

- Specify permissible arguments and consistent use of resources



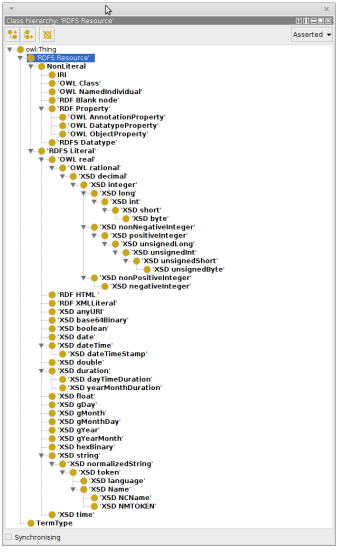
Types

- Specify permissible arguments and consistent use of resources
- Base-types limited to classes and datatypes defined in XSD, RDF, RDFS, and OWL
  - `xsd:int`, `owl:ObjectProperty`, `rdfs:Resource`, ...
  - `rdfs:Resource` is default and most general
  - We have added `ottr:IRI`, the type of non-literals



Types

- Specify permissible arguments and consistent use of resources
- Base-types limited to classes and datatypes defined in XSD, RDF, RDFS, and OWL
  - `xsd:int`, `owl:ObjectProperty`, `rdfs:Resource`, ...
  - `rdfs:Resource` is default and most general
  - We have added `ottr:IRI`, the type of non-literals
- Also supports lists and non-empty lists (e.g. NEList⟨List⟨xsd:string⟩⟩)



Types

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

Types

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

Types

- Similar to types in programming languages (e.g. Java)
- But adapted to ontologies
- Types put before the variable in parameter declarations, e.g.:

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

- Type checking instances

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

ax:SubClassOf(p:Margherita, p:Pizza) .
ax:SubClassOf("Margherita", p:Pizza) .
```

- Type checking instances

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

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

Types
Examples

- Type checking instances

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

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

- Type checking template definitions

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

Types
Examples

- Type checking instances

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

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

- Type checking template definitions

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

Types – Consistent use

Examples

r:InstanceOf[owl:NamedIndividual ?elem, owl:Class ?class] .

r:InstanceOf(_:margherita, p:Pizza)

r:InstanceOf(p:mypizza, _:margherita)

Types – Consistent use

Examples

r:InstanceOf[owl:NamedIndividual ?elem, owl:Class ?class] .

r:InstanceOf(_:margherita, p:Pizza)

r:InstanceOf(p:mypizza, _:margherita) OK? (depending on type hierarchy)

Types – Consistent use

Examples

`r:InstanceOf[owl:NamedIndividual ?elem, owl:Class ?class] .`

`r:InstanceOf(_:margherita, p:Pizza)`

`r:InstanceOf(p:mypizza, _:margherita)` OK? (depending on type hierarchy)

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

`r:Label(p:mypizza, "A pizza")`

Types – Consistent use

Examples

`r:InstanceOf`[owl:NamedIndividual ?elem, owl:Class ?class] .

`r:InstanceOf`(`_:margherita`, `p:Pizza`)

`r:InstanceOf`(`p:mypizza`, `_:margherita`) OK? (depending on type hierarchy)

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

`r:Label`(`p:mypizza`, `"A pizza"`) OK

Types – Consistent use

Examples

`r:InstanceOf[owl:NamedIndividual ?elem, owl:Class ?class] .`

`r:InstanceOf(_:margherita, p:Pizza)`

`r:InstanceOf(p:mypizza, _:margherita)` OK? (depending on type hierarchy)

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

`r:Label(p:mypizza, "A pizza")` OK

`r:InstanceOf(_:hawaii, p:Pizza)`

`r:Label(p:mypizza, _:hawaii)`

Types – Consistent use

Examples

`r:InstanceOf[owl:NamedIndividual ?elem, owl:Class ?class] .`

`r:InstanceOf(_:margherita, p:Pizza)`

`r:InstanceOf(p:mypizza, _:margherita)` OK? (depending on type hierarchy)

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

`r:Label(p:mypizza, "A pizza")` OK

`r:InstanceOf(_:hawaii, p:Pizza)`

`r:Label(p:mypizza, _:hawaii)` not OK

Types – Motivation

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

Types – Motivation

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

Types – Motivation

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

Types – Motivation

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

Types – Motivation

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

Types – Motivation

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

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

ottr:Triple(_:hammer, :hasLength, 5)
ottr:Triple(p:mypizza, _:hasLength, 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.

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

ottr:Triple(_:hammer, :hasLength, 5)        OK
ottr:Triple(p:mypizza, _:hasLength, 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.

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

ottr:Triple(_:hammer, :hasLength, 5)        OK
ottr:Triple(p:mypizza, _:hasLength, 5)      not OK
```

- 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.

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

ottr:Triple(_:hammer, :hasLength, 5)        OK
ottr:Triple(p:mypizza, _:hasLength, 5)      not OK
```

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

Exercises: Types and nonblanks

- We will go to the OTTR Primer at `ottr.xyz`
  - Same as `https://tinyurl.com/ottr-basics`
- Look at some of the exercises in sections 5

OTTR Overview

- 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 ($\star$)
  - Default and optional values ($\star$)
  - Formal analysis and manipulation of sets of templates ($\star$)
  - Annotations (instances) for documenting templates ($\star$)

OTTR Overview

- 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 ($\star$)
  - Default and optional values ($\star$)
  - Formal analysis and manipulation of sets of templates ($\star$)
  - Annotations (instances) for documenting templates ($\star$)
- The features are based on well-known principles in programming language design

OTTR Overview

- 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 ($\star$)
  - Default and optional values ($\star$)
  - Formal analysis and manipulation of sets of templates ($\star$)
  - Annotations (instances) for documenting templates ($\star$)
- 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

Discussion/Future work/Known issues

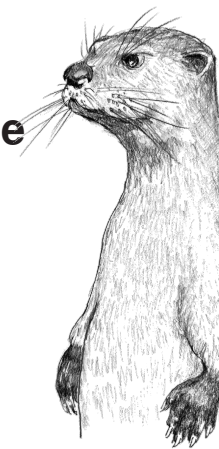
Will support:

- User-defined type systems
- Improved type system
  - `ottr:IRI` vs. `ottr:List`
  - blanks/non-blank
- Visual language for OTTR (creation/visualization)
- Language Server Protocol implementation for OTTR

Tutorial: Pattern-based knowledge base construction

Martin G. Skjæveland Leif Harald Karlsen Daniel P. Lupp

OTTR Template Libraries



Design Principles

- Libraries of best-practice templates created and curated by experts
- API for ontology development and interaction
- Instances $\Leftrightarrow$ their expansion

Design Principles

- Libraries of best-practice templates created and curated by experts
- API for ontology development and interaction
- Instances $\Leftrightarrow$ their expansion
- Published templates do not change
- High availability

Design Principles

- Libraries of best-practice templates created and curated by experts
- API for ontology development and interaction
- Instances $\Leftrightarrow$ their expansion
- Published templates do not change
- High availability
- Easy to navigate by humans
- Easy to maintain by admin

OTTR Template Libraries

Design Principles

- Libraries of best-practice templates created and curated by experts
- API for ontology development and interaction
- Instances $\Leftrightarrow$ their expansion
- Published templates do not change
- High availability
- Easy to navigate by humans
- Easy to maintain by admin

Core OTTR template library

- `http://tpl.ottr.xyz`
- $\approx$ 200 templates (2020)
- *Logical* templates:
  OWL, RDFS and RDF patterns

OTTR Template Libraries

Design Principles

- Libraries of best-practice templates created and curated by experts
- API for ontology development and interaction
- Instances $\Leftrightarrow$ their expansion
- Published templates do not change
- High availability
- Easy to navigate by humans
- Easy to maintain by admin

Core OTTR template library

- `http://tpl.ottr.xyz`
- $\approx$ 200 templates (2020)
- *Logical* templates:
  OWL, RDFS and RDF patterns
- Natural starting point for
  - template-based ontology development
  - new template libraries

OTTR Template Library: Structured along multiple dimensions

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*
- *Packages*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*
- *Utility*
- *Logical*
- *Domain*
- *System*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*
- *Logical*
- *Domain*
- *System*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*
- *Domain*
- *System*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*
- *System*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*
- *Draft*
- *Candidate*
- *Recommended*
- *Deprecated*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*
- *Candidate*
- *Recommended*
- *Deprecated*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*
- *Recommended*
- *Deprecated*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*
- *Deprecated*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*: well-integrated
- *Deprecated*

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*: well-integrated
- *Deprecated*: use discouraged

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*: well-integrated
- *Deprecated*: use discouraged

Version: change and maturity

- Semantic versioning: x.y.z
- *Patch* (x.y.**z**)
- *Minor* (x.**y**.z)
- *Major* (**x**.y.z)

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*: well-integrated
- *Deprecated*: use discouraged

Version: change and maturity

- Semantic versioning: x.y.z
- *Patch* (x.y.**z**), “unnoticeable” change
- *Minor* (x.**y**.z)
- *Major* (**x**.y.z)

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use
- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level
- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement
- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*: well-integrated
- *Deprecated*: use discouraged

Version: change and maturity
- Semantic versioning: x.y.z
- *Patch* (x.y.**z**), “unnoticeable” change
- *Minor* (x.**y**.z), backwards compatible
- *Major* (**x**.y.z)

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use

- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level

- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement

- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*: well-integrated
- *Deprecated*: use discouraged

Version: change and maturity

- Semantic versioning: x.y.z
- *Patch* (x.y.**z**), “unnoticeable” change
- *Minor* (x.**y**.z), backwards compatible
- *Major* (**x**.y.z), backwards **in**compatible

OTTR Template Library: Structured along multiple dimensions

Group: Vocabulary use
- *Modules*: RDF, RDFS, OWL vocabs
- *Packages*: project-specific

Layer: abstraction/granularity level
- *Base*: `Triple`
- *Utility*: `ListRelation`
- *Logical*: `ClassPartition`
- *Domain*: `NamedPizza`
- *System*: `MyPizzaDBFormat`

Status: maturity and endorsement
- *Incomplete*: parsable, WIP
- *Draft*, correct, unstable
- *Candidate*: stable, documented
- *Recommended*: well-integrated
- *Deprecated*: use discouraged

Version: change and maturity
- Semantic versioning: x.y.z
- *Patch* (x.y.**z**), "unnoticeable" change
- *Minor* (x.**y**.z), backwards compatible
- *Major* (**x**.y.z), backwards **in**compatible

IRIs

`http://tpl.ottr.xyz/[module/name]/[major.minor version]/[template name]`

`http://tpl.ottr.xyz/p/[package/name]/[major.minor version]/[template name]`

Documentation: annotation instances and docTTR

- Template metadata is captured by *annotation instances*

```
@@o-docttr:Version(
    id = ax:SubClassOf,
    status = ottr:draft,
    version = "0.2.1",
    previousVersion = <http://tpl.ottr...
    nextVersion = none)
ax:SubClassOf[ ?sub, ?super ] :: {
  ottr:Triple(?sub, rdfs:subClassOf, ?super)
}.
```

Documentation: annotation instances and docTTR

- Template metadata is captured by *annotation instances*

```
@@o-docttr:Version(
    id = ax:SubClassOf,
    status = ottr:draft,
    version = "0.2.1",
    previousVersion = <http://tpl.ottr...
    nextVersion = none)
ax:SubClassOf[ ?sub, ?super ] :: {
  ottr:Triple(?sub, rdfs:subClassOf, ?super)
}.
```

- *docTTR*: documentation tool and format for OTTR templates
  - Generated from a template library
  - JavaDoc-like HTML presentation

Technical platform

- Hosted at GitLab: `http://gitlab.com/ottr/templates`
- Git, version control system:

  `master`/`staging`/`work-in-progress` branches
- Issue tracker: template development and suggestions, and maintenance tasks
- Continuous integration/continuous delivery (CI/CD) service:

  Publish templates and documentation to webserver

- Published to webserver using content negotiation
- Use content delivery network (CDN) for fast and stable distribution

Maintenance of template libraries: Removing redundancy

- Formal foundation of OTTR templates permits analysis
- Relations between templates, e.g,. *depends*, *overlaps*, *contains*
- Two types of redundancy:

  Lack of reuse: Pattern captured by existing template, refactor.

  Uncaptured pattern: Pattern not captured by template, define new template and refactor

Maintenance of template libraries: Removing redundancy

- Formal foundation of OTTR templates permits analysis
- Relations between templates, e.g,. *depends*, *overlaps*, *contains*
- Two types of redundancy:

  Lack of reuse: Pattern captured by existing template, refactor.

  Uncaptured pattern: Pattern not captured by template, define new template and refactor

$$T_1(\ldots) \quad :: \quad A(\ldots), B(\ldots), C(\ldots), D(\ldots).$$
$$T_2(\ldots) \quad :: \quad A(\ldots), B(\ldots).$$

Maintenance of template libraries: Removing redundancy

- Formal foundation of OTTR templates permits analysis
- Relations between templates, e.g,. *depends*, *overlaps*, *contains*
- Two types of redundancy:

  Lack of reuse: Pattern captured by existing template, refactor.

  Uncaptured pattern: Pattern not captured by template, define new template and refactor

$T_1(\ldots) \quad :: \quad A(\ldots), B(\ldots), C(\ldots), D(\ldots).$
$T_2(\ldots) \quad :: \quad A(\ldots), B(\ldots).$

Maintenance of template libraries: Removing redundancy

- Formal foundation of OTTR templates permits analysis
- Relations between templates, e.g,. *depends*, *overlaps*, *contains*
- Two types of redundancy:

  Lack of reuse: Pattern captured by existing template, refactor.

  Uncaptured pattern: Pattern not captured by template, define new template and refactor

$$T_1(\ldots) \quad :: \quad T_2(\ldots), C(\ldots), D(\ldots).$$
$$T_2(\ldots) \quad :: \quad A(\ldots), B(\ldots).$$

Maintenance of template libraries: Removing redundancy

- Formal foundation of OTTR templates permits analysis
- Relations between templates, e.g,. *depends*, *overlaps*, *contains*
- Two types of redundancy:

  Lack of reuse: Pattern captured by existing template, refactor.

  Uncaptured pattern: Pattern not captured by template, define new template and refactor

$$
\begin{aligned}
T_1(\ldots) &:: T_2(\ldots), C(\ldots), D(\ldots).\\
T_2(\ldots) &:: A(\ldots), B(\ldots).\\
T_3(\ldots) &:: C(\ldots), D(\ldots), E(\ldots).
\end{aligned}
$$

Maintenance of template libraries: Removing redundancy

- Formal foundation of OTTR templates permits analysis
- Relations between templates, e.g,. *depends*, *overlaps*, *contains*
- Two types of redundancy:

  Lack of reuse: Pattern captured by existing template, refactor.

  Uncaptured pattern: Pattern not captured by template, define new template and refactor

$$
\begin{aligned}
T_1(\ldots) &:: T_2(\ldots), C(\ldots), D(\ldots).\\
T_2(\ldots) &:: A(\ldots), B(\ldots).\\
T_3(\ldots) &:: C(\ldots), D(\ldots), E(\ldots).\\
T_4(\ldots) &:: C(\ldots), D(\ldots).
\end{aligned}
$$

Maintenance of template libraries: Removing redundancy

- Formal foundation of OTTR templates permits analysis
- Relations between templates, e.g,. *depends*, *overlaps*, *contains*
- Two types of redundancy:

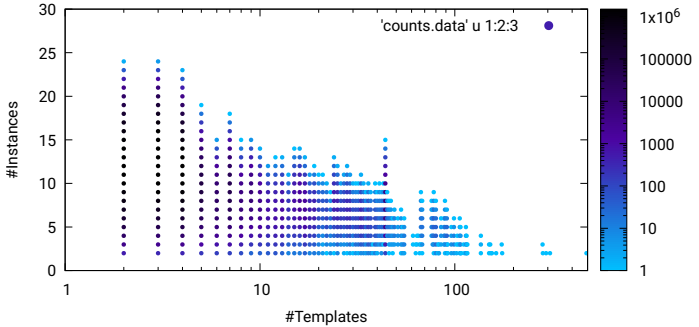
  Lack of reuse: Pattern captured by existing template, refactor.

  Uncaptured pattern: Pattern not captured by template, define new template and refactor

$$
\begin{aligned}
T_1(\ldots) &:: T_2(\ldots), T_4(\ldots).\\
T_2(\ldots) &:: A(\ldots), B(\ldots).\\
T_3(\ldots) &:: T_4(\ldots), E(\ldots).\\
T_4(\ldots) &:: C(\ldots), D(\ldots).
\end{aligned}
$$

Evaluation: Aibel use case

- 900+ templates
- 550 unique templates (350 superfluous?)
- 55 million candidate redundancies
- Need heuristics for how to fix



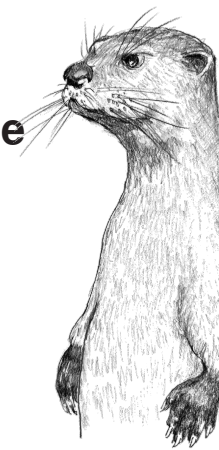
Future work

- Define and automate more quality checks, syntactic and semantic
- Define structures and metrics and heuristics over templates libraries
- Improve documentation
- Improve search over template library
  - text search
  - faceted search
  - SPARQL endpoint

Tutorial: Pattern-based knowledge base construction

Martin G. Skjæveland Leif Harald Karlsen Daniel P. Lupp

Modelling with OTTR templates



Modeling exercise: Phone factory

Phone factory

High-level:

- ontology that describes Phones and their Components
- new phones built at factory on a regular basis, hence need a simple way of populating the ontology with correct classes and instances
- differentiate between subclasses of `ex:Phone` (e.g., `ex:iPhone8` as a class of phones) and instances (e.g., `ex:myiphone1` as a concrete instance of `ex:iPhone8`)
- we will now build templates that help us populate such an ontology

Phone factory: Summary

- `ex:Phone` created new classes of phones
- `ex:ConsistsOf` related phones to their components

Phone factory: Summary

- `ex:Phone` created new classes of phones
- `ex:ConsistsOf` related phones to their components

What if the factory starts building laptops? We could

- create `ex:Laptop`, a template that creates new classes of laptops
- can reuse `ex:ConsistsOf` (!)

Phone and Laptop Factory

```
t:Phone[owl:Class ?name, NEList<owl:Class> ?components, ? xsd:string
?label] :: {
  ax:SubClassOf(?name, ex:Phone),
  ottr:Triple(?name, rdfs:label, ?label),
  t:ConsistsOf(?name, ?components)
}.

t:Laptop[owl:Class ?name, NEList<owl:Class> ?components, ? xsd:string
?label] :: {
  ax:SubClassOf(?name, ex:Laptop),
  ottr:Triple(?name, rdfs:label, ?label),
  t:ConsistsOf(?name, ?components)
}.
```

Phone and Laptop Factory

```
t:Phone[owl:Class ?name, NEList<owl:Class> ?components, ? xsd:string
?label] :: {
  ax:SubClassOf(?name, ex:Phone),
  ottr:Triple(?name, rdfs:label, ?label),
  t:ConsistsOf(?name, ?components)
}.

t:Laptop[owl:Class ?name, NEList<owl:Class> ?components, ? xsd:string
?label] :: {
  ax:SubClassOf(?name, ex:Laptop),
  ottr:Triple(?name, rdfs:label, ?label),
  t:ConsistsOf(?name, ?components)
}.

t:Tablet[...]....
```

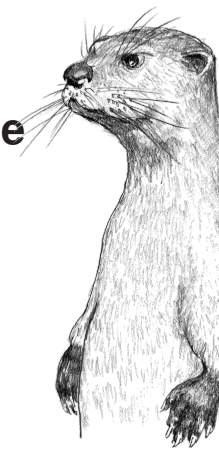
Device Factory

```
t:Device[owl:Class ?name, NEList<owl:Class> ?components, ? xsd:string
?label, ?deviceType = ex:Device] :: {
  ax:SubClassOf(?name, ?deviceType),
  ottr:Triple(?name, rdfs:label, ?label),
  t:ConsistsOf(?name, ?components)
}.
```

Tutorial: Pattern-based knowledge base construction

Martin G. Skjæveland Leif Harald Karlsen Daniel P. Lupp

Use Case: Failure Modes and Effects Analysis



Motivation - Takata Air Bag recall

- Takata designed and built an airbag that was installed in a large amount of cars between 2002 and 2015

Motivation - Takata Air Bag recall

- Takata designed and built an airbag that was installed in a large amount of cars between 2002 and 2015
- Defective inflators have caused 24 deaths, leading to the largest recall in history

Motivation - Takata Air Bag recall

- Takata designed and built an airbag that was installed in a large amount of cars between 2002 and 2015
- Defective inflators have caused 24 deaths, leading to the largest recall in history
- Was not identified as a potential failure in the failure modes and effects analysis (FMEA)

Motivation - Takata Air Bag recall

- Takata designed and built an airbag that was installed in a large amount of cars between 2002 and 2015
- Defective inflators have caused 24 deaths, leading to the largest recall in history
- Was not identified as a potential failure in the failure modes and effects analysis (FMEA)
- 34 million cars, massive amounts of data $\rightarrow$ went unnoticed until the scale of consequences became apparent

The challenge for industry

- Systems are comprised of many sub-systems and components, each of which can fail

The challenge for industry

- Systems are comprised of many sub-systems and components, each of which can fail
- Maintenance strategies and tasks are defined for “expected” failures for each of these

The challenge for industry

- Systems are comprised of many sub-systems and components, each of which can fail
- Maintenance strategies and tasks are defined for “expected” failures for each of these
- Standards exist which provide a coding for failure modes (FM) (malfunctions) e.g. ISO14224

The challenge for industry

- Systems are comprised of many sub-systems and components, each of which can fail
- Maintenance strategies and tasks are defined for “expected” failures for each of these
- Standards exist which provide a coding for failure modes (FM) (malfunctions) e.g. ISO14224
- 10 000 - 30 000 work orders per month (corrective and scheduled)
- Impossible to manually check work order to see if each failure on each component/ system was expected

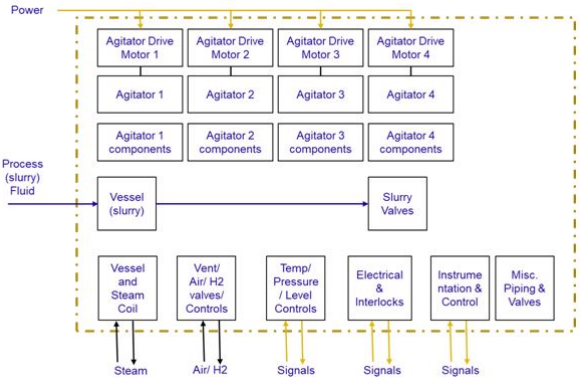
Our challenge

Use ontology reasoning to determine

- which work orders do not follow established FM coding;
- which components/ systems had unexpected failures.

Our Case Study: Pressure Vessel Industrial Process Plan

- 86 Functional Location IDs
- 100 FMEA entries using 16 FM codes
- 220 Work Order records (corrective and scheduled) using 20 FM codes



Data sources

Domain knowledge is encoded in tabular data and spreadsheets:

- FMEA tables: usually created in Excel
- Data about failure events: Captured in work orders stored in CMMS (computerised maintenace management system), usually extracted to csv
- Failure mode encoding of work order: inferred by engineers from natural language, coded using a Standard, stored in DB or csv

What does data look like? Level Gauge only

FMEA data

Component	Function	Failure mode observation	Failure Mode code	Failure mechanism	Failure Effect	Hid den	Mitigation
Radiation level gauge	Detect high level	Failure to function on demand	FTF	Clearance/ alignment failure	Overfilling asset	Yes	52W Shutter test

Work Order data

Date	#Item	#Action	#Symptom	Failure mode	Failure mode code
20/01/2017	Level Interlock	Override	High Level	Failure to function as intended	FTI
15/02/2017	Radiation Switch	Calibrate	Unknown	Abnormal Inst. reading	AIR
10/05/2017	Level Detector	Replace	High Level	Failure to function as intended	FTI
03/07/2017	Level Switch	Calibrate	Unknown	Abnormal Inst. reading	AIR
10/08/2017	Level Reading	Calibrate	High Level	Abnormal Inst. reading	AIR
27/07/2018	Level	Override	High Level	Failure to function as intended	FTI
31/10/2018	Level Alarm	Calibrate	High Level	Failure to function as intended	FTI

Lessons learned

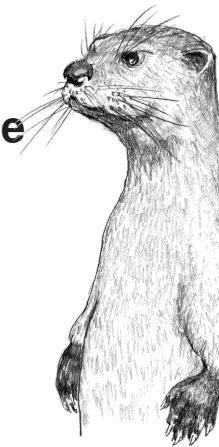
- structure of library plays a large role in maintenance of modeling via templates
- string manipulation, adding of prefixes currently done within query

Lessons learned

- structure of library plays a large role in maintenance of modeling via templates
- string manipulation, adding of prefixes currently done within query
- future work: functional language for term manipulation, compatible with OTTR

Tutorial: Pattern-based knowledge base construction

Martin G. Skjæveland Leif Harald Karlsen Daniel P. Lupp



Summing up

Reasonable Ontology Templates (OTTR): Example

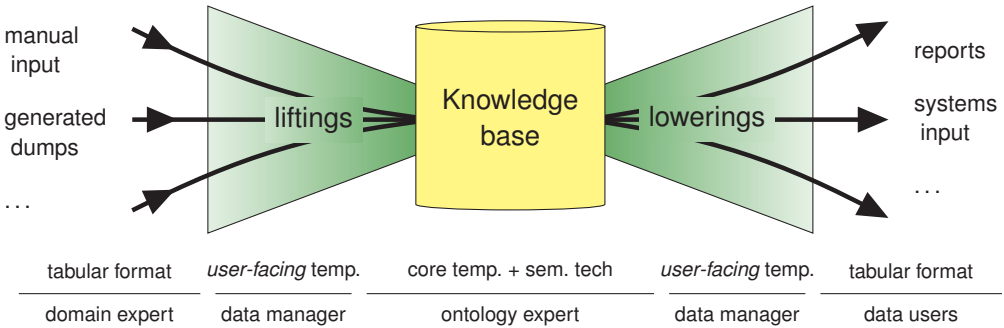
```
p:Margherita rdf:type owl:Class .
p:Margherita rdfs:subClassOf p:Pizza .
p:Margherita rdfs:label "Margherita"@it .
```

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

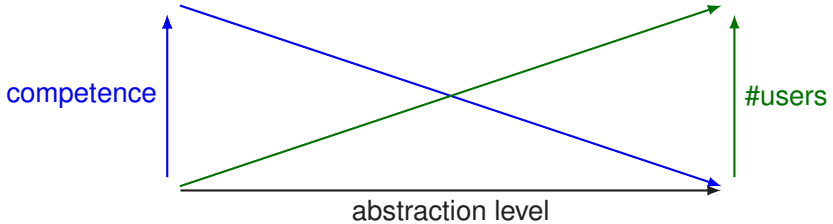
- Substitution
- Macro expansion

```
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) .
```

Vision: Template-driven methodology



Template libraries



Our vision: Libraries of best-practice templates created and curated by experts

- OWL or RDFS (e.g., `tpl.ottr.xyz`)
- Upper ontology specific: BFO, DOLCE, etc.
- Domain specific: equipment life-cycle, geological information, etc.
- User-facing: e.g., designed for specific inputs/outputs

Provide guidelines and tools to ensure high-quality template libraries

Discussion

- Is OTTR something you would like to use?
- What features, formats, and functionality would you like supported?
- Questions?

- `https://ottr.xyz/#Contact` — mailing list, issue tracker, email
- `https://gitter.im/ottr-talk`

- Please provide feedback to this tutorial:
  `https://tinyurl.com/ottr2020-feedback`