Tutorial: Scalable construction of sustainable knowledge bases

Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

Who are we?

Martin

Leif Harald

Daniel

Melinda

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.

Melinda Hodkiewicz professor at University of Western Australia. Engineering academic. Improve maintenance, asset management and safety practices. Involved in the development of ontology and natural language processing of maintenance records.

Tutorial plan

09:20–10:40	**Introduction:** Get to know us and each other, general motivation and introduction to the OTTR appraoch
10:40–11:20	*Coffee Break*
11:20–12:40	**Fundamental Features of OTTR:** Learn the basics of OTTR and how to use the various features in a hands-on session
12:40–14:00	*Lunch*
14:00–15:20	**Modeling in Practice:** Demonstration of modeling methodology and its uses in an industrial use case
15:20–16:00	*Coffee Break*
16:00–17:20	**OTTR for Maintenance:** Benefits of OTTR for ontology and library maintenance, applied in industrial use case

Who are you?

https://menti.com

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 sustainable 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 sustainable shared template libraries
- Use cases

Not about:

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

FMEA Use case

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

FMEA Use case

Goal:

- Use ontologies to capture FMEA and work order data
- Use reasoning to determine if a corrective work order was an unexpected failure mode at the component level, subsystem, or system level
- Use reasoning to ensure compliance with existing Standards

Aibel Use Case

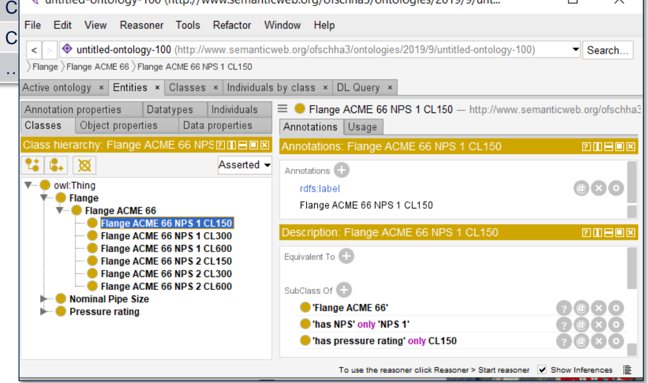
Requirements for building large (industrial) ontologies

- Support large number of classes
  - Some of which might have *very* similar shape
- Support a large number of (industry) standards
  - which might have apparently imcompatible 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	C
Flange6	Flange ACME 66 NPS 2 CL600	ACME 66	NPS 2	C
…	…	…	…	…



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!

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?!?

Industry needs

Based on our experience:

- High demand for semantic technologies in industry,
- for automating trivial, but complex and time-consuming tasks.

- However, “no” ontologies are available, and
- tools and competence for building them is a problem

- Need scalable tools and procedures
- Model must be extendible and highly maintainable
- Changes to model done in original source, not in the ontology

Tutorial: Scalable construction of sustainable knowledge bases

Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

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

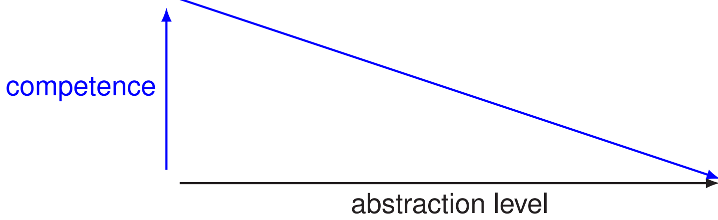
Problem:

- 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

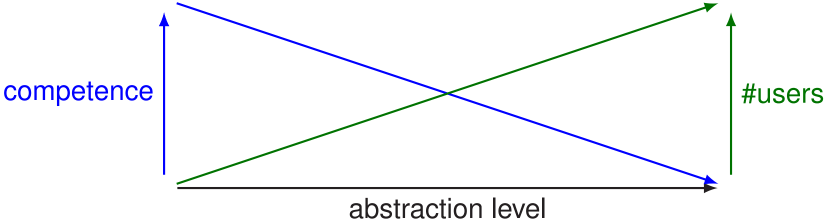
We need:

- 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”

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)
```

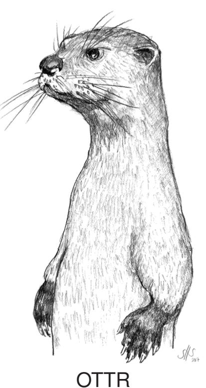
?

- Hide complexity
- Separation of concerns
- Support different users

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

Reasonable Ontology Templates (OTTR)

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



OTTR

Reasonable Ontology Templates (OTTR): Example

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

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

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

Reasonable Ontology Templates (OTTR): Example

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

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

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

Reasonable Ontology Templates (OTTR): Example

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

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

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

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

Reasonable Ontology Templates (OTTR): Example

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

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

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

```
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 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) .
```

Reasonable Ontology Templates (OTTR): Example

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

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

```
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 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) .
```

Reasonable Ontology Templates (OTTR): Example

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

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

```
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 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) .
```

Reasonable Ontology Templates (OTTR): Example

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

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

- Substitution
- Macro expansion

```
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 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) .
```

Reasonable Ontology Templates (OTTR): Example

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

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

- Substitution
- Macro expansion

```
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 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) .
```

Reasonable Ontology Templates (OTTR): Example

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

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

- Substitution
- Macro expansion

```
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) .
```

Reasonable Ontology Templates (OTTR): Example

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

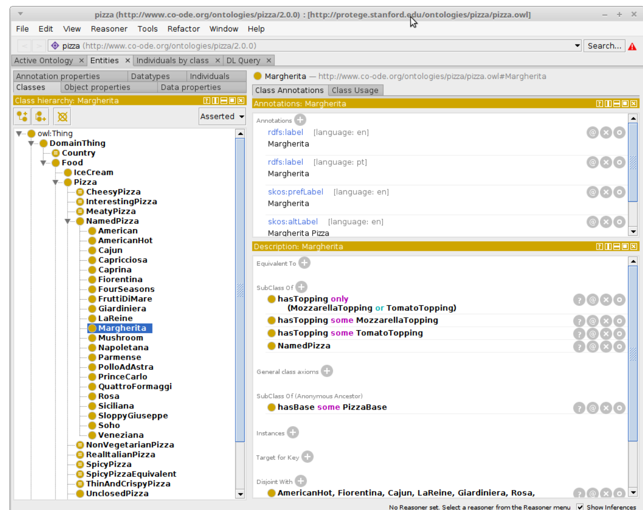
- Abstractions
- Don't Repeat Yourself (DRY)
- Compositional patterns
- Encapsulate complexity
- Uniform modelling
- Separate design and content
- Ensure completeness of input
- Simple input format
- Substitution
- Macro expansion

```
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



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)) .
```

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

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

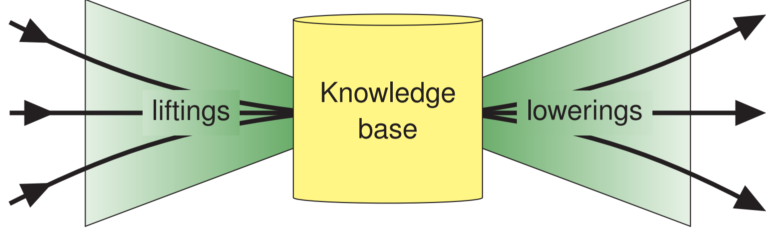
bOTTR: ETL mappings to OTTR instances

```
[] 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" ] .
```

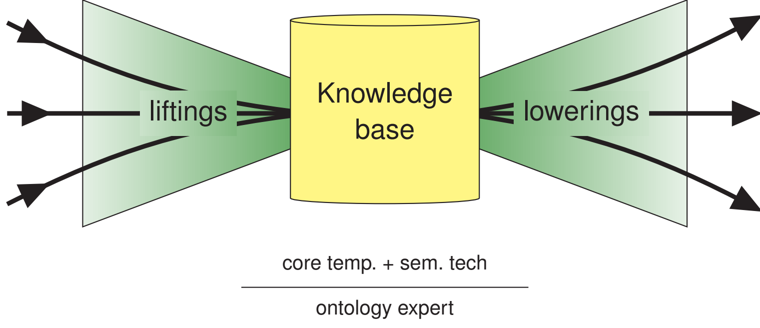
qOTTR: templates as queries, query for instances of pattern

```
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
}
```

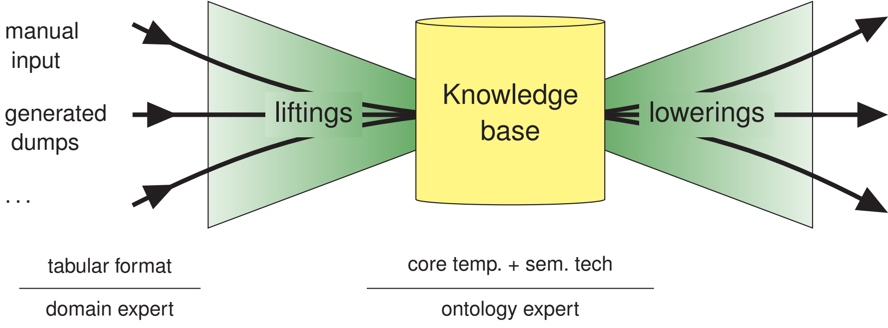
Vision: Template-driven methodology



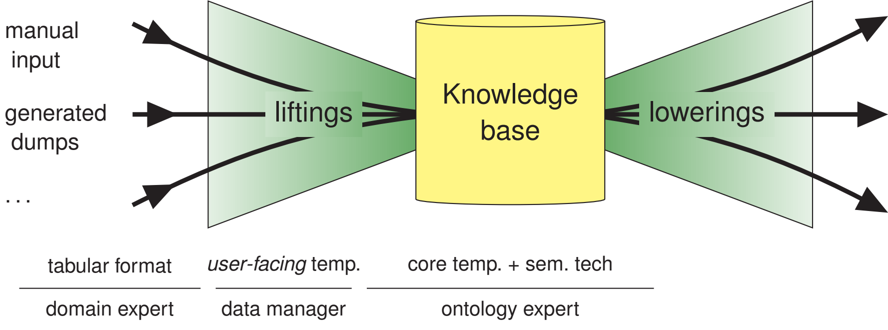
Vision: Template-driven methodology



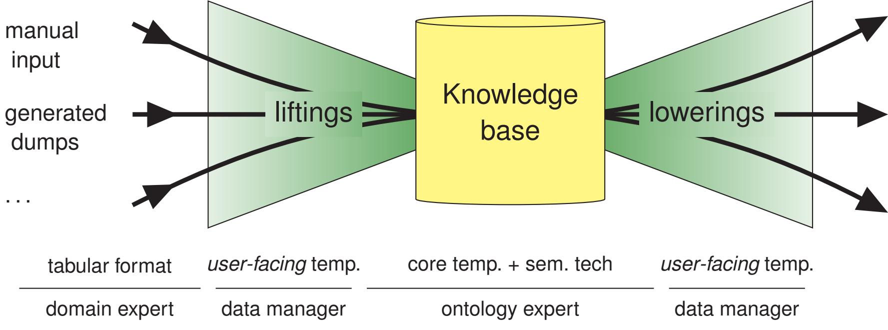
Vision: Template-driven methodology



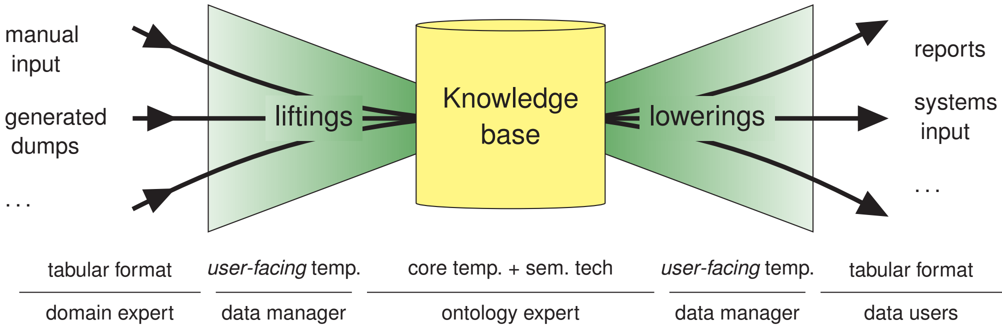
Vision: Template-driven methodology



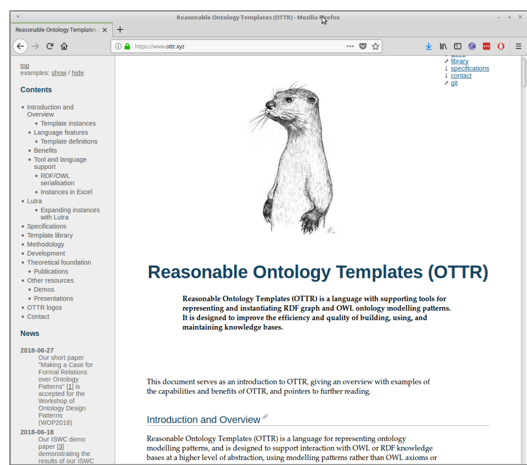
Vision: Template-driven methodology



Vision: Template-driven methodology



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



Resources: Lutra CLI

- Java executable CLI
- Open source
- Industry applications

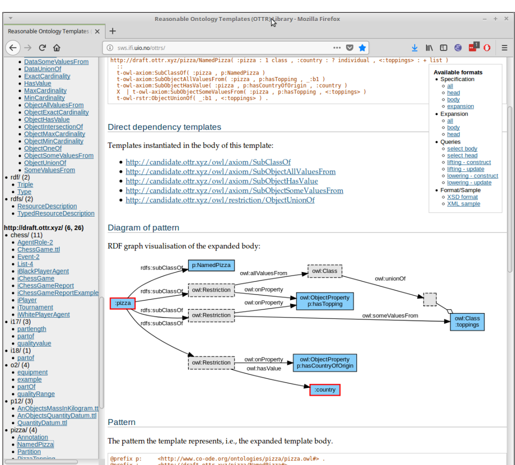
- Expand instances
- Type-checking, cycle detection, ...
- API

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

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


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


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

[ rdf:type owl:Ontology
 ] .

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

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


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

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


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


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

Resources: Library of OTTR templates

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



Related solutions

Spreadsheet tools: DataMaster
Modelling patterns: ODP
Programming: Tawny OWL
Scripting: OPPL
SPARQL based: SPIN, SPARQL-Generate
Mappings: RML
Logical frameworks: GDOL

Related solutions

Spreadsheet tools: DataMaster
Modelling patterns: ODP
Programming: Tawny OWL
Scripting: OPPL
SPARQL based: SPIN, SPARQL-Generate
Mappings: RML
Logical frameworks: GDOL

OTTR
Formal pattern based modelling framework for RDF/OWL

Exercise: Getting started with Lutra

- Download Lutra: **https://gitlab.com/ottr/lutra/lutra/-/releases**
  or visit WebLutra: **http://weblutra.ottr.xyz**
- Pizza spreadsheet:
  `http://spec.ottr.xyz/pOTTR/0.1/files/03-tabOTTR/PizzaOntologyInstances.xlsx`
- Run (without comments):

```
java -jar lutra.jar
  -I tabottr                      # input format
  -f                              # fetch template definitions by their IRI
  -O wottr                        # output format
  -o pizza.ttl                    # output file
  PizzaOntologyInstances.xlsx     # input
```

- Get help with `java -jar lutra.jar --help`
- Open in Protégé
- Edit spreadsheet, rerun and see changes

Tutorial: Scalable construction of sustainable knowledge bases

Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

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

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

We'll now look at more features!

```
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)
} .
```

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

UiO : University of Oslo

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

$$\texttt{ottr:Triple}[\text{?subject}, \text{?predicate}, \text{?object}] :: \texttt{BASE} .$$

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

$$\texttt{ottr:Triple}(\texttt{ex:martin}, \texttt{ex:knows}, \texttt{ex:daniel}) \quad \leadsto \quad \texttt{ex:martin ex:knows ex:daniel .}$$

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 .`

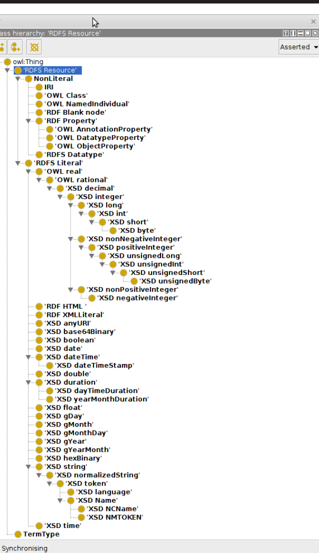
- Translation done by implementation
- Other base templates possible: E.g. `Quad`, OWL axioms, etc.
- Note: “Templates all the way down”

Exercises: Making templates and instance

- Log in to `https://tinyurl.com/ottr-basics`
- Do the exercises in sections 2 and 3

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 .
```

Types

Examples

- Type checking instances

$\texttt{ax:SubClassOf}\big[\text{owl:Class ?sub}, \text{owl:Class ?super}\big]\ .$

$\texttt{ax:SubClassOf}\big(\texttt{p:Margherita}, \texttt{p:Pizza}\big)\ .$

$\texttt{ax:SubClassOf}\big(\texttt{"Margherita"}, \texttt{p:Pizza}\big)\ .$

UiO : University of Oslo

- 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 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)
    } .
```

- 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

$\texttt{r:InstanceOf}\left[\text{owl:NamedIndividual ?elem}, \text{owl:Class ?class}\right]$.

$\texttt{r:InstanceOf}\left(\texttt{_:margherita}, \texttt{p:Pizza}\right)$

$\texttt{r:InstanceOf}\left(\texttt{p:mypizza}, \texttt{_:margherita}\right)$ OK? (depending on type hierarchy)

```
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)
```

`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

Missing values
Problem

```
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)
  } .

pz:Pizza(p:Margherita, "Margherita"@it) .
pz:Pizza(p:WheatEver, ???) .
```

Modifiers – Optional

- Optional (?) – permits expansion even if argument is a missing value
- Missing values are denoted by `none` (and empty cells in tabOTTR)
- Instances with missing values passed to parameter without ? are discarded
- Reduce the number of necessary template definitions, and reduce mental overhead

Optional

Example

```
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)
    } .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .
pz:Pizza(p:WheatEver, none) .
```

Optional

Example

```
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)
    } .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .    OK
pz:Pizza(p:WheatEver, none) .    OK
```

Optional

Example

```
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)
    } .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .    OK
pz:Pizza(p:WheatEver, none) .    OK
```

```
pz:Pizza(p:WheatEver, none) .
```

```
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)
} .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .
```

OK

```
pz:Pizza(p:WheatEver, none) .
```

OK

```
pz:Pizza(p:WheatEver, none) .
```

⇝

```
ottr:Triple(p:WheatEver, rdf:type, owl:Class)
ottr:Triple(p:WheatEver, rdfs:subClassOf, p:Pizza)
ottr:Triple(p:WheatEver, rdfs:label, none)
```

Optional

Example

```
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)
    } .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .    OK
pz:Pizza(p:WheatEver, none) .      OK
```

```
pz:Pizza(p:WheatEver, none) .

    ⇝

ottr:Triple(p:WheatEver, rdf:type, owl:Class)
ottr:Triple(p:WheatEver, rdfs:subClassOf, p:Pizza)
ottr:Triple(p:WheatEver, rdfs:label, none)
```

Optional
Example

```
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)
   } .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .     OK
pz:Pizza(p:WheatEver, none) .     OK
pz:Pizza(none, "No pizza"@en) .
```

Optional

Example

```
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)
   } .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .    OK
pz:Pizza(p:WheatEver, none) .    OK
pz:Pizza(none, "No pizza"@en) .    OK
```

Optional
Example

```
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)
   } .
```

```
pz:Pizza(p:Margherita, "Margherita"@it) .    OK
pz:Pizza(p:WheatEver, none) .    OK
pz:Pizza(none, "No pizza"@en) .    OK
```

Default Values

- We also allow default values for parameters

Default Values

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

Default Values

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

```
ax:SubObjectSomeValuesFrom[owl:Class ?class, owl:ObjectProperty ?prop, owl:Class ?range = owl:Thing] :: {
    ax:SubClassOf(?class, _:b),
    ax:ObjectSomeValuesFrom(_:b, ?prop, ?range)
} .
```

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

Default Values

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

$$\texttt{ax:SubObjectSomeValuesFrom}[\text{owl:Class ?class}, \text{owl:ObjectProperty ?prop}, \text{owl:Class ?range} = \texttt{owl:Thing}] :: \{$$
$$\quad \texttt{ax:SubClassOf}(\text{?class}, \texttt{_:b}),$$
$$\quad \texttt{ax:ObjectSomeValuesFrom}(\texttt{_:b}, \text{?prop}, \text{?range})$$
$$\} . \qquad (\text{i.e. ?class} \sqsubseteq \exists \text{?prop} . \text{?range})$$

$$\texttt{ax:SubObjectSomeValuesFrom}(\texttt{:Cat}, \texttt{:hasMother}, \texttt{:Cat})$$
$$\rightsquigarrow \{\texttt{ax:SubClassOf}(\texttt{:Cat}, \texttt{_:c}), \texttt{ax:ObjectSomeValuesFrom}(\texttt{_:c}, \texttt{:hasMother}, \texttt{:Cat})\} \qquad (\text{i.e. } \mathit{Cat} \sqsubseteq \exists \mathit{hasMother}.\mathit{Cat})$$

$$\texttt{ax:SubObjectSomeValuesFrom}(\texttt{:Container}, \texttt{:contains}, \texttt{none})$$
$$\rightsquigarrow \{\texttt{ax:SubClassOf}(\texttt{:Container}, \texttt{_:d}), \texttt{ax:ObjectSomeValuesFrom}(\texttt{_:d}, \texttt{:contains}, \texttt{owl:Thing})\} \qquad (\text{i.e. } \mathit{Container} \sqsubseteq \exists \mathit{contains}.\top)$$

Exercises: Types, nonblanks, optionals, defaults

- Log in to `https://tinyurl.com/ottr-basics`
- Do the exercises in sections 4 through 7

Multiple values

Problem

```
:Knows[owl:Class ?person, owl:Class ?knows] :: {
  ottr:Triple(?person, foaf:knows, ?knows)
} .
```

```
:Knows[owl:Class ?person, owl:Class ?knows] :: {
  ottr:Triple(?person, foaf:knows, ?knows)
} .
```

```
:Knows(ex:ann, ex:peter) .
:Knows(ex:ann, ex:mary) .
:Knows(ex:ann, ex:carl) .
```

Multiple values

Problem

```
:Knows[owl:Class ?person, List<owl:Class> ?knows] :: {
  ottr:Triple(?person, foaf:knows, ?knows)
} .

:Knows(ex:ann, (ex:peter, ex:mary, ex:carl)) .
```

Multiple values

Problem

```
:Knows[owl:Class ?person, List<owl:Class> ?knows] :: {
  ottr:Triple(?person, foaf:knows, ?knows)
} .

:Knows(ex:ann, (ex:peter, ex:mary, ex:carl)) .
```

Gets:

```
ottr:Triple(ex:ann, foaf:knows, (ex:peter, ex:mary, ex:carl))
```

Wants:

```
ottr:Triple(ex:ann, foaf:knows, ex:peter),
ottr:Triple(ex:ann, foaf:knows, ex:mary),
ottr:Triple(ex:ann, foaf:knows, ex:carl)
```

Multiple values

Problem

```
:Knows[owl:Class ?person, List<owl:Class> ?knows] :: {
  ottr:Triple(?person, foaf:knows, ?knows)
} .

:Knows(ex:ann, (ex:peter, ex:mary, ex:carl)) .
```

Gets:

```
ottr:Triple(ex:ann, foaf:knows, (ex:peter, ex:mary, ex:carl))
```

Wants:

```
ottr:Triple(ex:ann, foaf:knows, ex:peter),
ottr:Triple(ex:ann, foaf:knows, ex:mary),
ottr:Triple(ex:ann, foaf:knows, ex:carl)
```

Expansion modes

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

Expansion modes

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

Expansion modes

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

Expansion modes
Example

```
:Knows[ owl:Class ?person, List<owl:Class> ?knows] :: {
    cross | ottr:Triple(?person, foaf:knows, ++?knows)
} .
```

Expansion modes

Example

```
:Knows[ owl:Class ?person, List<owl:Class> ?knows] :: {
    cross | ottr:Triple(?person, foaf:knows, ++?knows)
} .
```

```
:Knows(A, (X, Y, Z)) .
```

Expansion modes

Example

```
:Knows[ owl:Class ?person, List<owl:Class> ?knows] :: {
    cross | ottr:Triple(?person, foaf:knows, ++?knows)
} .
```

```
:Knows(A, (X, Y, Z)) .
```

```
ottr:Triple(A, foaf:knows, X)
ottr:Triple(A, foaf:knows, Y)
ottr:Triple(A, foaf:knows, Z)
```

```
:Knows[List< owl:Class> ?person, List<owl:Class> ?knows] :: {
    cross | ottr:Triple(++?person, foaf:knows, ++?knows)
} .
```

```
:Knows((A, B, C), (X, Y, Z)) .
```

```
:Knows[List< owl:Class> ?person, List<owl:Class> ?knows] :: {
    cross | ottr:Triple(++?person, foaf:knows, ++?knows)
} .
```

```
:Knows((A, B, C), (X, Y, Z)) .
```

```
ottr:Triple(A, foaf:knows, X) ottr:Triple(B, foaf:knows, X) ottr:Triple(C, foaf:knows, X)
ottr:Triple(A, foaf:knows, Y) ottr:Triple(B, foaf:knows, Y) ottr:Triple(C, foaf:knows, Y)
ottr:Triple(A, foaf:knows, Z) ottr:Triple(B, foaf:knows, Z) ottr:Triple(C, foaf:knows, Z)
```

Expansion modes

Example

```
:Knows[List<owl:Class> ?person, List<owl:Class> ?knows] :: {
    zipMin | ottr:Triple(++?person, foaf:knows, ++?knows)
} .
```

```
:Knows((A, B, C), (X, Y, Z)) .
```

:Knows[List< owl:Class> ?person, List<owl:Class> ?knows] :: {
    zipMin | ottr:Triple(++?person, foaf:knows, ++?knows)
} .

:Knows((A, B, C), (X, Y, Z)) .

ottr:Triple(A, foaf:knows, X)

ottr:Triple(B, foaf:knows, Y)

ottr:Triple(C, foaf:knows, Z)

Expansion modes
Example

```
:Knows[List< owl:Class> ?person, List<owl:Class> ?knows] :: {
    zipMin | ottr:Triple(++?person, foaf:knows, ++?knows)
} .
```

```
:Knows((A, B), (X, Y, Z)) .
```

Expansion modes

Example

```
:Knows[List< owl:Class> ?person, List<owl:Class> ?knows] :: {
    zipMin | ottr:Triple(++?person, foaf:knows, ++?knows)
} .
```

```
:Knows((A, B), (X, Y, Z)) .
```

```
ottr:Triple(A, foaf:knows, X)
                              ottr:Triple(B, foaf:knows, Y)
```

Expansion modes

Example

```
:Knows[List< owl:Class> ?person, List<owl:Class> ?knows] :: {
    zipMax | ottr:Triple(++?person, foaf:knows, ++?knows)
} .
```

```
:Knows((A, B), (X, Y, Z)) .
```

Expansion modes

Example

```
:Knows[List< owl:Class> ?person, List<owl:Class> ?knows] :: {
    zipMax | ottr:Triple(++?person, foaf:knows, ++?knows)
} .
```

```
:Knows((A, B), (X, Y, Z)) .
```

```
ottr:Triple(A, foaf:knows, X)
                              ottr:Triple(B, foaf:knows, Y)
                                                            ottr:Triple(none, foaf:knows, Z)
```

Expansion modes – Motivation

- Removes repetition, e.g. instead of

$$\texttt{ax:SubClassOf}\big(\texttt{A}, \texttt{B}\big), \texttt{ax:SubClassOf}\big(\texttt{A}, \texttt{C}\big), \texttt{ax:SubClassOf}\big(\texttt{A}, \texttt{D}\big)$$

Expansion modes – Motivation

- Removes repetition, e.g. instead of

$$\texttt{ax:SubClassOf}(\texttt{A}, \texttt{B}), \texttt{ax:SubClassOf}(\texttt{A}, \texttt{C}), \texttt{ax:SubClassOf}(\texttt{A}, \texttt{D})$$

  we can simply write

$$\text{cross} \mid \texttt{ax:SubClassOf}(\texttt{A}, +\!\!+(\texttt{B}, \texttt{C}, \texttt{D}))$$

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

Expansion modes – Motivation

- Removes repetition, e.g. instead of

$$\texttt{ax:SubClassOf}(\texttt{A}, \texttt{B}), \texttt{ax:SubClassOf}(\texttt{A}, \texttt{C}), \texttt{ax:SubClassOf}(\texttt{A}, \texttt{D})$$

  we can simply write

$$\text{cross} \mid \texttt{ax:SubClassOf}(\texttt{A}, +\!\!+(\texttt{B}, \texttt{C}, \texttt{D}))$$

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

Expansion modes – Motivation

- Removes repetition, e.g. instead of

$$\texttt{ax:SubClassOf}(\texttt{A}, \texttt{B}), \texttt{ax:SubClassOf}(\texttt{A}, \texttt{C}), \texttt{ax:SubClassOf}(\texttt{A}, \texttt{D})$$

  we can simply write

$$\text{cross} \mid \texttt{ax:SubClassOf}(\texttt{A}, ++(\texttt{B}, \texttt{C}, \texttt{D}))$$

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

Exercises: Expansion modes

- Log in to `https://tinyurl.com/ottr-basics`
- Do the exercises in section 8

Reasonable Ontology Templates (OTTR): Example 2

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

Reasonable Ontology Templates (OTTR): Example 2

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

p:NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings]

Reasonable Ontology Templates (OTTR): Example 2

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

template name

p:NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings]

Reasonable Ontology Templates (OTTR): Example 2

$\texttt{Margherita} \sqsubseteq \texttt{NamedPizza}$

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

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

$\texttt{Margherita} \sqsubseteq \exists \texttt{hasTopping}.\texttt{Tomato}$

$\texttt{Margherita} \sqsubseteq \exists \texttt{hasTopping}.\texttt{Mozzarella}$

template name

p:NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings]

Reasonable Ontology Templates (OTTR): Example 2

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

template name — type — parameter name

p:NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings]

Reasonable Ontology Templates (OTTR): Example 2

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

template name — type — parameter name — optional input

p:NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings]

Reasonable Ontology Templates (OTTR): Example 2

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

template name — type — parameter name — optional input — list input

p:NamedPizza[owl:Class ?Name, ? owl:NamedIndividual ?Country, List<owl:Class> ?Toppings]

Reasonable Ontology Templates (OTTR): Example 2

$$
\begin{aligned}
&\texttt{Margherita} \sqsubseteq \texttt{NamedPizza}\\
&\texttt{Margherita} \sqsubseteq \exists\,\texttt{hasCountryOfOrigin}.\{\texttt{Italy}\}\\
&\texttt{Margherita} \sqsubseteq \forall\,\texttt{hasTopping}.(\texttt{Tomato} \sqcup \texttt{Mozzarella})\\
&\texttt{Margherita} \sqsubseteq \exists\,\texttt{hasTopping}.\texttt{Tomato}\\
&\texttt{Margherita} \sqsubseteq \exists\,\texttt{hasTopping}.\texttt{Mozzarella}
\end{aligned}
$$


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

Reasonable Ontology Templates (OTTR): Example 2

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

template name — type — parameter name — optional input — list input

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

$?Name \sqsubseteq \texttt{:NamedPizza}$

Reasonable Ontology Templates (OTTR): Example 2

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

template name — type — parameter name — optional input — list input

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

$?Name \sqsubseteq \texttt{:NamedPizza}$
$?Name \sqsubseteq \exists\,\texttt{hasCO}.\{?Cou$

Reasonable Ontology Templates (OTTR): Example 2

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

template name — type — parameter name — optional input — list input

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

$?Name \sqsubseteq \texttt{:NamedPizza}$
$?Name \sqsubseteq \exists\texttt{hasCO}.\{?Cou$
$?Name \sqsubseteq \forall\texttt{hasT}.(\sqcup ?Topp$

Reasonable Ontology Templates (OTTR): Example 2

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

template name · type · parameter name · optional input · list input

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

$?Name \sqsubseteq \texttt{:NamedPizza}$
$?Name \sqsubseteq \exists \texttt{hasCO}.\{?Cou$
$?Name \sqsubseteq \forall \texttt{hasT}.(\sqcup ?Top$
$?Name \sqsubseteq \exists \texttt{hasT}.?Toppin$

Reasonable Ontology Templates (OTTR): Example 2

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

template name · type · parameter name · optional input · list input

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

list expansion

$?Name \sqsubseteq \texttt{:NamedPizza}$
$?Name \sqsubseteq \exists \texttt{hasCO}.\{?Cou$
$?Name \sqsubseteq \forall \texttt{hasT}.(\sqcup ?Topp$
$?Name \sqsubseteq \exists \texttt{hasT}.?Toppin$

Reasonable Ontology Templates (OTTR): Example 2

$$\begin{aligned}
&\texttt{Margherita} \sqsubseteq \texttt{NamedPizza}\\
&\texttt{Margherita} \sqsubseteq \exists\,\texttt{hasCountryOfOrigin}.\{\texttt{Italy}\}\\
&\texttt{Margherita} \sqsubseteq \forall\,\texttt{hasTopping}.(\texttt{Tomato} \sqcup \texttt{Mozzarella})\\
&\texttt{Margherita} \sqsubseteq \exists\,\texttt{hasTopping}.\texttt{Tomato}\\
&\texttt{Margherita} \sqsubseteq \exists\,\texttt{hasTopping}.\texttt{Mozzarella}
\end{aligned}$$

template name / type / parameter name / optional input / list input

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

list expansion / marked for expansion

$?Name \sqsubseteq \texttt{:NamedPizza}$

$?Name \sqsubseteq \exists\,\texttt{hasCO}.\{?Cou$

$?Name \sqsubseteq \forall\,\texttt{hasT}.(\sqcup ?Topp$

$?Name \sqsubseteq \exists\,\texttt{hasT}.?Toppin$

Reasonable Ontology Templates (OTTR): Example 2

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

```
p:NamedPizza(Margherita, Italy, (Tomato, Mozzarella))
```

template name — type — parameter name — optional input — list input

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

list expansion — marked for expansion

$?Name \sqsubseteq \texttt{:NamedPizza}$
$?Name \sqsubseteq \exists \texttt{hasCO}.\{?Cou$
$?Name \sqsubseteq \forall \texttt{hasT}.(\sqcup ?Topp$
$?Name \sqsubseteq \exists \texttt{hasT}.?Toppin$

Reasonable Ontology Templates (OTTR): Example 2

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

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

template name — type — parameter name — optional input — list input

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

list expansion — cross; marked for expansion — ++

$?Name \sqsubseteq \texttt{:NamedPizza}$
$?Name \sqsubseteq \exists \texttt{hasCO}.\{?Cou$

$?Name \sqsubseteq \forall \texttt{hasT}.(\sqcup ?Top$
$?Name \sqsubseteq \exists \texttt{hasT}.?Toppin$

Reasonable Ontology Templates (OTTR): Example 2

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

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

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

template name — type — parameter name — optional input — list input

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

list expansion — marked for expansion

$?Name \sqsubseteq \texttt{:NamedPizza}$
$?Name \sqsubseteq \exists \texttt{hasCO}.\{?Cou$
$?Name \sqsubseteq \forall \texttt{hasT}.(\sqcup ?Top$
$?Name \sqsubseteq \exists \texttt{hasT}.?Toppin$

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
  - Default and optional values
- The features are based on well-known principles in programming language design
- Provides
  - a simple, yet powerful language for expressing ontology patterns
  - that ensures correctness, easy maintenance, and uniform modelling
  - across all levels of abstraction

Tools for instantiating templates

- tabOTTR: extracting data from spreadsheets
- bOTTR: extracting data from queries over various sources

- Transform spreadsheet data to instances
- Annotate/tag spreadsheet with 3 *instructions*:
  - `prefix`
  - `template`
  - `end`
- No mapping file, Lutra reads spreadsheet directly
- `https://spec.ottr.xyz/tabOTTR/0.3/`

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

bOTTR

- Transform query results to instances
- Supported sources:
  String sources: JDBC, CSV
  RDF sources: SPARQL, RDF
- bOTTR: RDF serialisation
- Multiple maps
- Integrate multiple sources
- `https://spec.ottr.xyz/bOTTR/0.1/`
- `:InstanceMap`:
  - `:template`
  - `:query`
  - `:source`
  - `:argumentMaps`(*)

bOTTR: RDF sources

- Transform query results to instances
- Supported sources:
  - *String sources*: JDBC, CSV
  - *RDF sources*: SPARQL, RDF
- bOTTR: RDF serialisation
- Multiple maps
- Integrate multiple sources
- `https://spec.ottr.xyz/bOTTR/0.1/`
- `:InstanceMap`:
  - `:template`
  - `:query`
  - `:source`
  - `:argumentMaps`(*)

```
[] a ottr:InstanceMap
   ottr:template ex:Person ;
   ottr:query """
      SELECT { ?firstName ?homepage }
      WHERE  { [] a voc:Person ;
                  voc:firstName ?firstName ;
                  voc:homepage ?homepage ;
                FILTER (?firstName = "Bob")
             } """;
   ottr:source ex:A . ## or ex:B .

   ex:A a ottr:SPARQLEndpointSource ;
      ottr:sourceURL "http://..." .

   ex:B a ottr:RDFFileSource ;
      ottr:sourceURL "http://...", "foaf.rdf" .
```

bOTTR

- Transform query results to instances
- Supported sources:
  - *String sources*: JDBC, CSV
  - *RDF sources*: SPARQL, RDF
- bOTTR: RDF serialisation
- Multiple maps
- Integrate multiple sources
- `https://spec.ottr.xyz/bOTTR/0.1/`
- `:InstanceMap`:
  - `:template`
  - `:query`
  - `:source`
  - `:argumentMaps`(*)

```
[] a ottr:InstanceMap
   ottr:template ex:Person ;
   ottr:query """
      SELECT firstName homepage
      FROM tblPerson;
   """;
   ottr:source [ a ottr:JDBCSource ;
     ottr:username "bob";
     ottr:password "X" ;
     ottr:jdbcDriver "com.mysql...";
     ottr:sourceURL "jdbc:mysql..." ] .
```

bOTTR

- Transform query results to instances
- Supported sources:
  - *String sources*: JDBC, CSV
  - *RDF sources*: SPARQL, RDF
- bOTTR: RDF serialisation
- Multiple maps
- Integrate multiple sources
- `https://spec.ottr.xyz/bOTTR/0.1/`
- `:InstanceMap`:
  - `:template`
  - `:query`
  - `:source`
  - `:argumentMaps`(*)

```
[] a ottr:InstanceMap
   ottr:template ex:Person ;
   ottr:query """
      SELECT firstName, age, homepage
      FROM CSVREAD('person.csv');
   """;
   ottr:source [ a ottr:H2Source ] ;
   ottr:argumentMaps (
      [ ottr:languageTag "en" ]
      [ ottr:type xsd:int ]
      [ ottr:type ottr:IRI ]
   ) .
```

Exercises

- `https://tinyurl.com/ottr-primer`
- Exercises in tabOTTR and/or bOTTR

Lunch until 14:00

After lunch:

- we design a template library together!
- modelling and maintenance of ontologies and template libraries (Aibel and FMEA use cases)

Tutorial: Scalable construction of sustainable knowledge bases

Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

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` (!)

UiO : University of Oslo

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)
}.
```

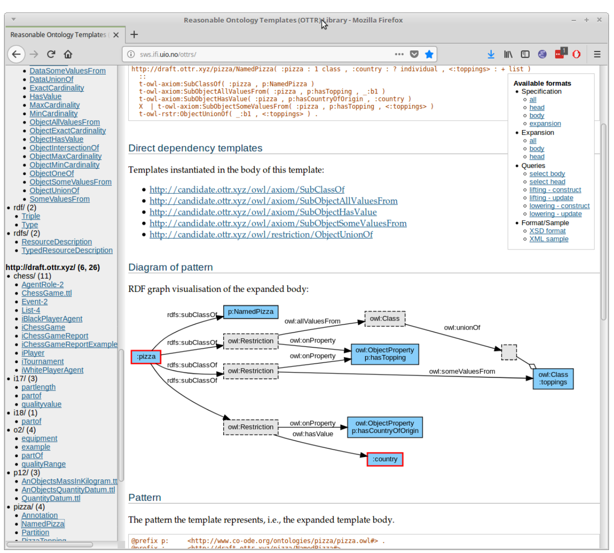
Template libraries

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

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

Provide guidelines and tools to ensure high-quality template libraries

OTTR template libraries



- `tpl.ottr.xyz`
- RDF, RDFS, OWL
- Examples

Work in progress: Modules

- A module is a collection of templates, with associated meta-information
- The module system will provide
  - Versioning
  - Access restrictions via private and public templates
  - A status system: *incomplete* < *draft* < *candidate* < *proposal* < *recommendation*
  - Publishing (with ensured backwards compatability)
  - Requirements (e.g., a template cannot depend on a template with lower status)
- *Lutra* will check for conformance with requirements

Tutorial: Scalable construction of sustainable knowledge bases

Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

FMEA Use Case

Motivation - Takata Air Bag recall

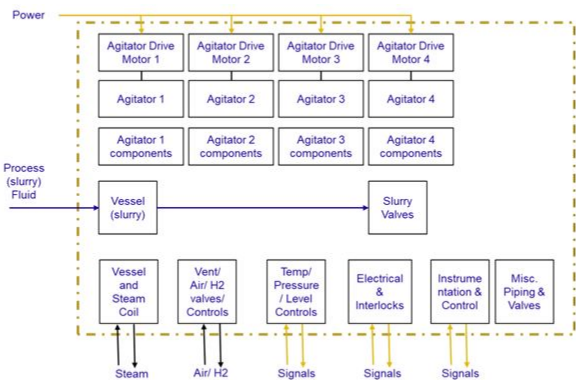
- Takata designed and built an airbag that was installed in a large amount of cars between 2002 and 2015
- Defective inflators have caused 24 deaths, leading to the largest recall in history
- Was not identified as a potential failure in the failure modes and effects analysis (FMEA)
- 34 million cars, 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
- Maintenance strategies and tasks are defined for “expected” failures
- 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 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



What does data look like? Level Gauge only

FMEA data

Component	Function	Failure mode observation	Failure Mode code	Failure mechanism	Failure Effect	Hidden	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

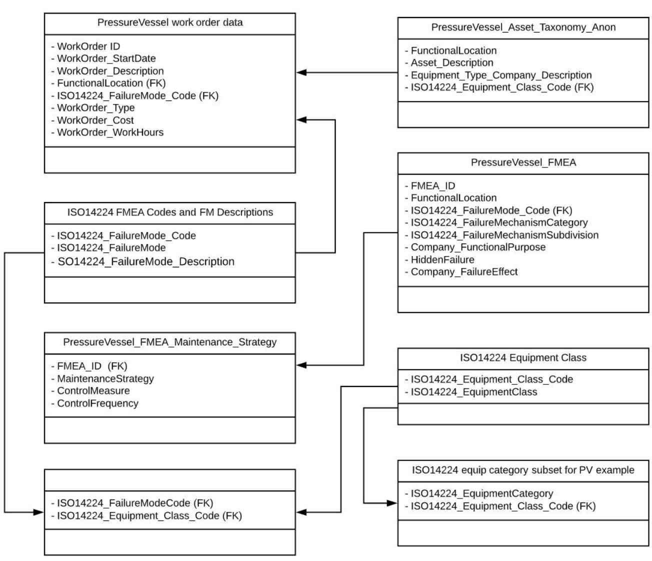
Date	#Item	#Action	#Symptom	Failure mode	Failure mode code
20/01/2017	Level Interlock	Override	High Level	Failure to function as intended	FTI
15/02/2017	Radiation Switch	Calibrate	Unknown	Abnormal Inst. reading	AIR
10/05/2017	Level Detector	Replace	High Level	Failure to function as intended	FTI
03/07/2017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

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

Case study data overview



The challenge

Use ontology reasoning to determine

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

Tutorial: Scalable construction of sustainable knowledge bases

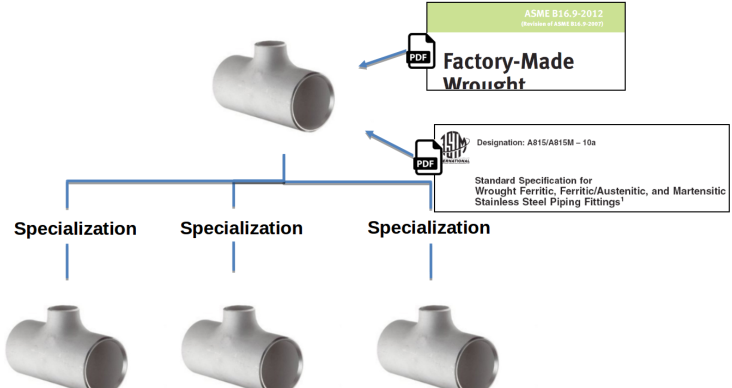
Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

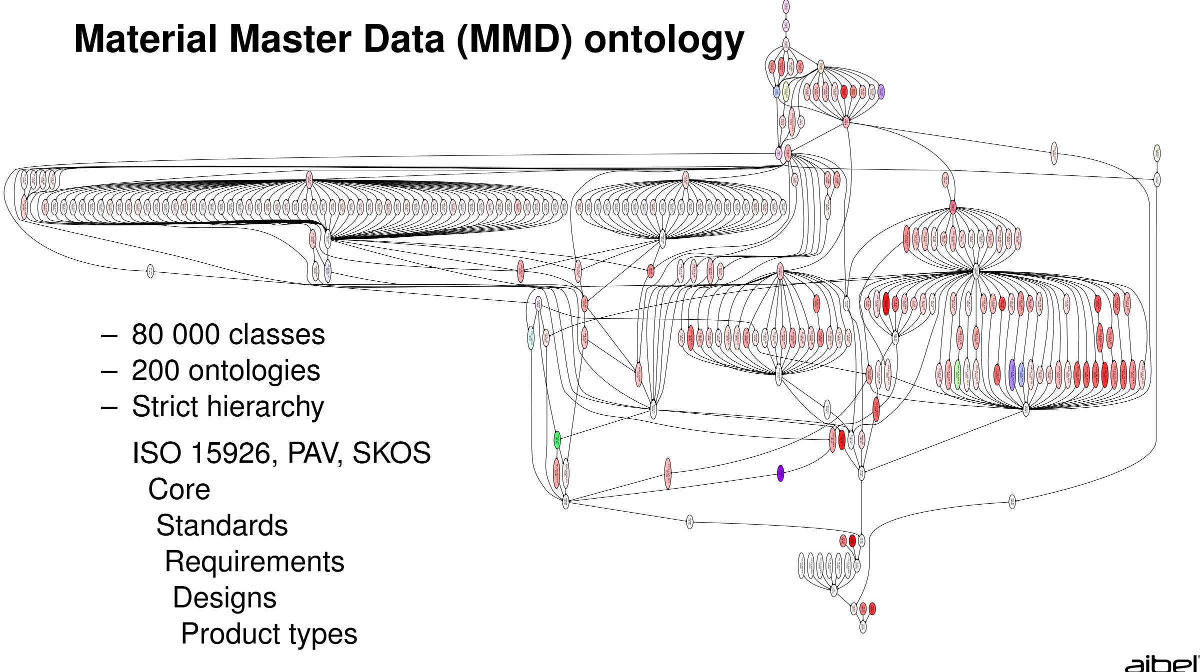
Aibel Use Case

– Large-scale ontology (80 000 classes) in use in capital projects
– Automating highly complex, but trivial work
– Representing industry standards and creative designs
– In-house developed
– Reasoning is an integral part
– quality assessment
– duplicate detection
– guiding end-user ontology development
– Reduced cost and improved quality and accountability

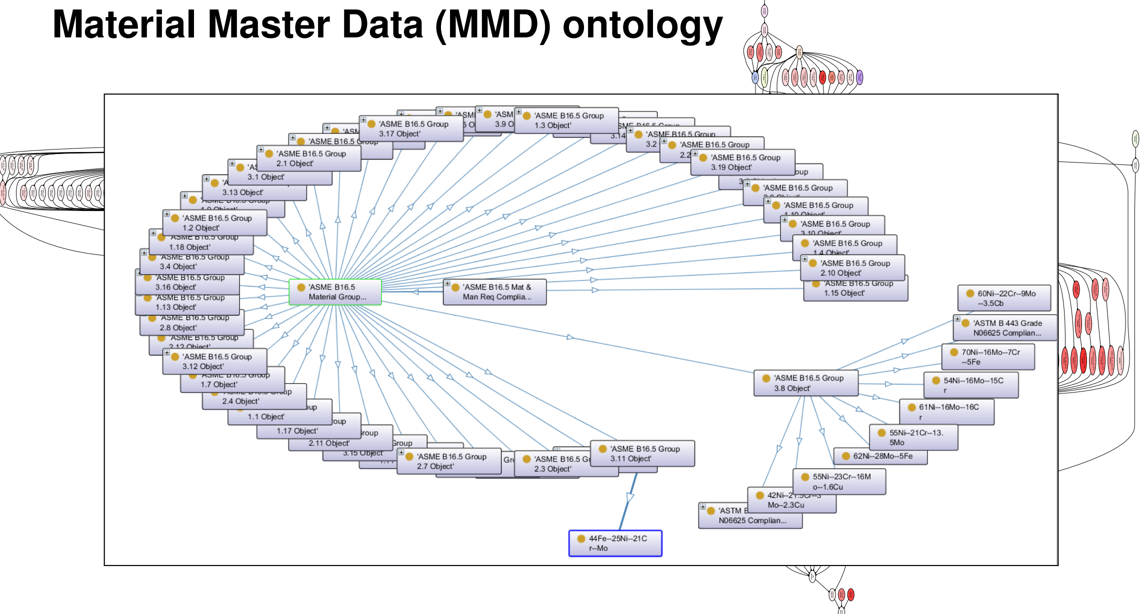
Engineering according to industry standards

– Material quality
– Wall thickness
– Manufacturing class
– …




Material Master Data (MMD) ontology
– 80 000 classes
– 200 ontologies
– Strict hierarchy
ISO 15926, PAV, SKOS
Core
Standards
Requirements
Designs
Product types
aibel

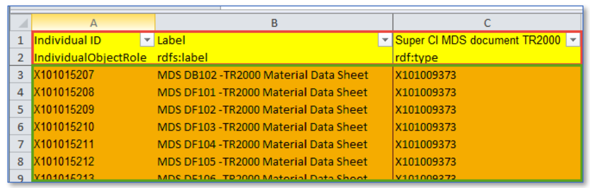
Material Master Data (MMD) ontology



aibel

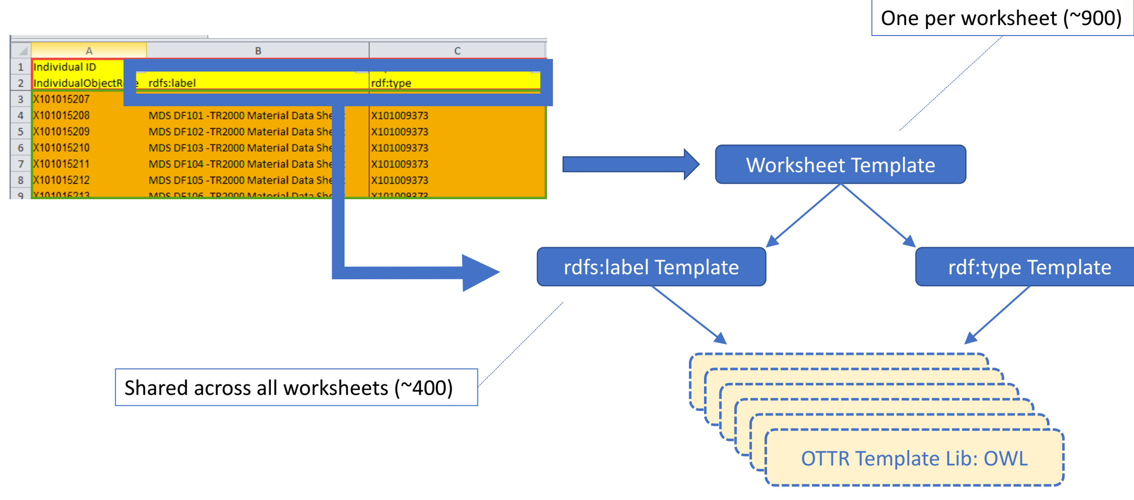
Current Tabular Conversion

- Based on ISO 15926 Part 7 Templates
- One *Template* for each Excel worksheet
- Each row converted to an *Instance* of the worksheet Template
- Conversion from template instances to OWL by auto-generated SPARQL Constructs



	A	B	C
1	Individual ID	Label	Super CI MDS document TR2000
2	IndividualObjectRole	rdfs:label	rdf:type
3	X101015207	MDS DB102 -TR2000 Material Data Sheet	X101009373
4	X101015208	MDS DF101 -TR2000 Material Data Sheet	X101009373
5	X101015209	MDS DF102 -TR2000 Material Data Sheet	X101009373
6	X101015210	MDS DF103 -TR2000 Material Data Sheet	X101009373
7	X101015211	MDS DF104 -TR2000 Material Data Sheet	X101009373
8	X101015212	MDS DF105 -TR2000 Material Data Sheet	X101009373
9	X101015213	MDS DF106 -TR2000 Material Data Sheet	X101009373

Template Hierarchy



With OTTR Aibel may…

- Explicitly represent semantics/interpretation of tabular data
- Make tabular conversion transparent and predictable
- Build a managed *internal* template library
- Extensively re-use internal and external templates
- Keep schema separate from tabular data
- Gain increased control of schemas/modelling patterns, while still allowing engineers to contribute bulk ontology content
- Reduce custom code-base

Tutorial: Scalable construction of sustainable knowledge bases

Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

Maintaining OTTR Libraries

OTTR template libraries

OTTR provides a solid formal foundation

- template construction,
- library maintenance, by ensuring uniformity of modelling and methods for removing redundancy,
- library exploration, by providing formal ways of relating and comparing templates.

OTTR template libraries

OTTR provides a solid formal foundation

- template construction,
- library maintenance, by ensuring uniformity of modelling and methods for removing redundancy,
- library exploration, by providing formal ways of relating and comparing templates.

Redundancy

- DRY (Don’t repeat yourself) is an imporant principle in software engineering:
  - ”Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.“

Redundancy

- DRY (Don’t repeat yourself) is an imporant principle in software engineering:
  - ”Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.“
- Redundancy makes maintaining a system cumbersome.

Redundancy

- DRY (Don’t repeat yourself) is an imporant principle in software engineering:
    - ”Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.“
- Redundancy makes maintaining a system cumbersome.
- Redundancy fixed by capturing patterns in abstractions (method, class, etc.) and refactoring.

Types of redundancy

We distinguish between two types of redundancy:

Lack of reuse: Pattern already captured by definition, refactor.

Uncaptured pattern: Pattern not captured by definition, define new template.

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubClassOf(?Name, _:b2),
    ax:ObjectAllValuesFrom(_:b2, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubClassOf(?Name, _:b2),
    ax:ObjectAllValuesFrom(_:b2, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubClassOf(?Name, _:b2),
    ax:ObjectAllValuesFrom(_:b2, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- Some of `t:Laptop`'s instances equals pattern of `ax:SubObjectAllValuesFrom`

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubClassOf(?Name, _:b2),
    ax:ObjectAllValuesFrom(_:b2, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- Some of `t:Laptop`'s instances equals pattern of `ax:SubObjectAllValuesFrom`
- The use of the instances unify

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubClassOf(?Name, _:b2),
    ax:ObjectAllValuesFrom(_:b2, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- Some of `t:Laptop`'s instances equals pattern of `ax:SubObjectAllValuesFrom`
- The use of the instances unify
- Refactor!

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3)),

    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- Some of `t:Laptop`'s instances equals pattern of `ax:SubObjectAllValuesFrom`
- The use of the instances unify
- Refactor!

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- No lack of reuse – Check if unify

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- No lack of reuse – Check if unify
- Uncaptured pattern!

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- No lack of reuse – Check if unify
- Uncaptured pattern!
- Make new template

```
<name>[?x1, ?x2, ?x3]
 :: {
   ax:SubClassOf(?x1, ?x2),
   ax:SubObjectAllValuesFrom(?x1, :hasComponent, _:c1),
   rstr:ObjectUnionOf(_:c1, ?x3),
   cross | ax:SubObjectSomeValuesFrom(?x1, :hasComponent, ++?x3)
} .
```

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- No lack of reuse – Check if unify
- Uncaptured pattern!
- Make new template

```
t:Device[?x1, ?x2, ?x3]
 :: {
   ax:SubClassOf(?x1, ?x2),
   ax:SubObjectAllValuesFrom(?x1, :hasComponent, _:c1),
   rstr:ObjectUnionOf(_:c1, ?x3),
   cross | ax:SubObjectSomeValuesFrom(?x1, :hasComponent, ++?x3)
} .
```

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
```

- No lack of reuse – Check if unify
- Uncaptured pattern!
- Make new template

```
t:Device[owl:Class ?Name,
         owl:Class ?Category,
         List<owl:Class> ?Parts]
 :: {
   ax:SubClassOf(?Name, ?Category),
   ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:c1),
   rstr:ObjectUnionOf(_:c1, ?Parts),
   cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Parts)
} .
```

Fixing redundancies

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    t:Device(?Name, :Phone, ?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, xsd:String ?Keyboard] :: {
    t:Device(?Name, :Laptop, ?Components),
    ax:SubObjectHasValue(?Name, :hasKBLayout, ?Keyboard)
} .
ax:SubObjectAllValuesFrom[owl:Class ?c, owl:ObjectProperty ?p, owl:Class ?r] :: {
    ax:SubClassOf(?c, _:b4), ax:ObjectAllValuesFrom(_:b4, ?p, ?r)
} .
t:Device[owl:Class ?Name, owl:Class ?Category, List<owl:Class> ?Parts] :: {
    ax:SubClassOf(?Name, ?Category),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:c1),
    rsrt:ObjectUnionOf(_:c1, ?Parts),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Parts)
} .
```

Needs User Interaction

Not always possible to automate, even for lack of reuse. E.g.:

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
} .
```

Needs User Interaction

Not always possible to automate, even for lack of reuse. E.g.:

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
} .
```

Needs User Interaction

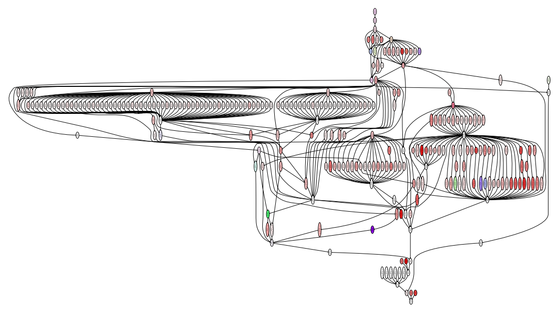
Not always possible to automate, even for lack of reuse. E.g.:

```
t:Phone[owl:Class ?Name, List<owl:Class> ?Components, owl:Class ?Network] :: {
    ax:SubClassOf(?Name, :Phone),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b1),
    ax:ObjectUnionOf(_:b1, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
    ax:SubObjectSomeValuesFrom(?Name, :hasNetworkType, ?Network)
} .
t:Laptop[owl:Class ?Name, List<owl:Class> ?Components, ? xsd:String ?Keyboard] :: {
    ax:SubClassOf(?Name, :Laptop),
    ax:SubObjectAllValuesFrom(?Name, :hasComponent, _:b3),
    rstr:ObjectUnionOf(_:b3, ?Components),
    cross | ax:SubObjectSomeValuesFrom(?Name, :hasComponent, ++?Components),
} .
```

Making `t:Phone` depend on `t:Laptop` is (maybe) a bad idea. Changing how `t:Laptop` is modelled, then changes how `t:Phone` is.

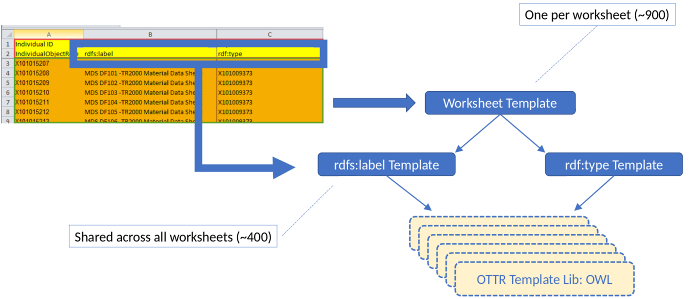
Evaluation: Aibel use case

- MMD ontology:
  100 000 classes
  200 modules
- 900+ spreadsheets



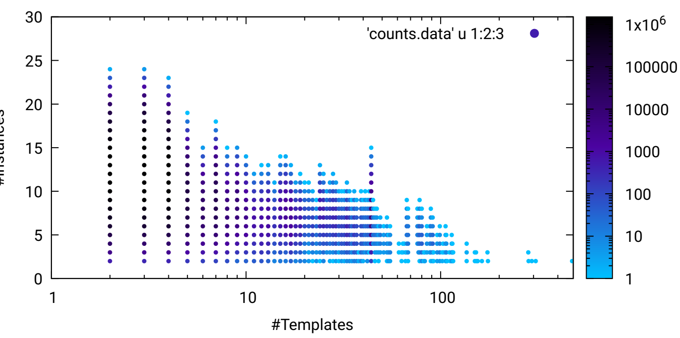
Evaluation: Aibel use case

- MMD ontology:
  100 000 classes
  200 modules
- 900+ spreadsheets
- All represented by OTTR templates



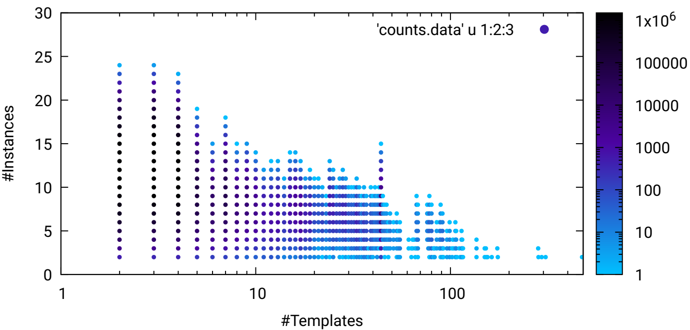
Evaluation: Aibel use case

- 900+ templates
- 550 unique templates (350 superfluous?)
- 55 million candidates



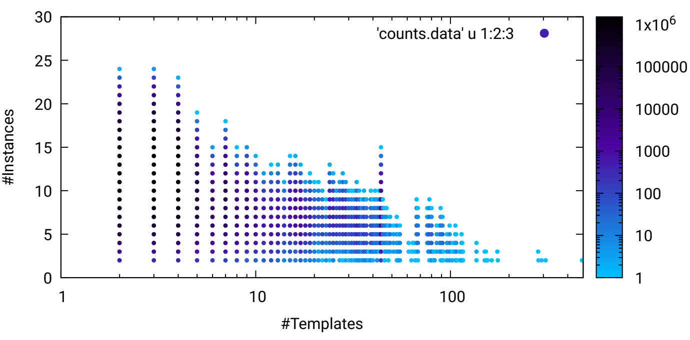
Evaluation: Aibel use case

- 900+ templates
- 550 unique templates (350 superfluous?)
- 55 million candidates
- reduced to under 3 million after fixing all lack of reuse



Evaluation: Aibel use case

- 900+ templates
- 550 unique templates (350 superfluous?)
- 55 million candidates
- reduced to under 3 million after fixing all lack of reuse
- Removing single uncaptured pattern: 1.8 million fewer redundancies



Use of candidates

- Choose an important template describing elbow pipes (ASME B16.9).

Use of candidates

- Choose an important template describing elbow pipes (ASME B16.9).
- This occurs in over 12,000 candidates.

Use of candidates

- Choose an important template describing elbow pipes (ASME B16.9).
- This occurs in over 12,000 candidates.
- By removing candidates that

Use of candidates

- Choose an important template describing elbow pipes (ASME B16.9).
- This occurs in over 12,000 candidates.
- By removing candidates that
  - contains non-domain specific instances (`rdfs:label`, `skos:comment`, etc.); and
  - contains few (less than 7) instances or templates,

got down to 24 candidates.

Use of candidates

- Choose an important template describing elbow pipes (ASME B16.9).
- This occurs in over 12,000 candidates.
- By removing candidates that
  - contains non-domain specific instances (`rdfs:label`, `skos:comment`, etc.); and
  - contains few (less than 7) instances or templates,

  got down to 24 candidates.
- Two of these (containing 7 instances) were repeated very often ($\geq$ 30 templates).

Use of candidates

- Choose an important template describing elbow pipes (ASME B16.9).
- This occurs in over 12,000 candidates.
- By removing candidates that
  - contains non-domain specific instances (`rdfs:label`, `skos:comment`, etc.); and
  - contains few (less than 7) instances or templates,

  got down to 24 candidates.
- Two of these (containing 7 instances) were repeated very often ($\geq$ 30 templates).
- Process above produced this template, which we could use in all the templates:

```
<name>[?x1, ?x2, ?x3, ?x4, ?x5, ?x6, ?x7, ?x8] :: {
  HasRevisionPremise(?x1, ?x2), ShortDescription-Geometry(?x1, ?x3),
  IsDefinedAsCommodityClassInSpec(?x1, ?x5), HasGeometricalPattern(?x1, ?x8),
  IsDefinedInIndustryStandard(?x1, ?x4), LongDescription-Geometry(?x1, ?x6),
  CADDescription-Geometry(?x1, ?x7)
} .
```

Tutorial: Scalable construction of sustainable knowledge bases

Martin G. Skjæveland Leif Harald Karlsen
Daniel P. Lupp Melinda Hodkiewicz

Summing up

Reasonable Ontology Templates (OTTR): Example

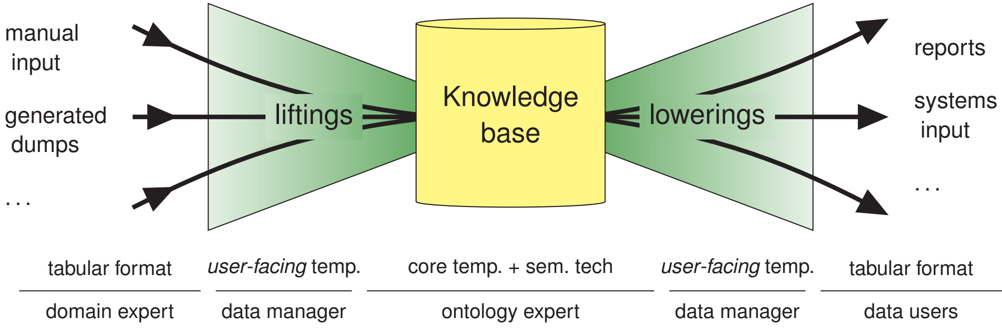
```
p:Margherita rdf:type owl:Class .
p:Margherita rdfs:subClassOf p:Pizza .
p:Margherita rdfs:label "Margherita"@it .
```

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

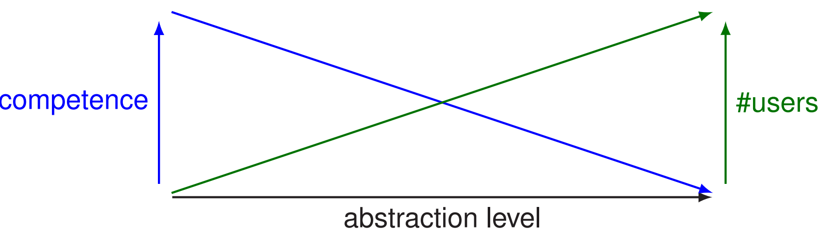
- Substitution
- Macro expansion

```
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) .
```

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

On the horizon

- General mapping language (bOTTR)
- use templates as queries
- Semantic reasoning and querying using templates
- if-then dependencies between templates
- GUI/Web app for template creation
- tools for constructions of terms, e.g., URI generation, unit conversion

Discussion Points

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

- `https://ottr.xyz/#Contact` — mailinglist, issue tracker, email

Feedback

`https://tinyurl.com/ottr-feedback`