Mechanisms for Importing Modules

Bijan Parsia Uli Sattler *Thomas Schneider*

School of Computer Science, University of Manchester, UK

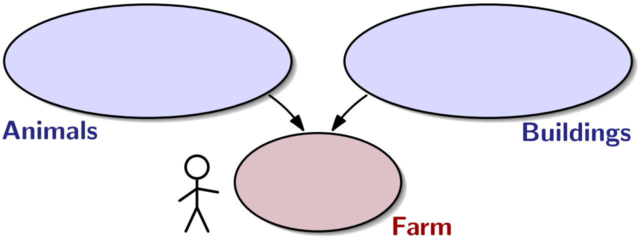
OWLED, 23 October 2009

And now for . . .

1. Motivation
2. An Import Mechanism for OWL
3. Discussion
4. Conclusion

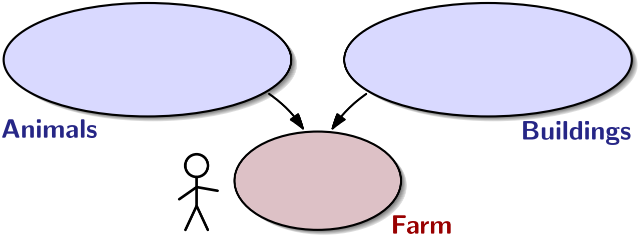
Why reuse ontologies?

Borrow knowledge



Why reuse ontologies?

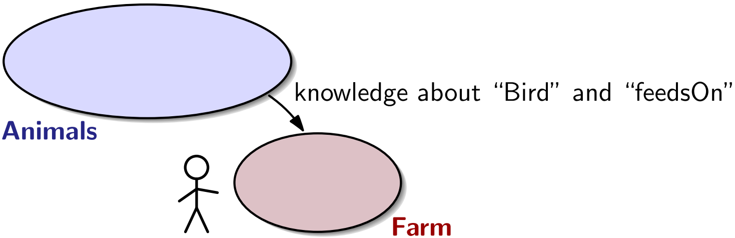
Borrow knowledge



- Provides access to well-established knowledge
- Doesn't require expertise in external disciplines

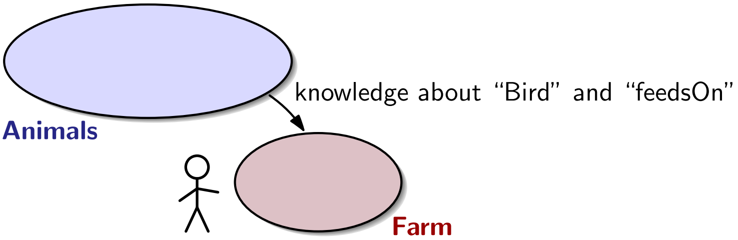
Why reuse ontologies?

Borrow knowledge about certain terms



Why reuse ontologies?

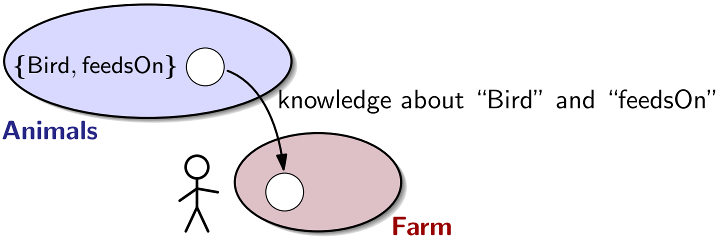
Borrow knowledge about certain terms



- Easy solution: Import(**Animals**) ✓ *Supported by OWL*

Why reuse ontologies?

Borrow knowledge about certain terms



- Easy solution: Import(**Animals**) ✓ *Supported by OWL*
- Economic solution: Import(appropriate module of **Animals**)

What is an “appropriate module”?

It should provide . . .

Coverage Import *everything* relevant for the chosen terms.

Economy Import *only* what’s relevant for them.
Compute that part quickly.

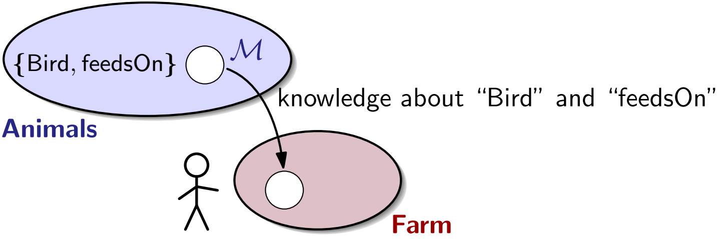
What is an “appropriate module”?

It should provide . . .

Coverage Import *everything* relevant for the chosen terms.

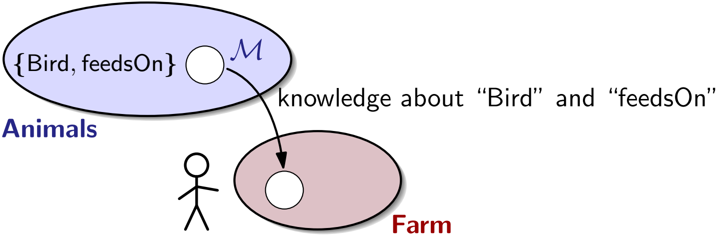
Economy Import *only* what’s relevant for them.
Compute that part quickly.

Covering modules



$$\begin{array}{lll} \textbf{Animals} & \text{entails} & \text{“Bird } \texttt{subClassOf} \text{ feedsOn } \texttt{some} \text{ Thing”} \\ & \Downarrow & \\ \mathcal{M} & \text{entails} & \text{“Bird } \texttt{subClassOf} \text{ feedsOn } \texttt{some} \text{ Thing”} \end{array}$$

Covering modules



Farm ∪ **Animals** entails "Bird `subClassOf` feedsOn `some` Thing"

$\Downarrow$

Farm ∪ $\mathcal{M}$ entails "Bird `subClassOf` feedsOn `some` Thing"

Coverage *and* economy . . .

. . . is provided by only very few module notions

- locality-based modules
- MEX-modules
- modules based on $\mathcal{E}$-connections

Our proposal

Look at how modular import *might* be realised in OWL:

- Modular import statements
- Changes required to syntax and structural specification
- Discussion of design choices

Our proposal

Look at how modular import *might* be realised in OWL:

- Modular import statements
- Changes required to syntax and structural specification
- Discussion of design choices

This is open for discussion!

And now . . .

1. Motivation
2. An Import Mechanism for OWL
3. Discussion
4. Conclusion

The new import mechanism. . .

. . . modifies the `directlyImports` association

Current state

`Import(`**Animals**`)`

The new import mechanism. . .

. . . modifies the `directlyImports` association

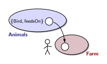
Current state

`Import(`**Animals**`)`

Addition

`ImportModule(` Bird feedsOn **Animals**`)`

The new import mechanism. . .



. . . modifies the `directlyImports` association

Current state

`Import(`**Animals**`)`

Addition

$$\texttt{ImportModule(}\ \underbrace{\text{Bird feedsOn}}_{\text{interface signature}}\quad \underbrace{\textbf{Animals}}_{\text{ontology IRI}}\texttt{)}$$

Structural specification

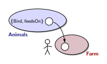
Only *one* addition to canonical parsing necessary:

`ImportModule`(Bird feedsOn **Animals**)

$\Downarrow$

1. Compute module $\mathcal{M}$ of **Animals** for $\{\text{Bird}, \text{feedsOn}\}$
2. Replace the above statement with `Import`($\mathcal{M}$)

No further changes required



Import closure of $\mathcal{O}$

Set consisting of $\mathcal{O}$ and all ontologies in import statements in $\mathcal{O}$

Axiom closure of $\mathcal{O}$

Set of all axioms in the import closure of $\mathcal{O}$

And now . . .

1 Motivation

2 An Import Mechanism for OWL

3 Discussion

4 Conclusion

Different behaviour

With plain `Import`, these properties are trivial:

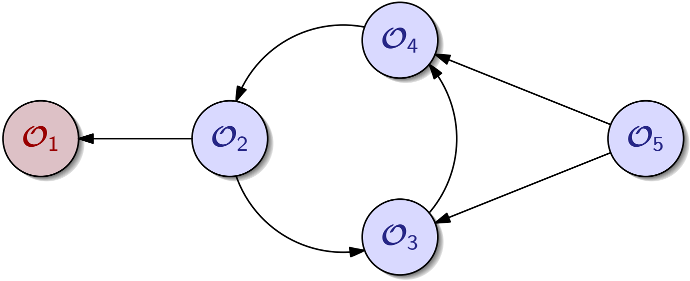
$$\begin{aligned}
\mathcal{O}_1 \text{ imports } \mathcal{O}_2 &= \mathcal{O}_2 \text{ imports } \mathcal{O}_1 \\
(\mathcal{O}_1 \text{ imports } \mathcal{O}_2) \text{ imports } \mathcal{O}_3 &= \mathcal{O}_1 \text{ imports } (\mathcal{O}_2 \text{ imports } \mathcal{O}_3) \\
\mathcal{O}_1 \text{ imports } (\mathcal{O}_2 \text{ imports } \mathcal{O}_3) &= \mathcal{O}_1 \text{ imports } (\mathcal{O}_3 \text{ imports } \mathcal{O}_2)
\end{aligned}$$

Different behaviour

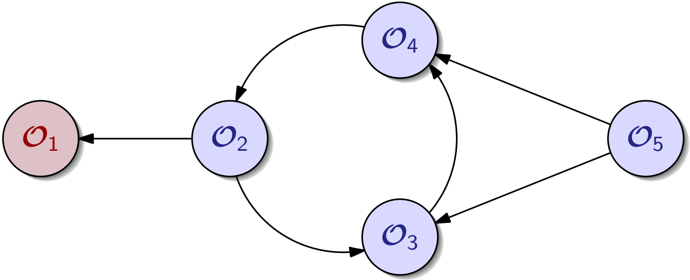
With the new `ImportModule`, they don't hold in general:

$$\begin{aligned}
\mathcal{O}_1 \text{ imports } \mathcal{O}_2 &\neq \mathcal{O}_2 \text{ imports } \mathcal{O}_1 \\
(\mathcal{O}_1 \text{ imports } \mathcal{O}_2) \text{ imports } \mathcal{O}_3 &\neq \mathcal{O}_1 \text{ imports } (\mathcal{O}_2 \text{ imports } \mathcal{O}_3) \\
\mathcal{O}_1 \text{ imports } (\mathcal{O}_2 \text{ imports } \mathcal{O}_3) &\neq \mathcal{O}_1 \text{ imports } (\mathcal{O}_3 \text{ imports } \mathcal{O}_2)
\end{aligned}$$

Parsing order for cycles and import chains

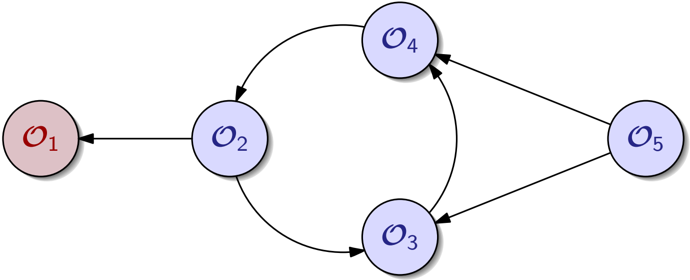


Parsing order for cycles and import chains



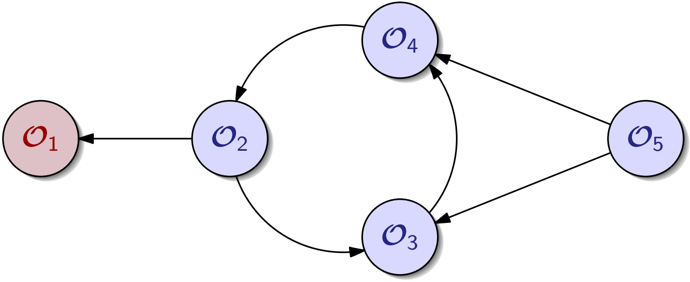
Parse $\mathcal{O}_5$

Parsing order for cycles and import chains



Parse $\mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4$ $\longleftarrow$ Parse $\mathcal{O}_5$

Parsing order for cycles and import chains



Parse $\mathcal{O}_1$ ← Parse $\mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4$ ← Parse $\mathcal{O}_5$

The choice of module type

Kind of "module"	Size	Extraction	Covered languages
The whole ontology	*big*	easy	any
based on conservativity	minimal	*hard*	*few*
MEX (Liverpool)	minimal	easy	*acyclic OWL EL*
based on $\mathcal{E}$-connections	small	easy	OWL 1 DL
based on locality	small	easy	≈ OWL 2 DL

The choice of module type

Kind of "module"	Size	Extraction	Covered languages
The whole ontology	*big*	easy	any
based on conservativity	minimal	*hard*	*few*
MEX (Liverpool)	minimal	easy	*acyclic OWL EL*
based on $\mathcal{E}$-connections	small	easy	OWL 1 DL
based on locality	small	easy	≈ OWL 2 DL

- Module experts' recommendation: locality-based modules
- + desirable robustness properties

  + implemented in the OWL API

Directive versus integrity constraint

Two ways of reading the statement

`ImportModule`(Bird feedsOn **Animals**)

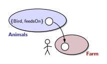
- *As a directive:*

  Extract the module for {Bird, feedsOn} from **Animals** and import it into **Farm**.

- *As an integrity constraint:*

  Make sure that **Farm** does not reuse terms other than ‘Bird’, ‘feedsOn’ from **Animals**.

Problems with the integrity constraint



Make sure that **Farm** does not reuse
terms other than ‘Bird’, ‘feedsOn’ from **Animals**.

Idea:

Module only guarantees to cover knowledge about ‘Bird’, ‘feedsOn’ – not e.g. ‘Slug’

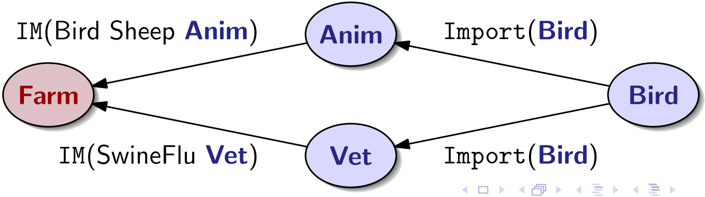
Problems with the integrity constraint

Make sure that **Farm** does not reuse terms other than ‘Bird’, ‘feedsOn’ from **Animals**.

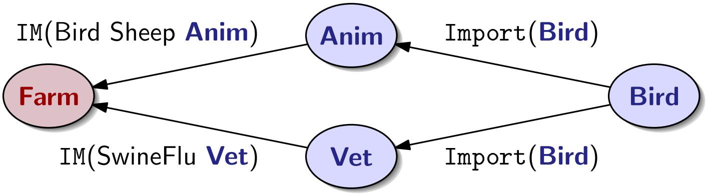
Idea:

Module only guarantees to cover knowledge about ‘Bird’, ‘feedsOn’ – not e.g. ‘Slug’

Clash:



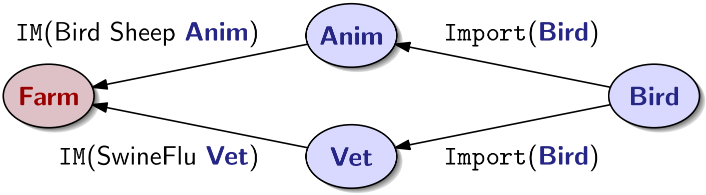
Problems with the integrity constraint



- *Permission* over *prohibition*?

  When deleting import statements, terms need to be traced!

Problems with the integrity constraint

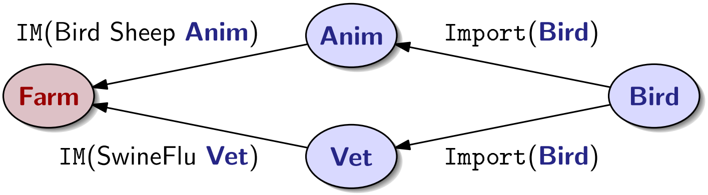


- *Permission* over *prohibition*?

  When deleting import statements, terms need to be traced!

- More unintuitive effects for cyclic import

Problems with the integrity constraint



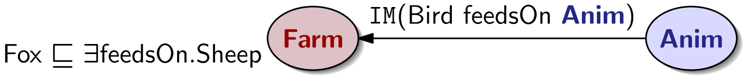
- *Permission* over *prohibition*?

  When deleting import statements, terms need to be traced!

- More unintuitive effects for cyclic import

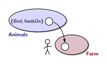
- *Lesson learnt:* Drop integrity constraint
  – except in “flat” import scenarios (e.g., collaboration)

Directive has a pitfall, too



- It can be unsafe to use these terms if they occur in **Animals**.
- Not clear whether they are in the module
- *Possible solution:*
  Treat them as distinct from the terms in **Animals**.

Variation for convenience

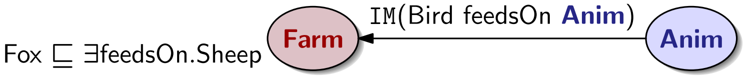


Drop the interface signature:

`ImportModule(`**Animals**`)`

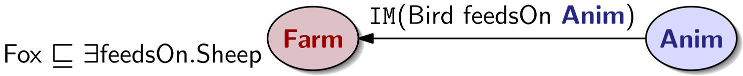
Interface signature = all terms from **Animals** reused in **Farm**

Where is the module computed?


$\text{Fox} \sqsubseteq \exists \text{feedsOn.Sheep}$
Farm
IM(Bird feedsOn **Anim**)
Anim

- In **Farm**?
  More economic than importing full **Animals**
- In **Animals**?
  Reduces communication, requires suitable protocols

Where is the module computed?



- In **Farm**?
  More economic than importing full **Animals**
- In **Animals**?
  Reduces communication, requires suitable protocols

Always: if **Animal** changes, the module needs to be recomputed

And now . . .

1 Motivation

2 An Import Mechanism for OWL

3 Discussion

4 Conclusion

Conclusion

Insights:

- Proposed extension is small and harmless
- Can be an official or unofficial extension of OWL

Conclusion

Insights:

- Proposed extension is small and harmless
- Can be an official or unofficial extension of OWL

Next steps:

- Experimental evaluation
- Guidance for specifying the interface signature
- Collaborative ontology development based on modules: methodology + tools

Conclusion

Insights:

- Proposed extension is small and harmless
- Can be an official or unofficial extension of OWL

Next steps:

- Experimental evaluation
- Guidance for specifying the interface signature
- Collaborative ontology development based on modules: methodology + tools

Thank you.