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

Animals

Farm

Buildings
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

Animals

knowledge about “Bird” and “feedsOn”

Farm
Why reuse ontologies?

Borrow knowledge about certain terms

Animals

knowledge about “Bird” and “feedsOn”

Farm

- Easy solution: Import(Animals) ✔ Supported by OWL
Why reuse ontologies?

Borrow knowledge about certain terms

```
{Bird, feedsOn}
```

knowledge about “Bird” and “feedsOn”

- 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

Animals

\{ \text{Bird, feedsOn} \} \quad \mathcal{M}

\text{knowledge about "Bird" and "feedsOn"}

Animals \quad \text{entails} \quad \text{"Bird subClassOf feedsOn some Thing"}

\mathcal{M} \quad \text{entails} \quad \text{"Bird subClassOf feedsOn some Thing"}
Covering modules

{Bird, feedsOn} \(\mathcal{M}\) 

knowledge about “Bird” and “feedsOn”

Animals

Farm

\(\text{Farm} \cup \text{Animals}\) entails “Bird subClassOf feedsOn some Thing”

\(\downarrow\)

\(\text{Farm} \cup \mathcal{M}\) entails “Bird subClassOf feedsOn some Thing”
... 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 . . .
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
ImportModule( Bird feedsOn Animals)

interface signature
ontology IRI
Only one addition to canonical parsing necessary:

\[
\text{ImportModule}(\text{Bird feedsOn Animals})
\]

1. Compute module \( \mathcal{M} \) of \textbf{Animals} for \{Bird, 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 $\mathcal{O}$
Motivation

2 An Import Mechanism for OWL

3 Discussion

4 Conclusion

And now . . .
Different behaviour

With plain Import, these properties are trivial:

\[ O_1 \text{ imports } O_2 = O_2 \text{ imports } O_1 \]
\[ (O_1 \text{ imports } O_2) \text{ imports } O_3 = O_1 \text{ imports } (O_2 \text{ imports } O_3) \]
\[ O_1 \text{ imports } (O_2 \text{ imports } O_3) = O_1 \text{ imports } (O_3 \text{ imports } O_2) \]
Different behaviour

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

\[ O_1 \text{ imports } O_2 \neq O_2 \text{ imports } O_1 \]
\[ (O_1 \text{ imports } O_2) \text{ imports } O_3 \neq O_1 \text{ imports } (O_2 \text{ imports } O_3) \]
\[ O_1 \text{ imports } (O_2 \text{ imports } O_3) \neq O_1 \text{ imports } (O_3 \text{ imports } O_2) \]
Parsing order for cycles and import chains

Animals

\[ \text{Fa}rm \{ \text{Bird}, \text{feedsOn} \} \]
Parsing order for cycles and import chains

Parse $O_5$
Parsing order for cycles and import chains

Parse $O_2, O_3, O_4$  \[\rightarrow\]  Parse $O_5$
Parsing order for cycles and import chains

Parse $O_1$  Parse $O_2, O_3, O_4$  Parse $O_5$
The choice of module type

<table>
<thead>
<tr>
<th>Kind of “module”</th>
<th>Size</th>
<th>Extraction</th>
<th>Covered languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>The whole ontology</td>
<td>big</td>
<td>easy</td>
<td>any</td>
</tr>
<tr>
<td>based on conservativity</td>
<td>minimal</td>
<td>hard</td>
<td>few</td>
</tr>
<tr>
<td>MEX (Liverpool)</td>
<td>minimal</td>
<td>easy</td>
<td>acyclic OWL EL</td>
</tr>
<tr>
<td>based on $\mathcal{E}$-connections</td>
<td>small</td>
<td>easy</td>
<td>OWL 1 DL</td>
</tr>
<tr>
<td>based on locality</td>
<td>small</td>
<td>easy</td>
<td>$\approx$ OWL 2 DL</td>
</tr>
</tbody>
</table>
# The choice of module type

<table>
<thead>
<tr>
<th>Kind of “module”</th>
<th>Size</th>
<th>Extraction</th>
<th>Covered languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>The whole ontology</td>
<td>big</td>
<td>easy</td>
<td>any</td>
</tr>
<tr>
<td>based on conservativity</td>
<td>minimal</td>
<td>hard</td>
<td>few</td>
</tr>
<tr>
<td>MEX (Liverpool)</td>
<td>minimal</td>
<td>easy</td>
<td>acyclic OWL EL</td>
</tr>
<tr>
<td>based on $E$-connections</td>
<td>small</td>
<td>easy</td>
<td>OWL 1 DL</td>
</tr>
<tr>
<td>based on locality</td>
<td>small</td>
<td>easy</td>
<td>$\approx$ OWL 2 DL</td>
</tr>
</tbody>
</table>

- 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: IM(Bird Sheep Anim) IM(SwineFlu Vet) IM(Bird) Import(Bird)
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

```
Anim
Farm Import(Bird)
IM(Bird Sheep Anim)
IM(SwineFlu Vet)
Vet Import(Bird)
```
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?

- 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.