DAML+OIL Technical Detail

Ian Horrocks
horrocks@cs.man.ac.uk

University of Manchester
Manchester, UK
Talk Outline

Overview of language design and motivation

Basic features
☞ quick review of walkthrough

Advanced features
☞ details not (sufficiently) covered in the walkthrough

Tricks of the Trade
☞ getting the most out of DAML+OIL

Limitations
☞ what it can’t do

Implementation challenges
Overview of Language Design and Motivation
Most existing Web resources only human understandable
  ● Markup (HTML) provides *rendering information*
  ● Textual/graphical information for **human consumption**

Semantic Web aims at **machine understandability**
  ● **Semantic** markup will be added to web resources
  ● Markup will use **Ontologies** for shared understanding

Requirement for a suitable ontology language
  ● Compatible with existing Web standards (XML, RDF)
  ● Captures common KR idioms
  ● Formally specified and of adequate expressive power
  ● Amenable to machine processing
    ➔ Can provide reasoning support

**DAML+OIL** language developed to meet these requirements
DAML+OIL Language Overview

DAML+OIL is an **ontology** language

☞ Describes **structure** of the domain (i.e., a Tbox)
  - RDF used to describe specific **instances** (i.e., an Abox)

☞ Structure described in terms of **classes** and **properties**

☞ Ontology consists of set of **axioms**
  - E.g., asserting class subsumption/equivalence

☞ Classes can be names or **expressions**
  - Various **constructors** provided for building class expressions

☞ **Expressive power** determined by
  - Kinds of class (and property) constructor supported
  - Kinds of axiom supported
Basic Features
Classes and Axioms

Ontology consists of set of axioms, e.g., asserting facts about classes:

```xml
<daml:Class rdf:ID="Animal"/>

<daml:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="MarriedPerson">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Person"/>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasSpouse"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>
```
Properties

Can also assert facts about properties, e.g.:

```xml
<daml:ObjectProperty rdf:ID="hasParent"/>

<daml:UniqueProperty rdf:ID="hasMother">
    <rdfs:subPropertyOf rdf:resource="#hasParent"/>
    <rdfs:range rdf:resource="#Female"/>
</daml:UniqueProperty>

<daml:TransitiveProperty rdf:ID="descendant"/>

<daml:ObjectProperty rdf:ID="hasChild">
    <daml:inverseOf rdf:resource="#hasParent"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="hasMom">
    <daml:samePropertyAs rdf:resource="#hasMother"/>
</daml:ObjectProperty>
```
Can use XMLS datatypes and values instead of classes and individuals:

```xml
<daml:DatatypeProperty rdf:ID="age">
  <rdf:type rdf:resource="" ../daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="" ../XMLSchema#nonNegativeInteger"
</daml:DatatypeProperty>

<xsd:simpleType name="over17">
  <xsd:restriction base="xsd:positiveInteger">
    <xsd:minInclusive value="18"/>
  </xsd:restriction>
</xsd:simpleType>

<daml:Class rdf:ID="Adult">
  <daml:Restriction>
    <daml:onProperty rdf:resource="#age"/>
    <daml:hasClass rdf:resource="#over17"/>
  </daml:Restriction>
</daml:Class>
```
Individuals

Can also assert facts about individuals, e.g.:

```xml
<Person rdf:ID="John"/>
<Person rdf:ID="Mary"/>

<rdf:Description rdf:about="#John">
  <hasParent:resource="#Mary"/>
  <age>25</age>
</rdf:Description>

<rdf:Description rdf:about="#John">
  <differentIndividualFrom:resource="#Mary"/>
</rdf:Description>

<rdf:Description rdf:about="#Clinton">
  <sameIndividualAs:resource="#BillClinton"/>
</rdf:Description>
```
Advanced Features
## Overview of Class Expressions

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>$C_1 \cap \ldots \cap C_n$</td>
<td>Human $\cap$ Male</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \cup \ldots \cup C_n$</td>
<td>Doctor $\cup$ Lawyer</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\neg C$</td>
<td>$\neg$Male</td>
</tr>
<tr>
<td>oneOf</td>
<td>${x_1 \ldots x_n}$</td>
<td>{john, mary}</td>
</tr>
<tr>
<td>toClass</td>
<td>$\forall P.C$</td>
<td>$\forall$ hasChild.Doctor</td>
</tr>
<tr>
<td>hasClass</td>
<td>$\exists P.C$</td>
<td>$\exists$ hasChild.Lawyer</td>
</tr>
<tr>
<td>hasValue</td>
<td>$\exists P.{x}$</td>
<td>$\exists$ citizenOf.{USA}</td>
</tr>
<tr>
<td>minCardinalityQ</td>
<td>$\geq nP.C$</td>
<td>$\geq 2$ hasChild.Lawyer</td>
</tr>
<tr>
<td>maxCardinalityQ</td>
<td>$\leq nP.C$</td>
<td>$\leq 1$ hasChild.Male</td>
</tr>
<tr>
<td>cardinalityQ</td>
<td>$=nP.C$</td>
<td>$=1$ hasParent.Female</td>
</tr>
</tbody>
</table>

- XMLS **datatypes** can be used in restrictions
- Arbitrary **nesting** of constructors
  - E.g., $\forall$ hasChild.(Doctor $\sqcap \exists$ hasChild.Doctor)
Most basic components of class expressions are **names**

- E.g., Person, Building
- Two **built-in** (pre-defined) class names:
  - **Thing** — class whose extension is whole (object) domain
  - **Nothing** — class whose extension is empty
- They are just “syntactic sugar”
  - **Thing** $\equiv C \cup \neg C$ for any class $C$
  - **Nothing** $\equiv \neg \text{Thing}$
Class Expressions: Restrictions

- Restrictions are classes: class of all objects satisfying restriction
- Basic structure is *property* plus restrictions on
  - *type* and/or
  - *number*

of objects that can be related to members of class via that property
toClass Restrictions

☞ E.g.:

```xml
<daml:Restriction>
  <daml:onProperty rdf:resource="#hasParent"/>
  <daml:toClass rdf:resource="#Person"/>
</daml:Restriction>
```

class of objects all of whose parents are persons

☞ Analogous universal quantification (∀) in FOL

☞ Analogous to box (□) in modal logic
toClass Restrictions

☞ Can be seen as local/relativised property range

```xml
<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </daml:Restriction>
</rdfs:subClassOf>
</daml:Class>
```

☞ Conversely, range is like asserting toClass restriction w.r.t. Thing

☞ Some “strange” inferences:

- instances with no conflicting property assertions may not be members of class (open world) — c.f. peter
- instances (provably) without any such property are members of class — c.f. paul
hasClass Restrictions

E.g.:

```xml
<daml:Restriction>
  <daml:onProperty rdf:resource="#hasFriend"/>
  <daml:hasClass rdf:resource="#Republican"/>
</daml:Restriction>
```

class of objects that have some friend that is a Republican

- Analogous existential quantification (∃) in FOL
- Analogous to diamond (◊) in modal logic
- Individuals with no relevant property assertions may still be members of class (incomplete knowledge)
**hasValue Restrictions**

☞ E.g.:

```xml
<daml:Restriction>
  <daml:onProperty rdf:resource="#hasFriend"/>
  <daml:hasValue rdf:resource="#Nixon"/>
</daml:Restriction>
```

class of objects that have some friend that is Nixon

☞ **Just a special case of hasClass using oneOf**

```xml
<daml:Restriction>
  <daml:onProperty rdf:resource="#hasFriend"/>
  <daml:hasClass>
    <daml:oneOf rdf:parseType="daml:collection">
      <rdf:Description rdf:about="#Nixon"/>
    </daml:oneOf>
  </daml:hasClass>
</daml:Restriction>
```
cardinality Restrictions

☞ E.g.:

```xml
<daml:Restriction>
  <daml:onProperty rdf:resource="#hasFriend"/>
  <daml:minCardinalityQ>2</daml:minCardinalityQ>
  <daml:hasClassQ rdf:resource="#Republican"/>
</daml:Restriction>
```

class of objects that have at least 2 friends that are Republicans

☞ Can specify min, max and exact cardinalities
  ● exact is shorthand for max plus min pair

☞ minCardinalityQ is generalisation of hasClass, e.g.:

```xml
<daml:Restriction daml:minCardinalityQ=1>
  <daml:onProperty rdf:resource="#hasFriend"/>
  <daml:hasClassQ rdf:resource="#Republican"/>
</daml:Restriction>
```
equivalent to hasClass Republican.
Also exist versions without qualifying concepts, e.g.:

```xml
<daml:Restriction>
  <daml:onProperty rdf:resource="#hasFriend"/>
  <daml:minCardinality>3</daml:minCardinality>
</daml:Restriction>
```

class of objects that have at least 3 friends

Same as Q version with qualifying class as `Thing`

```xml
<daml:Restriction>
  <daml:onProperty rdf:resource="#hasFriend"/>
  <daml:minCardinalityQ>3</daml:minCardinalityQ>
  <daml:hasClassQ rdf:resource=".../daml+oil#Thing"/>
</daml:Restriction>
```
cardinality Restrictions

Note that no unique name assumption:

- individual only instance of above class if it has 3 (provably) different friends
- maxCardinality restrictions can lead to sameIndividualAs inferences
RDF Syntax

- Syntax allows multiple properties/classes in single restriction
  
  ```xml
  <daml:Restriction>
    <daml:onProperty rdf:resource="#hasFriend"/>
    <daml:hasClass rdf:resource="#hasFriend"/>
    <daml:toClass rdf:resource="#Republican"/>
  </daml:Restriction>
  ```

- Result may not be as expected
  - at least one Republican friend and all friends Republicans
  - at least one Republican friend \textit{iff} all friends Republicans

- Bottom line: avoid such constructs! — use \texttt{intersectionOf 2} (or more) separate restrictions
Existentially defined classes

Class defined by listing members, e.g.:

```xml
<daml:Class>
  <daml:oneOf rdf:parseType="daml:collection">
    <rdf:Description rdf:about="#Italy">
    </rdf:Description>
    <rdf:Description rdf:about="#France">
    </rdf:Description>
  </daml:oneOf>
</daml:Class>
```
Class Expressions: Enumerations

- Strange properties compared to other classes
  - e.g., cardinality of class is known (2 in the above case)
- Powerful/useful but hard to deal with computationally
- Can sometimes substitute union of (primitive) classes, e.g.:
  ```xml
  <daml:Class>
    <daml:unionOf rdf:parseType="daml:collection">
      <daml:Class rdf:about="#Italy"/>
      <daml:Class rdf:about="#France"/>
    </daml:unionOf>
  </daml:Class>
  ```
  - but (max) cardinality inferences may be lost
Class Expressions: Booleans

- Standard boolean constructors (intersection, union, complement) can be used to combine classes
- Boolean constructors are properties not a classes
  - Class “wrapper” needed for nesting, e.g.:
    <daml:Class rdf:ID="Woman">
      <daml:intersectionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#Person"/>
        <rdfs:Class>
          <daml:complementOf rdf:resource="#Male"/>
        </rdfs:Class>
      </daml:intersectionOf>
    </daml:Class>
Datatypes

Can use XMLS datatypes and values instead of classes and individuals:

- Domain of classes and datatypes considered disjoint
  - no object can be both class instance and datatype value

- Two types of property: ObjectProperty and DatatypeProperty
  - ObjectProperty used with classes/individuals
  - DatatypeProperty used with datatypes/values

- Can use arbitrary XMLS datatypes
  - built-in (primitive and derived), e.g., \texttt{xsd:decimal}
  - user defined/derived, e.g., sub-ranges

- Datatypes can be used in restrictions and as range of datatype properties

- Data values can be used in \texttt{hasValue} and in RDF “ground facts”
Property Expressions

☞ Only property operator directly supported is inverseOf

☞ Other operators such as composition (\(\circ\)) and union (\(\sqcup\)) can sometimes be expanded out

- \(\exists(P_1 \circ P_2).C \equiv \exists P_1.(\exists P_2.C')\)
- \(\forall(P_1 \circ P_2).C \equiv \forall P_1.(\forall P_2.C')\)
- \(\exists(P_1 \sqcup P_2).C \equiv (\exists P_1.C') \sqcup (\exists P_2.C')\)
- \(\forall(P_1 \sqcup P_2).C \equiv (\forall P_1.C') \sqcap (\forall P_2.C')\)

☞ Can’t capture/expand
- intersection of properties
- property expressions (except inverse) in cardinality restrictions, e.g., \(\leq_1(P_1 \circ P_2)\) — but see “tricks of the trade”
## DAML+OIL Overview: Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>subClassOf</td>
<td>$C_1 \sqsubseteq C_2$</td>
<td>Human $\sqsubseteq$ Animal $\sqcap$ Biped</td>
</tr>
<tr>
<td>sameClassAs</td>
<td>$C_1 \equiv C_2$</td>
<td>Man $\equiv$ Human $\sqcap$ Male</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>hasDaughter $\sqsubseteq$ hasChild</td>
</tr>
<tr>
<td>samePropertyAs</td>
<td>$P_1 \equiv P_2$</td>
<td>cost $\equiv$ price</td>
</tr>
<tr>
<td>sameIndividualAs</td>
<td>${x_1} \equiv {x_2}$</td>
<td>${\text{President_Bush}} \equiv {\text{G_W_Bush}}$</td>
</tr>
<tr>
<td>disjointWith</td>
<td>$C_1 \sqsubseteq \neg C_2$</td>
<td>Male $\sqsubseteq \neg$Female</td>
</tr>
<tr>
<td>differentIndividualFrom</td>
<td>${x_1} \sqsubseteq \neg {x_2}$</td>
<td>${\text{john}} \sqsubseteq \neg {\text{peter}}$</td>
</tr>
<tr>
<td>inverseOf</td>
<td>$P_1 \equiv P_2^-$</td>
<td>hasChild $\equiv$ hasParent$^-$</td>
</tr>
<tr>
<td>transitiveProperty</td>
<td>$P^+ \sqsubseteq P$</td>
<td>ancestor$^+ \sqsubseteq$ ancestor</td>
</tr>
<tr>
<td>uniqueProperty</td>
<td>$\top \sqsubseteq \leq 1P$</td>
<td>$\top \sqsubseteq \leq 1\text{hasMother}$</td>
</tr>
<tr>
<td>unambiguousProperty</td>
<td>$\top \sqsubseteq \leq 1P^-$</td>
<td>$\top \sqsubseteq \leq 1\text{isMotherOf}^-$</td>
</tr>
</tbody>
</table>
Class Axioms

Allow facts to be asserted w.r.t. classes/class expressions, e.g., equivalence

- All class axioms can be transformed into `subClassOf`, e.g.:

  \[ C1 \equiv C2 \iff C1 \subseteq C2 \text{ and } C2 \subseteq C1 \]
  \[ C1 \text{ disjointWith } C2 \iff C1 \subseteq \neg C2 \]

  - but different forms may be useful for modelling and/or reasoning

- Most common axiom is `sub/sameClass` with name on l.h.s., e.g.:

  \[ \text{Triangle} \equiv \text{Polygon} \cap =3 \text{ hasAngle.} \]

  - sometimes called a **definition**
  - can have as many definitions as we like
  - no way to distinguish “main” definition
Class Axioms

multiple subClass axioms with same l.h.s. can be gathered together or separated, e.g.:

\[ C_1 \sqsubseteq C_2, \quad C_1 \sqsubseteq C_3 \quad \iff \quad C_1 \sqsubseteq C_2 \cap C_3 \]

but multiple equivalence axioms with same l.h.s. can **not** be gathered together

In general, both sides can be arbitrary expressions, e.g.:

\[ \text{Polygon} \sqsubseteq \exists \text{hasSide} \sqsubseteq =3 \text{hasAngle} \]

This feature is very powerful and allows many complex situations to be captured
subClass axioms can be seen as a form of rule, e.g.:

\[ C_1(x) \leftarrow C_2(x) \land P_1(x, y) \land P_2(y, z) \land C_3(z) \]

is equivalent to

\[ C_2 \sqcap \exists P_1. (\exists P_2. C_3) \sqsubseteq C_1 \]

Synonyms can also be captured by asserting name equivalence, e.g.:

\[ \text{Car} \equiv \text{Automobile} \]
Class Axioms

- No requirement to “define” class before use
  - But good practice in general (for detecting typos etc.)
- Axioms can be directly (or indirectly) cyclical, e.g.:

\[ \text{Person} \equiv \exists \text{hasParent}. \text{Person} \]

- Descriptive (standard FOL) semantics — not fixedpoint
Property Axioms

Allow facts to be asserted w.r.t. properties/property expressions, e.g.:

\[
\text{hasChild} \equiv \text{hasParent}^-
\]

- Equivalence reducible to subProperty as for classes
- Multiple axioms/definitions etc. as for classes
- Can also assert that a property is transitive
  - Useful/essential for part-whole, causality etc.
  - Easier to handle computationally than transitive closure operator
  - Can combine with subPropertyOf to get similar effect, e.g.:

\[
\text{directPartOf} \sqsubseteq \text{partOf} \text{ and transitive}(\text{partOf})
\]

similar to

\[
\text{directPartOf}^* \equiv \text{partOf}
\]

- Can only be applied to object properties
Property Axioms

☞ Symmetrical not directly supported but easily captured:

\[ \text{hasNeighbour} \equiv \text{hasNeighbour}^{-}\]

☞ Reflexive cannot be captured
Property Axioms

- Range/domain constraints equivalent to toClass restrictions on property/inverse subsuming Thing:

\[
\begin{align*}
\text{range}(P, C) & \iff \text{Thing} \sqsubseteq \forall P.C \\
\text{domain}(P, C) & \iff \text{Thing} \sqsubseteq \forall P^-.C
\end{align*}
\]

- Unique/unambiguous assertions equivalent to maxCardinality=1 restrictions on property/inverse subsuming Thing:

\[
\begin{align*}
\text{uniqueProperty}(P) & \iff \text{Thing} \sqsubseteq \leq 1P \\
\text{unambiguousProperty}(P) & \iff \text{Thing} \sqsubseteq \leq 1P^-
\end{align*}
\]

- Note that these are very strong statements
  - restriction asserted w.r.t. Thing
  - can result in “strange” (unexpected) inferences and/or compromise extensibility of ontology
  - almost always better asserted locally (particularly range/domain)
Individual Axioms

Allow facts to be asserted w.r.t. individuals, e.g., type

- RDF used for basic type/property assertions (Abox)

  \[
  \text{<Person rdf:ID="John"/>}
  \text{<rdf:Description rdf:about="#John">}
  \text{<hasParent:resource="#Mary"/>}
  \text{</rdf:Description>}
  \]
  i.e.,
  \[
  \text{John} \in \text{Person}, \langle \text{John}, \text{Mary} \rangle \in \text{hasParent}
  \]

- Can state same facts using DAML+OIL oneOf, e.g.:

  \[
  \text{<daml:oneOf rdf:parseType="daml:collection">}
  \text{<rdf:Description rdf:about="#John"/>}
  \text{</daml:oneOf>}
  \text{<rdfs:subClassOf rdf:resource="#Person"/>}
  \text{</daml:oneOf>}
  \]
Individual Axioms

- Datatype properties relate individuals to data values.
- Data values can be explicitly or implicitly typed, e.g.:

  `<rdf:Description rdf:about="#John"`  
  `<age>25</age>`  
  `<typedData>`  
  `</typedData>`  
  `<untypedData>1234</untypedData>`  
  `</rdf:Description>`
Individual Axioms

☞ No unique name assumption
☞ But can assert equality or inequality of individuals, e.g.:

```xml
<rdf:Description rdf:about="#Clinton">
  <differentIndividualFrom:resource="#Hillary"/>
  <sameIndividualAs:resource="#BillClinton"/>
</rdf:Description>
```

☞ Can again use oneOf to capture such (in)equalities

```xml
<daml:class>
  <daml:oneOf rdf:parseType="daml:collection">
    <rdf:Description rdf:about="#Clinton">
    </rdf:Description>
  </daml:oneOf>
  <rdfs:sameClassAs rdf:resource="#BillClinton"/>
</daml:class>
```
RDF Syntax

Slightly strange mixture of classes and properties, axioms and constructors

- Restrictions are classes
- Enumerations and booleans are properties
  - implicit sameClassAs axiom, e.g.:
    
    \[
    \text{<daml:Class rdf:ID="NonPerson">}
    \text{<daml:complementOf rdf:resource="#Person"/>}
    \text{</daml:Class>}
    \]

  - have to be “wrapped” in an anonymous class to combine (e.g., with other booleans) or assert subClassOf
    
    \[
    \text{<daml:Class rdf:ID="Car">}
    \text{<rdfs:subClassOf>}
    \text{<daml:Class>}
    \text{<daml:complementOf rdf:resource="#Person"/>}
    \text{</daml:Class>}
    \text{</rdfs:subClassOf>}
    \text{</daml:Class>}
    \]
Some constructors contain hidden axioms

- e.g., disjointUnionOf

```xml
<daml:Class rdf:about="#Person">
    <daml:disjointUnionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#Man"/>
        <daml:Class rdf:about="#Woman"/>
    </daml:disjointUnionOf>
</daml:Class>
```

includes **global** assertion about disjointness of Man and Woman

Combined restrictions also hidden axioms
Tricks of the Trade
Using Property Hierarchy

- Common requirement is to construct class where 2 properties have same value
  - e.g., class of “happyPerson” whose spouse is the same individual as their best friend
  - Can achieve something similar using subPropertyOf and cardinality restrictions:

$$\begin{align*}
\text{hasSpouse} & \sqsubseteq \text{hasSpouseOrBestFriend} \\
\text{hasBestFriend} & \sqsubseteq \text{hasSpouseOrBestFriend} \\
\text{happyPerson} & \sqsubseteq \mathbf{1} \text{hasSpouse} \sqcap \mathbf{1} \text{hasBestFriend} \sqsubseteq \text{hasSpouseOrBestFriend}
\end{align*}$$

- Note that all the properties must be locally unique
- Can also define bespoke part-whole hierarchy
oneOf is very powerful

E.g., can be define so called “spy-point”
- connected via some property to every object in domain

\[
\text{Thing} \sqsubseteq \exists P.\{\text{spy-point}\}
\]

Combined with inverse can be used to fix (min/max) cardinality of domain, e.g.:

\[
\{\text{spy-point}\} \sqsubseteq \leq 15P^-
\]
General Axioms

General axioms (expressions on l.h.s.) are very powerful

☞ Can capture (some kinds of) rules, e.g.:

\[
\text{period} = \text{lateGeorgian} \iff \text{culture} = \text{british} \\
\land \text{date} = 1760–1811
\]

can be captured as an axiom:

\[
\exists \text{culture.british} \\
\Box \exists \text{date.1760–1811} \sqsubseteq \exists \text{period.lateGeorgian}
\]

☞ Can be computationally expensive

- should relativise as much as possible
- e.g., above axiom only relevant to furniture
Other Useful Constructions

☞ Localised range/domain

\[ C \subseteq \forall P.D \]
\[ C \cap \geq 1P \subseteq D \]

☞ Localised unique/unambiguous

\[ C \subseteq \leq 1P \]
\[ C \subseteq \forall P.(\leq 1P1^-) \]
Limitations
What It Can’t Do

DAML+OIL has many limitations, mostly designed to maintain decidability/computability/well-definedness

☞ Limited property constructors
  ● e.g., no composition, transitive closure, product, ...

☞ Limited property types
  ● transitive and symmetrical, but not reflexive

☞ Only collection type is set
  ● e.g., no bags, lists

☞ Only unary and binary relations

☞ Restricted form of quantification (modal/guarded fragment)

☞ No comparison or aggregation of data values

☞ No defaults

☞ No variables (as in hybrid logics)

...
Implementation challenges
Implementation challenges

Even with existing language, challenges remain for would-be implementors

☞ Reasoning with oneOf is **hard**
  - decidable (contained in the C2 fragment of first order logic) but complexity increases from $\text{EXPTime}$ to $\text{NEXPTime}$
  - no known “practical” algorithm

☞ Scalability
  - class consistency in $\text{EXPTime}$ even without oneOf
  - inverse properties cause particular difficulties
  - web ontologies may be **large**

☞ Other reasoning tasks
  - Querying
  - Explanation
  - LCS/matching