Testing

Eva Blomqvist
eva.blomqvist@liu.se
Department of Computer and Information Science (IDA)
Linköpings universitet
Sweden

LiU expanding reality

Requirements?

- Functional requirements
  - Query results? – CQs
  - Inferences? – Reasoning requirements
  - Error checking? – Contextual statements
  - …

  "Verification testing"
- Non-functional requirements
  - Coverage
  - Efficiency
  - Documentation
  - Changeability – extendibility
  - …

  Ontology Evaluation
  Acceptance testing

Ontology Evaluation

- Many different perspectives!

- Example: Gómez-Pérez
  - Consistency – “correct model”
  - Completeness – “complete model”
  - Conciseness – “no redundancy or unessential axioms”
- Common ways to evaluate ontologies
  - Comparing to a “gold standard”
    - Other ontologies
    - Data source it should cover
    - Variants of precision and recall
  - Applying the ontology in an application
  - Manual evaluation – criteria, standards, requirements

Ontology Evaluation (cont.)
- What to evaluate?

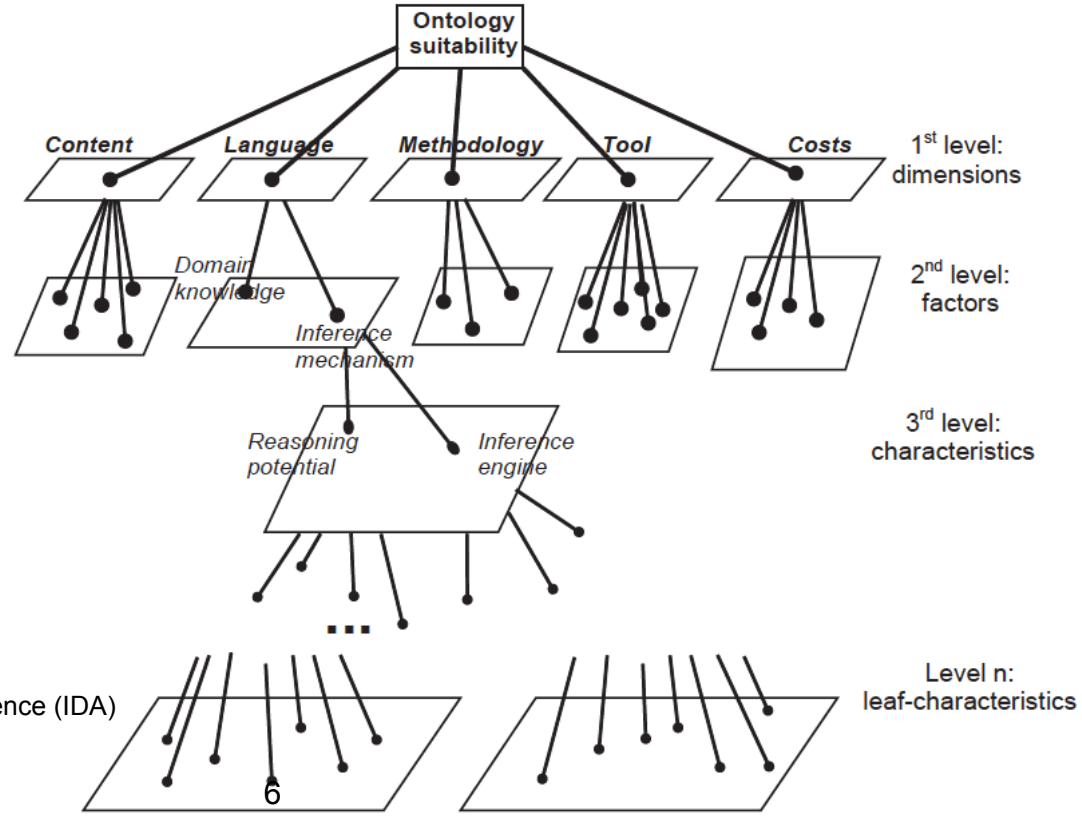
- Gangemi proposes three dimensions
  - Structural evaluations
  - Functional evaluations
  - Usability evaluations
- Structural evaluations look at an ontology from the perspective of syntax and formal semantics
- Functional evaluations look at an ontology from the perspective of the conceptualization and the goals of the ontology (e.g. the task)
- Usability evaluations look at an ontology from a communication perspective – what meaning does it actually convey to users?

Ontology Evaluation (cont.)
- Structural Measures

- Implemented in some OE tools, e.g. Swoogle
- Example of metrics
  - Size
  - Taxonomical depth
  - Property fan-out of classes
  - Class to property ratio
  - Instance to class ratio
  - Number of classes with formal definitions
  - Domain/range ratio of properties
  - OWL profile (or DL type) used
- Measures like this can give good information
  - …but needs to be used in combination with measures of coverage and domain-appropriateness, and interpreted with respect to our requirements, e.g. is bigger/deeper/… actually better?

Ontology Evaluation (cont.) - Example Approaches

- ONTOMETRIC
  - Multilayer framework of characteristics
  - Breaking down our requirements into detailed things that can be assessed on a scale – then combining assessments into an overall score
  - Manual



Ontology Evaluation (cont.) - Example Approaches

- OntoClean – taxonomy evaluation
- Tag concepts with certain characteristics, apply a set of rules that evaluates the combination of characteristics
  - Characteristics: Essence, Rigidity, Identity, Unity
  - Example: “being hard” – *essential* for a hammer, not for a sponge
  - For each concept, assess it according to above, then apply the rules ➔ result will tell you if your are misusing is-a relations
  - Will give good insights into your modeling choices
- Potential problems
  - Based on assumptions what “correctness” means
  - Very difficult to apply – may turn into random assessments

Ontology Evaluation (cont.)
- Usability Measures

- Level of documentation
- Level of axiomatization (?)
- Changeability - extensibility
- Comparison and analysis with respect to other ontologies – e.g. ontology agreement
  - Does this ontology agree with the ontologies we need to interoperate with?

Ontology Testing

- Evaluating the ontology with respect to its intended task – c.f. the functional dimension of Gangemi's perspective
- Testing = Evaluating the functional requirements

- At least three main types of tests – c.f. requirement types
  - Testing CQs through SPARQL queries
  - Testing reasoning requirements through verifying inferences
  - Testing contextual statements through "stress testing"

- Other types of testing
  - Inspection
  - Assessing coverage of domain vocabulary
  - Checking against ODPs

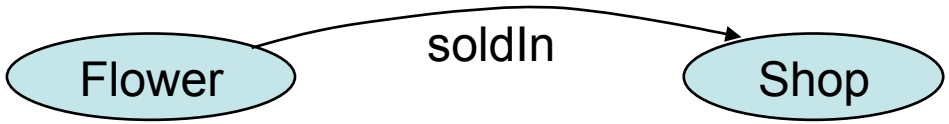
SPARQL

- Query language for RDF: http://www.w3.org/TR/rdf-sparql-query/
- Very similar to SQL in syntax… but be careful, the underlying data model is different!
- Based on triple patterns
  - RDF triple: subject – predicate – object
  - Ask for a value by replacing that value with a variable
  - Variables are marked by “?”

SPARQL basics

- SELECT ?x ?y …
  WHERE { <triple pattern> . <another triple pattern>}
  - The where-clause states the triple patterns that will be matched to the RDF graph of your knowledge base

- Example – Re-writing of a CQ into a SPARQL query
  - CQ: What flowers are sold in a particular shop?
  - Model:



Queries:

SELECT ?x WHERE { ?x a :Flower . ?x :soldIn ?y . ?y a :Shop}

SELECT ?x ?y WHERE { ?x :soldIn ?y}

SELECT ?x WHERE {?x :soldIn :myShop }

- The “a” is shorthand for rdf:type

SPARQL (cont.)

- Another simple query
  - Data:
    prefix foaf: <http://xmlns.com/foaf/0.1/> .
    :a foaf:name "Johnny Lee Outlaw" .
    :a foaf:mbox <mailto:jlow@example.com> .
    :b foaf:name "Peter Goodguy" .
    :b foaf:mbox <mailto:peter@example.org> .
    :c foaf:mbox <mailto:carol@example.org> .
  - Query:
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    SELECT ?name ?mbox
    WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox }
  - Query Result:

?name	**?mbox**
"Johnny Lee Outlaw“	mailto:jlow@example.com
"Peter Goodguy“	mailto:peter@example.org

SPARQL (cont.)

- Note: SPARQL is NOT SQL…
  - In the standard we are missing many aggregates compared to SQL – you cannot do counts and aggregations (although this exist in some engine implementations)

- Querying for literals is special: you have to give the type!

  Data

  ```
  :x  :p   "cat"@en .
  :y  :p   "42"^^xsd:integer .
  :z  :p   "abc"^^dt:specialDatatype .
  ```

  Example query

  ```
  SELECT ?x
  WHERE {?x ?p "abc"^^dt:specialDatatype . }
  ```

  Answer? – z

- Other SPARQL keywords: CONSTRUCT, and FILTER…

Anatomy of a Query

Declare prefix shortcuts (optional)

```
PREFIX foo: <…>
PREFIX bar: <…>
…
```

```
SELECT …
CONSTRUCT {…}
DESCRIBE …
ASK {…}
```

Query result clause

Define the dataset (optional)

```
FROM <…>
FROM NAMED <…>
```

```
WHERE {
    …
}
```

Query pattern

Query modifiers (optional)

```
GROUP BY …
HAVING …
ORDER BY …
LIMIT …
OFFSET …
BINDINGS …
```

Types of SPARQL Queries

SELECT queries

Project out specific variables and expressions:

```
SELECT ?c ?cap (1000 * ?people AS ?pop)
```

Project out all variables:

```
SELECT *
```

Project out distinct combinations only:

```
SELECT DISTINCT ?country
```

Results in a table of values :

?c	?cap	?pop
ex:France	ex:Paris	63,500,000
ex:Canada	ex:Ottawa	32,900,000
ex:Italy	ex:Rome	58,900,000

CONSTRUCT queries

Construct RDF triples/graphs:

```
CONSTRUCT {
?country a ex:HolidayDestination ;
ex:arrive_at ?capital ;
ex:population ?population .
} WHERE {…}
```

Results in RDF triples (in any RDF serialization):

```
ex:France a ex:HolidayDestination ;
ex:arrive_at ex:Paris ;
ex:population 635000000 .
ex:Canada a ex:HolidayDestination ;
ex:arrive_at ex:Ottawa ;
ex:population 329000000 .
```

ASK queries

Ask whether or not there are any matches:

```
ASK
```

Result is either "true" or "false" :

```
true, false
```

DESCRIBE queries

Describe the resources matched by the given variables:

```
DESCRIBE ?country
```

Result is RDF triples (in any RDF serialization) :

```
ex:France a geo:Country ;
ex:continent geo:Europe ;
ex:flag <http://…/flag-france.png> ;
…
```

courtesy of *Lee Feigenbaum*

Nuts & Bolts

URIs

Write full URIs:

```
<http://this.is.a/full/URI/written#out>
```

Abbreviate URIs with prefixes:

```
PREFIX foo: <http://this.is.a/URI/prefix#>
… foo:bar …
```

⇨ `http://this.is.a/URI/prefix#bar`

Shortcuts:

`a` ⇨ `rdf:type`

Literals

Plain literals:

`"a plain literal"`

Plain literal with language tag:

`"bonjour"@fr`

Typed literal:

`"13"^^xsd:integer`

Shortcuts:

`true` ⇨ `"true"^^xsd:boolean`

`3` ⇨ `"3"^^xsd:integer`

`4.2` ⇨ `"4.2"^^xsd:decimal`

Variables

Variables:

`?var1, ?anotherVar, ?and_one_more`

Comments

Comments:

```
# Comments start with a '#'
# continue to the end of the line
```

Triple Patterns

Match an exact RDF triple:

```
ex:myWidget ex:partNumber "XY24Z1" .
```

Match one variable:

```
?person foaf:name "Lee Feigenbaum" .
```

Match multiple variables:

```
conf:SemTech2009 ?property ?value .
```

courtesy of *Lee Feigenbaum*

SPARQL Example in TopBraid Composer

- Open the example ontology wine.rdf in TB Composer

- Query for all the Chardonnay's

```
SELECT ?vin
WHERE {
    ?vin a vin:Chardonnay .
}
```

- Query for all the Chardonnay's that have medium body

```
SELECT ?vin
WHERE {
    ?vin a vin:Chardonnay .
    ?vin vin:hasBody vin:Medium
}
```

1st Method: Testing Through SPARQL

- Given a CQ
  – Can you retrieve that information from the KB?
- Best practice: One requirement – One test case
- Proposed method:
  1. Create a new “test case” ontology and import the ontology to be tested
  2. Transform the CQ into a SPARQL query ★
  3. Add test data to the “test case” ontology – What are the expected results, based on the query from step 2?
  4. Run the SPARQL query – Check the actual results against the expected ones ★
  5. Store query and results as annotations of the “test case”

★ At these points you will discover the problems

1st Method: Testing Through SPARQL - Typical errors/problems discovered

- Missing datatype properties
  - Example: Person class missing the string datatype for storing a person's name
- Missing properties in general
- Property exists by misses domain/range or other restriction
  - Understandability of the model
  - Example: "has" property, without domain and range could apply to almost anything…
- Missing n-ary relations
  - Example: CQ indicates an "employment", i.e. n-ary between person, company, and duration – model contains direct relations between person-company and person-duration
- Missing inverse properties
  - Ease of use for query writing - optimization

2nd Method: Testing Inferences

- Given a reasoning requirement
  – Can the ontology perform this inference?
- Best practice: One requirement – One test case
- Proposed method:
  1. Create a new “test case” ontology and import the ontology to be tested
  2. a)
     Add test data to the “test case” ontology that should trigger the inference – What are the expected results?
     b)
     Add typical test data for the complete model – What are the expected results?
  3. Run the reasoner – Check the actual results against the expected ones, also note any unexpected side-effects (particularly from b) ★
  4. Store results as annotations of the “test case”

★ At this point you will discover the problems

2nd Method: Testing Inferences - Typical errors/problems discovered

- Expected facts are not inferred
  - Due to misinterpreting OWL semantics, e.g. mixing up equivalent and subclass statements
- Unexpected side-effects indicate other problems
  - Example: Confusing “is a” with “part of”, e.g. saying that City is a subclass of Country – an inferred statement says that our city instance “Stockholm” is of type Country
  - Example: Errors in domain and range
    - The domain of the isLocatedIn property is set to City, but we use it for all things that are located somewhere, i.e. our “Småland” instance that isLocatedIn “Sweden” is inferred to be a City.
    - The domain of the isLocatedIn property is set to the intersection of City and Region, instead of the union, i.e. our “Småland” instance that isLocatedIn “Sweden” is inferred to be both a City and a Region.

3rd Method: “Stress testing”

- Given a contextual statement, reasoning requirement or other requirement
  – What errors should the ontology be able to detect?
  – What side-effects do we want to avoid?
- Best practice: One requirement – One test case
- Proposed method:
  1. Create a new “test case” ontology and import the ontology to be tested
  2. Study the “boundaries” of data – are there any “end-conditions”? Are there cases where incorrect or nonsense data could be input by a user?
  3. Add test data to the “test case” ontology that should trigger an inconsistency or might provoke an undesired side-effect
  4. Run the reasoner – Check the actual results against the expected ones, also assess any unexpected side-effects ★
  5. Store results as annotations of the “test case”

★ At this points you will discover the problems

3rd Method: “Stress testing” - Typical errors/problems discovered

- Expected inconsistency does not appear
  - Due to missing disjointness?
  - Due to misinterpreting OWL semantics, e.g. due to the OWA or assuming an UNA
    - Some “error checking” may need to be added in the software if these things are essential
- Unexpected inferences
  - Things are inferred to be the same although they are not – e.g. due to missing UNA and cardinality restrictions

Exercise – Testing an Ontology

- Given an ontology with some problems
  - Try to find the problems and analyse the ontology using the methods described
  - Document your testing, and the mistakes you find

- http://ontologydesignpatterns.org/wiki/Training:Advanced_Ontology_Engineering_at_FOI_-_2011/Ontology_Testing