RDF/S and SPARQL Expressiveness in Engineering Design Patterns

Hacène Cherfi - INRIA Sophia Antipolis
Olivier Corby - INRIA Sophia Antipolis
Cyril Masia-Tissot - Semantic-Systems S.A.

Extended from Web Intelligence Conference (WI’07) short paper

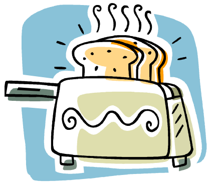
Outline

- Context and relevance
- RDF/S and SPARQL features with `Corese`
- Expressiveness needs
  - Order specification
  - Quantity/unit expression
  - Metadata description
- Conclusion and future work

Context

- SW is about "integration/combination of data from diverse sources, whereas original Web concentrated on interchange of documents"
- Based on W3C standards: RDF/S and OWL
- RDF/S lightweight ontology/data definition
  - Simple, readable, understandable
  - Primarily intended for machine consumption
  - Has expressiveness limitations
- OWL ontology definition
  - Description logics (DL)-reasoning-oriented
  - Needs inference engine

Topic relevant?…I would say:

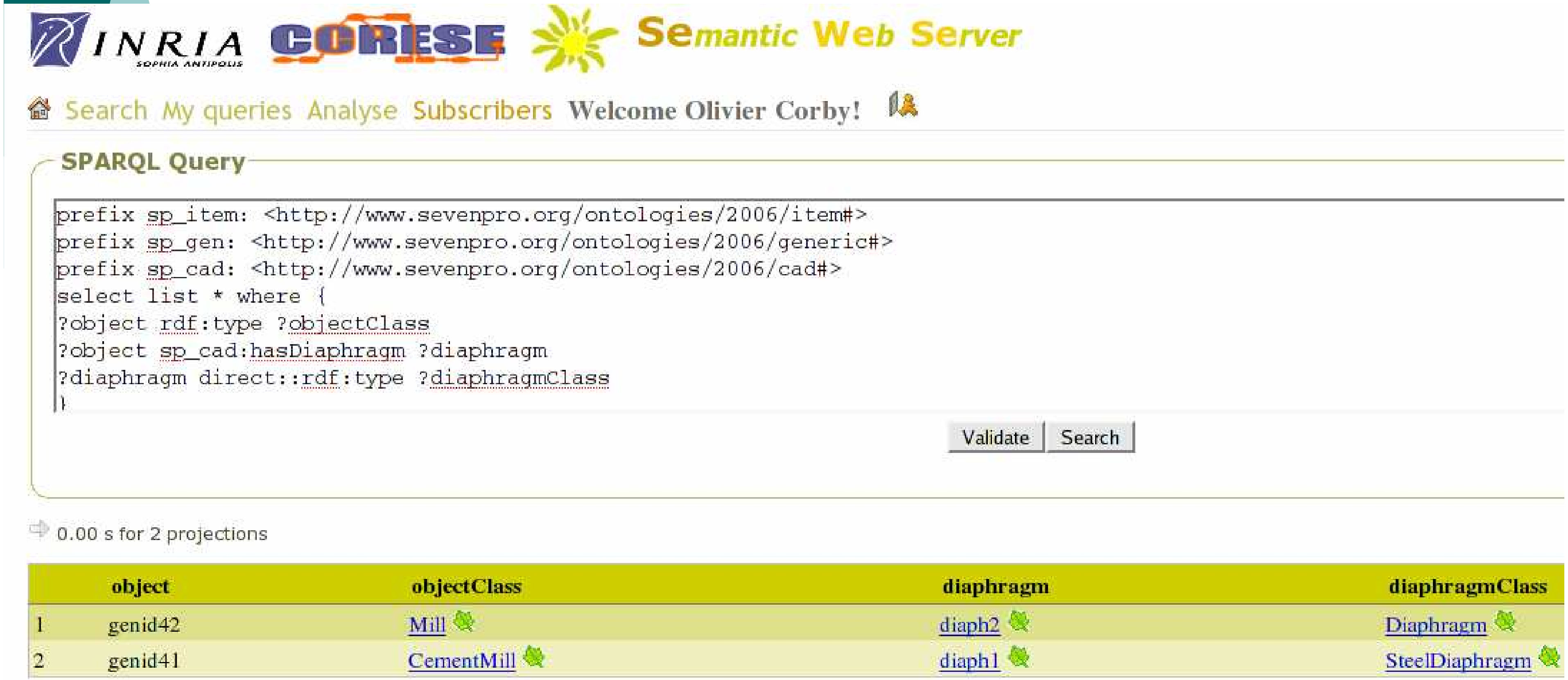


- ○ RDF/S data structure is directed labeled graph
- ○ Textual serialization (XML/ N3 triple)
  - Tractable as unit of information
  - Loses readability
  - Linkability between RDF triples in large graphs

Our purpose

- ○ Put additional information
  - On property values
  - Over selected instances
- ○ Without defining
  - Ontological property (holding for all instances)
  - Ad-hoc property (overcharging these instances)

SPARQL features (e.g. sub-classification)



```
prefix sp_item: <http://www.sevenpro.org/ontologies/2006/item#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
prefix sp_cad: <http://www.sevenpro.org/ontologies/2006/cad#>
select list * where {
?object rdf:type ?objectClass
?object sp_cad:hasDiaphragm ?diaphragm
?diaphragm direct::rdf:type ?diaphragmClass
}
```

0.00 s for 2 projections

	object	objectClass	diaphragm	diaphragmClass
1	genid42	Mill	diaph2	Diaphragm
2	genid41	CementMill	diaph1	SteelDiaphragm

Ordering instance properties

- Use case: for engineering element (instance), specify sequence of operations (properties) performed on it

- Tentative #1: Property reification with `rdf:Statement`

```
<rdf:Description rdf:ID="partBody_10">
<sp_cad:hasFeature rdf:ID="s111"
rdf:resource="#featExtrude_12"/>
</rdf:Description>
```

```
<rdf:Statement rdf:about="#s111">
<rdf:subject rdf:resource="#partBody_10"/>
<rdf:predicate
rdf:resource="&sp_cad;hasFeature"/>
<rdf:object
rdf:resource="#featExtrude_12"/>
</rdf:Statement>
```

Ordering instance properties

- Tentative #1: Property reification with `rdf:Statement`
  - (++) Add as many information as necessary

```
<rdf:Statement rdf:about="#s111">
<rdf:subject rdf:resource="#partBody_10"/>
<rdf:predicate rdf:resource="&sp_cad;hasFeature"/>
<rdf:object rdf:resource="#featExtrude_12"/>

<sp_gen:position
rdf:datatype="&xsd;integer">1</sp_gen:position>
</rdf:Statement>
```

  - After querying
  - (--) Lose connection between genuine triple (line1) and statement on triple (lines2 to 6)

```
Subject          Predicate          Object
1 #partBody_10 sp_cad:hasFeature   #featExtrude_12
2     #s111    rdf:type            rdf:Statement
3     #s111    rdf:subject         #partBody_10
4     #s111    rdf:predicate       sp_cad:hasFeature
5     #s111    rdf:object          #featExtrude_12
6     #s111    sp_gen:position     "1"^^xsd:integer
```

Ordering instance properties

- Tentative #2: Property values with `rdf:Seq` and `rdf:List`
  - (++) Easy to query `rdfs:member` (--) Difficult to query No recursive mechanism
  - (--) Arbitrarily order (--) Many blank nodes (BN)

```
<rdf:Description
rdf:about="#partBody_10">
<sp_cad:hasFeatures>
<rdf:Seq>
<rdf:li
rdf:resource="#featExtrude_12"/>
<rdf:li rdf:resource="#featHole_15"/>
</rdf:Seq>
</sp_cad:hasFeatures>
</rdf:Description>
```

```
<rdf:Description
rdf:about="#partBody_10">
<sp_cad:hasFeatures
rdf:parseType='Collection'>
<rdf:Description
rdf:about="#featExtrude_12"/>
<rdf:Description
rdf:about="#featHole_15"/>
</sp_cad:hasFeatures>
</rdf:Description>
```

Ordering user-defined property values

- Tentative #3: using Container `rdf:Seq`
  - In ontological level

```
<rdf:Property rdf:ID="hasPropertySequence"/>
<rdfs:domain rdf:resource="&rdfs;Resource"/>
<rdfs:range rdf:resource="&rdf;Seq"/>
</rdf:Property>
```

  - In instance definition

```
<rdf:Description rdf:about="#partBody_10">
<sp_cad:hasFeature rdf:about="#featExtrude_12" />
<sp_cad:hasFeature rdf:about="#featHole_15" />
<sp_cad: hasPropertySequence>
<rdf:Seq>
<rdf:li rdf:resource="#featHole_15" />
<rdf:li rdf:resource="#featExtrude_12" />
</rdf:Seq>
</sp_cad: hasPropertySequence>
</rdf:Description>
```

  - (++) Easy to query with SPARQL (see in ex16)
  - (--) Property and order definition in different levels (see ambiguity in ex17)

Possible solution: our proposition (1/2)

- Property with explicit order
  - Define in ontology

```
<rdf:Property rdf:ID="order">
<rdfs:domain rdf:resource="&rdfs;Resource"/>
<rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
```

  - Use as instance specification with `parseType=“Resource”` and `rdf:value`

```
<rdf:Description rdf:about="#partBody_10">
<sp_cad:hasFeature rdf:parseType="Resource">
<rdf:value rdf:resource="#featHole_15" />
<sp_gen:order rdf:datatype=’&xsd;integer’>1</sp_gen:order>
</sp_cad:hasFeature>
<sp_cad:hasFeature rdf:parseType="Resource">
<rdf:value rdf:resource="#featExtrude_12" />
<sp_gen:order rdf:datatype=’&xsd;integer’>2</sp_gen:order>
</sp_cad:hasFeature>
</rdf:Description>
```

Possible solution: our proposition (2/2)

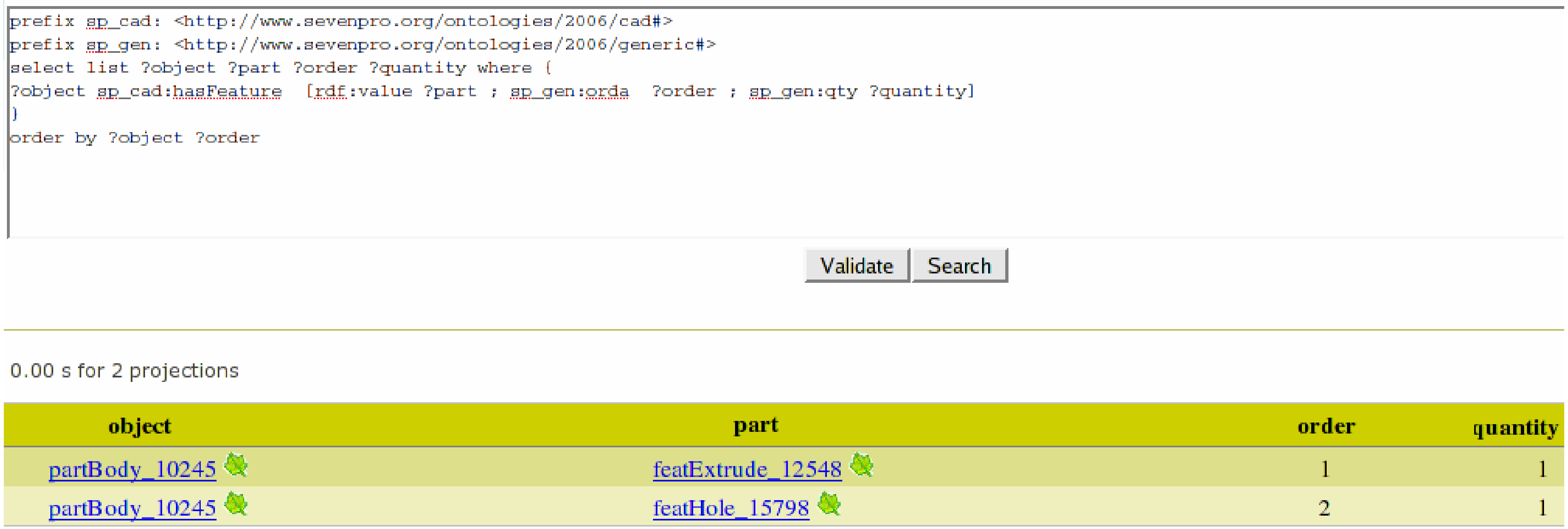
- Property with explicit order
  - BN is created

```
#partBody_10 sp_cad:hasFeature _:bn1
_:bn1 rdf:value #featHole_15
_:bn1 sp_gen:order "1"^^xsd:integer
#partBody_10 sp_cad:hasFeature _:bn2
_:bn2 rdf:value #featExtrude_12
_:bn2 sp_gen:order "2"^^xsd:integer
```

  - Query with operator [ ]matching BNs

```
select ?object ?part ?ordering where {
?object sp_cad:hasFeature
[rdf:value ?part ; sp_gen:order ?ordering]
}
order by ?object ?ordering
```

Order (1/2)

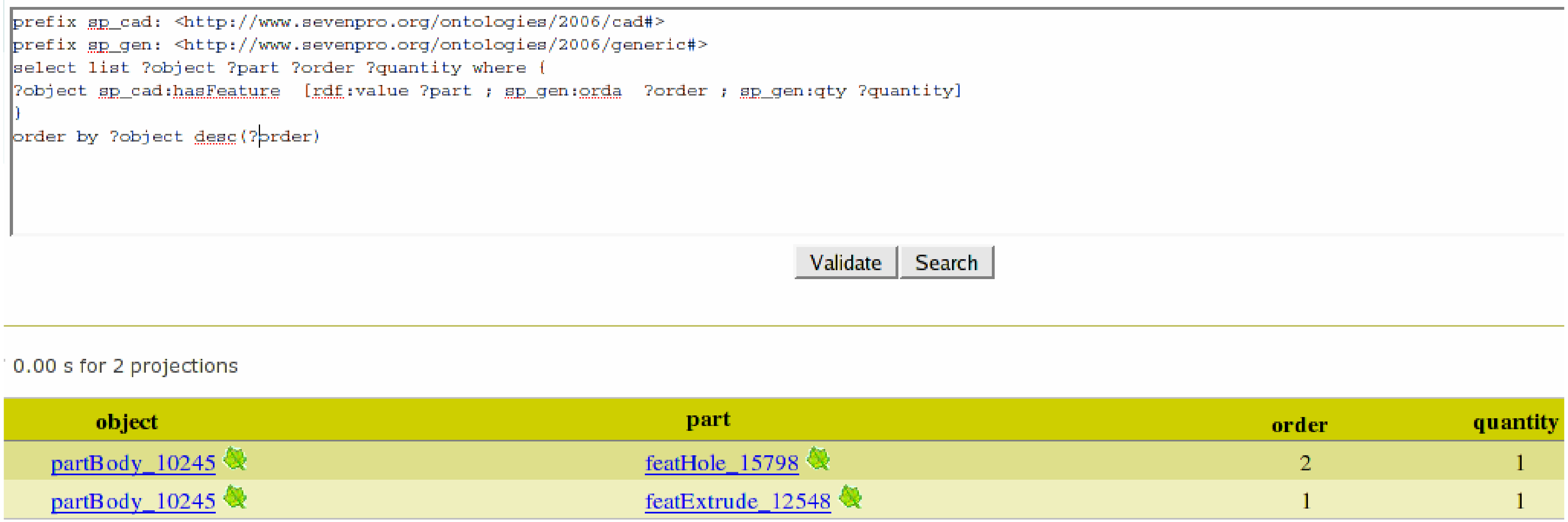


```
prefix sp_cad: <http://www.sevenpro.org/ontologies/2006/cad#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select list ?object ?part ?order ?quantity where {
?object sp_cad:hasFeature  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity]
}
order by ?object ?order
```

0.00 s for 2 projections

object	part	order	quantity
partBody_10245	featExtrude_12548	1	1
partBody_10245	featHole_15798	2	1

Order (2/2)



```
prefix sp_cad: <http://www.sevenpro.org/ontologies/2006/cad#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select list ?object ?part ?order ?quantity where {
?object sp_cad:hasFeature  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity]
}
order by ?object desc(?order)
```

0.00 s for 2 projections

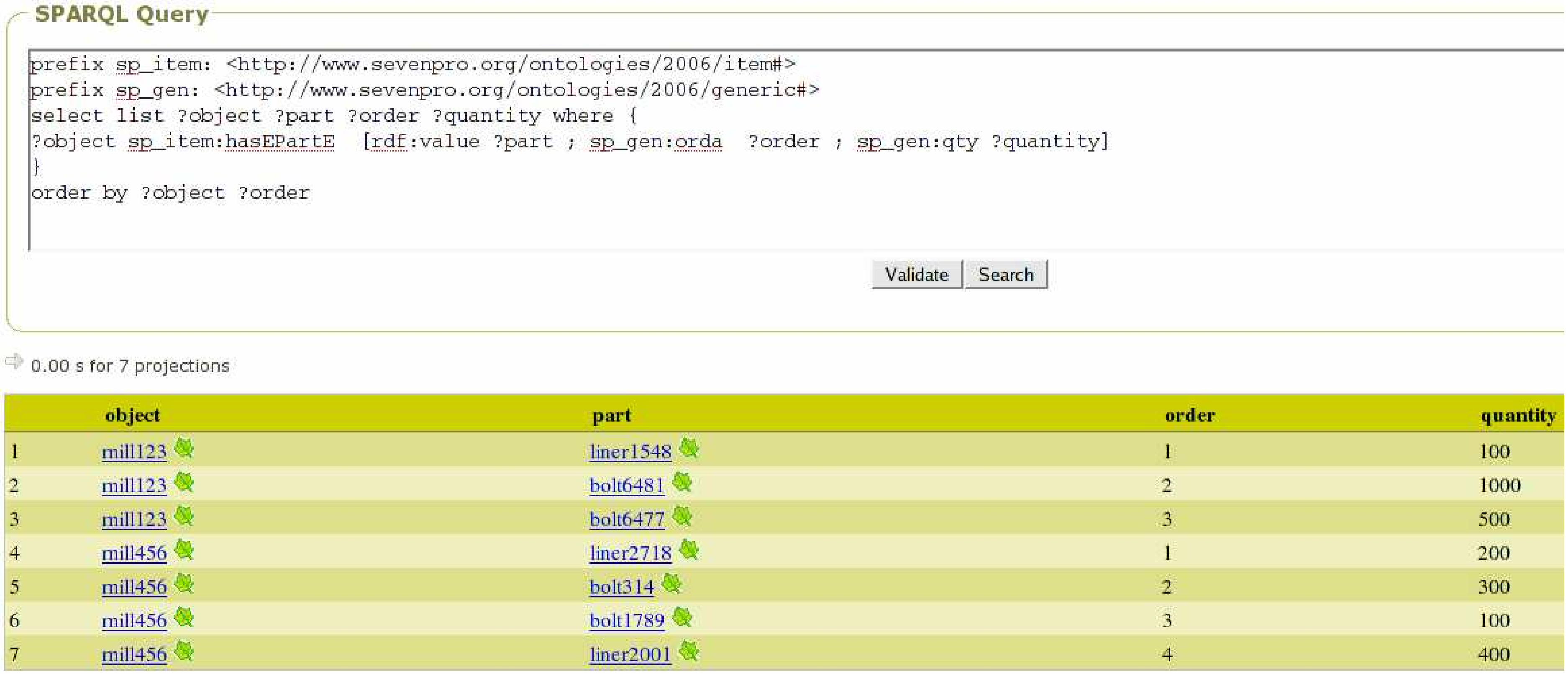
object	part	order	quantity
partBody_10245	featHole_15798	2	1
partBody_10245	featExtrude_12548	1	1

Quantity, unit, additional information

○ Use cases:

1. Quantity: specify how much/many (n) of something
   - Without create (n) instance properties
2. Unit: specify
   - Size, speed of object (metric/us scale)
   - Weight system (international/us scale)
   - Temperature (°C/°F)
   - etc.

Order on statements and quantity (1/2)



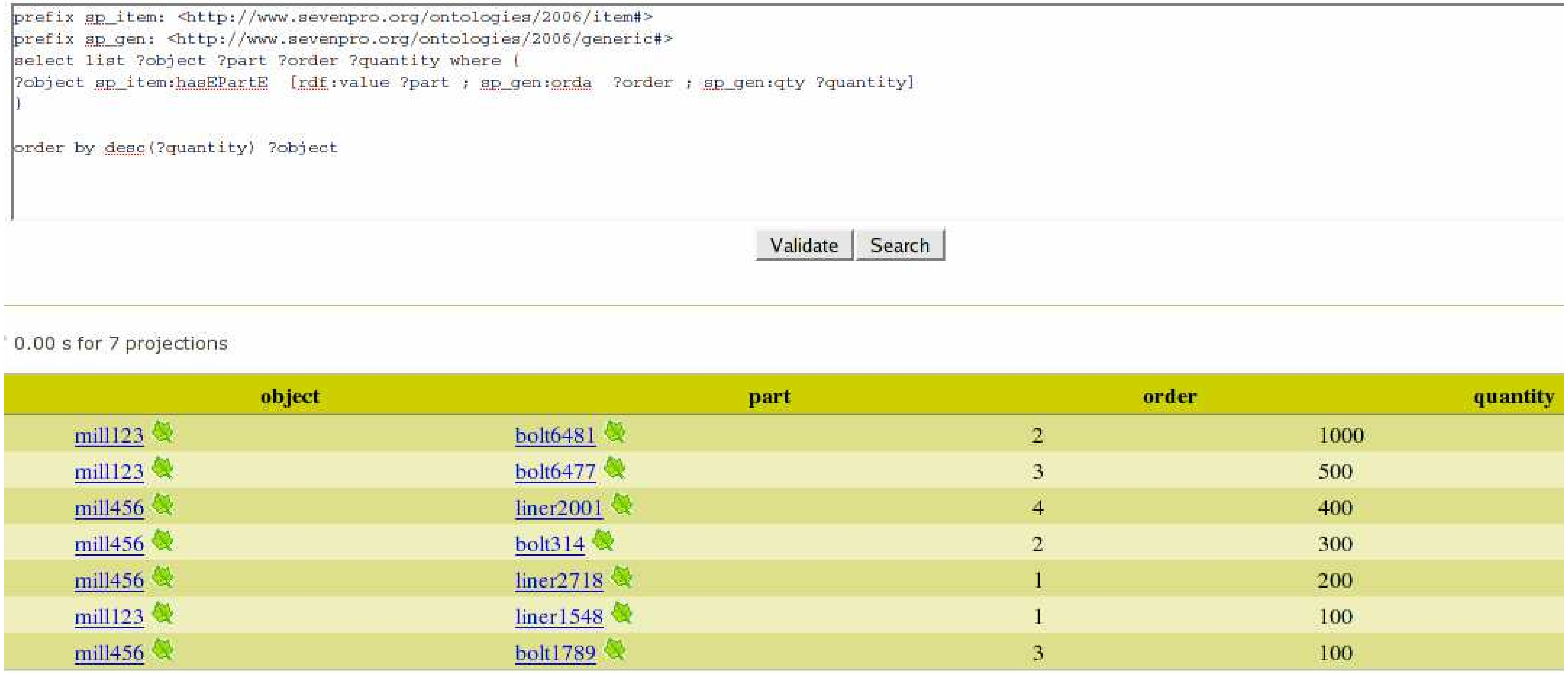
SPARQL Query

```
prefix sp_item: <http://www.sevenpro.org/ontologies/2006/item#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select list ?object ?part ?order ?quantity where {
?object sp_item:hasEPartE  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity]
}
order by ?object ?order
```

0.00 s for 7 projections

	object	part	order	quantity
1	mill123	liner1548	1	100
2	mill123	bolt6481	2	1000
3	mill123	bolt6477	3	500
4	mill456	liner2718	1	200
5	mill456	bolt314	2	300
6	mill456	bolt1789	3	100
7	mill456	liner2001	4	400

Order on statements and quantity (2/2)



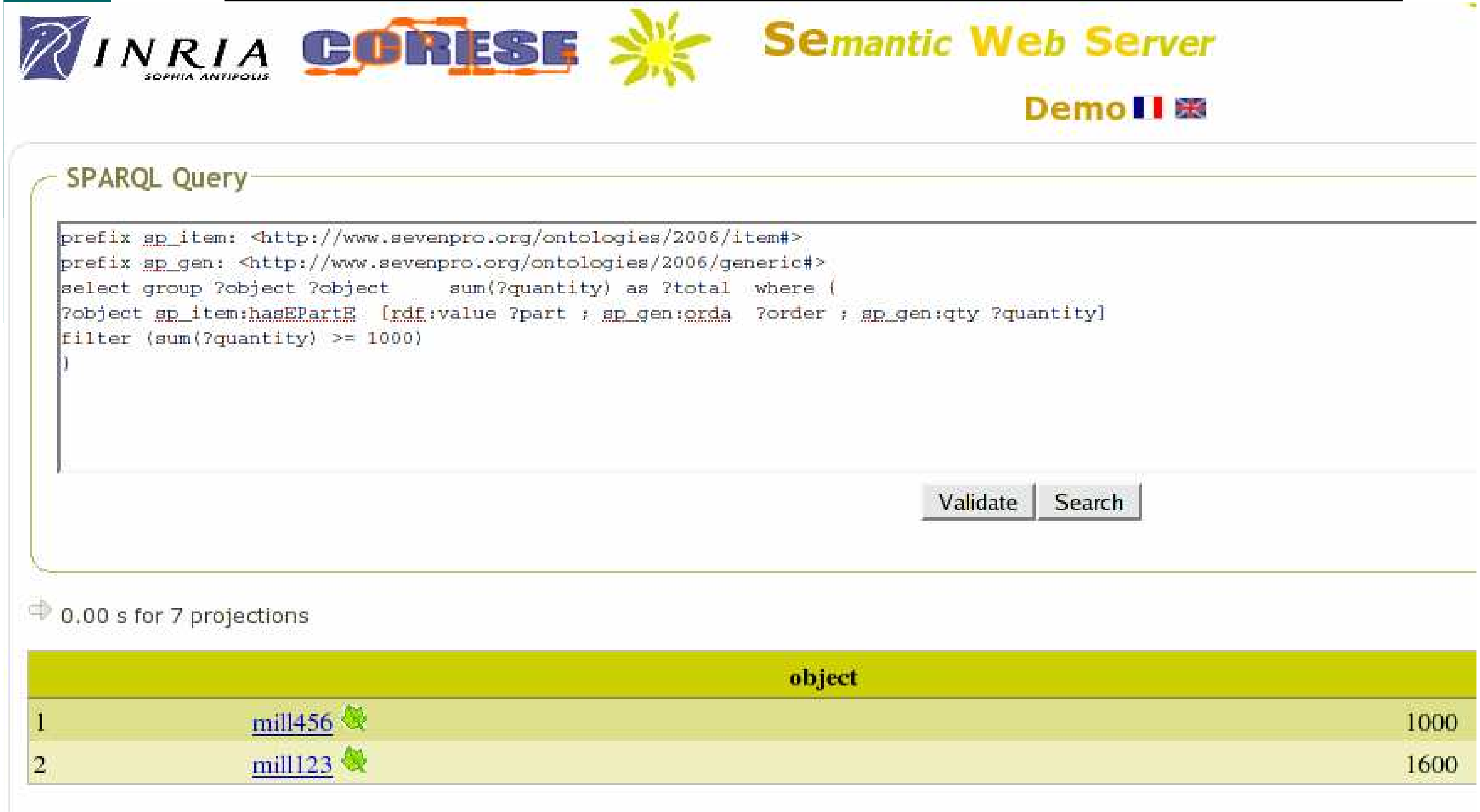
```
prefix sp_item: <http://www.sevenpro.org/ontologies/2006/item#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select list ?object ?part ?order ?quantity where {
?object sp_item:hasEPartE  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity]
}

order by desc(?quantity) ?object
```

0.00 s for 7 projections

object	part	order	quantity
mill123	bolt6481	2	1000
mill123	bolt6477	3	500
mill456	liner2001	4	400
mill456	bolt314	2	300
mill456	liner2718	1	200
mill123	liner1548	1	100
mill456	bolt1789	3	100

Query combination capabilities
E.g: sum on quantity (1/2)



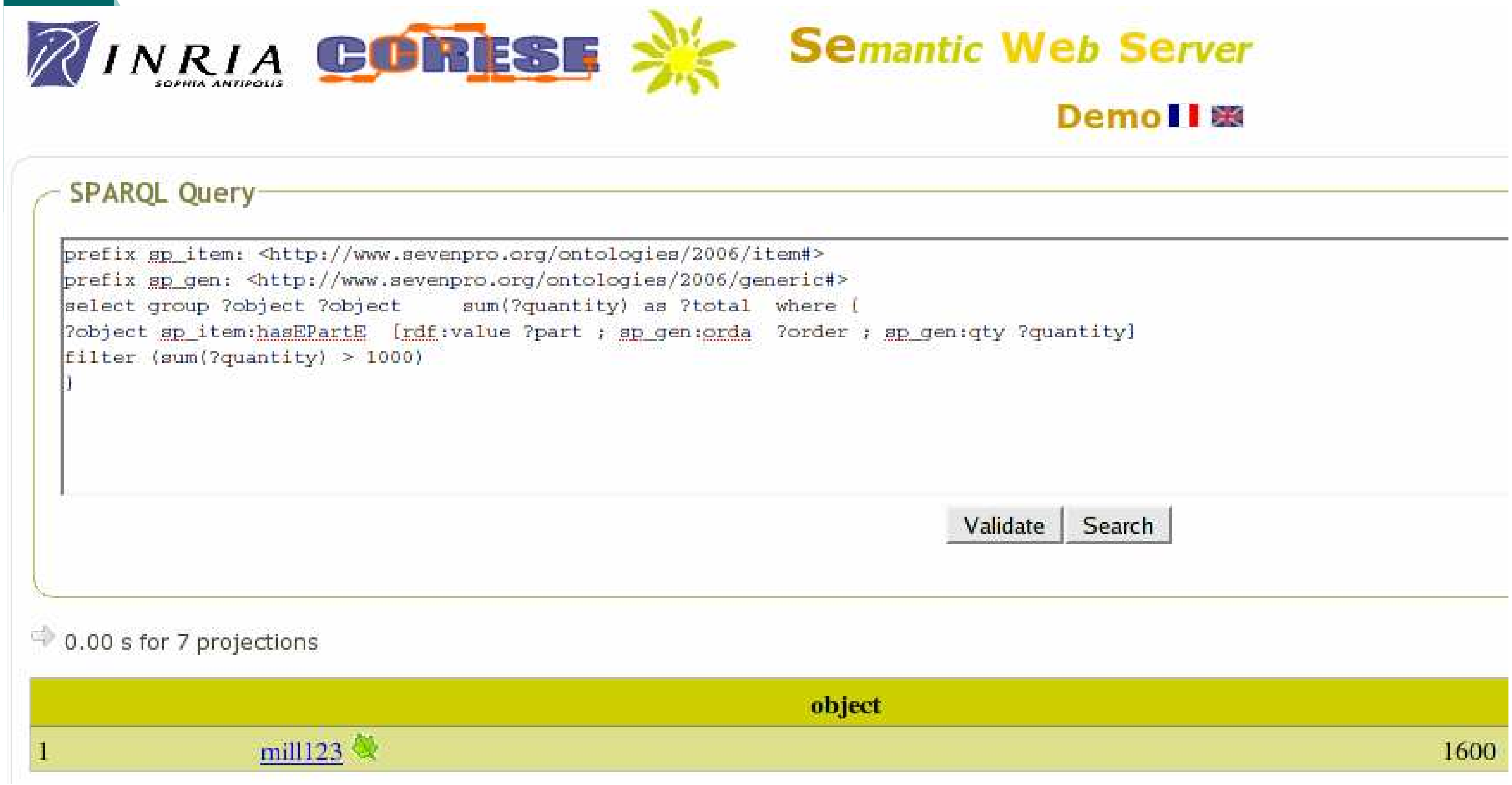
SPARQL Query

```
prefix sp_item: <http://www.sevenpro.org/ontologies/2006/item#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select group ?object ?object     sum(?quantity) as ?total  where {
?object sp_item:hasEPartE  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity]
filter (sum(?quantity) >= 1000)
}
```

0.00 s for 7 projections

	object	
1	mill456	1000
2	mill123	1600

Query combination capabilities
E.g: sum on quantity (2/2)



SPARQL Query

```
prefix sp_item: <http://www.sevenpro.org/ontologies/2006/item#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select group ?object ?object     sum(?quantity) as ?total  where {
?object sp_item:hasEPartE  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity]
filter (sum(?quantity) > 1000)
}
```

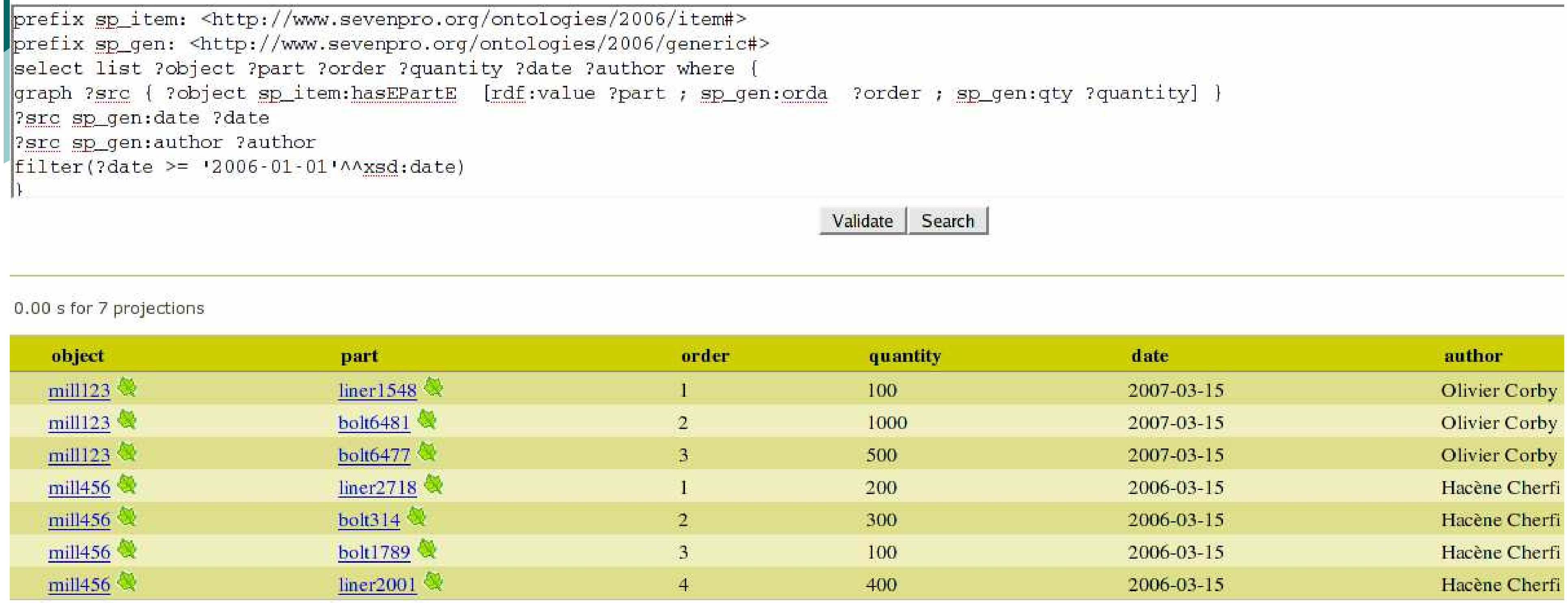
0.00 s for 7 projections

	object	
1	mill123	1600

Annotation on metadata

- Use cases: annotate origin of information
  - In RDF triple:
    - E.g: author, version number, date
  - Using SPARQL `graph` source capability

E.g.: using graph source (1/2)

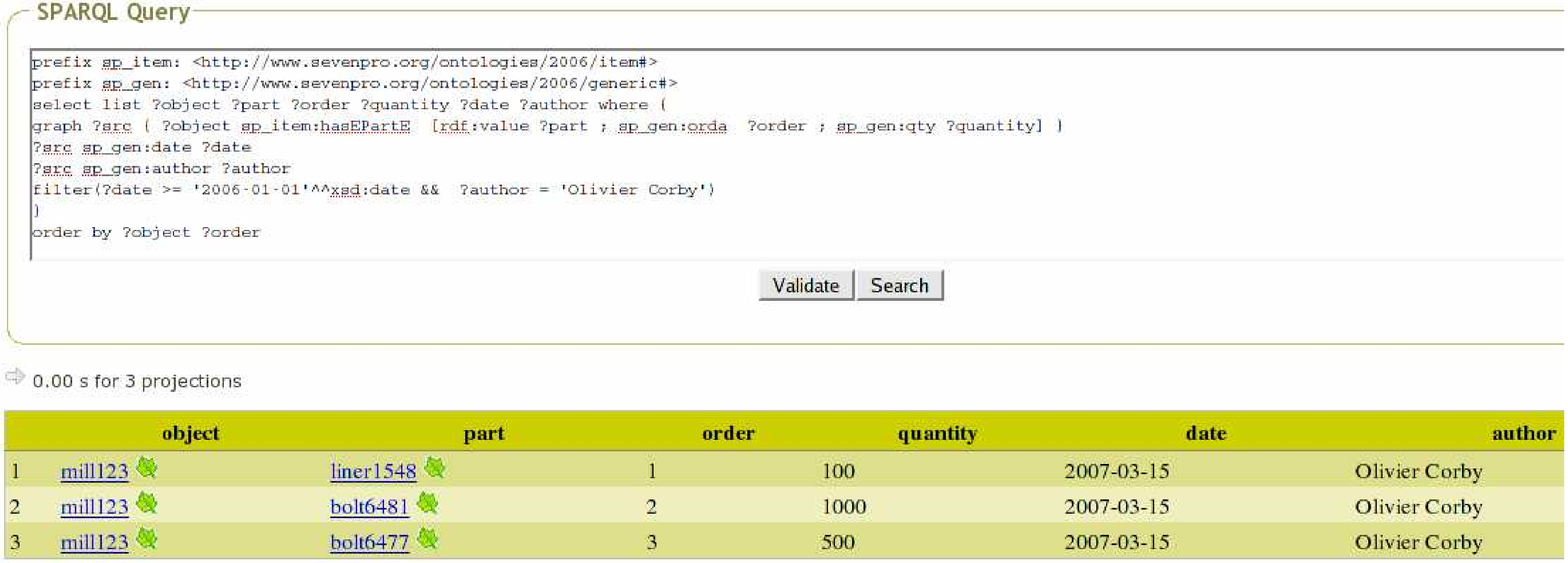


```
prefix sp_item: <http://www.sevenpro.org/ontologies/2006/item#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select list ?object ?part ?order ?quantity ?date ?author where {
graph ?src { ?object sp_item:hasEPartE  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity] }
?src sp_gen:date ?date
?src sp_gen:author ?author
filter(?date >= '2006-01-01'^^xsd:date)
}
```

0.00 s for 7 projections

object	part	order	quantity	date	author
mill123	liner1548	1	100	2007-03-15	Olivier Corby
mill123	bolt6481	2	1000	2007-03-15	Olivier Corby
mill123	bolt6477	3	500	2007-03-15	Olivier Corby
mill456	liner2718	1	200	2006-03-15	Hacène Cherfi
mill456	bolt314	2	300	2006-03-15	Hacène Cherfi
mill456	bolt1789	3	100	2006-03-15	Hacène Cherfi
mill456	liner2001	4	400	2006-03-15	Hacène Cherfi

E.g.: using graph source (2/2)



```
prefix sp_item: <http://www.sevenpro.org/ontologies/2006/item#>
prefix sp_gen: <http://www.sevenpro.org/ontologies/2006/generic#>
select list ?object ?part ?order ?quantity ?date ?author where {
graph ?src { ?object sp_item:hasEPartE  [rdf:value ?part ; sp_gen:orda  ?order ; sp_gen:qty ?quantity] }
?src sp_gen:date ?date
?src sp_gen:author ?author
filter(?date >= '2006-01-01'^^xsd:date &&  ?author = 'Olivier Corby')
}
order by ?object ?order
```

0.00 s for 3 projections

	object	part	order	quantity	date	author
1	mill123	liner1548	1	100	2007-03-15	Olivier Corby
2	mill123	bolt6481	2	1000	2007-03-15	Olivier Corby
3	mill123	bolt6477	3	500	2007-03-15	Olivier Corby

Conclusion and future work

- Express new features:
  1. N-ary relations (order, unit, …)
     - With standard RDF `parseType="Resource"`, and `value`
     - With standard SPARQL BN matching `[]`
  2. Annotation on metadata
     - With standard SPARQL `graph` source
- In Engineering domain and wider…

- Perspectives
  - Evaluate BN usage impact of our proposition in (1) on massive RDF graphs
  - Extend (2) to *context types* in engineering (design, specification, …) and relate contexts to corresponding ontologies