Querying OWL Ontologies

Steffen Staab
with
Claudia d’Amato, Alexander Kubias, Simon Schenk, & Jeff Pan
Motivation Scenario

Verizon (Source M. Brodie, WWW-2006):

- Status quo @ 2006:
  500 (dot.net) services for in-house and external customers
- Coverage: estimated 1% of business
- My cautious extrapolation: 10,000 services in 2016

→ Service governance a real problem
  - How to engineer them?
  - How to retrieve them in interactive software development mode?
  - How to understand them?
Agenda

- How to engineer them?
  - Methodology by Grimm, Motik, Preist
- How to retrieve them in interactive software development mode?
  - Efficient OWL Querying using DL-trees
- How to understand them?
  - Expressive OWL Querying using SAIQL
Agenda

- How to engineer them?
  - Methodology by Grimm, Motik, Preist

- How to retrieve them in interactive software development mode?
  - Efficient OWL Querying using DL-trees

- How to understand them?
  - Expressive OWL Querying using SAIQL
How to engineer them?

Method by Grimm, Motik, Preist:

1. Service described as concept expression

2. Service executions are represented by instantiations

3. Background knowledge described in ontology

4. Query described as concept expression

5. Retrieval described as DL proof

Running Example

- CreditCardPayment = Service ⊑
  ∀require.CreditCardNr ⊑
  ∃delivers.MoneyTransfer

- require(myTicketPay,myCCnr),
  delivers(myTicketPay,myMoTr)

- MasterCardNr ▽ CreditCardNr

- Query=Service ⊑
  ∃=1 require.MasterCardNr ⊑
  ∃=1 require.Top ⊑
  ∃=1 delivers.MoneyTransfer

- CreditCardPayment ▽ Query
Agenda

- How to engineer them?
  - Methodology by Grimm, Motik, Preist

- How to retrieve them in interactive software development mode?
  - Efficient OWL Querying using DL-Trees

- How to understand them?
  - Expressive OWL Querying using SAIQL
Resource Retrieval is performed
  - by matching a given request R with every available resource descriptions

Approach inefficient with growing number of services

Improve the retrieval efficiency by the use of a tree-index
Overall Idea

Currently

Available Resources

Request

Match Test

O(n)

The Idea

Match Test

Available Resources

O(log n)
Hierarchical agglomerative clustering

Classical Setting

- Applied to feature vectors representations
- Similarity measured as inverse of geometrical distance
Hierarchical agglomerative clustering

Classical Setting
- Applied to feature vectors representations
- Similarity measured as inverse of geometrical distance
Hierarchical agglomerative clustering

Classical Setting

- Applied to feature vectors representations
- Similarity measured as inverse of geometrical distance
Hierarchical agglomerative clustering

Classical Setting

- Applied to feature vectors representations
- Similarity measured as inverse of geometrical distance
Hierarchical agglomerative clustering

Classical Setting
- Applied to feature vectors representations
- Similarity measured as inverse of geometrical distance
Hierarchical agglomerative clustering

Classical Setting
- Applied to feature vectors representations
- Similarity measured as inverse of geometrical distance
Hierarchical agglomerative clustering

Classical Setting

- Applied to feature vectors representations
- Similarity measured as inverse of geometrical distance
Open issues

Set up agglomerative clustering for Description Logics representations

Issues:
1. Which clusters to merge?
   - A similarity measure applicable to complex DL concepts is required

2. How to re-represent merged clusters?
   - A conceptual clustering method is needed for producing intensional cluster descriptions
   - A “good” averaging procedure
Here some intuitive explanation of the GCS ins missing
The Semantic Similarity Measure

**Rationale:** Two concepts are more similar as much their extensions are similar.

\[ \Delta I \]

\[ \text{C} \equiv \text{credit-card-payment} \]
\[ \text{D} \equiv \text{debit-card-payment} \]
\[ \text{GCS}(C,D) \equiv \text{card-payment} \]

\[ \text{E} \equiv \text{car-transfer} \]
\[ \text{D} \equiv \text{debit-card-payment} \]
\[ \text{GCS}(E,D) \equiv \text{service} \]
**Measure Definition**

**GCS-Based Similarity Measure:** Let $T$ be an ALC T-Box and let $C$ and $D$ be two $ALE(T)$-concept descriptions. The semantic similarity measure $s: ALE(T) \times ALE(T) \rightarrow [0,1]$ is defined as:

$$s(C, D) = \frac{\min(|C^I|, |D^I|)}{|GCS(C, D)^I|} \cdot \left(1 - \frac{|GCS(C, D)^I|}{|\Delta^I|}\right) \cdot \left(1 - \frac{\min(|C^I|, |D^I|)}{|GCS(C, D)^I|}\right)$$

$$\frac{1}{2} \cdot \left(1 - \frac{2}{5} \cdot \left(1 - \frac{1}{2}\right)\right) = \frac{1}{2} \cdot \left(1 - \frac{1}{5}\right) = \frac{2}{5}$$

$$\frac{1}{3} \cdot \left(1 - \frac{3}{5} \cdot \left(1 - \frac{1}{3}\right)\right) = \frac{1}{3} \cdot \left(1 - \frac{2}{5}\right) = \frac{1}{5}$$
Modified *average-link* algorithm

- Merge based on GCS-based similarity measures (instead of Euclidean distance)
Modified *average-link* algorithm

- Merge based on GCS-based similarity measures (instead of Euclidean distance)
- Intentional cluster descriptions generated by means of the GCS of the clusters to merge (Instead of Euclidean average)
DL-Link Algorithm

Modified *average-link* algorithm

- Merge based on GCS-based similarity measures (instead of Euclidean distance)
- Intentional cluster descriptions generated by means of the GCS of the clusters to merge (Instead of Euclidean average)
- Flattening Post-processing
Retrieval Procedure

\[ R = \text{Flight} \sqcap \exists \text{from.Cologne} \sqcap \exists \text{to.Bari} \]

\[ R \sqsubseteq C_i \quad ? \]

\[ C = \text{Flight} \sqcup \text{Hotel} \]

\[ C_{11} = \text{Flight} \]

\[ C_{11} = \text{Flight} \sqcap \exists \text{From.Paris} \sqcap \exists \text{to.Bari} \]

\[ C_{112} = \text{Flight} \sqcap \exists \text{from.Cologne} \sqcap \exists \text{to.Bari} \]

\[ C_{12} = \text{Hotel} \]

Available Resources
Data Sets for Experiments

SWS Discovery Data Set
- 93 ALE(T) service descriptions referring to
- ALC ontology (bank, post, means of communication, geographical information)
- developed based on another dataset in order to fit the methodology by Grimm et al

Wine ontology
- 112 concepts (primitive and defined)
- 9 object properties
- 93 distinct individuals
- from Protégé Library
- ALCIO(D): constructor “not” have been discarded in service descriptions
Evaluation Methodology 1

SWS Discovery Data set
- All service descriptions have been clustered by the use of the DL-Link algorithm and a DL-Tree has been obtained.

Generated Queries
- 96 corresponding to the leaf nodes of DL-Tree
- 20 corresponding to some inner nodes
- 116 randomly generated by conjunction /disjunction of primitive and/or defined concepts of the ontology and/or service descriptions.

Wine Ontology:
- For the 93 distinct individual names, their MSCs have been computed and then clustered obtaining the DL-Tree.

Generated Queries
- 93 corresponding to the leaf nodes of DL-Tree
- 116 randomly generated by conjunction /disjunction of primitive and/or defined concepts in the ontology.
Evaluation Methodology 2

Efficiency of the DL-Tree based method measured by

- Average number of matches in the DL-Tree for finding all resources satisfying the query
- Mean execution time per each query
  - Laptop PowerBook G4 1.67 GHz 1.5 GB RAM

Compared with Linear Matching approach

- Number of matches
- Mean execution time per each query, which retrieved all resources for this query
### Evaluation Results

#### SWS Discovery Data Set

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Metrics</th>
<th>Leaf Node</th>
<th>Inner Node</th>
<th>Random Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL-Tree based</td>
<td>avg.</td>
<td>41.4</td>
<td>23.8</td>
<td>40.3</td>
</tr>
<tr>
<td></td>
<td>range</td>
<td>13 - 56</td>
<td>19 - 27</td>
<td>19 - 79</td>
</tr>
<tr>
<td></td>
<td>avg. exc. time</td>
<td>266.4 ms.</td>
<td>180.2 ms.</td>
<td>483.5 ms.</td>
</tr>
<tr>
<td>Linear</td>
<td>avg.</td>
<td>96</td>
<td>96</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>avg. exc. time</td>
<td>678.2 ms.</td>
<td>532.5 ms.</td>
<td>1589.3 ms.</td>
</tr>
</tbody>
</table>

#### Wine Ontology

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Metrics</th>
<th>Leaf Node</th>
<th>Random Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL-Tree based</td>
<td>avg.</td>
<td>34.3</td>
<td>16.1</td>
</tr>
<tr>
<td></td>
<td>range</td>
<td>12 - 43</td>
<td>9 - 31</td>
</tr>
<tr>
<td></td>
<td>avg. exc. time</td>
<td>285.4 ms.</td>
<td>155.2 ms.</td>
</tr>
<tr>
<td>Linear</td>
<td>avg.</td>
<td>93</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>avg. exc. time</td>
<td>1263.1 ms.</td>
<td>1096.3 ms.</td>
</tr>
</tbody>
</table>
Conclusions about DL-Tree

DL-Tree is a sound and complete indexing method for improving the efficiency of the resource retrieval task.

Experimental results show improvement at the small scale for more than 50% - larger improvements for larger scale to be expected.

Based on the exploitation of
- A new Semantic Similarity Measure, GCS-based Similarity
- A new conceptual clustering algorithm, DL-link
- The GCS as generalization procedure
Open issues / Next steps

Efficiency of similarity computation
- GCS computation is very frequent and very expensive
- Computation of extension of GCS is too expensive even in the batch process

Other DL languages
- For the DL-tree index, e.g. DL-lite instead of ALE
- For the retrieval process, e.g. OWL DL instead of ALC

DL-tree for Other Matches
- \(\text{CreditCardPayment} = \text{Service} \sqcap \exists_1 \text{require.CreditCardNr} \sqcap \exists_1 \text{require.Top} \sqcap \exists_1 \text{delivers.MoneyTransfer}\)
- \(\text{require(myTicketPay,myCCnr)}, \text{delivers(myTicketPay,myMoTr)}\)
- \(\text{CreditCardNr} \sqsubseteq \text{DigitalData}\)
- \(\text{Query=Service} \sqcap \forall \text{require.DigitalData} \sqcap \exists \text{delivers.MoneyTransfer}\)
- \textbf{Doesnot work: CreditCardPayment} \sqsubseteq \exists \text{Query}
Agenda

- How to engineer them?
  - Methodology by Grimm, Motik, Preist
- How to retrieve them in interactive software development mode?
  - Efficient OWL Querying using DL-trees
- How to understand them?
  - Expressive OWL Querying using SAIQL
Example Use Case 1

Situation:

- **Given:** Large OWL DL ontology with many descriptions of services, including payment services, telecom services and others
- **Task:** Build a new information system using payment services and their corresponding ontology
- **Solution:** Extract and re-use an appropriate part of the ontology
- **Problems:**
  - The ontology is very large and changes very regularly, because new service descriptions appear and old ones are changed or disappear
  - The extraction cannot be done manually
Example Use Case

Situation:

- **Given:** Large OWL DL ontology about cars and motorbikes
- **Task:** Build a new information system about cars
- **Solution:** Extract and re-use an appropriate part of the ontology
- **Problems:**
  - The ontology is very large and changes very often
  - The extraction cannot be done manually
Example Use Case

EquivalentClasses(MotorBike restriction(hasWheel cardinality(2)))
EquivalentClasses(Car restriction(hasWheel cardinality(4)))

Class(Convertible partial Car restriction(hasConvertibleTop someValuesFrom(owl:Thing)))
Class(Van partial restriction(hasWheel cardinality(4)) restriction(hasSlidingDoor someValuesFrom(owl:Thing)))

DisjointClasses(Convertible Van)

Individual(c type(Convertible))
Individual(v type(Van))
Individual(m type(MotorBike))
Example Use Case

We need:

- Schema and instance information about cars
- A fully working OWL DL ontology

Query in natural language:

- „Retrieve all concept names, which are subconcepts of the restriction $\geq 4$ hasWheel, their descriptions and their instances!“
Shortcomings of Current Query Languages

RDF Query Languages: SPARQL or RQL
- Only match triple patterns
- Not aware of OWL semantics
- Very cumbersome to query OWL DL ontologies

In our example use case:
- SPARQL would not retrieve the concept Convertible
- If SPARQL is combined with reasoner, there remain some ambiguous serializations
Shortcomings of Current Query Languages

OWL Query Language: **OWL-QL**
- Restricted access to T-Box
- Only named concepts and instances can be retrieved
- Isolated concepts and instances are returned

In our example use case:
- OWL-QL could not return concept descriptions
- OWL-QL does not return axioms.
  Thus, we do not receive a fully working OWL DL ontology.
Requirements for OWL SAIQL

Major requirements
1. Consider the semantics of OWL DL
   - But: Ensure finite answers to queries
2. Handle concept names, instances, but also concept descriptions and property names
3. Return OWL DL axioms

Minor Requirements
1. Allow join-like conditions on selected concepts and instances
2. Use syntax that is similar to OWL abstract syntax
SAIQL Syntax

CONSTRUCT
  AxiomPatterns*

FROM OntologyName

LET VariableTypings*

WHERE
  AxiomPatterns*
SAIQL Syntax

- Typed Variables \( V \) for
  - Concept names \( V_C \subseteq V \)
  - Concept descriptions \( V_{CD} \subseteq V \)
  - Property names \( N_{IP} \subseteq V \)
  - Instance names \( N_I \subseteq V \)

- Later to be bound to:
  
- \( N_C = \{ \text{Car, Convertible, MotorBike, Van} \} \)

- \( N_{IP} = \{ \text{hasWheel, hasConvertibleTop, hasSlidingDoor} \} \)

- \( N_I = \{ c, v, m \} \)

- \( N_{CD} = N_C \cup \{ \text{restriction( hasWheel cardinality}(2)\text{), restriction( hasWheel cardinality}(4)\text{), intersectionOf( Car restriction( hasConvertibleTop someValuesFrom( owl:Thing ) ))} \} \)
• Axiom patterns
  • Defined analogously to axioms, but allow variables \( V \)
  • Range of variables can be \( N_C, N_I, N_{IP} \) or \( N_{CD} \)
  • The name of a variable must start with a “?”
• Examples:
  • \( pc = \text{SubClassOf(?X ?Z)} \)
  • \( Pi = \text{Individual(?i type(?X))} \)
OWL DL Semantics

- Let $N_C$, $N_{IP}$, $N_{DP}$, $N_I$ be the sets of URI references to denote concepts, individual-valued properties, data-valued properties and individuals.

- An OWL DL interpretation is a tuple $I = (\Delta^I, \Delta^D, .^I, .^D)$ where
  - the individual domain $\Delta^I$
  - the datatype domain $\Delta^D$
  - $.^I$ is an individual interpretation function
  - $.^D$ is a datatype interpretation function

- An individual interpretation function $.^I$ is a function that maps
  - each individual name $a \in N_I$ to an element $a^I \in \Delta^I$
  - each concept name $C \in N_C$ to a subset $C^I \subseteq \Delta^I$,
  - each individual-valued property name to a binary relation $R^I \subseteq \Delta^I \times \Delta^I$
  - each data-valued property name to a binary relation $T^I \subseteq \Delta^I \times \Delta^I$. 
SAIQL Semantics

- Finite sets of
  - Concept names $N_C$
  - Concept descriptions $N_{CD}$
    - $N_{CD}$ is a finite set of concept descriptions that consists of all concept descriptions appearing in an ontology $O$
  - Property names $N_{IP}$
  - Data property names $N_{DP}$
  - Instance names $N_I$

- $N_C = \{ \text{Car, Convertible, MotorBike, Van} \}$
- $N_{IP} = \{ \text{hasWheel, hasConvertibleTop, hasSlidingDoor} \}$
- $N_{DP} = \emptyset$
- $N_I = \{ c, v, m \}$
- $N_{CD} = N_C \cup \{ \text{restriction( hasWheel cardinality(2) ), restriction( hasWheel cardinality(4) ), intersectionOf( Car restriction( hasConvertibleTop someValuesFrom( owl:Thing ) ) )}, ... \}$
Model-Theoretic Semantics for SAIQL

- Extend OWL-DL semantics for OWL-SAIQL
  - axiom patterns
  - E.g.: $pc = \text{SubClassOf}(C, D)$,
    where $C, D \in N_{CD}$: $pc^I = 1$ if $C^I \subseteq D^I$
      $0$ otherwise

- Define solution space as Cartesian product of finite sets $N_C$, $N_I$, $N_{IP}$ or $N_{CD}$ corresponding to typed Variables $V$, described as syntactic substitutions $S_{ALL}$

- For each substitution set in the solution space:
  - Ground-instantiate axiom patterns in WHERE clause
  - Solution is valid if WHERE clause is true
  - For each valid solution:
    - Ground-instantiate axiom patterns in CONSTRUCT clause
    - Add to axioms of result set
Basic Evaluation Strategy

Three steps that run one after the other:
- Evaluation of LET clause
- Evaluation of WHERE clause
- Evaluation of CONSTRUCT clause
Basic Evaluation Strategy (Example)

Evaluation of LET clause:

- Three finite sets are retrieved:
  - \( N_C = \{\text{Car, Convertible, MotorBike, Van}\} \)
  - \( N_I = \{c, m, v\} \)
  - \( N_{CD} = \{\text{Car, Convertible, MotorBike, Van, =2 hasWheel, =4 hasWheel, Car } \cap \text{ hasSlidingDoor, …}\} \)
Evaluation of LET clause:

- Set of all syntactically possible solutions is created:
  - $S_{\text{ALL}} = \{
      \{[?i/c], [?X/Car], [?Z/=2 \text{ hasWheel}]\},
      \{[?i/m], [?X/Car], [?Z/=2 \text{ hasWheel}]\},
      \{[?i/v], [?X/Car], [?Z/=2 \text{ hasWheel}]\},
      \{[?i/c], [?X/Van], [?Z/=2 \text{ hasWheel}]\},
      \ldots\}

Evaluation of WHERE clause:

- Test single solution:
  - {[?i/v], [?X/Van], [?Z/=4 hasWheel]}

- Ground instantiated axiom patterns

  Class(Van partial restriction(hasWheel cardinality(4)))
  AND Individual(v type(Van))
  AND Class(Van partial =4 hasWheel)

- Reasoner: Where pattern implied by ontology!
Evaluation of CONSTRUCT clause:

- Valid solution:
  \{[?i/v], [?X/Van], [?Z/=4 hasWheel]} \n
- Resulting axioms in the CONSTRUCT clause:
  \[
  \text{Class} (\text{Van} \text{ partial} = 4 \text{ hasWheel}); \\
  \text{Individual} (v \text{ type(Van)})
  \]
Conclusion on SAIQL

SAIQL is a new query language for OWL DL
- It can query the T-Box and the A-Box in a uniform way
- It can handle concept names, instances and concept descriptions
- It returns axioms, i.e. a fully working OWL DL ontology

SAIQL is a well suited for ontology extraction and ontology re-use

Basic implementation is already available
Current activities

- Ontologies & Semantic Web
- Engineering, Querying, Reasoning
- Multimedia
- Information extraction & mining
- Software & Service Engineering
- Peer-to-Peer
- Web2.0
- Tagora
- X-Media
- NeOn

BMBF
DAAD
SOAINVO
TWOUSE
Thank You!
Preliminaries for SAIQL Semantics - Example <is web>[

EquivalentClasses(MotorBike restriction(hasWheel cardinality(2)))
EquivalentClasses(Car restriction(hasWheel cardinality(4)))

Class(Convertible partial Car restriction(hasConvertibleTop someValuesFrom(owl:Thing)))
Class(Van partial restriction(hasWheel cardinality(4)) restriction(hasSlidingDoor someValuesFrom(owl:Thing)))

DisjointClasses(Convertible Van)
Individual(c type(Convertible)), Individual(v type(Van))
Individual(m type(MotorBike))

- $N_C = \{ \text{Car, Convertible, MotorBike, Van} \}$
- $N_{IP} = \{ \text{hasWheel, hasConvertibleTop, hasSlidingDoor} \}$
- $N_{DP} = \emptyset$
- $N_I = \{c,v,m\}$
- $N_{CD} = N_C \cup \{ \text{restriction( hasWheel cardinality(2) ), restriction( hasWheel cardinality(4) ), intersectionOf( Car restriction( hasConvertibleTop someValuesFrom( owl:Thing ) ) ) }, \ldots \}$