Ontology Based Data Access (OBDA)

Ian Horrocks
Information Systems Group
Department of Computer Science
University of Oxford
Motivation

- Huge **quantity** of data increasing at an exponential rate
- Identifying & accessing **relevant** data is of critical importance
- Handling data **variety & complexity** often turns out to be main challenge
- **Semantic Technology** can seamlessly integrate heterogeneous data sources
Semantic Technology

Rich *conceptual schemas* used to integrate heterogeneous sources

- **User Centric**
  - Schema modelled according to user intuitions
  - Independent of physical structure/storage of data

- **Declarative**
  - Improved understandability
  - Easier design, maintenance and evolution

- **Logic-based semantics**
  - Precise and formally specified meaning
  - Machine processable

- **Used at both design and query time**
  - Check validity and consequences of design
  - Easier query formulation and enriched query answers
Semantic Technology: Scalability Challenge

- Conceptual schemas use OWL ontology language
- OWL based on description logic SROIQ
  - ✔ Declarative
  - ✔ Clear semantics
  - ✔ Well understood computational properties (e.g., algorithms, decidability, complexity)
  - ✘ N2ExpTime-complete combined complexity
  - ✘ NP-hard data complexity (-v- AC0 for databases)

How can we provide (empirically) scalable query answering?
OWL Profiles

**OWL 2** defines language subsets, aka **profiles** that can be “more simply and/or efficiently implemented”

- **OWL 2 QL**
  - Based on **DL-Lite**
  - Efficiently implementable via rewriting into relational queries (OBDA)
Pipelines from oil facilities?
Q(\(x\)) ← (\(x\), rdf:type, :Pipeline) ∧
(\(x\), :fromFacility, \(y\)) ∧
(\(y\), rdf:type, :OilFacility)

Pipelines from oil facilities?
Pipelines from oil facilities?

\[ Q(?x) \leftarrow (?x, \text{rdf:type, Pipeline}) \land \
(?x, \text{fromFacility, } ?y) \land \
(?y, \text{rdf:type, OilFacility}) \]
Pipelines from oil facilities?

\[ Q(?x) \leftarrow (?x, \text{rdf:type}, \text{Pipeline}) \land (\text{fromFacility}, ?y) \land (?y, \text{rdf:type}, \text{OilFacility}) \]
Pipelines from oil facilities?

\[ Q(x) \leftarrow (x, \text{rdf:type}, \text{Pipeline}) \land (x, \text{fromFacility}, y) \land (y, \text{rdf:type}, \text{OilFacility}) \]

\{(p1, \text{rdf:type}, \text{Pipeline})
  (p1, \text{fromFacility}, f1)
  (f1, \text{rdf:type}, \text{OilFacility})
  (p2, \text{rdf:type}, \text{OilPipeline})
  (p2, \text{fromFacility}, f2)
  (f2, \text{rdf:type}, \text{OilFacility})
  (p3, \text{rdf:type}, \text{OilPipeline})\}
Pipelines from oil facilities?

Q(?x) ← (?x, rdf:type, :Pipeline) ∧ (?x, :fromFacility, ?y) ∧ (?y, rdf:type, :OilFacility)
Pipelines from oil facilities?

\[
Q(?x) \leftarrow (?x, \text{rdf:type, Pipeline}) \land 
(?x, \text{fromFacility, ?y}) \land 
(?y, \text{rdf:type, OilFacility})
\]
Q(?x) ← (?x, rdf:type, :Pipeline) ∧ (?x, :fromFacility, ?y) ∧ (?y, rdf:type, :OilFacility)

Pipelines from oil facilities?

OWL 2 QL ontology

SubClassOf(:OilPipeline
ObjectIntersectionOf(:Pipeline
ObjectSomeValuesFrom(:fromFacility :OilFacility)))

:p1, :p2, :p3

(:p1, rdf:type, :Pipeline)
(:p1, :fromFacility, :f1)
(:f1, rdf:type, :OilFacility)
(:p2, rdf:type, :OilPipeline)
(:p2, :fromFacility, :f2)
(:f2, rdf:type, :OilFacility)
(:p3, rdf:type, :OilPipeline)
Pipelines from oil facilities?

\[ Q(?x) \leftarrow (?x, \text{rdf:type}, \text{:Pipeline}) \land \\
(\forall y. (?x, \text{:fromFacility}, ?y) \land \\
(?y, \text{rdf:type}, \text{:OilFacility}) ) \]

\[ Q'(?x) \leftarrow (\forall x. (?x, \text{rdf:type}, \text{:Pipeline}) \land \\
(\exists y. (?x, \text{:fromFacility}, ?y) \land \\
(?y, \text{rdf:type}, \text{:OilFacility}) ) \lor (\exists x. (?x, \text{rdf:type}, \text{:OilPipeline}) ) \]

\[ \text{SubClassOf} (\text{:OilPipeline} \text{ ObjectIntersectionOf} (\text{:Pipeline} \text{ ObjectSomeValuesFrom} (\text{:fromFacility} \text{:OilFacility}))) \]
Pipelines from oil facilities?

Q(?x) ← (?x, rdf:type, :Pipeline) \land
    (?x, :fromFacility, ?y) \land
    (?y, rdf:type, :OilFacility)

Q′(?x) ← (?x, rdf:type, :Pipeline) \land
    (?x, :fromFacility, ?y) \land
    (?y, rdf:type, :OilFacility)
\lor (?x, rdf:type, :OilPipeline)

SubClassOf(:OilPipeline
          ObjectIntersectionOf(:Pipeline
                              ObjectSomeValuesFrom(:fromFacility :OilFacility))))

OWL 2 QL ontology

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>ID</th>
<th>Oil</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>N</td>
<td>f1</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>Y</td>
<td>f2</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>Y</td>
<td>Null</td>
<td></td>
</tr>
</tbody>
</table>
Pipelines from oil facilities?

\[
Q(x) \leftarrow (x, \text{rdf:type}, \text{Pipeline}) \land
(x, \text{fromFacility}, y) \land
(y, \text{rdf:type}, \text{OilFacility})
\]

\[
Q'(x) \leftarrow (x, \text{rdf:type}, \text{Pipeline}) \land
(x, \text{fromFacility}, y) \land
(y, \text{rdf:type}, \text{OilFacility})
\lor (x, \text{rdf:type}, \text{OilPipeline})
\]

\[
(\text{p1}, \text{rdf:type}, \text{Pipeline})
(\text{p1}, \text{fromFacility}, \text{f1})
(\text{f1}, \text{rdf:type}, \text{OilFacility})
(\text{p2}, \text{rdf:type}, \text{OilPipeline})
(\text{p2}, \text{fromFacility}, \text{f2})
(\text{f2}, \text{rdf:type}, \text{OilFacility})
(\text{p3}, \text{rdf:type}, \text{OilPipeline})
\]

\[
\text{OilPipeline} = \text{select ID from Pipeline where Oil = "Y"}
\]

\[
\text{Pipeline}
\begin{array}{ccc}
\text{ID} & \text{Oil} & \text{From} \\
p1 & N & f1 \\
p2 & Y & f2 \\
p3 & Y & \text{Null}
\end{array}
\]
Pipelines from oil facilities?

\[ Q(?x) \leftarrow (\?x, \text{rdf:type}, :\text{Pipeline}) \land (\?x, \text{:fromFacility}, ?y) \land (\?y, \text{rdf:type}, :\text{OilFacility}) \]

\[ Q'(?x) \leftarrow (\?x, \text{rdf:type}, :\text{Pipeline}) \land (\?x, \text{:fromFacility}, ?y) \land (\?y, \text{rdf:type}, :\text{OilFacility}) \lor (\?x, \text{rdf:type}, :\text{OilPipeline}) \]

R2RML mappings

\[
\text{OilPipeline} = \text{select ID from Pipeline where Oil} = "Y"
\]

\[
\text{select Pipeline.ID from Pipeline, \ldots where Pipeline.From} = \text{Facility.ID and \ldots UNION select ID from Pipeline where Oil} = "Y"
\]

OWL 2 QL ontology

\[
\text{SubClassOf}(\text{OilPipeline} \\text{ObjectIntersectionOf}(\text{Pipeline} \\text{ObjectSomeValuesFrom}(\text{:fromFacility :OilFacility})))
\]
Pipelines from oil facilities?

\[ Q(?x) \leftarrow (?x, \text{rdf:type}, \text{:Pipeline}) \land (?x, \text{:fromFacility}, ?y) \land (?y, \text{rdf:type}, \text{:OilFacility}) \]

\[ Q'(?x) \leftarrow (?x, \text{rdf:type}, \text{:Pipeline}) \land (?x, \text{:fromFacility}, ?y) \land (?y, \text{rdf:type}, \text{:OilFacility}) \lor (?x, \text{rdf:type}, \text{:OilPipeline}) \]

\begin{align*}
\text{select Pipeline.ID from Pipeline where Oil = "Y"} \\
\end{align*}

\(\text{(R2RML) mappings}\)

\[ \text{ObjectIntersectionOf(}\text{Pipeline} \land \text{ObjectSomeValuesFrom(}\text{:fromFacility} \land \text{OilFacility}))) \]

\(\text{SubClassOf(}\text{:OilPipeline}\)

\[ \text{pipeline} \]

\(\text{map}\)

\begin{tabular}{|c|c|c|}
\hline
ID & Oil & From \bigstrut[t] \\
\hline
p1 & N & f1 \bigstrut[t] \\
p2 & Y & f2 \bigstrut[t] \\
p3 & Y & Null \bigstrut[t] \\
\hline
\end{tabular}
OWL 2 QL and Query Rewriting

Given QL ontology $\mathcal{O}$ query $\mathcal{Q}$ and mappings $\mathcal{M}$:
Given QL ontology $\mathcal{O}$ query $Q$ and mappings $\mathcal{M}$:

- Use $\mathcal{O}$ to rewrite $Q \rightarrow Q'$ s.t. answering $Q'$ without $\mathcal{O}$ is equivalent to answering $Q$ w.r.t. $\mathcal{O}$ for any dataset.
OWL 2 QL and Query Rewriting

Given QL ontology $\mathcal{O}$ query $Q$ and mappings $\mathcal{M}$:

- Use $\mathcal{O}$ to rewrite $Q \rightarrow Q'$ s.t. answering $Q'$ without $\mathcal{O}$ is equivalent to answering $Q$ w.r.t. $\mathcal{O}$ for any dataset

- Map ontology queries $\rightarrow$ DB queries (typically SQL) using mappings $\mathcal{M}$ to rewrite $Q'$ into a DB query
OWL 2 QL and Query Rewriting

Given QL ontology $\mathcal{O}$ query $Q$ and mappings $M$:

- Use $\mathcal{O}$ to **rewrite** $Q \rightarrow Q'$ s.t. answering $Q'$ without $\mathcal{O}$ is equivalent to answering $Q$ w.r.t. $\mathcal{O}$ for any dataset

- **Map** ontology queries $\rightarrow$ DB queries (typically SQL) using mappings $M$ to rewrite $Q'$ into a DB query

- **Evaluate** (SQL) query against DB

![Diagram](image.png)
Query rewriting:
- uses ontology & mappings
- computationally hard
- ontology & mappings small

Query evaluation:
- ind. of ontology & mappings
- computationally tractable
- data sets very large
Query rewriting:
- uses ontology & mappings
- computationally hard
- ontology & mappings small

Query evaluation:
- ind. of ontology & mappings
- computationally tractable
- data sets very large

Other features:
- support for query formulation
Optique Architecture

Query rewriting:
- uses ontology & mappings
- computationally hard
- ontology & mappings small

Query evaluation:
- ind. of ontology & mappings
- computationally tractable
- data sets very large

Other features:
- support for query formulation
- "Bootstrapping" Ontology & mappings

Diagram:
- Application
- Query Formulation
- Ontology & Mapping Management
- Query Transformation
- Query Planning
- Stream Adapter
- Query Execution
- Q.E.
- End-user
- IT-expert
- Ontology
- Mappings
- Streaming data
- Databases
Query Rewriting — Issues

1 Rewriting
   - May be large (worst case exponential in size of ontology)
   - Queries may be hard for existing DBMSs

2 Mappings
   - May be difficult to develop and maintain

3 Expressivity
   - OWL 2 QL (necessarily) has (very) restricted expressive power, e.g.:
     - No functional or transitive properties
     - No universal (for-all) restrictions
     - ...
OWL Profiles – Beyond QL?

OWL 2 defines language subsets, aka profiles that can be “more simply and/or efficiently implemented”

- **OWL 2 QL**
  - Based on DL-Lite
  - Efficiently implementable via rewriting into relational queries (OBDA)

- **OWL 2 RL**
  - Based on “Description Logic Programs” ($\simeq$ DL $\cap$ Datalog)
  - Implementable via Datalog query answering

- **OWL 2 EL**
  - Based on $\mathcal{EL}^{++}$
  - Implementable via Datalog query answering plus “filtration”
RL/Datalog Query Ans. via Materialisation

Given (RDF) data DB, RL/Datalog ontology $\mathcal{O}$ and query $Q$: 
Given (RDF) data DB, RL/Datalog ontology $O$ and query $Q$:

- **Materialise** (RDF) data DB $\rightarrow$ DB$'$ s.t. evaluating $Q$ w.r.t. DB$'$ equivalent to answering $Q$ w.r.t. DB and $O$

  nb: Closely related to chase procedure used with DB dependencies
RL/Datalog Query Ans. via Materialisation

Given (RDF) data DB, RL/Datalog ontology $\mathcal{O}$ and query $Q$:

- **Materialise** (RDF) data DB $\rightarrow$ DB$'$ s.t. evaluating $Q$ w.r.t. DB$'$ equivalent to answering $Q$ w.r.t. DB and $\mathcal{O}$
  
  nb: Closely related to **chase** procedure used with DB dependencies

- **Evaluate** $Q$ against DB$'$
Materialisation — Example

\[ \{ \text{OilEquip} \sqsubset \forall \text{hasPart.OilEquip} \]
Materialisation — Example

\[ 0 \}
\[
\text{OilEquip} \subseteq \forall \text{hasPart} . \text{OilEquip}
\]

\[ DB \}
\[
\text{OilEquip}(a) \\
\text{hasPart}(a, b) \\
\text{hasPart}(b, c)
\]
Materialisation — Example

\[ O \{ \text{OilEquip} \sqsubseteq \forall \text{hasPart}. \text{OilEquip} \] 

\[ DB \{ \text{OilEquip}(a), \text{hasPart}(a, b), \text{hasPart}(b, c) \] 

\[ Q_1 \quad Q(x) \gets \text{OilEquip}(x) \]
Materialisation — Example

\[ \emptyset \}

\text{OilEquip} \subseteq \forall \text{hasPart} \cdot \text{OilEquip} \]

\begin{align*}
\text{DB} \} \quad \\
\text{OilEquip}(a) \\
\text{hasPart}(a, b) \\
\text{hasPart}(b, c) \\
\end{align*}

\[ Q_1 \quad Q(x) \leftarrow \text{OilEquip}(x) \quad \leadsto \quad \{ a \} \]
Materialisation — Example

\[ \emptyset \]

\[ O \{ \text{OilEquip} \sqsubseteq \forall \text{hasPart}. \text{OilEquip} \quad \text{OilEquip}(x) \land \text{hasPart}(x, y) \rightarrow \text{OilEquip}(y) \} \]

\[ \text{DB} \{ \text{OilEquip}(a) \quad \text{hasPart}(a, b) \quad \text{hasPart}(b, c) \} \]

\[ Q_1 \quad Q(x) \leftarrow \text{OilEquip}(x) \]
Materialisation — Example

\[
\emptyset \begin{cases} 
\text{OilEquip} \subseteq \forall \text{hasPart}. \text{OilEquip} & \text{OilEquip}(x) \land \text{hasPart}(x, y) \rightarrow \text{OilEquip}(y) \\
\end{cases}
\]

DB \begin{cases} 
\text{OilEquip}(a) \\
\text{hasPart}(a, b) \\
\text{hasPart}(b, c) \\
\end{cases}

DB' \begin{cases} 
\text{OilEquip}(a) \\
\text{hasPart}(a, b) \\
\text{hasPart}(b, c) \\
\end{cases}

Q_1 \begin{cases} 
Q(x) \leftarrow \text{OilEquip}(x) \\
\end{cases}
Materialisation — Example

\[ O \{ \text{OilEquip} \sqsubseteq \forall \text{hasPart}. \text{OilEquip} \quad \text{OilEquip}(x) \land \text{hasPart}(x, y) \rightarrow \text{OilEquip}(y) \} \]

\[ \text{DB} \{ \text{OilEquip}(a) \\
\qquad \text{hasPart}(a, b) \\
\qquad \text{hasPart}(b, c) \} \]

\[ \text{Q}_1 \quad Q(x) \leftarrow \text{OilEquip}(x) \]

\[ \text{DB}' \{ \text{OilEquip}(a) \\
\qquad \text{hasPart}(a, b) \\
\qquad \text{hasPart}(b, c) \} \]
Materialisation — Example

\[\mathcal{O}\begin{cases}
\text{OilEquip} \sqsubseteq \forall \text{hasPart}. \text{OilEquip} \\
\text{Oil Equip}(x) \land \text{hasPart}(x,y) \rightarrow \text{Oil Equip}(y)
\end{cases}\]

\[\begin{aligned}
\text{DB} &\begin{cases}
\text{OilEquip}(a) \\
\text{hasPart}(a,b) \\
\text{hasPart}(b,c)
\end{cases} \\
\text{DB}' &\begin{cases}
\text{OilEquip}(a) \\
\text{hasPart}(a,b) \\
\text{hasPart}(b,c) \\
\text{OilEquip}(b) \\
\text{OilEquip}(c)
\end{cases}
\end{aligned}\]

\[\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Oil Equip}(x)\]
Materialisation — Example

\[ O \{ \text{OilEquip} \subseteq \forall \text{hasPart}. \text{OilEquip} \quad \text{OilEquip}(x) \land \text{hasPart}(x, y) \rightarrow \text{OilEquip}(y) \} \]

\[ DB \{ \text{OilEquip}(a) \quad \text{hasPart}(a, b) \quad \text{hasPart}(b, c) \} \]

\[ Q_1 \quad Q(x) \leftarrow \text{OilEquip}(x) \]

\[ DB' \{ \text{OilEquip}(a) \quad \text{hasPart}(a, b) \quad \text{hasPart}(b, c) \quad \text{Oil Equip}(b) \quad \text{Oil Equip}(c) \} \]

\[ \sim \sim \{ a, b, c \} \]
Materialisation — Issues

1 Scalability
   - Ptime complete
   - Efficiently implementable in practice?

2 Updates
   - Additions relatively easy (continue materialisation)
   - But what about retraction?

3 Migrating data to RDF
   - Materialisation assumes data in “special” (RDF triple) store
   - How can legacy data be migrated?

4 Expressivity
   - $QL \not\subseteq RL$; in particular, no invention of new individuals
Materialisation: Scalability

- Efficient **Datalog/RL** engine is critical
- Existing approaches mainly target distributed “shared-nothing” architectures, often via **map reduce**
  - High communication overhead
  - Typically focus on small fragments (e.g., RDFS), so don’t really address expressivity issue
  - Even then, query answering over (distributed) materialized data is non-trivial and may require considerable communication
RDFox Datalog Engine

- Targets SOTA **main-memory, multi-core** architecture
  - Optimized in-memory storage with ‘mostly’ lock-free parallel inserts
  - Memory efficient: commodity server with 128 GB can store >$10^9$ triples
  - Exploits multi-core architecture: 10-20 x speedup with 32/16 threads/cores
  - LUBM 120K (>10$^{10}$ triples) in 251s (20M t/s) on T5-8 (4TB/1024 threads)
RDFox Datalog Engine

- **Incremental addition and retraction** of triples
  - Retraction via novel FBF “view maintenance” algorithm
  - Retraction of 5,000 triples from materialised LUBM 50k in less than 1s

- Many other **novel features**
  - Handles more general (than RL) Dalalog and SWRL rules
  - SPARQL features such as BIND and FILTER in rule bodies
  - Native equality handling (owl:sameAs) via rewriting
  - Stratified negation as failure (NAF)
Materialisation: Data Migration

- Need to specify a suitable **migration** process
  - Use **R2RML** mappings to extract data and transform into RDF
  - But where do these mappings come from?

- Recall query rewriting:
  - **Mappings** $\mathcal{M}$ are R2RML mappings
  - Run mappings in reverse to extract and transform data

- “Lazy ETL”
  - Deploy query rewriting (OBDA) system
  - Extend $\mathcal{O}$ and $\mathcal{M}$ as needed
  - Use $\mathcal{M}$ to ETL data into RDF store
Materialisation: Expressivity

- RL is more powerful than QL, but QL $\not\subseteq$ RL
  - In particular, no “individual creation” (RHS existentials)
  - Can’t express, e.g., $\text{OilPipeline} \sqsubseteq \text{Pipeline} \sqcap \exists \text{fromFacility.OilFacility}$

- Recall **OWL 2 EL**
  - Based on $\mathcal{EL}^{++}$
  - Implementable via Datalog query answering plus “filtration”
OWL 2 EL via Datalog + Filtration

Given (RDF) Data Set, EL ontology $\mathcal{O}$ and query $Q$: 
OWL 2 EL via Datalog + Filtration

Given (RDF) Data Set, EL ontology $O$ and query $Q$:

- **Over-approximate** $O$ into Datalog program $D$
OWL 2 EL via Datalog + Filtration

Given (RDF) Data Set, EL ontology $\mathcal{O}$ and query $\mathcal{Q}$:

- **Over-approximate** $\mathcal{O}$ into Datalog program $D$
- **Evaluate** $\mathcal{Q}$ over $D$ + Data Set (via materialisation)
OWL 2 EL via Datalog + Filtration

Given (RDF) Data Set, EL ontology $\mathcal{O}$ and query $Q$:

- **Over-approximate** $\mathcal{O}$ into Datalog program $D$

- **Evaluate** $Q$ over $D$ + Data Set (via materialisation)

- Use (polynomial) **Filtering Procedure** to eliminate spurious answers
Discussion

- **QL-Rewriting** has many advantages
  - Data can be left untouched and in legacy storage
  - Exploits existing DB infrastructure and scalability
  - ...

- But what if **more expressiveness/flexibility** is needed?
  - Query answering for EL and RL still tractable (polynomial)
  - Critically depend on Datalog scalability – RDFox to the rescue!
  - Easy migration path from QL-rewriting via “lazy ETL”
Future Work

- **Piloting**, evaluation and tuning
- Porting to other large-scale architectures
- Semantic (data) **partitioning** for distributed architectures
- (Incremental maintenance of) **aggregations**
- Improved **query planning**
- **Stream** reasoning
- **Hybrid rewriting/materialisation** (on demand) approach
- Expressiveness beyond RL/EL via **PAGOdA** techniques
- ...
Acknowledgements
Thank you for listening
Thank you for listening

Any questions?