A Datalog+ RuleML 1.01 Architecture for Rule-Based Data Access in Ecosystem Research

(Long version: cs.unb.ca/~boley/talks/RulesOBDA.pdf)

Harold Boley (UNB, RuleML)
Rolf Grütter (WSL)
Gen Zou (UNB)
Tara Athan (RuleML)
Sophia Etzold (WSL)

The 8th International Web Rule Symposium (RuleML 2014)
August 18-20, 2014, Prague, Czech Republic
What is Knowledge-Based Data Access?

- KBDA applies **AI / Semantic Technologies** to databases
- Founded on (logical) **knowledge** representation for:
  - **Ontology-Based Data Access (OBDA)**, e.g. data **integration/federation** and query optimization
  - **Rule-Based Data Access (RBDA)**, e.g. **Datalog / deductive databases** and query answering
- **Knowledge Base** as generalized **global** schema for data in **local** (e.g., relational or graph) DBs
- KB module amplifies data storage & query execution of **distributed, heterogeneous** (No)SQL DBs
- Provides **multi-purpose** knowledge level for data
Preview: Unified Architecture
Why Knowledge-Based Data Access?

- Domain knowledge utilized to deal with data torrent
  - Domain experts conceptually **fold** data / **unfold** queries via **Mappings** defined **with** IT (SQL, SPARQL, ...) experts
  - User concepts are captured in **Knowledge Base** for **domain-enriched** database materialization / querying **without** IT experts
- **Engines** use KB to deduce answers implicit in DBs
- **Analytics** enabled by queries exploring hypotheses
- KB as major organizational resource also for, e.g.:
  - Data validation (consistency, completeness, ...)
  - Schema-level query answering (even without DBs)
RBDA Realizes Uniform KBDA – 1 of 3: Queries as Rules

1. a) A conjunctive *query* is a special Datalog *rule* whose body can be *rewritten* (see 2.) and *unfolded* (see 3.), and whose head instantiates the distinguished answer variables of the body

   b) KBDA ontologies beyond RDF Schema (RDFS) often permit *Boolean* conjunctive queries corresponding to *integrity* rules

2. ...

3. ...
RBDA Realizes Uniform KBDA – 2 of 3: KBs as Rules

1. ...

2. KBDA $KB$ supports, e.g., query *rewriting* through global-schema-level reasoning, including with RDFS *taxonomies* or Datalog *rule* axioms, and DL-Lite *(OWL 2 QL)* or *(head-)*existential *rules*; KBDA *rules* also permit Description Logic Programs *(OWL 2 RL)*, Datalog$^\pm$, and Disjunctive Datalog. [Semantics of *ontology languages* customizable for expressivity and efficiency requirements by adding/deleting *rules* *(SPIN)*]

3. ...
RBDA Realizes Uniform KBDA – 3 of 3: Mappings as Rules

1. ...

2. ...

3. KBDA data integration is centered on Global-As-View (GAV) mappings, which are Datalog rules for, e.g., unfolding each global head predicate to (a join, i.e. conjunction, of) local body predicates.
Example: Forest/Orchard Knowledge
EntityWithTree KB: Named Root Class (1)

Subsumption axioms (in higher-order rule syntax):
EntityContainingAtLeastOneTree ← Forest.
EntityContainingAtLeastOneTree ← Orchard.
Forest ← Woodland.

Root of taxonomy tree of tree-containing entities to see the forest for the trees

“←” is taxonomy-style ‘subsumes’ infix
EntityWithTree KB: Named Root Class (2)

Subsumption axioms (in higher-order rule syntax):
EntityContainingAtLeastOneTree :- Forest.
EntityContainingAtLeastOneTree :- Orchard.
Forest :- Woodland.

Root of taxonomy tree of tree-containing entities to see the forest for the trees

“:-” is rule-style ‘if’ infix
EntityWithTree KB: Constructed Root Class

Subsumption axioms (in higher-order rule syntax):

\[ \exists \text{contains.Tree} : - \text{Forest.} \]
\[ \exists \text{contains.Tree} : - \text{Orchard.} \]
\[ \text{Forest} : - \text{Woodland.} \]

Cf. ontology-style (description logic) axioms:

\[ \exists \text{contains.Tree} \equiv \text{Forest} \]
\[ \exists \text{contains.Tree} \equiv \text{Orchard} \]

Entities each having a contains property with at least one value in class Tree
Three Dimensions of $\text{KBDA}_s$: $R,Q,m$

Rule-Based Data Querying
Query Rewriting and Unfolding

- Mediator strategy uses: KB to *rewrite* \( Q \) to \( Q' \) and Mappings (*rules*) to *unfold* \( Q' \) to \( Q_i'' \)
- KB can be *ontology*, e.g. in OWL 2 QL (DL-Lite), or *rules*
- Abstract (relational/graph/...) queries \( Q_i'' \) -grounded (to SQL/SPARQL/...) for \( DB_i \)
- Each (relational/graph/...) database \( DB_i \) left as original; answers at ♦

(a)

---

Q → Rewriting → Q' → Unfolding → Mappings → Q_i'' → DB_i

Two ‘backward’ transformations
Query Rewriting and Unfolding

- Mediator strategy uses:
  - KB to *rewrite* Q to Q’ and Mappings (rules) to *unfold* Q’ to Q_i’
  - KB can be **ontology**, e.g. in OWL 2 QL (DL-Lite), or **rules**
  - Abstract (relational/graph/...) queries Q_i’ ♦ -grounded (to SQL/SPARQL/...) for DB_i
  - Each (relational/graph/...) database DB_i left as original; answers at ♦

(a)

**KB**

Q → Rewriting
Q’

Global schema

**Mappings**

Q_1’
Q_2’
Q_n’

DB_1
DB_2
DB_n

Unfolding

Local schemas
Query Rewriting Use of KB

- In Information Retrieval:
  Query expansion
  - With increased recall
  - Without loss of precision

- From Logic Programming (for Horn expressivity):
  Can use resolution method for KB-enrichment of a given (conjunctive) query with expanded (conjunctive) queries so that, for any DB, the answers to the enriched queries no longer using the KB are the same as the answers to the original query using the KB.
Ontology-to-Rule Clausification of EntityWithTree KB Omitting Orchard

Description Logic subsumptions (as higher-order rules):

\[ \exists \text{contains}.\text{Tree} \ :- \ \text{Forest}. \]
\[ \text{Forest} \ :- \ \text{Woodland}. \]

Horn Logic rules (the first with conjunctive head):

\[ (\text{contains}(\text{x} \ s(\text{x})) \ \land \ \text{Tree}(s(\text{x}))) \ :- \ \text{Forest}(\text{x}). \]
\[ \text{Forest}(\text{x}) \ :- \ \text{Woodland}(\text{x}). \]

Horn Logic rules (the first head split into two conjuncts):

\[ \text{contains}(\text{x} \ s(\text{x})) \ :- \ \text{Forest}(\text{x}). \]
\[ \text{Tree}(s(\text{x})) \ :- \ \text{Forest}(\text{x}). \]
\[ \text{Forest}(\text{x}) \ :- \ \text{Woodland}(\text{x}). \]
KB Rules Perform Rewriting of Given Query

**KB with Horn rules (from above) for rewriting of query rules:**
contains(?x s(?x)) :- Forest(?x).
Tree(s(?x)) :- Forest(?x).
Forest(?x) :- Woodland(?x).

*Rewriting Datalog query rule to obtain extra query rules:*

\begin{align*}
q(?z) & :- \text{contains}(?z ?y) \land \text{Tree}(?y). \\
q(?z) & :- \text{Forest}(?z) \land \text{Tree}(s(?z)). \\
q(?z) & :- \text{contains}(?z s(?x)) \land \text{Forest}(?x). \\
q(?z) & :- \text{Forest}(?z) \land \text{Forest}(?z). \\
q(?z) & :- \text{Forest}(?z). \\
q(?z) & :- \text{Woodland}(?z).
\end{align*}

Q: Given

Expansion

Q': Given $\cup$ Expansion
Query Unfolding Use of Mappings to Original Database Sources

- Datalog rules **bridging** between:
  - KB
  - Distributed DBs
- Use **partial deduction**-like unfolding (and simplification) of (conjunctive) KB queries to (conjunctive) abstract DB queries
  - Abstract relational queries grounded to SQL, abstract graph queries grounded to SPARQL, etc.
  - Lower-level optimization and execution by SQL, SPARQL, etc. engines
- Generated queries distributed over multiple DBs as indicated by “source.” name prefixes

Intuitive idea of **query unfolding**
Sample Mapping Rules to Three Local Data Sources

**Map KB predicates to locDB/regionDB tables for geo data:**
contains(?x ?y) :- locDB.cnt(?x ?kind ?y).
contains(?x ?y) :- regionDB.sub(?x ?r) ∧ locDB.cnt(?r ?kind ?y).

**Map KB predicates to locDB/ecoDB tables for forestry data:**
Tree(?t) :- locDB.cnt(?plot "tree" ?t).
Tree(?t) :- ecoDB.Plant(?plot "tree" ?size ?t).
Forest(?x) :- ecoDB.Habitat(?plot "forest" ?size ?x).
Woodland(?x) :- ecoDB.Habitat(?plot "wood" ?size ?x).
Mapping Rules Perform Unfolding of Rewritten Queries

**Union of conjunctive queries as Datalog rules (rewritten):**

- \( q(\texttt{?z}) :\) contains(\(\texttt{?z} \texttt{?y}\)) \(\land\) Tree(\(\texttt{?y}\)).
- \( q(\texttt{?z}) :\) Forest(\(\texttt{?z}\)).
- \( q(\texttt{?z}) :\) Woodland(\(\texttt{?z}\)).

**Unfolding above queries via mappings from previous slide:**

- \( q(\texttt{?z}) :\) locDB.cnt(\(\texttt{?z} \texttt{?kind} \texttt{?y}\)) \(\land\) locDB.cnt(\(\texttt{?plot} \texttt{"tree"} \texttt{?y}\)).
- \( q(\texttt{?z}) :\) locDB.cnt(\(\texttt{?z} \texttt{"tree"} \texttt{?y}\)).
- \( q(\texttt{?z}) :\) locDB.cnt(\(\texttt{?z} \texttt{?kind} \texttt{?y}\)) \(\land\) ecoDB.Plant(\(\texttt{?plot} \texttt{"tree"} \texttt{?size} \texttt{?y}\)).
- \( q(\texttt{?z}) :\) regionDB.sub(\(\texttt{?z} \texttt{?r}\)) \(\land\) locDB.cnt(\(\texttt{?r} \texttt{?kind} \texttt{?y}\)) \(\land\) locDB.cnt(\(\texttt{?plot} \texttt{"tree"} \texttt{?y}\)).
- \( q(\texttt{?z}) :\) regionDB.sub(\(\texttt{?z} \texttt{?r}\)) \(\land\) locDB.cnt(\(\texttt{?r} \texttt{"tree"} \texttt{?y}\)).
- \( q(\texttt{?z}) :\) regionDB.sub(\(\texttt{?z} \texttt{?r}\)) \(\land\) locDB.cnt(\(\texttt{?r} \texttt{?kind} \texttt{?y}\)) \(\land\) ecoDB.Plant(\(\texttt{?plot} \texttt{"tree"} \texttt{?size} \texttt{?y}\)).
- \( q(\texttt{?z}) :\) ecoDB.Habitat(\(\texttt{?plot} \texttt{"forest"} \texttt{?size} \texttt{?z}\)).
- \( q(\texttt{?z}) :\) ecoDB.Habitat(\(\texttt{?plot} \texttt{"wood"} \texttt{?size} \texttt{?z}\)).
Three Dimensions of $\text{KBDA}_s$: $R, Q, w$

Rule-based Data Querying

Warehouse
Database Materialization after Folding

- Warehouse strategy uses:
  - Mappings (same rules as for unfolding) to fold DB_i to DB and KB to materialize Database DB to DB’
  - KB can be ontology, e.g. in OWL 2 RL (DLP), or rules
  - Query is left as original; answers at solid triangular arrow head

(b)
Database Materialization after Folding

- Warehouse strategy uses:
  Mappings (same **rules** as for **unfolding**) to **fold** $DB_i$ to DB and $KB$ to **materialize** Database DB to $DB'$
- $KB$ can be **ontology**, e.g. in OWL 2 RL (DLP), or **rules**
- Query is left as original; answers at solid triangular arrow head

---

(b)

Q

Global schema

```
<table>
<thead>
<tr>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB'</td>
</tr>
<tr>
<td>Materialization</td>
</tr>
<tr>
<td>DB</td>
</tr>
<tr>
<td>Folding</td>
</tr>
</tbody>
</table>

Mappings

```

Local schemas

```
| $DB_1$ |
| $DB_2$ |
| ... |
| $DB_n$ |
```

---
EWT KB and DB: ‘Populated’ KB

Subsumptions and Data (as higher-order rules and facts):

- \( \exists \text{contains.Tree} : \text{Forest} \).
- \( \exists \text{contains.Tree(e)} \).
- \( \exists \text{Forest(f)} \).
- \( \exists \text{Woodland(w)} \).

Diagram:

```
                   \( \exists \text{contains.Tree} \)
                   /   \e
```
```
  \( \text{Forest} \)
  /   
```
```
    \( \text{Woodland} \)
```
```
      \( f \)
```
```
      \( w \)
```
Applying All KB Rules to All DB Facts (1)

Subsumptions and Data (as higher-order rules and facts):

∃contains.Tree :- Forest.
Forest :- Woodland.

∃contains.Tree

 existential quantifier

 e f

 Forest

 f w

 Woodland

 w

∃contains.Tree(e).
∃contains.Tree(f).
Forest(f).
Forest(w).
Woodland(w).
Applying All KB Rules to All DB Facts (2)

Subsumptions and Data (as higher-order rules and facts):

\[ \exists \text{contains.Tree} : - \text{Forest.} \]
\[ \text{Forest} : - \text{Woodland.} \]

\[ \exists \text{contains.Tree} \]
\[ \text{Forest} \]
\[ \text{Woodland} \]

\[ \text{w} \]
\[ \text{w} \]
\[ \text{w} \]

\[ \text{e} \]
\[ \text{f} \]

\[ \text{Forest(e).} \]
\[ \text{Forest(f).} \]
\[ \text{Forest(w).} \]
\[ \text{Woodland(w).} \]
Fixpoint with No New Rule-Derived Facts

Subsumptions and Data (as higher-order rules and facts):

\[ \exists \text{contains.Tree} : \text{Forest.} \]
\[ \text{Forest} :: \text{Woodland.} \]

\[ \exists \text{contains.Tree} \]

\[ \text{Forest}(e) \]
\[ \text{Forest}(f) \]
\[ \text{Forest}(w) \]
\[ \text{Woodland}(w) \]
Three Dimensions of KBDA<sub>s</sub>

KBDA<sub>s</sub> strategy

Knowledge-Based Data Access
Unified Architecture

- Combines strategies (a)-(c) of earlier slides
- Meets the needs of ΔForest case study
Unified Architecture

• Combines strategies (a)-(c) of earlier slides
• Meets the needs of ΔForest case study
**RBDAₜ-Style KBDAₜ Architecture: Expressivity of Rule Systems**

- The language of the global schema can be generalized from unary/binary (OBDAₜ) to n-ary predicates (RBDAₜ).
- When decidability of querying is not required, RBDAₜ expressivity can be extended from Datalog, Datalog⁺, and description logic to Datalog⁺, Horn logic, and FOL, as enabled by Deliberation RuleML 1.01.
- Features customizable with the MYNG 1.01 GUI.
- Moreover, Reaction RuleML 1.0 can express updates, as needed for KDₜ (Ontology-based Data Management).
RBDA$_s$-Style KBDA$_s$ Architecture: Uniformity via Rule Systems

- Rule-based style of Unified Architecture (earlier slide)
- Presentation syntax (":-"), serialization ("<RuleML>"), and semantics approach (model theory) uniform from queries (+ integrity constraints) to KBs to mappings to abstract DBs
- Division of labor between KB rules and mapping rules can be modified without crossing paradigm boundaries
  - Allows KB- and mapping-directed normal forms
- Assumptions (unique-name and closed-world) of DBs accommodated by default assumptions of rule systems
Forest: Schemas for Architecture (a)-(d)
Questions Addressed

1. Are there sufficiently many eligible plots in order to perform an analysis per main tree species?
2. Are there sufficiently many eligible plots in order to perform an analysis per main tree species and climatic region?
3. Which eligible plots represent pure tree stands and which eligible plots represent mixed tree stands?
Query Rewriting

q(?plot) :- EligiblePlot(?plot)
    .TreeStandKey(?id ?plot "oak")
    .TreeStandAbundance(?id ?pct)
    ?pct >= 15.

Exist ?id .TreeStandKey(?id ?plot ?sp) .TreeStandAbundance(?id ?pct) :-

q(?plot) :- EligiblePlot(?plot)
    .TreeStandMerged(?plot "oak" ?pct)
    ?pct >= 15.
Query Unfolding

q2(?plot) :- treestandmerged(?plot "oak" ?pct)
   ?pct >= 15.

.treestandmerged(?plot "oak" ?pct) :-
   ekf.dom(?plot "quercus petraea" ?pct1)
   ekf.dom(?plot "quercus robur" ?pct2)

q2(?plot) :- ekf.dom(?plot "quercus petraea" ?pct1)
   ekf.dom(?plot "quercus robur" ?pct2)
   ?pct1+?pct2 >= 15.
Conclusions

- Ontology-Based Data Access (OBDA) founded on three kinds of rules: *Query rules* (including integrity rules), *KB rules* (for query rewriting and DB materialization), as well as *mapping rules* (for query unfolding and DB folding)
- OBDA complemented by Rule-Based Data Access (RBDA) and generalized to Knowledge-Based Data Access (KBDA)
- Specified an RBDA-uniform KBDA$_S$ architecture with unified mediator, warehouse, and bidirectional strategies
- RuleML used for XML-serialized rules, MYNG-customized rule expressivity, and platform-independent RBDA
- Introduced $\Delta$Forest specialization of RBDA architecture for statistical data analysis in ecosystem research at WSL
Future Work (1)

- Translate simplified presentation syntax into released XML serialization of Deliberation RuleML 1.01 / MYNG 1.01
- Support implementations of specified architecture reusing (open source) KBDA technology (cf. RBDA wiki page)
- For high-precision RBDQs language support, complement current techniques of Datalog⁺ RuleML for Datalog⁻ RuleML using context-sensitive/semantic validators for “-” constraints
- Evaluate (mediator/warehouse, relational/graph, ...) trade-offs for KBDQs in PSOA RuleML as executed in PSOATransRun
- Develop ΔForest study at WSL for extended and new data sources of big volume, variety, and velocity (e.g., about climate change)
- Augment geospatial KBDQs mappings with Optique mapping (bootstrapping, repair, ...) techniques
Future Work (2)

• Compare engines for OBDQs and RBDQs, including HYDRA and RDFox, w.r.t. expressivity and efficiency
• Adapt Semantic Automated Discovery and Integration (SADI) test cases for KBDQs experiments in PSOA RuleML querying
• Evaluate Abstract Logic-based Architecture Storage systems & Knowledge base Analysis (ALASKA) for RBDA
• Extend the KBDA architecture with semantic annotation rules for (Deep) Web data extraction (Deep Web Mediator)
• Use Grailog for KBDA data and knowledge visualization
• Explore synergies between the logical KBDA approach with statistical approaches, e.g. from Statistical Relational AI