A use case: the WONDER system

Some technical details

Summary

COMP718: Ontologies and Knowledge Bases
Lecture 9: Ontology/Conceptual Model based Data Access

Maria Keet

email: keet@ukzn.ac.za
home: http://www.meteck.org

School of Mathematics, Statistics, and Computer Science
University of KwaZulu-Natal, South Africa

10 April 2012
Outline

1. A use case: the WONDER system
   - Proposed solution
   - Realisation of the solution
   - Summary

2. Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - ‘Impedance’ mismatch
     - Mapping assertions
   - Query answering
An ontology with a very large ABox

Thus far, we’ve seen mostly small ontologies with lots of features, but with little data, and mostly theory.

This and next lecture:
- Scale up to realistic size knowledge base handling large amounts of data
- Set up an ontology-driven information system

To realise this, we need
- A language of relatively low computational complexity
- A way to store large amounts of data
- Some mechanism to link up the previous two ingredients
- Query (and reason over) the combination of the previous three

Use the “Ontology-Based Data Access” approach
An ontology with a very large ABox

- Thus far, we’ve seen mostly small ontologies with lots of features, but with little data, and mostly theory
- This and next lecture:
  - Scale up to realistic size knowledge base handling large amounts of data
  - Set up an ontology-driven information system
- To realise this, we need
  - A language of relatively low computational complexity
  - A way to store large amounts of data
  - Some mechanism to link up the previous two ingredients
  - Query (and reason over) the combination of the previous three
- Use the “Ontology-Based Data Access” approach
An ontology with a very large ABox

- Thus far, we’ve seen mostly small ontologies with lots of features, but with little data, and mostly theory
- This and next lecture:
  - Scale up to realistic size knowledge base handling large amounts of data
  - Set up an ontology-driven information system
- To realise this, we need
  - A language of relatively low computational complexity
  - A way to store large amounts of data
  - Some mechanism to link up the previous two ingredients
  - Query (and reason over) the combination of the previous three
- Use the “Ontology-Based Data Access” approach
Access to relational databases

- **“Sysadmin interface”** for developers and users:
  - SQL or any of its close variants (e.g., StruQL)
  - Need to know how the data is stored in the database
  - Writing large queries is still time-consuming even for experts
  - One-off queries or some manual query management for recurring queries

- **“End-user interface”**
  - Canned queries and pre-computed queries
  - Inflexible for data analysis
  - Burden on sysadmin for application layer updates to meet whims of the user

- Database integration
Access to relational databases

- "Sysadmin interface" for developers and users:
  - SQL or any of its close variants (e.g., StruQL)
  - Need to know *how* the data is stored in the database
  - Writing large queries is still time-consuming even for experts
  - One-off queries or some manual query management for recurring queries

- "End-user interface"
  - Canned queries and pre-computed queries
  - Inflexible for data analysis
  - Burden on sysadmin for application layer updates to meet whims of the user

- Database integration
Access to relational databases

- **“Sysadmin interface”** for developers and users:
  - SQL or any of its close variants (e.g., StruQL)
  - Need to know *how* the data is stored in the database
  - Writing large queries is still time-consuming even for experts
  - One-off queries or some manual query management for recurring queries

- **“End-user interface”**
  - Canned queries and pre-computed queries
  - Inflexible for data analysis
  - Burden on sysadmin for application layer updates to meet whims of the user

- Database integration
Access to relational databases

- “Sysadmin interface” for developers and users:
  - SQL or any of its close variants (e.g., StruQL)
  - Need to know *how* the data is stored in the database
  - Writing large queries is still time-consuming even for experts
  - One-off queries or some manual query management for recurring queries

- “End-user interface”
  - Canned queries and pre-computed queries
  - Inflexible for data analysis
  - Burden on sysadmin for application layer updates to meet whims of the user

- Database integration
Outline

1. A use case: the WONDER system
   - Proposed solution
   - Realisation of the solution
   - Summary

2. Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - ‘Impedance’ mismatch
     - Mapping assertions
   - Query answering
Web-accessible databases
A use case: the WONDER system

Web-accessible databases

- Fail to meet the usability requirement
+ Maximal query expressiveness
Web-accessible databases

- Meets the usability requirement
- Not all queries can be expressed
Case study and problem: the Horizontal Gene Transfer DB

- Lots of data made available on the Web by the Life Science field
- HGT-DB is a web-accessible genomics database about prokaryotic organisms
- Web interface with pre-computed queries or restricted SQL queries
- Contains 477 organisms and 1,445,840 genes
- 4GB genomics database
- Tables with 16-46 columns
Sample Information Request:
*Retrieve all genes of the organisms Neisseria for which horizontal gene transfer is predicted or have a GC3 value > 80*

- Simple HTML interface for posing canned queries and to retrieve text-files of pre-computed queries
- Substantial limitations on expressiveness of queries: **Domain users cannot extract all the information contained in the database!**

**Problem:** Users (geneticists) need to know *what* is in the database, *how* the data is stored, and need to know SQL or use the pre-computed queries, which is what limits their data analysis
Sample Information Request:
*Retrieve all genes of the organisms Neisseria for which horizontal gene transfer is predicted or have a GC3 value > 80*

- Simple HTML interface for posing canned queries and to retrieve text-files of pre-computed queries
- Substantial limitations on expressiveness of queries: **Domain users cannot extract all the information contained in the database!**

**Problem:** Users (geneticists) need to know what is in the database, how the data is stored, and need to know SQL or use the pre-computed queries, which is what limits their data analysis
Case study and problem

- Sample Information Request:
  
  *Retrieve all genes of the organisms Neisseria for which horizontal gene transfer is predicted or have a GC3 value > 80*
  
  - Simple HTML interface for posing canned queries and to retrieve text-files of pre-computed queries
  
  - Substantial limitations on expressiveness of queries: **Domain users cannot extract all the information contained in the database!**

- **Problem:** Users (geneticists) need to know what is in the database, how the data is stored, and need to know SQL or use the pre-computed queries, which is what limits their data analysis
Proposed solution

A solution

- **Solution**: add a semantic layer to the database, let the users construct queries graphically, and generate the SQL/SPARQL/EQL query automatically from the graphics

- **Constraints**: any solution needs to be scalable, usable, and web-based

- **Realisation**: graphical, web-based ‘Ontology’-Based Data Access (COnceptual MOdel-based Data Access – COMODA)
A solution

- **Solution**: add a semantic layer to the database, let the users construct queries graphically, and generate the SQL/SPARQL/EQL query automatically from the graphics

- **Constraints**: any solution needs to be scalable, usable, and web-based

- **Realisation**: graphical, web-based ‘Ontology’-Based Data Access (COnceptual MOdel-based Data Access – COMODA)
A solution

- **Solution**: add a semantic layer to the database, let the users construct queries graphically, and generate the SQL/SPARQL/EQL query automatically from the graphics
- **Constraints**: any solution needs to be scalable, usable, and web-based
- **Realisation**: graphical, web-based ‘Ontology’-Based Data Access (COnceptual MOdel-based Data Access – COMODA)
A use case: the WONDER system

Proposed solution

Approach

User → User Queries → Semantic Layer

User Queries

Semantic Layer

DL-Lite Ontology

OWL2 QL

Mappings → Data Layer

DB
Web-ONtology baseD Extraction of Relational data (WONDER)
Components of this particular system

- Developed in collaboration at “La Sapienza” University in Rome and Free University of Bozen-Bolzano
- **Formal languages**: *DL-Lite* family, OWL 2 QL
  - This context: ontology ≡ Description Logics Knowledge Base
- **OBDA-enabled reasoner**: **QuOnto**
- **RDBMS**: Oracle, PostgreSQL, DB2, ...
- **Developer interface**: OWL ontology development tool, OBDA plugin to manage the mappings and data access
- **End-user interface**: OBDA plugin for Protégé for SPARQL queries and results, and the WONDER system for graphical querying
Querying in the basic OBDAs uses (unions of) conjunctive queries:

- A conjunctive query is the formal counterpart of an SQL (or relational algebra) select-project-join (SPJ) query:
  \[ q(\vec{x}) \leftarrow \exists \vec{y}. \text{conj}(\vec{x}, \vec{y}) \]  
- \( \text{conj}(\vec{x}, \vec{y}) \) is a conjunction of atoms and equalities over the free variables \( \vec{x} \) and the existentially quantified variables \( \vec{y} \)
- The variables in \( \vec{x} \) are the \textit{distinguished variables} (i.e.: appear in the head) and in \( \vec{y} \) are the \textit{non-distinguished variables}
- A union of CQs (UCQ) is a disjunction of CQs, corresponding to a union of SPJ queries

- (U)CQs (in SPARQL notation) embedded into ordinary SQL code is more compact, but still user-unfriendly
Why graphical querying?

- Querying in the basic OBDA uses (unions of) conjunctive queries:
  - A conjunctive query is the formal counterpart of an SQL (or relational algebra) select-project-join (SPJ) query:
    \[ q(\vec{x}) \leftarrow \exists \vec{y}.\text{conj}(\vec{x}, \vec{y}) \]
  - \( \text{conj}(\vec{x}, \vec{y}) \) is a conjunction of atoms and equalities over the free variables \( \vec{x} \) and the existentially quantified variables \( \vec{y} \)
  - The variables in \( \vec{x} \) are the *distinguished variables* (i.e., appear in the head) and in \( \vec{y} \) are the *non-distinguished variables*
  - A union of CQs (UCQ) is a disjunction of CQs, corresponding to a union of SPJ queries

- (U)CQs (in SPARQL notation) embedded into ordinary SQL code is more compact, but still user-unfriendly
Why graphical querying?

- Querying in the basic OBDAs uses (unions of) conjunctive queries:
  - A conjunctive query is the formal counterpart of an SQL (or relational algebra) select-project-join (SPJ) query:
    \[ q(\vec{x}) \leftarrow \exists \vec{y}. \text{conj}(\vec{x}, \vec{y}) \]
  - \( \text{conj}(\vec{x}, \vec{y}) \) is a conjunction of atoms and equalities over the free variables \( \vec{x} \) and the existentially quantified variables \( \vec{y} \)
  - The variables in \( \vec{x} \) are the *distinguished variables* (i.e.: appear in the head) and in \( \vec{y} \) are the *non-distinguished variables*
  - A union of CQs (UCQ) is a disjunction of CQs, corresponding to a union of SPJ queries
- (U)CQs (in SPARQL notation) embedded into ordinary SQL code is more compact, but still user-unfriendly
A use case: the WONDER system

Proposed solution

Sample query

```sql
SELECT stbl.gene
FROM sparqltable
(SELECT $gene $orgName $gcVal $predVal
WHERE {$gene :GeneHasOrganism $org.
$org :OrganismHasOrganismInfo $info.
$info :OrganismName $orgName.
$gene :GeneHasHGTPredictionGene $pred.
$pred :Prediction $predVal.
$gene :GeneHasGCstatsGene $gcstats.
$gcstats :GC3 $gcVal}) stbl
WHERE stbl.orgName LIKE '%Neisseria%' AND
(stbl.predVal = 'hgt' OR stbl.gcVal > '80')
```

*Retrieve all genes of the organisms Neisseria for which horizontal gene transfer is predicted or have a GC3 value > 80*
Proposed solution

Approach to graphical querying

- Accessing information comprises three activities:
  - Browsing the ontology, to understand the structure of the information;
  - Formulating a query, to express an information request; and
  - Retrieving data that answers the query
- The WONDER Web interface consists of a separate component for each of these activities.
Procedure to realise the solution

- Reverse engineer the database into an ORM conceptual data model with the domain experts, cleaning and refining it
- Transform it into a $DL$-lite$_A$ ontology, and put it in Protégé
- Declare the mappings (matching OWL classes and properties to SQL queries over the database)
- Develop the web-based front-end for browsing, query formulation, and displaying the results, using QuOnto for the automated reasoning at the back-end
A use case: the WONDER system

Some technical details

Summary

Realisation of the solution

Section ORM 2 diagram (in NORMA)
A use case: the WONDER system

Realisation of the solution

Example: Diagram – $DL$-lite$_A$ correspondence

$\delta(\text{Abbrev}) \sqsubseteq \text{Organism}$

$\rho(\text{Abbrev}) \sqsubseteq \text{xsd:string}$

$\text{Organism} \sqsubseteq \delta(\text{Abbrev})$

(funct $\text{Abbrev}$)

$\exists \text{GeneIsOnChromosomeOfOrganism} \sqsubseteq \text{Gene}$

$\exists \text{GeneIsOnChromosomeOfOrganism}^- \sqsubseteq \text{Organism}$

$\text{Gene} \sqsubseteq \exists \text{GeneIsOnChromosomeOfOrganism}$

$\text{Organism} \sqsubseteq \exists \text{GeneIsOnChromosomeOfOrganism}^-$

(funct $\text{GeneIsOnChromosomeOfOrganism}$)

... An abbreviation is for an organism
An abbreviation is of type string
Each organism has an abbreviation
Each individual has a single abbreviation
Domain of object property
Range of object property
Each gene belongs to some organism
Each organism has some gene
Each gene belongs to at most one organism

Figure 2: Section of the HGT application ontology.
### Note: semantics of the icons

<table>
<thead>
<tr>
<th>Construct</th>
<th>Graphical Element</th>
<th>Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>![Class Icon]</td>
<td>$C \sqsubseteq T$</td>
</tr>
<tr>
<td>Object Property</td>
<td>![Object Property Icon]</td>
<td>$\exists P \sqsubseteq C$</td>
</tr>
<tr>
<td>Data Property</td>
<td>![Data Property Icon]</td>
<td>$\exists P \neg \sqsubseteq D$</td>
</tr>
<tr>
<td>SubClass Relationship</td>
<td>![SubClass Icon]</td>
<td>$\delta(A) \sqsubseteq C$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\rho(A) \sqsubseteq \top_d$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C \sqsubseteq D$</td>
</tr>
</tbody>
</table>
Realisation of the solution

Example: mapping concepts & relations of the Ontology to SQL query over the relational database

```
SELECT id, abbrev  ~  OrganismHasGene(  
gene(id),  
organism(abbrev))
FROM organism
JOIN genes
ON abbrev = idorganism

SELECT id, kegg  ~  GeneHasGeneFunction(  
gene(id), function(id))  
KEGG(function(id), kegg)
FROM genes
```

Figure 3: Extract of the mapping from the HGT-DB database to the $DL-Lite_A$ application ontology.
Mappings between ontology and data

```
M:0
  o PromiscuousBacterium(getPromBact($abbrev,$ccount,$percentage))
    SELECT organism.abbrev, ccount, organism.percentage
    FROM ( SELECT idorganisme, COUNT(distinct cstart) as ccount
      FROM COMCLUSTG2 GROUP BY idorganisme
    ) flexcount, organism
    WHERE organism.abbrev = flexcount.idorganisme AND
      organism.percentage > 10 AND flexcount.ccount > 5

M:1
  o PromBactPrime(getPromBactPrime($abbrev,$ccount,$percentage,$hgt))
    SELECT organism.abbrev, ccount, organism.percentage, organism.hgt
    FROM ( SELECT idorganisme, COUNT(distinct cstart) as ccount
      FROM COMCLUSTG2 GROUP BY idorganisme
    ) flexcount, organism
    WHERE organism.abbrev = flexcount.idorganisme AND
      organism.percentage > 10 AND flexcount.ccount > 10 AND
      organism.hgt > 150
```
Example: Browsing & selecting
### Realisation of the solution

#### Example: Query pane icons

<table>
<thead>
<tr>
<th>Construct</th>
<th>Graphical Element</th>
<th>Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class node</td>
<td>![C, D]</td>
<td>$C(x), D(x)$</td>
</tr>
<tr>
<td>Object Property link</td>
<td>![C&lt;sup&gt;P&lt;/sup&gt;D]</td>
<td>$C(x), P(x, y), D(y)$</td>
</tr>
<tr>
<td>Data Property node and link</td>
<td>![C&lt;sup&gt;A&lt;/sup&gt;]</td>
<td>$C(x), A(x, y)$</td>
</tr>
</tbody>
</table>
Example: Adding a constraint

Add a Constraint for GCValue

Relational Constraint

GCValue ≥ 60

IN Constraint

GCValue = (OR) GCValue

Between Constraint

<= GCValue <=
Example: Managing constraints

Figure: Constraint manager for getting “Give me the names of the organisms of which the abbreviation starts with a b, but not being a Bacillus, and the prediction and KEGG code of those organisms genes that are putatively either horizontally transferred or highly expressed”
Figure: Query to retrieve the genes of *Neisseria* spp. that have a GC3 content > 80 or are predicted to be horizontally transferred. The textual version of the graphically constructed query (on the right) is generated automatically by the WONDER system.
### A use case: the WONDER system

#### Realisation of the solution

### Example: Section of the results

<table>
<thead>
<tr>
<th>GCValueFirmicutes</th>
<th>geneID</th>
<th>name of the gene</th>
<th>family</th>
</tr>
</thead>
<tbody>
<tr>
<td>'61.0000015258786906'</td>
<td>'boer1_1574'</td>
<td></td>
<td>'no rank:cellular organism; superkingdom:Bacteria; phylum:Firmicutes; class:Bacilli; order:Bacillales; family:Bacillaceae; genus:Bacillus; species group:Bacillus cereus group; species:Bacillus cereus;'</td>
</tr>
<tr>
<td>'63.799999237006055'</td>
<td>'boer1_3144'</td>
<td></td>
<td>'no rank:cellular organism; superkingdom:Bacteria; phylum:Firmicutes; class:Bacilli; order:Bacillales; family:Bacillaceae; genus:Bacillus; species group:Bacillus cereus group; species:Bacillus cereus;'</td>
</tr>
<tr>
<td>'62.799999237060055'</td>
<td>'bcda_469'</td>
<td></td>
<td>'no rank:cellular organism; superkingdom:Bacteria; phylum:Firmicutes; class:Bacilli; order:Bacillales; family:Bacillaceae; genus:Bacillus; species:Bacillus clausii;'</td>
</tr>
</tbody>
</table>
Query and user results

- Domain users have more freedom in constructing the queries and thanks to the query loading/saving feature, the overall service is more usable.
- While using the WONDER interface, domain users came up with new queries that are interesting for their studies.
- The user is aided in the formulation of complex constraints over the queries (Constraint Manager).
- Syntactical correctness of the query is ensured by the formal foundation of the interface.
The overhead caused by the graphical interface is negligible w.r.t. the standard OBDA setting.

The approach is sufficiently scalable with pretty large databases (> 4GB).

Achieved seamless integration of different (Semantic) Web Technologies: OWL 2, AJAX, JavaScript, SVG and XSLT.
A use case: the WONDER system

Builds upon the theory, technology, and implementation developed for Ontology-Based Data Access

Graphical ontology browsing, query formulation, and query execution in a Web browser

Rigorous formal characterisation and uses a coupling with an OWL file, (U)CQs (in SPARQL syntax) and EQL-Lite queries managed by the DIG-QUOnto reasoner

This WONDER system meets the scalability and usability requirements, and allows domain experts to query through a web browser the database without the need to learn SPARQL or EQL-Lite
Summary

1 A use case: the WONDER system
   - Proposed solution
   - Realisation of the solution
   - Summary

2 Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - ‘Impedance’ mismatch
     - Mapping assertions
   - Query answering