COMP60411
Modelling Data on the Web
More error handling & RDF, a graph-based DM

Week 5

Tim Morris
Uli Sattler
University of Manchester
Week 2 coursework

• Most coursework is graded!
  – Q3, SE3, M3
  – CW1
  – CW2, CW2 not yet

• In general,
  – Pay attention to the feedback
    • check the rubrics
    • try to regenerate
    • try rubric on your friend’s essays
  – If you don’t understand
    • read: slides, articles (see materials’ page), other
    • think/draw
    • check & ask on the forum and/or TAs
    • we’re happy to explain further!
  – Remember, you’ll get essays (and MCQs) on the exam
    • Practice and learn now!
    • It will help!
(Technical) Terms & Meaning

- In CS (as a (technical) subject area), people
  - make up & use new **terms**
  - to capture relevant **concepts**
- For people to be able to communicate, we need to
  - agree on the **meaning** of (new) terms…how?
    ➡ We **define** their meaning and agree to use that one, e.g., for
      - self-describing
      - format
      - (core) data model
      - external/internal representation
      - ...
- You need to check whether you use right terms for context
  - always
  - stick to it: repetition is totally ok & necessary
Example term: Robustness

- Related to SE4:

  “which style of query is the "most robust" in the face of such format changes.”


In computer science, robustness is the ability of a computer system to cope with errors during execution and cope with erroneous input.

- How do queries cope/fail/do in the face of such format changes?
  - plain
  - functional
  - typed
Example term: validity

- (Not) being well-formed is a property of (XML) documents
- (Being) being valid is a property between a document and a schema
  - e.g., we can think of a situation where
  - D is valid wrt S1 but
  - D is not valid wrt S2

- Discuss:
  - How does validity relate to precision of data?
  - Does a schema-aware parser fix invalid documents?
  - Can I fix an invalid document?
Formats for ExtRep of data (SE4)

- A **format** consists of
  1. a **core data model** (csv, table, XML, JSON, …)
  2. a **conceptual model**, independent of (1)
  3. **schema(s)** formalising/describing the format
     - documents describing (some aspects of our) design
     - e.g., occupancy.rnc, occupancy.sch,…
  4. the set of conforming ExtReps (e.g., XML documents)
     - concrete embodiments of our design

- (2) the CM can be
  - explicit/tangible (formalised or unformalised) or implicit;
    - written down in a note versus ‘in our head’ or by example
    - ER-Diagram, XSD versus drawing, description in English

- (3) the schemas can be more/less precisely specifying (4)
- (4) the documents are usually implicit
  - you can’t enumerate them all because there are infinitely many
Formats for ExtRep of data (SE4)

• Consider 2 formats $F_1 = <D_{S_1}, C_{M_1}, S_1, D_1>$
  $F_2 = <D_{S_2}, C_{M_2}, S_2, D_2>$

• it may be that
  • $S_1$ only captures some aspects of $D_1$
  • $S_1$ is only a description in English
  • $D_1 = D_2$ but $S_1 \neq S_2$
  • $D_{S_1} = D_{S_2}$ and $C_{M_1} = C_{M_2}$ but $S_1 \neq S_2$ and $D_1 \neq D_2$
  • …and that $F_1$ makes better use of $D_{S_1}$’s features than $D_{S_2}$

• When you design a format, you design each of its aspect and
  – how much you make explicit
  – how you formalise $C_M, S$
Consider this ‘format by example’ for addresses

Discuss: is this a good format for addresses? Does it make good use of JSON’s features?

```json
{
  "person": [
    {
      "ID": 1,
      "first_name": "Zita",
      "last_name": "Speltz",
      "address": "2395 Gloucester Pl",
      "city": "Halliwell Ward",
      "county": "Greater Manchester",
      "postal": "BL1 6DS",
      "email": "wilda@brigham.co.uk",
      "phone1": "01950-109108",
      "phone2": "01300-561046"
    },
    {
      "ID": 2,
      "first_name": "Zachary",
      "last_name": "Freeburger",
      "address": "58 Gloucester Rd",
      "city": "Holbrook",
      "county": "Derbyshire",
      "postal": "DE56 0TX",
      "email": "zachary.freeburger@freeburger.co.uk",
      "phone1": "01888-641397",
      "phone2": "01240-433924"
    }
  ]
}
```
How to Deepen your Understanding

Bloom’s Taxonomy

- Concepts & terms
- Produce new or original work
  Design, assemble, construct, conjecture, develop, formulate, author, investigate
- Justify a stand or decision
  Appraise, argue, defend, judge, select, support, value, critique, weigh
- Draw connections among ideas
  Differentiate, organize, relate, compare, contrast, distinguish, examine, experiment, question, test
- Use information in new situations
  Execute, implement, solve, use, demonstrate, interpret, operate, schedule, sketch
- Explain ideas or concepts
  Classify, describe, discuss, explain, identify, locate, recognize, report, select, translate
- Recall facts and basic concepts
  Define, duplicate, list, memorize, repeat, state

- ...in your project
- Compare - in SEs
- Apply - use in CWs, Ms
- Describe & discuss, make & consider examples
- Read & repeat
How to Deepen your Understanding

**Bloom’s Taxonomy**

- **Remember**: Recite, recollect, recall, relate, recognize, report, state, summarize
- **Understand**: Classify, describe, discuss, explain, identify, locate, recognize, report, select, state
- **Apply**: Use, implement, solve, utilize, demonstrate, interpret, operate, schedule, sketch
- **Analyze**: Differentiate, organize, relate, compare, contrast, distinguish, examine, experiment, question, test
- **Evaluate**: Appraise, argue, defend, judge, select, support, value, critique, weigh
- **Create**: Produce new or original work, design, assemble, construct, conjecture, develop, formulate, author, investigate

- ...in your project
- Compare - in SEs
- Apply - use in CWs, Ms
- Describe & discuss, make & consider examples
- Read & repeat
Error Handling
Postel’s Law

Be liberal in what you accept, and conservative in what you send.

• Liberality
  – Many DOMs, all expressing the same thing
  – Many surface syntaxes (perhaps) for each DOM

• Conservativity
  – What *should* we send?
    • It depends on the receiver!
  – Minimal standards?
    • Well-formed XML?
    • Valid according to a popular schema/format?
    • HTML?
XPath for Validation

- Can we use XPath to determine constraint violations?

```xml
simple.rnc
grammar {
  start = element a { b-descr+ }
  b-descr = element b { empty }
}
```

```xml
valid.xml
<a>
  <b/>
  <b/>
  <b/>
</a>
```

- count(//b) = 3
- count(//b/*) = 0
- count(//b/text()) = 0

- ✔
- ✔
- ✔

```xml
invalid.xml
<a>
  <b/>
  <b>Foo</b>
  <b><b/></b>
</a>
```

- ✔
- ✗
- ✗

- count(//b) = 4
- count(//b/*) = 1
- count(//b/text()) = 1

- ✔
- ✗
- ✗

```xml
valid.xml
<a>
  <b/>
  <b>Foo</b>
</a>
```

- ✔
- ✔
- ✔

- count(//b) = 0
- count(//b/*) = 0
- count(//b/text()) = 0

- ✔
- ✔
- ✔
XPath for Validation

- Can we use XPath to determine constraint violations?

```
grammar {
  start = element a { b-descr+ }
  b-descr = element b { empty }
}
```

**valid.xml**
```
<a>
  <b/>
  <b/>
  <b/>
</a>
```

```
count(//b/(* | text()))
```

-0 = 0

Yes!

**invalid.xml**
```
<a>
  <b/>
  <b>Foo</b>
  <b><b/></b>
</a>
```

```
<valid.xml>
<invalid.xml>

✗ = 1

✗ = 1

✗ = 1

✗ = 2

✗ = 1

✗ = 1

No!

• Can we use XPath to determine constraint violations?
**XPath for Validation**

- Can we use XPath to determine constraint violations?

```rnc
grammar {
start = element a { b-descr+ }
b-descr = element b { empty}
}
```

**valid.xml**
```
<a>
  <b/>
  <b/>
</a>
```

= valid

```xml
if (count(//b/(* | text()))=0)
then “valid”
else “invalid”
```

**invalid.xml**
```
<a>
  <b/>
  <b>Foo</b>
  <b>b/></b>
</a>
```

= invalid

Can even “locate” the errors!
XPath (etc) for Validation

• We could have **finer control**
  – Validate parts of a document
  – A la wildcards
    • But with more control!

• We could have **high expressivity**
  – Far reaching dependencies
  – Computations

• Essentially, **code based validation!**
  – With XQuery and XSLT
  – But still a little declarative

• We always **need it**

  The essence of Schematron
Schematron
Schematron

- A **different** sort of schema language
  - **Rule** based
    - **Not** grammar based or object/type based
  - **Test** oriented
  - **Complimentary** to other schema languages

- Conceptually **simple**: patterns contain rules
  - A rule sets a **context** and contains
    - **asserts** (As) - act “when test is false”
    - **reports** (Rs) - act “when test is true”
  - A&Rs contain
    - a **test** attribute: XPath expressions, and
    - **text content**: natural language description of the error/issue

```xml
<assert test="count(/b/*|text()) = 0">
  Error: b elements must be empty
</assert>

<report test="count(/b/*|text()))!= 0">
  Error: b elements must be empty
</report>
```

**Assert what should be the case!**

**Things that should be reported!**
Schematron by example: for PLists

- “PList has at least 2 person child elements”

```
<pattern>
  <rule context="PList">
    <assert test="count(person) >= 2">
      There has to be at least 2 persons!
    </assert>
  </rule>
</pattern>
```

- equivalently as a “report”:

```
<pattern>
  <rule context="PList">
    <report test="count(person) < 2">
      There has to be at least 2 persons!
    </report>
  </rule>
</pattern>
```

is valid w.r.t. these

```
<PList>
  <person FirstName="Bob" LastName="Builder"/>
  <person FirstName="Bill" LastName="Bolder"/>
  <person FirstName="Bob" LastName="Builder"/>
</PList>
```

is not valid w.r.t. these

```
<PList>
  <person FirstName="Bob" LastName="Builder"/>
</PList>
```

Ok, could handle this with RelaxNG, XSD, DTDs…
Schematron by example: for PLLists

• “Only 1 person with a given name”

```xml
<pattern>
  <rule context="person">
    <let name="F" value="@FirstName"/>
    <let name="L" value="@LastName"/>
    <assert test="count(//person[@FirstName = $F and @LastName = $L]) = 1">
      There can be only one person with a given name,
      but there is <value-of select="$F"/> <value-of select="$L"/> at least twice!
    </assert>
  </rule>
</pattern>
```

above example is not valid w.r.t. these and causes nice error:

...  
Engine name: ISO Schematron  
Severity: error  
Description: There can be only one person with a given name,  
but there is Bob Builder at least twice!

Ok, could handle this with **Keys** in XML Schema!
Schematron by example: for PLists

• “At least 1 person for each family”

```
<pattern>
  <rule context="person">
    <let name="L" value="@LastName"/>
    <report test="count(//family[@name = $L]) = 0">
      There has to be a family for each person mentioned, but <value-of select="$L"/> has none!
    </report>
  </rule>
</pattern>
```

above example is not valid w.r.t. these and causes nice error:

```
<PList>
  <person FirstName="Bob" LastName="Builder"/>
  <person FirstName="Bill" LastName="Bolder"/>
  <person FirstName="Bob" LastName="Milder"/>
  <family name="Builder" town="Manchester"/>
  <family name="Bolder" town="Bolton"/>
</PList>
```
Schematron: informative error messages

If the test condition **true**, the content of the report element is displayed to the user.

<table>
<thead>
<tr>
<th>informative?</th>
<th>yes!</th>
</tr>
</thead>
</table>

The University of Manchester
Tip of the iceberg

- **Computations**
  - Using XPath functions and variables

- **Dynamic checks**
  - Can pull stuff from other file

- **Elaborate reports**
  - diagnostics has (value-of) expressions
  - “Generate paths” to errors
    - Sound familiar?

- **General case**
  - Thin shim over XSLT
  - Closer to “arbitrary code”
Schematron - Interesting Points

- **Friendly**: combine Schematron with WXS, RelaxNG, etc.
  - Schematron is good for that
  - **Two phase validation**
    - RELAX NG has a way of embedding
    - WXS 1.1 incorporating similar rules

- **Powerful**: arbitrary XPath for context and test
  - Plus variables
Schematron - Interesting Points

- **Lenient**: what isn’t forbidden is permitted
  - Unlike all the other schema languages!
  - We’re not performing runs
    - We’re firing rules
  - Somewhat easy to use
    - If you know XPath
    - If you don’t need coverage

- **No traces in PSVI**: a document D either
  - passes all rules in a schema S
    - success -> D is valid w.r.t. S
  - fails some of the rules in S
    - failure -> D is not valid w.r.t. S

- …up to application what to do with D
  - possibly depending on the error messages…think of SE3
Schematron presumes…

• …well formed XML
  – As do all XML schema languages
    • Work on DOM!
  – So can’t help with e.g., overlapping tags
    • Or tag soup in general
    • Namespace Analysis!? 

• …authorial (i.e., human) repair
  – At least, in the default case
    • Communicate errors to people
    • Thus, not the basis of a modern browser!
      – Unlike CSS

• Is this enough liberality?
  – Or rather, does it support enough liberality?
Graph shaped Data Models
- Motivation
Recall: core concepts

• We look at **data models**,  
  • shape: none, tables, trees, graphs,…

• and **core DMs** for the above  
  – [tables] csv files, SQL tables  
  – [trees] sets of feature-value pairs, XML, JSON  
  – [graphs] RDF

• and **schema languages** for the above  
  – [SQL tables] SQL  
  – [XML] RelaxNG, XSD, Schematron,…  
  – [JSON] JSON Schema

• and **manipulation mechanisms**  
  – [SQL tables] SQL  
  – [XML] DOM, SAX, XQuery,…  
  – [JSON] JSON API,…
Recall: core concepts

• Each Data Model was motivated by
  – representational needs of some domain and
  – pain points
    • Fundamental Pain Points
      – Mismatch between the domain and the data structure
    • Tech-specific Pain Points
      – XPath Limitations

• Alleviating pain
  – Try to squish it in
    • E.g., encoding trees in SQL
    • E.g., layering
  – Polyglot persistence
    • Use multiple data models

• Either way
  – It’s important to understand the pain
  – And trade offs between different coping strategies
Domains we have discussed

- People, addresses, personal data
  - with(out) management structure
- SwissProt protein data
- Cartoons
- Arithmetic expressions
  - [CW1] easy, binary expressions with students, attempts, etc.
  - [CW2, CW3] nested expressions of varying parity
From Flat File to Relational (1)

- **Domain**: People, addresses, personal data
- **Pain Points** in 1 (flat) csv file:
  - *variable numbers* of the "same" attribute
    - phone number
    - email address
    - …
  - inserting columns is painful
    - lots of partial columns
  - companies have addresses
    - more than one!
    - and phone numbers, etc.
From Flat File to Relational (2)

- **Domain**: People, addresses, personal data

- **Better Format**:
  - in 2 (flat) csv files

- **Pain Points**:
  - sorting destroys the relationship
    - we used row numbers to connect
    - sorting changes the row number!
  - hard to see the record
  - no longer a flat file
  - CSV format makes assumptions
Use Relational Model for this Domain

• M1

• Design a conceptual model for this domain
  – normalise it
  – create different tables for suitable aspects of this domain
  – linked via “foreign keys” offered by relational formalism

► no more pain points:
  • this domain fits nicely our “table” relational data model (RDM)
  • RDM also comes with a suitable
    • data manipulation language for
      • querying
      • sorting
      • inserting tuples
    • schema language
      • constraining values
      • expressing functional/key constraints
From Relational to JSON & XML (1)

• **Domain**: People, addresses, management structure

• **Pain points** in relational/SQL tables:
  - cumbersome: too many joins (1 per management level)!
  - (nigh) impossible: ensuring integrity - unbounded ‘manages’ paths require **recursive** queries/joins to avoid cyclic management structure
  - …but fits nicely into XML or JSON
  - if management tree = employees tree

<table>
<thead>
<tr>
<th>Employees</th>
<th>Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee ID</td>
<td>Postcode</td>
</tr>
<tr>
<td>1234123</td>
<td>M16 0P2</td>
</tr>
<tr>
<td>1234124</td>
<td>M2 3OZ</td>
</tr>
<tr>
<td>1234567</td>
<td>SW1 A</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
From Relational to JSON & XML (2)

- **Domain**: Proteins
- **Pain points** in relational/SQL tables:
  - cumbersome:
    - querying: too many tables/joins!
  - …but fits nicely into XML or JSON
  - see Uniprot exports!

<table>
<thead>
<tr>
<th>Protein ID</th>
<th>Full Name</th>
<th>Short Name</th>
<th>Organism</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234123</td>
<td>Fanconi anemia group J</td>
<td>FAC J</td>
<td>Halorubrum</td>
<td>...</td>
</tr>
<tr>
<td>1234567</td>
<td>ATP-dependent RNA helicase BRIP1</td>
<td>N/A</td>
<td>Gallus gallus</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein ID</th>
<th>Alternative Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234123</td>
<td>ATP-dependent RNA helicase BRIP1</td>
</tr>
<tr>
<td>1234123</td>
<td>BRCA1-interacting protein C-terminal helicase 1</td>
</tr>
<tr>
<td>1234123</td>
<td>BRCA1-interacting protein 1</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein</th>
<th>Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234123</td>
<td>BRIP1</td>
</tr>
<tr>
<td>1234123</td>
<td>BACH1</td>
</tr>
<tr>
<td>1234567</td>
<td>helicas</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
From Relational to JSON & XML (3)

• **Domain:** Arithmetic expressions
  • e.g, \(((3 \times 4) + 6 + 6)\)

• **Pain points:**
  – cumbersome:
    • querying: too many tables/joins!
  – impossible: how to write these?!
  – …but fits nicely into XML or JSON
  – see our coursework!

### Direct Subexpression

<table>
<thead>
<tr>
<th>Expression</th>
<th>HasSubExpression</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>124</td>
</tr>
<tr>
<td>123</td>
<td>712</td>
</tr>
<tr>
<td>123</td>
<td>712</td>
</tr>
<tr>
<td>124</td>
<td>715</td>
</tr>
<tr>
<td>124</td>
<td>716</td>
</tr>
</tbody>
</table>

### Direct Subexpression

<table>
<thead>
<tr>
<th>Expression ID</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Plus</td>
</tr>
<tr>
<td>124</td>
<td>Times</td>
</tr>
<tr>
<td>125</td>
<td>Minus</td>
</tr>
</tbody>
</table>

### Atoms

<table>
<thead>
<tr>
<th>Atom ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>712</td>
<td>6</td>
</tr>
<tr>
<td>713</td>
<td>9</td>
</tr>
<tr>
<td>714</td>
<td>12</td>
</tr>
<tr>
<td>715</td>
<td>3</td>
</tr>
<tr>
<td>716</td>
<td>4</td>
</tr>
</tbody>
</table>
New Domains

• with new requirements:

• Sociality
  – friend-of/knows/likes/acquainted-with/trusts/…
  – works-with/colleague-of/…
  – interacts-with/reacts-with/binds-to/activates/…
  – student-of/fan-of/…
  – …
  – such relationships form social/professional/bio-chemical/academic networks
  – we focus on social here: knows

• How are they different to “manages”

• How do we capture these?
“Knows” in SQL - ER Diagram

simple!
“Knows” in SQL tables

CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);

CREATE TABLE knows
(
Who int,
Whom int,
FOREIGN KEY (Who)
    REFERENCES Persons(P_Id),
FOREIGN KEY (Whom)
    REFERENCES Persons(P_Id)
);
“Knows” in SQL - Queries (1)

CREATE TABLE Persons
(
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);

CREATE TABLE knows
(
    Who int,
    Whom int,
    FOREIGN KEY (Who)
        REFERENCES Persons(P_Id),
    FOREIGN KEY (Whom)
        REFERENCES Persons(P_Id)
);

How many friends does Bob Builder have?

SELECT COUNT(DISTINCT k.Whom)
FROM Persons P, knows k
WHERE ( P.PersonID = k.Who AND
    P.FirstName = "Bob" AND
    P.LastName = "Builder" );
“Knows” in SQL - Queries (2)

CREATE TABLE Persons
(
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);

CREATE TABLE knows
(
  Who int,
  Whom int,
  FOREIGN KEY (Who)
    REFERENCES Persons(P_Id),
  FOREIGN KEY (Whom)
    REFERENCES Persons(P_Id)
);

Give me the names of Bob Builder’s friends?

SELECT P2.FirstName , P2.LastName
FROM knows k, Persons P1, Persons P2
WHERE ( P1.PersonID = k.Who AND
        P2.PersonID = k.Whom AND
        P1.FirstName = “Bob” AND
        P1.LastName = “Builder” );
“Knows” in SQL - Queries (3)

```
CREATE TABLE Persons
(
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);

CREATE TABLE knows
(
    Who int,
    Whom int,
    FOREIGN KEY (Who)
        REFERENCES Persons(P_Id),
    FOREIGN KEY (Whom)
        REFERENCES Persons(P_Id)
);
```

Give me the names of Bob Builder’s friends’ friends?

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE (  k1.whom = k2.who AND
          P1.PersonID = k1.Who AND
          P3.PersonID = k2.Whom AND
          P1.FirstName = "Bob" AND
          P1.LastName = "Builder" );
```
### “Knows” in SQL - Queries (4)

**CREATE TABLE** Persons

```sql
CREATE TABLE Persons
(  
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

**CREATE TABLE** knows

```sql
CREATE TABLE knows
(  
Who int,
Whom int,
FOREIGN KEY (Who)
  REFERENCES Persons(P_Id),
FOREIGN KEY (Whom)
  REFERENCES Persons(P_Id)
);
```

**SELECT** P3.FirstName, P3.LastName

```sql
SELECT P3.FirstName , P3.LastName 
FROM knows k1, knows k2, knows k3,….Persons P1, Persons P3 
WHERE (  
  (k1.whom = k2.who OR k1.whom = P3.PersonID) AND 
  (k2.whom = k3.whom OR  k2.Whom = P3.PersonID) AND 
  ….. 
  P1.FirstName = “Bob” AND 
P1.LastName = “Builder” 
);
```
“Knows” in SQL - Pain Points

• Fundamental Pain Points:
  – variable number of “relationships” -> split tables/normalise
    ➞ queries require joins
    ➞ performance may deteriorate & queries become error prone
  – domain may require unbounded joins
    • to explore a network of friends/paths of unbounded depth
    • requires recursive queries or bounds on domain structure/depth

• Technology Specific Pain Points:
  • does your SQL DBMS support
    • recursive queries?
    • transitive closure?
  – if yes: fine
  – if not: we can’t query whole, unbounded networks!
“Knows” in XML

- Let’s use the Same Conceptual Model

- And let’s follow the SQL for the logical model/schema!
Knowings XSD
Example Document

```xml
<knowings>
  <people>
    <person id="1">
      <FirstName>Bob</FirstName>
      <LastName>Builder</LastName>
      <Address>Somewhere Cool</Address>
      <City>Manchester</City>
    </person>
    <person id="2">
      <FirstName>Wendy</FirstName>
      <Address>88 Jackson Crescent</Address>
      <City>Manchester</City>
    </person>
  </people>
  <knows>
    <who personref="1"/>
    <whom personref="2"/>
  </knows>
</knowings>
```
Counting Friends!

How many friends does Bob Builder have?

```sql
SELECT COUNT(DISTINCT k.Whom)  
FROM Persons P, knows k  
WHERE ( P.PersonID = k.Who AND  
  P.FirstName = "Bob" AND  
  P.LastName = "Builder" );
```

```xml
count(  
  //whom  
  [../who/@personref =  
    //person[FirstName="Bob"  
    and LastName="Builder"]/@id])
```
Get those friends!

Give me the names of Bob Builder’s friends?

```
SELECT P2.FirstName, P2.LastName
FROM knows k, Persons P1, Persons P2
WHERE ( P1.PersonID = k.Who AND
        P2.PersonID = k.Whom AND
        P1.FirstName = "Bob" AND
        P1.LastName = "Builder" );
```

Get the whole person

```xml
//@id =
//@personref =

[../who//@personref =
//@person[FirstName="Bob"
and LastName="Builder"]//@id]//@personref
```
Get those friends!

Give me the names of Bob Builder’s friends?

```
SELECT P2.FirstName, P2.LastName
FROM knows k, Persons P1, Persons P2
WHERE (P1.PersonID = k.Who AND
       P2.PersonID = k.Whom AND
       P1.FirstName = "Bob" AND
       P1.LastName = "Builder");
```

Bit of XQuery to get the names

```
for $p in //person[@id = //whom
    [./who/@personref = //person[FirstName="Bob"
        and LastName="Builder"]/@id]/@personref
   ]
return <name>{$p/FirstName} {$p/LastName}</name>
```
Get those friends!

Function it up a bit

```xml
declare function local:friendsOf($person) { 
    for $p in $person/../person[@id = //whom[../who/@personref = $person/@id]/@personref]
    return $p
}
```

```xml
declare function local:fullNameOf($person) { 
    <name>{$person/FirstName} {Proof}{Proof}$person/LastName</name>
}
```

```xml
for $f in local:friendsOf('//person[FirstName="Bob" and LastName="Builder"]')
    return local:fullNameOf($f)
```
All friends of friends

Give me the names of friends of friends of Bob Builder!

```
SELECT P3.FirstName, P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE (k1.whom = k2.who AND
       P1.PersonID = k1.Who AND
       P3.PersonID = k2.Whom AND
       P1.FirstName = "Bob" AND
       P1.LastName = "Builder" );
```

See next slide!
All friends of friends in Network

```perl
declare function local:friendsOf($person) {
    for $p in $person/../$person[@id = //whom
        [../who/@personref = $person/@id]/@personref]
    return $p
}

declare function local:friendsOfFriend($person) {
    for $p in local:friendsOf($person)
    return
        if (empty($p))
            then $p (: done :)
        else (local:friendOf($p))
}

declare function local:fullNameOf($person) {
    <name>{$person/FirstName} {rgba:120:80:120} {$person/LastName}</name>
}

for $f in local:friendsOfFriend(///person[FirstName="Bob"
    and LastName="Builder"])
return local:fullNameOf($f)
```
All friends in Network

Give me the names of people in Bob Builder’s network?

```sql
SELECT P3.FirstName, P3.LastName
FROM knows k1, knows k2, knows k3, ....Persons P1, Persons P3
WHERE (
    (k1.whom = k2.who OR k1.whom = P3.PersonID) AND
    (k2.whom = k3.whom OR k2.Whom = P3.PersonID) AND
    ....
    P1.FirstName = "Bob" AND
    P1.LastName = "Builder" );
```

See next slide!
All friends in Network

```xml
declare function local:friendsOf($person) {
    for $p in $person/../person[@id = //whom
        ./../who/@personref = $person/@id]/@personref]
    return $p
}

declare function local:friendTreeOf($person) {
    for $p in local:friendsOf($person)
    return
        if (empty($p))
            then $p (: Base case of the recursion! :) )
        else ($p, local:friendTreeOf($p))
};

declare function local:fullNameOf($person) {
    <name>{$person/FirstName} {$person/LastName}</name>
};

for $f in local:friendTreeOf(//person[FirstName="Bob"
    and LastName="Builder"])
return local:fullNameOf($f)
```
Is this robust?

• What if we have:
  – Bob knows Wendy
  – Wendy knows Farmer Pickles
  – Farmer Pickles knows Bob?
Cycles Cause Problems

• We now have to implement **cycle detection**
  – And perhaps some other stuff!?

• **New pain points**
  – Identity of node through 1 relation was tough
    • Managing the IDs, personrefs, etc. was...unpleasant
    • If we add other sorts of nodes, could get tediouser
      – Key and Keyref were themselves a touch challenging!
  – Tree like sets were ok, but cycles are hard
    • This will be true for formats like “GraphML”!
Let’s re-evaluate our format

```
<knowings>
  <people>
    <person id="1">
      <FirstName>Bob</FirstName>
      <LastName>Builder</LastName>
      <Address>Somewhere Cool</Address>
      <City>Manchester</City>
    </person>
    <person id="2">
      <FirstName>Wendy</FirstName>
      <Address>88 Jackson Crescent</Address>
      <City>Manchester</City>
    </person>
  </people>

  <knows>
    <who personref="1"/>
    <whom personref="2"/>
  </knows>

```

“Knowings”? Really?

Why People but “knows” as direct child?

None of these issues touch the data structure mismatch problem

Couldn’t we just embed who each person knows in that element?
Graph shaped Data Models

- Graph Basics
“Knows” forms a Graph

Linkedin Maps: Egon Willighagen's Professional Network as of January 25, 2011
Graph Basics

• A graph $G = (V,E)$ is a pair with
  – $V$ a set of vertices (also called) nodes, and
  – $E \subseteq V \times V$ a set of edges

• Example: $G = (\{a,b,c,d\}, \{(a,b), (b,c), (b,d), (c,d)\})$
  – where are $a, \ldots, d$ in this graph’s picture?

• Variants:
  – (in)finite graphs: $V$ is a (in)finite set
  – (un)directed graphs: $E$ (is) is not a symmetric relation
    • i.e., if $G$ is undirected, then $(x,y) \in E$ implies $(y,x) \in E.$
  – node/edge labelled graphs: a label set $S$, labelling function(s)
    • $L: V \rightarrow S$ (node labels)
    • $L: E \rightarrow S$ (edge labels)
Graph Basics (2)

• Example: node-labelled graph
  – \( L: V \rightarrow \{A,P\} \)

• Example: edge-labelled graph
  – \( L: E \rightarrow \{p,r,s\} \)

• Example: node-and-edge-labelled graph
  – \( L: V \rightarrow \{A,P\} \)
  – \( L: E \rightarrow \{p,r,s\} \)
Graph Basics: External Representation

- **Pictures** are a BAD external representation for graphs

\[
G = (\{a,b,c,d\}, \\
\{(a,b), (b,c), (b,d), (b,c)\}, \\
L: V \to \{A,P\} \\
L: a \mapsto A, b \mapsto P, c \mapsto A, d \mapsto A)
\]
Graph Basics: External Representation

- **Pictures** are a BAD **external representation** for graphs
  - it captures loads of irrelevant information
    - colour
    - location, geometry,
    - shapes, strokes, …
  - what if labels are more complex/structured?
  - how do we parse a picture into an **internal representation**?
RDF
- a data structure formalisms for graphs
A Graph Formalism: RDF

- **Resource Description Framework**
- a **graph-based** data structure formalism
- a W3C standard for the representation of **graphs**
- comes with various syntaxes for ExtRep
- is based on **triples**

(subject, predicate, object)
RDF: basics

• an RDF graph $G$ is a set of triples
  $\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$

• where each
  • $s_i \in U \cup B$
  • $p_i \in U$
  • $o_i \in U \cup B \cup L$

U: URIs (for resources), incl. rdf:type
B: Blank nodes
L: Literals

(subject, predicate, object)
RDF: an example

- an RDF **graph** $G$ is a **set** of **triples**
  $$\{(s_i, p_i, o_i) | 1 \leq i \leq n\}$$
- where each
  - $s_i \in U \cup B$, $p_i \in U$, $o_i \in U \cup B \cup L$

```plaintext
{(ex:bparsia, foaf:knows, ex:bparsia/),
 (ex:bparsia, rdf:type, foaf:Person),
 (ex:bparsia, rdf:type, Agent),
 (ex:sattler, foaf:title, "Dr."),
 (ex:bparsia, foaf:title, "Dr.")},
 (ex:sattler, foaf:knows, ex:alvaro),
 (ex:bparsia, foaf:knows, ex:alvaro) }
```

Abbreviate: ex: for http://www.cs.man.ac.uk/
foaf: for http://xmlns.com/foaf/0.1/

U: URIs (for resources)
B: Blank nodes
L: Literals
RDF: an example (2)

- an RDF graph $G$ is a set of triples
  $$\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$$
- where each
  - $s_i \in U \cup B$, $p_i \in U$, $o_i \in U \cup B \cup L$

Abbreviate: ex: for http://www.cs.man.ac.uk/
foaf: for http://xmlns.com/foaf/0.1/
RDF syntaxes

• “serialisation formats”
  – for ExtRep of RDF graphs
• there are various:
  – Turtle
  – N-Triples
  – JSON-LD
  – N3
  – RDF/XML
  – …
• plus translators between them
  - e.g. www.easyrdf.org/converter

5 triples in Turtle:

```rml
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:sattler
  foaf:title "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows
    [ foaf:title "Count";
    foaf:lastName "Dracula"
  ] .
```

{(ex:bparsia, foaf:knows, ex:bparsia/),
 (ex:bparsia, rdf:type, foaf:Person),
 ...}
RDF syntaxes - Turtle & JSON-LD

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://www.cs.man.ac.uk/> .

ex:sattler
    foaf:title "Dr." ;
    foaf:knows ex:bparsia ;
    foaf:knows
        [ foaf:knows
            [ foaf:title "Count";
                foaf:lastName "Dracula"
            ]
        ].

[{
    "@id": "_:b0",
    "http://xmlns.com/foaf/0.1/title": [
        {
            "@value": "Count"
        }
    ],
    "http://xmlns.com/foaf/0.1/lastName": [
        {
            "@value": "Dracula"
        }
    ],
    {
        "@id": "http://www.cs.man.ac.uk/bparsia",
        {
            "@id": "http://www.cs.man.ac.uk/sattler",
            "http://xmlns.com/foaf/0.1/title": [
                {
                    "@value": "Dr."
                }
            ],
            "http://xmlns.com/foaf/0.1/lastName": [
                {
                    "@id": "http://www.cs.man.ac.uk/bparsia"
                },
                {
                    "@id": "_:b0"
                }
            ]
        }
    }
}]

ex:bparsia
    foaf:knows
        [ foaf:title "Dr.";
            foaf:lastName "Dracula"
        ].

_x
    foaf:knows
        [ foaf:title "Count";
            foaf:lastName "Dracula"
        ].

ex:sattler
    foaf:knows
        [ foaf:title "Dr.";
            foaf:lastName "Dracula"
        ].

Dr.

ex:bparsia

Count

Dracula
RDFS

a schema language for RDF

- and an unusual schema language!
RDFS: A different sort of schema

- In RDF, we have `rdf:type`
- **RDFS** is a **schema language** for RDF
- In RDFS, we also have
  - `rdfs:subClassOf`
    - e.g. `(foaf:Person, rdfs:subClassOf, foaf:Agent)`
    - `(ex:Woman, rdfs:subClassOf, foaf:Person)`
  - `rdfs:subPropertyOf`
    - e.g. `(ex:hasDaughter, rdfs:subPropertyOf, ex:hasChild)`
  - `rdfs:domain`
    - e.g. `(ex:hasChild, rdfs:domain, foaf:Person)`
    - `(foaf:currentProject, rdfs:domain, foaf:Person)`
  - `rdfs:range`
    - e.g. `(ex:hasChild, rdfs:range, foaf:Person)`
    - `(foaf:currentProject, rdfs:range, foaf:Project)`
**Inference: Default Values++**

- **RDFS does not describe/constrain structure**
  - That is, unlike XML style schema languages, RDFS can’t be used to “validate” documents/graphs
    - at least easily
    - The primary goal of RDFS is *adding extra information*
    - Sorta like default values!

```xml
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:sattler
  foaf:title  "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows [ foaf:title "Count";
                foaf:lastName "Dracula"
              ].
@

@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:person rdfs:subClassOf foaf:Agent
+

@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:sattler rdf:type foaf:Person.
ex:sattler rdf:type foaf:Agent
ex:bparsia rdf:type foaf:Person.
ex:bparsia rdf:type foaf:Agent
```

=>
What do schemas usually do again?

• So far, we’ve met schemas that describe ExtReps:
  – what’s allowed
  – what’s required
  – what’s assumed
    • default values
  – what’s expected
  – what’s forbidden

• In RDFS, we can only state
  – what’s assumed/known, and thus
  – what can be inferred
    • here:  
      ex:bparsia rdf:type foaf:Person.
      ex:alvaro rdf:type foaf:Person.
      foaf:knows rdfs:domain foaf:Person.
      foaf:knows rdfs:range foaf:Person

ex:bparsia

foaf:knows

ex:alvaro
For more inference...

• ...we cordially invite you to take our course from the Ontology Engineering and Automated Reasoning theme:
  – COMP62342 Ontology Engineering for the Semantic Web
  – COMP60332 Automated Reasoning and Verification
SPARQL
a query language for graphs
SPARQL

• We have
  – A data structure: graphs!
  – A data definition language (sort of...RDFS)
    • Plus loads of external representations (turtle, N3, N-triples, JSON-LD,..)
  – Manipulation: you can use
    • rdflib in Python
    • a fine query & manipulation language:

• SPARQL
  – Standardised query language for RDF
    • Not the only graph query language out there!
    • E.g., neo4j has it’s own language “Cypher”
      – http://neo4j.com/developer/cypher/
      – has “graph structural” features like “shortest path”
      – lacks “unbounded path” queries
Basic Graph Patterns

• Any set of Turtle statements can be part of a SPARQL query
  – e.g. {ex:sattler rdf:type foaf:Person}
  – (We put it in braces here!)

• We can replace URIs, bNodes, or Literals with variables
  – e.g., {?x rdf:type foaf:Person}

• Arbitrary sets!
SPARQL Clauses (1)

- We combine a BGP with a **query type**
  - ASK
    - E.g., ASK WHERE {ex:sattler rdf:type foaf:Person}
    - Returns true or false (only)
  - SELECT
    - E.g., SELECT ?p WHERE {?p rdf:type foaf:Person}
    - Very much like SQL select
  - Note
    - Ask returns a boolean (not an RDF graph!)
    - SELECT returns a table (not an RDF graph!)
    - SPARQL is *not* closed over graphs!
      - unusual: compare to SQL and XQuery!
SPARQL Clauses (2)

• There are two query types that return graphs:
  – CONSTRUCT
    • E.g., CONSTRUCT {?p rdf:type :Befriended}
      » WHERE {?p foaf:knows ?q}
    • Like XQuery element and attribute constructors
  – DESCRIBE
    • E.g., DESCRIBE ?p WHERE {?p rdf:type foaf:Person}
    • Implementation dependent!
    • A “description” (as a graph)
      – Whatever the service deems helpful!
      – A bit akin to querying system tables in SQL
Example Data

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://www.cs.man.ac.uk/> .

ex:bobthebuilder
  foaf:firstName "Bob";
  foaf:lastName "Builder";
  foaf:knows ex:wendy ;
  foaf:knows ex:farmerpickles;
  foaf:knows ex:bijanparsia.

ex:wendy
  foaf:firstName "wendy";
  foaf:knows ex:farmerpickles.

ex:farmerpickles
  foaf:firstName "Farmer";
  foaf:lastName "Pickles";
  foaf:knows ex:bobthebuilder.

ex:bijanparsia
  foaf:firstName "Bijan";
  foaf:lastName "Parsia".
Counting Friends!

How many friends does Bob Builder have?

```sql
SELECT COUNT(DISTINCT COUNT (?friend))
WHERE {ex:bobthebuilder
  foaf:firstName "Bob"
  foaf:lastName "Builder"
  foaf:knows ?friend };
```

```
SELECT COUNT(DISTINCT k.Whom)
FROM Persons P, knows k
WHERE ( P.PersonID = k.Who AND
  P.FirstName = "Bob" AND
  P.LastName = "Builder" );
```

See Page 42: Our SQL example

This is your first SPARQL query
Finding Friends’ Friends?

Give me Bob Builder’s friends’ friends?

```sql
SELECT P3.FirstName, P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE (k1.whom = k2.who AND
       P1.PersonID = k1.Who AND
       P3.PersonID = k2.Whom AND
       P1.FirstName = "Bob" AND
       P1.LastName = "Builder" );
```

See Page 43:
Another SQL example

```sparql
SELECT ?first, ?last
WHERE {ex:bobthebuilder
         foaf:firstName "Bob";
         foaf:lastName "Builder";
         foaf:knows ?x.
         ?y foaf:firstName ?first;
         foaf:lastName ?last}
```

Your second SPARQL query
Friends network?

Give me everybody in Bob Builder’s friends’ friends…?

```
SELECT P3.FirstName, P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE ( k1.whom = k2.who AND
    P1.PersonID = k1.Who AND
    P3.PersonID = k2.Whom AND
    aaaaaaaaaaargh );
```

See Page 44: no SQL example!

```
SELECT ?first, ?last
WHERE {ex:bobthebuilder
    foaf:firstName "Bob";
    foaf:lastName "Builder";
    foaf:knows+ ?friend.
    ?friend foaf:firstName ?first;
    foaf:lastName ?last}
```

Your third SPARQL query
SPARQL and Inference

- **SPARQL queries are sensitive to RDF(S) inference**
  - The way XPath is sensitive to default values!
  - Also sensitive to more expressive language’s inferences
    - Like OWL!
      - In OWL, we can say that foaf:knows is *transitive*
      - So we don’t necessarily need the property path to make our queries!

- **Inference has a cost**
  - May be surprising
  - May be computationally expensive!
Solves all problems?

- No!
  - We have to filter out Bob
    - Because he will be in the cyclic paths
    - Foo!
      - But pretty easy with a FILTER
  - But pretty reasonable
    - Path expressions help a lot!

- Fairly normalised
  - We don’t get nice pre-assembled chunks like with XML

- No validation!
  - This is a formalism specific quirk
  - Work is being done
Retrospective
&
Pulling it all together

Work in groups on
2 Questions
Poly-

• How can we vary?
  – Same data model, same formalism, same implementation
    • But different domain models!
  – Same data model, same formalism, same domain model
    • Different implementations, e.g., SQLite vs. MySQL
  – Same data model, same domain model
    • Different formalisms!
      – Usually, but not always, implies different implementations
      – XML in RDBMS

• We can be explicitly or implicitly poly-
  – If we encode another data model into our home model
    • We are still poly-
    • But only implicitly so
    • Key Cost: Ad hoc implementation
  – If we split our domain model across multiple formalisms/implementations
    • We are explicitly poly
    • Key Cost: Model and System integration
Key point

• Understand your domain
  – What are you trying to represent and manipulate

• Understand your use case
  – including (frequent, relevant) queries, error sources,…

• Understand the fit between domain and data model(s)
  – To see where there are sufficiently good fits

• Understand your infrastructure
  – And the cost of extending

• Understand integration vs. workaround costs

• Then make a reasonable decision
  – There will always be tradeoffs
Question 1

Consider again the Conceptual Model you started to work on last week: can you
• finish/improve/extend it?
• add adjectives?
• add examples?

- format
- formalism
- core data model
- data model
- database
- external repr.
- ...

- domain model
- schema
- schema language
- application
- system
- internal repr.
- ...

- robust
- extensible
- scalable
- self-describing
- valid
- expressive
- verbose
- ...
Question 2

Consider a format for a reporting system for health & safety incidents, as exemplified by the printed example document:

• sketch a system for
  • gathering this data
  • reporting it monthly
• which kind of schema(s) would you use to describe it?
  • why?
• does this format make good use of XML’s features?
• how could you improve these?
Good Bye!

- We hope you have learned a lot!
- It was a pleasure to work with you!
- Speak to us about projects
  - taster/MRes
  - MSc
- Enjoy the rest of your programme
  - COMP62421 query processing
  - COMP62342 rich modelling, inference, semantic web, symbolic AI