MashQL
A Data Mashup Language for the Data Web

Dr. Mustafa Jarrar
mjarrar@cs.ucy.ac.cy
HPCLab, University of Cyprus

Published As:

University of Cyprus © 2009
Can we query and mash up the Data Web as simple as filtering and piping Web Feeds?

- We fundamentally investigate this problem from a Query Formulation viewpoint.

- A “data mashup” is a query.
Outline

- Challenges
- Related Work
- MashQL (A Query Formulation Language)
- Evaluation
- Conclusions and Discussion
Challenges

I don’t know the schema! (Black-box query)

Problem Definition

We need a way to allow end-users formulate queries over structured data assuming that:

- The user does not know the schema. (1)
Challenges

Does not adhere to schema or ontology!

Problem Definition

How to allow end-users formulate queries over structured data assuming that:

- The user does not know the schema. \( (1) \)
- There is no offline or inline schema\ontology. \( (2) \)
Challenges

Allow me to easily Compose what I need!

Problem Definition

How to allow end-users formulate queries over structured data assuming that:

- The user does not know the schema. (1)
- There is no offline or inline schema/ontology. (2)
- The query may involve multiple sources. (3)
Challenges

General Solution!

How to allow end-users formulate queries over structured data assuming that:

1. The user does not know the schema.
2. There is no offline or inline schema\ontology.
3. The query may involve multiple sources.
4. The query language is sufficiently expressive (not a single-purpose interface)
### Related Work

**Query formulation** is the art of accessing and consuming structured data *easily*. (interdisciplinary subject)

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Query by Form</th>
<th>Query by Example</th>
<th>Conceptual Queries (ConQuer)</th>
<th>NL Queries</th>
<th>Interactive Queries (Lorel)</th>
<th>Visual Scripting (DeriPipes)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Don't know the schema</em></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><em>Schema-free data</em></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td><em>Multiple sources</em></td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><em>Expressive</em></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td><em>Intuitive</em></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

University of Cyprus © 2009
MashQL

A graphical query formulation Language.

all of these assumptions:
- The user does not know the schema.
- There is no offline or inline schema\ontology.
- The query may involve multiple sources.
- The query language is expressive.
  (not a single-purpose interface)
MashQL

A general structured-data retrieval solution. (not merely an interface)

Without loosing this generality, we

- Focus on RDF, as the most primitive query language, other data models can be mapped into it.

- Follow Yahoo Pipes’ visualization, in order to illustrate that Data Web can be queried and mashed up as Web Feeds.
Example 1

“Lara’s articles after 2007?”

- Interactive Query Formulation.
- MashQL queries are translated into and executed as SPARQL.
Example 1

“Lara’s articles after 2007?”

RDF Input

From:
- http://www.site1.com/rdf
- http://www.site2.com/rdf

Query

Everything

University of Cyprus © 2009
Example 1

“Lara’s articles after 2007?”

RDF Input
- From: http://www.site1.com/rdf
- http://www.site2.com/rdf

Query
- Everything
  - A1 rdf:Type bibo:Article
  - A1 :Title “Data Web”
  - A1 :Author “Tom Lara”
  - A1 :Author “Bob Hacker”
  - A1 :Year 2007

- A2 rdf:Type bibo:Article
  - A2 :Title “Semantic Web”
  - A2 :Author “Tom Lara”
  - A2 :Year 2005

Background queries
- SELECT X WHERE {?X ?P ?O}
- Union
- SELECT X WHERE {?S ?P ?X}

SELECT X WHERE {?S rdf:Type ?X}
Example 1

“Lara’s articles after 2007?”

RDF Input

From:
- http://www.site1.com/rdf
- http://www.site2.com/rdf

Query

Everything

<table>
<thead>
<tr>
<th>Type</th>
<th>Author</th>
<th>Title</th>
<th>PubYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lara</td>
<td>Lara</td>
<td>Tom</td>
<td>2007</td>
</tr>
<tr>
<td>Lara</td>
<td>Lara</td>
<td>Tom</td>
<td>2005</td>
</tr>
</tbody>
</table>

Background query

```
SELECT P WHERE {?Everything ?P ?O}
```
Example 1

“Lara’s articles after 2007?”

RDF Input

Query

SELECT P WHERE {?Everything ?P ?O}
Example 1

“Lara’s articles after 2007?”

RDF Input

From:
- http://www.site1.com/rdf
- http://www.site2.com/rdf

Query

Everything
- Author "^Lara"
- Year\PubYear > 2007

Background query

SELECT P WHERE {?Everything ?P ?O}
Example 1

"Lara’s articles after 2007?"

RDF Input
From:
http://www.site1.com/rdf
http://www.site2.com/rdf

Query
Everything
○ Author  “^Lara
○ Year\PubYear  > 2007
○ Title  ✓ArticleTitle

PREFIX S1: <http://site1.com/rdf>
PREFIX S2: <http://site1.com/rdf>
SELECT ?ArticleTitle
FROM <http://site1.com/rdf>
FROM <http://site2.com/rdf>
WHERE {
  {?X S1:Author ?X1} UNION {?X S2:Author ?X1}
  {?X S1:PubYear ?X2} UNION {?X S2:Year ?X2}
  {(?X S1:Title ?ArticleTitle) UNION (?X S2:Title }
  {?ArticleTitle))
  FILTER regex(?X1, “^Hacker”) 
  FILTER (?X2 > 2000)}
Example 2

“Recent articles from Cyprus?”

Query

Article
- Title  
- Title
- Author
- Address
- Address
- Country “Cyprus”
- Year > 2008

→ Retrieve every Article that: has a title, written by author, who has address, this address has a country called Cyprus, and the article is published after 2008.
The Intuition of MashQL

A query is a tree

• The root is called the query *subject*.

• Each branch is a *restriction*.

• Branches can be expanded, (information path)

• Object value filters

Def. A **Query** $Q$ with a subject $S$, denoted by $Q(S)$, is a set of restrictions on $S$. $Q(S) = R_1 \text{ AND ... AND } R_n$.

Def. A **Subject** $S \in (I \cup V)$, where $I$ is an identifier and $V$ is a variable.

Def. A **Restriction** $R = \langle R_x, P, O_f \rangle$, where $R_x$ is an optional restriction prefix that can be (maybe | without), $P$ is a predicate ($P \in I \cup V$), and $O_f$ is an object filter.

University of Cyprus © 2009
The Intuition of MashQL

An Object filter is one of:
- Equals
- Contains
- MoreThan
- LessThan
- Between
- one of
- Not(f)
- Information Path (sub query)

Def. An object filter \( O_f = <O, f> \), where \( O \) is an object and \( f \) is a filtering function one of:

\( O_f = <O, \text{Equals}(X, T, Lt)> \), where \( X \) can be a variable or a constant, \( T \) is a datatype, and \( Lt \) is a language tag.

\( O_f = <O, \text{Contains}(X, T, Lt)> \), where \( O \) is an object variable, \( X \) is a regex literal, \( T \) is a data type, and \( Lt \) is a language.

\( O_f = <O, \text{MoreThan}(X, T)> \), where \( O \) is an object variable, \( X \) is a variable or a constant, \( T \) is a datatype.

\( O_f = <O, \text{LessThan}(X, T)> \), where \( O \) is an object variable, \( X \) is a variable or a constant, \( T \) is a datatype identifier.

\( O_f = <O, \text{Between}(X, Y, T)> \), where \( X \) and \( Y \) are variables or constants, \( T \) is a datatype identifier.

\( O_f = <O, \text{OneOf}(V)> \), where \( O \) is an object variable, and \( V \) is a set of values \( \{v, v_2, \ldots, v_n\} \), \( v \) is a variable or constant.

\( O_f = <O, \text{Not}(f)> \), where \( f \) is one of the functions defined above.

\( O_f = <O, Q_i(O)> \), where \( O \) is an object \( (O \in V \cup I) \), and \( Q_i(O) \) is a sub-query with \( O \) being the query subject.
More MashQL Constructs

- Resection Operators \{Required, Maybe, or Without\}
  
  All restriction are required (i.e. AND), unless they are prefixed with “maybe” or “without”

```sql
SELECT ?SongTitle, AlbumTitle
WHERE {
{?X :Title ?SongTitle.
?X :Artist :Shakera
Optional{?X :Album ?AlbumTitle}
Optional{?X :Copyright ?X2}
Filter !Bound(?X2).
```
More MashQL Constructs

- **Union operator (denoted as “\”) between**

- **Objects**

  ```sql
  SELECT ?Person
  WHERE {
    ?Person :WorkFor :Google
    UNION
    ?Person WorkFor :Yahoo
  }
  ```

- **Predicates**

  ```sql
  SELECT ?FName
  WHERE {
    ?Person :Surname ?FName
    UNION
    ?Person :Firstname ?FName
  }
  ```

- **Subjects**

  ```sql
  SELECT ?AgentName, ?AgentPhone
  WHERE {
    {?Person rdf:type :Person.
     ?Person :Name ?AgentName.
     ?Person :Phone ?AgentPhone}
    UNION
    {?Company rdf:type :Company.
     ?Company :Name ?AgentName.
     ?Company :Phone ?AgentPhone}
  }
  ```

- **Queries**

  ```sql
  SELECT ?CustName,
  WHERE {
    {?Person :Name ?CustName.
     UNION
     {?Company :Title ?CustName.
      FILTER regex(?X1, "Paris")}}
  ```
More MashQL Constructs

And several other constructs, including:

- Types
- and Reverse Predicates
- Datatypes and Language Tags
- ....
Formal Syntax and Semantics

Def. 1 (Dataset): A dataset D is a set of triples, each triple t is formed as <S, P, O>, where S ∈ I, P ∈ I, and O ∈ I ∪ L.

Def. 2 (Typed Literals): Every object literal must have a datatype D: If O ∈ L then O ∈ D.

Def. 3 (Language Tags): An object literal (O = <O Equals(X T L t) may have a language tag L t.

Def. 4 (Query): A query Q with a subject S, denoted by Q(S), is a set of restrictions on S. Q(S) = R_1 AND ... AND R_n.

Def. 5 (Subject): A subject S ∈ (I ∪ V), where I is an identifier and V is a variable.

Def. 6 (Restriction): A restriction R = <R_x, P, Of>, where Rx is an optional restriction prefix that can be (maybe | without), P is a predicate (P ∈ I ∪ V), and O_i is an object.

Def. 7 (Object Filter): An object filter O_f = <O, f>, where O is an object and f is a filtering function. An object filter can have one of the following nine forms:

1. O_f = <O>, where O is an object, O ∈ V. This is the simplest object filter, i.e., it does not add any restriction on the object value of the retrieved triples.

2. O_f = <O, Equals(X, T, L_t)>, where X can be a variable or a constant, T is a datatype, and L_t is a language tag. This filter restricts the retrieved results, such that the object value O should be equal to X, with datatype T, and with language L_t.

3. O_f = <O, Contains(X, T, L_t)>, where O is an object variable, X is a regex literal, T is a data type, and L_t is a language. This filter restricts the retrieved results, such that the object value O should be equal to regex(X), with datatype T, and with language L_t. A regex literal is a literal that contains a regular expression matching pattern.

4. O_f = <O, MoreThan(X, T)>, where O is an object variable, X is a variable or a constant, T is a datatype. This filter restricts the retrieved results, such that the object value O should be more than X and with datatype T.

5. O_f = <O, LessThan(X, T)>, where O is an object variable, X is a variable or a constant, T is a datatype. This filter restricts the retrieved results, such that the object value O should be less than X and with datatype T (see rule-9).

6. O_f = <O, Between(X, Y, T)>, where X and Y are variables or constants, T is a datatype identifier. This filter restricts the retrieved results, such that the object value O should be more than or equals X, less than or equals Y, and with datatype T.

7. O_f = <O, OneOf(V)>, where O is an object variable, and V is a set of values {v_1, ... , v_n}, v_i is a variable or constant. This filter restricts the retrieved results, such that the object value O should be equal to one of the values in V.

8. O_f = <O, Not(f)>, where f is one of the functions defined above. This filter extends all of the above functions with simple negation. The filter is same as the Equals filter but with negation, i.e., Not Equal.

9. O_f = <O, Q_i(O)>, where O is an object (O ∈ V ∪ I), and Q_i(O) is a sub-query with O being the query subject. The restrictions defined in the sub-query Q_i(O) should be satisfied as well. Notice that this definition is recursive; however, this does not mean the query itself is recursive.

Def. 8 (Types): A subject (S ∈ I) or an object (O ∈ I) can be prefixed with “a” or “an” to mean the instances of this subject/object type, instead of the subject/object itself.

Def. 9 (Union): A union can be declared between objects, predicates, subjects and/or queries, in the following forms:

1. O_i = <O_1 | O_2 | ... | O_n>, to indicate unions between objects, where O_i ∈ I.

2. P_i = <P_1 | P_2 | ... | P_n>, to indicate unions between predicates, where P_i ∈ I.

3. S_i = <S_1 | S_2 | ... | S_n>, to indicate unions between subjects, where S_i ∈ I.

4. O_i = <O_1 | O_2 | ... | O_n>, to indicate unions between queries.

Def. 10 (Reverse): <~P> indicates the reverse of the predicate P. Let R_1 be a restriction on S such that <S P O>, and R_2 be <O ~P S>, R_1 and R_2 have the same meaning.
Query Formulation Algorithm

Formalization of the background queries

Select Subject S

(1) \( S \in S_T : \pi_O(\sigma_{P=\text{Type}}(D)) \)
(1') \( O_1 : \{ (?S \text{ Type} \ ?O1) \} \)

(2) \( S \in S_I : \pi_S(D) \cup \pi_O(\sigma_{O \in}(D)) \)
(2') \( S_1 : \{ (?S \ ?P1 \ ?O1) \} \) UNION \( O_1 : \{ (?S \ ?P1 \ ?O1) \}. \) Filter isURI(?O1)

(3) \( S \in V \)

Select a property P

(4) \( (S \in S_T) \rightarrow P \in \pi_{P} (\sigma_{P\text{Type} \land O1=Subject}(D) \times S_1=S_2 \sigma(D)) \)
(4') \( P_2 : \{ (?S \text{ Type} \ ?S) \} \)

(5) \( (S \in S_I) \rightarrow P \in \pi_P (\sigma_{S=Subject}(D)) \)
(5') \( P_1 : \{ (<S> \ ?P1 \ ?O1) \} \)

(6) \( (S \in V) \rightarrow P \in \pi_P (\sigma(D)) \)
(6') \( P_1 : \{ (?S \ ?P1 \ ?O1) \} \)

Add a filter on P

(7) \( P \in V \)

(8) \( (S \in S_I) \land (P \in V) \rightarrow O \in \pi_{O1} (\sigma_{S1=S \land O1 \in}(D)) \)
(8') \( O_1 : \{ (?S1 \ ?P1 \ ?O1) \} \) Filter isURI(?O1)

(9) \( (S \in S_I) \land (P \not\in V) \rightarrow O \in \pi_{O1} (\sigma_{S1=S \land P1=P \land O1 \in}(D)) \)
(9') \( O_1 : \{ (<S> \ ?P1 \ ?O1) \} \) Filter isURI(?O1)

(10) \( (S \in S_T) \land (P \in V) \rightarrow O \in \pi_{O2} (\sigma_{P1=\text{Type} \land O1=S(D) \times S1=S2 \sigma(D)) \)
(10') \( O_1 : \{ (<S \text{ Type} \ ?S) \} \)

(11) \( (S \in S_T) \land (P \not\in V) \rightarrow O \in \pi_{O2} (\sigma_{P1=rdf:Type \land O1=S(D) \times S1=S2 \sigma(D)} \)
(11') \( O_1 : \{ (?S \text{ rdf:Type} \ ?S) \} \)

(12) \( (S \in V) \land (P \in V) \rightarrow O \in \pi_O (\sigma(D)) \)
(12') \( O_1 : \{ (?S1 \ ?P1 \ ?O1) \} \)

(13) \( (S \in V) \land (P \not\in V) \rightarrow O \in \pi_O (\sigma_{P}(D)) \)
(13') \( O_1 : \{ (?S1 \ ?P1 \ ?O1) \} \)
MashQL-SPARQL Mapping Rules

Rule-1: The symbol \( \exists \) before a variable means that it will be returned in the results; i.e., included in the SELECT part of in SPARQL. If the output of the query is input to another, use “CONSTRUCT *”.

Rule-2: In any of the following rules, if a subject, predicate, or object is italicized: it is seen as a SPARQL variable, i.e. prefixed with “?”. If T \( \neq \) Null: Append the mapping with: \( \text{FILTER} (?O = T) \)

Rule-3: If S is a subject and R = < , P, O>, the mapping is: \{S P O\}.

Rule-4: If S is a subject and R = <maybe, P, O>, the mapping is: \{OPTIONAL(S P O)\}.

Rule-5: If S is a subject and R = < without, P, O>, the mapping is: \{S P O. \text{FILTER (!bound(?O))}\}.

Rule 6. If Of = \(<O, \text{Equals}(X, T, Lt)>\):
- Append the mapping with: \( \text{FILTER} (?O = X) \)
- If T \( \neq \) Null: Append the mapping with: \( \text{FILTER} (\text{datatype} (?O)=T) \)
- If Lt \( \neq \) Null: Append the mapping with: \( \text{FILTER} (\text{lang} (?O) = Lt) \)

Rule 7. If Of = \(<O, \text{Cor}(X, Y, T)>\):
- Append the mapping with: \( \text{FILTER} (?O < X) \)
- If T \( \neq \) Null: Append the mapping with: \( \text{FILTER} (\text{datatype} (?O)=T) \)

Rule 8. If Of = \(<O, \text{Between}(X, Y, T)>\):
- Append the mapping with: \( \text{FILTER} (?O \geq X) \&\& \text{FILTER} (?O \leq Y) \)
- If T \( \neq \) Null: Append the mapping with: \( \text{FILTER} (\text{datatype} (?O)=T) \)

Rule 9. If Of = \(<O, \text{OneOf}(V)>\):
- Append the mapping with: \{\text{FILTER} (?O = V1)\} \&\& \ldots \&\& \text{FILTER} (?O = Vn)\}
- If Vi is a regex-ed literal, the ith filter above should be replaced with: \( \text{FILTER} \text{Regex}(?O, \text{Vi}) \)

Rule 12. If Of = \(<O, \text{Not}(f)>\): The f filter will be generated as above, but with a negation.

Rule 13. If Of = \(<O, \text{Qi}(O)>\): Repeat all mapping rules to generate \( \text{Qi} (\text{O}) \).

Rule 14. If a subject S is prefixed with “a” or “an”: Append the mapping with: \{?S rdf:type :S\}

Rule 15. If an object O is prefixed with “a” or “an”: Append the mapping with: \{?O rdf:type :O\}

Rule 16. Given On , If n >1 and Oi \( \in \) I: The mapping in rules 3-4 will be:{\{S P :O1\} UNION \ldots UNION \{S P :On\}}

Rule 17. Given Pn , If n >1 and Pi \( \in \) I: The mapping in rules 3-4 will be: {\{S :P1 O\} UNION \ldots UNION \{S :Pn O\}}

Rule 18. Given Sn , If n >1 and Si \( \in \) I: Regenerate the query n times, each time with Si as a root, and with a UNION between the queries.

Rule 19. Given Qn , If n >1 : Add UNION between the n queries.

Rule 20. If S is a subject and R = \(<-P, O>\), the mapping is: \{O P S\}.
MashQL Editor

- Alpha version (will be public soon)
- Web Ajax-based.
- Open sources Java Script libraries (from Yahoo)
- Oracle 11g as RDF store.
- **Graphs Summaries** for fast user-interaction.
  - URI Normalization based on heuristics.
  - (but some URIs are too cryptic)
MashQL Firefox Add-On (Light-mashups @ your browser)
Evaluation (DBLP, Experiment 1)

(User Interaction Response Time)
How long it takes to generate the next list?

Query

Any Article

Title "World-Wide Web"

Creator

Type Person

Name "Berners-Lee"

Year > 2007

<table>
<thead>
<tr>
<th>Query</th>
<th>DBLP (9M triples)</th>
<th>DBLP (4M triples)</th>
<th>DBLP (2M triples)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GS</td>
<td>Oracle</td>
<td>GS</td>
</tr>
<tr>
<td>Q₁</td>
<td>0.003</td>
<td>0.005</td>
<td>0.003</td>
</tr>
<tr>
<td>Q₂</td>
<td>0.001</td>
<td>0.136</td>
<td>0.001</td>
</tr>
<tr>
<td>Q₃</td>
<td>0.001</td>
<td>0.187</td>
<td>0.001</td>
</tr>
<tr>
<td>Q₄</td>
<td>0.001</td>
<td>1.208</td>
<td>0.001</td>
</tr>
</tbody>
</table>

University of Cyprus © 2009
Evaluation (DBPedia, Experiment 2)

P: (?S :Type :Album)
(?S :PreviousAlbum ?O1)
(?O1 ?P ?O2)

<table>
<thead>
<tr>
<th>Query</th>
<th>DBPedia (32 M)</th>
<th>DBPedia (16 M)</th>
<th>DBPedia (8 M)</th>
<th>DBPedia (4 M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>0.003</td>
<td>0.003</td>
<td>0.003</td>
<td>0.003</td>
</tr>
<tr>
<td>Q₂</td>
<td>0.002</td>
<td>0.002</td>
<td>0.002</td>
<td>0.002</td>
</tr>
<tr>
<td>Q₃</td>
<td>0.005</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
</tr>
<tr>
<td>Q₄</td>
<td>0.005</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
</tr>
<tr>
<td>Q₅</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>Q₆</td>
<td>0.007</td>
<td>0.007</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>Q₇</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>Q₈</td>
<td>0.007</td>
<td>0.007</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>Q₉</td>
<td>0.007</td>
<td>0.007</td>
<td>0.007</td>
<td>0.006</td>
</tr>
</tbody>
</table>
### Evaluation (DBPedia, Experiment 3)

#### Query

<table>
<thead>
<tr>
<th>Query</th>
<th>(B32) 32 M GS</th>
<th>Oracle</th>
<th>DBPedia (16 M) GS</th>
<th>Oracle</th>
<th>DBPedia (8 M) GS</th>
<th>Oracle</th>
<th>DBPedia (4 M) GS</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.001</td>
<td>0.027</td>
<td>0.001</td>
<td></td>
<td>0.001</td>
<td></td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>0.026</td>
<td>17.450</td>
<td>0.010</td>
<td></td>
<td>0.011</td>
<td></td>
<td>0.010</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>0.048</td>
<td>49.656</td>
<td>0.010</td>
<td></td>
<td>0.017</td>
<td></td>
<td>0.011</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>0.087</td>
<td>5348.7</td>
<td>0.022</td>
<td></td>
<td>0.017</td>
<td></td>
<td>0.011</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>0.135</td>
<td></td>
<td>0.036</td>
<td></td>
<td>0.032</td>
<td></td>
<td>0.011</td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>0.185</td>
<td></td>
<td>0.055</td>
<td></td>
<td>0.047</td>
<td></td>
<td>0.016</td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td>0.234</td>
<td></td>
<td>0.076</td>
<td></td>
<td>0.061</td>
<td></td>
<td>0.023</td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td>0.265</td>
<td></td>
<td>0.098</td>
<td></td>
<td>0.077</td>
<td></td>
<td>0.027</td>
<td></td>
</tr>
<tr>
<td>Q9</td>
<td>0.280</td>
<td></td>
<td>0.110</td>
<td></td>
<td>0.092</td>
<td></td>
<td>0.034</td>
<td></td>
</tr>
</tbody>
</table>

### SPARQL Query

1. P3: (?S Type ?Album)
2. (?S RelatedTo1 ?O1)
3. (?O1 RelatedTo2 ?O2)
4. (?O2 RelatedTo3 ?O3)
5. (?O3 RelatedTo4 ?O4)
6. (?O4 RelatedTo5 ?O5)
7. (?O5 RelatedTo6 ?O6)
8. (?O6 RelatedTo7 ?O7)
9. (?O7 RelatedTo8 ?O8)
10. (?O8 RelatedTo9 ?O9)

University of Cyprus © 2009
Conclusions

- "Query and mash up the Data Web as simple as filtering up Web Feeds" is a query formulation problem.

- End-users can navigate, query, and mash up unknown graphs.
  without knowing the schema. Data is schema-free. Multiple sources.

- MashQL is expressive as SPARQL.
  Except NAMED GRAPH.

- MashQL is not merely a SPARQL interface, or limited RDF.
  It has its own path-pattern intuition (can be similarly used for XML and DB).
Future Work

- Reasoning, Keyword Search, Aggregation Functions, etc.
- Results Presentation (Should be tacked fundamentally).
- Firefox add-on Mashup/query editor.
- RDF summaries for SPARQL optimization.
- ...
Thank You

Dr. Mustafa Jarrar
mjarrar@cs.ucy.ac.cy
HPCLab, University of Cyprus