

# XML Retrieval

# Overview

---

- 1 Introduction
- 2 Basic XML concepts
- 3 Challenges in XML IR
- 4 Vector space model for XML IR
- 5 Evaluation of XML IR

# IR and relational databases

IR systems are often contrasted with relational databases (RDB).

- Traditionally, IR systems retrieve information from *unstructured text* (“raw” text without markup).
- RDB systems are used for querying *relational data*: sets of records that have values for predefined attributes such as employee number, title and salary.

	RDB search	unstructured IR
objects	records	unstructured docs
main data structure	table	inverted index
model	relational model	vector space & others
queries	SQL	free text queries

Some structured data sources containing text are best modeled as structured documents rather than relational data (Structured retrieval).

# Structured retrieval

---

Basic setting: queries are structured or unstructured; documents are structured.

## Applications of structured retrieval

Digital libraries, patent databases, blogs, tagged text with entities like persons and locations (named entity tagging)

## Example

- Digital libraries: *give me a full-length article on fast fourier transforms*
- Patents: *give me patents whose claims mention RSA public key encryption and that cite US patent 4,405,829*
- Entity-tagged text: *give me articles about sightseeing tours of the Vatican and the Coliseum*

# Why RDB is not suitable in this case

---

Three main problems

- 1 An unranked system (DB) would return a potentially large number of articles that mention the Vatican, the Coliseum and sightseeing tours without ranking them by relevance to query.
- 2 Difficult for users to precisely state structural constraints – may not know which structured elements are supported by the system.  
*tours AND (COUNTRY: Vatican OR LANDMARK: Coliseum)?*  
*tours AND (STATE: Vatican OR BUILDING: Coliseum)?*
- 3 Users may be completely unfamiliar with structured search and advanced search interfaces or unwilling to use them.

Solution: adapt ranked retrieval to structured documents to address these problems.

# Structured Retrieval

## RDB search, Unstructured IR, Structured IR

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured docs	trees with text at leaves
main data structure	table	inverted index	?
model	relational model	vector space & others	?
queries	SQL	free text queries	?

Standard for encoding structured documents: Extensible Markup Language (XML)

- structured IR → XML IR
- also applicable to other types of markup (HTML, SGML, ...)

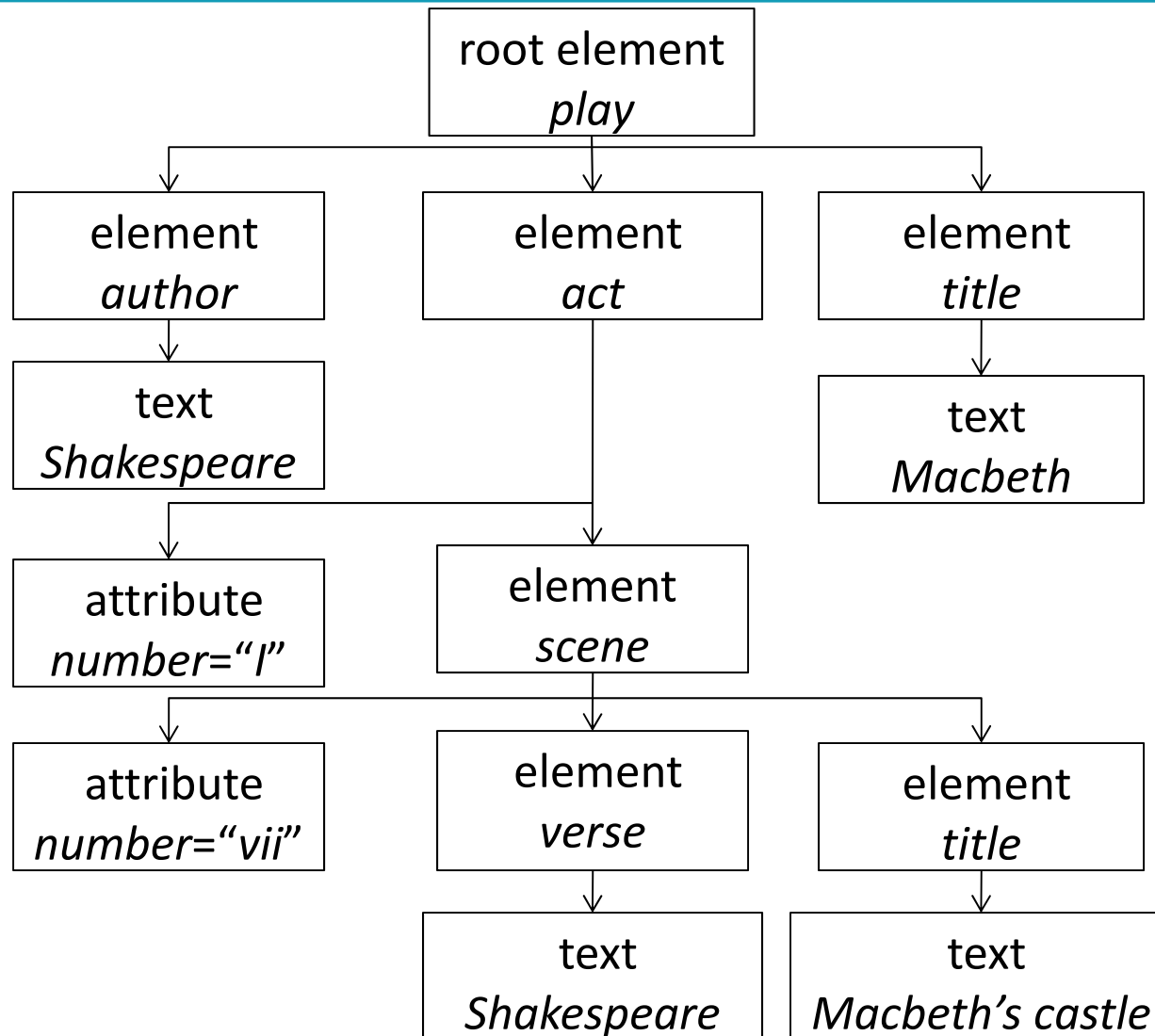
# XML document

---

- Ordered, labeled tree
- Each node of the tree is an XML element, written with an opening and closing XML tag (e.g. `<title...>`, `</title...>`)
- An element can have one or more XML attributes (e.g. `number`)
- Attributes can have values (e.g. `vii`)
- Attributes can have child elements (e.g. `title`, `verse`)

```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="1">
<scene number=""vii">
<title>Macbeth's castle</title>
<verse>Will I with wine
...</verse>
</scene>
</act>
</play>
```

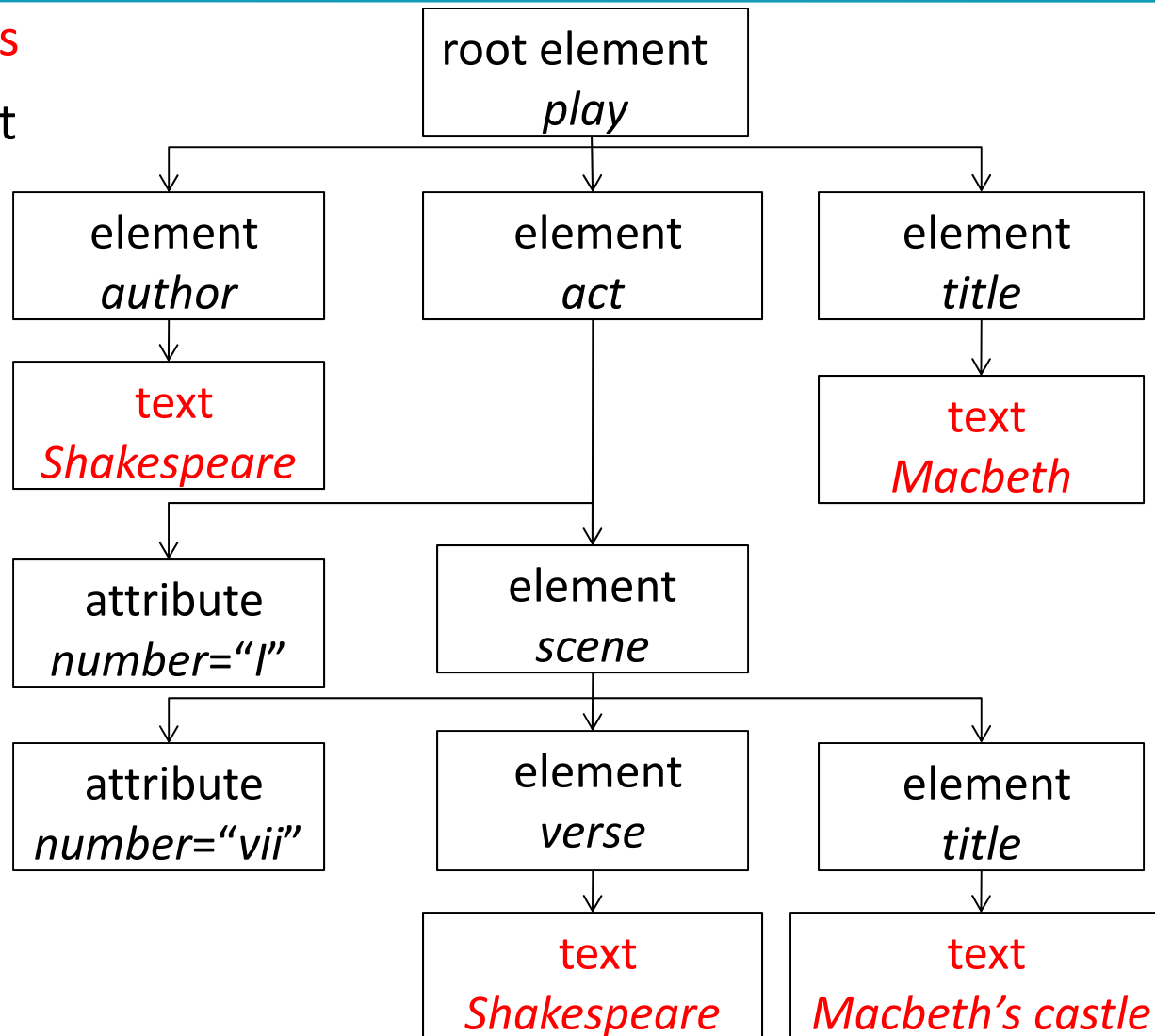
# XML document





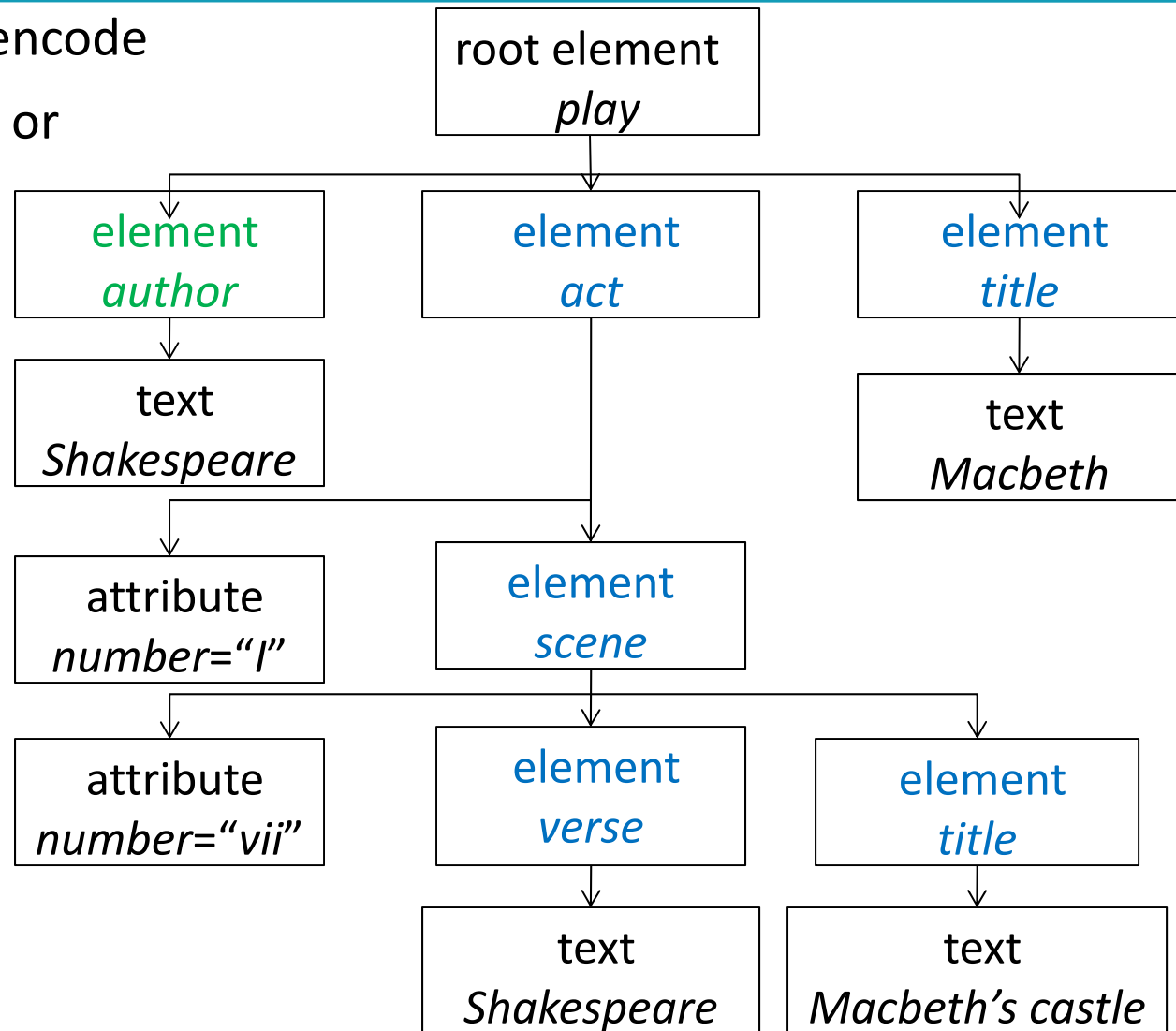
# XML document

The **leaf nodes**  
consist of text



# XML document

The internal nodes encode  
document structure or  
metadata functions



# XML basics

---

- **XML Documents Object Model (XML DOM):** standard for accessing and processing XML documents
  - The DOM represents elements, attributes and text within elements as nodes in a tree.
  - With a DOM API, we can process an XML documents by starting at the root element and then descending down the tree from parents to children.
- **XPath:** standard for enumerating path in an XML document collection.
  - We will also refer to paths as XML contexts or simply contexts
- **Schema:** puts constraints on the structure of allowable XML documents. E.g. a schema for Shakespeare's plays: scenes can occur as children of acts.
  - Two standards for schemas for XML documents are: XML DTD (document type definition) and XML Schema.

## First challenge: document parts to retrieve

---

Structured or XML retrieval: users want us to return parts of documents (i.e., XML elements), not entire documents as IR systems usually do in unstructured retrieval.

### Example

If we query Shakespeare's plays for *Macbeth's castle*, should we return the scene, the act or the entire play?

- In this case, the user is probably looking for the scene.
- However, an otherwise unspecified search for *Macbeth* should return the play of this name, not a subunit.

Solution: structured document retrieval principle

# Structured document retrieval principle

## Structured document retrieval principle

One criterion for selecting the most appropriate part of a document:

*A system should always retrieve the most specific part of a document answering the query.*

- Motivates a retrieval strategy that returns the smallest unit that contains the information sought, but does not go below this level.
- Hard to implement this principle algorithmically. E.g. query: title:*Macbeth* can match both the title of the tragedy, *Macbeth*, and the title of Act I, Scene vii, *Macbeth's castle*.
  - But in this case, the title of the tragedy (higher node) is preferred.
  - Difficult to decide which level of the tree satisfies the query.

## Second challenge: document parts to index

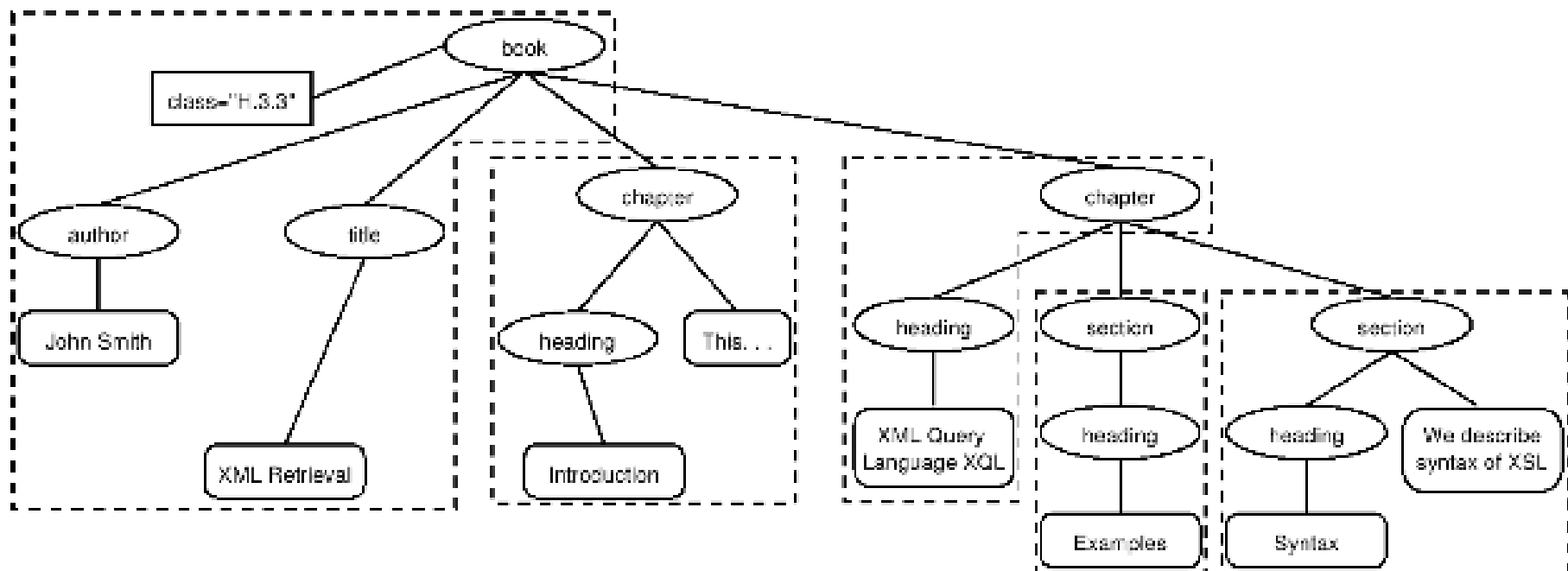
---

Central notion for indexing and ranking in IR: documents unit or **indexing unit**.

- In unstructured retrieval, usually straightforward: files on your desktop, email messages, web pages on the web etc.
- In structured retrieval, there are four main different approaches to defining the indexing unit
  - ① non-overlapping pseudodocuments
  - ② top down
  - ③ bottom up
  - ④ all

# XML indexing unit: approach 1

Group nodes into non-overlapping pseudodocuments.



Indexing units: books, chapters, section, but without overlap.

Disadvantage: pseudodocuments may not make sense to the user because they are not coherent units.

## XML indexing unit: approach 2

---

Top down (2-stage process):

- 1 Start with one of the latest elements as the indexing unit, e.g. the book element in a collection of books
- 2 Then, postprocess search results to find for each book the subelement that is the best hit.

This two-stage retrieval process often fails to return the best subelement because the relevance of a whole book is often not a good predictor of the relevance of small subelements within it.



## XML indexing unit: approach 3

---

Bottom up:

Instead of retrieving large units and identifying subelements (top down), we can search all leaves, select the most relevant ones and then extend them to larger units in postprocessing.

Similar problem as top down: the relevance of a leaf element is often not a good predictor of the relevance of elements it is contained in.

## XML indexing unit: approach 4

Index all elements: the least restrictive approach. Also problematic:

- Many XML elements are not meaningful search results, e.g., an ISBN number.
- Indexing all elements means that search results will be highly redundant.

### Example

For the query *Macbeth's castle* we would return all of the *play*, *act*, *scene* and *title* elements on the path between the root node and *Macbeth's castle*. The leaf node would then occur 4 times in the result set: 1 directly and 3 as part of other elements.

We call elements that are contained within each other **nested elements**. Returning redundant nested elements in a list of returned hits is not very user-friendly.

## Third challenge: nested elements

---

Because of the redundancy caused by the nested elements it is common to restrict the set of elements eligible for retrieval.

Restriction strategies include:

- discard all small elements
- discard all element types that users do not look at (working XML retrieval system logs)
- discard all element types that assessors generally do not judge to be relevant (if relevance assessments are available)
- only keep element types that a system designer or librarian has deemed to be useful search results

In most of these approaches, result sets will still contain nested elements.

## Third challenge: nested elements

---

Further techniques:

- remove nested elements in a postprocessing step to reduce redundancy.
- collapse several nested elements in the results list and use highlighting of query terms to draw the user's attention to the relevant passages.

### Highlighting

- Gain 1: enables users to scan medium-sized elements (e.g., a section); thus, if the section and the paragraph both occur in the results list, it is sufficient to show the section.
- Gain 2: paragraphs are presented in-context (i.e., their embedding section). This context may be helpful in interpreting the paragraph.

## Nested elements and term statistics

Further challenge related to nesting: we may need to distinguish different contexts of a term when we compute term statistics for ranking, in particular inverse document frequency (idf).

### Example

The term *Gates* under the node *author* is unrelated to an occurrence under a content node like *section* if used to refer to the plural of *gate*. It makes little sense to compute a single document frequency for *Gates* in this example.

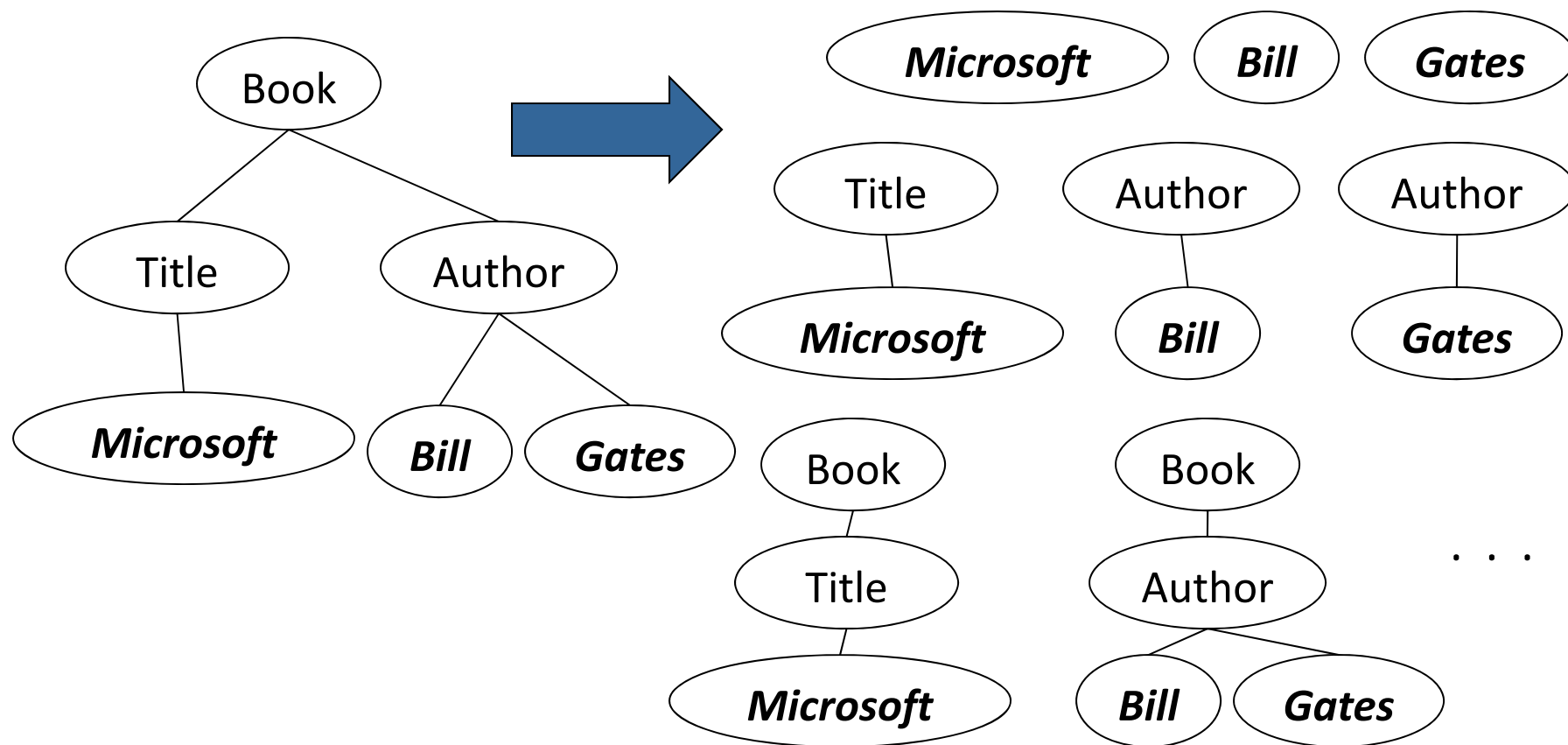
Solution: compute idf for XML-context term pairs.

- sparse data problems (many XML-context pairs occur too rarely to reliably estimate df)
- compromise: consider the parent node  $x$  of the term and not the rest of the path from the root to  $x$  to distinguish contexts.

# Main idea: lexicalized subtrees

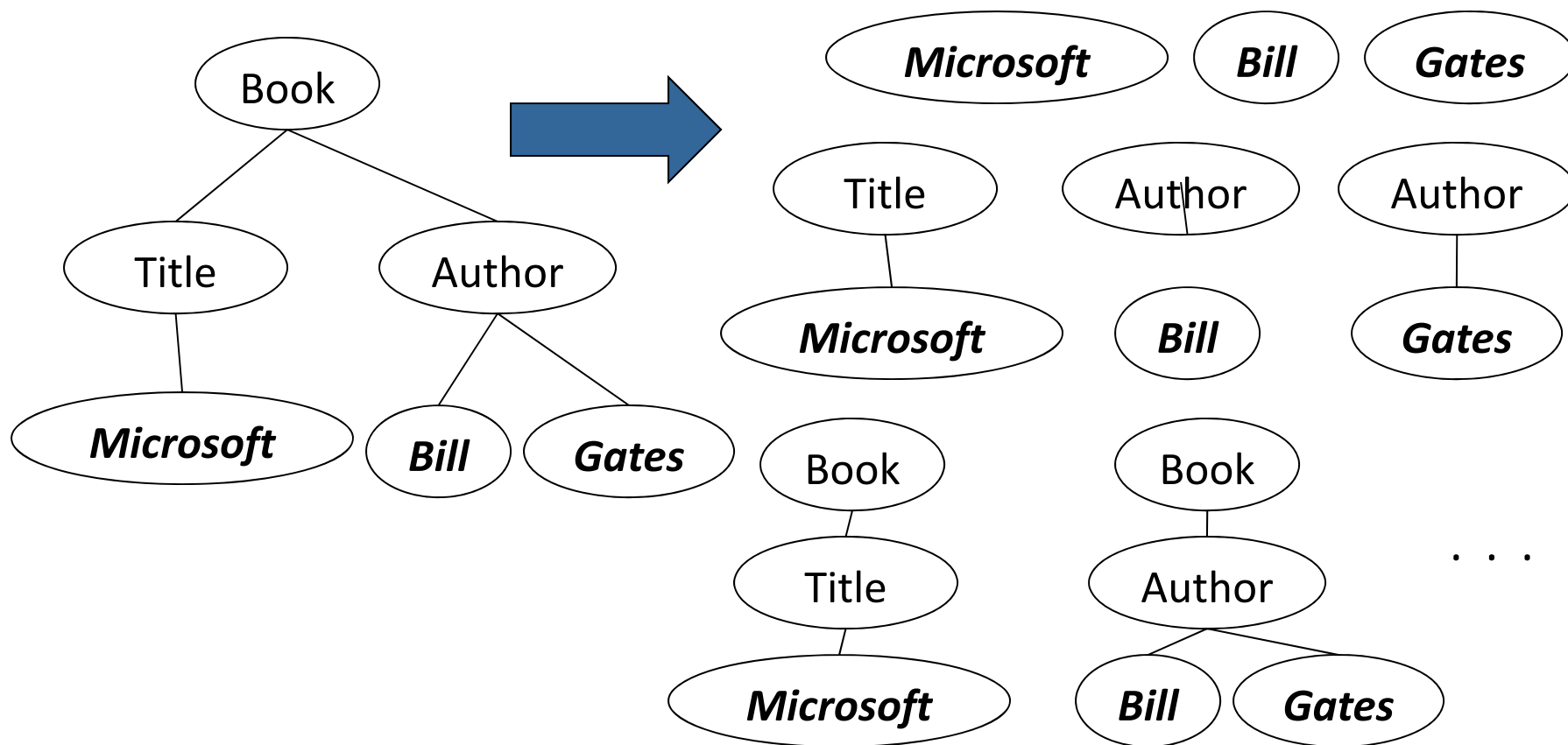
Aim: to have each dimension of the vector space encode a word together with its position within the XML tree.

How: Map XML documents to lexicalized subtrees.



# Main idea: lexicalized subtrees

- 1 Take each text node (leaf) and break it into multiple nodes, one for each word. E.g. split *Bill Gates* into *Bill* and *Gates*
- 2 Define the dimensions of the vector space to be lexicalized subtrees of documents – subtrees that contain at least one vocabulary term.



## Lexicalized subtrees

---

We can now represent queries and documents as vectors in this space of lexicalized subtrees and compute matches between them, e.g. using the vector space formalism.

### **Vector space formalism in unstructured VS. structured IR**

The main difference is that the dimensions of vector space in unstructured retrieval are vocabulary terms whereas they are lexicalized subtrees in XML retrieval.



## Structural term

---

There is a tradeoff between the dimensionality of the space and the accuracy of query results.

- If we restrict dimensions to vocabulary terms, then we have a standard vector space retrieval system that will retrieve many documents that do not match the structure of the query (e.g., *Gates* in the title as opposed to the author element).
- If we create a separate dimension for each lexicalized subtree occurring in the collection, the dimensionality of the space becomes too large.

**Compromise:** index all paths that end in a single vocabulary term, in other words all XML-context term pairs. We call such an XML-context term pair a structural term and denote it by  $\langle c, t \rangle$ : a pair of XML-context  $c$  and vocabulary term  $t$ .

## Context resemblance

---

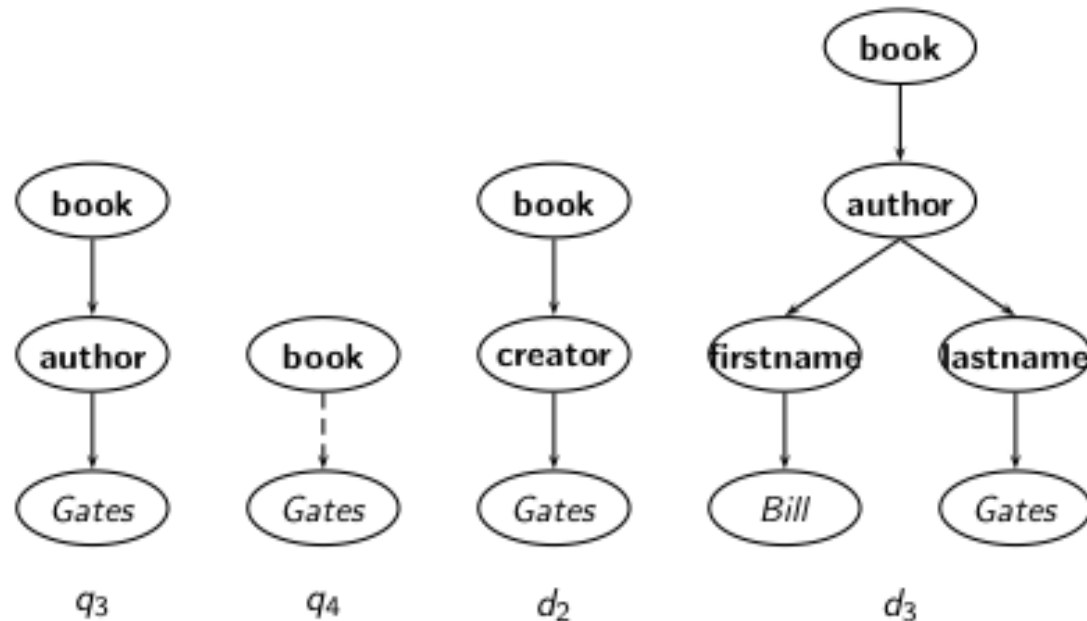
A simple measure of the similarity of a path  $c_q$  in a query and a path  $c_d$  in a document is the following *context resemblance* function CR:

$$\text{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

$|c_q|$  and  $|c_d|$  are the number of nodes in the query path and document path, resp.

$c_q$  matches  $c_d$  iff we can transform  $c_q$  into  $c_d$  by inserting additional nodes.

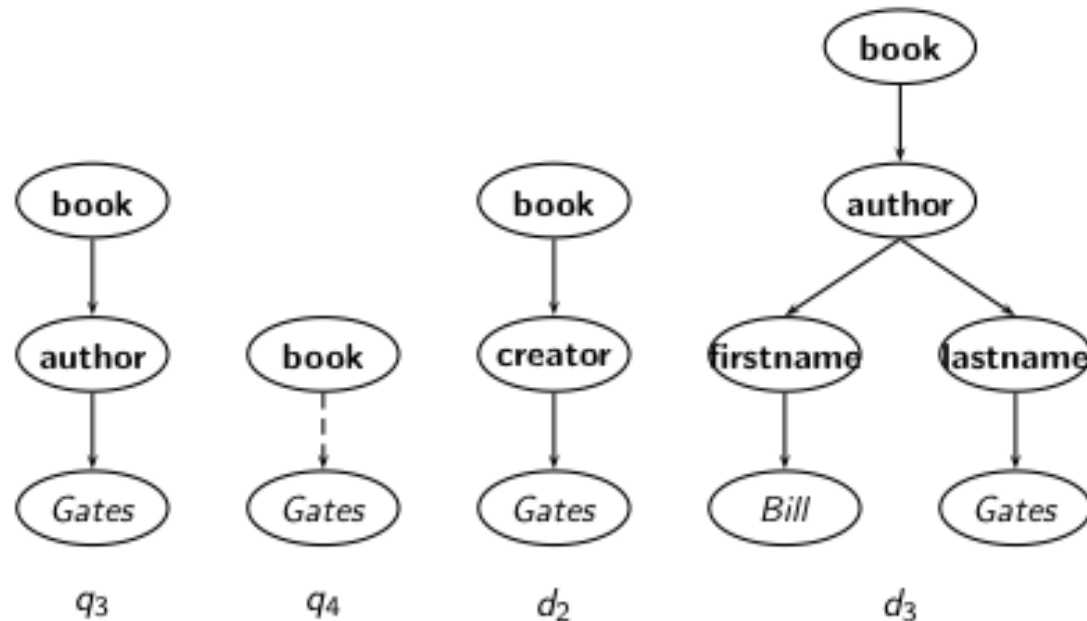
# Context resemblance example



$$CR(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

$CR(c_q, c_d) = 3/4 = 0.75$ . The value of  $CR(c_q, c_d)$  is 1.0 if  $q$  and  $d$  are identical.

# Context resemblance example



$$CR(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

$$CR(c_q, c_d) = ? \quad CR(c_q, c_d) = 3/5 = 0.6.$$

## Document similarity measure

The final score for a document is computed as a variant of the cosine measure, which we call SIMNOMERGE.

$\text{SIMNOMERGE}(q, d) =$

$$\sum_{c_k \in B} \sum_{c_l \in B} \text{CR}(c_k, c_l) \sum_{t \in V} \text{weight}(q, t, c_k) \frac{\text{weight}(d, t, c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d, t, c)}}$$

- $V$  is the vocabulary of non-structural terms
- $B$  is the set of all XML contexts
- $\text{weight}(q, t, c)$ ,  $\text{weight}(d, t, c)$  are the weights of term  $t$  in XML context  $c$  in query  $q$  and document  $d$ , resp. (standard weighting e.g.  $\text{idf}_t \times \text{wf}_{t,d}$ , where  $\text{idf}_t$  depends on which elements we use to compute  $\text{df}_t$ .)

$\text{SIMNOMERGE}(q, d)$  is not a true cosine measure since its value can be larger than 1.0.

# SIMNOMERGE algorithm

---

SCOREDOCUMENTSWITHSIMNOMERGE( $q, B, V, N, \text{normalizer}$ )

```
1  for  $n \leftarrow 1$  to  $N$ 
2  do  $\text{score}[n] \leftarrow 0$ 
3  for each  $\langle c_q, t \rangle \in q$ 
4  do  $w_q \leftarrow \text{WEIGHT}(q, t, c_q)$ 
5     for each  $c \in B$ 
6     do if  $\text{CR}(c_q, c) > 0$ 
7         then  $\text{postings} \leftarrow \text{GETPOSTINGS}(\langle c, t \rangle)$ 
8             for each  $\text{posting} \in \text{postings}$ 
9                 do  $x \leftarrow \text{CR}(c_q, c) * w_q * \text{weight}(\text{posting})$ 
10                     $\text{score}[\text{docID}(\text{posting})] + = x$ 
11 for  $n \leftarrow 1$  to  $N$ 
12 do  $\text{score}[n] \leftarrow \text{score}[n] / \text{normalizer}[n]$ 
13 return  $\text{score}$ 
```

## Initiative for the Evaluation of XML retrieval (INEX)

INEX: standard benchmark evaluation (yearly) that has produced test collections (documents, sets of queries, and relevance judgments).  
Based on IEEE journal collection (since 2006 INEX uses the much larger English Wikipedia test collection).  
The relevance of documents is judged by human assessors.

### INEX 2002 collection statistics

12,107	number of documents
494 MB	size
1995—2002	time of publication of articles
1,532	average number of XML nodes per document
6.9	average depth of a node
30	number of CAS topics
30	number of CO topics

# INEX topics

---

Two types:

- 1 content-only or **CO topics**: regular keyword queries as in unstructured information retrieval
- 2 content-and-structure or **CAS topics**: have structural constraints in addition to keywords

Since CAS queries have both structural and content criteria, relevance assessments are more complicated than in unstructured retrieval



## INEX relevance assessments

---

INEX 2002 defined component coverage and topical relevance as orthogonal dimensions of relevance.

### Component coverage

Evaluates whether the element retrieved is “structurally” correct, i.e., neither too low nor too high in the tree.

We distinguish four cases:

- 1 Exact coverage (E): The information sought is the main topic of the component and the component is a meaningful unit of information.
- 2 Too small (S): The information sought is the main topic of the component, but the component is not a meaningful (self-contained) unit of information.
- 3 Too large (L): The information sought is present in the component, but is not the main topic.
- 4 No coverage (N): The information sought is not a topic of the component.

## INEX relevance assessments

---

The **topical relevance** dimension also has four levels: highly relevant (3), fairly relevant (2), marginally relevant (1) and nonrelevant (0).

### Combining the relevance dimensions

Components are judged on both dimensions and the judgments are then combined into a digit-letter code, e.g. 2S is a fairly relevant component that is too small. In theory, there are 16 combinations of coverage and relevance, but many cannot occur. For example, a nonrelevant component cannot have exact coverage, so the combination 3N is not possible.

## INEX relevance assessments

The relevance-coverage combinations are quantized as follows:

$$\mathbf{Q}(rel, cov) = \begin{cases} 1.00 & \text{if } (rel, cov) = 3E \\ 0.75 & \text{if } (rel, cov) \in \{2E, 3L\} \\ 0.50 & \text{if } (rel, cov) \in \{1E, 2L, 2S\} \\ 0.25 & \text{if } (rel, cov) \in \{1S, 1L\} \\ 0.00 & \text{if } (rel, cov) = 0N \end{cases}$$

This evaluation scheme takes account of the fact that binary relevance judgments, which are standard in unstructured IR, are not appropriate for XML retrieval. The quantization function  $\mathbf{Q}$  does not impose a binary choice relevant/nonrelevant and instead allows us to grade the component as partially relevant. The number of relevant components in a retrieved set  $A$  of components can then be computed as:

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} \mathbf{Q}(rel(c), cov(c))$$

## INEX evaluation measures

---

As an approximation, the standard definitions of precision and recall can be applied to this modified definition of relevant items retrieved, with some subtleties because we sum graded as opposed to binary relevance assessments.

### Drawback

Overlap is not accounted for. Accentuated by the problem of multiple nested elements occurring in a search result.

Recent INEX focus: develop algorithms and evaluation measures that return non-redundant results lists and evaluate them properly.

## Recap

---

- Structured or XML IR: effort to port unstructured (standard) IR know-how onto a scenario that uses structured (DB-like) data
- Specialized applications (e.g. patents, digital libraries)
- A decade old, unsolved problem
- <http://inex.is.informatik.uni-duisburg.de/>

# A Data Mashup Language for the Data Web

Mustafa Jarrar, Marios D. Dikaiakos

University of Cyprus

LDOW 2009, April 20, 2009

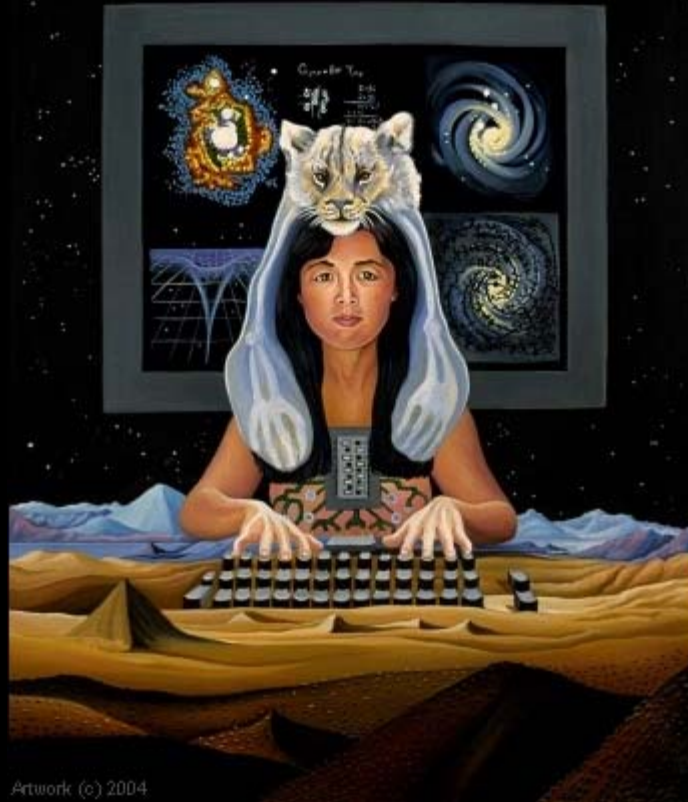
Edited & Presentation by Sangkeun Lee, IDS Lab

Original Slides : <http://www.cs.ucy.ac.cy/~mjarrar/Internal/MashQL.V07.ppt>

Imagine We are in 2050.

The internet is a database  
Information about every little thing

Structured,  
granular data



Semantics,  
linked data

(oracle?)

How we will ~~yahoo/google~~ this knowledge !!?

# Outline

---

- Introduction & Motivation
- The MashQL Language
- The Notion of Query Pipes
- Implementation
- Use cases
- Discussion and Future Directions



# Introduction & Motivation

---

- We are witnessing
  - A rapid emergence of the Data Web
  - Many companies started to make their content freely accessible through APIs
    - E.g. Google Base, eBay, Flickr, eBay
  - Many accessible data in RDF, RDFa

# Web 2.0 and the phenomena of APIs

The screenshot shows a Wikipedia article titled "University of Cyprus" in a Windows Internet Explorer browser window. The browser's address bar shows the URL "University of Cyprus - Wikipedia, the free encyclopedia". The page features the Wikipedia logo, a navigation menu, and a search box. The main content area displays the article title "University of Cyprus" and a summary: "From Wikipedia, the free encyclopedia". The article text describes the University of Cyprus (UCY) as a young university established in 1989, with its first students in 1992 and approximately 3,500 students in 2004/2005. It mentions that the university is based in Nicosia, Cyprus, and that its official languages are Greek and Turkish. The article also notes that the university's credit point system is based on ECTS and that Turkish Cypriots are eligible for admission upon passing special examinations. A sidebar on the right provides additional information about the university, including its establishment year (1989), type (public), number of undergraduates (3,500), location (Nicosia, Cyprus), and website (http://www.ucy.ac.cy). A red button in the top right corner of the article area says "25,190 have donated" and "» Donate now!".

University of Cyprus - Wikipedia, the free encyclopedia - Windows Internet Explorer

Sign in / create account

article discussion edit this page history

**WIKIPEDIA**  
The Free Encyclopedia

navigation

- Main page
- Contents
- Featured content
- Current events
- Random article

interaction

- About Wikipedia
- Community portal
- Recent changes
- Contact Wikipedia
- Donate to Wikipedia
- Help

search

Go Search

toolbox

- What links here
- Related changes
- Upload file
- Special pages

**You** can help Wikipedia change the world!  
"Sharing knowledge is one way to peace!" - Elmar Höfner

**25,190** have donated  
» Donate now!

## University of Cyprus

From Wikipedia, the free encyclopedia

The **University of Cyprus** (UCY) (in Greek: Πανεπιστήμιο Κύπρου in Turkish: Kibris Universitesi) is a young university established in 1989. It admitted its first students in 1992 and has currently approximately 3,500 students (2004/2005).

Based at the Capital of Cyprus, Nicosia. Teaching is mainly in Greek. The official languages are Greek and Turkish, but only a few Turkish speakers are registered. Since September 2005, the University's credit point system is based on ECTS. Turkish Cypriots who hold a six-year high-school diploma are eligible for admission upon passing special examinations set by the University.

Those eligible to participate in the entrance examination for the University of Cyprus are Cypriot citizens or those with at least one parent of Cypriot origin. Prospective students must have graduated from a six-year high school, and have completed the necessary application forms within the time limits set by the Ministry of Education and Culture. A limited number of positions is granted to the handicapped or those with special needs.

**University of Cyprus**  
Πανεπιστήμιο Κύπρου  
Kibris Universitesi

<b>Established</b>	1989
<b>Type</b>	public
<b>Undergraduates</b>	3,500
<b>Location</b>	Nicosia, Cyprus
<b>Website</b>	http://www.ucy.ac.cy

API



# Web 2.0 and the phenomena of APIs

The screenshot shows a YouTube video player for a video titled "Syrtaki". The video is from the channel "armostis", which has 12 videos and was joined 10 months ago. The video description states it is a sample of how they dance syrtaki from the "Celebrity Rhythms" show at the University of Cyprus in 2005. The video has 19,153 views, a 4.5-star rating from 29 ratings, and 14 comments. The interface includes navigation tabs for Videos, Categories, Channels, and Community, a search bar, and a "Share" button. The video player shows a progress bar at 00:27 / 00:46. The "Comments & Responses" section shows a comment from "ba1782" (1 week ago). The "Related Videos" section lists other videos like "Zorbas Syrtaki" and "Syrtaki - Culture Festival-Purdue Diversity Week April 2007".

API

# Web 2.0 and the phenomena of APIs

The screenshot shows the Amazon.com homepage with a search bar containing the text "terry halpin". The search results are categorized under "Most Gifted in Software". Three products are displayed: "Battlefield 1942: The Complete..." for \$9.99, "World of Warcraft 60 Day Pre-Paid..." for \$29.99, and "Spore Creature Creator" for \$9.99. Below the software results, there is a section titled "Shop for Blockbuster Movie Character Costumes from Costume Supercenter" featuring three costume images: a black and yellow outfit, a blue dress, and a Batman costume. The browser window title is "Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more - Mozi...".

API

# Web 2.0 and the phenomena of APIs

**upcoming** Sign in or join now Help?

Home Events Places Friends Groups Popular Add New Event » a YAHOO! service

My Events My Scrapbook My Friends' Events

**Brussel** < change my location  
Brussel Hoofdstad / Vlaams Brabant / Belgium

Find events What? (e.g. jazz, U2, Lakers) Where? Brussel, Vlaams Braba GO

**This week on Upcoming**

- Wed** Sonny Rollins  
Bezar  
Vic Chesnutt - Loney, Dear  
Ancienne Belgique
- Sat** The New Pornographers @ Botanique  
Botanique, Rotonde
- Mon** iliketrails  
Botanique, Rotonde
- Tue** The Cat Empire @ AB  
AB club  
2nd Annual Mobile Reference De:  
Courtyard Marriott Hotel Brussels  
See all...

**Make Upcoming smarter**  
Know something going on in Brussel that we don't? Add it to Upcoming to share it with the world.

**See more events nearby:**

**Notice anything different?**  
Say Hi to the new Upcoming.  
[Learn more...](#)

November 2007

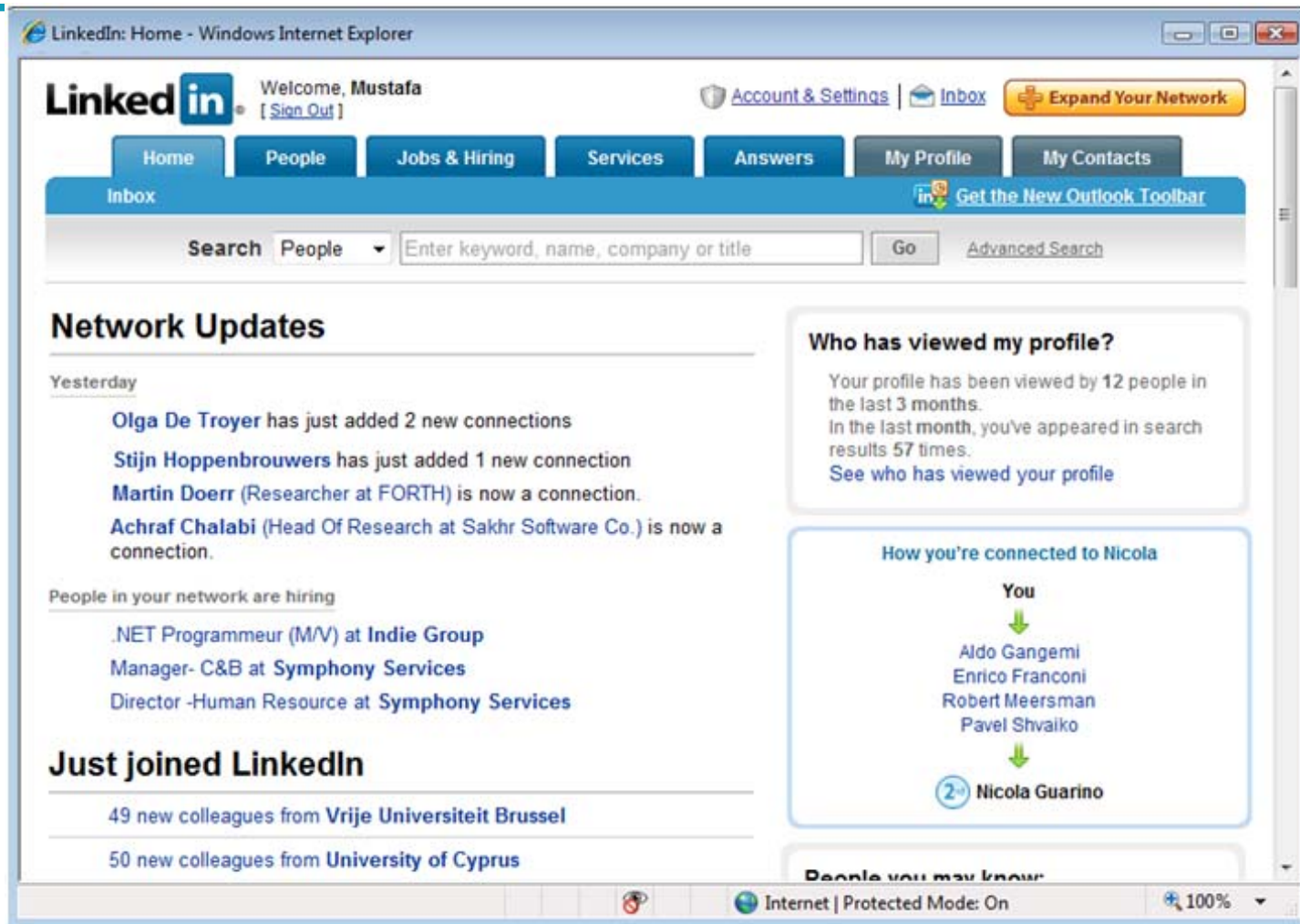
Show me events:  
This weekend  
Next 7 days  
Next 30 days  
Everything!

**Browse by category**

Music	Social	Education
<b>+2</b> The New Pornographers @ Botanique Nov 24 at Botanique, Rotonde	<b>+1</b> Yulbiz-Brussels #2 Dec 15 at Café Walvis	<b>+3</b> Barcamp Brussels (#4) Dec 1 at mVillage Business Center
<b>+1</b> iliketrails Nov 26 at Botanique, Rotonde		<b>+1</b> Profoss: Virtualisation Jan 22 at Brussel
<b>+1</b> The Cat Empire @ AB Nov 27 at AB club		
<b>-</b> The Mice @ AB		

API

# Web 2.0 and the phenomena of APIs



API

# Web 2.0 and the phenomena of APIs



API



# Web 2.0 and the phenomena of APIs

Google Base - Mozilla Firefox

**Google Base** BETA Post it on Base. Find it on Google.

Simply describe your items on Base to make them as easy as possible for people to find when they search. You do not need a website to put your stuff online. [Learn more...](#)

**Examples of popular item types:**

- [Events and activities](#)
- [Housing](#)
- [Personals](#)
- [Products](#)
- [Recipes](#)
- [Vehicles](#)
- [Hotels](#)
- [Jobs](#)
- [People profiles](#)
- [Reviews](#)
- [Services](#)

**Already use Google Base?**  
[MustafaJarrar@gmail.com's items](#)

**Get started - post items on Base:**

- [One at a time](#)  
Have 10 or fewer items? Fill out our web form for each item you would like to post.
- [Data feed](#)  
Upload a spreadsheet or XML file describing lots of items.

**And many, many others APIs**

API

# Web 2.0 and the phenomena of APIs

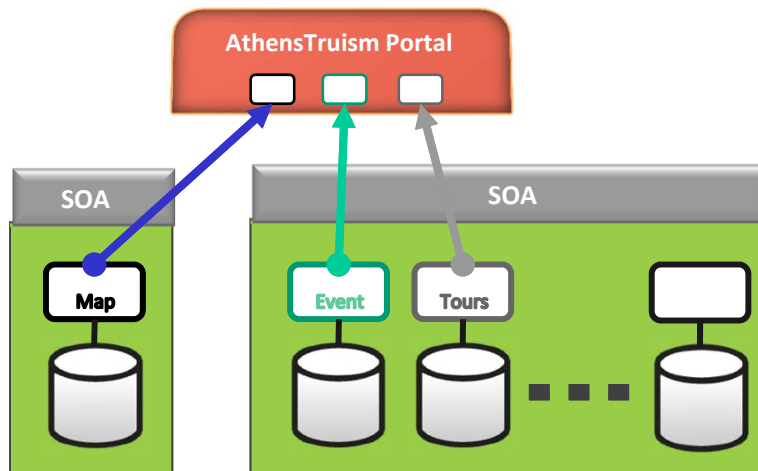
---



Moving to the **Data Web**, in parallel to the web of documents.

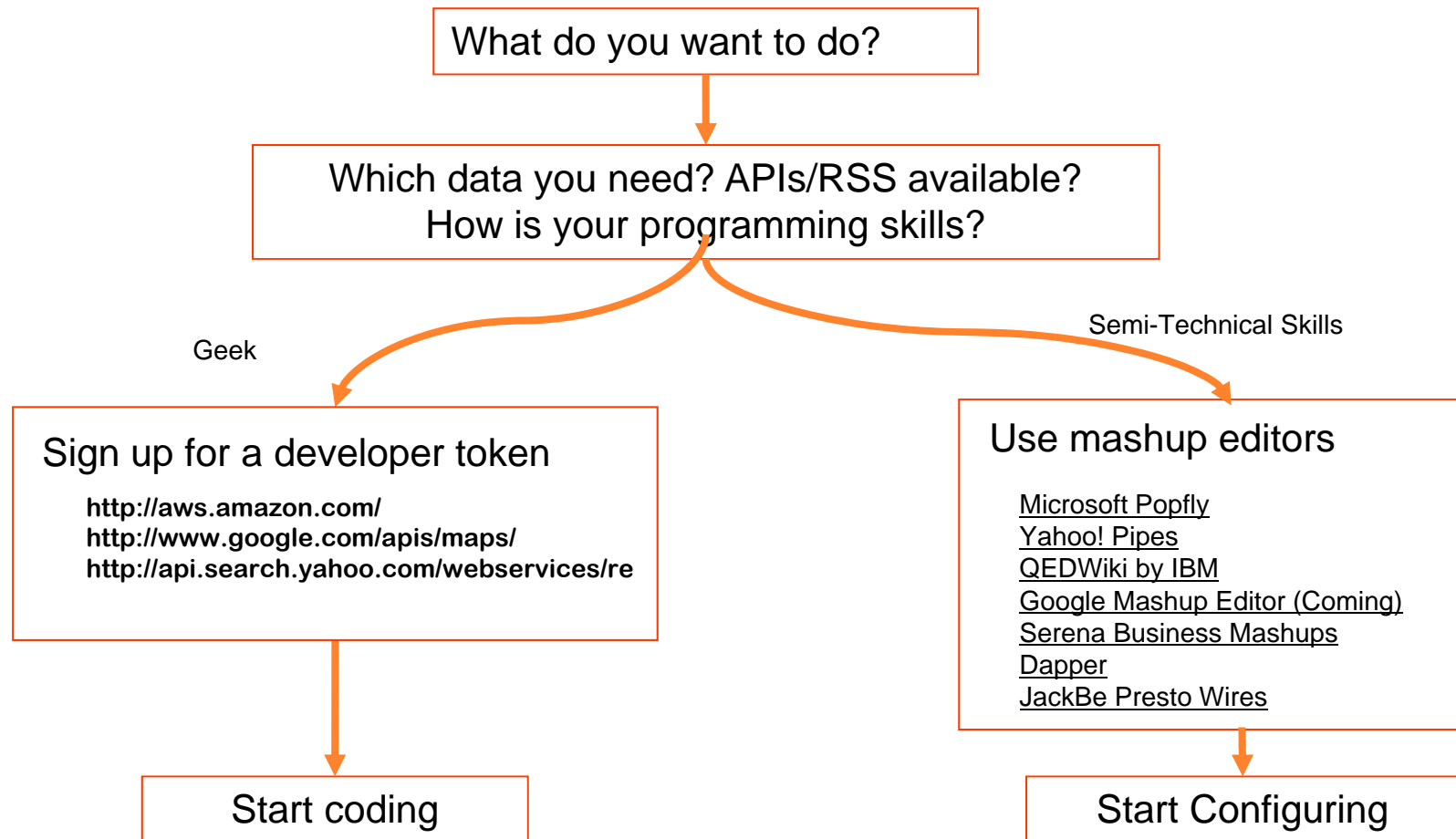
# Introduction & Motivation

- A Mashup?
  - A Web application that consumes data originated from third parties and retrieved via APIs
  - Problem
    - Building mashups is an art that is limited to skilled programmers
    - Some mashup editors have been proposed by Web 2.0 communities, but...?

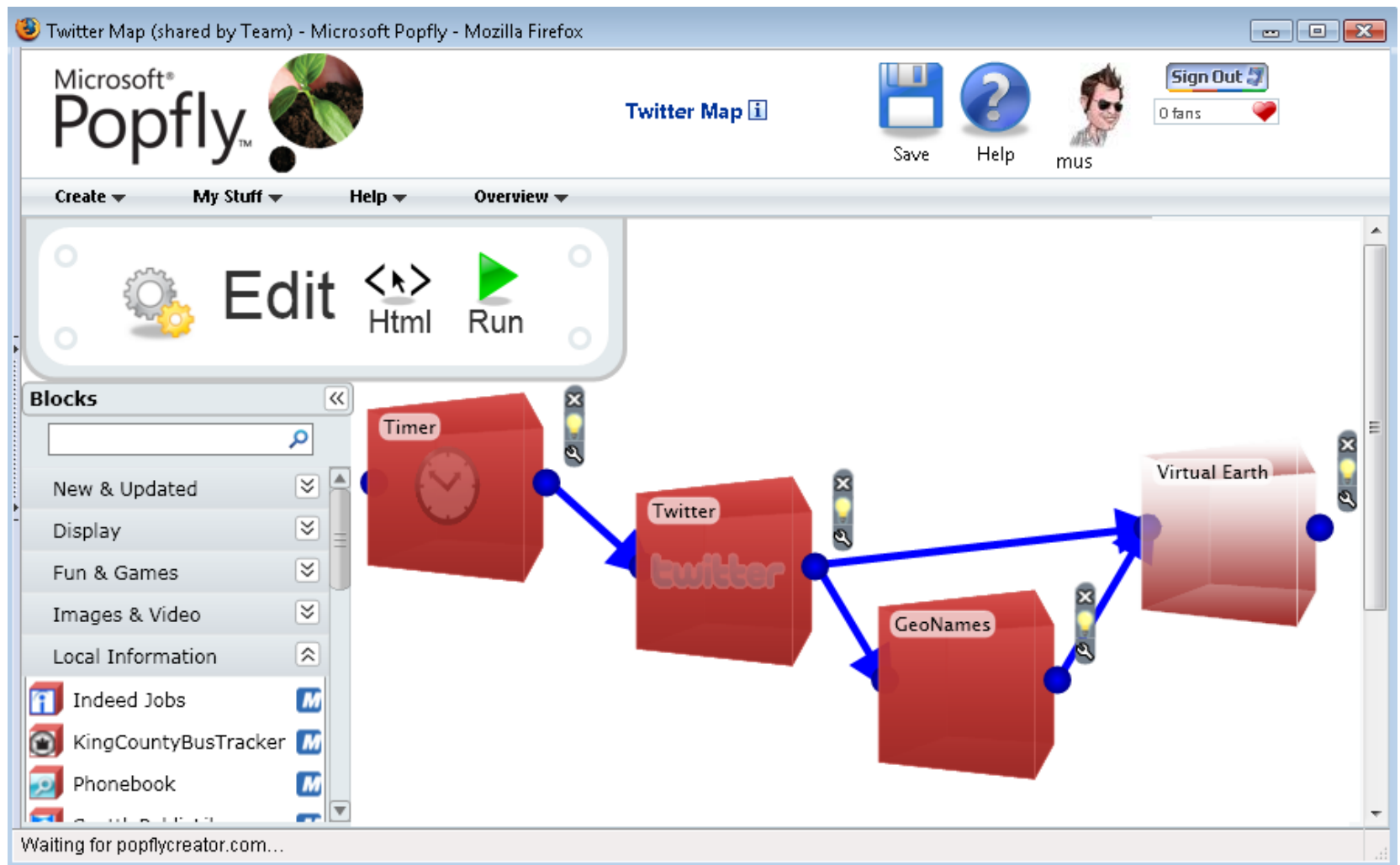


(A puzzle of APIs)  
 $(API_1 + API_2) + API_3 = \text{money}$

# How to Build a Mashup?



# Mashup Editors



# Mashup Editors

The screenshot shows the Microsoft Popfly Mashup Editor interface. The browser window title is "Oceans of the World (shared by Team) - Microsoft Popfly - Mozilla Firefox". The page header includes the Microsoft Popfly logo, the title "Oceans of the World", and navigation links for "Save", "Help", and "Sign Out". A user profile for "mus" with "0 fans" is visible. The main workspace is a red canvas with a "flickr" block. A "Switch to an advanced view" button is at the top right of the canvas. A help panel on the right contains "More Help" links: "How do I..?", "Videos", and "Samples". Below these are two paragraphs of text: "When you zoom in on a block you can choose what you want the block to do by selecting one of the operations from the top most drop down list." and "Each operation requires some inputs. These inputs will change what the block will output. For information on what each input does, hover over the text box." A third paragraph states: "Inputs can be mapped from the outputs of any blocks that are connected to the zoomed in block. You can do this by selecting the incoming block from the left hand".

**Operations:**

- getGeotaggedPhotos
- getGeotaggedPhotos
- getPhotos
- getInterestingPhotos
- getTags
- getUserPhotos
- getPhotoset

Latitude and longitude and have been tagged as "geotagged".

number [custom] 15

sort [custom] interestingness-desc

No blocks are sending data to this block, so you can only type in custom input values.

OK

# Mashup Editors

The screenshot displays the IBM DAMIA Mashup Editor interface. The main workspace shows a workflow diagram with the following operators: \$Excel1, \$Filter2, \$Import3, \$Transform4, \$Merge5, \$Sort6, and \$Publish7. The \$Publish7 operator is highlighted in green. Below the workspace, the configuration panel for the selected operator is visible, including fields for Name, Feed Type, Title, Link, and Description. A help panel on the right provides information about the Publish operator.

**Name:** \$Publish7

**Feed Type:** RSS

**Common RSS Channel elements:**

**Title:** Floria Policy Holders at Risk

**Link:** http://damia.alphaworks.ibm.com

**Description:** Joins sample excel spreadsheet with Florida weather alerts

**The Publish operator is used to create an RSS feed, adding header information or content specific to the feed. You can use the output of the Publish operator as a Feed for other operators or Mashups. All types of input streams can be converted to a different output feed type using the Publish operator.**

**Parameters**

**Name:** The name of the Publish operator. This name starts with a dollar sign (\$) and does not contain spaces.

# Mashup Editors

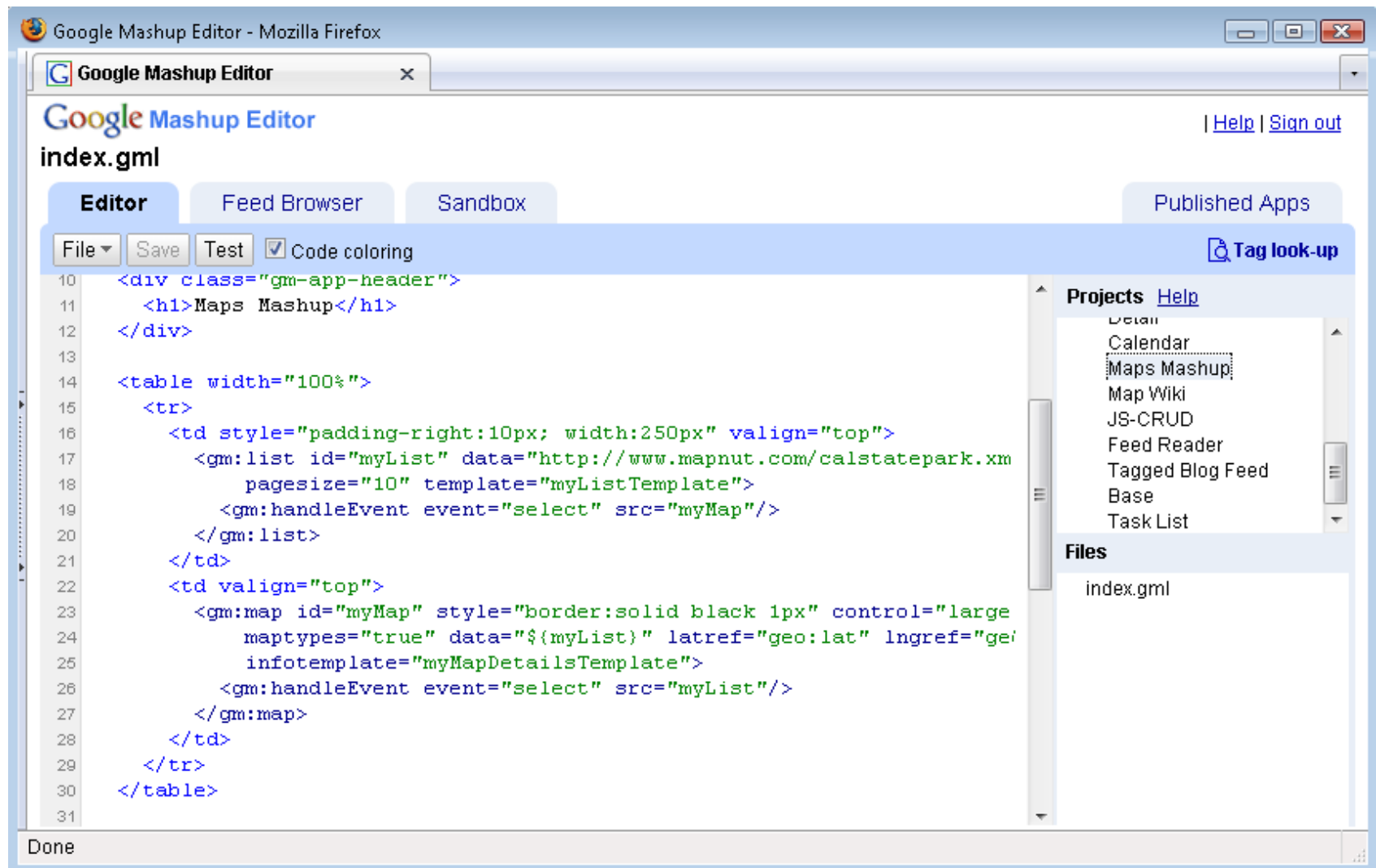
The screenshot displays the 'Pipes' mashup editor interface within a Windows Internet Explorer browser window. The title bar reads 'Pipes: editing 'LondonEvents' - Windows Internet Explorer'. The interface includes a navigation menu on the left with categories like Sources, User inputs, Operators, Url, String, and Date. The main workspace shows a workflow diagram with the following components:

- Fetch Feed**: A pipe with a URL field containing 'http://upcoming.yahoo.com/syndic'.
- Google Base**: A pipe with a search query: 'Find events and activities with keywords text within 20 miles of London'.
- Union**: A central pipe that receives input from both the 'Fetch Feed' and 'Google Base' pipes.
- Filter**: A pipe that filters the output of the 'Union' pipe. It is configured to 'Permit items that match all of the following' with a rule: 'item.description Contains text'.
- Pipe Output**: The final output of the workflow.

A status bar at the bottom right indicates 'Debugger: Union (25 items)'.



# Mashup Editors



## Limitations of Mashup Editors

---

- Focus only on providing encapsulated access to (some) public APIs and feeds (rather than querying data sources).
- Still require programming skills.
- Cannot play the role of a *general-purpose data retrieval*, as mashups are sophisticated applications.
- Lacks a formal framework for pipelining mashups.

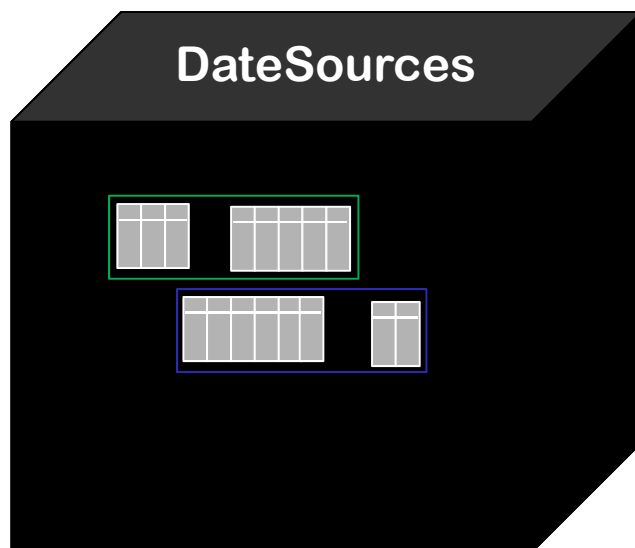
# Vision

---

- Position
  - The author propose to regard the web as a database
  - Mashup is seen as a query over one or multiple sources
- So, instead of developing a mashup as an application that access structured data through APIs,
- We regard *a mashup as a query*
- *Challenges*
  - But the problem then is: users need to know the schema and technical details of the data sources they want to query.

# Vision and Challenges

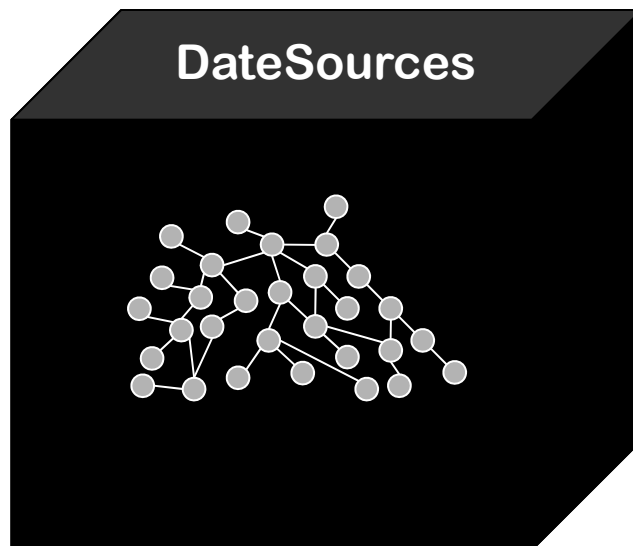
How a user can query a source without knowing its schema, structure, and vocabulary?



```
SELECT S.Title FROM Google.Scholar S
Where (S.Author='Hacker')
Union
SELECT P.PattentTitle FROM Ggoogle.Patent P
Where (P.Inventor ='Hacker')
Union
SELECT A.Title FROM Citeseer A
Where (P.Author ='Hacker')
```

# Vision and Challenges


How a user can query a source without knowing its schema, structure, and vocabulary?



```
SELECT S.Title FROM Google.Scholar S
Where (S.Author='Hacker')
Union
SELECT P.PattentTitle FROM Ggoogle.Patent P
Where (P.Inventor ='Hacker')
Union
SELECT A.Title FROM Citeseer A
Where (P.Author ='Hacker')
```

# MashQL

---

- A simple query language for the Data Web, in a mashup style.
- MashQL allows **querying a dataspace(s) without any prior knowledge about its schema, vocabulary or technical details** (a source may not have a schema at all).  Explore unknown graph
- **Does not assume any knowledge about RDF, SPARQL, XML, or any technology**, to get started.
- Users only use drop-lists to formulate queries (query-by-diagram/interaction).

# MashQL Example 1

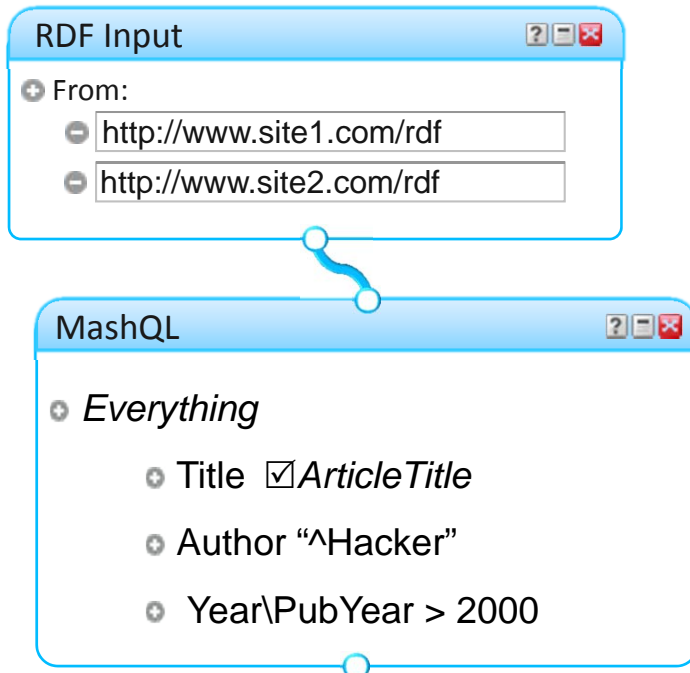
<http://www.site1.com/rdf>

```
<:a1> <:Title>      "Web 2.0"  
<:a1> <:Author>    "Hacker B."  
<:a1> <:Year>      2007  
<:a1> <:Publisher> "Springer"  
<:a2> <:Title>      "Web 3.0"  
<:a2> <:Author>    "Smith B."  
<:a2> <:Cites>     <:a1>
```

<http://www.site2.com/rdf>

```
<:4> <:Title>      "Semantic Web"  
<:4> <:Author>    "Tom Lara"  
<:4> <:PubYear>   2005  
<:5> <:Title>      "Web Services"  
<:5> <:Author>    "Bob Hacker"
```

## Hacker's Articles after 2000?



# MashQL Example 1

<http://www.site1.com/rdf>

```
<:a1> <:Title>      "Web 2.0"  
<:a1> <:Author>    "Hacker B."  
<:a1> <:Year>      2007  
<:a1> <:Publisher> "Springer"  
<:a2> <:Title>      "Web 3.0"  
<:a2> <:Author>    "Smith B."  
<:a2> <:Cites>     <:a1>
```

<http://www.site2.com/rdf>

```
<:4> <:Title>      "Semantic Web"  
<:4> <:Author>    "Tom Lara"  
<:4> <:PubYear>   2005  
<:5> <:Title>      "Web Services"  
<:5> <:Author>    "Bob Hacker"
```

Hacker's Articles after 2000?

RDF Input

From:

- 
- 

MashQL

Everything

a1  
a2  
4  
5

Types  Instances

Interactive  
query formulation



# MashQL Example 1

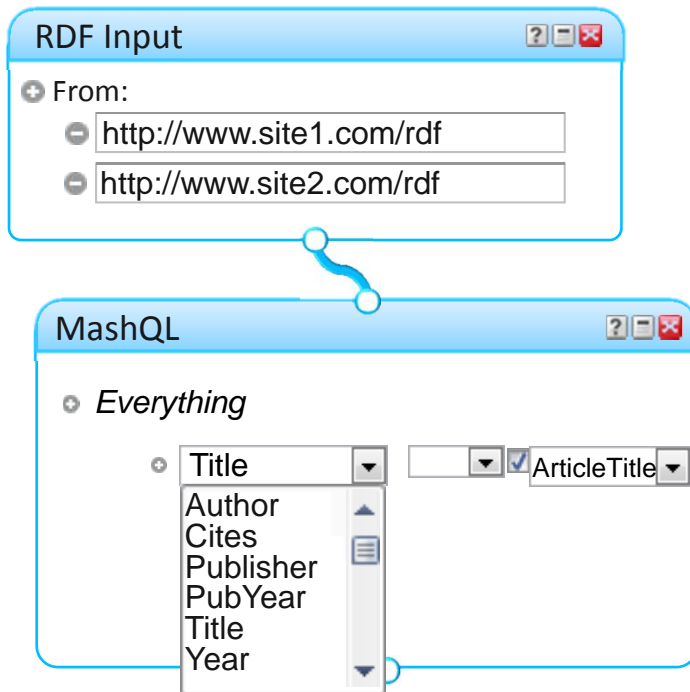
<http://www.site1.com/rdf>

```
<:a1> <:Title>      "Web 2.0"  
<:a1> <:Author>    "Hacker B."  
<:a1> <:Year>      2007  
<:a1> <:Publisher> "Springer"  
<:a2> <:Title>      "Web 3.0"  
<:a2> <:Author>    "Smith B."  
<:a2> <:Cites>     <:a1>
```

<http://www.site2.com/rdf>

```
<:4> <:Title>      "Semantic Web"  
<:4> <:Author>    "Tom Lara"  
<:4> <:PubYear>   2005  
<:5> <:Title>      "Web Services"  
<:5> <:Author>    "Bob Hacker"
```

## Hacker's Articles after 2000?



# MashQL Example 1

<http://www.site1.com/rdf>

```
<:a1> <:Title>      "Web 2.0"  
<:a1> <:Author>    "Hacker B."  
<:a1> <:Year>      2007  
<:a1> <:Publisher> "Springer"  
<:a2> <:Title>      "Web 3.0"  
<:a2> <:Author>    "Smith B."  
<:a2> <:Cites>     <:a1>
```

<http://www.site2.com/rdf>

```
<:4> <:Title>      "Semantic Web"  
<:4> <:Author>    "Tom Lara"  
<:4> <:PubYear>   2005  
<:5> <:Title>      "Web Services"  
<:5> <:Author>    "Bob Hacker"
```

## Hacker's Articles after 2000?

The screenshot shows two windows. The top window, titled "RDF Input", has a "From:" section with two entries: "http://www.site1.com/rdf" and "http://www.site2.com/rdf". A blue line connects the bottom of this window to the top of the "MashQL" window below. The "MashQL" window has a tree view on the left with "Everything" expanded, showing "Title" (with a checked "Article title" checkbox) and "Author". The "Author" node is selected, and a dropdown menu is open showing options: "Author", "Cites", "Publisher", "PubYear", "Title", and "Year". To the right of this menu is a "Con" dropdown menu with options: "Equals", "Contains", "OneOf", "Not", "Between", "LessThan", and "MoreThan". Further right is a "Hacker" dropdown menu.

# MashQL Example 1

<http://www.site1.com/rdf>

```
<:a1> <:Title>      "Web 2.0"  
<:a1> <:Author>    "Hacker B."  
<:a1> <:Year>      2007  
<:a1> <:Publisher> "Springer"  
<:a2> <:Title>      "Web 3.0"  
<:a2> <:Author>    "Smith B."  
<:a2> <:Cites>     <:a1>
```

<http://www.site2.com/rdf>

```
<:4> <:Title>      "Semantic Web"  
<:4> <:Author>    "Tom Lara"  
<:4> <:PubYear>   2005  
<:5> <:Title>      "Web Services"  
<:5> <:Author>    "Bob Hacker"
```

Hacker's Articles after 2000?

The screenshot shows two windows from the MashQL application. The top window, titled "RDF Input", has a "From:" section with two entries: "http://www.site1.com/rdf" and "http://www.site2.com/rdf". A blue line connects this window to the "MashQL" window below. The "MashQL" window shows a query configuration with a tree view on the left containing "Everything", "Title" (with a checked "Article title" checkbox), "Author '^Hacker'", and "Year | PubYe". The right side of the window shows a query editor with a dropdown menu set to "mor" and a text field containing "2000". Below the text field is a list of operators: "OneOf", "Not", "Between", "LessThan", and "MoreThan".

# MashQL Example 1

<http://www.site1.com/rdf>

```
<:a1> <:Title>      "Web 2.0"
<:a1> <:Author>     "Hacker B."
<:a1> <:Year>       2007
<:a1> <:Publisher>  "Springer"
<:a2> <:Title>      "Web 3.0"
<:a2> <:Author>     "Smith B."
<:a2> <:Cites>      <:a1>
```

<http://www.site2.com/rdf>

```
<:4> <:Title>      "Semantic Web"
<:4> <:Author>     "Tom Lara"
<:4> <:PubYear>    2005
<:5> <:Title>      "Web Services"
<:5> <:Author>     "Bob Hacker"
```

## Hacker's Articles after 2000?

RDF Input

From:

- 
- 

MashQL

- Everything
  - Title  Article title
  - Author  "Hacker"
  - Year/PubYear  > 2000



```
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:Title ?ArticleTitle}UNION
  {?X S2:Title ?ArticleTitle}}
  {?X S1:Author ?X1} UNION {?X S2:Author ?X1}
  {?X S1:PubYear ?X2} UNION {?X S2:Year ?X2}
  FILTER regex(?X1, "^Hacker")
  FILTER (?X2 > 2000)}
```

# MashQL Example 2

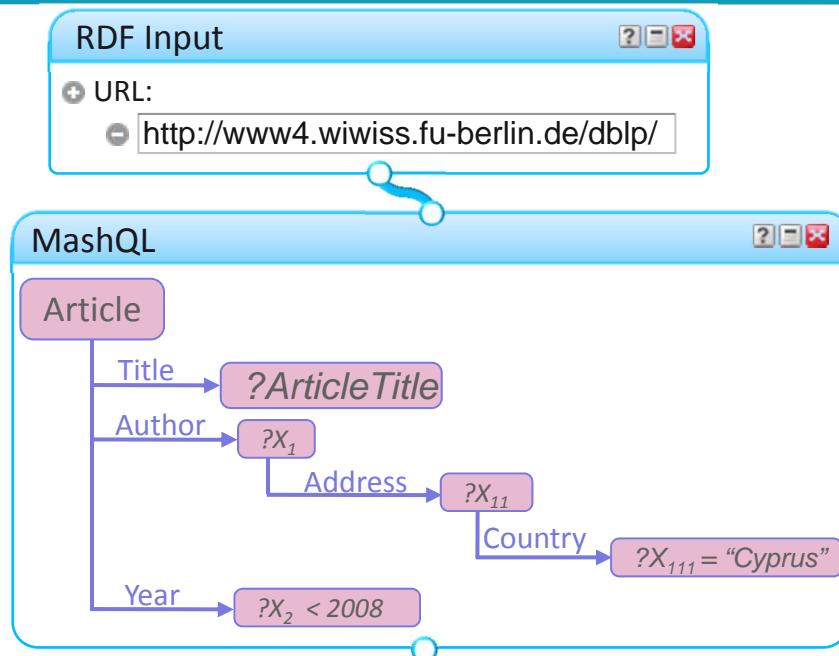
The screenshot shows two windows. The top window, titled 'RDF Input', contains a URL field with the text 'http://www4.wiwiss.fu-berlin.de/dblp/'. A blue line connects this window to the bottom window, titled 'MashQL'. The 'MashQL' window displays a query for 'Article' with the following conditions:

- Title  ArticleTitle
- Author
  - Address
    - Country "Cyprus"
- Year > 2008

The recent articles from Cyprus

→ Retrieve every Article that has a title, written by an author, who has an address, this address has a country called Cyprus, and the article 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  $S$ .  $Q(S) = R_1 \text{ AND } \dots \text{ AND } R_n$ .

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

Dif. 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.

# The Intuition of MashQL

The image shows two windows from the MashQL interface. The top window, titled 'RDF Input', contains a URL field with the text 'http://www4.wiwiss.fu-berlin.de/dblp/'. A blue line connects this window to the 'MashQL' window below. The 'MashQL' window displays a query for 'Article' with the following filters:

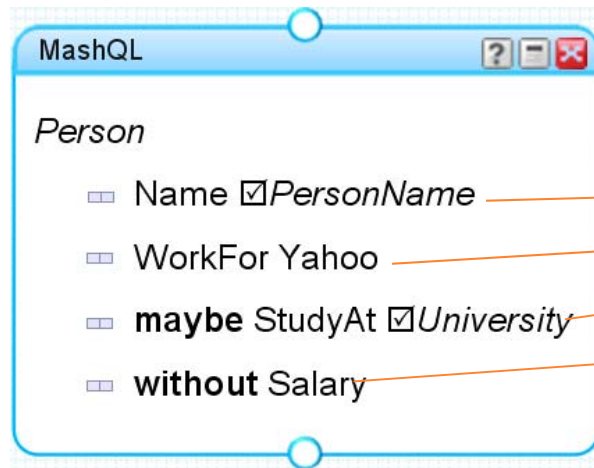
- Title  ArticleTitle
- Author
  - Address
  - Country "Cyprus"
- Year > 2008

An Object filter is one of :

- Equals
- Contains
- MoreThan
- LessThan
- Between
- one of
- Not(f)
- Information Path (sub query)

# More MashQL Constructs

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



```
SELECT ?PersonName, ?University
WHERE {
  ?Person :Name ?PersonName.
  ?Person :WorkFor :Yahoo.
  OPTIONAL{?Person :StudyAt ?University}
  OPTIONAL{?Person :Salary ?X1}
  FILTER (!Bound(?X1))
}
```



# More MashQL Constructs

- Union operator (denoted as “\”) between Objects, Predicates, Subjects and Queries

MashQL

- Person
  - WorkFor Google\Yahoo

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

MashQL

Person

- Surname\Firstname  FName

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

MashQL

a Person\Company

- Name  AgentName
- Phone  AgentPone

```
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}}
```

# MashQL Queries

---

- In the background, MashQL queries are translated into and executed as SPARQL queries.
- At the moment, we focus on RDF (/RDFa) as a data format, and SPARQL (/Oracle's SPARQL) as a backend query language. However, MashQL can be easily mappable to other query languages.

# MashQL Compilation

Depending on the pipeline structure, MashQL generates either SELECT or CONSTRUCT queries:

- SELECT returns the results in a tabular form (e.g. ArticleTitle, Author)
- CONSTRUCT returns the results in a triple form (e.g. Subject, Predicate, Object).

```
...  
CONSTRUCT *  
WHERE{?Job :JobIndustry ?X1.  
      ?Job :Type ?X2.  
      ?Job :Currency ?X3.  
      ?Job :Salary ?X4.  
      FILTER(?X1="Education" || ?X1="HealthCare")  
      FILTER(?X2="Full-Time" || ?X2="Fulltime" ||  
             ?X2="Contract")  
      FILTER(?X3="^Euro" || ?X3="^€")  
      FILTER(?X4>=75000 || ?X4<=120000)}
```

```
...  
SELECT ?Job ?Firm  
WHERE  
  {?Job :Location ?X1. ?X1 :Country ?X2.  
   FILTER (?X2="Italy" || ?X2="Spain" ||  
          ?X2="Greece" || ?X2="Cyprus")}  
   OPTIONAL{{?job :Organization ?Firm} UNION  
            {?job :Employer ?Firm}}
```

# MashQL Editor

Pipes: editing pipe - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http:192.168.1.2:8990/MashQL-MashQL-context-root/test/editor1.jsp

MashQL Pipes

MashQL Example1

Welcome Demo

Publish New Save Properties LOGOUT

**MashQL Query** +  
User Input +  
Fetch RDF +  
Fetch RDFa +  
HTML to RDF +  
Fetch XML +  
Fetch Feed +  
Fetch Database +  
Fetch CSV +  
Fetch XSL +  
My Pipes -  
Pipe1  
ArticlePipe  
GJobs  
UKJobs  
MyConferences

**RDFInput**

URL  
+ http://www.example1.com  
-

**MashQL**

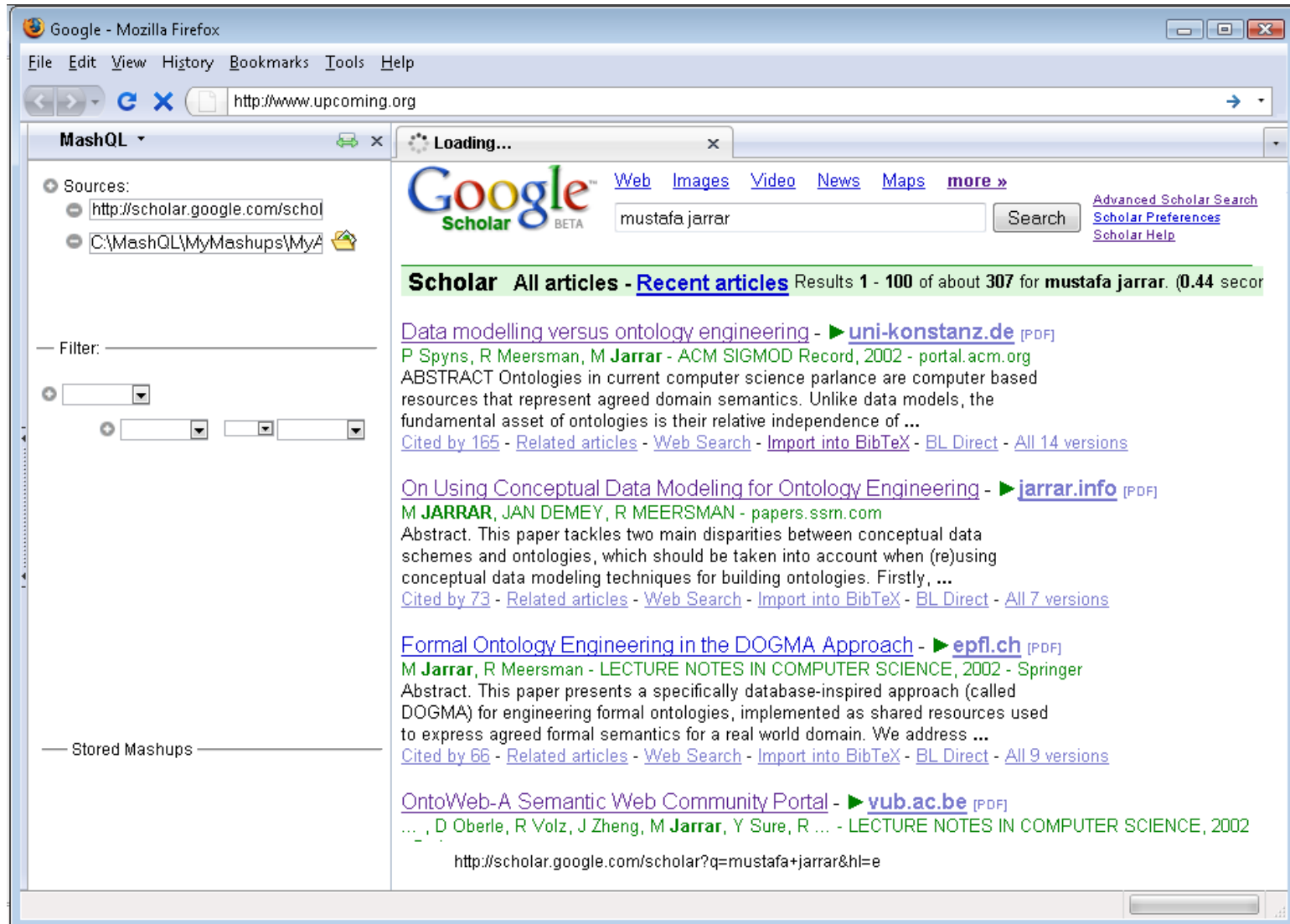
+ Retrieve every Article that:  
+ Title  ArticleTitle  
+ Author  
+ Address  
+ Country "Cyprus"  
+ Year > 2008

**Output**

Done

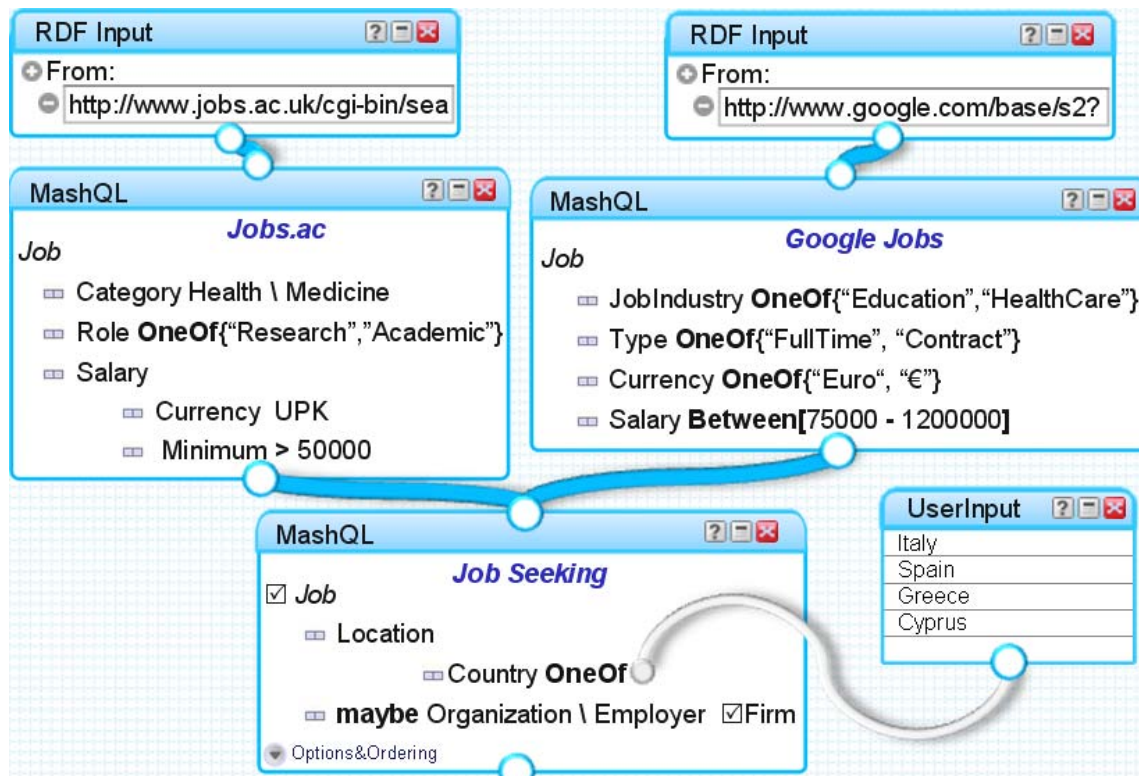
**Under Construction**

# MashQL Firefox Add-On (Light-mashups @ your browser)



# Use Case: Job Seeking

A mashup of job vacancies based on Google Base and on Jobs.ac.uk.



```

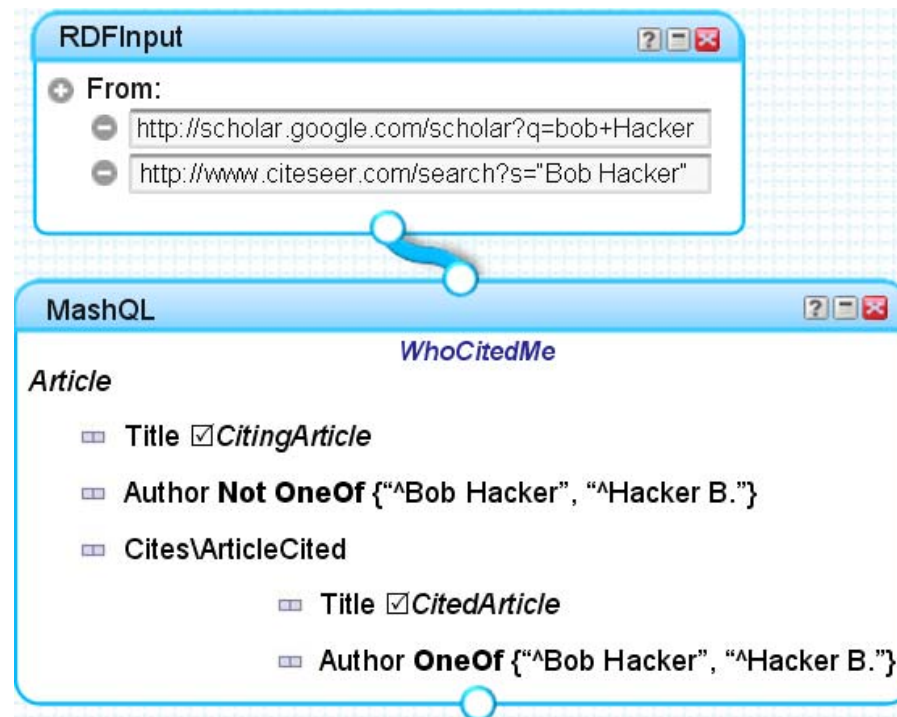
...
CONSTRUCT *
WHERE {
  {{?Job :Category :Health}UNION
  {?Job :Category :Medicine}}
  ?Job :Role ?X1.
  ?Job :Salary ?X2.
  ?X2 :Currency :UPK.
  ?X2 :Minimum ?X3.
  FILTER(?X1="Research" || ?X1="Academic")
}

...
CONSTRUCT *
WHERE{?Job :JobIndustry ?X1.
?Job :Type ?X2.
?Job :Currency ?X3.
?Job :Salary ?X4.
FILTER(?X1="Education" || ?X1="HealthCare")
FILTER(?X2="Full-Time" || ?X2="Fulltime") ||
?X2="Contract")
FILTER(?X3="^Euro" || ?X3="^€")
FILTER(?X4>=75000 || ?X4<=120000)}

...
SELECT ?Job ?Firm
WHERE
  {?Job :Location ?X1. ?X1 :Country ?X2.
  FILTER (?X2="Italy" || ?X2="Spain") ||
  ?X2="Greece" || ?X2="Cyprus")}
  OPTIONAL{{?job :Organization ?Firm} UNION
  {?job :Employer ?Firm}}
    
```

# Use Case: My Citations

A mashup of cited Hacker's articles (but no self citations), over Scholar and Sitemer



# Evaluation

---

## Query Execution :

- The performance of executing a MashQL query is bounded to the performance to executing its backend language (i.e. SPARQL/SQL).
- A query with medium size complexity takes one or few seconds (Oracle's SPARQL, [Chong et al 2007]).



# Conclusions

---

- A formal but yet simple query language for the Data Web, in a mashup and declarative style.
- Allows people to discover and navigate unknown data spaces(/graphs) without prior knowledge about the schema or technical details.
- Can be use as a general purpose data retrieval and filtering