siQuest

enRoute Questopia

The Team

- Andres Narvaez
- Charlie Han
- Scott Weaver
- Sowmya Moturi
- Tarun Kothuri

Agenda

- Introduction
- The Process
  - Converters
  - Analyzing
  - Query Building
  - Searching
  - Results
- Demonstration
- Experiences
- Q & A

Try telling your search engine ..

I Want Something Like This File Over Here ..

D'You Know What I Mean ?

D'You Know What This Is ?

- A search result leads you to a very effective document, but you cannot find other documents like it..
- You found an article lying in your computer or The teacher gave a paper to study
- Instead of trying to think of search queries to get the relevant results..
  - Use our system..
  - And it will find it for you...

- We can identify the correct terms and build an appropriate search query...
  “so that you find what you need.... and not finding what you think you need...”
- A translator/communicator from human generated content to keywords/queries understood by the search infrastructure..

Documents

- Lucene only indexes special document files
- You have to generate the documents on your own
- If we want to index special file formats, we need converters for various file formats

Pre-existing Text Extractors

- Many popular file formats are binary, so normal text extraction doesn't work.
- Luckily, there are many text extraction libraries already written for Java
- Text extraction can be used and added as a field into the Lucene Document type so that the data can be indexed.

Supported File Formats

- Text
- PDF (provided by Multivalent)
- HTML (also provided by Multivalent)
- Microsoft Office 2003 and earlier Formats (provided by POI):
  - Word
  - PowerPoint
  - Excel
- XML (Java XML library)
- Word 2007 Documents (Unzip the file and parse the XML)

SiQuest Framework

- Use what's available
  - Lucene
- Provide a framework
  - Indexes
  - Searches
  - Extracts index data

SiQuest Framework (Cont.)

- SqIndexer
- Uses Lucene index writer
- SqAnalyzerFactory
- Standard
- Stop
- Whitespace
- Simple
- Snowball
- SqTextExtractor
- Interface to extract text from various file formats

SiQuest Framework (Cont.)

- SqInfoExtractor
  - Provides interface to indexed documents
- SqTerm
  - Encapsulates data for indexed term
    - field
    - text
    - frequency
- SqPreferences
  - Contains global and user application preferences

SiQuest Framework Test UI

Query Builder Engine

- Load user preferences for indexing and query building
- Take the given documents as input
- Index those documents
- Extract the top ranking terms
- Generate API specific query string as output

Query Builder Engine: First Pass

- Index the documents using a select analyzer
- Take 25 most frequent terms in the document and “and” them together for the search
- Pass the generated string to query API and evaluate the outcome
- Increase the number of terms with different selection of analyzers, and find the best fit

Query Builder Improvements

- Fine-tune the algorithm based on our findings with the first pass algorithm
- Improvements:
  - Term Frequency (tf)
  - Inversed Document Frequencey (idf)
  - Weighted/Scored Keywords
  - Keywords selection by percentile

Query Builder Testing

- Using too few terms results in documents that are limited in scope and are unrelated.
- Using too many terms results in documents that are almost completely unrelated.
- Experimentally, using around 50 terms has the best fit 20+ source documents

Query Builder Testing (cont'd)

- Standard analyzer (default) gave best result
- A scoring scheme using tf, idf (but inverted) helps pruning out irrelevant keywords
- Using percentile helps getting the most relevant terms depending on the number of source documents

The Searching

- The siQuest Engine generates a query based on the file(s)
- We need to perform a search using this query
- Using the Query, Search on
  - Google
  - Yahoo!
  - Live!
- Return the combined Results

The Methods

- SOAP
  - WSDL File defines the services
  - Calls should be built upon the WSDL Definition
  - All messages should be in well defined XML
- AJAX / REST
  - Ordinary HTTP Request
  - Response has data instead of a formatted html page
  - Response is in Javascript Object Notation

The Applications

- SiQuest Engine
- SiQuest Web Service
- SiQuest RWeb Quest
  - The Pages

SiQuest WebService

- Expose SiQuest Engine functionality as a web service
- Other applications can call the siQuest API
  - Various Consuming applications can be completely independent of siQuest
  - Client App can be in a Different language
- Both Interfaces exposed
  - SOAP / XML
  - REST / JSON
- Written in Java

A SOAP Request to sQ-WS

```
<?xml version="1.0" encoding="utf-8" ?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <env:Body>
   <n1:getResults
xmlns:n1="http://webservice.siquest.villanova.edu/siQuestSOAP/">
<filename>C:/Users/Tarun/Documents/CSC9010/PC2TeamGuide.pdf</filename>
     <analyzer>Standard</analyzer>
     <searchService>GOOGLE</searchService>
     <searchService>YAHOO</searchService>
   </n1:getResults>
 </env:Body>
</env:Envelope>
```

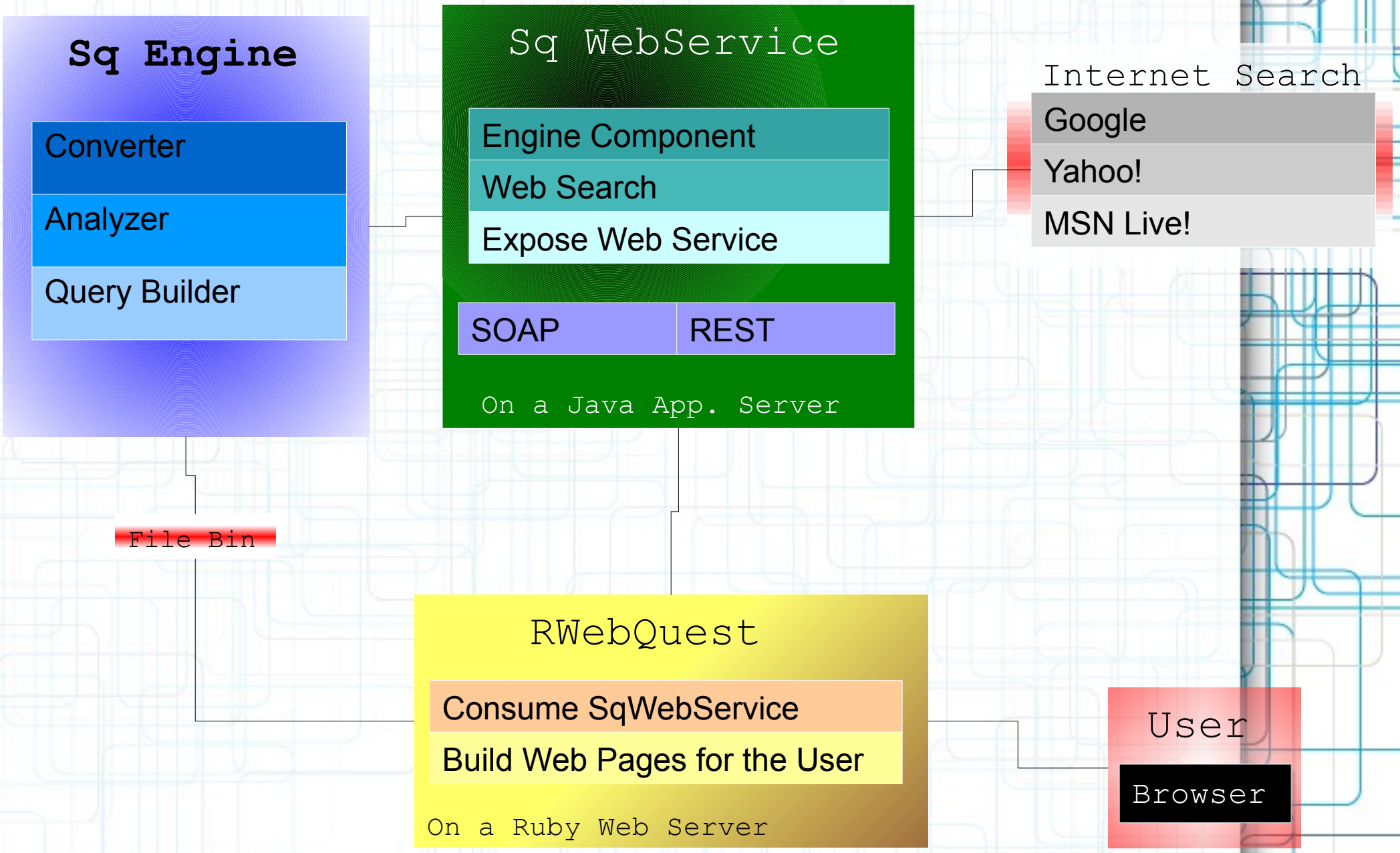
REST / JSON

- A Service
  - SiQuest WSDL
    http://localhost:8080/siQuestService/wsdl
  - A Rest Response from SqQuest
    http://localhost:8080/siQuestService/SiQue
- SOAP is too much overhead for a simple well defined service

SiQuest RWebQuest

- A Ruby On Rails Application
  - Rapid development
- Convention over Configuration (CoC)
  - Productivity booster
- Demonstrates the flexibility of web services
- A Browser interface is intuitive

The Larger Picture



User Interface

- A simple Web User interface.
- Allows user to select/upload the documents.
  - Support uploading multiple files
  - Javascript methods to add multiple files to the request
- Accepts various formats of documents.
- Provides an option of choosing Analyzers.
- Display the results

Tag Cloud

- Tag cloud of input text document
- Makes use of the most important keywords identified from the algorithm.
- CSS styling for tags
  - eg:(span.tagcloud1{font-size:1.4em;padding:0em;color:#ACC1F3;z-index:9;position:relative}
- Distinct Tags based on score of each keyword.
- Style changes based on the score

Mapping Score to Cloud Weights

- Get the terms from the query
- Get the scores for all the terms
- Find the maximum and minimum score
- Calculate factor by
    - Factor = (max - min)/10
- Calculate css weight class by
    - Weight = (score-min)/factor
- Display the words in the tag cloud based on the weight

The Pages

- CSS Styles for the whole page
- The page consists of different sub components
  - each of which are different files
- The pages then get combined before displayed.

Live Demo

Experiences/Lessons Learned

- Indexing various document formats is Complicated
  - Thanks Lucene
  - Thanks various libraries
- Using Lucene isn't a cakewalk either
  - Getting indexing to work properly took effort
- Query Generation
  - lot of thought and tuning
- Multiplatforms (OS, IDEs)
  - Thanks JAVA, WebServices
  - Java and Ruby

Thank You