Building Search Applications with MarkLogic Server

Ron Avnur
Mark Logic Corporation
ron@marklogic.com

ABSTRACT

Keyword search is recognized as an important technique to unlocking the information found in both structured and semi-structured information. With XML as the data model and XQuery as the programming language, MarkLogic Server[1] allows developers to build search into their information-centric applications.

We are increasingly interested in lowering the learning curve of application development. This demo will show a tool that interrogates a corpus of information, presents a user interface to define the behavior of an application, and then compiles and deploys a search application over a set of XML documents.

1. Introduction

MarkLogic Server is a flexible platform for building applications that unlock the valuable information found in semi-structured and unstructured documents. Using XML as the underlying data format, the system can easily ingest and query documents of varying shapes and sizes[2]. We commonly see keyword search queries employed in applications built with MarkLogic.

Developers use XQuery[3] to build their applications and to query or interact with the data they've loaded into the system. Keyword search queries are facilitated by the Full Text extensions of XQuery[4]. While an easy language to learn, we're always interested in providing a lower barrier to querying the database.

The common use of keyword search in applications is detailed in section 2, where we discuss common design patterns and application features that we aim to generalize. In section 3 we then describe the options that our new tool, Application Builder, aims to present its end-users. Last we summarize the demonstration that will illustrate installing the system, loading content, configuring options, and enjoying a deployed keyword-search application.

2. Keyword Search Applications

Reviewing various keyword search applications or systems, one may find the following common components:

1. Search Grammar: A collection of rules for applying boolean operators to help a user express complex keyword constraints. For example: "cat OR dog" to find all documents that contain either the word "cat" or the word "dog".
2. Multiple sorting options: The ability to order results by various options. Each option consists of a combination of term-frequency based scoring algorithms (such as TF/IDF) and metadata about each result item (such as publication date).
3. Declarative search constraints: Uniquely identifying subsets of the corpus for finer-grained search. An application may generally search for keywords anywhere within a document, but users sometimes want to focus a query to restrict against only a slice of every document. For example, given a search for "summary:performance" a grammar may search for the word *performance* within only the *summary* field, where the field is defined as text within documents that is a descendant of either elements named "abstract" or "synopsis."
4. Rendering of search results: Results are displayed in a summary format to help a user choose which item she'd like to see more details about. This information may include a snippet of the narrative accompanied by metadata about the document.
5. Result set characterization: Given a result to a search, query systems typically share information about the results to help the user learn about the set. At the very least, systems typically indicate how many results matched a given criteria and display some top *k* items. In addition, they may also display aggregate information on various axes (or columns). For example, a search across books may display unique authors, editors, publication decade, and price. Some aggregates list all the unique values with frequencies, such as authors, whereas others identify buckets of values, such as price along with the aggregate frequency of the bucket.

Our tool aims to generalize these common search application features, proving a user interface for specifying desired behavior as input to a compiler that generates the XQuery-based application.

3. Application Builder

This section described the six tabs that separate the components of the Application Builder, where the title of each sub-section matches the name of the tab.

3.1 Appearance

The Appearance tab lets users select a name, logo, and other metadata for the application. They can also select a skin for the application, applying a common style sheet to give the application a common visual theme.

KEYS'09, June 28, 2009, Providence, Rhode Island, USA.

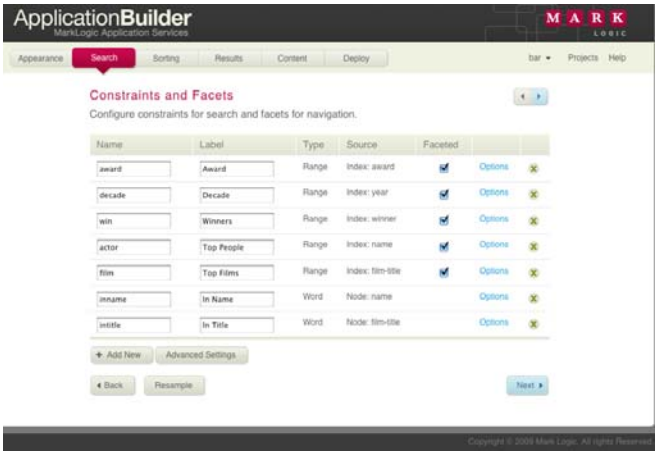

Figure 1. The search tab of the Application Builder

3.2 Search

The search tab allows the user to customize a default search grammar, select components of the content to use to help characterize results, and identify constraints.

In addition to customizing behavior of boolean operators, the user can customize search behavior with respect to case, punctuation, and diacritic sensitivity. She can also choose to apply language-specific stemming to search terms, increasing recall for search terms. Wildcard searches can also be controlled in this section.

The application interrogates the content and system configuration to identify which elements in the corpus can be used to characterize result sets. Given the system configuration, elements that have been configured with scalar indexes are considered eligible because any aggregates and grouping based on these elements can be applied efficiently (without fetching each document off disk). The user can then choose which elements to use in rendering characterization information.

3.3 Sorting

The application designer may want to allow users to sort results by several methods. Sort options include scoring algorithms based on the popularity of terms in documents versus the database (such TF/IDF) as well as any elements containing scalar information. Each scoring option can be composed of one or more relevance-based or scalar-based components. As an example, a developer can provide an ordering that uses a publication date as the primary and relevance score as the secondary sort ordering.

3.4 Results

The results screen of a search application typically displays a subset of the search results with options to move the display window forward or backwards within the result set. The information displayed for each result can be configured to select result titles, metadata displayed, and representation for a snippet of the content in the result item. To facilitate design, the system renders some sample content according to the choices selected as a form of preview, providing visual feedback to the designer.

3.5 Content

As an end-user browses the results, she may choose to click on a result in order to view a detailed rendering of the selected item. The content tab let's the application designer configure rendering options that will determine how a result is displayed for end-users in the deployed application. A preview of sample content is provided in this tab as well.

3.6 Deploy

The last tab consists mainly of a "deploy" button that passes all the options selected to an application compiler that generates the XQuery application and deploys it within a MarkLogic Server environment for use. End users can then be directed to this application to begin interacting with and searching across the information.

4. Demonstration

Our demonstration will illustrate building an application without writing any code. We will first illustrate how to load content into the system without needing any scripting, programming, or a schema. We will then open the application builder and choose search criteria, sorting options, result set presentation, and content rendering. We will then deploy the application, review it, and perhaps modify the application configuration and re-deploy.

Building applications so quickly can help content experts first understand and analyze their content set in order to facilitate building information-centric applications with keyword search at their core. The application generated by Application Builder is built to scale. While the data loaded during the presentation may not be large, this application can scale to deployments with many millions of documents spread across a large shared-nothing cluster of machines running MarkLogic Server.

5. REFERENCES

[1] MarkLogic Server. DOI= http://marklogic.com/product/marklogic-server.html.

[2] Holstege, M. 2008. Big, Fast, XQuery: Enabling Content Applications. IEEE Bulletin of the Technical Committee on Data (December 2008) Vol. 31 No. 4. 41-48.

[3] Boag S. et al. (editors) . XQuery 1.0: An XML Query Language. W3C Recommendation. W3C. January 2007. DOI= http://www.w3.org/TR/xquery/

[4] Amer-Yahia S. et al. (editors). XQuery and XPath Full Text 1.0. W3C Candidate Recommendation. W3C. May 2008. DOI=http://www.w3.org/TR/xpath-full-text-10/