INLS 490-154: Introduction to Information Retrieval System Design and Implementation. Fall 2008.

10. User Interface for Search

Chirag Shah*
School of Information & Library Science (SILS)
UNC Chapel Hill NC 27514
chirag@unc.edu

1 Introduction

This time we will see how the back-end that we have been working with for retrieval can be extended to the user with a front-end interface. We will do so by employing Indri, a new search engine for Lemur, and creating web-based interface that a user can access using a browser.

2 Using Indri

Indri, another open source project from UMass and CMU groups, has been incorporated into Lemur Toolkit project and allows one to focus on creating search services to connect to the back-end that may be created with some of the tools we saw before with Lemur. In addition to this, Indri is also designed to support huge collections and a large number of queries. The way Indri works is by having a client-server architecture. The server is implemented using an application called 'indrid'. Code 1 shows a parameter file for this application.

Code 1: Parameter file for 'indrid'

```
<parameters>
    <index>myindex</index>
    <port>12345</port>
</parameters>
```

Running this application does nothing, but start a server that starts listening to on the port specified in the parameter file. You can run this on the terminal and possibly, keep it running in the background. As you can see, Indri can use the index created by Lemur.

Once Indri daemon is running, you can use an Indri application called 'runquery' to execute a query. A sample parameter file for 'runquery' is shown in Code 2.

Code 2: Parameter file for 'runquery'

```
<parameters>
    <index>myindex</index>
    <server>localhost:12345</server>
    <count>10</count>
    <trecFormat>false</trecFormat>
    <printSnippets>true</printSnippets>
    <query>#combine(behavioral genetics)</query>
</parameters>
```

Running server and client as different processes allows us to separate them on different machines, if needed. As we can see, the parameter file for 'runquery' has an option of specifying the address and the port of the search server. Thus, one can even run multiple servers, possibly representing different collections or indices, even on the same machine.

Application 'runquery' prints a rank-list along with snippets for each of the results (given they are enabled in the parameter file). However, this output is not yet ready for the user. We need to address two issues: (1) converting those snippets into HTML format, and (2) extracting the document texts.

Let us address the first issue. To do so, open 'runquery.cpp' file found in 'runquery' folder in your Indri distribution and find a line that says

`indri::api::SnippetBuilder builder(false);`

Changing `false` to `true` in the above line formats the snippets in HTML. Make sure to compile your modified application by running 'make' in the root directory of Indri.

Now, let us look at extracting the document text for a results. We have already seen how to do so using a Lemur application called 'dumpindex'. However, 'dumpindex' expects a document ID that is internal to the index and we do not have it from the output that 'runquery' generates. We can get around this problem by first using 'dumpindex' to find the associated internal ID for a document giving the document name that 'runquery' gives us as shown below.

```
dumpindex <index_name> di docno <doc_name>
```

The above command will output a number, which is that document's internal ID in the given index. Now we can use that ID to extract the actual text using the following command.

```
dumpindex <index_name> dt <doc_id>
```

If you want to avoid using 'dumpindex', you can modify 'runquery' application itself to return you the results in the format you want, including the full text. To do so, once again open 'runquery.cpp' and find a function named `_printResultRegion( )`. This function has lines to print the results. The results are stored in an array called `documents[]`. You can iterate over this array and use pointers to their texts (`documents[i]->text`).

3 Developing a search interface

Let us turn our attention to bringing these background services to the user by developing a front-end. We will leave out the details, but here are the steps that we will execute.

1. Start the indri daemon in the background using 'indrid'.
2. Create an HTML form with a query box and a submit button.
3. Use the query to create a parameter file for runquery on the fly.
4. Execute 'runquery'.
5. Parse the results generated by 'runquery'.
6. Display results on the interface.

Following the above steps will create a web-interface that lets a user execute a query and get a number of results back. We can enhance this interface by adding dynamic interactions to it. Once again, we will leave out the details, but simply point to a technology called AJAX for enabling such dynamic interfaces. AJAX stands for Asynchronous JavaScript And XML, and does not require any additional support or software than a JavaScript enabled web-browser on the client side and a server-side technology, such as PHP, on the server. Even though the name AJAX includes XML, in practice, XML is not required.

A typical architecture of AJAX is shown in Figure 1. As we can see, on the client side, the presentation is done using HTML and CSS. In this presentation of the page, several components are included. If we have some code associated with a component and it is set to evoke a JavaScript call based on an event, such as key press or mouse over, that component can be made to change its behavior or do some function independent of the rest of the components. This allows one to asynchronously update all the components on a given page.

In the figure shown, as a JavaScript function is called by a component, it creates an object of type XMLHttpRequest. This object is sent to the server, where the server-side pages process the object and respond using XML or other kind of data. Function `callback()` on the client side then uses that returned data and renders it for the component that issued the request. While all these processing is taking place, the rest of the components on the page are not disturbed. This allows us to update a given component without reloading the entire page and that in turn helps in providing a dynamic user interface.

4 Summary

- Indri is a new search engine from the Lemur project.
- We can create back-end with Lemur or Indri using applications such as 'BuildIndex' or 'IndriBuildIndex', and 'indrid'.
- We can provide front-end with Indri using application 'runquery' and connecting it through a web-server.

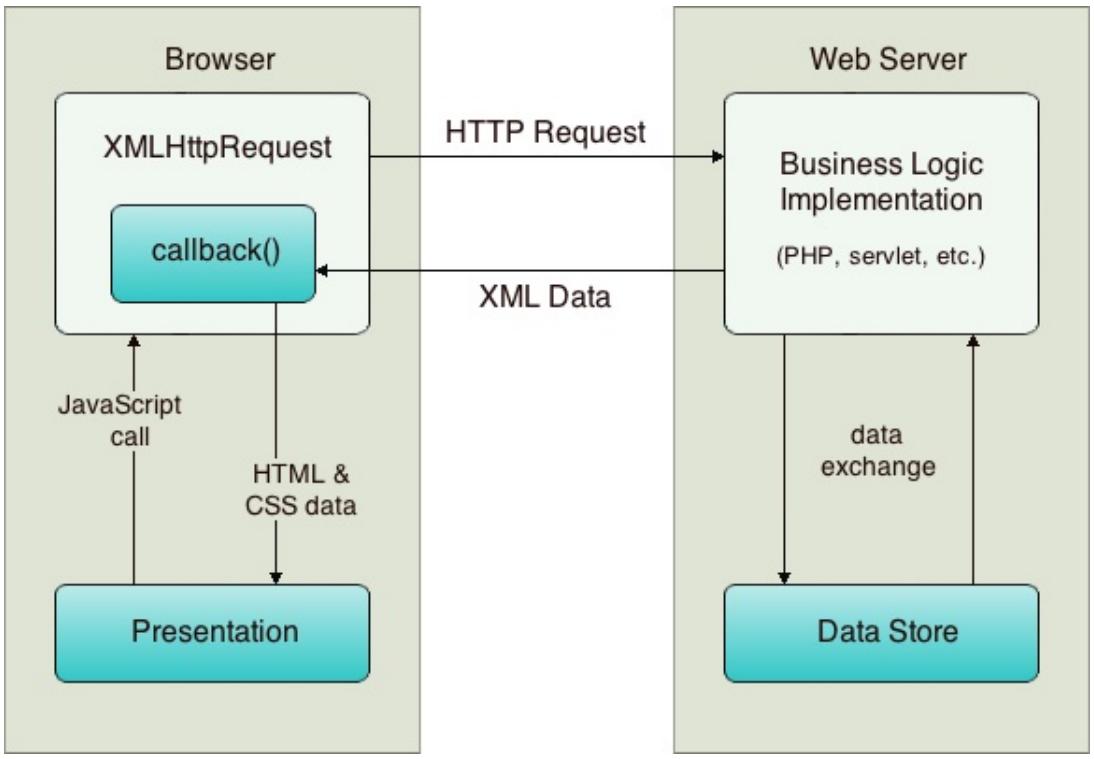

Figure 1: Creating a campaign

- AJAX helps in creating dynamic user interfaces that allows us to asynchronously update the components of a page without reloading the entire page on the client side.