The Technology of Lexical Navigation

James W. Cooper
IBM Thomas J Watson Research Center
PO Box 704
Yorktown Heights, NY 10598
914-784-7285
jwcnmr@watson.ibm.com

ABSTRACT

Lexical Navigation provides users a convenient technique for moving between related documents and terms within a collection without ever having to formulate an exact query to retrieve these related entities. It consists of a visual interface client, a server and a back end index and database. We discuss how these components are constructed and utilized to provide useful feedback to the user on additional related information that may be helpful in his information retrieval.

Categories and Subject Descriptors

H5.2 [**Information Interfaces and Presentation**] User Interfaces – *Graphical User Interfaces, Natural language, Theory and methods, Windowing systems.*

General Terms

Algorithms, Management, Measurement, Design, Experimentation, Human Factors.

Keywords

Text mining, Search, Document display, Databases, Lexical navigation, Client-Server, SOAP, XML, Java, JavaScript, JavaServer Pages.

1. INTRODUCTION

We have previously described the concept of Lexical Navigation [1] and the layout algorithms for the representation of a lexical network [2]. In this paper, we discuss the browsing interface and the technical underpinnings that make a responsive navigation system that can approach the ideal of query-free document retrieval.

Our system is constructed using the Textract text mining system that recognizes names [3] and multiword technical terms [4] and relations between them [5], a search engine and a relational database.

In this discussion, we will describe how we constructed a lexical navigation system for 824 documents describing the 200 most prescribed drugs, as obtained from rxlist.com. There are five documents for each drug, but there are fewer than 1000 documents because some drugs appear under more than one name. The system is in no way limited to such small collections, but this collection merely provides a convenient and interesting set of publicly available example documents.

We start with this collection of drug documents and run the Textract processor on this collection. This gives us

- A file of the terms in the entire collection, reduced to their canonical forms.
- A file of the terms in each document.
- Files containing the named and unnamed relations [5] discovered in the collection
- A pair of files representing a concordance of all the sentences containing salient terms in the entire collection.

We construct a DB2 database consisting of the following tables:

- Documents
- Terms
- TermDocs (Terms per document)
- Relations

We load the Documents table with a series of document key numbers, along with the title and URL of each document. We load the Terms table from the collection-wide aggregation of terms in canonical form along with their frequencies and IQs, where IQ is a measure of term selectivity. Terms having a high IQ appear only in a few documents.

We also load the TermDocs table with the terms in each document, where each document is referenced using the integer document key from the Documents table. By putting these data into a database where we can fetch them rapidly, we can look up the principal multi-word terms in a document or the documents which contain any specified term.

The Relations table contains both named and unnamed relations discovered by Textract, as well as the computed strengths of the unnamed relations. The unnamed relations are computed using a mutual information calculation and the named relations discovered pattern recognition on each document, looking for common patterns such as appositives and parentheticals. The named relations detection algorithm finds common relations like “CEO of,” “is located in”, “makes,” and “similar-to.” These sort of relations are more likely to be detected in news story writing than in technical writing, and so the unnamed relations become considerable more important in technical documents.

We construct the Relations database table to contain both of the related terms, the strength of the relation and the relation name or “none” for unnamed relations. Named relations are assigned a strength of “100” automatically. The weight of the unnamed relations comes from the mutual information computation.

Textract also produces a concordance of all the sentences containing salient terms in the entire collection. From this concordance, we build a special search index called the *context thesaurus* that allows us to provide a list of terms in the collection that occur near the query phrase. This index is similar to and inspired by the Phrase Finder of Xu and Croft [7]. We construct the context thesaurus by constructing a pseudo-document of the sentences surrounding each term anywhere in the collection and index that document, giving the document that term as the title. We discard each pseudo-document after indexing it.

In addition to the database of terms discovered by Textract, we also index all of the documents using a standard search engine. Currently are using IBM's GTR search engine for this purpose. This same engine is used to index the pseudo-documents for the context thesaurus described above.

2. THE SERVER STRUCTURE

The data management system is constructed using our Java class library called KSS (for Knowledge System Server), which is effectively a Façade design pattern wrapping access to the underlying DB2 database and the GTR search engine. Then a non-visual server-side bean is used to make calls to the KSS library.

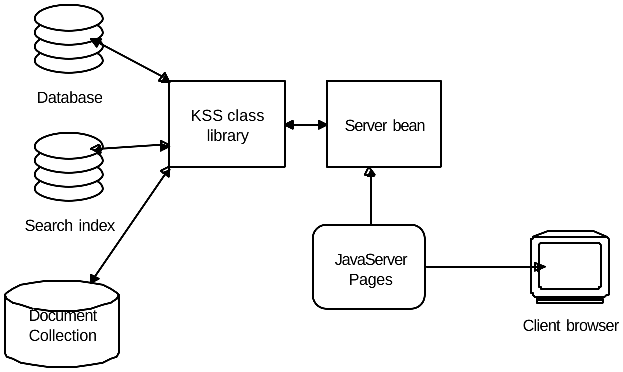


Figure 1 – The logical entities used in Lexical Navigation

The methods of this server-side bean are accessed by JavaServer Pages, which generate the HTML pages as shown in Figure 1.

3. THE LEXICAL NAVIGATION SCENARIO

A typical user starts with a simple, short query (Figure 2)

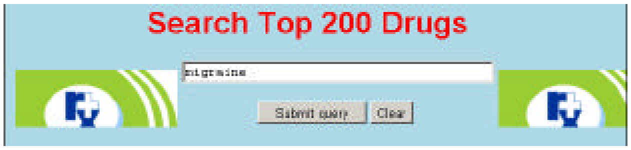
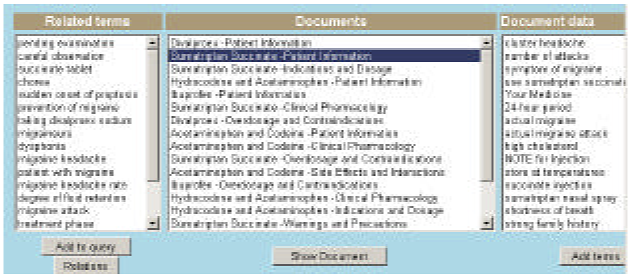


Figure 2 – The initial query screen

The initial query page is a simple JavaScript form with a single field named "query." When the form is sent the web server, the servlet engine loads the JavaBean and executes the bean's setQuery method. This causes Java code to be executed that makes two calls to the search engine, one against the document index and one against the context thesaurus index. Then, for each document, the bean looks up the most salient terms in each document and caches them for output.

The returned JSP presents both a list of documents and a list of context thesaurus terms that occur near that query in the collection. You can then narrow or focus the query using some of these suggested terms. These terms provide an entry to the actual terms we have recognized in the collection rather than terms we might hope to find in the collection and thus provide a much more accurate way of refining a query.

In addition, you can click on each of the document titles and see the highest-ranking terms that appear in them. This is a sort of simplistic document summary, but it has the additional advantage that it can be produced even from documents where traditional summarization techniques fail, since it is not dependent on paragraph structure. You can also add these terms to the query to find "more documents like this." We show a typical result in Figure 3.



pending examination
careful observation
succinate tablet
chorea
sudden onset of prophesis
prevention of migraine
taking divalproex sodium
migrainours
dysphoria
migraine headache
patient with migraine
migraine headache rate
degree of fluid retention
migraine attack
treatment phase

Divalproex - Patient Information
Sumatriptan Succinate - Patient Information
Sumatriptan Succinate - Indications and Dosage
Hydrocodone and Acetaminophen - Patient Information
Ibuprofen - Patient Information
Sumatriptan Succinate - Clinical Pharmacology
Divalproex - Overdosage and Contraindications
Acetaminophen and Codeine - Patient Information
Acetaminophen and Codeine - Clinical Pharmacology
Sumatriptan Succinate - Overdosage and Contraindications
Acetaminophen and Codeine - Side Effects and Interactions
Ibuprofen - Overdosage and Contraindications
Hydrocodone and Acetaminophen - Clinical Pharmacology
Hydrocodone and Acetaminophen - Indications and Dosage
Sumatriptan Succinate - Warnings and Precautions

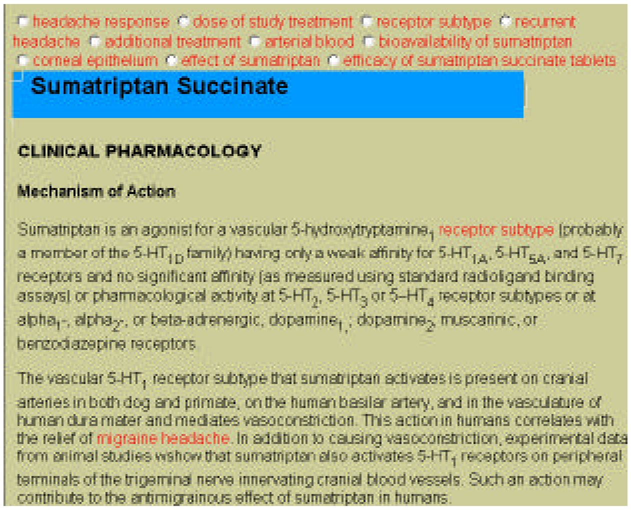
cluster headache
number of attacks
symptoms of migraine
see sumatriptan succinat
Your Medicine
24-hour period
actual migraine
actual migraine attack
high cholesterol
NOTE for Injection
store at temperatures
succinate injection
sumatriptan nasal spray
shortness of breath
strong family history

Figure 3 – The result of the query "migraine" in a collection of prescription drug information. The left column contains the co-occurring terms, the middle column the document titles, and the right column the principal terms in the selected document.

The application shown in Figure 3 is a web page that is generated using JavaServer pages on the web server. This has the advantage of not being affected by firewalls and requiring limited resources on the client system.

This output page is created from a second JSP that contains the three Select lists shown. However, the rightmost list is not filled with specific data in advance. Instead, the data for filling it are stored in a JavaScript array of arrays of terms, one for each document. When you click on any document, the array of terms for that document is loaded into the list using a JavaScript function.

You can then view any document you select by clicking on a "Show Document" button under the document listbox. However, rather than just showing the document itself, the server bean takes the top 10 terms found in the document and converts them into hyperlinks which allow you to perform additional queries using them. It also puts these top terms in a summary list at the top of the document. This is similar to the Active Markup we described previously [8], and is illustrated in Figure 4. These term highlightings are constructed dynamically in Java within the server-side bean, and thus are suited for a changing environment, where the updated documents can be fetched from the server. This differs from our original Active Markup procedure in that no Java classes are invoked on the client: all of the computation is carried out on the server and returned in HTML.



○ headache response ○ dose of study treatment ○ receptor subtype ○ recurrent headache ○ additional treatment ○ arterial blood ○ bioavailability of sumatriptan ○ corneal epithelium ○ effect of sumatriptan ○ efficacy of sumatriptan succinate tablets

Sumatriptan Succinate

CLINICAL PHARMACOLOGY

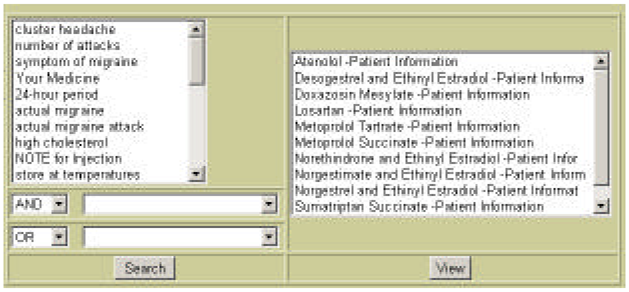
Mechanism of Action

Sumatriptan is an agonist for a vascular 5-hydroxytryptamine$_1$ receptor subtype (probably a member of the 5-HT$_{1D}$ family) having only a weak affinity for 5-HT$_{1A}$, 5-HT$_{5A}$, and 5-HT$_7$ receptors and no significant affinity (as measured using standard radioligand binding assays) or pharmacological activity at 5-HT$_2$, 5-HT$_3$ or 5–HT$_4$ receptor subtypes or at alpha$_1$-, alpha$_2$-, or beta-adrenergic, dopamine$_1$; dopamine$_2$; muscarinic, or benzodiazepine receptors.

The vascular 5-HT$_1$ receptor subtype that sumatriptan activates is present on cranial arteries in both dog and primate, on the human basilar artery, and in the vasculature of human dura mater and mediates vasoconstriction. This action in humans correlates with the relief of migraine headache. In addition to causing vasoconstriction, experimental data from animal studies wshow that sumatriptan also activates 5-HT$_1$ receptors on peripheral terminals of the trigeminal nerve innervating cranial blood vessels. Such an action may contribute to the antimigrainous effect of sumatriptan in humans.

Figure 4 – The document display for Sumatriptan Succinate with salient terms summarized at the top and highlighted throughout.

You can then click on any highlighted term or on any of the radio buttons at the top of the document and see a display of all the salient terms in that document in a list box. You can select any term and ask to see other documents containing that term. Here the server-side bean simply makes a database query against the TermDocs database table to retrieve this information. In addition, as shown in Figure 5, you can form a simple boolean query to find documents containing any combination of terms as well.



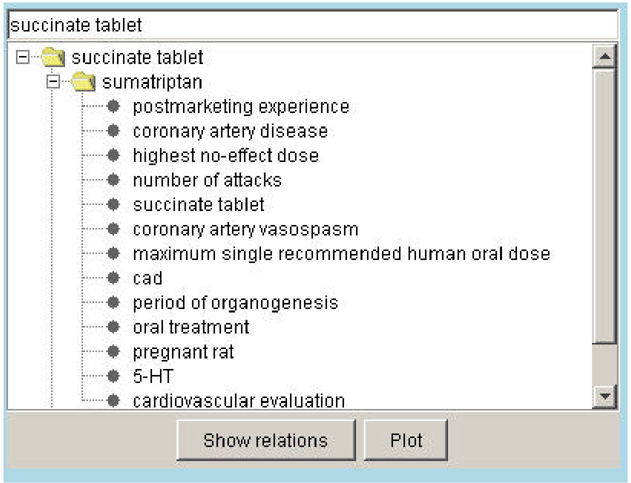
cluster headache
number of attacks
symptom of migraine
Your Medicine
24-hour period
actual migraine
actual migraine attack
high cholesterol
NOTE for Injection
store at temperatures

Atenolol -Patient Information
Desogestrel and Ethinyl Estradiol -Patient Informa
Doxazosin Mesylate -Patient Information
Losartan -Patient Information
Metoprolol Tartrate -Patient Information
Metoprolol Succinate -Patient Information
Norethindrone and Ethinyl Estradiol -Patient Infor
Norgestimate and Ethinyl Estradiol -Patient Inform
Norgestrel and Ethinyl Estradiol -Patient Informat
Sumatriptan Succinate -Patient Information

Figure 5 – The major terms in the original document are shown on the left, and the documents containing the selected term are shown on the right.

4. DISPLAYING RELATIONS

You can use the Context Thesaurus table to provide a definitive entry to terms that are actually in the collection. This is particularly useful if you want to investigate the space of lexical relations. Once you select a term from the Context Thesaurus table, you can query the database of related terms for all those related to the selected term. We can represent how terms are related using a Java tree list as shown in Figure 6.



succinate tablet

succinate tablet
sumatriptan
postmarketing experience
coronary artery disease
highest no-effect dose
number of attacks
succinate tablet
coronary artery vasospasm
maximum single recommended human oral dose
cad
period of organogenesis
oral treatment
pregnant rat
5-HT
cardiovascular evaluation

Figure 6 – A Java TreeList of relations between terms

The term relations are kept in the Relations table we described above. Since the left and right term relations are not duplicated as right and left terms, two queries are needed. The results are then combined and reduced in a hash table. The relations are returned as an array of Java objects which are instances of a class we call Relns. This class provides accessor functions for obtaining the term names, strength and relation name, and is just a special case of the Relations object we used earlier, but with public getter and setter methods for each internal parameter. The SOAP (Simple Object Access Protocol) serializer and deserializer then use Java introspection and these methods to construct the XML data stream and reconstruct the object at the other end of the wire [9].

In our earlier papers [2, 6], we returned these objects using Java Remote Method Invocation (RMI). However, Java RMI had not been widely accepted and may not be compatible with all browsers and with firewalls. Consequently, we redesigned this system using a SOAP web service that transmits these same Relns objects as a stream of XML data. These data are then reassembled as Relns objects using the SOAP deserializer classes within the Java applet.

This SOAP system provides a powerful way of exchanging fairly complex Java objects across disparate clients and networks, using HTTP or other well-accepted protocols.

An outline of the Java applet and the Relns object is shown in Figure 7.

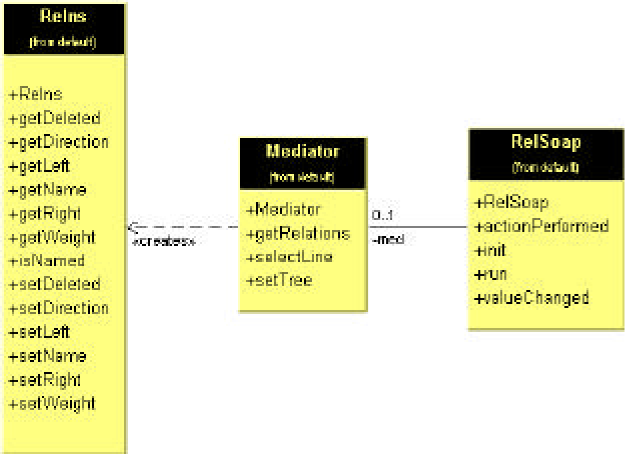


Figure 7 – A simplified UML diagram of the Relns object and the RelSoap Java applet.

We can then generate a plot of these relations using the incremental graphical layout algorithms developed by Tunkelang [6]. We show such a plot in Figure 8. Note that our system represents both named and unnamed relations between terms.

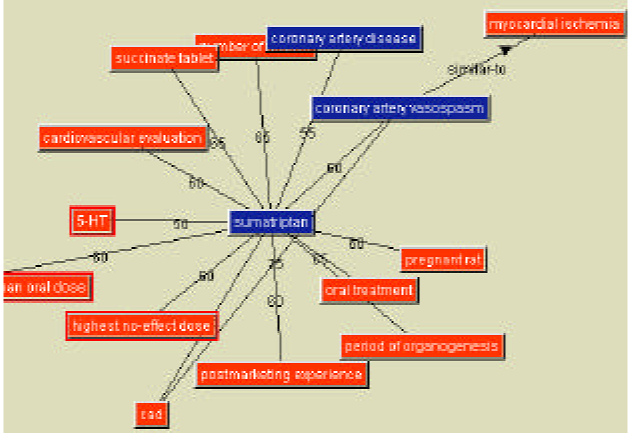


Figure 8 – A graphical plot of the relations between terms in a collection, allowing term browsing.

This display is a client-side Java applet that looks at each object and asks whether the relation is named or unnamed. If unnamed, it displays the strength of the relation. If named, it displays the name of the relation. We further represent named relations as directional using an arrowhead, and unnamed relations as bi-directional, by using no arrowhead. Double clicking on any frontier term causes a query back to the server to ask if that object has further relations. If it does they are returned in an array of Relns objects as before. Otherwise a zero-length array is returned. In either case, the display changes the color of the box containing that term, indicating that it has now been expanded. The graphical layout is recomputed each time to minimize overlap between the terms.

From the representation in Figure 8, it is possible to browse through the Relations collection and discover term relations that reveal information contained in disparate documents very efficiently. It is then also possible to select terms to either add to queries or to view a list of documents containing those terms or that relation, since all of that information is stored in the database. It is also possible to view the documents containing that term as we showed in Figure 5.

5. CONCLUSION

We have used the Textract text-mining engine, along with a database, a web server and SOAP-based web services to allow a user to navigate through the documents, terms and relations in a collection. This approach allows a user to discover a significant amount of information regarding that collection without ever having to formulate a sophisticated query.

6. REFERENCES

[1] Cooper, J. W. and Byrd, R J, "Lexical Navigation: Visually Prompted Query Refinement," ACM Digital Libraries Conference, Philadelphia, 1997.

[2] Cooper, James W. and Byrd, Roy J., OBIWAN – "A Visual Interface for Prompted Query Refinement," Proceedings of HICSS-31, Kona, Hawaii, 1998.

[3] Ravin, Y. and Wacholder, N. 1996, "Extracting Names from Natural-Language Text," IBM Research Report 20338.

[4] Justeson, J. S. and S. Katz "Technical terminology: some linguistic properties and an algorithm for identification in text." *Natural Language Engineering*, 1, 9-27, 1995.

[5] Byrd, R.J. and Ravin, Y. Identifying and Extracting Relations in Text. *Proceeedings of NLDB* 99, Klagenfurt, Austria.

[6] Tunkelang, D. D., Byrd, R. J., and Cooper, J. W., "Lexical Navigation: Using Incremental Graph Drawing for Query Refinement," Graph Drawing 97.

[7] Xu, Jinxi and Croft, W. Bruce. "Query Expansion Using Local and Global Document Analysis," *Proceedings of the 19th Annual ACM-SIGIR Conference*, 1996, pp. 4-11

[8] Neff, Mary S. and Cooper, James W. Document Summarization for Active Markup, *in Proceedings of the 32nd Hawaii International Conference on System Sciences*, Wailea, HI, 1999.

[9] Cooper, James W. "Soaping Your Windows," *JavaPro*, May, 2001.