LexEVS 6.1 Design Document - Detailed Design - Performance - Text Search (Contains)

Contents of This Page

Document Information

Author: Bauer, Scott
Email: Bauer.Scott@mayo.edu
Team: LexEVS
Contract: ST12-1106
Client: NCI CBIIT
National Institutes of Heath
US Department of Health and Human Services

Revision History

Version	Date	Description of Changes	Author
1.0	2013/03/05	Initial Version	Bauer, Scott

Overview

Search mechanisms using Lucene can become overburdened as increasing numbers of use cases make for a larger and more complex Lucene index. When a use case as comprehensive as searching an entire multi-source terminology service for a term or fragment is proposed, then a more restricted manner of approaching the problem is in order. Furthermore the use of the power of the LexEVS API becomes overkill, with the necessity of the union of large datasets prior to the effective query. Subsequently, a more restricted approach is necessary.

A Simplified Search API

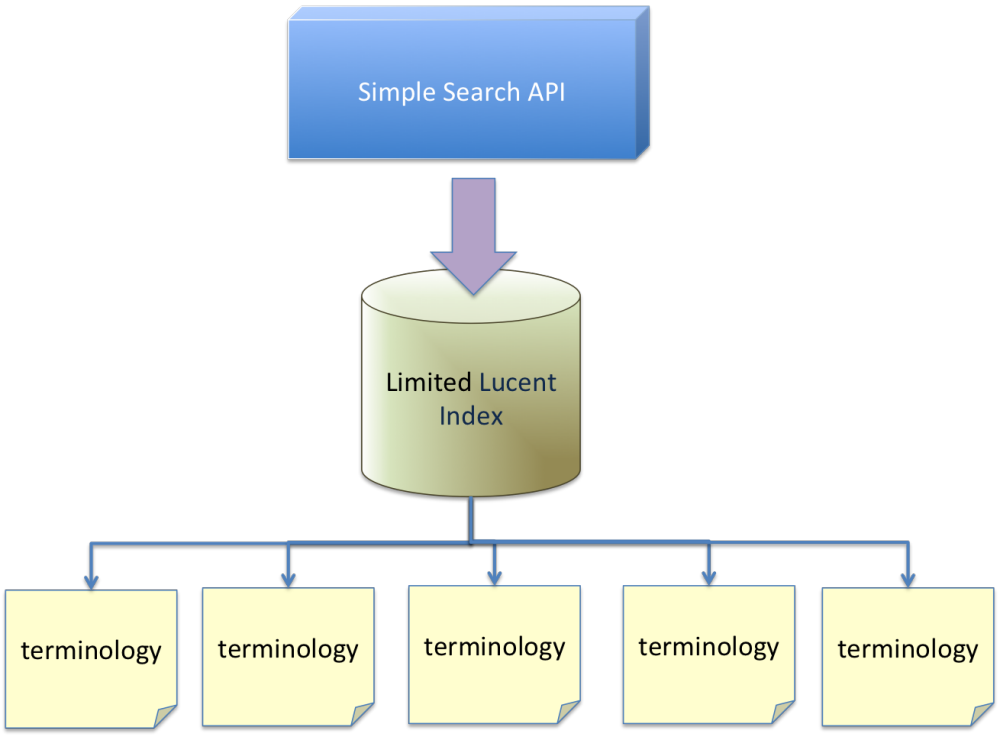
The model for Entity in LexEVS/LexGrid is robust and complete enough that it allows extensive metadata and properties to be compounded within the entity's scope when loaded from an extensively defined source into the LexGrid data model. However, the lexical, or human readable aspects of this model element are relatively restricted and among those aspects that are human readable few need to be searched in order to return results that fit into the majority of use cases. Reducing the matching algorithms available to a text match in these circumstances can also allow relatively good performance over a larger data set in Lucene. The proposition here is to use a "contains" match algorithm only.

A Higher Performance Lucene

Lucene indexes can be created to perform searches in a wide variety of text match algorithms, natural language processing paradigms, customized normalization methods, regular expression implementations and more. The resulting indexes can can be large and complex enough to slow result returns on a given Lucene query. Breaking out a restricted Lucene index using the simplest of these could provide needed speed in result returns.

Multi-Terminology Searches

No matter what the implementation, searching sources that have millions of records is challenging. Searching several sources of size can be even more so. Combining a restricted-search API with a well-tuned Lucene index should make the results-getting performance much more efficient.



This functionality will be implemented by the Search Extension - as described below.

```
/*
* Copyright: (c) 2004-2013 Mayo Foundation for Medical Education and
* Research (MFMER). All rights reserved. MAYO, MAYO CLINIC, and the
* triple-shield Mayo logo are trademarks and service marks of MFMER.
*
* Except as contained in the copyright notice above, or as used to identify
* MFMER as the author of this software, the trade names, trademarks, service
* marks, or product names of the copyright holder shall not be used in
* advertising, promotion or otherwise in connection with this software without
* prior written authorization of the copyright holder.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package org.LexGrid.LexBIG.Extensions.Generic;

import java.util.Set;

import org.LexGrid.LexBIG.Exceptions.LBParameterException;
import org.LexGrid.LexBIG.Utility.Iterators.ResolvedConceptReferencesIterator;

/**
 * A simplified Search Extension.
 *
 * Query syntax is described by the
 * {@link http://lucene.apache.org/core/old_versioned_docs/versions/2_9_1/queryparsersyntax.html Lucene Query
Syntax}
 */
public interface SearchExtension extends GenericExtension {
```

```
    /**
     * Search based on a given text string over all coding schemes.
     *
     * @param text
     *             The search text
     * @return
     *             A ResolvedConceptReferencesIterator
     *
     * @throws LBParameterException
     */
    public ResolvedConceptReferencesIterator search(String text) throws LBParameterException;

    /**
     * Search based on a given text string over given coding schemes.
     *
     * @param text
     *             The search text
     * @param codingSchemes
     *             The coding schemes to include in the search
     * @return
     *             A ResolvedConceptReferencesIterator
     *
     * @throws LBParameterException
     */
    public ResolvedConceptReferencesIterator search(
            String text,
            Set<CodingSchemeReference> codingSchemes) throws LBParameterException;

    /**
     * Search based on a given text string over given coding schemes, excluding
     * the listed.
     *
     * NOTE: If a coding scheme appears in both codingSchemesToInclude
     * and codingSchemesToExclude, the exclude will be given priority.
     *
     * @param text
     *             The search text
     * @param codingSchemesToInclude
     *             The coding schemes to include in the search
     * @param codingSchemesToExclude
     *             The coding schemes to include in the search
     * @return
     *             A ResolvedConceptReferencesIterator
     *
     * @throws LBParameterException
     */
    public ResolvedConceptReferencesIterator search(
            String text,
            Set<CodingSchemeReference> codingSchemesToInclude,
            Set<CodingSchemeReference> codingSchemesToExclude) throws LBParameterException;

}
```

Query Syntax

Query format of the Search Extension follows the Lucene Query Syntax. Any search string allowed by this syntax is accepted by the Search Extension.

By default, all search terms will be joined by an "AND" operator, unless otherwise specified.

For instance, a search of "Heart Attack" will by default translate to "Heart AND Attack"

The following characters are stripped and not indexed. This means that if passed in as a search string, they will not play a role in matching of a term.

For example, "@heart" and "$heart" will both be indexed identically – as "heart"

Characters to remove:

',', '.', '/', '\', '`', '\'', '"', '+', '*', '=', '@', '#', '$', '%', '^', '&', '?', '!'

The following characters will be translated into whitespace during indexing

This means words will be broken and indexed separately.

For example, "Heart-Attack" will be indexed separately as "heart" and "attack"

Characters treated as whitespace:

'-', ';', '(', ')', '{', '}', '[', ']', '<', '>', '|'