STARTS

Stanford Protocol Proposal for Internet Retrieval and Search

Last modified: August 28, 1996

Luis Gravano, Kevin Chang, Hector Garcia-Molina, Andreas Paepcke
Digital Library Project
Stanford University

Document sources are available everywhere, both within the internal networks of organizations and on the Internet. The source contents are often hidden behind search interfaces. These interfaces vary from source to source. Also, the algorithms with which the associated search engines rank the documents in query results are often incompatible across sources. Even individual organizations use search engines from different vendors to index their internal document collections. These organizations could benefit from unified query interfaces to multiple search engines, for example, that would give users the illusion of a single combined document source. Building *metasearchers* (i.e., services that provide such a unified view of the multiple sources) is nowadays a hard task because different search engines are largely incompatible and do not allow for interoperability.

Given a query, a metasearcher has to perform (at least) three tasks to provide a unified interface over a (large) number of document sources:

- Choose the best sources to evaluate the query
- Evaluate the query at these sources
- Merge the query results from these sources

The existing search engines do not help with the three tasks above. In general, text search engines:

- Do not export information about the sources (the source-metadata problem)
- Use different query languages (the query-language problem)
- Rank documents in the query results using secret algorithms (the rank-merging problem)

To improve this situation, the Digital Library project at Stanford coordinated search engine vendors and other key players to informally design a protocol that would allow basic interactions of sources in the three areas above. Below is the fourth (and final) draft of our informal "standards" effort. This draft is based on feedback from people from Excite, Fulcrum, GILS, Harvest, Hewlett-Packard Laboratories, Infoseek, Microsoft Network, Netscape, PLS, Verity, and WAIS, among others. This draft also incorporates feedback received from the participants of a workshop on STARTS that we organized at Stanford on August 1st, 1996.

We first define the architecture that we have in mind throughout the proposal (Section 1). Then, in Section 2 we specify a query language that is based on a simple subset of the Z39.50-1995 type-101 query language and the GILS attribute set. Section 3 describes what information should accompany the query results so we can merge the results from different sources in a meaningful way. Section 4 defines the metadata that each source should export to describe its contents (e.g., for resource discovery) and capabilities (e.g., for querying). We defer the discussion of syntax and communication issues to Section 5. In all cases, we propose a basic, simple solution that every source should support, and describe how more sophisticated sources can extend these solutions (Section 6). Finally, Section 7 lists issues that we decided to ignore in our proposal, mainly to keep our design simple.

1. Architecture, assumptions, and related efforts

In this section we describe the basic architecture underlying our proposal, and the assumptions that we make. We conclude by mentioning some of the most relevant related efforts.

In our architecture, there is a (potentially large) number of *resources*. Each resource consists of one or more *sources,* and simply exports contact information for its sources. A source is a collection of flat *documents* (e.g., we do not consider any nesting of documents) with an associated *search engine* that accepts queries from clients and produces results. Sources may be "small" (e.g., the collection of papers written by some university professor) or "large" (e.g., the collection of WWW pages indexed by a crawler).

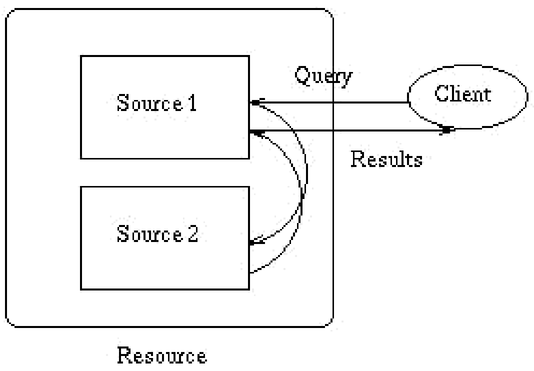
Our protocol is meant for machine-to-machine communication: users should not have to write queries using the proposed query language, for instance. Also, all communication with the sources is sessionless in our protocol, and the sources are stateless.

A metasearcher or any end client, in general, would typically issue queries to multiple sources. For this, a client will perform the following tasks:

- Extract the source list from the resources periodically
  (to find out what sources are available for querying)
- Extract metadata and content summaries from the sources periodically
  (to be able to decide, given a query, what sources are potentially useful for the query)

Given a user query:

- Issue the query to a source at a resource (Source 1 in the figure below), maybe specifying other sources at the resource where to also evaluate the query (Source 2 below)
- Issue the query to other promising resources
- Get the results from the multiple resources, merge them, and present them to the user



Our protocol describes how to query sources and what information the sources export about themselves. It does not describe an architecture for "metasearching," for example. However, we do describe the facilities that a metasearcher would need from the sources in order to perform searches. Metasearchers often have to search across simple sources as well as across sophisticated sources. On the one hand, it is important to have some agreed-upon minimal functionality that is simple enough for all sources to comply. On the other hand, it is important to allow the more sophisticated sources to export their richer features. Therefore, our protocol keeps the requirements to a minimum, while it provides optional features that fancy sources can use if they wish.

The most relevant standards effort in terms of shared goals is the Z39.50 standard. Z39.50 provides most of the functionality that we describe below. Our proposal is much simpler, and keeping it simple was one of our main concerns. There are currently efforts to define a simple profile of the Z39.50 standard based on STARTS. Other related efforts are Harvest and RDM. Both efforts provide a framework for querying and indexing multiple sources of documents. We view our effort as complementary in that it defines the pieces of information that sources should export to gatherers, and the query language and query-result format that the brokers should support, for example (using Harvest's terminology).

2. Query language/interface

In this section we describe the basic features of the query language that a source should support. This query language is based on a simple subset of the type-101 queries of the Z39.50-1995 standard. Similarly, the basic attribute set is mostly a subset of the GILS attribute set, which in turn includes the Bib-1 use attribute set. (We discuss syntax and communication issues in Section 5.1.)

A query consists of two parts:

- a *filter expression*, and
- a *ranking expression*.

A filter expression is Boolean in nature and defines the documents that qualify for the answer. The ranking expression associates a score with these documents and ranks them accordingly.

Example:

Consider the following query with filter expression:

((author "Garcia Molina") and (title "databases"))

and ranking expression:

list((body-of-text "distributed") (body-of-text "databases"))

This query returns documents having "`Garcia Molina`" as one of the authors and the word "`databases`" in their title. The documents that match the filter expression are then ranked according to how well their text matches the words "`distributed`" and "`databases`."

In principle, a query need not contain a filter expression. If this is the case, we assume that all documents qualify for the answer, and are ranked according to the ranking expression. Similarly, a query need not contain a ranking expression. If this is the case, the result of the query is the set of objects that match the (Boolean) filter expression. Sources might support just one type of expressions. In this case, they indicate what type they support as part of their metadata (e.g., a source may indicate that it only supports filter expressions).

Both the filter and the ranking expressions may contain multiple *terms*. The filter and ranking expressions combine these terms with operators like "`and`" and "`or`" (e.g., `((author "Garcia Molina") and (title "databases"))`). The ranking expressions also combine terms using the "`list`" operator, which simply groups together a set of terms (e.g., `list((body-of-text "distributed") (body-of-text "databases"))`). Also, the terms of a ranking expression may have a *weight* associated with them, indicating their relative importance in the ranking expression. Next we define the filter and ranking expressions more precisely.

L-strings

An *l-string* is either a string (e.g., "`Garcia Molina`"), or a string qualified with its associated language and, optionally, with its associated country. The language-country qualification follows the format described in RFC 1766. For example `[en-US "behavior"]` is an l-string meaning that the string "`behavior`" represents a word in American English. To support multiple character sets, the actual string in an l-string is a Unicode sequence encoded using UTF-8. A nice property of this encoding is that the code for a plain English string is the ASCII string itself, unmodified.

Atomic terms

A *term* (e.g., `(author "Garcia Molina")`) is an *l-string* modified by an unordered list of *attributes.* An attribute is either a *field* or a *modifier*. For example, the term `(date-last-modified > "1996-08-01")` has field `date-last-modified` and modifier `>`. (This term matches documents that were modified after August 1, 1996.)

We now define the "Basic-1" set of attributes. The attributes not marked with **(New)** are from the GILS attribute set (which in turn inherits all of the Z39.50-1995 Bib-1 use attributes). In the Appendix A we give the numbers associated with these attributes in this attribute set, together with pointers to definitions of their semantics. (Section 6 explains how to use other attribute sets for databases covering different domains, for example.)

- **Fields:** What portion of the text is associated with the term (e.g., the `author` portion, the `title` portion, etc.). At most one should be specified for each term. If no field is specified, "`Any`" is assumed. Those fields marked with **(Req)** must be supported. ("Supported" means that the source must recognize these fields. However, the source may freely interpret them.) The rest of the fields are optional. (Our fields correspond to the Z39-50 "use attributes.")

- `Title` **(Req)**
- `Author`
- `Body-of-text`
- `Document-text` **(New)**
  (For relevance feedback)
- `Date/time-last-modified` **(Req)**
  (Formatted according to the International Standard ISO 8601 (e.g., "`1996-12-31`"))
- `Any` **(Req)**
- `Linkage` **(Req)**
  (URL of the document)
- `Linkage-type`
  (MIME type of the document)
- `Cross-reference-linkage`
  (List of URLs in document)
- `Language`
  (The language(s) of the document, as a list of language tags as defined in RFC 1766.)
  (For example, a query term `(language "en-US")` matches a document with value for the language field `"en-US es"`. This document has parts in American English and in Spanish.)
- `Free-form-text` **(New)**
  (A string, maybe representing a query in some query language not in the protocol, that the source somehow knows how to interpret)

* **Modifiers:** What values the term represents (e.g., treat the term as a `stem`, as its `phonetics` (`soundex`), etc.). Zero or more modifiers can be specified for each term. (Our modifiers correspond to the Z39.50 "relation attributes.")
  - `<, <=, =, >=, >, !=`
    (If applicable (e.g., for fields like "`Date/time-last-modified`"), default: `=`)
  - `Phonetic` (soundex)
    (Default: no soundex)
  - `Stem`
    (Default: no stemming)
  - `Thesaurus` **(New)**
    (Default: no thesaurus expansion)
  - `Right-truncation`
    (Default: the term "as is," without right-truncating it)
  - `Left-truncation`
    (Default: the term "as is," without left-truncating it)
  - `Case-sensitive` **(New)**
    (Default: case insensitive)

Complex filter expressions

We use *operators* to build complex filter expressions from the terms. The "Basic-1"-type filter expressions use the following operators (which are borrowed from the Z39.50-1995 type-101 queries). If a source supports filter expressions, it must support all these operators.

- `and`
- `or`
- `and-not`
- `prox`, specifying two terms, the required distance between them, and whether the order of the terms matters.

Example:

Consider two terms `t1` and `t2` and the following filter expression:

```
(t1 prox[3,T] t2)
```

The documents that match this filter expression contain `t1` followed by `t2` with at most three words in between them. `"T"` (for "true") indicates that the word order matters (i.e., that `t1` has to be appear before `t2`).

Complex ranking expressions

We also use operators to build complex ranking expressions from the terms. The "Basic-1"-type ranking expressions use the operators above ("`and`," "`or`," "`and-not`," and "`prox`") plus a new operator, `list`, which simply groups together a set of terms. (The "Boolean" operators would most likely be interpreted as "fuzzy-logic"operators by the search engines in order to rank the documents.) If a source supports ranking expressions, it must support all these operators.

Each term in a ranking expression may have a weight (a number between 0 and 1) associated with it, indicating the relative importance of the term in the query.

Example:

The following ranking expression indicates that the term "`distributed`" is more important than the term "`databases`":

```
list(("distributed" 0.7) ("databases" 0.3))
```

The default field for both terms is `Any`, as we explained above.

Global settings

When we query a source, we also specify the following together with the query.

- Drop stop words: whether the source should delete the stop words from the query or not. (A metasearcher knows if it can turn off the use of stop words at a source from the source's metadata (Section 4).) (Default: Drop the stop words.)
- Default attribute set used in the query (optional, for notational convenience).
(Default: Basic-1.)
- Default language used in the query (optional, for notational convenience, and overridden by the specifications at the l-string level).
(No default.)

- Sources (in the same resource) where to evaluate the query in addition to the source where the query is submitted. (See Section 1.)
  (Default: no other source.)
- Answer specification:
  - Fields returned in the query answer
    (Default: `Title`, `Linkage`)
  - Fields used to sort the query results, and whether the order is ascending (`"a"`) or descending (`"d"`)
    (Default: `Score` of the documents for the query, in descending order.)
  - Documents returned:
    - Minimum acceptable document score
      (No default.)
    - Maximum acceptable number of documents
      (Default: 20 documents.)

As we mentioned earlier, we describe the formal syntax and format for the queries in Section 5.1.

3. Merging of ranks

After receiving a query, the source reports the number of documents in the result. Also, since the source might modify the given query before processing it, the source reports the query that it actually processed.

Example:

Consider a source that does not support the ranking-expression part of the queries. Consider the query with filter expression:

```
((author "Garcia Molina") and (title "databases"))
```

and ranking expression:

```
list((body-of-text "distributed") (body-of-text "databases"))
```

If the source simply ignores the ranking expressions, the actual query that the source processes has filter expression:

```
((author "Garcia Molina") and (title "databases"))
```

and an empty ranking expression. This actual query is returned with the query results.

- To merge the query results from multiple sources into a single, meaningful rank, a source should return the following information **for each document in the query result**:

- The unnormalized score of the document for the query
- The id of the source(s) where the document appears
- Statistics about each query term in the ranking expression (as modified by the query fields, if possible):
  - `Term-frequency:` the number of times that the query term appears in the document

- `Term-weight`: the weight of the query term in the document, as assigned by the search engine associated with the source (e.g., the normalized `tf.idf` weight for the query term in the document, or whatever other weighing of terms in documents the search engine might use)
- `Document-frequency`: the number of documents in the source that contain the term (this information is also provided as part of the metadata for the source)

Also:

- `Document-size`: the size of the document (in bytes)
- `Document-count`: the number of tokens (as determined by the source) in the document

In addition, every source provides the query results for a given **sample document collection** and a given set of queries (both provided and designed by Stanford) as part of the metadata for the source. (See below.) A metasearcher may use these sample results to calibrate document scores from different sources.

A document may appear in multiple sources at a resource. Therefore, if a query submitted to a source `S` requests evaluation at multiple sources, a document in the query result might appear in multiple such sources. If this is the case, source `S` can either report such a document multiple times (but using the same URL to refer to the document in the multiple sources), or it can report it once with merged score, weights and frequencies.

4. Source metadata

To select the *right* sources for a query and to query them we need information about their contents and capabilities. In this section we propose two pieces of metadata that every **source** is required to export: a list of metadata attributes, describing properties of the source, and a content summary of the source. Each piece is a separate object, to allow metasearchers to retrieve just the metadata that they need. Also, we describe the information that a **resource** exports. This information identifies the metadata objects for the sources in the resource. In Sections 5.3, 5.4, and 5.5 we describe the syntax of these objects and how to retrieve them.

4.1. Source metadata attributes

Each source exports information about itself by giving values to the metadata attributes below. For example, the value for the metadata attribute "`FieldsSupported`" is a list of the fields that can be used for querying the source (e.g., "`Linkage-type`").

Below we define the "MBasic-1" set of metadata attributes, borrowing from the Z39.50-1995 Exp-1 and the GILS attribute set. The attributes not marked with **(New)** are from these two attribute sets. The attributes marked **(Req)** are required, and the sources must support them.

- `FieldsSupported` **(New) (Req)**
  (What optional fields are supported in addition to the required ones. Also, each field is optionally accompanied by a list of the languages that are used in that field in the source. Required fields can also be listed here with their corresponding language list.)

- `ModifiersSupported` **(New) (Req)**
  (What optional modifiers are supported in addition to the required ones. Also, each modifier is optionally accompanied by a list of the languages for which it is supported at the source. (Modifiers like `stem` are language dependent.))
- `QueryPartsSupported` **(New)**
  (Whether the source supports ranking expressions only ("`R`"), filter expressions only ("`F`"), or both ("`RF`"). Default: "`RF`.")
- `ScoreRange` **(New) (Req)**
  (This is the minimum and maximum score that a document can get for a query; we use this information for merging ranks. Valid bounds include `-infinity` and `+infinity`, respectively.)
- `RankingAlgorithmID` (e.g., `Acme-1`) **(New) (Req)**
  (Even when we do not know the actual algorithm used it is useful to know that two sources use the same algorithm.)
- `TokenizerIDList` **(New)**
  (E.g., `(Acme-1 en-US) (Acme-2 es)`, meaning that tokenizer `Acme-1` is used for strings in American English, and tokenizer `Acme-2` is used for strings in Spanish. Even when we do not know how the actual tokenizer works, it is useful to know that two sources use the same tokenizer.)
- `SampleDatabaseResults` **(New) (Req)**
  (The URL to get the query results for a sample document collection; see Section 3.)
- `StopWordList` **(New) (Req)**
- `TurnOffStopWords` **(New) (Req)**
  (Whether we can turn off the use of stop words at the source or not.)
- `SourceLanguage`
  (List of languages present at the source.)
- `SourceName`
- `Linkage` **(Req)**
  (URL where the source should be queried.)
- `ContentSummaryLinkage` **(New) (Req)**
  (The URL of the source content summary; see below.)
- `DateChanged`
  (The date when the source metadata was last modified.)
- `DateExpires`
  (The date when the source metadata will be reviewed, and therefore, when the source metadata should be extracted again.)
- `Abstract` (of the source)
- `AccessConstraints`
  (A description of the constraints or legal prerequisites for accessing the source.)
- `Contact`
  (Contact information of the administrator of the source.)

4.2. Source content summary

Each source exports data about its contents. This data can be used to decide if the source is relevant for a given query, for example, and includes:

- List of words that appear in the source, specifying whether:
  - The words listed are stemmed or not.

- The words listed include stop words or not.
- The words listed are case sensitive or not.
- The words listed are accompanied by the field corresponding to where in the documents they occurred (e.g., `(title "databases")`, `(abstract "databases")`, etc.).

  If possible, the words listed should not be stemmed, and should include the stop words. Also, the words should be case sensitive, and be accompanied by their corresponding field information, as shown above.

  In addition, the words might be qualified with their corresponding language (e.g., `[en-US "behavior"]`).

- Statistics for each word listed, including at least one of the following:
  - Total number of postings for each word (i.e., the number of times that the word appears in the source)
  - Document frequency for each word (i.e., the number of documents that contain the word)
- Total number of documents in source

4.3. Resource definition

Each resource exports contact information about the sources that it contains. More specifically, a resource simply exports its list of sources, together with the URLs where the metadata attributes for the sources can be accessed and the format of this data. (Currently there is only one format for this metadata, as Sections 5.3 and 5.4 describe.) Using this information, a metasearcher learns how and where to contact each of the sources in the resource.

5. Syntax and communication

We represent queries, query results, and metadata using attribute-value pairs in Harvest's Summary Object Interchange Format (SOIF). (See Appendix B for the formal definition of SOIF that we use.) The SOIF objects corresponding to the queries are sent using HTML and the POST method, using a single tag "`SOIF`," whose value is the SOIF object representing the query itself. The query results are also obtained through HTTP. The metadata SOIF objects can be retrieved using HTTP, FTP, or other protocols, as specified in their associated URLs.

To learn about the sources available at a resource, we just need to know the URL of the SOIF object for the resource description. As we will see below, this object points to the metadata-attribute SOIF objects for the sources at the resource. In turn, these objects have information on how and where to query the sources themselves, as well as pointers to the content summaries of the sources.

5.1. Query language

A query is a SOIF object of template type "`SQuery`." (The template type of a SOIF object determines the SOIF attributes that the object has.)

Example:

Below is a SOIF object for a query. The number in brackets after each SOIF attribute (e.g., "`50`" after the `FilterExpression` SOIF attribute) is the number of bytes of the value for that attribute, to facilitate parsing.

```
@SQuery{
Version{10}: STARTS 1.0
FilterExpression{50}: ((author "Garcia Molina") and (title "databases"))
RankingExpression{61}: list((body-of-text "distributed") (body-of-text "databases"))
DropStopWords{1}: T
DefaultAttributeSet{7}: basic-1
DefaultLanguage{5}: en-US
AnswerFields{12}: title author
MinDocumentScore{3}: 0.5
MaxNumberDocuments{2}: 10
}
```

We now define each attribute for the `SQuery` template, and give a grammar for the values that may accompany the attribute.

Attribute	Value
Version	Alpha-Numeric-String
FilterExpression	FILTER
RankingExpression	RANKING
DropStopWords	Boolean
DefaultAttributeSet	ATTRIBUTE-SET
DefaultLanguage	LANGUAGE
Sources	SOURCE-ID-LIST
AnswerFields	FIELD-LIST
SortByFields	SORT-FIELD-LIST
MinDocumentScore	Number
MaxNumberDocuments	Number

We now give a grammar for the attribute values:

```
FILTER          -> TERM | ( TERM PROX-OP TERM ) |
                   ( FILTER BOOLEAN-OP FILTER )

RANKING         -> TERM | ( TERM PROX-OP TERM ) |
                   list( RANKING-LIST ) | list( W-RANKING-LIST )|
                   ( RANKING BOOLEAN-OP RANKING ) |
                   ( W-RANKING BOOLEAN-OP W-RANKING )

TERM            -> L-STRING |
                   ( FIELD MODIFIER-LIST L-STRING )

L-STRING        -> " String " | [ LANGUAGE " String " ]

RANKING-LIST    -> RANKING | RANKING RANKING-LIST

W-RANKING-LIST  -> W-RANKING | W-RANKING W-RANKING-LIST

W-RANKING       -> ( RANKING Number )

BOOLEAN-OP      -> and | or | and-not

PROX-OP         -> prox[ Number , Boolean ]

ATTRIBUTE-SET   -> basic-1 | ...
```

```
FIELD           -> BASIC1-FIELD | { basic-1 BASIC1-FIELD } | ...

BASIC1-FIELD    -> title | author | body-of-text | document-text |
                   date-last-modified | any | linkage | linkage-type |
                   cross-reference-linkage | language | free-form-text

FIELD-LIST      -> FIELD| FIELD FIELD-LIST

MODIFIER        -> BASIC1-MODIFIER | { basic-1 BASIC1-MODIFIER } | ...

BASIC1-MODIFIER -> RELATION | phonetic | stem | thesaurus |
                   right-truncation | case-sensitive

RELATION        -> < | <= | = | >= | > | !=

MODIFIER-LIST   -> | MODIFIER MODIFIER-LIST

LANGUAGE        -> LanguageCode | LanguageCode-CountryCode | ...
                   (See RFC 1766.)

SOURCE-ID-LIST  -> SOURCE-ID | SOURCE-ID SOURCE-ID-LIST

SOURCE-ID       -> Alpha-Numeric-String

SORT-FIELD-LIST -> SORT-FIELD | SORT-FIELD SORT-FIELD-LIST

SORT-FIELD      -> ( FIELD a ) | ( FIELD d ) | (Score d)
```

5.2. Query results

The results for a query start with a SOIF object of type "`SQResults`," followed by a series of SOIF objects of template type "`SQRDocument`." Each of the latter objects corresponds to a document in the query result.

The `SQResults` object has the following SOIF attributes and associated values:

SOIF Attribute	Value
Version	Alpha-Numeric-String
ActualFilterExpression	FILTER
ActualRankingExpression	RANKING
NumDocSOIFs	Number

(Number of `SQRDocument` SOIF objects that follow.)

The `SQRDocument` objects have the following SOIF attributes and associated values:

SOIF Attribute	Value
Version	Alpha-Numeric-String
RawScore (*)	Number
Sources	SOURCE-ID-LIST
FIELD (**)	Alpha-Numeric-String
TermStats	TERM-STATS-LIST
DocSize	Number (of Kilobytes)
DocCount	Number (of tokens in document)

where:

```
TERM-STATS-LIST -> TERM Number Number Number |
                   TERM Number Number Number TERM-STATS-LIST
```

(*) `RawScore` is the score for the query that was assigned by the source to the document. This score is not normalized so the best document always has a score of 1, for example.

(**) An object has multiple such field-value pairs, in general.

Example:

A portion of the result for the example query above may look like the following.

```
@SQResults{
Version{10}: STARTS 1.0
ActualFilterExpression{50}: ((author "Garcia Molina") and
(title "databases"))
ActualRankingExpression{26}: (body-of-text "databases")
/* maybe "distributed" was a stop word */
NumDocSOIFs{1}: 4
}

@SQRDocument{
Version{10}: STARTS 1.0
RawScore{4}: 0.82
Sources{8}: Source-2
linkage{51}: http://www-db.stanford.edu/pub/gravano/1995/vldb.ps
title{44}: Generalizing GlOSS to Vector-Space Databases
author{34}: Luis Gravano, Hector Garcia-Molina
TermStats{89}: (body-of-text "distributed") 10 0.31 190
               (body-of-text "databases") 15 0.51 232
DocSize{3}: 248
DocCount{5}: 10213
}
...
@SQRDocument{
...
}
```

5.3. Source metadata attributes

The metadata attributes for a source are in a SOIF object of template type "`SMetaAttributes`," with the following SOIF attributes and associated values:

SOIF Attribute	Value
Version	Alpha-Numeric-String
SourceID	SOURCE-ID
FieldsSupported	L-FIELD-LIST
ModifiersSupported	L-MODIFIER-LIST
QueryPartsSupported	Alpha-String
ScoreRange	Number Number
RankingAlgorithmID	Alpha-Numeric-String
TokenizerIDList	TOKENIZER-LIST
SampleDatabaseResults	Alpha-Numeric-String
StopWordList	Alpha-Numeric-String-List
TurnOffStopWords	Boolean

```
DefaultMetaAttributeSet          META-ATTRIBUTE-SET
META-ATTRIBUTES(*)               Alpha-Numeric-String
```

where:

```
L-FIELD-LIST             -> L-FIELD | L-FIELD L-FIELD-LIST

L-FIELD                  -> FIELD | (FIELD LANGUAGE-LIST)

L-MODIFIER-LIST          -> L-MODIFIER | L-MODIFIER L-MODIFIER-LIST

L-MODIFIER               -> MODIFIER | (MODIFIER LANGUAGE-LIST)

LANGUAGE-LIST            -> LANGUAGE | LANGUAGE LANGUAGE-LIST

TOKENIZER-LIST           -> TOKENIZER | TOKENIZER TOKENIZER-LIST

TOKENIZER                -> (Alpha-Numeric-String LANGUAGE-LIST)

META-ATTRIBUTE-SET       -> mbasic-1 | ...

META-ATTRIBUTES          -> MBASIC1-META | (mbasic-1 MBASIC1-META) | ...

MBASIC1-META             -> source-language | source-name|
                            linkage | content-summary-linkage |
                            date-changed | date-expires | abstract |
                            access-constraints | contact
```

(*) An object has multiple such attribute-value pairs, in general.

Example:

```
@SMetaAttributes{
Version{10}: STARTS 1.0
SourceID{8}: Source-1
FieldsSupported{17}: (basic-1 author)
ModifiersSupported{19}: (basic-1 phonetics)
QueryPartsSupported{2}: RF
ScoreRange{7}:  0.0 1.0
RankingAlgorithmID{8}: Acme-1
...
DefaultMetaAttributeSet{6}: mbasic-1
source-language{8}: en-US es
source-name{18}: Stanford DB Group
linkage{26}: http://www-db.stanford.edu/cgi-bin/query
content-summary-linkage{38}: ftp://www-db.stanford.edu/cont_sum.txt
date-changed{9}: 1996-03-31
}
```

5.4. Source content summary

The content summary for a source is in a SOIF object of template type "`SContentSummary`," with the following SOIF attributes and associated values:

SOIF Attribute	Value
Version	Alpha-Numeric-String
Stemming	Boolean

```
StopWords                       Boolean
CaseSensitive                   Boolean
Fields                          Boolean
NumDocs                         Number

Field                           FIELD
Language                        LANGUAGE
TermFreq   (*)                  SIMPLE-TERM-STATS-LIST
DocFreq    (*)                  SIMPLE-TERM-STATS-LIST
TermDocFreq(*)                  MEDIUM-TERM-STATS-LIST
```

where:

```
SIMPLE-TERM-STATS-LIST -> TERM Number |
                          TERM Number SIMPLE-TERM-STATS-LIST

MEDIUM-TERM-STATS-LIST -> TERM Number Number |
                          TERM Number Number MEDIUM-TERM-STATS-LIST
```

(*) Only one of the three attribute-value pairs should appear in the record, depending on what statistics are available. For example, if the source only returns the term frequency for each term, then the SOIF object contains attribute `TermFreq`, but not `DocFreq` or `TermDocFreq`.

The last five attributes are actually a repeating group: `Field` and `Language` apply to all the frequencies that follow, until the next `Field` and/or `Language` attribute.

Example:

```
@SContentSummary{
Version{10}: STARTS 1.0
Stemming{1}: F
StopWords{1}: F
CaseSensitive{1}: F
Fields{1}: T
NumDocs{3}: 892

Field{5}: title
Language{5}: en-US
TermDocFreq{11023}: "algorithm" 100 53
                    "analysis" 50 23
...

Field{5}: title
Language{2}: es
TermDocFreq{1211}: "algoritmo" 23 11
                   "datos" 59 12
...

}
```

5.5. Resource metadata

The metadata attributes for a resource are in a SOIF object of template of type "`SResource`," with the following SOIF attributes and associated values:

```
SOIF        Attribute Value
_________________________________
Version     Alpha-Numeric-String
SourceList  SOURCE-DESCRIPTION-LIST
```

where:

```
SOURCE-DESCRIPTION-LIST -> SOURCE-DESCRIPTION |
                           SOURCE-DESCRIPTION SOURCE-DESCRIPTION-LIST

SOURCE-DESCRIPTION -> SOURCE-ID URL METADATA-SYNTAX

METADATA-SYNTAX    -> Stanford-1 | ...
```

where the second component is the URL of the SOIF with the metadata attributes for the source, and the third component defines how metadata, etc. is packaged and transmitted, to allow for alternatives to be defined in the future.

Example:

```
@SResource{
Version{10}: STARTS 1.0
SourceList{83}: Source_1 ftp://www.stanford.edu/source_1 Stanford-1
                Source_2 ftp://www.stanford.edu/source_2 Stanford-1
}
```

6. Extending the basic features

6.1. Query language

Each source can extend the basic syntax for queries that we described above by adding new fields and modifiers. To avoid naming these new attributes chaotically, a source has two options:

- Use attribute sets that are registered within the Z39.50-1995 standard. For example, a source can define a new field "`editor`" by using the Bib-1 attribute "`editor`," and referring to it as `{1.2.840.10003.3.1 editor}`, where "`1.2.840.10003.3.1`" is the object identifier for the Bib-1 attribute set. (See http://lcweb.loc.gov/z3950/agency/objects/attrbute.html.)
- Use attribute sets that are named consistently by each search-engine vendor. For example, Acme can support a new attribute set called "`acme-1`" that includes a modifier "`topic`." Then, a query can refer to this attribute as `{acme-1 topic}`. (Acme would only need to give a name to such an extension set so that its search engine recognizes that `{acme-1 topic}` maps to the right Acme operator. It should also provide a document describing, formally or not, how to use the extensions, their allowed operand types, etc.)

The queries can either explicitly refer to the attribute set of each attribute mentioned, or they can define once and for all the default attribute set used in the query.

A query that uses non-basic attribute sets cannot expect to be evaluated everywhere without being modified.

6.2. Metadata

Each source can extend the basic set of metadata attributes. To avoid naming these new attributes chaotically, a source should use attribute sets that are registered with the Z39.50-1995 standard, like Exp-1 and GILS. For example, a source can support a new metadata attribute "`contact-telephone`" by using the corresponding GILS attribute, and referring to it as `{1.2.840.10003.3.5 contact-telephone}`, where "`1.2.840.10003.3.5`" is the object identifier for the GILS attribute set. (See http://lcweb.loc.gov/z3950/agency/objects/attrbute.html.)

7. Issues that we ignored

In this section we list issues that we deliberately have not addressed, mostly to keep our design simple. Some of these issues are important ones, and we might consider them in future versions of the protocol.

- Allowing for non-textual sources
- Allowing for more structured (e.g., nested) documents
- Reporting errors
- Addressing security issues

- Including other query languages into the protocol (e.g., SQL)
- Specifying what thesaurus should be used to answer a query
- Allowing arbitrary wildcards in the queries
- Allowing more complex, non-unary modifiers (e.g., range queries for geographical sources)

- Allowing for multiple result formats
- Highlighting query terms in the documents in the query results, and other presentation issues

- Specifying the stemming algorithms that the sources use
- Specifying the tokenization algorithms that the sources use in more detail
- Specifying the legal field-modifier combinations at each source
- Specifying the fields that are searchable vs. those that are retrieve-only
- Specifying the data types associated with the fields more formally
- Specifying the available thesauri

- Retrieving just pieces of the source content summaries, instead of the entire SOIFs
- Implementing a "push" model for the source metadata, instead of just a "pull" model as with the current design
- Implementing sessions

Appendix A.

Mapping some of the Basic-1 attributes to the Z39.50-1995 Bib-1 and GILS attribute set

We now list each Bib-1 or GILS attribute that we included in the Basic-1 attribute set. The pair of numbers (in parenthesis) by these attributes are their associated type and value in these attribute sets. (For the semantics of these attributes please check ftp://ftp.loc.gov/pub/z3950/defs/bib1.txt and http://www.usgs.gov/gils/prof_v2.html.)

- **Fields**: Our fields correspond to the Z39-50 "use attributes." All of these attributes are from the GILS attribute set, which in turn inherits all of the Bib-1 use attributes.
  - `Title` (1 4)
  - `Author` (1 1003)
  - `Body-of-text` (1 1010)
  - `Date/time-last-modified` (1 1012)
  - `Any` (1 1016)
  - `Linkage` (1 2021)
    (URL of the document)
  - `Linkage-type` (1 2022)
    (MIME type of the document)
  - `Cross-reference-linkage` (1 2047)
    (URLs in document)
  - `Language` (1 54)
    (The language(s) of the document)

- **Modifiers**: Our modifiers correspond to the Z39.50 "relation attributes."
  - `<, <=, =, >=, >, !=` Bib-1: (2 1 through 6)
  - `Phonetic` (soundex) Bib-1: (2 100)
  - `Stem` Bib-1: (2 101)
  - `Right-truncation` Bib-1: (5 1)
  - `Left-truncation` Bib-1: (5 2)

Mapping some of the MBasic-1 attributes to the Z39.50-1995 Exp-1 and GILS attribute set

We now list each Exp-1 or GILS attribute that we included in the MBasic-1 attribute set. In parenthesis by these attributes is their associated number in the Exp-1 or GILS attribute sets.

- `SourceLanguage` Exp-1: (2)
- `SourceName` Exp-1: (3)
- `Linkage` GILS: (2021)
- `DateChanged` Exp-1: (10)
- `DateExpires` Exp-1:(11)
- `Abstract` (of the source) GILS: (62)
- `AccessConstraints` GILS:(2004)
- `Contact` Exp-1:(18)

Appendix B. Formal description of SOIF

The SOIF Grammar (as described in the Harvest manual) is as follows:

```
SOIF           -> OBJECT SOIF |
                  OBJECT
OBJECT         -> @ TEMPLATE-TYPE { URL ATTRIBUTE-LIST }
ATTRIBUTE-LIST -> ATTRIBUTE ATTRIBUTE-LIST |
                  ATTRIBUTE
ATTRIBUTE      -> IDENTIFIER {VALUE-SIZE} DELIMITER VALUE
```

```
TEMPLATE-TYPE  -> Alpha-Numeric-String
IDENTIFIER     -> Alpha-Numeric-String
VALUE          -> Arbitrary-Data
VALUE-SIZE     -> Number
DELIMITER      -> ":<tab>"
```

You can also find a description of SOIFs in the W3C draft for Netscape’s Resource Description Messages (RDM).

The STARTS home page
Luis Gravano
gravano@cs.stanford.edu