STARTS: Stanford Proposal for Internet Meta-Searching *

Luis Gravano
Computer Science Department
Stanford University
gravano@cs.stanford.edu

Chen-Chuan K. Chang
Computer Science Department
Stanford University
kevin@db.stanford.edu

Héctor García-Molina
Computer Science Department
Stanford University
hector@cs.stanford.edu

Andreas Paepcke
Computer Science Department
Stanford University
paepcke@db.stanford.edu

Abstract

Document sources are available everywhere, both within the internal networks of organizations and on the Internet. Even individual organizations use search engines from different vendors to index their internal document collections. These search engines are typically incompatible in that they support different query models and interfaces, they do not return enough information with the query results for adequate merging of the results, and finally, in that they do not export metadata about the collections that they index (e.g., to assist in resource discovery). This paper describes *STARTS*, an emerging protocol for Internet retrieval and search that facilitates the task of querying multiple document sources. *STARTS* has been developed in a unique way. It is not a standard, but a group effort coordinated by Stanford's Digital Library project, and involving over 11 companies and organizations. The objective of this paper is not only to give an overview of the *STARTS* protocol proposal, but also to discuss the process that led to its definition.

1 Introduction

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 and models that vary from source to source. Even individual organizations use search engines from different vendors to index their internal document collections. These organizations can benefit from *metasearchers*, which are services that provide unified query interfaces to multiple search engines. Thus, users have the illusion of a single combined document source. This paper describes *STARTS* [1], an emerging protocol for Internet retrieval and search. The goal of *STARTS* is to facilitate the main three tasks that a metasearcher performs:

- Choosing the best sources to evaluate a query
- Evaluating the query at these sources
- Merging the query results from these sources

STARTS has been developed in a unique way. It is not a standard, but a group effort involving over 11 companies and organizations. The objective of this paper is not only to give an overview of the *STARTS* protocol proposal, but also to discuss the process that led to its definition. In particular,

- We will describe the history of the project, including the current status of a reference implementation, and will highlight some of the existing "tensions" between information providers and search engine builders (Sections 2 and 3).
- We will explain the protocol, together with some of the tradeoffs and compromises that we had to make in its design (Section 4).
- We will comment on other work that is closely related to *STARTS* (Section 5).

2 History of our Proposal

The Digital Library project at Stanford coordinated search engine vendors and other key players to informally design a protocol that would allow searching and retrieval of information from distributed and heterogeneous sources. We were initially contacted by Steve Kirsch, president of Infoseek (`http://www.infoseek.com`), in June 1995. His idea was that Stanford should collect the views of the search engine vendors on how to address the problem at hand. Then Stanford, acting as an unbiased party, would design a protocol proposal that would reconcile the vendors' ideas. The key motivation behind this informal procedure was to avoid the long delays usually involved in the definition of formal standards.

In July, 1995, we started our effort with five companies: Fulcrum (`http://www.fulcrum.com`), Infoseek, PLS

* This material is based upon work supported by the National Science Foundation under Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA, NASA, and the industrial partners of the Stanford Digital Libraries Project. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the other sponsors.

[1] *STARTS* stands for "**Sta**nford Protocol Proposal for Internet **R**e**t**rieval and **S**earch."

(http://www.pls.com), Verity (http://www.verity.com), and WAIS. Microsoft Network (http://www.msn.com) joined the initial group in November. We circulated a preliminary draft describing the main three problems that we wanted to address (i.e., choosing the best sources for a query, evaluating the query at these sources, and merging the query results from the sources). We scheduled meetings with people from the companies to discuss these problems and get feedback. We met individually with each company between December, 1995, and February, 1996. During each meeting, we would show a couple of slides for each problem to agree on its definition, terminology, etc. After this, we would discuss the possible solutions for each problem in detail.

Based on the comments and suggestions that we received, we produced a first draft of our proposal by March, 1996. We then produced two revisions of this draft using feedback from the original companies, plus other organizations that started participating. Among these companies and organizations are Excite (http://www.excite.com), GILS (http://info.er.usgs.gov:80/gils/), Harvest (http://harvest.transarc.com), Hewlett-Packard Laboratories (http://www.hpl.hp.com), and Netscape (http://www.netscape.com). Finally, we held a workshop at Stanford with the major participants on August 1st, 1996 (http://www-db.stanford.edu/~gravano/workshop_participants.html). The goal of this one-day workshop was to iron out the controversial aspects of the proposal, and to get feedback for its final draft [1].

Defining *STARTS* has been a very interesting experience: we wanted to design a protocol that would be simple, yet powerful enough to allow us to address the three problems at hand. We could have adopted a "least common denominator" approach for our solution. However, many interesting interactions would have been impossible under such a solution. Alternatively, we could have incorporated the sophisticated features that the search engines provide, but that also would have challenged interoperability, and would have driven us away from simplicity. Consequently, we had to walk a very fine line, trying to find a solution that would be expressible enough, but not too complicated or impossible to quickly implement by the search engine vendors.

Another aspect that made the experience challenging was dealing with companies that have secret, proprietary algorithms, as those for ranking documents. (See Section 4.2.) Obviously, we could not ask the companies to reveal these algorithms. However, we still needed to have them export enough information so that a metasearcher could do something useful with the query results.

As mentioned above, the *STARTS*-1.0 specification is already completed. A reference implementation of the protocol has been built at Cornell University by Carl Lagoze. (See http://www-diglib.stanford.edu for information.) Also, the Z39.50 community is designing a profile of their Z39.50-1995 standard based on *STARTS*. (This profile was originally called *ZSTARTS*, but has since changed its name to *ZDSR*, for Z39.50 Profile for Simple Distributed Search and Ranked Retrieval.) Finally, we will try to find a sponsor to present *STARTS* under the World-Wide Web Consortium (W3C), so that a formal standard can emerge from it.

Our goal in presenting this paper at the SIGMOD conference is to also get the SIGMOD community involved in this effort. We believe that the Internet has become central to "management of data" (the *MOD* in *SIGMOD*), and searching and resource discovery across the Internet is one of the most important problems.

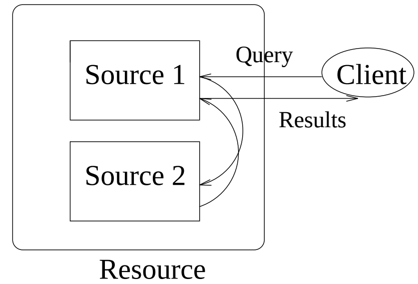

Figure 1: A metasearcher queries a source, and may specify that the query be evaluated at several sources at the same resource.

3 Our Metasearch Model and its Associated Problems

In this section we describe the basic metasearch model underlying our proposal, and the three main problems that a metasearcher faces today. These problems motivated our effort.

For the purpose of the *STARTS* protocol, we view the Internet as a potentially large number of *resources* (e.g., Knight-Ridder's Dialog information service, or the CS-TR sources [2]). Each resource consists of one or more *sources* (Figure 1). A source is a collection of text documents (e.g., Inspec and the Computer Database in the Dialog resource), with an associated search engine that accepts queries from clients and produces results. We assume that documents are "flat," in the sense that we do not, for example, allow any nesting of documents. We do not consider non-textual documents or data either (e.g., geographical data) to keep the protocol simple. Sources may be "small" (e.g., the collection of papers written by some university professor) or "large" (e.g., the collection of World-Wide Web pages indexed by a crawler).

As described in the Introduction, a metasearcher (or any end client, in general) would typically issue queries to multiple sources, for which it needs to perform three main tasks. First, the metasearcher chooses the best sources to evaluate a query. Then, it submits the query to these sources. Finally, it merges the results from the sources and presents them to the user that issued the query. To query multiple sources within the same resource, the metasearcher issues the query to one of the sources at the resource (Source 1 in Figure 1), specifying the other "local" sources where to also evaluate the query (Source 2 in Figure 1). This way, the resource can eliminate duplicate documents from the query result, for example, which would be difficult for the metasearcher to do if it queried all of the sources independently.

Building metasearchers is nowadays a hard task because different search engines are largely incompatible and do not allow for interoperability. In general, text search engines:

- Use different query languages (*the query-language problem*; Section 3.1)
- Rank documents in the query results using secret algorithms (*the rank-merging problem*; Section 3.2)
- Do not export information about the sources in a standard form (*the source-metadata problem*; Section 3.3)

[2]The CS-TR sources constitute an emerging library of Computer Science Technical Reports (http://www.ncstrl.org).

Below we visit each of these metasearch problems. The discussion will illustrate the need for an agreement between search engine vendors so that metasearchers can work effectively. Finally, Section 3.4 summarizes the metasearch requirements that should be facilitated by the agreement.

3.1 The Query-Language Problem

A metasearcher submits queries over multiple sources. But the interfaces and capabilities of these sources may vary dramatically. Even the basic query model that the sources support may vary.

Some search engines (e.g., Glimpse) only support the *Boolean retrieval model* [2]. In this model, a query is a condition that documents either do or do not satisfy. The query result is then a *set* of documents. For example, a query "`distributed and systems`" returns all documents that contain both the words "distributed" and "systems" in them.

Alternatively, most commercial search engines also support some variation of the *vector-space retrieval model* [2]. In this model, a query is a list of terms, and documents are assigned a score according to how *similar* they are to the query. The query result is then a *rank* of documents. For example, a query "`distributed systems`" returns a rank of documents that is typically based on the number of occurrences of the words "distributed" and "systems" in them. [3] A document in the query result might contain the word "distributed" but not the word "systems," for example, or vice versa, unlike in the Boolean-model case above.

Even if two sources support a Boolean retrieval model, their query syntax often differ. A query asking for documents with the words "distributed" and "systems" might be expressed as "`distributed and systems`" in one source, and as "`+distributed +systems`" in another, for example.

More serious problems appear if different fields (e.g., `abstract`) are available for searching at different sources. For example, a source might support queries like `(abstract ''databases'')` that ask for documents that have the word "databases" in their abstract, whereas some other sources might not support the `abstract` field for querying.

Another complication results from different stemming algorithms or stop-word lists being implicit in the query model of each source. (Stemming is used to make a query on "systems" also retrieve documents on "system," for example. Stop words are used to not process words like "the" in the queries, for example.) If a user wants documents about the rock group "The Who," knowing about the stop-word behavior of the sources would allow a metasearcher, for example, to know whether it is possible to disallow the elimination of stop words from queries at each source.

As a result of all this heterogeneity, a metasearcher would have to translate the original query to adjust it to each source's syntax. To do this translation, the metasearcher needs to know the characteristics of each source. (The work in [3, 4] illustrates the complexities involved in query translation.) As we will see in Section 4.1, querying multiple sources is much easier if the sources support some common query language. Even if support for most of this language is optional, query translation is much simpler if sources reveal what portions of the language they support.

[3] These ranks also typically depend on other factors, like the number of documents in the source that contain the query words, for example.

3.2 The Rank-Merging Problem

A source that supports the vector-space retrieval model ranks its documents according to how "similar" the documents and a given query are. Unfortunately, there are many ways to compute these similarities. To make matters more complicated, the ranking algorithms are usually proprietary to the search engine vendors, and their details are not publicly available.

Merging query results from sources that use different and unknown ranking algorithms is hard. (See [5, 6] for algorithms for merging multiple document ranks.) For example, source S_1 might report that document d_1 has a *score* of 0.3 for some query, while source S_2 might report that document d_2 has a score of 1,000 for the same query. If we want to merge the results from S_1 and S_2 into a single document rank, should we rank d_1 higher than d_2, or vice versa? (Some search engines are designed so that the top document for a query always has a score of, say, 1,000.)

It is even hard to merge query results from sources that use the same ranking algorithm, even if we know this algorithm. The reason is that the algorithm might rank documents differently based on the collection where the document appears. For example, if a source S_1 specializes in computer science, the word *databases* might appear in many of its documents. Then, this word will tend to have a low associated weight in S_1 (e.g., if S_1 uses the *tf·idf* formula for computing weights [2]). The word *databases*, on the other hand, might have a high associated weight in a source S_2 that is totally unrelated to computer science and contains very few documents with that word. Consequently, S_1 might assign its documents a low score for a query containing the word *databases*, while S_2 assigns a few documents a high score for that query. Therefore, it is possible for two very similar documents d_1 and d_2 to receive very different scores for a given query, if d_1 appears in S_1 and d_2 appears in S_2. Thus, even if the sources use the same ranking algorithm, a metasearcher still needs additional information to merge query results in a meaningful way.

3.3 The Source-Metadata Problem

A metasearcher might have thousands of sources available for querying. Some of these sources might charge for their use. Some of the sources might have large response times. Therefore, it becomes crucial that the metasearcher just contact sources that might contain useful documents for a given query. The metasearcher then needs information about each source's contents.

Some sources freely deliver their entire document collection, whereas others do not. Often, those sources that have for-pay information are of the second type. If a source exports all of its contents (e.g., many World-Wide Web sites), then it is not as critical to have it describe its collection to the metasearchers. After all, the metasearchers can just grab all of the sources' contents and summarize them any way they want. This is what "crawlers" like AltaVista (`http://www.altavista.digital.com`) do. However, for performance reasons, it may still be useful to require that such sources export a more succinct description of themselves. In contrast, if a source "hides" its information (e.g., through a search interface), then it is even more important that the source can describe its contents. Otherwise, if a source does not export any kind of content summary, it becomes hard for a metasearcher to assess what kind of information the source covers.

3.4 Metasearch Requirements

In summary, a sophisticated metasearcher will need to perform the following tasks in order to efficiently query multiple resources:

- Extract the list of sources from the resources periodically (to find out what sources are available for querying) (Section 4.3.3)
- Extract metadata and content summaries from the sources periodically (to be able to decide what sources are potentially useful for a given query) (Section 4.3.2)

Also, given a user query:

- Issue the query to one or more sources at one or more resources (Sections 4.1 and 4.3.1)
- Get the results from the multiple resources, merge them, and present them to the user (Section 4.2)

4 Our Protocol Proposal

In this section we define a protocol proposal that addresses the metasearch requirements of Section 3.4. This 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. Finally, we do not deal with any security issues, or with error reporting in our proposal. The main motivation behind these (and many of the other) decisions is to keep the protocol simple and easy to implement.

Our protocol does not describe an architecture for "metasearching." However, it does describe the facilities that a source needs to provide in order to help a metasearcher. The facilities provided by a source can range from simple to sophisticated, and one of the key challenges in developing our protocol was in deciding the right level of sophistication. In effect, metasearchers often have to search across simple sources as well as across sophisticated ones. On the one hand, it is important to have some agreed-upon minimal functionality that is simple enough for all sources to comply with. 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 sophisticated sources can use if they wish.

Our protocol mainly deals with *what information* needs to be exchanged between sources and metasearchers (e.g., a query, a result set), and not so much with *how* that information is formatted (e.g., using Harvest SOIFs [4]) or transported (e.g., using HTTP). Actually, what transport to use generated some heated debate during the *STARTS* workshop. Consequently, we expect the *STARTS* information to be delivered in multiple ways in practice. For concreteness, the *STARTS* specification and examples that we give below use SOIFs just to illustrate how our content can be delivered. However, *STARTS* includes mechanisms to specify other formats for its contents.

[4] SOIF objects are typed, ASCII-based encodings for structured objects; see `http://harvest.transarc.com/afs/transarc.com/public/trg/Harvest/user-manual/`.

4.1 Query Language

In this section we describe the basic features of the query language that a source should support. To cover the functionality offered by most commercial search engines, queries have both a Boolean component: the *filter expression*, and a vector-space component: the *ranking expression*. (See Section 4.1.1.) Also, queries have other associated properties that further specify the query results. For example, a query specifies the maximum number of documents that should be returned, among other things. (See Section 4.1.2.)

4.1.1 Filter and Ranking Expressions

Queries have a filter expression (the Boolean component) and a ranking expression (the vector-space component). The *filter expression* specifies some condition that must be satisfied by every document in the query result (e.g., all documents in the answer must have "Ullman" as one of the authors). The *ranking expression* specifies words that are desired, and imposes an order over the documents in the query result (e.g., the documents in the answer will be ranked according to how many times they contain the words "distributed" and "databases" in their body).

Example 1 *Consider the following query with filter expression:*

```
((author ``Ullman'') and (title ``databases''))
```

and ranking expression:

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

This query returns documents having "Ullman" 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. Some search engines only support filter or ranking expressions, but not both (e.g., Glimpse only supports filter expressions). Therefore, we allow sources to support just one type of expression. In this case, the sources indicate (Section 4.3.1) what type they support as part of their metadata.

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 ``Ullman'') and (title ``databases''))`). The ranking expressions also combine terms using the "`list`" operator, which simply groups together a set of terms, as in Example 1. Also, the terms of a ranking expression may have a weight associated with them, indicating their relative importance in the ranking expression.

In defining the expressive power of the filter and ranking expressions we had to balance the needs of search engine builders and metasearchers. On the one hand, builders in general want powerful expressions, so that all the features of their engine can be called upon. On the other hand, metasearchers want simpler filter and ranking expressions, because they know that not all search engines support the same advanced features. The simpler the filter and ranking expressions are, the more likely it is that engines will have common features, and the easier it will be to interoperate. Also, those metasearchers whose main market is Internet searching prefer simple expressions because most of their customers use simple queries. In contrast, search engine builders cater to a broader mix of customers. Some of these customers require sophisticated query capabilities.

Next we define the filter and ranking expressions more precisely. We start by defining the *l-strings*, which are the basic building blocks for queries. Then we show how these strings are adorned with fields and modifiers to build atomic terms. Finally, we describe how to construct complex filter and ranking expressions.

Atomic Terms

One of the most heavily discussed issues in our workshop was how to support multiple languages and character sets. Our initial design had not supported queries using multiple character sets or languages. However, the search engine vendors felt strongly against this limitation. So, we decided early on in our workshop to include multi-lingual/character support, but the question was how far to go. For example, did we want to support a query asking for documents with the Spanish word "taco"? Did we also want to handle queries asking for documents whose abstract was in French, but that also included the English word "weekend"? Another issue was how to handle dialects, e.g., how to specify that a document is written, say, in British English vs. in American English.

During the workshop we also discussed whether we could make the multi-language support invisible to those who just wanted to submit English queries. That is, we do not want to specify English explicitly everywhere if no other language is used. The design we settled on does allow English and ASCII as the defaults, while giving the query writer substantial power to specify languages and character sets used.

A *term* in our query language is an *l-string* modified by an unordered list of *attributes* (e.g., `(author ``Ullman'')`). To allow queries in languages other than English, an l-string is either a string (e.g., `` ``Ullman'' ``), or a string qualified with its associated language and, optionally, with its associated country. For example, `[en-US ``behavior'']` is an l-string, meaning that the string "behavior" represents a word in American English. The language-country qualification follows the format described in RFC 1766 (`http://andrew2.andrew.cmu.edu/rfc/rfc1766.html`). (Countries are optional.) 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.

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.

To make interoperability easier, we decided to define a "recommended" set of attributes that sources should try to support. This set needed to be large enough so that users can express their queries. At the same time, the set needed to be simple enough to not compromise interoperability. The choice of the recommended attribute set was fodder for heated discussion, especially around what attributes we should require the sources to support. In effect, requiring that sources support some attributes would make the protocol more expressive, but harder to adhere to by the sources.

We considered several candidate attribute sets that had already been defined within different standards efforts. (See Section 5.) Unfortunately, none of the existing attribute sets contained just the attributes that we needed, as determined from our discussions. Therefore, we decided to pick the GILS [5] attribute set, which in turn inherits all of the Z39.50-1995 Bib-1 use attributes. (See Section 5.) The GILS set contained most of the attributes that we needed, and we simplified it to include only those attributes. We also added a few attributes that were not in the GILS set but that were considered necessary in our discussions.

Below is the "Basic-1" set of attributes (i.e., fields and modifiers), which are the attributes that we recommend that sources support. The attributes not marked as new are from the GILS attribute set. In [1] we explain how to use other attribute sets for sources covering different domains, for example.

- **Fields**: A field specifies what portion of the document 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 as required must be supported, meaning 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/GILS "use attributes.")

Field	*Required?*	*New?*
`Title`	Yes	No
`Author`	No	No
`Body-of-text`	No	No
`Document-text`	No	Yes
`Date/time-last-modified`	Yes	No
`Any`	Yes	No
`Linkage`	Yes	No
`Linkage-type`	No	No
`Cross-reference-linkage`	No	No
`Languages`	No	No
`Free-form-text`	No	Yes

The `Document-text` field provides a way to pass documents to the sources as part of the queries, which could be useful to do *relevance feedback* [2]. Relevance feedback allows users to request documents that are similar to a document that was found useful.

The value of the `Linkage` field of a document is its URL, and it is returned with the query results so that the document can be retrieved outside of our protocol.

The `Linkage-type` of a document is its MIME type, while its `Cross-reference-linkage` is the list of the URLs that are mentioned in the document.

The `Free-form-text` field provides a way to pass to the sources queries that are not expressed in our query language, adding flexibility to our proposal. A search engine vendor asked for this capability so that informed metasearchers could use the sources' richer native query languages, for example.

- **Modifiers**: A modifier specifies what values the term represents (e.g., treat the term as a stem, as its phonetics (soundex), etc.). Zero or more modifiers can be

[5] The Government Information Locator Service, GILS, is an effort to facilitate access to governmental information.

specified for each term. All the modifiers below are optional, i.e., the search engines need not support them. (Our modifiers correspond to the Z39.50 "relation attributes.")

Modifier	*Default*	*New?*
`<, <=, = >=, >, !=`	=	No
`Phonetic`	No soundex	No
`Stem`	No stemming	No
`Thesaurus`	No thesaurus expansion	Yes
`Right-truncation`	No right truncation	No
`Left-truncation`	No left truncation	No
`Case-sensitive`	Case insensitive	Yes

The `<, <=, =, >=, >, !=` modifiers only make sense for fields like "`Date/time-last-modified`," for example.

Example 2 *Consider the following filter expression:*

```
(title stem ``databases'')
```

The documents that satisfy this expression have the word "databases" in their title, or some other word with the same stem, like "database."

Complex Filter Expressions

Our complex filter expressions are based on a simple subset of the type-101 queries of the Z39.50-1995 standard. We use operators to build complex filter expressions from the terms. As with the attributes, we wanted to choose a set of operators that would both be easy to support and be sufficiently expressive. The "Basic-1"-type filter expressions use the following operators. 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 3 *Consider two terms* t_1 *and* t_2 *and the following filter expression:*

$$(t_1 \ \texttt{prox[3,T]} \ t_2)$$

The documents that match this filter expression contain t_1 *followed by* t_2 *with at most three words in between them.* ``` ``T'' ``` *(for "true") indicates that the word order matters (i.e., that* t_1 *has to appear before* t_2*).*

Note that `not` is not one of our operators, to prevent users from asking for documents with the sole qualification that they not contain the word "databases" in them, for example. Such a query would be too expensive to evaluate. Instead, we have the `and-not` operator. Thus, all queries always have a "positive" component.

The proximity operator is an interesting example of a compromise that we had to reach: some search engine vendors found our initial proposal, which allowed for unidirectional or bidirectional "paragraph" and "sentence" distance, for example, unacceptably complicated to implement. Later, we simplified the proximity operator to only allow for unidirectional word distance. A search engine vendor still thought that this operator was too complicated, while other participants, especially information providers, found it unreasonably limiting. We finally managed to agree on the current specification.

Complex Ranking Expressions

We also use operators to build complex ranking expressions from 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 "`list`" operator represents the most common way of constructing vector-space queries: these queries are typically just flat lists of terms. Our original design did not allow for any other operators in the ranking expressions. However, some search engine vendors felt that this language was not expressive enough, and asked that the Boolean-like operators be included. If a source supports ranking expressions, it must now support all these operators. But again, a source might choose to simply ignore the Boolean-like operators from ranking expressions, and process a ranking expression like ``` (``distributed'' and ``databases'') ``` as if it were ``` list(``distributed'' ``databases'') ```.

The Boolean-like operators would most likely be interpreted as "fuzzy-logic" operators by the search engines in order to rank the documents, as the following example illustrates.

Example 4 *Consider two ranking expressions:*

```
R1 = (``distributed'' and ``databases'')
R2 = list(``distributed'' ``databases'')
```

Consider a source with a document that has a weight (as determined by the local search engine) of 0.3 for the word "distributed" and a weight of 0.8 for the word "databases." Then, the search engine might assign the document a score of $\min\{0.3, 0.8\} = 0.3$ *for ranking expression* R_1 *(interpreting the* `and` *operator as the* `minimum` *function). The same engine might use a different scoring algorithm for "*`list`*" queries with the same terms, and assign the document a score of, say,* $0.5 * 0.3 + 0.5 * 0.8 = 0.55$ *for ranking expression* R_2*.*

Thus, by interpreting the Boolean-like operators and the `list` *operator for building ranking expressions differently, sources can provide richer semantics for user queries.*

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

Example 5 *Consider the following ranking expression:*

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

The weights in the expression indicate that the search engines should treat the term ``` ``distributed'' ``` *as more important than the term* ``` ``databases'' ``` *when ranking the documents in the query results.*

4.1.2 Further Result Specification

To complete the specification of the query results, our queries include the following information in addition to a filter and a ranking expression:

- **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 4.3).
- **Default attribute set and language** used in the query. This is optional, just for notational convenience, since queries may include attributes from attribute sets other than "Basic-1," and terms may correspond to languages other than English.
- **Sources** (in the same resource) where to evaluate the query in addition to the source where the query is submitted. (See Section 3.)
- **Answer specification**:
  - Fields to be returned in the query answer (Default: `Title`, `Linkage`)
  - Fields to be used to sort the query results, and whether the order is ascending or descending (Default: `Score` of the documents for the query, in descending order.)
  - Documents to be returned:
    * Minimum acceptable document score
    * Maximum acceptable number of documents

A complete query is represented as a list of attribute-value pairs, providing the filter and ranking expressions, the answer specification, etc. Below is an example of such a query encoded using Harvest's SOIF. As discussed in Section 4, we encode the *STARTS* information using SOIF here just to illustrate how our query content could be delivered, but other encodings are possible.

Example 6 *Below is a SOIF object for a query. The number in brackets after each SOIF attribute (e.g., ``48'' 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{48}: ((author ``Ullman'') and
                       (title stem ``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
}
```

This query specifies that the sources should eliminate any stop words from the filter and ranking expressions before processing them, and that the word "databases" in the filter expression should be stemmed. Then, for example, a document having the word "database" in its title will match the subexpression `(title stem ``databases'')`*. The query results should contain the* `title` *and* `author` *of the documents, in addition to the* `linkage` *(URL) of the documents, which is always returned. Also, only documents with a score for the ranking expression of at least 0.5 should be in the answer. Furthermore, only the 10 documents with the top score are to be returned.*

4.2 Merging of Ranks

There are three types of complications that arise in interpreting query results from multiple sources. One is that each source may execute a different query, depending on its local query capabilities. Thus, a source might ignore parts of a query that it does not support, for example. Another complication is that sources may use different algorithms to rank the documents in the query results. Furthermore, the sources do not reveal their ranking algorithms. A third complication is that the ranking information by itself is insufficient for merging multiple query results, even if all the sources execute the same query using the same ranking algorithm. In effect, the actual document ranks might depend on the contents of each source, as described in Section 3.2. We will now discuss how our protocol copes with these issues.

As mentioned above, sources are not required to support all of the features of the query language of Section 4.1. So, a source might decide to ignore certain parts of a query that it receives, for example. Then, each source returns the query that it actually processed together with the query results, as the following example illustrates. Since we do not include any way of reporting errors in our protocol, this mechanism assists the metasearchers in interpreting the query results.

Example 7 *Consider a source that does not support the ranking-expression part of queries. Consider the query with filter expression:*

```
((author ``Ullman'') and (title stem
                  ``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 ``Ullman'') and (title stem
                  ``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 [2] 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 KBytes)
- `Document-count`: the number of tokens (as determined by the source) in the document

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.

Example 8 *The result for the query of Example 6 from the Source-1 source may look like the following.*

```
@SQResults{
Version{10}: STARTS 1.0
Sources{8}: Source-1
ActualFilterExpression{48}: ((author ``Ullman'')
                  and (title stem ``databases''))
ActualRankingExpression{26}: (body-of-text
                               ``databases'')
NumDocSOIFs{1}: 1
}

@SQRDocument{
Version{10}: STARTS 1.0
RawScore{4}: 0.82
Sources{8}: Source-1
linkage{47}:
   http://www-db.stanford.edu/~ullman/pub/dood.ps
title{68}: A Comparison Between Deductive and
   Object-Oriented Database Systems
author{18}: Jeffrey D. Ullman
TermStats{89}:
   (body-of-text ``distributed'') 10 0.31 190
   (body-of-text ``databases'') 15 0.51 232
DocSize{3}: 248
DocCount{5}: 10213
}
```

The first SOIF object reports properties of the entire query result. For example, we learn from the value of `Actual-RankingExpression` *that Source-1 eliminated the term* `(body-of-text ``distributed'')` *from the ranking expression. Presumably, the word "distributed" is a stop word at Source-1. We also find out that there is only one document in the query result. All of the other documents in Source-1 either do not satisfy the filter expression, or have a score lower than 0.5 for the ranking expression.*

The second SOIF object corresponds to the only document in the query result. This document, whose URL is given as the value for the `linkage` *attribute, has a score of 0.82 for the ranking expression, and satisfies the filter expression. In effect, the word "database" appears in the document's title ("database" shares its stem with "databases"), and "Ullman" is one of the authors of the document.*

The document SOIF object also contains statistics about the document, which are crucial for rank merging. For example, we know that the word "distributed" appears 10 times in the document, and the word "databases" 15 times. The size of the document is 248 KBytes, and there are 10,213 words in it.

Using all this information, a metasearcher can then rerank the documents that it obtained from the various sources, following its own criteria and without actually retrieving the documents, as the following example illustrates.

Example 9 *Consider the following SOIF object describing the only document in the result for the query of Example 6 from source Source-2.*

```
@SQRDocument{
Version{10}: STARTS 1.0
RawScore{4}: 0.27
Sources{8}: Source-2
linkage{37}: http://elib.stanford.edu/lagunita.ps
title{73}: Database Research: Achievements and
       Opportunities into the 21st. Century
author{48}: Avi Silberschatz, Mike Stonebraker,
       Jeff Ullman
TermStats{89}:
       (body-of-text ``distributed'') 20 0.12 901
       (body-of-text ``databases'') 34 0.15 788
DocSize{3}: 125
DocCount{4}: 9031
}
```

This document has a lower score than the document from Source-1 of Example 8. However, the Source-2 document might be a better match for the query than the Source-1 document, and the lower score could just be an artifact of the ranking algorithm that the sources use, or be due to the characteristics of the holdings of both sources. A metasearcher could then simply discard the sources' scores, and compute a new score for each document based on, say, the number of times that the words in the ranking expression appear in the documents. Then, such a metasearcher would rank the Source-2 document higher than the Source-1 document, since the former document contains the words "distributed" and "databases" 20 and 34 times, respectively, whereas the latter document only contains these words 10 and 15 times, respectively.

Example 9 shows one simple-minded way in which a metasearcher can re-rank documents from multiple query results. More sophisticated schemes could also use the document frequencies of the query terms, for example. However, there are still unresolved issues when merging document ranks from multiple sources. For example, one possibility is to rank documents as if they all belonged in a single, large document source. Alternatively, we could use information about the originating sources to design the final document rank. The goal of our protocol is not to resolve these issues, but simply to provide the "raw material" so that metasearchers can experiment with a variety of formulas and approaches for combining multiple query results.

From our discussions with the search engine vendors, it became clear that it would be hard for some of them to provide the statistics above with their query results. The reason is that by the time the results are returned to the user, these statistics, which are typically used to compute the document scores, are lost. Since returning just the document scores with the query results is not enough for rank merging, we are asking sources to at least provide the query results for a given *sample document collection* and a given set of queries as part of their metadata. We are currently investigating how to design this sample collection and queries. This way, the metasearchers would treat each source as a "black box" that receives queries and produces document ranks. However, the metasearchers would try to approximate how each source ranks documents using their knowledge of what is in the sample collection. So, if the sample queries are carefully designed, the metasearchers might be able to draw some conclusions on how to calibrate the query results in order to produce a single document rank.

4.3 Source Metadata

To select the right sources for a query and to query them we need information about the sources' contents and capabilities. In this section we describe two pieces of metadata that every source is required to export: a list of metadata attribute-value pairs, 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. For simplicity, each of these two pieces is retrieved as a single "blob." We do not ask sources to support more sophisticated interfaces, like a search interface, to export this data.

In this section we also describe the information that a resource exports. This information identifies the metadata for the sources in the resource.

4.3.1 Source Metadata Attributes

Each source exports information about itself by giving values to the metadata attributes below. A metasearcher can use this information to rewrite the queries that it sends to each source, since each source may support different parts of the query language of Section 4.1, for example.

As with the attribute sets for documents, several attribute sets have been defined to describe sources. Unfortunately, none of these sets contain exactly the attributes that we need, as determined from our discussions. Therefore, we defined the "MBasic-1" set of metadata attributes, borrowing from two well known attribute sets, the Z39.50-1995 Exp-1 and the GILS attribute sets. We added a few attributes, marked as new below, that are not in these two attribute sets, and that the participating organizations concluded were necessary. Some attributes are marked as required, and the sources must support them.

Field	*Required?*	*New?*
`FieldsSupported`	Yes	Yes
`ModifiersSupported`	Yes	Yes
`FieldModifierCombinations`	Yes	Yes
`QueryPartsSupported`	No	Yes
`ScoreRange`	Yes	Yes
`RankingAlgorithmID`	Yes	Yes
`TokenizerIDList`	No	Yes
`SampleDatabaseResults`	Yes	Yes
`StopWordList`	Yes	Yes
`TurnOffStopWords`	Yes	Yes
`SourceLanguages`	No	No
`SourceName`	No	No
`Linkage`	Yes	No
`ContentSummaryLinkage`	Yes	Yes
`DateChanged`	No	No
`DateExpires`	No	No
`Abstract`	No	No
`AccessConstraints`	No	No
`Contact`	No	No

The `FieldsSupported` attribute for a source lists the optional fields (Section 4.1.1) that are supported at the source for querying, in addition to the required fields like `Linkage` and `Title`. 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.

Similarly, the `ModifiersSupported` attribute lists the modifiers (Section 4.1.1) that are supported at a source. Each modifier is optionally accompanied by a list of the languages for which it is supported at the source, since modifiers like `Stem` are language dependent.

To keep the metadata objects simple, we do not require sources to indicate what supported fields are actually searchable, as opposed to being only retrievable. For example, a source might report the `Language` of each document in the query results, although it might not accept queries involving that field. However, we do ask sources to report what combinations of fields and modifiers are legal at the source in the `FieldModifierCombinations` attribute. For example, asking that an author name be stemmed might be illegal at a source, even if the `Author` field and the `Stem` modifier are supported in other contexts at the source.

The `QueryPartsSupported` attribute specifies whether the source supports ranking expressions only, filter expressions only, or both.

The `ScoreRange` attribute lists the minimum and maximum score that a document can get for a query at the source (including $-\infty$ and $+\infty$); we use this information for merging ranks, to interpret the scores that come from the sources.

The `RankingAlgorithmID` attribute contains some form to identify the ranking algorithm the source uses. Even when we do not know the actual algorithm used it is useful to know that two sources use the same algorithm (e.g., `Acme-1`), for merging ranks.

The `TokenizerIDList` attribute has values like `(Acme-1 en-US) (Acme-2 es)`, for example, meaning that the source uses tokenizer Acme-1 to extract the indexable tokens from strings in American English, and tokenizer Acme-2 for strings in Spanish. The inclusion of this metadata attribute was controversial: our original proposal required that sources export a list of the characters that they used as token separators (e.g., " ,;.", meaning that a blank space, a comma, etc., are used to delimit tokens). Obviously this information would not be sufficient to completely specify the tokens at each source, but it could at least help metasearchers decide if a query on "Z39.50" should include this term as is, or should instead contain two terms, namely "Z39" and "50," for example, as would be the case if "." were a separator. Alternatively, it was proposed that sources export some regular expression describing what their tokens looked like. Both of these proposals were not general enough to describe tokens for arbitrary languages and character sets, and were deemed too complicated to support. Therefore, we settled on the current proposal, which simply requires that sources name their tokenizers. This way, a metasearcher can learn how a particular tokenizer works by submitting a query to a source that uses it, and examining the actual query that the source processes, as specified in the query results (Section 4.2). A metasearcher would need to do this not on a source-by-source basis, but only once per tokenizer.

The `SampleDatabaseResults` attribute provides the URL to get the query results for a sample document collection (Section 4.2).

The `Linkage` attribute reports the URL where the source should be queried, while the `ContentSummaryLinkage` attribute gives the URL of the content summary of the source (Section 4.3.2).

Example 10 *Consider the following SOIF object with some of the metadata attributes for a source Source-1:*

```
@SMetaAttributes{
Version{10}: STARTS 1.0
SourceID{8}: Source-1
```

```
FieldsSupported{17}: [basic-1 author]
ModifiersSupported{19}: {basic-1 phonetics}
FieldModifierCombinations{39}: ([basic-1 author]
                                {basic-1 phonetics})
QueryPartsSupported{2}: RF
ScoreRange{7}:  0.0 1.0
RankingAlgorithmID{6}: Acme-1
...
DefaultMetaAttributeSet{8}: mbasic-1
source-languages{8}: en-US es
source-name{17}: 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
}
```

This source supports the `Author` *field for searching, in addition to the required fields, and the* `Phonetics` *modifier. It also accepts queries with both filter and ranking expressions, and the document scores it produces range from 0 to 1. Source-1 contains documents in American English (*`en-US`*) and Spanish (*`es`*). Queries should be submitted to this source at* `http://www-db.stanford.edu/cgi-bin/query`*, and its content summary is available at* `ftp://www-db.stanford.-edu/cont_sum.txt`*.*

4.3.2 Source Content Summary

Content summaries help the metasearchers in choosing the most promising sources for a given query. These summaries could be manually generated, like the ones associated with the `Abstract` metadata attribute of Section 4.3.1. However, this approach usually yields outdated and incomplete summaries, and is a burden on the source administrators.

On the other end of the spectrum, a source summary could simply be the entire contents of the source. This approach is similar to the one taken by several World-Wide Web "crawlers."

In addition to its `Abstract`, we require that each source export partial data about its contents. This data is automatically generated, is orders of magnitude smaller than the original contents, and has proven useful in distinguishing the more useful from the less useful sources for a given query [7, 8]. Our content summaries include:

- List of words that appear in the source. This list is preceded by information indicating 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 or not (e.g., `(title ``algorithm'')`).

  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

Example 11 *Consider the following SOIF object with part of the content summary for Source-1 from Example 10:*

```
@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
...
}
```

This content summary reports statistics on unstemmed, case insensitive words that are qualified with field information. For example, the English word "algorithm" appears in the title of 53 documents, while the Spanish word "datos" appears in the title of 12 documents in Source-1. The summary also tells us that there are 892 documents in the source. A metasearcher can use this information to decide whether a given query is likely to have good matches in Source-1 [7, 8].

4.3.3 Resource Definition

So far, we have focused on *sources*. As discussed in Section 3, our model allows several sources to be grouped together as a single *resource* (e.g., Knight-Ridder's Dialog information service). 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. Using this information, a metasearcher learns how and where to contact each of the sources in the resource.

Example 12 *Consider the following SOIF object with contact information for a resource. This object reports that there are two sources available for querying at the resource, Source-1 and Source-2, and also gives the URLs where to obtain their corresponding metadata-attribute SOIF objects.*

```
@SResource{
Version{10}: STARTS 1.0
SourceList{83}:
      Source-1 ftp://www.stanford.edu/source_1
      Source-2 ftp://www.stanford.edu/source_2
}
```

5 Related Efforts

There have been many efforts aimed at supporting the major functions of digital libraries such as search, assessment, and acquisition of information. In this section we discuss work relevant to our proposal. In particular, we first look at standards or architectures for information retrieval and interoperability. We then discuss attribute sets and their associated mechanisms for describing documents and sources. Finally, we comment on existing Internet metasearchers.

The most relevant standards effort in terms of shared goals is the Z39.50-1995 standard [9], which provides most of the functionality we have described. For instance, its Explain facility requires the Z39.50 servers to export their "source metadata" so that the clients can dynamically configure themselves to match individual servers, thus providing the option to support more than the least common denominator of the servers. The standard also specifies query languages such as the type-101 query language, which we used in Section 4.1. In addition, the Scan service allows the clients to access the sources' contents incrementally.

While similar in functionality, our proposal is much simpler than Z39.50, and keeping it simple was one of our main concerns. Moreover, as our proposal is specifically tailored to facilitate metasearching, we require some information not exported in Z39.50. For example, we need the term and document statistics as part of the query results to help in merging multiple document ranks. However, we do see our proposal as a step toward bridging the gap between the library community where Z39.50 has been widely used and the Internet search community. As we mentioned in the Introduction, there are currently efforts under way to define a simple profile of the Z39.50 standard based on *STARTS* [10].

In addition to the Z39.50 standard effort, other projects focus on providing a framework for indexing and querying multiple document sources. One such project, Harvest [11], includes a set of tools for gathering and accessing information on the Internet. The Harvest *gatherers* collect and extract indexing information from one or more sources. Then, the *brokers* retrieve this information from one or more gatherers, or from other brokers. The brokers provide a querying interface to the gathered information. A project related to Harvest is Netscape's RDM (Resource Description Messages) [12], which focuses on indexing and accessing structured metadata descriptions of information objects. Our work complements Harvest and RDM in that we define the information and functionality that sources should export to help in metasearching. Thus, the Harvest brokers (or RDM clients) could act as metasearchers, and benefit from the *STARTS* information that sources export.

Other related efforts focus on defining attribute sets for documents and sources. As discussed in the paper, we have built on some of these efforts in defining our protocol. Relevant attribute sets for documents include the Z39.50 Bib-1 attribute set [13], the Dublin Core [14], and the Warwick Framework [15]. The Bib-1 attribute set registers a large set of bibliographic attributes that have been widely adopted in library cataloging. On the other hand, the focus of the Dublin Core is primarily on developing a simple yet usable set of attributes to describe the essential features of networked documents (e.g., World-Wide Web documents), which is also the intention of our "Basic-1" set. The Warwick Framework proposes a container architecture as a mechanism for incorporating attribute values from different sets in a single information object. In contrast, we chose to support only a simple, "flat" document model, albeit with the ability to mix different attribute models [1]. Regarding source-metadata attribute sets, the most notable efforts include the Z39.50 Exp-1 attribute set [9] and the GILS profile [16], upon which we based our "MBasic-1" attribute set (Section 4.3.1).

Several metasearchers already exist on the Internet for querying multiple World-Wide Web indexes. However, not all of them support the three major metasearch tasks described in Section 1 (i.e., selecting the best sources for a query, querying these sources, and merging the query results from the sources). Examples of metasearchers include MetaCrawler [17] (`http://metacrawler.cs.washington.edu:8080/home.html`) and SavvySearch (`http://guaraldi.cs.colostate.edu:2000/`).

Both MetaCrawler and SavvySearch support the three metasearch tasks above to some degree. First, they provide some sort of source selection. For example, SavvySearch ranks its accessible sources for a given query based on information from past searches and estimated network traffic. Second, they support a unified query interface for accessing the underlying sources, although this interface tends to be the least common denominator of that of the underlying sources. For query features that are not supported uniformly by the underlying sources, a post-filtering step is required for the metasearcher to locally implement the missing functionality. For instance, MetaCrawler processes phrase searches in the "verification mode." Third, these metasearchers re-rank the documents in the query results. Specifically, MetaCrawler re-ranks the documents by actually retrieving and analyzing them. SavvySearch simply reports the documents according to the originating sources, and using the source rank mentioned above.

Finally, the Stanford InfoBus, designed within the Digital Library project [18], hosts a variety of metasearchers. In [19], we discuss a metadata architecture for the InfoBus. This architecture is based on the requirements of the InfoBus services, and uses the *STARTS* information that sources should export.

6 Conclusion

The Internet and the World-Wide Web have become an important component of many data management systems. As such, searching and resource discovery across Internet resources has become a central issue, and we believe that the *STARTS* protocol provides simple but fundamental facilities for such access. If implemented, *STARTS* can significantly streamline the implementation of metasearchers, as well as enhance the functionality they can offer. We also think that our discussion of issues and "tensions" emerging from our *STARTS* experience can provide useful lessons for anyone dealing with Internet data access.

References

[1] Luis Gravano, Chen-Chuan Kevin Chang, Héctor García-Molina, and Andreas Paepcke. STARTS: Stanford protocol proposal for Internet retrieval and search. Technical Report SIDL-WP-1996-0043, Stanford University, August 1996. Accessible at `http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0043`.

[2] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison Wesley, 1989.

[3] Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. Boolean query mapping across heterogeneous information sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):515–521, August 1996.

[4] Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. Predicate rewriting for translating boolean queries in a heterogeneous information system. Technical Report SIDL-WP-1996-0028, Stanford University, 1996. Accessible at `http://www-diglib.stanford.edu/cgi-bin/-WP/get/SIDL-WP-1996-0028`.

[5] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the* 18^{th} *Annual SIGIR Conference*, 1995.

[6] Ellen M. Voorhees, Narendra K. Gupta, and Ben Johnson-Laird. The collection fusion problem. In *Proceedings of the* 3^{rd} *Text Retrieval Conference (TREC-3)*, 1995.

[7] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. The effectiveness of *GlOSS* for the text-database discovery problem. In *Proceedings of the 1994 ACM SIGMOD Conference*, May 1994.

[8] Luis Gravano and Héctor García-Molina. Generalizing *GlOSS* for vector-space databases and broker hierarchies. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 78–89, September 1995.

[9] National Information Standards Organization. *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification (ANSI/NISO Z39.50-1995)*. NISO Press, Bethesda, MD, 1995. Accessible at `http://lcweb.loc.gov/z3950/agency/`.

[10] ZDSR profile: Z39.50 profile for simple distributed search and ranked retrieval, Draft 4, October 1996. Accessible at `http://lcweb.loc.gov/z3950/agency/-profiles/zdsr.html`.

[11] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, Michael F. Schwartz, and Duane P. Wessels. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, Colorado, August 1994. Accessible at `http://harvest.transarc.com/`.

[12] Darren Hardy. Resource description messages (RDM), July 1996. Accessible at `http://www.w3.org/pub/-WWW/TR/NOTE-rdm.html`.

[13] Z39.50 Maintenance Agency. Attribute set Bib-1 (Z39.50-1995): Semantics. Accessible at `ftp://-ftp.loc.gov/pub/z3950/defs/bib1.txt`, September 1995.

[14] Stuart Weibel, Jean Godby, Eric Miller, and Ron Daniel, Jr. OCLC/NCSA metadata workshop report. Accessible at `http://www.oclc.org:5047/-oclc/research/publications/weibel/metadata/-dublin_core_report.html`, March 1995.

[15] Carl Lagoze, Clifford A. Lynch, and Ron Daniel, Jr. The Warwick Framework: A container architecture for aggregating sets of metadata. Technical Report TR96-1593, Cornell University, Computer Science Department, June 1996.

[16] Eliot Christian. Application profile for the government information locator service (GILS), Version 2, October 1996. Accessible at `http://www.usgs.gov/gils/-prof_v2.html`.

[17] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th International WWW Conference*, 1995.

[18] Andreas Paepcke, Steve B. Cousins, Héctor García-Molina, Scott W. Hassan, Steven K. Ketchpel, Martin Röscheisen, and Terry Winograd. Towards interoperability in digital libraries: Overview and selected highlights of the Stanford Digital Library Project. *IEEE Computer Magazine*, May 1996.

[19] Michelle Baldonado, Chen-Chuan K. Chang, Luis Gravano, and Andreas Paepcke. The Stanford Digital Library Metadata Architecture. Technical Report SIDL-WP-1996-0051, Stanford University, October 1996. To appear in the International Journal of Digital Libraries; accessible at `http://www-diglib.stanford.edu/cgi-bin/-WP/get/SIDL-WP-1996-0051`.