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."

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

(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.

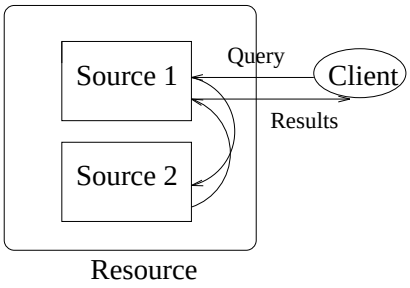

Figure 1: A metasearcher queries a source, and may specify that the query be evaluated at several sources at the same resource.

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).

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

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.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

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

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

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.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.