Reflections on SDLIP and other search interfaces

Sergio Guzman-Lara & Greg Janée
December 11, 2002

1) The problem

Designing and implementing a search interface is a relatively simple problem: all that's required is a means of submitting a search request and returning either results or a handle to a result set. There is no need to rely on complex, third-party code. So any kind of simple interface based on HTTP or SOAP would work just fine in today's distributed computing environment.

The real value of adhering to a *standard* search interface is the facilitation of interoperability. There are two kinds of interoperability to consider. "Internal interoperability" is interoperability amongst NSDL providers, i.e., is CI-mandated. This kind of interoperability would make it easier for clients of NSDL to work with multiple NSDL search services. "External interoperability" is interoperability amongst clients and providers outside NSDL's scope, e.g., among the community of traditional libraries.

It should be noted that search interfaces are often separated from query languages—and query language interoperability is an equally large, if not larger problem than search interface interoperability. From a client's perspective, it's not pragmatically useful that 50 search engines all speak the same search interface/protocol if there are 50 different query languages to contend with.

However, getting consensus on query languages is difficult. Currently there is no query language that would give us the wide interoperability we would like: no query language (at least for textual search engines) is being used by such a large community that would make it a clear choice. Furthermore, it is unclear if CQL (see below), or any other proposal, will ever become a "standard."

Still, there may be value in NSDL providing an intermediate approach that standardizes the way boolean constraints, ranking, and certain other high-level query features are expressed, while leaving constraint types and expressions to be defined by the individual search engines or families of search engines that support them. This approach is demonstrated in a limited way by the ADL query language (http://www.alexandria.ucsb.edu /middleware/dtds/ADL-query.dtd).

The remainder of this document focuses mostly on search interfaces.

2) Why we chose SDLIP

Although we did not have formal requirements, what we were basically looking for was a protocol for the communication between portals and the search server that

allowed us to use our own query language. At the time we discarded Z39.50 because we thought it was too complex for our needs, and chose SDLIP because:

- It is not tied to a particular query language.
- It has support for load balancing.
- It is simple.
- There is already an open source implementation in Java.
- At the time it seemed to be a likely future standard for "search middleware."
- It has support for stateful servers (in case we ever need this).

3) Problems with SDLIP

- The SDLIP "protocol" is a misnomer. SDLIP is not actually a wire protocol, but rather, a set of standard APIs and open source toolkits. While the existence of toolkits greatly simplifies client and server construction, the lack of an underlying, true protocol is a weakness in this case. It *forces* the use of the toolkits, which means that users for whom there is no toolkit available are largely out of luck. So it's not surprising that…

- The main problem we encountered is that some portals, depending on the technology they use, have problems talking to the current Java implementation of the SDLIP client. For example, the group at Utah State University uses PHP, and currently there is no easy way to have a PHP-Java communication. Similarly, the WGBH group uses Perl, so they are figuring out how to communicate from Perl to Java (implementing SDLIP in Perl seems harder).

- Describing the search server's services via WSDL (assuming that we decide to) might be more difficult to do if we stick with SDLIP than if we move to SOAP.

- SDLIP provides no real external interoperability, as there is no viable community using the protocol (that we're aware of).

- SDLIP's real strengths may not be relevant to NSDL's needs. For example, SDLIP's support for both synchronous and asynchronous searching, and its division into multiple "interfaces" that can be hooked together in several different ways to better support mobile clients, may not be applicable to NSDL.

4) Z39.50

In contrast to SDLIP, there really *is* an existing federation of Z39.50 servers, and there has been for some time now. Thus, NSDL's support for Z39.50 would have real value to the extent that NSDL wants to support that community.

However, from a technical standpoint there is really nothing to recommend Z39.50, and its *only* value today is in the value of interoperating with legacy systems. It is simply too complex, dated, and burdened.

It may be worth noting that there are several projects that are trying to put simple wrappers around Z39.50, notably YAZ (http://www.indexdata.dk/yaz/) and ZOOM (http://zoom.z3950.org/). These are promising projects, but they fall short of the mark. The ADEPT project has tried using these toolkits with tantalizingly close success (i.e., we've created great-looking demos), but to date we've failed to create what we consider to be a robust, operational system out of either of them. The problem is that these toolkits are still quite buggy, and so when bugs are uncovered the simple wrapper becomes useless and one is forced to confront the intricacies of both Z39.50 and the toolkit's internal structures and mappings. ADEPT's development of a Z39.50 gateway has been stalled literally for months at a time while we have awaited charitable bug fixes from the open source developers. The lesson here is that you can put a gas pedal and wheels on an ocean liner and pretend it's a sports car, but that doesn't change the fact that it's still an ocean liner.

5) ZING/SRW

ZING/SRW (http://www.loc.gov/z3950/agency/zing/srw/) merits a closer look. Its name (Z39.50 International—Next Generation) leads one to believe that it's a revision of Z39.50, but from a first reading it appears to have almost no relation to Z39.50, and in fact it looks remarkably like other, contemporary search interfaces such as those offered by ADEPT, SDLIP, etc.

ZING's query language, Common Query Language (CQL), is yet another text-based query language for text searching. xCQL is a straight-forward XML encoding of CQL, but it is not sufficiently general to support the kind of extensibility mentioned above.

6) SOAP

Another alternative is of course to define ad hoc interfaces in SOAP, which seems to be gaining wide acceptability. Switching to SOAP, however, might not immediately solve the problem of easy client access: we still may need to write some (SOAP) clients in different programming languages. Moving to SOAP would require defining the methods to be implemented by our server (we could easily use the methods already defined in SDLIP) and write a SOAP implementation of them.

7) Summary

A search interface is a simple problem. A search interface that adheres to a standard is useful only to the extent that it facilitates internal and/or external interoperability.

Search interfaces are often distinct from query languages, and query language interoperability is equally important.

Z39.50 is an ocean liner among sports cars.

ZING/SRW may be more appropriate for NSDL's needs than SDLIP, and merits a closer look.