# Distributed Search in WAN Located Databases (Repositories) Using Z39.50 Protocol

Krasimir Trichkov[1]

*Abstract* – **This standard specifies a client/server based protocol for Information Retrieval. It specifies procedures and structures for a client to search a database provided by a server, retrieve database records identified by a search, scan a term list, and sort a result set. The protocol addresses communication between corresponding information retrieval applications, the client and server.**

*Keywords* – **Internet, Z39.50, Zebra, Yaz, Zap, Php, Database.**

## I. Introduction

Z39.50 [1] protocol specifies formats and procedures governing the exchange of messages between a client and server enabling the client to request that the server search a database and identify records which meet specified criteria, and to retrieve some or all of the identified records. This standard, ANSI/NISO Z39.50-1995 [2,3], *Information Retrieval (Z39.50) Application Service Definition and Protocol Specification*, is one of a set of standards produced to facilitate the interconnection of computer systems. It is positioned with respect to other related standards by the Open Systems Interconnection (OSI) basic reference model (ISO 7498). This standard defines a protocol within the application layer of the reference model, and is concerned in particular with the search and retrieval of information in databases.

## II. Basics of the Protocol

The client may initiate requests on behalf of a user; the protocol addresses communication between corresponding information retrieval applications, the client and server (which may reside on different computers); it does not address interaction between the client and user.

Z39.50 provides the following basic capabilities, all of which are supported in Z39.50 as well. The client may send a search, indicating one or more databases, and including a query as well as parameters which determine whether records identified by the search should be returned as part of the response. The server responds with a count of records identified and possibly some or all of the records. The client may then retrieve selected records. The client assumes that records selected by the search form a "result set" (an ordered set, order determined by the server), and records may be referenced by position within the set.

Optional capabilities include:

1. The client may specify an *element set* indicating data elements to retrieve in cases where the client does not wish to receive complete database records. For example, the client might specify "If 5 or less records are identified, transmit 'full' records; if more than 5 records are found, transmit 'brief' records".
2. The client may indicate a *preferred syntax* for response records, for example, USMARC [4,5].
3. The client may *name* a result set for subsequent reference.
4. The client may *delete* a named result set.
5. The server may impose *access control* restrictions on the client, by demanding authentication before processing a request.
6. The server may provide *resource control* by sending an unsolicited or solicited status report; the server may suspend processing and allow the client to indicate whether to continue.

Fig. 1 explains distributed search.

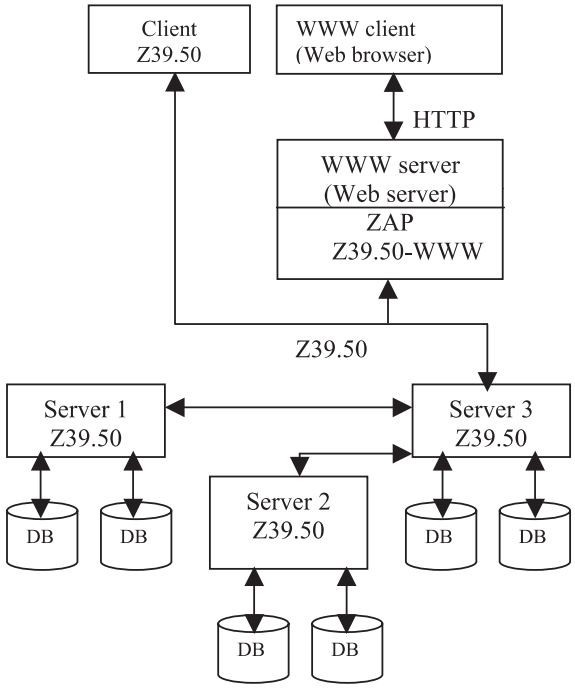


Fig. 1. Distributed search

[1]Krasimir Trichkov is with Institute of Computer and Communication Systems - Bulgarian Academy of Sciences, Acad. G. Bonchev bl.2, 1113 Sofia, Bulgaria, e-mail: krasi@hsi.iccs.bas.bg

This Z39.50 network was implemented in E-culture portal, which is part of the REGNET project (www.regnet.org). REGNET aims to set up a functional Network of Cultural Service Centers through Europe which will provide IT-services dedicated to cultural heritage organizations and it is funded by European Commission. For software development are used Apache web server [10] under Linux Operation System [11] (www.apache.org), Zebra Server [6] and Zap module [6]. To exchange data between REGNET centers is used Z39.50 protocol. Sample record that uses in Regnet is presented below.

Table 1. Example of record

<gils>
<Title> Spirit </Title>
<Creator> Romil Kalinov </Creator>
<Contribution> UBA </Contribution>
<Date> 2001.10.29 </Date>
<Description> Spirit, 2001 </Description>
<Identifier> 1 </Identifier>
<Type> Image </Type>
<Language> en </Language>
<Subject> Mixed technique </Subject>
<Publisher> 2001.11.06 </Publisher>
<Format> jpeg </Format>
<Source>Romil Kalinov , Sofia, Spirit, 2001, Mixed technique, 66 x 48 cm, Price: $ 135
Dimension 501x 709pixels, resolution 72 pixels/inch
</Source>
<Relation>
http://www3.iccs.bas.bg/RecordsUBA/romil.jpg
</Relation>
<Coverage> Contemporary Bulgarian Art
</Coverage>
<Rights> UBA </Rights>
</gils>

## III. Record Types

Indexing is a per-record process, in which either insert/modify/delete will occur. Before a record is indexed search keys are extracted from whatever might be the layout the original record (sgml,html,text, etc..). The Zebra system currently supports two fundamental types of records: structured and simple text. To specify a particular extraction process, use either the command line option -t or specify a recordType setting in the configuration file.

The Zebra system is designed to support a wide range of data management applications. The system can be configured to handle virtually any kind of structured data. Each record in the system is associated with a *record schema* which lends context to the data elements of the record. Any number of record schema can coexist in the system. Although it may be wise to use only a single schema within one database, the system poses no such restrictions.

Records pass through three different states during processing in the system.

1. When records are accessed by the system, they are represented in their local or native format. This might be SGML or HTML files, News or Mail archives, MARC records. If the system doesn't already know how to read the type of data you need to store, you can set up an input filter by preparing conversion rules based on regular expressions and possibly augmented by a flexible scripting language (Tcl). The input filter produces as output an internal representation:

2. When records are processed by the system, they are represented in a tree-structure, constructed by tagged data elements hanging off a root node. The tagged elements may contain data or yet more tagged elements in a recursive structure. The system performs various actions on this tree structure (indexing, element selection, schema mapping, etc.),

3. Before transmitting records to the client, they are first converted from the internal structure to a form suitable for exchange over the network - according to the Z39.50 standard.

The RecordType parameter in the *zebra.cfg* file, or the $-t$ option to the indexer tells Zebra how to process input records. Two basic types of processing are available - raw text and structured data. Raw text is just that, and it is selected by providing the argument *text* to Zebra. Structured records are all handled internally using the basic mechanisms described in the subsequent sections. Zebra can read structured records in many different formats. How this is done is governed by additional parameters after the "grs" keyboard, separated by "." characters.

Four basic subtypes to the *grs* type are currently available:

grs.sgml - This is the canonical input format – described below. It is a simple SGML-like syntax.

grs.regx.*filter* – This enables a user-supplied input filter. The mechanisms of these filters are described below.

grs.tcl.*filter* – Similar to grs.regx but using Tcl for rules.

grs.marc.*abstract syntax* – This allows Zebra to read records in the ISO2709 (MARC) encoding standard. In this case, the last parameter *abstract syntax* names the .abs file (see below) which describes the specific MARC structure of the input record as well as the indexing rules.

## IV. Canonical Input Format

Although input data can take any form, it is sometimes useful to describe the record processing capabilities of the system in terms of a single, canonical input format that gives access to the full spectrum of structure and flexibility in the system. In Zebra, this canonical format is an "SGML-like" syntax [6,7].

To use the canonical format specify grs.sgml as the record type.

Consider a record describing an information resource (such a record is sometimes known as a *locator record*). It might contain a field describing the distributor of the information resource, which might in turn be partitioned into various fields providing details about the distributor, like this in Table 2:

Table 2. Fields details

```
<Distributor>
<Name> ICCS </Name>
<Organization> ICCS </Organization>
<Street-Address>Sofia, Bulgaria</Street-Address>
<City> Sofia </City>
<Zip-Code> 1113 </Zip-Code>
<Country> Bulgaria </Country>
<Telephone> (359 2) 9792774 </Telephone>
</Distributor>
```

The keywords surrounded by <...> are *tags*, while the sections of text in between are the *data elements*. A data element is characterized by its location in the tree that is made up by the nested elements. Each element is terminated by a closing tag - beginning with </, and containing the same symbolic tag-name as the corresponding opening tag. The general closing tag - <>/ - terminates the element started by the last opening tag. The structuring of elements is significant. The element *Telephone*, for instance, may be indexed and presented to the client differently, depending on whether it appears inside the *Distributor* element, or some other, structured data element such a *Supplier* element.

## V. Record Root

The first tag in a record describes the root node of the tree that makes up the total record. In the canonical input format, the root tag should contain the name of the schema that lends context to the elements of the record. The following is a GILS record that contains only a single element (strictly speaking, that makes it an illegal GILS record, since the GILS profile includes several mandatory elements - Zebra does not validate the contents of a record against the Z39.50 profile, however - it merely attempts to match up elements of a local representation with the given schema) – Table 3:

Table 3. Match up elements

```
<gils>
<title> Geometry of motion </title>
</gils>
```

## VI. Variants

Zebra allows you to provide individual data elements in a number of *variant forms*. Examples of variant forms are textual data elements which might appear in different languages, and images which may appear in different formats or layouts. The variant system in Zebra is essentially a representation of the variant mechanism of Z39.50.

The following is an example of a title element which occurs in two different languages – Table 4.

Table 4. Different languages

```
<title>
<var lang lang "eng">Geometry of motion </>
<var lang lang "bg">Text in Bulgarian</>
</title>
```

The syntax of the *variant element* is <var class type value>. The available values for the *class* and *type* fields are given by the variant set that is associated with the current schema .Variant elements are terminated by the general end-tag </>, by the variant end-tag </var>, by the appearance of another variant tag with the same *class* and *value* settings, or by the appearance of another, normal tag. In other words, the end-tags for the variants used in the example above could have been saved.

Variant elements can be nested – Table 5.

Table 5. Variant of elements

```
<title>
<var lang lang "eng"><var body iana "text/plain">
Geometry of motion
</title>
```

Associates two variant components to the variant list for the title element. Given the nesting rules described above, we could write Table 6.

Table 6. Variant list for the title element

```
<title>
<var lang lang "eng"><var body iana "text/plain">
Geometry of motion
</title>
```

The title element above comes in two variants. Both have the IANA body type “text/plain”, but one is in English, and the other in Danish. The client, using the element selection mechanism of Z39.50, can retrieve information about the available variant forms of data elements, or it can select specific variants based on the requirements of the end-user [8,9].

## VII. Exchange Formats

Converting records from the internal structure to en exchange format is largely an automatic process. Currently, the following exchange formats are supported:

1. GRS-1. The internal representation is based on GRS-1/XML, so the conversion here is straightforward. The system will create applied variant and supported variant lists as required, if a record contains variant information.
2. XML [12,13]. The internal representation is based on GRS-1/XML so the mapping is trivial. Note that XML

schemas, preprocessing instructions and comments are not part of the internal representation and therefore will never be part of a generated XML record. Future versions of the Zebra will support that.

3. SUTRS. Again, the mapping is fairly straightforward. Indentation is used to show the hierarchical structure of the record. All "GRS" type records support both the GRS-1 and SUTRS representations.

4. ISO2709-based formats (USMARC, etc.). Only records with a two-level structure (corresponding to fields and subfields) can be directly mapped to ISO2709. For records with a different structuring (eg., GILS), the representation in a structure like USMARC involves a schema-mapping, to an "implied" USMARC schema (implied, because there is no formal schema which specifies the use of the USMARC fields outside of ISO2709). The resultant, two-level record is then mapped directly from the internal representation to ISO2709.

5. Explain. This representation is only available for records belonging to the Explain schema.

6. Summary. This ASN-1 based structure is only available for records belonging to the Summary schema - or schema which provide a mapping to this schema (see the description of the schema mapping facility above).

7. SOIF. Support for this syntax is experimental, and is currently keyed to a private Index Data OID (1.2.840.10003.5.1000.81.2). All abstract syntaxes can be mapped to the SOIF format, although nested elements are represented by concatenation of the tag names at each level.

## VIII. Software Components

*A. Zebra. (http://www.indexdata.dk/zebra/).*

Zebra is a fielded free-text indexing and retrieval engine with a Z39.50 frontend. You can use any compatible, commercial or freeware Z39.50 client to access data stored in Zebra. Zebra may be used free-of-charge in non-profit applications by non-commercial organizations. Zebra is a high-performance, general-purpose structured text indexing and retrieval engine. It reads structured records in a variety of input formats (eg. email, XML, MARC) and allows access to them through exact Boolean search expressions and relevance-ranked free-text queries. Zebra supports large databases (more than ten gigabytes of data, tens of millions of records). It supports incremental, safe database updates on live systems. You can access data stored in Zebra using a variety of Index Data tools (eg. YAZ and PHP/YAZ) as well as commercial and freeware Z39.50 clients and toolkits.

*B. Yaz (http://www.indexdata.dk/yaz/).*

The YAZ toolkit offers several different levels of access to the Z39.50 and ILL protocols. The level that you need to use depends on your requirements, and the role (server or client) that you want to implement.

*C. Zap (http://www.indexdata.dk/zap/).*

ZAP is a module which allows you to build simple WWW interfaces to Z39.50 servers. ZAP hides most of the complexity of session management, parallel searching. The integration of system into the popular Apache server offers several advantages to the operators and users of the software, including simplified maintenance of the Module, and improved performance. However, it is also possible to run the software as a CGI-script if required.

This is free software (open source) that can works on various operating systems (as Windows and Linux) and various Web Servers (as Apache and IIS).

## IX. Conclusion

The essence and functional possibilities on communication protocol Z39.50 was presented. Definite are special futures of the protocol and its application for information search in distributed databases. Definitely are software component of the protocol. Proposed the decision for works with distributed and heterogeneous databases using Z39.50. The protocol is platform and software independent.

This protocol is applied for Bulgarian partnership in the international project REGNET – REGional NETwork (Regional Networks of Culture Heritage).

## References

[1] http://www.loc.gov/z3950/agency
[2] http://www.niso.org/z3950.html
[3] http://www.ansi.org
[4] http://www.loc.gov/marc
[5] http://lcweb.loc.gov/marc/umb/um07to10.html
[6] http://www.indexdata.dk
[7] http://www.gils.net
[8] http://www.amico.org
[9] http://dublincore.org
[10] Peter Wainwright, *Professional Apache*, Wrox, November 1999
[11] David Pitts, Bill Ball, et al., *Red Hat Linux 6*, Sams, 1999.
[12] Alex Homer, *XML in IE5 Programmer's Reference*, Wrox Press, 1999.
[13] Lee Anne Phillips, *Using XML*, QUE, 2000.