

# The Alexandria Digital Library Architecture <sup>★</sup>

James Frew<sup>1</sup>, Michael Freeston<sup>2</sup>, Nathan Freitas<sup>2</sup>, Linda Hill<sup>2</sup>, Greg Janee<sup>2</sup>,  
Kevin Lovette<sup>2</sup>, Robert Nideffer<sup>3</sup>, Terence Smith<sup>4</sup>, and Qi Zheng<sup>2</sup>

<sup>1</sup> Donald Bren School of Environmental Science and Management  
University of California, Santa Barbara

`frew@bren.ucsb.edu`

<sup>2</sup> Alexandria Digital Library, University of California, Santa Barbara  
{`freeston,nathan,lhill,gjane,kal,zheng`}@alexandria.ucsb.edu

<sup>3</sup> Department of Studio Art, University of California, Irvine

`nideffer@arts.ucsb.edu`

<sup>4</sup> Department of Computer Science, University of California, Santa Barbara

`smithtr@cs.ucsb.edu`

**Abstract.** Since 1994, the Alexandria Digital Library Project has developed three prototype digital libraries for georeferenced information. This paper describes the most recent of these efforts, a three-tier client-server architecture that relies heavily on a middleware layer to present a single uniform set of interfaces to multiple heterogeneous servers. These standard interfaces, all of which are implemented in HTTP, support session management, collection discovery and evaluation, metadata searching, metadata retrieval, and online holding retrieval. An XML-based metadata encoding scheme and a simple boolean query language have also been developed. The architecture described by these interfaces has been implemented at UCSB.

## 1 ADL Background

The Alexandria Project [1] is a consortium of researchers, developers, and educators, spanning the academic, public, and private sectors, exploring a variety of problems related to distributed digital libraries for georeferenced information.

*Distributed* means the library's components may be spread across the Internet, as well as coexisting on a single desktop. *Georeferenced* means that all the library's holdings are associated with one or more regions (*footprints*) on the surface of the Earth.

The centerpiece of the Alexandria Project is the Alexandria Digital Library (ADL) [2], an online information system inspired by the Map and Imagery Laboratory [3] in the Davidson Library at the University of California, Santa Barbara (UCSB). The ADL currently provides access over the World Wide Web to a subset of the MIL's holdings, as well as other georeferenced datasets.

---

<sup>★</sup> The work described herein has been supported by the NSF-DARPA-NASA Digital Libraries Initiative, under cooperative agreement NSF IR94-11330.

There have been three distinct system architectures associated with ADL since the Project's inception in 1994. The first, or "Rapid Prototype," architecture [4] employed a desktop geographic information system (GIS) as the user interface to a single ADL catalog database. The second, or "Web Prototype," architecture [5] replaced the GIS with an HTTP server, which presented a user interface of dynamically-generated HTML pages. This paper describes the third ADL architecture, as implemented in July 1998.

## 2 ADL Architecture Overview

The ADL architecture (Fig. 1) is a 3-tier client-server architecture:

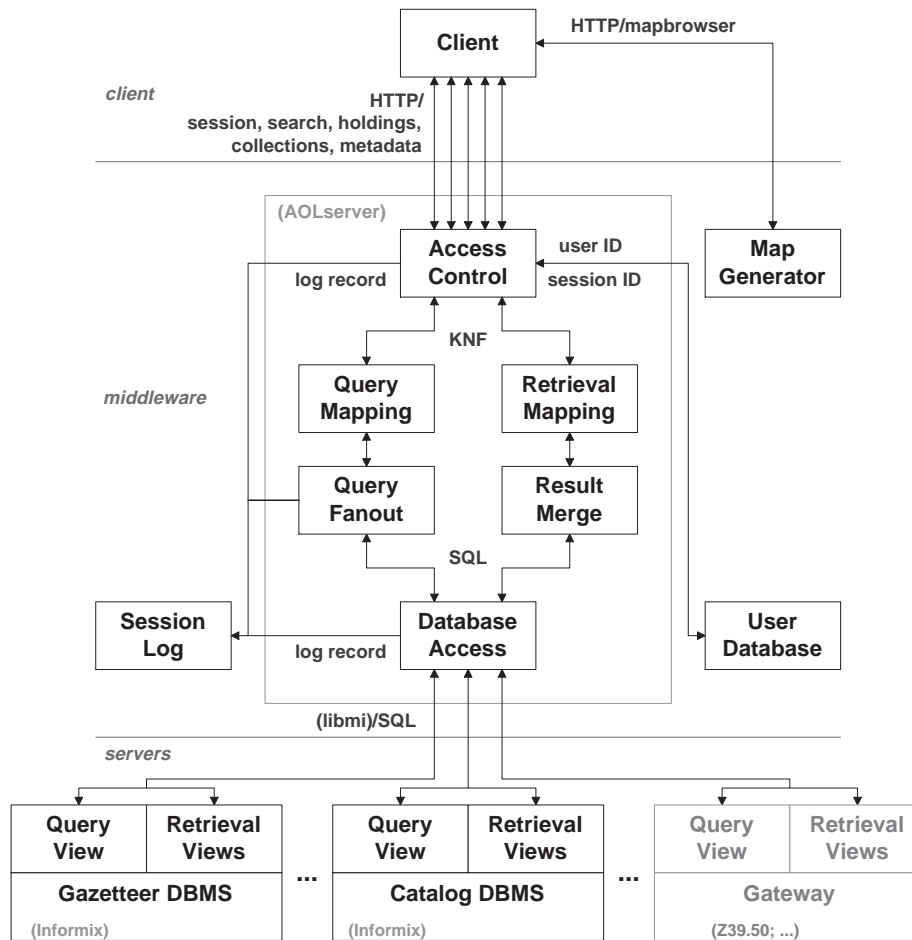


Fig. 1. ADL architecture

The crux of the architecture is a middleware layer, which maps an assortment of heterogeneous information servers (generalizations of library catalogs) into a few standard client interfaces for metadata queries, metadata retrieval, and digital holding retrieval. These interfaces are intended to be generic enough to support arbitrary clients, of which the ADL client is the first sample implementation.

### 3 Client

The ADL client is a graphical interface designed to support interactive queries by ADL users. The ADL client-middleware interfaces have been designed on the assumption that the ADL client is a *program*, as opposed to a human user. Specifically, the client is assumed to be capable of maintaining enough local state to support the notion of a “session,” as well as supporting complex real-time user interactions (e.g. rollover help). Thus, the ADL client requires less support from the rest of the ADL architecture than did its predecessor “Web Prototype” [6] client, which was driven by server-generated HTML pages.

The current version of the ADL client is implemented entirely in Java. It is distributed as a self-installing stand-alone application, for any platform (Windows, Macintosh, Sun, SGI, etc.) supporting release 1.1 or higher of the Java Runtime Environment [7].

Further discussion of the ADL client is beyond the scope of this paper. The client is described in detail at, and may be downloaded from, <http://www.alexandria.ucsb.edu/adljigi>.

### 4 Client-Middleware Interface

The ADL client communicates with an ADL middleware layer via HTTP [8]. We selected HTTP as the basic protocol owing to its ubiquity, simplicity, and the ease with which current HTTP servers can be extended to support the level of functionality we require.

Each family of interfaces supported by the ADL middleware is bound to a particular URL. This eliminates the need for top-level switch logic to vector requests to the appropriate handlers, but does require that the client be aware of which URL implements which interface.

Five standard interfaces are currently supported:

1. **session**: create or terminate a logical “session” between the client and the middleware;
2. **collections**: list the library collections supported by the middleware;
3. **search**: identify library holdings that satisfy specific boolean constraints on standard high-level search-oriented metadata.
4. **metadata**: retrieve metadata for specified library holdings; and
5. **holdings**: request either a (possibly reduced-resolution) graphic representation of a specified library holding, or the holding itself.

A sixth interface, `mapbrowser`, allows a client to request a base map for a particular portion of the Earth's surface, specifying projection and feature layers (coastlines, highways, etc.). Although this service is unrelated to the contents of the library, it simplifies the development of graphical clients that display maps as navigational tools.

All ADL interfaces are implemented as HTTP GETs or POSTs to the interface's URL. By convention, the pathname portion of the URL ends with *interface/method*. Method parameters are passed according to the CGI [9] encoding conventions; they are referenced below using the format `name=value`. Except as noted, return values are provided as ASCII text (MIME type `text/plain`). An error message may be returned in lieu of whatever other return values the method supports.

#### 4.1 Session Interface

The `session` interface establishes a logical connection between the client and the middleware that persists between HTTP transactions. Since HTTP is a stateless protocol, the client and the middleware must cooperate to preserve enough state to allow for incremental queries, user authentication, etc. In the current ADL architecture, almost all of this state is maintained by the client; the `session` interface merely allows for the assignment and de-assignment of unique *session identifiers*.

The `session` interface thus exists primarily to support tracking of system activity, especially user interactions, for evaluation purposes [10]. All ADL interfaces accept a session identifier as a parameter, so in theory any ADL system event can be traced back to the specific user responsible for it. However, the architecture is explicitly designed to not *require* this level of monitoring. In almost all cases, the behavior of an ADL interface does not change if the supplied session identifier is null or otherwise "invalid."

The session interface exposes the following methods:

1. `start` initiates an ADL session. It accepts user data and client data, and returns a unique session identifier, which may be used in any subsequent calls to ADL interfaces:

- `user_name=name`
- `user_pass=password`
- `client_data=data`

The client data is an optional, opaque string that the middleware may use to tailor its behavior to specific clients. The user name and password identify the user to the rest of the ADL system.

2. `end` terminates an ADL session:

- `session_id=session_id`
- `client_data=client_data`

Although `end` is not strictly necessary, calling it allows the middleware to recycle any resources that may have been dedicated to supporting the specified session. Moreover, `end` is a POST method, and its optional contents are

assumed to be client-specific session history information. These are passed by the middleware to its session logging database, for later evaluation by the ADL developers.

## 4.2 Collections Interface

The `collections` interface allows the client to quickly determine which library *collections* (logical groupings of library holdings) are accessible from the middleware. This interface exposes two methods:

1. `list` returns a list of simple text identifiers for all the collections that the middleware has access to. It has no arguments.
2. `scan` returns detailed metadata for a specific collection:

– `collection_id=collection_id`

The collection metadata are returned as XML-tagged text. Since this encoding strategy is also used by other ADL interfaces, we describe it briefly in the next section.

**Metadata Format.** All but the simplest metadata are encoded by the middleware with XML tags before being passed to the client. This encoding is based on two concepts: *sections* and *name-value pairs*.

Metadata are grouped into top-level sections by XML elements whose semantics are defined as part of the interface. For example, collection metadata returned by the `scan` method of the `collections` interface will always include one `BUCKET` element for each searchable parameter that the collection supports. (*Bucket* is the ADL term for a high-level searchable attribute, generally comprising the union of several more precise lower-level attributes. The current set of ADL buckets is described under “Search Buckets” below.)

Explicit name-value pairs are used to encapsulate metadata whose semantics may otherwise be opaque to ADL. Such metadata are packaged in adjacent `NAME` and `VALUE` XML elements:

```
<collection>
  <name>collection.name      </name><value>ADL Catalog</value>
  <name>collection.id       </name><value>adl_catalog</value>
  <name>collection.description</name><value>
    The Alexandria Digital Library (ADL) Catalog provides metadata for
    geospatial data (e.g., maps, aerial photographs, etc., whether in
    digital or in hard-copy form). Coverage is worldwide but primarily
    concentrated in the southern California area.
  </value>
  <bucket>
    <name>bucket.name       </name><value>Geographic Locations</value>
    <name>bucket.id        </name><value>location      </value>
    <name>bucket.type      </name><value>map          </value>
```

```

    <name>bucket.scheme    </name><value>none                </value>
    <name>bucket.description</name><value>
        Latitude and longitude of item's point location, or bounding
        latitudes and longitudes of item's geographical footprint
    </value>
</bucket>
<bucket>
    <name>bucket.name    </name><value>Format                </value>
    <name>bucket.id      </name><value>format                </value>
    <name>bucket.type    </name><value>tree                 </value>
    <name>bucket.scheme</name><value>ADL Format List</value>
    <domain>
        <name>HTML </name><value>10</value>
        <name>Paper</name><value>50</value>
    </domain>
    <name>bucket.description</name><value>
        Format(s) in which item is available.
    </value>
</bucket>
...
</collection>

```

### 4.3 Search Interface

The `search` interface allows the client to query the searchable metadata in one or more of the collections accessible to the middleware. This interface exposes two methods:

1. `start` initiates a query:
  - `session_id=session_id`
  - `results=max_results`
  - `collections=collection,...`
  - `query=query`

The query is applied to each specified *collection*, but not in any particular order. Once *max\_results* results have been accumulated, `start` will return, regardless of how the results are distributed across the specified collections. `start` immediately returns a *query identifier*, which is required if the query is to be terminated prematurely by `stop`. *Object identifiers* for individual library holdings are then streamed back to the client as the query progresses. This “two-part” return is a deliberate design choice, intended to facilitate the construction of multithreaded clients – they are free to collect the query results in parallel with other activities, once the query identifier has been retrieved.

The form and content of a *query* is described in the following section.

2. `stop` terminates a query before it returns:
  - `session_id=session_id`

– `query_id=query_id`

This provides the client with an “escape hatch” for hung or long-running queries. It is not strictly necessary to the architecture, although, like the `end` method in the `session` interface, it can be of considerable assistance to the middleware and underlying servers, by enabling them to free resources no longer required to support a particular client.

**Search Buckets.** The `query` passed to the `start` method specifies combinations of, and/or boolean constraints on, the ADL *search buckets*, a standard set of high-level searchable metadata. Each collection supported by the middleware must specify a mapping from its own metadata into the search bucket attributes. This mapping will almost always be many-to-one; i.e., there will inevitably be a loss of precision in querying the search buckets versus querying the collection-specific metadata directly. However, by exposing only a single high-level set of searchable metadata, the ADL middleware allows clients to be built that can both exploit search bucket semantics (e.g. spatially manipulate the “location” bucket), and search all of ADL via a single connection.

The current set of ADL search buckets is:

1. *geographic location*: latitude-longitude bounding box of the holding’s footprint.
2. *type*: the “logical type” of the library holding, usually a categorization of its form (e.g., map, aerial photograph, etc.) or its content (e.g., hydrographic feature, airport, etc.)
3. *format*: the format(s) in which the library holding can be delivered, both online and offline.
4. *about (freetext)*: words from the holding’s title, abstract, subject heading, index terms, keywords, etc. that indicate the topics and themes of the holding
5. *about (assigned)*: words from subject headings, index terms, and keywords that were assigned by catalogers to indicate the topics and themes of the holding
6. *originator*: words from author, investigator, publisher, and similar attributes.
7. *date range*: beginning and ending dates indicating coverage, publication, or other event relevant to the content of the holding.
8. *identifier*: any standard identifier or number (e.g., ISSN, ISBN, report number, URL) associated with the holding.

Two important design characteristics distinguish the ADL search buckets from otherwise similar “umbrella” metadata schemes such as GILS [11] or the “Dublin Core” [12]. First, the ADL search buckets have specific semantics, both in terms of constraints on their content, and in terms of how they may be searched. This allows for much more powerful searches (e.g., spatial searches against the “location” bucket) than if the buckets were only specified to contain arbitrary text. Second, the particular set of attributes comprising the ADL search buckets is intended to be optimal for digital georeferenced information (e.g., fields indicating the Earth location or digital format of the holding are far

more important for *discovering* information than is the ability to easily distinguish an author from a publisher.)

**KNF Query Language.** The *query* passed to the `start` method is written in a simple language we call KNF (“Kevin’s Normal Form”). The syntax of KNF is quite similar to the LISP programming language, and supports both method invocation and simple boolean expressions. Queries are assembled from boolean combinations of predicates, each of which is applied to a particular search bucket. For example,

```
(and
  (contains
    (rectangle
      (coord -133.1262 31.9119)
      (coord -110.6262 40.0369)))
  (or
    (overlaps
      (rectangle
        (coord -122.7626 38.3006)
        (coord -121.4461 37.3357)))
    (overlaps
      (polygon
        (coord -113.12 31.91)
        (coord -110.62 31.91)
        (coord -110.62 40.03)
        (coord -113.12 40.03)
        (coord -113.12 31.91))))))
```

specifies that the `location` bucket must contain the first rectangle, as well as at least one of either the second rectangle or the polygon.

It may be helpful to think of KNF as an alternative to the `WHERE` clause in an SQL `SELECT` statement. To continue the analogy, the `start` method’s `collections` parameter is analogous to a `SELECT` statement’s `FROM` clause. The `SELECT` is implicitly the object identifiers that `start` returns.

#### 4.4 Metadata Interface

The `metadata` interface provides access to partial and full metadata for specified holdings. A typical sequence is for the `metadata` interfaces to be invoked to evaluate the results of a `search`.

The `metadata` interface exposes two methods, both of which are passed an object identifier

- `object_id=object_id`

as their single parameter:

1. `scan` returns a small subset of the specified object's metadata, e.g.:

```
<doc>
  <section>scan
    <group>location
      <name>west_bounding_longitude</name><value>-115.625000</value>
      <name>east_bounding_longitude</name><value>-115.500000</value>
      <name>north_bounding_latitude</name><value>33.125000</value>
      <name>south_bounding_latitude</name><value>33.000000</value>
    </group>
    <group>title
      <name>title</name><value>
        Westmorland East; Digital Raster Graphic (DRG) Data
      </value>
    </group>
    <group>date
      <name>begin_date</name><value>1996-05-08</value>
      <name>end_date</name><value>1996-05-08</value>
    </group>
    <group>type
      <name>type</name><value>MAPS</value>
      <name>available_as</name><value>^TIFF^</value>
    </group>
    <group>collection
      <name>collection_id</name><value>ius_catalog</value>
      <name>object_id</name><value>adl_catalog:800089</value>
      <name>parent_id</name><value>adl_catalog:909</value>
    </group>
  </section>
</doc>
```

The `scan` subset includes:

- (a) those search buckets which are most likely to have a single, unambiguous value (`location`, `date`, `type`);
- (b) the holding's `title`, which, if present, is useful for quick evaluation (but which is often absent, and therefore a poor high-level *search* term); and
- (c) identifiers for the holding's collection and any parent-level metadata, as well as for the holding itself, which can be used in subsequent data or metadata retrievals.

These attributes were selected to maximize the utility of the `scan` metadata for quick evaluation, while minimizing the amount of per-holding information that a client must manipulate. For example, the current ADL client automatically retrieves all `scan` metadata for all holdings returned by the `search` interface.

2. `full` returns all available metadata for the specified holding.

As mentioned above (in “Metadata Format”), there is no guarantee that a client will be able to interpret any particular attribute in a given holding’s metadata. However, the XML packaging always allows a client to unambiguously *parse* the metadata, and (for example) display it in the hierarchy implied by the nesting of the `group` tags.

#### 4.5 Holdings Interface

The `holdings` interface allows the client to retrieve library holdings using their object identifiers. These identifiers are usually obtained from the `search` interface. However, since ADL object identifiers are persistent, they can be saved (or, for example, emailed to a colleague) and then passed directly to the `holdings` interface during a subsequent session.

The `holdings` interface exposes two methods, both of which are passed an object identifier

– `object_id=object_id`

as their single parameter:

1. `thumbnail` returns a reduced-resolution image rendition of the corresponding holding, if one is available. The image is returned as a MIME-typed byte stream.

*Thumbnail* images are small (typically 100 by 100 8-bit pixels, compressed with GIF or JPEG) in order to minimize download time. They are intended to help a library user make quick “go/no-go” decisions about whether to pursue a (possibly much) larger, full-resolution version of the holding.

2. `access` returns a URL, from which higher-resolution versions of the holding may be retrieved.

In earlier versions of ADL, the equivalent of `access` returned the holding itself, as a MIME-typed byte stream. The extra level of indirection was added to the current version for three reasons:

- (a) One metadata field (the URL returned by `access`) can now be used to group together a suite of high-resolution renditions of a single holding. For example, most of ADL digital holdings are available at both “full” and “browse” resolutions, where “browse” is informally defined as “big enough to make an informed decision as to whether the full resolution version is worth retrieving.” Our browse images are typically about 500 by 500 pixels.
- (b) Large holdings can be moved to wherever space is available, without having to rewrite the corresponding catalog database. The `access` interface need only maintain a relatively simple mapping between object identifiers and storage locations.
- (c) All of the *distribution* restriction issues currently facing ADL are related to full-resolution holdings, not lower-resolution versions or metadata. Enforcing these restrictions in the `access` interface means we don’t have to worry about them in other interfaces.

## 5 Middleware

The ADL middleware is responsible for implementing the client-middleware interfaces described above, and mapping them into interactions with an arbitrary number of metadata catalogs. Additionally, the middleware implements whatever access controls ADL requires.

The current ADL middleware layer is implemented by an AOLserver [13] HTTP server. All ADL-specific functionality described in this section is implemented by C functions and Tcl scripts executing in the context of the AOLserver.

### 5.1 Access Control

The ADL middleware supports whatever access policy is dictated by ADL. Two specific access controls are supported in the current implementation: host-based and user-based.

Host-based access control is used to refuse connections from clients that are not running at an ADL-approved Internet address. This is currently used to limit access to ADL to only those hosts connected to an IP network managed by the University of California. To this end, the access-control module maintains an explicit list of network numbers from which it will entertain connections. This mechanism is inherently unscalable, although it suffices to meet the UC-only distribution restrictions imposed by third parties on some proprietary materials in the ADL collections (e.g., commercial remote sensing imagery).

The `session` interface can be used to implement a crude user-based access control mechanism, simply by configuring the other interfaces so that they refuse to accept invalid session identifiers. So far, we have not seen any advantage to the easily-defeated increment of security that this would provide.

### 5.2 Query and Retrieval Mapping

The ADL middleware maps KNF queries and `holdings` requests into queries specific to the underlying ADL servers. If multiple servers, or multiple databases on a single server, must be queried, this layer handles the requisite fan-out/fan-in. Note that we currently assume that multiple databases are disjoint; thus, the fan-in process is currently simple collation, with no duplicate detection or other conflict resolution.

In the current implementation, a basic architectural assumption is that the underlying servers will expose views that are, as closely as the particular server allows, exact correlates to the ADL search buckets and metadata sections. Thus, the translation functionality of the mapping layer is currently limited to “translitterating” KNF into various dialects of server query languages (e.g., SQL).

### 5.3 Database Connections

The ADL middleware maintains a pool of client connections to whatever underlying servers are currently supported. The database connection layer is responsible for presenting a single functional interface to these shared connections. This

serves to both localize whatever special knowledge (e.g. database client library) is needed to communicate with a particular server, and to minimize the setup or teardown time associated with making or breaking database client-server connection.

## 6 Servers

The ADL system currently includes two collection servers, the ADL catalog and the ADL gazetteer. Accommodating their quite different schemas has been a good test of our search bucket mapping strategy (i.e., congruent views in both the catalog and the gazetteer).

Both the catalog and the gazetteer are implemented as Informix Dynamic Server–Universal Data Option [14] databases. The only Informix-specific features that we currently exploit are the MapInfo [15] spatial data types, and their associated operators and indices, to implement the “location” search bucket.

## 7 Status

As of July 1998 the ADL architecture described herein is up and running on the ADL testbed system at UCSB (subject to the aforementioned access restrictions), and is also being installed at the San Diego Supercomputer Center. Access to the UCSB system is currently restricted to Internet domains controlled by the University of California.

## 8 Acknowledgements

We thank the staff of UCSB’s Davidson Library Map and Imagery Laboratory for their support of the ADL testbed system.

## References

1. Alexandria Project. <http://www.alexandria.ucsb.edu>
2. Smith, T., Andresen, D., Carver, L., Dolin, R., Fischer, C., Frew, J., Goodchild, M., Ibarra, O., Kemp, R., Kothuri, R., Larsgaard, M., Manjunath, B., Nebert, D., Simpson, J., Wells, A., Yang, T., Zheng, Q.: A digital library for geographically referenced materials. *Computer* 29:5 (1996) 54-60
3. UCSB Davidson Library Map and Imagery Laboratory. <http://www.sdc.ucsb.edu>
4. Frew, J., Carver, L., Fischer, C., Goodchild, M., Larsgaard, M., Smith, T., Zheng, Q.: The Alexandria Rapid Prototype: building a digital library for spatial information. 1995 ESRI User Conference Proceedings, May 22-25, 1995, Environmental Systems Research Institute, Inc., Redlands, CA (1995) <http://www.esri.com/base/common/userconf/proc95/to300/p255.html>
5. Frew, J., Freeston, M., Kemp, R., Simpson, J., Smith, T., Wells, A., Zheng, Q.: Alexandria Digital Library Testbed. *D-Lib Magazine* (July/August 1996) <http://www.dlib.org/dlib/july96/alexandria/07frew.html>

6. Andresen, D., Carver, L., Dolin, R., Fischer, C., Frew, J., Goodchild, M., Ibarra, O., Kothuri, R., Larsgaard, M., Manjunath, B., Nebert, D., Simpson, J., Smith, T., Yang, T., Zheng, Q.: The WWW prototype of the Alexandria Digital Library. Proceedings of ISDL'95: International Symposium on Digital Libraries, 22-25 August 1995, Japan (1995)
7. Java Runtime Environment. <http://www.javasoft.com/products/jdk/1.1/jre>
8. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (January 1997) <ftp://nis.nsf.net/documents/rfc/rfc2068.txt>
9. NCSA HTTPd Development Team: The Common Gateway Interface. (March 1996) <http://hoohoo.ncsa.uiuc.edu/cgi/>
10. Hill, L., Dolin, R., Frew, J., Kemp, R., Larsgaard, M., Montello, D., Rae, M., Simpson, J.: User evaluation: summary of the methodologies and results for the Alexandria Digital Library, University of California at Santa Barbara. Proceedings of the 60th Annual Meeting of the American Society for Information Science, November 1-6, 1997, Washington, DC (1997) <http://www.asis.org/annual-97/alexia.htm>
11. Christian, E.: GILS: what is it? where's it going? D-Lib Magazine, (December 1996) <http://www.dlib.org/dlib/december96/12christian.html>
12. Weibel S., Miller, E.: Dublin Core metadata. (1997) [http://purl.oclc.org/metadata/dublin\\_core](http://purl.oclc.org/metadata/dublin_core)
13. AOLserver. <http://www.aolserver.com>
14. Informix Dynamic Server – Universal Data Option. <http://www.informix.com>
15. MapInfo SpatialWare. <http://www.mapinfo.com/spatialware/>