Int J Digit Libr (2000) 2: 259–268

INTERNATIONAL JOURNAL ON
Digital Libraries

The Alexandria Digital Library architecture*

J. Frew[1], M. Freeston[2], N. Freitas[3], L. Hill[3], G. Janée[3], K. Lovette[3], R. Nideffer[4], T. Smith[5], Q. Zheng[3]

[1] Donald Bren School of Environmental Science and Management, University of California, Santa Barbara, CA, USA;
E-mail: frew@bren.ucsb.edu
[2] Department of Computing Science, University of Aberdeen, Scotland, UK
[3] Alexandria Digital Library, University of California, Santa Barbara, CA, USA
[4] Department of Studio Art, University of California, Irvine, CA, USA
[5] Department of Computer Science, University of California, Santa Barbara, CA, USA

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.

Key words: Digital library – Geospatial data – Interface specification – Metadata – Three-tier architecture

1 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 work described herein was supported by the NSF-DARPA-NASA Digital Libraries Initiative, under cooperative agreement NSF IR94-11330.

The centerpiece of the Alexandria Project is the Alexandria Digital Library (ADL) [20], an online information system inspired by the Map and Imagery Laboratory [24] 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.

2 Architecture overview

There have been three distinct system architectures associated with ADL since the Project's inception in 1994. The first, or "Rapid Prototype", architecture [7] used a desktop geographic information system (GIS) as the user interface to a single ADL catalog database. The second, or "Web Prototype", architecture [8] replaced the GIS with an HTTP server, which presented a user interface of dynamically-generated HTML pages. This paper describes the third ADL architecture, a generalization of the Web Prototype with interfaces supporting multiple clients and servers.

The ADL architecture (Fig. 1) follows the three-tier model [18]. *Servers* manage library collections, *middleware* implements standard services on these collections, and *clients* present these services to library users.

ADL *servers* are responsible for maintaining collections of metadata describing the library's holdings, and for implementing query and retrieval mechanisms against that metadata. (An ADL *holding* is an object cataloged by ADL; the term does not carry the same connotation of ownership as in a traditional library.) A holding's metadata may include references to online representations of the holding (e.g., URLs), but this is not required; i.e., ADL can also be used as an online catalog for offline or

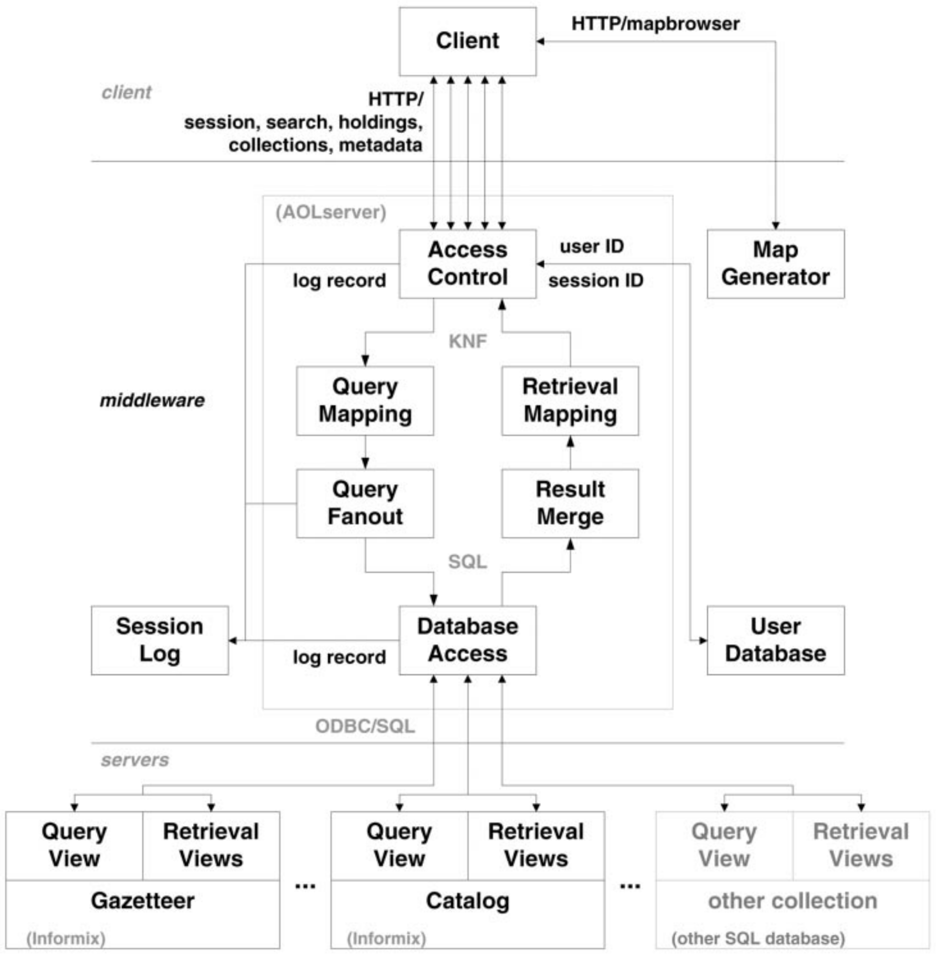


Fig. 1. ADL architecture

non-digital geospatial information. As such, ADL servers are generalizations of traditional library catalogs.

As long as they expose the proper interfaces to the middleware, ADL servers are otherwise autonomous; i.e., a site can implement an ADL server to "publish" specific collections that are accessed only through another site's middleware.

ADL *clients* are responsible for presenting ADL services to library users. These users may be interactive (e.g., humans using a GUI), or they may be other programs using ADL as a data source. ADL clients are assumed to be capable of maintaining enough local state to support whatever notion of a *session* (global context) that they require. ADL clients may also support complex real-time user interactions (e.g., rollover help).

The crux of the ADL architecture is the *middleware* layer, which maps an assortment of heterogeneous collection servers into a few standard client interfaces for metadata queries, metadata retrieval, and digital holding retrieval. These client interfaces are intended to be generic enough to support arbitrary clients.

The middleware's client interfaces are *stateless* – each invocation of a middleware method by a client is a transaction unto itself, bearing no relation to previous transactions except as conveyed by its parameters. This both simplifies the implementation and execution overhead of the middleware, and allows for clients of arbitrary complexity. However, statelessness also makes *iterative* (successively refined) queries more difficult to support.

The middleware's client interfaces are the primary public artifact of the ADL architecture, since they define and circumscribe the library's capabilities. In effect, they are a snapshot of the ADL Project's research into the services a digital geospatial library should provide. In the next section, we explore these interfaces in detail.

3 Client interfaces

The ADL architecture defines five standard client interfaces:

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.
5. `holdings`: request either a (possibly reduced-resolution) graphic representation of a specified library holding, or information on how to retrieve 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 a 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.

3.1 Session interface

The `session` interface establishes a logical connection between the client and the middleware that persists between method invocations. Since the ADL client interfaces are designed to be implementable with stateless protocols (e.g., HTTP), 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:

```
sessionID = start(name, password, client)
end(sessionID, client)
```

`start` initiates an ADL session. `name` and `password` identify the user to the rest of the ADL system. `client` is optional information that the middleware may use to tailor its behavior for specific clients. `sessionID` is a unique session identifier, which may be used in any subsequent calls to ADL interfaces.

`end` terminates the ADL session referenced by `sessionID`. 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. `client`, if present, is assumed to be client-specific session history information. This information is passed by the middleware to its session logging database, for later evaluation by the ADL developers.

3.2 Collections interface

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

```
collectionIDSet = list(sessionID)
collectionMetadata = scan(sessionID,
                          collectionID)
```

`list` returns a set identifiers for all the collections that the middleware has access to.

`scan` returns detailed metadata for the collection referenced by `collectionID`. These metadata are specified by an ADL-developed collection metadata content standard [12]. The elements supported by this standard include general information (scope and purpose, update frequency, use restrictions, etc.), domain specifications (numbers and types of holdings, spatial and temporal coverages, etc.), and search capabilities (i.e., the ADL search buckets the collection supports; see Sect. 3.3.1)

Scan returns the collection metadata as an XML [4] document. Since this encoding strategy is also used by other ADL interfaces, we describe it briefly in the next section.

3.2.1 Metadata formats

ADL uses two XML-based strategies to encode and transmit metadata:

Semantic XML uses tags that directly denote the semantic content of the metadata. This strategy is used when the metadata content is understood by ADL, as in the case of the collection metadata described above. A semantic XML document type definition (DTD) defines the structure and attributes of the metadata. For example, the DTD for the ADL collection metadata begins with

```
<!ELEMENT ADL-collection-metadata
  (internal-name, lowercase-name,
   uppercase-name, full-title, short-title,
   responsible-party?, scope-and-purpose?,
   subject-coverage?, content-type?,
   relationships?, creation-date?,
   last-update-date?, update-frequency?,
   metadata-schema?, sample-metadata-record?,
   terms-and-conditions?, contact?,
   ((item-type-hierarchy, item-type-thesaurus?)
     |~item-type-description)?,
   ((item-format-hierarchy,
     item-format-thesaurus?)
     |~item-format-description)?,
   ADL-search-buckets?, alert?, item-counts?,
   spatial-coverage?, temporal-coverage?)>
```

The complete DTD for the ADL collection metadata is available at `http://alexandria.ucsb.edu/docs/metadata/ADL-collection-metadata.dt`.

Syntactic XML uses tags that describe only a hierarchy of named *sections* and, within sections, *name-value pairs* that associate values with arbitrary attributes. This strategy is used when the semantic content of the metadata is opaque to ADL, as in the case of the `full` method of the `metadata` interface, which returns holding metadata of unknown type or origin. An example of syntactic XML can be found in Sect. 3.4.

3.3 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:

```
queryID, holdingIDSet = start(sessionID,
       maxResults, collectionIDSet, query)
stop(sessionID, queryID)
```

`start` applies `query` to each collection referenced by `collectionIDSet` (in no specific order). The form and content of a `query` are described in Sects. 3.3.1 and 3.3.2.

`start` immediately returns a query identifier, and then streams back a set of holding identifiers (one per individual library holding), until either the query terminates, or `maxResults` holding identifiers have been returned. 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.

`stop` terminates the query referenced by `queryID` regardless of its progress. 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.

3.3.1 Search buckets

The `query` passed to the `start` method specifies combinations of, and/or Boolean constraints on, the ADL *search buckets* [9], 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 locations*: the set of points, bounding boxes, or polygons, in latitude-longitude coordinates, that specify the holding's footprint.
2. *dates*: any dates or date ranges associated with the content or preparation of the holding.
3. *types*: the "logical types" of the holding. Types are drawn from collection-specific domains which characterize the form (e.g., map, aerial photograph, etc.) or content (e.g., hydrographic feature, airport, etc.) of the collection's holdings.
4. *formats*: the formats or representations (online or offline) in which the holding can be delivered. Formats are also drawn from collection-specific domains.
5. *assigned terms*: terms from the topics and themes (e.g., subject heading, index terms, keywords, etc.) of the holding. Assigned terms must be drawn from specified controlled vocabularies.
6. *topical text*: terms from the topics and themes of the holding (e.g., title, abstract, assigned terms, etc.). Topical text need not be drawn from controlled vocabularies.
7. *originators*: words from author, investigator, publisher, and similar attributes.
8. *identifiers*: any standard identifiers or numbers (e.g., ISSN, ISBN, report number, URL) associated with the holding.

ADL does not require that collections populate every search bucket, however, the buckets were chosen with the expectation that they will be well-populated by most collections.

Two important design characteristics distinguish the ADL search buckets from otherwise similar "umbrella" metadata schemes such as the GILS core attributes [17] or the Dublin Core [25]. 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:

– *spatial* search looks for locations that either *contain* or *overlap* a query location.
– *temporal* search looks for dates that occur either *on* or *before* or *on* or *after* a query date.
– *text* search looks for text that contains *all* or *any* of the query words, or the *exact* query phrase.
– *hierarchical* search looks for terms that are *equal* to or are a *subcategory* of a query term.

This allows for much more powerful searches than if the buckets were only specified to contain arbitrary text, or (as in Dublin Core elements) arbitrary combinations of types (text, coordinates, dates, etc.).

Second, the attributes comprising the ADL search buckets were selected specifically to describe and query georeferenced spatial information, as opposed to generic digital information. This bias is apparent when comparing the ADL search buckets to the Dublin Core elements, for example:

- ADL has no “title” search bucket. We have found that titles are not a very useful discriminant for geospatial information. In particular, many scientific datasets have titles that are simply concatenations or abbreviations of attributes that are already represented in other search buckets.
- ADL provides separate search buckets for “geographic locations” (space coverage) and “dates” (time coverage), while Dublin Core blurs this distinction with a single “coverage” element. This reflects the importance of geographic queries in ADL.
- Dublin Core provides three elements (“Creator”, “Publisher”, “Contributor”) to distinguish concepts which ADL merges into the single search bucket “originators”. In this case, we have found the finer resolution of the Dublin Core elements to be irrelevant for queries against most ADL content (maps and images).

3.3.2 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 at least one of either the second rectangle or the polygon. KNF queries are strictly Boolean – there is no precedence or weighting of predicates other than that enforced by conjunctions and grouping.

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 holding identifiers that `start` returns.

3.4 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:

```
scanMetadata = scan(holdingID)
fullMetadata = full(holdingID)
```

`scan` returns a small subset of the specified holding’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>format</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:

1. Those search buckets which are most likely to have a single, unambiguous value (`location`, `date`, `type`).
2. The holding's `title`.
3. 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 holding identifiers returned by the `search` interface.

`full` returns all available metadata for the specified holding. As mentioned above (Sect. 3.2.1), 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.

3.5 Holdings interface

The `holdings` interface allows the client to retrieve library holdings using their holding identifiers. These identifiers are usually obtained from the search interface. However, since ADL holding 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 three methods:

```
image = thumbnail(holdingID)
image = browse(holdingID)
URL = access(holdingID)
```

`thumbnail` returns a reduced-resolution image rendition of the corresponding holding, 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.

`browse` also returns a reduced-resolution (but more detailed than thumbnail) image rendition of the corresponding holding. *Browse* resolution is informally defined as "big enough to make an informed decision as to whether the full-resolution holding is worth retrieving". Our browse images are typically about 500 by 500 pixels.

`access` returns the URL of an HTML document (the *access report*), from which full-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:

1. One metadata field (the URL returned by `access`) can now be used to group together multiple copies, multiple formats, and multiple packagings of, and other accessibility information related to, a single holding.
2. Large holdings can be moved to wherever space is available, without having to rewrite the corresponding catalog database. The `access` method need only maintain a relatively simple mapping between holding identifiers and storage locations.
3. 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` method means we don't have to worry about them elsewhere.

4 Implementation

4.1 Client

The current implementation of the ADL architecture includes a sample client "JIGI" (Java Interface to Geographic Information). As its name implies, JIGI 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 [21]. As a standalone application, JIGI has more functionality than a browser-hosted applet, and also avoids the incompatibilities between different browsers' implementations of the Java virtual machine. JIGI also requires far less support from the rest of the ADL architecture than did its predecessor "Web Prototype" [3] client, which was driven by server-generated HTML pages.

JIGI implements a graphical user interface (GUI) to ADL, using four cooperating windows. A *map browser* (Fig. 2) allows the user to draw a rectangular region on the Earth's surface as a query against the geographic location bucket. The map browser can also display the

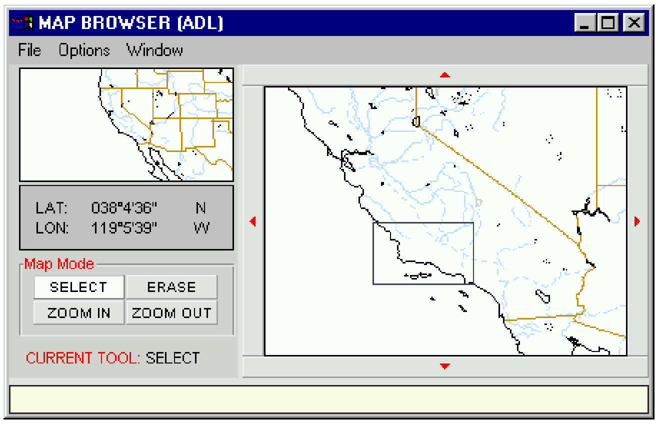
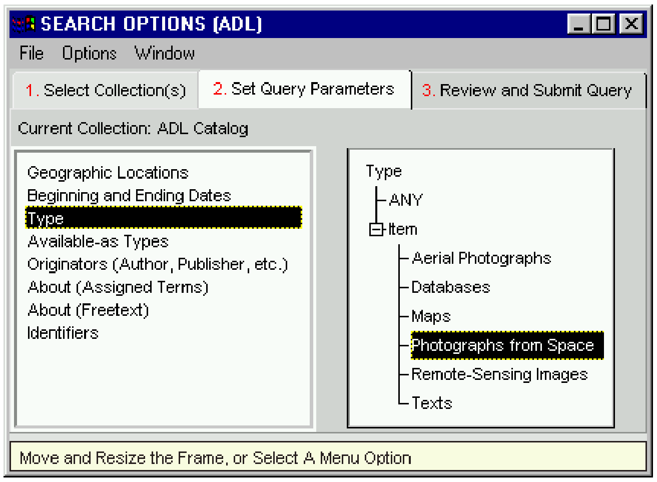
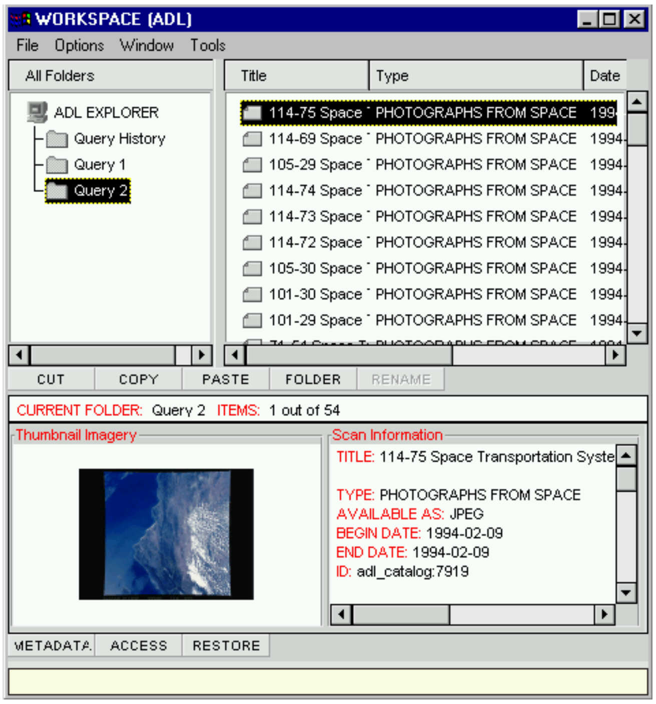


Fig. 2. JIGI map browser



Current Collection: ADL Catalog

Fig. 3. JIGI search options window



Title	Type	Date
114-75 Space	PHOTOGRAPHS FROM SPACE	199
114-69 Space	PHOTOGRAPHS FROM SPACE	1994
105-29 Space	PHOTOGRAPHS FROM SPACE	1994
114-74 Space	PHOTOGRAPHS FROM SPACE	1994
114-73 Space	PHOTOGRAPHS FROM SPACE	1994
114-72 Space	PHOTOGRAPHS FROM SPACE	1994
105-30 Space	PHOTOGRAPHS FROM SPACE	1994
101-30 Space	PHOTOGRAPHS FROM SPACE	1994
101-29 Space	PHOTOGRAPHS FROM SPACE	1994

CURRENT FOLDER: Query 2 ITEMS: 1 out of 54

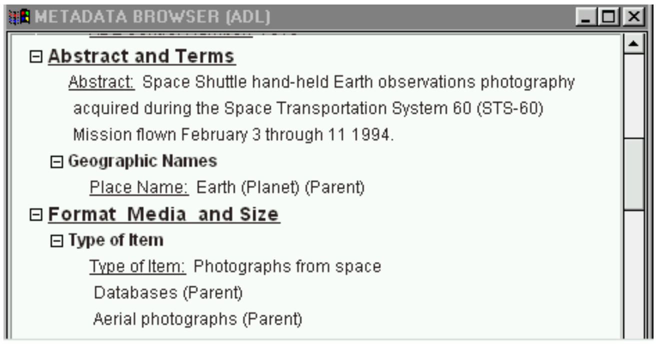
TITLE: 114-75 Space Transportation Syste
TYPE: PHOTOGRAPHS FROM SPACE
AVAILABLE AS: JPEG
BEGIN DATE: 1994-02-09
END DATE: 1994-02-09
ID: adl_catalog:7919

Fig. 4. JIGI workspace

footprints (geographic extents) of holdings returned by a query.

A *search options* window (Fig. 3) allows the user to select one or more collection servers to query, and to specify constraints on any search bucket that the collections support. The search options window also controls query submission and any other query limitations (e.g., maximum size of result set).

A *workspace* (Fig. 4) accumulates query results in a tree structure that can be browsed like the directories in a file system. Selected holdings may have their thumbnail images displayed; their footprints plotted (in the map browser window), their full metadata displayed (in the metadata browser), or their access URLs invoked. The



⊟ Abstract and Terms
Abstract: Space Shuttle hand-held Earth observations photography acquired during the Space Transportation System 60 (STS-60) Mission flown February 3 through 11 1994.
⊟ Geographic Names
Place Name: Earth (Planet) (Parent)
⊟ Format Media and Size
⊟ Type of Item
Type of Item: Photographs from space
Databases (Parent)
Aerial photographs (Parent)

Fig. 5. JIGI metadata browser

workspace also allows query result sets to be saved to local files and reloaded into subsequent JIGI sessions.

A *metadata browser* (Fig. 5) displays the full metadata for selected holdings, as an expandable text hierarchy.

JIGI currently does not directly access digital holdings. Instead, if the holding's `access` interface returns a URL, JIGI passes that URL to the user's preferred web browser. For current ADL holdings, an access URL typically invokes a web page containing legal disclaimers, a browse image, and FTP URLs for the holding's contents, in various formats.

4.2 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.

ADL clients communicate with the middleware via HTTP [5]. 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 direct requests to the appropriate handlers, but does require that the client be aware of which URL implements which interface.

All ADL interfaces are implemented as HTTP GETs or POSTs to the interface's URL. (For example, `session.end` is a POST method, since its `client` argument may be arbitrarily large.) By convention, the pathname portion of the URL ends with `interface/method`. Method parameters are passed according to the CGI [15] encoding conventions. 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.

The current ADL middleware layer is implemented by an AOLserver [2] HTTP server. All ADL-specific functionality is implemented by C functions and Tcl scripts executing in the context of the AOLserver.

4.2.1 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. Although this mechanism is inherently unscalable, 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.

4.2.2 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 "transliterating" KNF into various dialects of server query languages (e.g., SQL).

4.2.3 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 connections.

4.3 Servers

The ADL architecture imposes only the most general requirement on collection servers: that they expose query and metadata interfaces that the middleware can map to the client interfaces described in Sect. 3 above. In particular, there is no requirement that the collection server implements a specific internal metadata schema, as long as appropriate views of that schema are exposed to the middleware. There is not even an architectural requirement that the server be a database system.

Nonetheless, the currently-implemented ADL collection servers are all relational databases. Comprehensive relational schemata allow the collection servers to fulfill a custodial role that, while not required by the ADL architecture, is often important to the organizations supporting the collections. For these organizations, the collection server doubles as a long-term metadata repository, which would have to be duplicated if it could not be directly connected to ADL.

Using relational databases for collection servers also means that the middleware can formulate queries in a semi-standard language (SQL). However, since SQL queries are schema-specific, this does not by itself guarantee a standard middleware-collection interface. What is also required are standard views in the collection databases. These views present the appropriate collection-specific metadata attributes in terms of attributes that are, as nearly as possible, equivalent to the ADL search buckets. As part of the middleware-collection connection protocol, the collection server returns the text of the specific SQL queries that apply to its "bucket view". The middleware then loads these queries into its query mapping facility.

The general strategy, then, is for the collection server databases to implement views that correspond as closely as possible to the relevant client-middleware interfaces, and to furnish the middleware SQL query templates for those views at the time a middleware-collection connection is established. This strategy has, so far, allowed us to support several different collections with quite different internal schemata, most notably a heterogeneous catalog [6] with over one million entries, and a global gazetteer [11] with about 6 million entries.

The collections servers in the current ADL are implemented using the Informix Dynamic Server – Universal Data Option (IDS-UDO) [13] database management system. IDS-UDO extends the standard relational data model with abstract data types, which we exploit to support queries at the collection level that more closely match the capabilities of the ADL search buckets. Specifically, we use the MapInfo SpatialWare [23] spatial data types to implement spatial searches, and the Verity [14] text data types to implement text searches. Compared to standard relational database types, these extended types provide both additional search functionality, and more efficient indexing for very large collections. However, there is nothing inherent in the middleware-collection interface that would prohibit collections being implemented in any particular metadata management system.

5 Status and directions

From July 1998 to the present (December 1999) the ADL architecture described herein has been implemented on the ADL testbed system at UCSB. Access to the UCSB system is currently restricted to Internet domains controlled by the University of California, primarily to limit access to certain proprietary collections. The ADL testbed servers have been supporting well over 1 million catalog records, and about 6 million gazetteer records.

Two factors are now driving the evolution of the ADL architecture. First, work is under way to incorporate the ADL testbed system into the California Digital Library (CDL) [16]. The association with CDL poses both constraints and opportunities – constraints because CDL requires a standard web browser as a user interface, rather than a standalone client; and opportunities in the expansion of the ADL user community to a much wider audience. Second, the ADL architecture and testbed form the technical core of the Alexandria Digital Earth Prototype (ADEPT), the ADL Project's current major research thrust. ADEPT seeks to extend the ADL architecture with a much broader array of services, including visualization, data fusion, personal collections, and collaborative instructional environments.

5.1 ADL-4 architecture preview

In response to the CDL and ADEPT challenges, the "ADL-3" architecture described in this paper is being revised. The new "ADL-4" architecture, while still under development, will incorporate the following changes or additions to the current architecture:

– *ubiquitous XML*: All externally-visible data streams in the ADL-4 architecture use XML-based encodings. All XML elements have source attributes that reference the elements' semantics. XML DTDs have been implemented for queries (replacing KNF) and access reports (replacing arbitrary HTML). Using validated XML instead of idiosyncratic formats makes it easier for other systems to connect to ADL, and permits richer client functionality.
– *middleware-server interface*: The need to support an increasing number and diversity of collection servers has led to the formalization of the middleware-server interface in ADL-4. A single generic driver is used to connect to relational databases; additional drivers allow collections to be supported by indexed files or gateways to non-ADL services.
– *stateful middleware*: While statelessness is attractive from the standpoint of scalability and simplicity, the combination of stateless middleware with a stateless client (e.g., a web browser) severely limits the library's overall functionality. The ADL-4 middleware architecture thus implements session management, result set caching, and other features to allow stateless clients to perform iterative queries.
– *multiple clients*: The ADL-4 architecture supports two sample clients, the CDL web interface and a revised version of JIGI. The CDL web interface is implemented with a thin server-side layer that translates the ADL-4 middleware protocols into HTML pages.
– *servlets*: The ADL-4 middleware moves from an AOLserver-specific implementation to one that extends a generic HTTP server with *servlets* [22], Java components that run in the server's execution context.
– *remote holdings*: ADL-4 uses the San Diego Supercomputer Center's Storage Resource Broker (SRB) [19] to connect collections to (possibly remote) large scale holding storage. This provides a standard way for ADL collections to catalog, and expose access to, distributed repositories.

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://alexandria.ucsb.edu
2. America Online, Inc.: AOLserver. http://www.aolserver.com
3. 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. In: Proc. ISDL'95: International Symposium on Digital Libraries, 22–25 August 1995, Japan (1995) http://alexandria.ucsb.edu/public-documents/papers/japan-paper
4. Bray, T, Paoli, J., Sperberg-McQueen, C.: Extensible Markup Language (XML) 1.0. World Wide Web Consortium Recommendation REC-xml-19980210 (1998) http://www.w3.org/TR/REC-xml
5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. Internet Engineering Task Force RFC 2068 (January 1997) http://www.ietf.org/rfc/rfc2068.txt
6. Fischer, C., Frew, J., Larsgaard, M., Smith, T. R., Zheng, Q.: Alexandria Digital Library: rapid prototype and metadata schema. In: Adam, N., Bhargava, B., Halem, M., Yesha, Y. (eds.): Digital libraries: research and technology advances. ADL'95 Forum, McLean, VA, May 1995, Selected Papers. LNCS 1082. Berlin Heidelberg New York: Springer, pp. 221–41 (1996)
7. 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. In: 1995 ESRI User Conference Proceedings, May 22–25, 1995, Environmental Systems Research Institute, Inc., Redlands, CA (1995) http://www.esri.com/library/userconf/proc95/to300/p255.html
8. 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
9. Frew, J., Freeston, M., Hill, L., Janée, G., Larsgaard, M., Zheng, Q.: Generic Query Metadata for Geospatial Digital Libraries. In: Proc. 3rd IEEE META-DATA Conference,

April 6–7, 1999, Bethesda, MD (1999) http://computer.org/proceedings/meta/1999/papers/55/jfrew.htm

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. In: Proc. 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. Hill, L., Frew, J., Zheng, Q.: Geographic names: the implementation of a gazetteer in a georeferenced digital library. D-Lib Magazine (January 1999) http://www.dlib.org/dlib/january99/hill/01hill.html
12. Hill, L., Janée, G., Dolin, R., Frew, J., Larsgaard, M.: Collection metadata solutions for digital library applications. Journal of the American Society for Information Science 50(13): 1169–1181 (1999)
13. Informix Software, Inc.: Universal Data Option for Informix Dynamic Server (1998) http://www.informix.com/informix/techbriefs/udo/udo.htm
14. Informix Software, Inc.: Verity Text DataBlade (1999) http://www.informix.com/informix/products/options/udo/datablade/dbmodule/verity1.htm
15. NCSA HTTPd Development Team: The Common Gateway Interface. (March 1996) http://hoohoo.ncsa.uiuc.edu/cgi
16. Ober, J.: The California Digital Library. D-Lib Magazine (March 1999) http://www.dlib.org/dlib/march99/03ober.html
17. Open Systems Environment Implementors Workshop/Special Interest Group on GILS: Application profile for the Government Information Locator Service (GILS) (November 1997) http://www.gils.net/prof_v2.html
18. Sadoski, D.: Three tier software architectures. In: Software Technology Review. Software Engineering Institute, Carnegie Mellon University (January 1997) http://www.sei.cmu.edu/str/descriptions/threetier.html
19. San Diego Supercomputer Center: Storage Resource Broker; http://www.npaci.edu/DICE/SRB
20. 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): 54–60 (1996)
21. Sun Microsystems, Inc.: Java Runtime Environment; http://www.javasoft.com/products/jdk/1.1/jre
22. Sun Microsystems, Inc.: Java Servlet API; http://java.sun.com/products/servlet
23. Thomson, D.: Locational relational. Database Programming and Design (June 1997) pp. 67-69; http://www.mapinfo.com/community/library/sw_article.pdf
24. UCSB Davidson Library Map and Imagery Laboratory. http://sdc.ucsb.edu
25. Weibel S., Kunze, J., Lagoze, C., Wolf, M.: Dublin Core metadata for resource discovery. Internet Engineering Task Force RFC 2413 (September 1998) http://www.ietf.org/rfc/rfc2413.txt