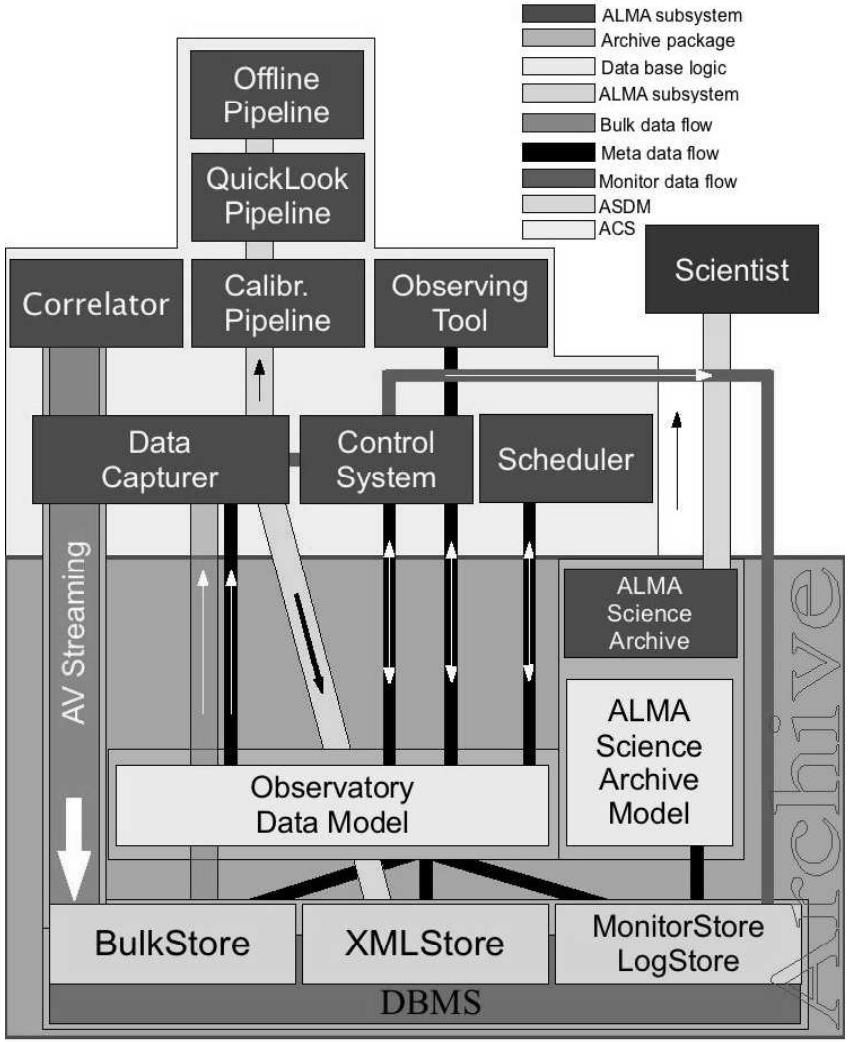
ARCHIVE TUTORIAL

Holger Meuss Andreas Wicenec

- Overview
- dbConfig file (Archive backend)
- Starting the Archive
- `XMLEntityStruct`: communicating documents
- Unique Identifiers for documents
- The XMLstore Interface
- Advanced UID usage
- Command line tools
- Useful Web pages
- UserRepository
- Not covered:
  - bulk data transfer
  - Archive manager (going to be replaced)
  - MonitorStore and LogStore
  - Science Archive

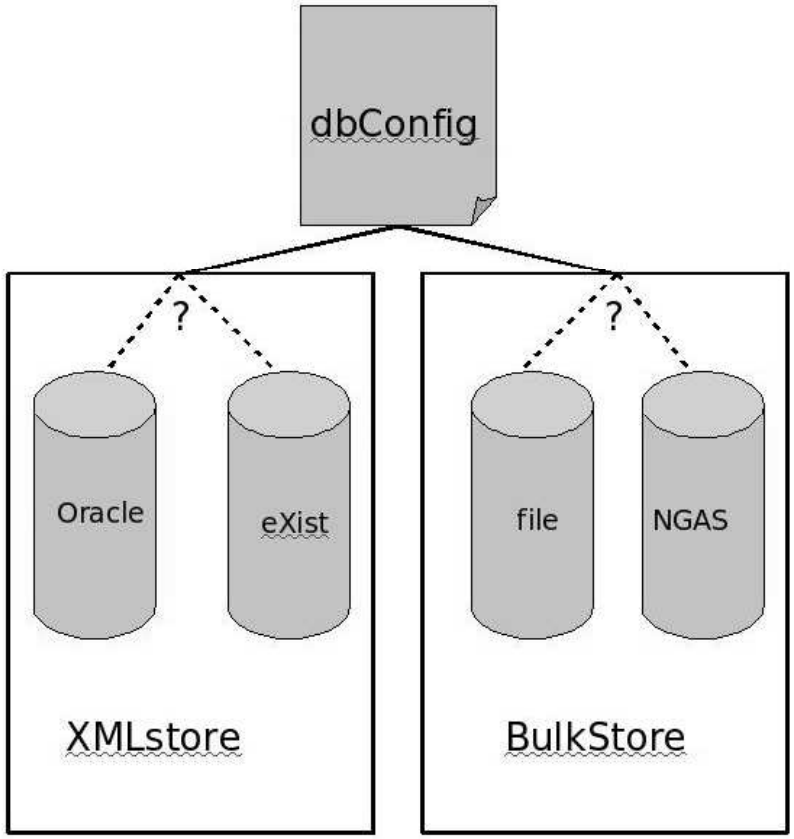
ALMA Archive Architecture



A. Wicenec 2006-09-18

ARCHIVE BACKENDS

- The Archive has two areas for storage: XMLstore and BulkStore
- Each area is implemented by local and central version
- Configuration file controls which version is used



DBCONFIG FILE

- ➜ specifies which Archive backend is used
- ➜ changed in the future (backward-compatible)
- ➜ default: in `archive_database.jar`. Override:
  1. `dbConfig.properties` in working directory
  2. `dbConfig.properties` in `$ACSDATA/config`

archive.db.backend=xmldb

archive.db.mode=test

```
archive.xmldb.driver=org.exist.xmldb.DatabaseImpl
archive.xmldb.name=db
archive.oracle.location=archive1:1521
```

archive.oracle.user=almatest

```
archive.ngast.server=archive1
```

archive.ngast.storeInNgast=False

archive.ngast.testDir=$ACS.data/tmp

```
archive.bulkreceiver.schema=sdmDataHeader
```

STARTING THE ARCHIVE

Either through master component **or** through `archive start`

- `archive start`:
  - starts tomcat, user repository and brings master component to operational
  - prerequisite: ACS and Archive components running
- Master component: bring to operational
  - will **not** start tomcat (necessary for eXist)
  - do **tomcat start** before

All of this: **only** in test code, not in operational code!

XMLEntityStruct

- used in most communication with Archive
- struct XmlEntityStruct

```
{
  string xmlString;
  string entityId;
  string entityTypeName;
  string schemaVersion;
  string timeStamp;
};
```

- Construct an XmlEntityStruct out of binding class:

```
EntitySerializer serializer = EntitySerializer
                .getEntitySerializer(logger);
XmlEntityStruct xmlEntity = serializer
                .serializeEntity(obsproposal);
                // obsproposal Castor binding class
```

UNIQUE IDENTIFIERS (UIDS)

have the form of a URI:

`uid://`*archiveID*`/`*global*`/`*local*

e.g.: `uid://X01/X2a/Xd`

- ➜ Each part is a hexadecimal number
- ➜ Assign UID to an entity (Java only):
  `containerServices.assignUniqueEntityId(EntityT entity)`

- ➜ Otherwise: UIDs must be fetched:
  `IDENTIFIER_ARCHIVE.getIdNamespace()`
  (or other possibilities)
- ➜ UID syntax checking provided
- ➜ No semantics behind UIDs (apart from archiveID)

Archive Interface

- Interface for storing, deleting and querying XML documents
- Document is identified by UID
- Document versions stored internally, but not visible to outside

- `void` **store**`(in xmlentity::XmlEntityStruct entity)`
- `void` **update**`(in xmlentity::XmlEntityStruct entity)`
- `void` **un/delete**`(in URI identifier)`
- `StatusStruct` **status**`(in URI identifier)`
- `xmlentity::XmlEntityStruct` **retrieve**`(in URI identifier)`
- `Cursor` **query**`(in string query, in string schema)`

`void` **store** `(in XmlEntityStruct entity):`

- ➜ Stores a new XML document using `entityId` in `entity` as UID.
- ➜ If UID already exists: exception

`void` **update** `(in XmlEntityStruct entity):`

- ➜ Updates an existing XML document using `entityId` in `entity` as UID.
- ➜ If UID does not yes exist: exception

`void` **delete** `(in URI identifier):`

- ➜ deletes (logically) a document
- ➜ can always be restored

`void` **undelete** `(in URI identifier):`

- ➜ restores a deleted version

`StatusStruct` **status** `(in URI identifier):`

- returns status information for document

A `StatusStruct` contains the following fields:

- `URI schema`
- `string owner`
- `string locks` (unused in the moment)
- `boolean deleted`
- `boolean dirty`
- `boolean hidden`

`boolean` **exists** `(in URI identifier):`

- returns true if documents exists in database

`boolean` **checkUIDsyntax** `(in string uid):`

- in `ARCHIVE_IDENTIFIER`
- returns true if syntax is correct

`XmlEntityStruct` **retrieve** `(in URI identifier):`

- ➜ retrieves a document

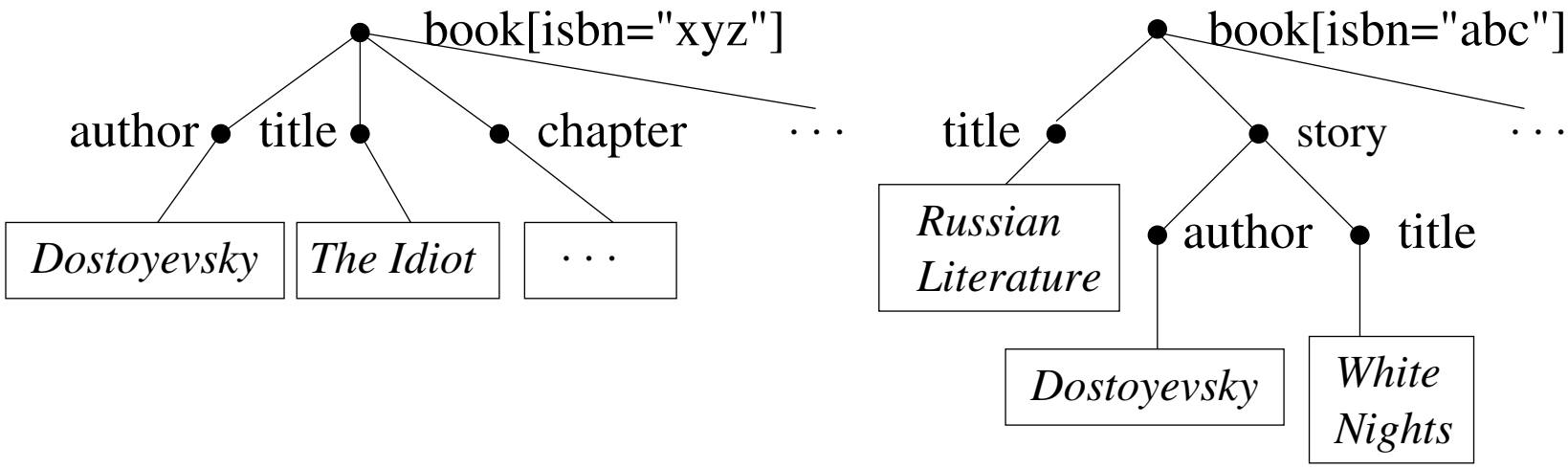
`Cursor` **query** `(in string query, in string schema):`

- ➜ XPath query against document
- ➜ Returns `Cursor` object containing document fragments
- ➜ query must contain namespace prefixes
- ➜ **Cursors must be closed after usage**: `cursor.close()`
- ➜ Variations: `queryRecent(in string schemaname, in string timestamp)` and `queryUIDs( in string query, in string schema)` both return array of matching document UIDs.

EXCURSION: XPATH

W3C standard for "pointing" at parts XML documents, similar to paths in file systems:

```
/book/title
/book//title
/book[./author="Dostoyevsky"]/title
/book@isbn
```



EXAMPLE XML DOCUMENT:

```
<book isbn="xyz">
  <author> Dostoyevsky </author>
  <title> The Idiot </title>
  <chapter> ... </chapter>
  ...
</book>
```

Syntactic Constructs:

- `/` child step
- `//` descendant step
- `.` self step
- `..` parent step
- `@` attribute step
- Simple functions
- Unabbreviated syntax: richer language

CURSOR

- Simple instrument to deal with big result sets
- works like `Iterator` in Java:
  - Method `hasNext()` to check whether more results exist
  - Method `next()` to get next result of type `QueryResult`:
    - `identifier`: UID of full document (`URI`)
    - `xml`: matching document fragment (`string`)
- Method `nextBlock(in short size)`: gets next `size` results in array
- Results are living in archive
- **Cursors must be closed after usage**: `cursor.close()`

Permissions and Locking

Permissions infrastructure prepared, but probably not necessary

Optimistic locking via timestamps and "dirty" entities:

- `update` compares timestamps of document to ensure that latest document was used and no other update gets overwritten.
  - Can be disabled with **forceUpdate**
- **updateRetrieve(in URI identifier)** flags entity as dirty
  - To be used if user is sure to update the entity
  - Protection against someone else trying to retrieve (and then update)
  - Dirty entities cannot be retrieved (only with **retrieveDirty**)
  - are invisble to queries (apart from **queryDirty**)
  - Next `update` unsets dirty flag

EXAMPLE

- get Archive and Identifier reference:

```
Operational archive = ...
Identifier ident = ...
```

- create document structure

```
XmlEntityStruct struct = new XmlEntityStruct();
struct.xmlString = "<example>example</example>";
...
```

- get an ID:

```
containerServices.assignUniqueEntityId(struct);
```

- store document structure

```
archive.store(struct);
```

- retrieve document

```
XmlEntityStruct struct = archive.retrieve(id);
```

ADVANCED UID USAGE:

UID ranges: fetch range one time, get UIDs many times

- ➜ Java and C++ libraries running **inside** your components
- ➜ A UID range object is an XML document itself, stored in the Archive and has `X0` as local part.
- ➜ get a range from `ARCHIVE_IDENTIFIER`:

```
Range idRan = new Range(ident.getNewRange());
```

- ➜ assign UIDs:

```
ran->assignUniqueEntityId(xmlEntity);
```

- ➜ Assigning references: Fetch given range from Archive. It is *locked*: no UIDs can be assigned, only references to documents:

```
URI rangeUid = idRan->rangeId();
Range refRan = new Range(
          ident.getExistingRange(rangeUid, "subsystem name"));
Xmlentity entityRef;
refRan.assignUniqueEntityRef(entityRef);
```

COMMAND LINE TOOLS: All use dbConfig file

- `tomcat [start|stop|status]`
- `archive [start|stop]`
- `archiveLoadSchema -[ucl] [file|directory]`
  - `c`: Clean database first (removes **everything**). **BUG!**
  - `u`: Update schemas already stored. **BUG!**
  - `l`: Load from an `INTLIST`
  - In the moment: use no command line parameter to avoid bugs in `archiveLoadSchema`
- `archiveQuery [-i] [-q XPathQuery schemaName [-w fileName]] [-x UID]`
  - Returns full (`-x`) or UIDs (`-q`) of matching documents
  - Many symbols in queries have to be escaped: `/, ", ',` blanks
- `archiveCleanTest`:
  Deletes all data from database **if** in test mode.

Most tools display help when no parameter given

WEB RESOURCES:

- Archive Twiki home:
  `almasw.hq.eso.org/almasw/bin/view/Archive/WebHome`
- Release notes:
  `almasw.hq.eso.org/almasw/bin/view/Archive/ReleaseNotes`
- UID ranges:
  `almasw.hq.eso.org/almasw/bin/view/Archive/RangeUidUsage`
- Archive tutorial: `almasw.hq.eso.org/almasw/pub/Archive/ArchiveTutorial/archive-tutorial.html`
  Old and not updated, but contains overview of functionality
- Command line tools:
  `almasw.hq.eso.org/almasw/bin/view/Archive/ArchiveSwDocs`
  Under construction, not much usefull info there yet