Harvester Implementation and Use

(see also Harvester Implementation Guidelines: http://www.openarchives.org/OAI/2.0/guidelines-repository.htm)

Be a Polite OAI Neighbor

- Re-use existing free harvester software/libraries: http://www.openarchives.org/tools/index.html
- If you insist on writing your own harvester, read http://www.robotstxt.org/wc/robots.html
- Provide meaningful `User-Agent` & `From` headers
  - Should be present in HTTP headers of all robot requests
  - Should be configured even if using someone else's harvester

Harvesting Sequence

- Issue `Identify` request
  - Check OAI-PMH version
  - Check `baseURL`, `granularity`, `compression`
- Issue `ListMetadataFormats` request
  - Get information regarding selected `metadataPrefix`
- Issue `ListSets` request if using sets
  - Check set structure matches expectation
- Issue `ListIdentifier` or `ListRecords` request
  - Continue until end of complete list

Listen to the Repository

- Check Identify's `<granularity>` element if you wish to use finer than YYYY-MM-DD
- If you harvest with sets, remember that ":" indicates hierarchy
  - harvesting "a" will recursively harvest "a:b", "a:b:c", and "a:d"
- Check for and handle non-200 HTTP status codes, 503, 302 and 4xx in particular
- Empty resumptionToken => end of complete list
- Ask for compressed responses if the repository supports them

Harvesting *Everything*

- Issue an **`Identify`** request to find protocol version, finest datestamp granularity supported, if compression is supported…
- Issue a **`ListMetadataFormats`** request to obtain a list of all `metadataPrefixes` supported.
- Harvest using a **`ListRecords`** request for each `metadataPrefix` supported. Knowledge of the datestamp granularity allows for less overlap in incremental harvesting if granularities finer than a day are supported.
- Set structure can be inferred from the `setSpec` elements in the header blocks of each record returned (consistency checks are possible).
- Items may be reconstructed from the constituent records.
- Provenance and other information in `<about>` blocks may be re-assembled at the item level if it is the same for all metadata formats harvested. However, this information may be supplied differently for different metadata formats and may thus need to be store separately for each metadata format.

Harvesting v1.1 and v2.0

- Not difficult to handle both cases, test `Identify` response:
  - v1.1: `<Identify> <protocolVersion>`
  - v2.0 `<OAI-PMH> <Identify> <protocolVersion>`
- Different error and exception handling
- Many similarities, harvesters can share lots of code

Harvesting Demo

- Harvester written in Perl (Uses `LWP`, `Expat` and `XML::Parser`, no schema validation)
- Handles v1.0, v1.1 and v2.0
- Sequence of requests: `Identify`, `ListMetadataFormats`, `ListSets` then `ListRecords/ListIdentifiers`
- Support for incremental harvesting, uses `responseDate` from last harvest to get new start `datestamp`
- Supports response compression (`gzip`, `compress`)
- UTF-8 conditioning to deal with some "imperfect" repositories

Harvesting logs

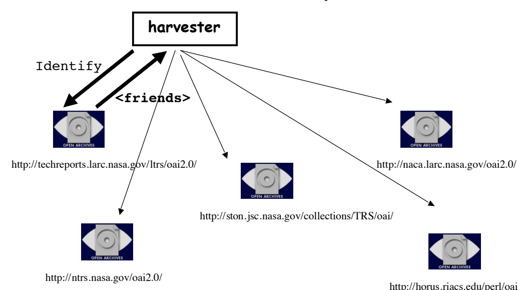
- Alan Kent's v2.0 harvester logs: http://www.inquirion.com:8123/public/collList;collListCmd=list
- Alan Kent's summary of v1.1 harvesting results http://www.mds.rmit.edu.au/~ajk/oai/interop/summary.htm
- Celestial v1.1 harvesting logs http://celestial.eprints.org/cgi-bin/status
- DP9 gateway using arc harvested information http://arc.cs.odu.edu:8080/dp9/index.jsp

`<friends>` example (1)

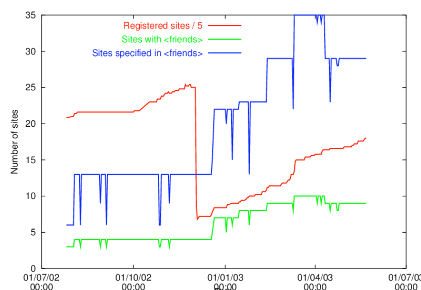
A light-weight, data-provider driven way to communicate the existence of "others", e.g.
http://techreports.larc.nasa.gov/ltrs/oai2.0/?verb=Identify

```
…
<description>
 <friends  …namespace stuff… >
   <baseURL>http://naca.larc.nasa.gov/oai2.0</baseURL>
   <baseURL>http://ntrs.nasa.gov/oai2.0</baseURL>
   <baseURL>http://horus.riacs.edu/perl/oai/</baseURL>
   <baseURL>http://ston.jsc.nasa.gov/collections/TRS/oai/</baseURL>
 </friends>
</description>
…
```

`<friends>` example (2)



Use of <friends>



Aggregator / Cache / Proxy Implementation

(see also Aggregator Implementation Guidelines: http://www.openarchives.org/OAI/2.0/guidelines-aggregator.htm)

<provenance> & datestamps

- Reminder: datestamps are local to the repository, a re-exporting service **must** use new local datestamps
- Such services **should** use the <provenance> container to preserve the original datestamps and other information

Identifiers are Local

- Identifiers are local to the repository
- Unless you *absolutely* did not change the metadata *and* the identifier corresponds to a recognized URI scheme, use a new identifier upon re-exporting
  - use the <provenance> container to preserve the harvesting history

oai-identifier

- Just one option for identifiers in OAI-PMH
- The v2.0 oai-identifier scheme is not compatible with v1.1:
  - repositoryName now domain name based
  - not reliant upon OAI centralized registration
- One-to-one mapping for escaping characters: %3F allowed, %3f not
  - allows simple comparison

Derived from the same item?

3 ways to determine if records share provenance from the same item:

1. both records have the same identifier *and* the baseURL in the request elements of the OAI-PMH reponses which include the record are the same;
2. both records have the same identifier *and* that identifier belongs to some recognized URI scheme;
3. the provenance containers of both records have the same entries for both the identifier and baseURL;

<provenance> example (1)

Consider a request from crosswalker.oa.org:

```
http://odd.oa.org?verb=GetRecord
  &identifier=oai:odd.oa.org:z1x2y3&metadataPrefix=odd_fmt
```

and the following response from odd.oa.org:

```
<responseDate>2002-02-08T08:55:46.1</responseDate>
<request verb="GetRecord" metadataPrefix="odd_fmt"
     identifier="oai:odd.oa.org:z1x2y3">http://odd.oa.org</request>
<GetRecord …namespace stuff…
  <record>
    <header>
      <identifier>oai:odd.oa.org:z1x2y3</identifier>
      <datestamp>1999-08-07T06:05:04Z</datestamp>
    </header>
    <metadata> …metadata record in odd_fmt… </metadata>
  </record>
</GetRecord>
```

<provenance> example (2)

Imagine that crosswalker.oa.org cross-walks harvested metadata from odd_fmt into oai_marc and then re-exposes the metadata with new identifiers.

A request from getmarc.oa.org:

```
http://crosswalker.oa.org?verb=GetRecord
  &identifier=oai:cw.oa.org:z1x2y3
  &metadataPrefix=oai_marc
```

might then yield the following response from crosswalker.oa.org:

<provenance> example (3)

```
<record>
  <header>
    <identifier>oai:cw.oa.org:z1x2y3</identifier>
    <datestamp>2002-02-09T01:15:43Z</datestamp>
  </header>
  <metadata> ...metadata record in oai_marc... </metadata>
  <about>
    <provenance …namespace stuff… >
      <originDescription harvestDate="2002-02-08T08:55:46Z"
                         altered="true">
        <baseURL>http://odd.oa.org</baseURL>
        <identifier>oai:odd.oa.org:z1x2y3</identifier>
        <datestamp>1999-08-07T06:05:04Z</datestamp>
        <metadataNamespace>http://odd.oa.org/odd_fmt</..>
      </originDescription>
    </provenance>
  </about>
</record>
```

<provenance> example (4)

This `oai_marc` record is then re-exposed by `getmarc.oa.org` with the same identifier `oai:cw.oa.og:z1x2y3` (because the record has not been altered).

The associated `<provenance>` container might be:

<provenance> example (5)

```
<record>
  <header>
    <identifier>oai:cw.oa.org:z1x2y3</identifier>
    <datestamp>2002-03-01T01:46:11Z</datestamp>
  </header>
  <metadata> ...metadata record in oai_marc... </metadata>
  <about>
    <provenance …namespace stuff…>
      <originDescription harvestDate="2002-03-01T01:23:45" altered="false">
        <baseURL>http://crosswalker.oa.org/<baseURL>
        <identifier>oai:cw.oa.org:z1x2y3</identifier>
        <datestamp>2002-02-09T01:15:43Z</datestamp>
        <metadataNamespace>http://../oai_marc</metadataNamespace>
          <originDescription harvestDate="2002-02-08T08:55:46Z" altered="true">
            <baseURL>http://odd.oa.org</baseURL>
            <identifier>oai:odd.oa.org:z1x2y3</identifier>
            <datestamp>1999-08-07T06:05:04Z</datestamp>
            <metadataNamespace>http://odd.oa.org/odd_fmt</metadateNamespace>
          </originDescription>
       </originDescription>
    </provenance>
  </about>
</record>
```

Case Studies

Example data-provider implementations for:

arXiv.org

NSDL metadata repository

arXiv (1)

http://arXiv.org/oai2

- Existing system, running >11 years, written mostly in Perl
- Flat file system for 'database'
- 230k papers with metadata in homebrew format
- ~200 updates/day. OAI repository just one view of system, must integrate with daily update schedule

arXiv (2)

- Write in Perl
  - Easy integration with rest of system, reuse code from v1.0/v1.1 interface
  - Use `libwww; XML::DOM`
- Daily rebuild of datestamp database
  - No existing date in system appropriate
  - Base on Unix `cdate` of metadata files
- On-the-fly metadata translation
  - Straightforward, avoids data duplication

arXiv (3)

- Flow control to avoid loading server and to avoid harvesters tripping robot alarms
  - `resumptionTokens` to limit response size (1500 records or 15k identifiers / response)
  - 503 `Retry-After` replies based on client ip
- Implement `resumptionTokens` that include all state
  - Avoid need to cache result sets / clean cache

NSDL Metadata Repository

http://services.nsdl.org:8080/nsdloai/OAI

- Implemented as an integral part of a new system
- Expect heavy load; db target size >10M items; stateless resumptionTokens
- Java servlets; Xerces; Oracle (JDBC interface); strict validation throughout
- Based on rewrite of Cocoa (NCSA UIUC)
- Integral to NSDL services model: provides data for user interface and search services