**Characterizing current EduPak systems**

**(Taken from** [**http://wiki.nsdl.org/index.php/TNS\_Internal:EduPakArchRoadmap**](http://wiki.nsdl.org/index.php/TNS_Internal:EduPakArchRoadmap) **on 9/18/2009****)**

**NDR 1.x**

Basic I/O

NDR API provides basic CRUD functionality. The API is basically a proprietary XML-based protocol that has REST-like and RPC-like characteristics. The basic unit of data is the 'object', which itself is composed of 'properties' (key/value), 'datastreams' (XML or binary blob), and 'relationships' (named relationships to other NDR objects), and there is a pre-defined ontology of object types, and property/relationship/datastream names. The API allows wholesale creation and deletion of single objects, or updates of individual components in an object. A Java-based toolkit is provided that allows manipulation of NDR objects as java objects, thus hiding the details of the XML API, which is used in an underlying driver layer.

In defining the various object types (Agent, Aggregator, Metadata, etc) and required relationships, the NDR model is specifically oriented towards representing provenance, and controlling access to data that is contributed by multiple applications.

* The XML-based API itself has proven somewhat cumbersome and/or difficult to use in its raw form. The NDR toolkit helps immensely here.
* Even with the NDR toolkit, manipulating higher-level concepts (e.g. 'collections', which are represented my multiple interrelated objects) has proven to be difficult and/or error prone. Creating a collection, for example, requires multiple steps that must be performed serially and correctly (e.g. creating the correct object types, adding the proper relationships to connect them in the correct way).
* Graph traversal can be slow and cumbersome
* A toolkit 2.0 prototype has been created to present a "high level" interface to constructs such as collections in the NDR.

Search and Query

The NDR API provides for matching objects based on their properties and relationships, as well as convenience methods for finding resources by their URL or finding all objects that have a relationship that points toward a given object.

* NDR Search API is limited to "structural" components such as relationships or properties. Datastream content, such as full text or metadata, is NOT searchable via the NDR API.
* In developing the Web Feed Ingest, Jim ran across many situations where tasks such as "List all titles of collections" could not be addressed by NDR search API (or NSDL metadata-oriented search), and could ONLY be addressed by brute force crawling of individual objects, which is very expensive
* Outside the NDR API, some of our infrastructural applications (such as OAI provider or sitemap generators) can and do use Fedora's Resource Index directly to perform more complex queries. Still, these queries are limited to NDR "structural" components such as properties and relationships, rather than "content' present in datastreams.

Interoperability

The NDR 1.x strives to achieve interoperability through its shared data model. Resource objects are non-redundant, and are shared in the sense that all aggregations and metadata involving a given resource are co-located around the single physical object. All data, regardless of source, contribute to a single "big graph" of related objects, which itself is visible to every application. nsdl\_dc is used as a least-common-denominator metadata format.

Scalability

The NDR was designed for managing millions of objects from many sources, and to accommodate both large highly parallel bulk operations and small ad-hoc updates.

* The NDR has been able to successfully manage over 7 million objects, thanks in part to the adoption of PostgreSQL-backed MPTStore triple store.
* NDR API searches are very quick (typically < 10ms), and consume few system resources.
* Data access involving a fetch from Fedora typically take ~100ms, and heavy read/access load has not caused a problem for the NSDL in production.
* RI queries outside the NDR API, as used by PROAI, can be quite expensive and I/O intensive with large data sets, and became a problem in NSDL production running during the day - so we ran the job at night.
* Ingesting into the repository is slower than ideal. On the NSDL's production hardware, concurrent data ingests never exceeded 10 records/second.
* Fedora rebuilds are relatively slow - on a 7-million object repository, a rebuild of the RI and Fedora's internal SQL registry took several days in NSDL hardware.
* As the repository gets large, if Fedora's journalling is enabled, the opening of a new journal file can take a noticeably large time. At its peak size, it would take up to two minutes to open a journal in NSDL production

Availability

The NDR has several potential points of failure: the NDR software itself, Fedora, and databases used by Fedora. Luckily, Fedora journalling allows for maintaining mirror servers that can step into production in the case of a failure.

* Re-configuring the leader/follower cluster necessarily requires the re-start of the leader Fedora instance, which implies at least a few seconds of downtime when this happens
* Upgrading Fedora versions requires a major migration procedure, which requires downtime or read-only operation for the duration of the process

Security

The NDR provides a proprietary public-key based authentication system [inspired by the system used by Amazon S3.](http://docs.amazonwebservices.com/AmazonS3/latest/index.html?RESTAuthentication.html) Write access to NDR objects is controlled by a permissions system, which uses the auth:authorizedToChange relationship to enumerate the specific agents (or aggregators of agents) allowed to modify given objects. Read access to objects is not restricted. Some operations (such as attaching Aggregators/MetadataProviders to other Agents) are restricted to so-called "trusted" applications.

* In practice, access keys are assigned one per NDR-aware application, not on a user-based level. If the application manages users (such as EV blog), the the application itself is responsible for managing user rights within the application - the NDR doesn't do anything to assist. The application (under its own, single NDR access account) then performs actions on behalf of the user
* SSL is not supported in NDR 1.x so far

Extensibility

Although the NDR defines a basic relationship and property ontology, applications may define their own properties, relationships, or datastreams as they wish. These extensions are visible to all applications, and may be used in NDR search requests. In addition, the ordering and membership of objects may be given specific meaning (e.g collections)

* Object types are a fixed set
* For custom properties, relationships, or datastreams, there is no guarantee that another application will understand anything outside of the basic ontology.

**NCS 2.7.x + DDS**

The NCS is an application for cataloging 'records' (discussed below), which are essentially XML files containing mostly arbitrary content. The DDS query and indexing service, based on Lucene, which can store the the content of XML elements as searchable fields in its index. Although the NCS and DDS are distinct applications, the NCS is 'built around' a DDS instance such that this local DDS indexes all the records that have been generated by NCS. It cannot function without a DDS index of its data, hence NCS is distributed with its own internal DDS instance in EduPak. At various points in this section, NCS and "NCS + DDS" are taken to mean the same thing: The NCS application with its included DDS instance.

Basic model and I/O

At the most basic level, the data model is any XML document with a schema. In addition, XML document models that extend a 'record' or an 'annotation' pattern acquire special features that enhance their searching and discoverablilty. Several schemas are configured by default (ncs\_item, ncs\_data, etc). Access to NCS CRUD functionality is through the NCS GUI and its editing framework. NCS stores records on the filesystem, and updates DDS index which then indexes these records. The DDS CRUD API is not exposed to applications outside the NCS. Records can be exported to the NDR from the NCS.

* The 'record' pattern specifies some expected elements. For example, these elements, expressed in XPath here, are expected to be present in a 'record' (they can reside at any XPath - these are examples). An XML schema that is configured to specify these elements acquire uniform search-ability by title, url, description, ID, geospatial bounding box and so forth:
  + /record/general/recordID
  + /record/general/url
  + /record/general/title
  + /record/general/description
* The 'annotation' pattern is used to define a special relationship 'annotates'. An XML schema that is configured to specify this relationship acquires special behavior: It injects the content from the annotation record into the index document corresponding to the record being annotated. The record being annotated can then be searched using the data found in the annotation (relation 'isAnnotatedBy'), and search results in the REST service return the annotation XML along with the record XML.

Search and query

DDS maintains a Lucene index of all records, and exposes the DDS search API for queries over indexed fields.

* By default, all elements and attributes present in the XML documents are indexed for searching. Thse search fields are generated automatically using field names that correspond to the XPath locations.
* There are a 'standard' set of fields to index that are expected to be present for the 'record' and 'annotation' data models (enumerated in Basic model and I/O section)
* Additional custom fields may be configured for indexing by specifying XPath locations.
* Modifying the index field definitions (e.g. adding a new field to be indexed) would imply the need to re-build the index, unless the change is irrelevant to the current set of data.

Interoperability

The search REST API allows other applications read-only access to almost all data (i.e. data configured to be indexed into the DDS). There is no fully-developed external write API to NCS + DDS; all data in NCS + DDS is either created by hand via the cataloging interface, or imported from another source by hand (with some exceptions)

* The NSDL.org ui takes advantage of data in the NCS through its search API for associating brand images with collections or records.
* Besides the standard fields, the fields accessible via DDS API are entirely dependent on the configuration of the index, and the record formats present in NCS.

DDS REST search API can return JSON as well as XML, making it very JavaScript friendly

Availability

Events that require downtime for NCS + DDS are rare. NCS stores records on file system, and DDS can be re-built from these records. NCS can also save and import records from an NDR.

Scalability

* Once the number of records exceeds several thousand or so, update performance becomes a problem. Editing a record in the NCS results in a real-time update to the index.
* On the read/access side, Lucene and DDS perform well.

Security

Accounts/login, data access controls for reads or writes.

NCS maintains its own local user, password, and role list which are used for physical human login access. The DDS update API is disabled by default for external applications, but when active it permits access to all index records for clients at specified IP addresses

* Access to records in NCS is based on role, and is not enforced on an individual record or collection basis.

Extensibility

NCS can accommodate basically any XML-based record format.

**DDS 3.4.x Stand-Alone**

The DDS Stand-Alone application provides DDS search services for records that are managed by external applications. Records are placed into DDS Stand-Alone via the DDS CRUD Rest API, via regularly-occurring NDR imports or from files on disc managed by another application. A DDS repository, like the NCS, consists of collections of XML records. External applications can define collections of arbitrary XML formats. If desired, schema validation must be performed by the external application prior to placing them into DDS. The DDS Stand-Alone provides the same data model, configuration and search capabilities as NCS+DDS (see above).

Defining collections of records

* The DDS CRUD API has methods to put and delete collections, defined by XML format (nsdl\_dc, etc.) and a unique collection identifier (key). Once a collection is defined, API methods may then be used to put and delete records in those collections.
* NDR collections are configured using NDR handle. An NDR handle identifier becomes a collection in the DDS. DDS updates its index at regular intervals to reflect any changes made in the NDR.
* If DDS is configured off of files and disc, the external application writes collections records to disc, which are used to define the collections in DDS. DDS then updates its indexes at regular intervals to reflect changes made to the XML files on disc.

Java Bean support

* DDS provides native support for modeling Java Beans that are serialized to XML using the java.beans.XMLEncoder class. Properties defined in a Bean automatically become searchable fields in the index. This provides a simple way to search over data in a Bean and marshal to and from Java Objects.

Availability

* DDS indexes can be re-built or updated simultaneously while supporting searches. The process of updating or re-indexing a record consists of two separate operations: delete and then (re)index. Thus individual records become unavailable in searches momentarily at the time they are re-indexed.
* When the DDS CRUD API is enabled, changes are made durable as a store of files on disc. The Lucene index can be restored from the file store or backup copy of files.
* DDS CRUD API does not have the concept of a user or login - it's strictly all-or-nothing access based on IP, or restricted to a local tomcat running behind a firewall.

Scalability

The NCS and DDS were designed for relatively small collections of rich content. The DDS Stand-Alone has been used with repositories as large as 300,000 records. The NCS+DDS up to about 20,000 (on a slow machine). Further development can enable scaling to larger sizes.