DSpace –

A Sustainable Solution for Institutional Digital Asset Services –

Spanning the Information Asset Value Chain: Ingest, Manage, Preserve, Disseminate

INTERNAL REFERENCE SPECIFICATION

Technology & Architecture

Michael J. Bass,
David Stuve,
Robert Tansley

Hewlett-Packard Company
Building 10-500 MIT
77 Massachusetts Avenue, Cambridge MA 02139
+1 617 253 6617
mick_bass@hp.com

Margret Branschofsky,
Peter Breton [1],
Peter Carmichael [2],
Bill Cattey,
Dan Chudnov,
Joyce Ng

Masssachusetts Institute of Technology
Building 10-500 MIT
77 Massachusetts Avenue, Cambridge MA 02139
+1 617 253 xxxx
dspace-dev@mit.edu

[1] Under contract to MIT

[2] Under contract to MIT

Version 2002-03-01

CONTENTS

1. AUDIENCE

This document is intended to provide a technology and architecture overview of the DSpace system. It is aimed at individuals who wish to understand, evaluate, or provide feedback on the technology and architecture choices that the design team has made.

This document is intended to be the basis for more detailed technical documents that detail the interfaces of each of the components of the DSpace system architecture.

2. TECHCHNOLOGY & ARCHITECTURE

2.1 Philosophy and Values

2.1.1 Information-Centric

We wish to construct a system that will enable the information that the system deals with to outlive the system itself. It is an overt expectation that information assets managed by the DSpace system will outlive the current system, the current implementation of components within the architecture, as well as external implemented services that access and/or add value to the corpus.

This means that historic silo-oriented views of information living "inside" an asset management system will not be sufficient. Assets will be used by many parties, each with different world views, for many purposes. Systems and services in the future will flow around existing information assets, and standards-based mechanisms for access to those assets will need to be a part of the web infrastructure.

2.1.2 Pre-Competitive: Low Adoption Barriers

DSpace is pre-competitive. Because in the long-term we wish to increase the baseline capabilities for information asset management in the web infrastructure, we must be committed to DSpace being both useful and adoptable by many institutions. Low adoption barriers and a focus on providing useful functionality will provide the grounding use cases, quick feedback, and a channel for widespread and visible deployment of demonstrated results.

To ensure that adoption barriers remain low, HP and MIT have agreed to license all software produced within the joint project with an open-source, BSD license.

Further, where third-party components are incorporated into the DSpace system, we strive to choose components that are freely available under similar terms.

2.1.3 Suitable for Research

In addition to DSpace being useful to those institutions that adopt and use it, we are committed to DSpace as a useful vehicle for research. As we have designed the initial DSpace system we have in most cases chosen to keep the footprint of the system as small as possible while still meeting the needs of early adopter customers. This small footprint keeps the barriers to experimentation low.

We expect that as a result of future research, some components of the architecture will be replaced, enhanced, or subsumed in various ways, as we explore different techniques and toolkits that enable a lower long-term cost-of-management and a higher long-term value for a very large corpus of information.

2.1.4 Use and Lead Standards

Because the system must be information-centric, we are committed to using relevant standards for creating, describing, and accessing information and information assets.

In many cases there is no clear standard, or existing standards are not yet mature. In these cases we hope to test, inform, and influence the development of useful and appropriate standards.

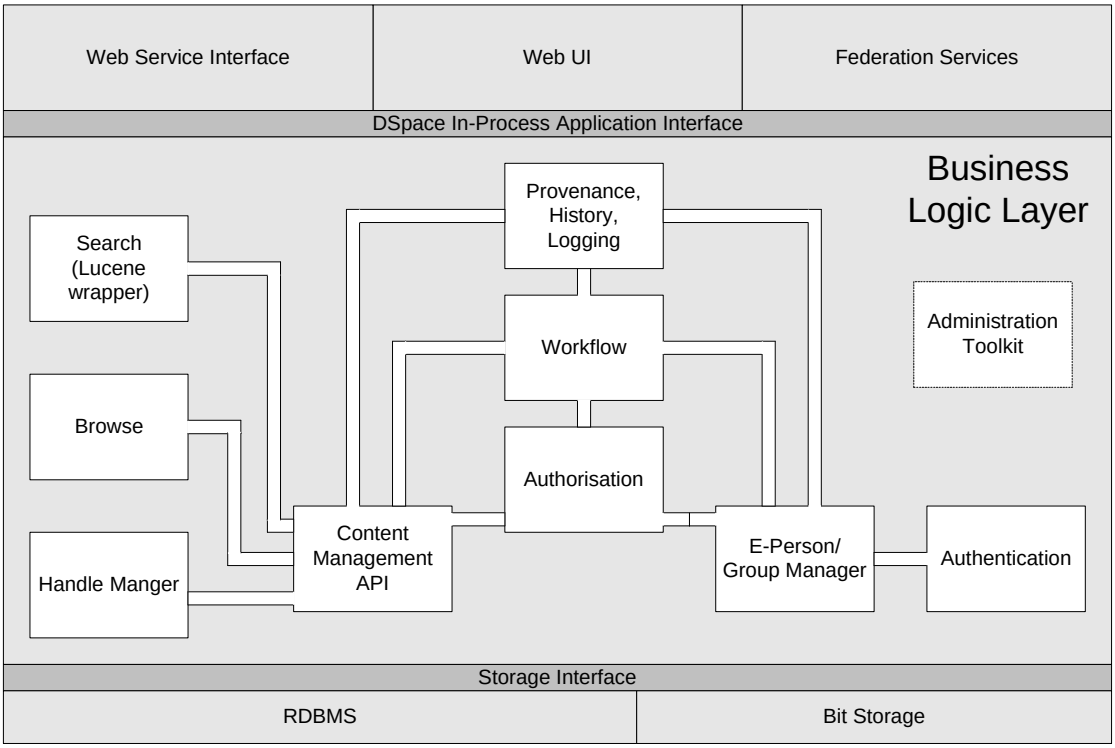
2.1.5 Sustainable

A core part of the DSpace value proposition is enabling institutions with a sustainable ability to retain information assets and offer services upon them. This means that economic and social consideration must come must be considered with as much weight as technical considerations in evaluating alternative directions for the project.

2.2 Architecture Overview

2.2.1 Three-Layer Architecture

The DSpace platform is separated into three distinct layers, as illustrated below. From the bottom up, these layers are the storage, business logic and service layers.



DSpace System Architecture

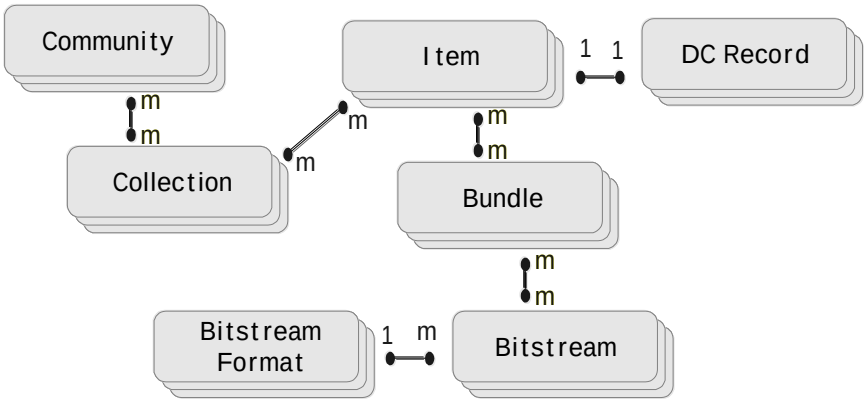
The lowest layer is the storage layer. This presently consists of a relational database for storing metadata and a "bitstream" storage module for storing content data. Each of these has an API accessible to the business logic layer. The union of these APIs comprises the storage interface.

The central layer contains the modules that perform the business logic of the system. The diagram above displays the internal plumbing between these modules. Each module has a "public" API. The union of these APIs comprises the DSpace "in-process application interface." It is on this API that services such as the Web user interface and future interoperability and federation services are built.

The top layer of the Dspace platform is the services layer. At present, the only implemented service is the Web user interface, though an Open Archives Initiative metadata harvesting protocol service is to be added shortly.

2.2.2 Data Model

Content and metadata in DSpace are logically organized into the simple data model illustrated below.



DSpace Logical Data Model

- Content in DSpace is at the highest level organized into **communities**. These correspond to organizational bodies in an institution, such as departments, labs, research centers or schools.
- A community is organized into **collections** of logically-related material. For example, a technical report series might be a collection.
- An **item** is an "archival atom"; that is, a grouping of content and metadata that it makes sense to archive as a single unit. This may take the form of a journal article, a dataset, or perhaps a technical report together with a dataset used in experiments described by the report. Precisely what constitutes an archival atom is largely a policy-driven decision.
- Each item has one **Dublin Core metadata record**. Other metadata might be stored in an item as a serialized bitstream, but we store Dublin Core for every item for interoperability and ease of discovery. The Dublin Core may be entered by end-users as they submit content, or it might be derived from other metadata as part of an ingest process.
- The content of items, and any serialized metadata, are stored in **bitstreams**. These are organized into **bundles** of closely tied bitstreams. For example, an item might contain a dataset in flat text file, and a technical report in an HTML document. The dataset text file would be stored in one bundle, and the HTML files and associated image files that make up the technical report would be grouped together in another bundle. We use the METS[1] metadata standard to store the relationships between bitstreams in a bundle.
- Each bitstream is linked to one **bitstream format**. This is a set of information, maintained by the institution running the archive, describing as much as we know about the format and encoding of the bitstream, including MIME type and name of the type (e.g. "Adobe PDF"). This may also hold information such as the specification of the format, and source code for manipulating the format. Each format additionally has a "support level", indicating how well the hosting institution is likely to be able to preserve content in the format in the future.

The relationships between communities, collections, items, bundles and bitstreams may all be many-many. This is because it is unlikely that all material in DSpace will be organized in a strict hierarchy. Duplication of content or objects should be avoided, since apart from inefficiency of storage, rights management information, such as distribution and access policies, needs to be uniformly applied to a piece of content wherever it appears in DSpace.

2.2.3 Relationship with OAIS

DSpace is deeply informed by the Open Archival Information Systems (OAIS) reference model[1]. The OAIS reference model provides a thorough vocabulary for describing media archive systems, and for crosschecking the functional and operational plans for a proposed archive. Where possible, DSpace adopted the OAIS model and vocabulary to articulate DSpace design objectives and terminology.

At a high level, several mappings from the OAIS model to DSpace may be particularly informative: First, in OAIS "producers" submit information to an "archive", which provides access to "consumers" who comprise the "designated community" of an archive. In DSpace, producers are primarily MIT faculty and their designates; the primary designated community is all of MIT, and a secondary designated community is made up of academic researchers world-wide. The DSpace platform will provide the tools for the MIT

Libraries to administer the archive, as well as to accept submissions from producers and allow access appropriately to our communities. These distinctions, also indicated in the OAIS separation of "submission", "archival", and "dissemination" "information packages", are shaping our development and implementation efforts by the clear boundaries between tools and processes the OAIS defines for each.

Another element of the OAIS model as adopted by DSpace is the OAIS concept of an "information object", made up of a "data object" and its "representation information". The DSpace team is currently defining internal and external specifications, which address the concerns motivating the distinctions between these within OAIS, and believes that these distinctions are critical concerns for the success of the project.

2.3 Subsystem Details

2.3.1 Relational Database

We chose to use a relational database management system (PostgreSQL) to manage data in DSpace for several reasons. First, DSpace captures a great deal of information about relationships between users, content, user groups, and content groupings. These naturally fit the relational model. Second, this information can change regularly, particularly as content and users are added to the system, so database updates need to be transactionally safe according to the ACID model to maintain the integrity of relationships across changes. Finally, SQL provides a straightforward and well-known query environment which suits nearly all our needs for searching, browsing, access control, and user management.

We chose PostgreSQL because it addresses these concerns as an ACID-compliant relational database engine, including a SQL92 implementation. Additionally, because PostgreSQL is distributed with an Open Source license, there are no barriers to our development in running multiple instances, or to anyone wishing to codevelop or implement DSpace elsewhere. The availability of PostgreSQL on many platforms further reinforces its wide availability and low-barrier adoption requirements to future DSpace users or developers.

2.3.2 Bitstream storage

The goals of the bitstream storage system are to provide a simple API for pluggable low-level storage (e.g., as flat files, database BLOBs, WebDAV, etc) and to enable adaptive, negotiated storage and retrieval. Both storage and retrieval are policy-driven, so that different types of bitstreams (e.g., streaming audio or video) can use different storage mechanisms. This usage is transparent to the clients of the storage system.

The current implementation of the bitstream storage system is fairly simple. Clients access bitstreams via a Bitstream Storage Manager, which provides a high-level API to store, fetch and delete bitstreams (note that bitstreams cannot be modified in-place via the API). The Bitstream Storage Manager uses the database to store metadata about the bitstreams and to provide a limited transactional capability for bitstreams. Internally, the Bitstream Storage Manager delegates the actual store, fetch and delete operations to storage components, which handle the low-level storage details. As shipped, DSpace uses a single storage component, which stores all data in the file system.

2.3.3 Persistent Naming

Researchers require a stable point of reference for their works. The simple evolution from sharing of citations to emailing of URLs broke when Web users learned that sites can disappear or be reconfigured without notice, and that their bookmark files containing critical links to research results couldn't be trusted long term. To help solve this problem, a core DSpace feature is the creation of persistent URLs for every item stored in DSpace. To persist URLs, DSpace requires a storage- and location-independent mechanism for creating and maintaining URLs.

Currently DSpace uses the CNRI Handle System[3] for creating these URLs. We run the CNRI Handle Server, which offers generic URL redirection across the naming authority and name resolution mechanisms maintained centrally by CNRI. Our implementation of Handles uses a DSpace-specific storage mechanism, so Handle information is stored inside DSpace, rather than the default external storage provided. These fit together when a new item is submitted to DSpace: A new Handle/URL is assigned to that item, and is displayed prominently to the submitter and to anyone browsing that item later. When anyone browses to the Handle URL later, the URL passes through the main CNRI Handle server, which redirects to the DSpace Handle server, which looks up the item inside DSpace and displays the item. This model allows us to revise our internal mechanisms for retrieving items in the future should we so need, as well as physically move content, without compromising researchers' bookmarks.

2.3.4 Personal Workspaces

A large component of the DSpace project is to do with the submission of content to the archive. Submission is not a simple, one-shot interaction; the Web user interface guides the user through submission via an interactive series of steps. Additionally, a user may start submitting one item, decide to postpone, and in the meantime wish to submit another, separate item. Other submission mechanisms might have similar characteristics. To enable this functionality, DSpace allows each user (“e-person”) to have a “personal workspace” in which incomplete submissions may be stored and worked on.

When the user starts a submission, a fresh item is created in their workspace. Metadata and content files are added to this item until the user considers it complete, whereby they “commit” the submission. In OAIS terms (see section 2.2.3), the user is building up the Submission Information Package (SIP). When the submission process is completed, the system initiates the workflow associated with the collection the user submitted to. If for some reason the submission is rejected from the collection, or requires edits, it is returned to the user’s personal workspace so they may perform the edits without having to restart from scratch.

2.3.5 Workflow

After they are submitted, documents do not normally go directly into the archive. Submissions will typically go through some sort of editorial review where they can be reviewed, rejected, or edited. We call that editorial review process a workflow. Each collection within DSpace can have its own workflow to meet the needs of its community.

Our workflow module is a simple state machine that tracks a submission's state on its trip into the DSpace archive. The workflow engine has a simple API to read and manipulate the state of a submission, and it is capable of triggering events such as email notification of the submission's state to members of review teams and the submitter.

2.3.6 Index & Search

Search is an essential component of discovery in DSpace. Users' expectations from a search engine are quite high, so a goal for DSpace is to supply as many search features as possible.

DSpace's indexing and search module has a very simple API which allows for indexing new content, regenerating the index, and performing searches on the entire corpus, a community, or collection. Behind the API is the Java freeware search engine Lucene[4]. Lucene gives us fielded searching, stopwords, stemming, and the ability to incrementally add new indexed content without regenerating the entire index.

2.3.7 Browse

Another important mechanism for discovery in DSpace is the browse. This is the process whereby the user views a particular index, such as the title index, and navigates around it in search of interesting items. The browse subsystem provides a simple API for achieving this by allowing a caller to specify an index, and a subsection of that index. The browse subsystem then discloses the portion of the index of interest. Indices that may be browsed are item title, item issue date and authors. Additionally, the browse can be limited to items within a particular collection or community.

Currently this is implemented using SQL views and an SQL function for removing leading articles in titles. For example, the title "The DSpace Project" would be indexed under "D" and not "T". In this way, the indices are accessed dynamically. We do not need to periodically or incrementally produce a static index.

2.3.8 People & Groups

Many of DSpace's features such as document discovery and retrieval can be used anonymously, but users must be authenticated to perform functions such as submission, email notification, customized views, or administration. Users are also grouped for easier administration.

DSpace calls users "e-people", to reflect that some users may be machines rather than actual people. E-people authenticate with username/password pairs or X509 certificates. E-people can be members of 'groups' to make administrator's lives easier when manipulating authorization policies.

2.3.9 Authorization & Policies

DSpace has flexible rights management as a stated goal. 'Flexible' refers to the ability to control access to individual digital objects (e.g. communities, collections, items and bitstreams.) Policies could be defined to restrict access to an object based on a user's identity, membership in a group of users, a period of time of having elapsed, or having special permission (such as making a micropayment.)

DSpace's authorization module works from a list of policies, which detail an action, and who is allowed to perform that action. Each object in the system can have its own policy entry, but most will be inherited from the object's container. For example, almost all of the items in a collection will have the same set of policies for who can view them. The 'who' in the policy statement can be individual users, groups of users, or reference to a function that returns a Boolean (such as a function needed to check an elapsed time period or receipt of payment.)

2.3.10 History

The goals of the history subsystem are to capture a time-based record of significant changes in DSpace, in a manner suitable for later refactoring or repurposing, and to provide a corpus of data suitable for research by HP Labs and other interested parties. Note that the history data is not expected to provide current information about the archive; it simply records what has happened in the past.

Currently, the History subsystem is explicitly invoked when significant events occur (e.g., DSpace accepts an item into the archive). The History subsystem then creates RDF data describing the current state of the object. The RDF data is modelled using Harmony/ABC, an ontology for describing temporal-based data, and stored in the filesystem. Some simple indices for unwinding the data are available.

2.3.11 Logging

To facilitate system administration, troubleshooting, and debugging during development, DSpace uses a standard mechanism for logging DSpace activity. Using the Apache log4j logging toolkit, DSpace provides logging output at UNIX-standard "debug, info, and warn" levels:

- "debug" information is verbose and of primary interest during development;
- "info" messages encapsulate a DSpace-standard format for reporting completion of significant DSpace tasks (e.g. "submit item" or "approve item"), and ties actions during a specific user's session together for immediate or retrospective troubleshooting;
- "warn" messages record significant anomalies for immediate or retrospective reporting and analysis.

Our implementation uses standard log4j settings for run-time configuration of reporting levels (for instance, "debug" messages are off by default) and message layout. Our server-side configurations record messages to a regularly rotated file in a standard DSpace directory. These files are readily available for analysis using standard UNIX tools (grep, perl) or console-based log monitoring tools (chainsaw).

Aside from DSpace-specific logging, we also use the standard Apache, PostgreSQL, Resin, Handle System, and HP-UX logging tools for efficient monitoring of those independent server processes.

2.3.12 User Interface

Currently, the only available means for accessing the DSpace system is via the Web user interface. The Web UI allows users to view communities, collections and items, to perform searches and browse indices, and to download and view content. It also allows users to submit content and metadata, and to perform workflow tasks, using a section of the UI known as "My DSpace". The user interface has been through a number of

usability tests, particularly focused on the submission UI, since this is the most complex part of the UI.

The Web UI is implemented using a Java Servlet engine, Resin[5], with support for Java Server Pages. A combination of Servlets and JSPs are used in a model-view-controller style. Servlets receive incoming HTTP requests and handle the processing and business logic, and these forward the request to a suitable JSP for display. This means the JSPs are as close to pure HTML as possible, making customization, internationalization and error handling as simple as possible.

At present, due to tight development schedules, the user interface code currently performs much of the business logic that should be present in the business logic layer. One of the tasks immediately facing the development team is separating out this business logic such that other services can access this business logic via the content management API.

2.3.13 Import/Export

As with any content management system, DSpace has the need to import content, whether sharing content with another system or assuming managment of legacy content. Other systems will also want to interoperate with DSpace.

DSpace's import capability is currently done with an intermediate file format for content, and an importer that can place that content into the DSpace system. The format of legacy content varies widely, so custom front-ends to convert the content and metadata into DSpace's import format are done on a case by case basis. DSpace plans on using OAI (Open Archives Initiative[6]) to expose and share metadata with other systems. Content export can be done with DSpace's import file-format.

2.3.14 DSpace "Public" API

As described above in section 2.2.1, each component in the business logic layer has a “public” API, and the union of these forms the DSpace in-process application interface. However, this is not a truly “public” API – external applications will not be able to use this API directly. This API is “trusted” – it is up to the services in the service layer to ensure users are authenticated. This is because authentication techniques will vary greatly between services, and some services may use an external authentication mechanism, such as a federation service using a global access control list external to DSpace.

Access to the API will be enabled via the implementation of services in the service layer. For example, a SOAP service could allow remote access to and manipulation of DSpace content.

2.3.15 Dissemination

Using the terminology of the OAIS model (section 2.2.3), the act of delivering content and/or metadata to a user is called *dissemination*. The current version of DSpace has a very simple approach to dissemination: Uploaded bitstreams may be downloaded as-is. This works fine in a bounded environment, such as a department all using similar Web-based computing systems, and for a limited amount of time. However, in the long term content needs to be accessible by a wider audience, who may be using a wide variety of computing equipment. Additionally, file interchange formats and standards, and rendering software and hardware change over time. A method is needed whereby a client’s capability for rendering media is assessed, and an appropriate version or rendition of the content is accessed or computed. Referring again to OAIS terminology, we need to define mechanisms for obtaining appropriate Dissemination Information Packages from the Archival Information Package held within DSpace.

A number of methodologies for addressing this are under review, including the FEDORA work[2], the Repository Access Protocol (RAP), other Web Services work, and the work of the Device Independence group in HP Labs.

3. REFERENCES

[1] **CCSDS 650.0-R-2**: *Reference Model for an Open Archival Information System (OAIS)*. Red Book. Issue 2. June 2001 http://www.ccsds.org/documents/pdf/CCSDS-650.0-R-2.pdf

[2] Flexible and Extensible Digital Object and Repository Architecture (FEDORA): http://www.cs.cornell.edu/cdlrg/fedora.html

[3] Corporation for National Research Initiatives. The Handle System. http://www.handle.net/

[4] Jakarta Lucene. http://jakarta.apache.org/lucene/docs/index.html

[5] Caucho Technology. Resin XML Application Server. http://www.caucho.com/

[6] The Open Archives Initiative. http://www.openarchives.org/

METS: Metadata Encoding and Transmission Standard. http://www.loc.gov/standards/mets/