Toward the next generation: Recommendations for the next DSpace Architecture

Written by the DSpace architecture review group
John Mark Ockerbloom, chair
January 24, 2007

Introduction

Since its initial public release in 2002, DSpace has become a widely used and trusted digital content repository system, with nearly 200 DSpace sites registered on the DSpace wiki by the end of 2006, and many more unregistered sites. Designed to work "out of the box" for basic repository needs while still being customizable, and to manage and preserve content over long time periods, the system has been put to a wide variety of uses, and has been entrusted with important intellectual content produced by many institutions. As DSpace's adoption spreads, it needs to evolve to better support the diversity of applications and repositories that are built using it, and to more effectively preserve content managed by the software.

While a certain amount of evolution can take place simply by patches, contributions, and reimplementation of specific components, it is also necessary to periodically review the basic architecture of the whole system to ensure that it continues to meet the needs of its clients as their own uses and best practices evolve. Recorded discussion of "DSpace 2.0" began in 2004, with proposals by Robert Tansley and others. More recently, in 2006 a group of thirteen DSpace committers, technical experts, and other interested parties, was convened to review the DSpace architecture as a whole and make recommendations for the architecture of the next major release. This is the report of the group.

Procedure and scope of the review

The Architecture Review Group was organized in the summer of 2006. Its members held discussions online, and to a large extent, in public forums such as the DSpace-devel mailing list and the DSpace wiki. Initial work of the group included drafting a set of design issues and working principles for DSpace development. We wrote and conducted a survey of the DSpace maintainer and developer community. We then met face to face for a week in late October in Cambridge, Massachusetts, to agree on our basic recommendations for the next DSpace architecture, and to plan next steps. These next steps would include organizing two subgroups to assess and recommend open source frameworks and workflow systems. In some cases, we prepared straw-man proposals or specifications for consideration by the group. In the course of our discussions, we came up with a set of **recommendations** for the next generation of DSpace that represent the consensus of the group on a set of issues ranging from data model revisions to event system changes. We made our recommendations with the expectation that a new architecture based on them could be implemented within 18 to 24 months with a small

team, and that the basic architecture could have a useful lifespan at least as long as that of the current, original DSpace architecture (which will have lasted about 7 years by the time we expect DSpace 2 to come out.)

We have not produced detailed specifications for most of the recommendations we give. Given the composition and the time constraints of the group, it seemed best to spend our efforts on forging a consensus about general directions and structures for DSpace 2 that would best serve and be supported by the larger DSpace community. In a few cases, we have put forward more specific **proposals** that may be useful for DSpace implementers to follow, but that we did not as a group consider to be requirements of the new architecture.

Some of the architectural elements needed in the next generation of DSpace have been actively developed in the larger open source community, in particular implementation frameworks and workflow managers. We formed subgroups to study open source efforts in these areas, to survey ongoing work in these areas, review the advantages and disadvantages of particular systems in these areas, and recommend systems that could be adopted for DSpace 2.

Our expectation is that a small core group of implementers would produce detailed specifications and implementation of the core and standard distribution of DSpace 2, based on the recommendations of this group. A follow-on architecture oversight committee would review and check the detailed designs and specifications produced by the implementers, using these recommendations as a basic roadmap.

We have also made a few recommendations for releases of DSpace 1. While these recommendations are not strictly within the scope of this group's assignment, we hope that they will help the DSpace community prepare for and transition to the new architecture more smoothly.

Manifesto

As one of our first activities, we ratified a set of principles that any new DSpace architecture should uphold. These are as follows:

1. DSpace is primarily open source software for building digital repositories.

DSpace is intended to be free and open source software for digital repositories that enables services for access, provision, stewardship and re-use of digital assets with a focus on educational and research materials; in short, to fulfill the mission of the DSpace Federation.

2. DSpace will be usable based purely on free and open source software.

Although setups including custom and/or proprietary features and technologies will be possible, it will always be possible to deploy DSpace using only free and open source software.

3. DSpace will have a decoupled, stable, and application-neutral core.

DSpace will always have a "core" system that supports a variety of higher-level applications, whose full scope is not bounded unnecessarily. It will define stable APIs to enable diverse and innovative applications and functionality built on this core, without need to modify the source code of the core.

Exactly what constitutes the "core" was the subject of much discussion in the group. In this document, we intend the "core" to mean the central modules (mostly in the business logic layer), and the data model for the information they manage, on which DSpace applications and extensions depend. By this definition, the "core" does not by itself constitute a full DSpace installation. A full installation is instead provided by the "standard distribution", which consists of the core plus applications and extensions that are part of official DSpace releases and support out-of-the-box functionality. The standard distribution is the subject of the next manifesto statement.

4. While usable for a variety of applications, DSpace will retain useful "out-of-the-box" functionality for common use cases.

DSpace cannot support all the variable and emerging definitions and innovations in the repository space in a single interface application. DSpace will provide out-of-the-box functionality for a common set of use cases (e.g. an open access preprints application, a general content archive) that can be installed with minimum possible effort. It will also provide modular support for the easy construction of new applications.

5. DSpace will employ and support existing, open standards where possible and practical.

Open and established standards (when available) will be employed to support applicable DSpace functions, including interoperability of various kinds with other systems, and the migration of data into and out of other systems. DSpace architecture and implementation will attempt to take advantage of external development wherever possible. It will also, however, consider the maturity of particular standards before implementing them.

6. DSpace releases should be minimally disruptive.

The architecture should reinforce good behavior in making changes, customizations, and improvements to future releases of the system, so that upgrades are minimally disruptive for current adopters.

7. DSpace will support an exit strategy for content.

It will be possible to export all data necessary for the future re-use and stewardship of content held in a DSpace repository, in open, well-documented formats, for enabling migration into other systems and/or backup.

8. DSpace will continue to evolve.

There are many unsolved problems associated with stewardship of digital materials, which will require research and experimentation (including some failures) to solve. In addition to providing a robust, stable and functional system, DSpace will enable innovation and experimentation, and will be designed with the knowledge that future development and re-architecting will inevitably be necessary.

Survey

We also prepared a survey of DSpace maintainers, in the form of a questionnaire that was made available online for about a week, and advertised on several DSpace mailing lists. The questionnaire drew 116 responses. Though the responding population was self-selected, it was still usefully large (particularly in comparison to the number of registered DSpace sites) and we expect that the sample included many of the most involved participants in the DSpace community. The survey included both multiple choice questions and opportunities for user comment.

Full summary results of the survey, including anonymized comments, can be found on the DSpace Wiki at

http://wiki.dspace.org/index.php/PreReviewSurvey

Although the survey was open for a relatively short period of time, and its open nature meant that some organizations may have submitted more than one result affecting the weighting, some interesting results can be seen.

While the majority of the DSpace users surveyed use it as a university "institutional repository", there are significantly many other uses of DSpace, including uses by government organizations, corporations and museums. Only around half the respondents describe their use as "production", and at least 1/3 of respondents intend to store non-textual content (e.g. image, audio, video). Most users have some kind of custom Dublin Core metadata, with a much smaller number having XML or RDF metadata stored in DSpace. Surprisingly, nearly a quarter of respondents have said they found it necessary to modify the database schema in some way.

Most users stay somewhat up to date with DSpace (with the vast majority using 1.3.x or 1.4), the main stumbling block being merging local customizations with new versions. Around half of respondents have made modifications they describe as 'significant' (as opposed to minor cosmetic changes and configuration). In general the documentation is regarded as adequate but needs to be updated faster, and the version that particular documentation describes needs to be clearly labeled.

Use of third-party "add-ons" remains very low. Only 60% use the CNRI Handle system and are happy with it. The most demanded new features are modularity, more easily customizable user interface, support for complex objects and versioning.

While installation causes few problems, documentation and ease of upgrading and customization need work. More encouraging, the current data model is useful for most people (70% say "good" or "very good"), and the community seems to feel positively about collaboration/communication tools and their ability to get involved (only a couple of respondents rating "poor" or "very poor" on either count).

Recommendations

GENERAL CONCERNS

Before we discuss specific recommendations, we generally address issues of scalability and interoperability, two issues that cut across the architecture and that are of widespread concern in the DSpace community.

Scalability

Concerns about the scalability of DSpace tend to fall into three general areas. The first is capacity, or how well DSpace repositories can handle large quantities of data. The second is ingest rate, or more generally, throughput: how fast DSpace can ingest (or export) content from (or to) elsewhere. The third is concurrency, or, how well DSpace performs under heavy simultaneous accesses to the repository.

Maintainers of large-scale, heavily used DSpace repositories have reported issues, particularly related to performance, in some aspects of DSpace at large scale. While the implementation of some of DSpace's components and database tables may need to be revised to perform satisfactorily under heavy load or high usage, we do not believe that the scaling issues we have encountered represent fundamental problems of the architecture. Still, it is useful to set benchmarks about what scales we should expect a second-generation DSpace architecture to handle, and what scales might be better served by other systems. We therefore make the following general recommendations:

> **Recommendation**: DSpace's design and implementation should accommodate at least 10 million items (assuming adequate underlying storage capacity).
>
> **Recommendation**: DSpace should not impose any limit of its own on the size of content files (though the underlying storage, or third-party software used in the standard DSpace distribution, might).
>
> **Recommendation**: DSpace should not require more than one second (not including transfer time and any requested content processing or filtering) per item to ingest items in a 10 million item repository.

Recommendation: DSpace should be able to handle 10 concurrent updating users, and 100 concurrent reading users, given adequate bandwidth and reasonable CPU and memory capacity.

Recommendation: DSpace's design should continue to accommodate load-balanced clusters of servers managing a single repository. (Some larger DSpace sites now use that implementation strategy.)

We hope that some of the specific architectural changes we recommend below, such as the event system, will further aid in DSpace's scalability.

Interoperability

As with scalability, interoperability can mean different things to different people. One important sense is data interoperability, allowing data to be shared between DSpace and other systems, and migrated between them when appropriate. Another is service interoperability, in which DSpace services can be requested and performed on behalf of external systems, whether other repositories or applications build on top of DSpace. A related, but more intimate form of interoperability important to many DSpace adopters and customizers, is being able to modify the functionality of DSpace itself through connections to well-defined APIs, preferably without having to modify the "core" DSpace modules. We therefore make the following general recommendations:

Recommendation: DSpace should have a published concrete data model (for both content and metadata) that can be fully imported and exported.

Recommendation: DSpace should have a published, documented, and stable core interface capable of supporting applications that can exploit the full range of DSpace data.

Recommendation: DSpace should have a standard distribution that includes applications for widely used protocols relevant to repository interoperation.

Having addressed basic architectural principles, we turn to specific architectural issues.

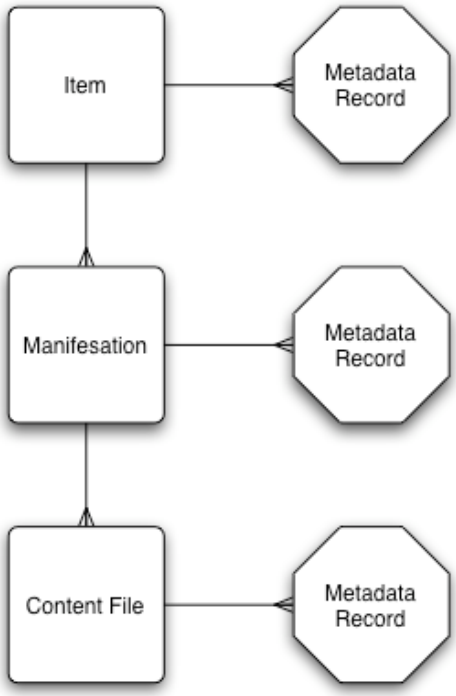
INFORMATION MODEL

As noted in the manifesto and by our community, DSpace is fundamentally focused on the data it manages, both content and metadata. Any general review of the architecture, then, must carefully consider the data model necessary to support the range of uses required by the DSpace community. In particular, the repositories of community members are using a wider range of metadata (and for a variety of uses), are relying more heavily on persistent identifiers, are often concerned with versioning issues, and may need more complex and flexible repository organization. Any revised data model that better addresses these needs, however, must also support a smooth transition from DSpace's existing data model.

Item structure

> **Recommendation**: DSpace's Item model should be revised to clarify the roles, and improve the documentation and metadata, of components of an item, as described below.

In the new abstract data model we recommend, Items can contain multiple Manifestations (similar to the notion of Manifestation in the FRBR model), and each Manifestation can contain multiple Content files. Items, Manifestations, and Content files all have their own persistent identifiers, and can all have their own metadata records (possibly multiple metadata records).



Manifestations replace Bundles from the DSpace 1.x model, and represent FRBR manifestations (unlike Bundles in DSpace 1.x, which could contain metadata such as licensing information).

Content Files replace Bitstreams from the DSpace 1.x model. The renaming is partly for clarity (the term “bitstream” is specifically used to mean something different in other quarters, e.g. in PREMIS), and partly because Bitstreams in 1.x could also be metadata (e.g. licensing, METS manifests). In the new model, Content Files are just content, that is, the constituents of Manifestations.

Metadata records are not constrained to follow a built-in scheme, and are not constrained to be flat. Note that the abstract model does not speak to how metadata is stored – i.e. whether it is an XML file or a node/property graph (e.g. JSR-170 or RDF). For example, metadata records for multiple entities in this abstract model may be stored in a single

XML file such as a METS file. In this case, the METS file is a serialization of the abstract data model, and does not exist within the abstract data model.

These adjustments to the Item and sub-Item model allow more precise addressing and handling of DSpace content, and more flexible metadata more clearly applied to particular parts of an item. Some DSpace 1.x special purpose Bundles, such as licenses, would become metadata; others would become Manifestations.

Identifiers

> **Recommendation**: The DSpace architecture should accommodate persistent identifiers not based on Handles.
>
> **Recommendation**: The recommendation above notwithstanding, Handles should remain the default persistent identifier standard in the DSpace standard distribution.

DSpace has used CNRI's Handles as the basis of its identifier scheme since its initial release, and the Handle system remains robust, flexible, and low in overhead. Nonetheless, for various reasons, whether technical concerns or strategic priorities, some DSpace installations may prefer to use different identifier schemes. We therefore recommend that DSpace continue to use content identifiers based on Handles by default, but be designed to allow other identifier schemes to be implemented at minimal cost. DSpace's architecture should therefore not depend fundamentally on the Handle system, although identifier schemes that can be abstractly modeled as the triple *(scheme, namespace, scoped-label)*, which describes both Handles and a number of other persistent identifier schemes, may still be the type preferred by DSpace.

We also propose that these persistent identifiers should be used for Communities, Collections, any other "containers" and Items. Within Items, it is proposed that each sub-entity has an ID hierarchically related to the Item ID. For example, if the ID of an Item is `hdl:123.456/4`, the identifiers of objects within might be:

`info:dspace/site/123.456/item/123.456/4/metadata1` (Item-level metadata record)
`info:dspace/site/123.456/item/123.456/4/manifestation1` (Manifestation)
`info:dspace/site/123.456/item/123.456/4/manifestation1/metadata1` (Manifestation metadata record)
`info:dspace/site/123.456/item/123.456/4/manifestation1/contentfile1` (Content File)
`info:dspace/site/123.456/item/123.456/4/manifestation1/contentfile2` (Content File)
`info:dspace/site/123.456/item/123.456/4/manifestation1/contentfile2/metadata1` (Content File metadata)
`info:dspace/site/123.456/item/123.456/4/manifestation2` (Alternative manifestation)

Note these IDs are for illustrative purposes and not a specific recommendation for a precise ID scheme.

This identifier scheme facilitates the versioning model described later in this report. Also, the hierarchical nature of the identifiers enables "fall back" if a particular Content File or Manifestation is no longer available for some reason. Since the Item ID is in the identifier, it will be trivial in such cases to determine the originating Item and retrieve alternative Manifestations.

While applications can store arbitrary metadata with Items, Manifestations and Content Files, there will be a standard metadata set that DSpace will require for each – for example for size, format and checksum of Content Files, or Dublin Core title for Items. In some cases, this metadata might be automatically derived from supplied metadata, rather than having to be provided explicitly.

> **Recommendation**: EPeople should have persistent URIs

As DSpace repositories grow in size and depth of coverage and audience, the need to unambiguously identify people in metadata and permissions is increasingly important. Email addresses can be short lived, and personal names on their own can be ambiguous. We recommend therefore giving persistent identifiers in the form of URIs to EPeople. Such URIs do not necessarily have to be resolvable, but the URI form supports resolution if desired, and also promotes compatibility with RDF-based data stores. The group as a whole did not prescribe a particular URI schema to use, but info: or Handle schemes could be possibilities.

The architecture review group discussed the desirability of global authority control for named persons and groups in institutional repository systems. Such systems are beyond the scope of DSpace to define on its own, and have historically been challenging to coordinate. If a global name identifier scheme later becomes widespread, though, we hope that the URI-based identifiers for EPeople could easily include or adapt to it.

Versioning

> **Recommendation**: DSpace's data model should include native support for item versions.

We heard requests from a number of users for versioning support, as is offered in some other repository systems. Versioning, in which multiple revisions of items are stored in a repository, will become increasingly important as DSpace repositories grow older and content is migrated to new formats and technologies. Versioning in our model covers linear revisions, not alternative representations. Hence, video, audio and transcribed-text "versions" of a particular lecture, say, could be deposited as different Manifestations within an Item, or different Items related to one another, but would not be considered two Versions of the same Item.

Versions can be used to support not only migrations, but also corrections and technical modifications of essentially equivalent semantic content. In some systems, versions are also used for semantically different content, such as pre-refereed and published versions of a paper. While the system we recommend would not technically prevent such versions from being created, we recommend against using versions for this type of content management, but instead to create a new item for this type of version and provide relationship links between the items. Among other things, following this practice would make it safer to consistently use identifiers to cite a particular expression of intellectual

content without tying it down to a specific technological instantiation of that content. In order to make this practice realistic, DSpace applications should give robust support to item relationship metadata and make it easier to create, view, and follow.

We further recommend that DSpace include identifiers for specific versions of items, and for manifestations and content files within those versions. Identifiers without a version designation should refer to the latest version of an item. Each version would have its own metadata (which would usually be very similar from version to version).

For versions of a particular Item (for example, a correction or format migration), we recommend a simple, linear approach. A Version of an Item is a "snapshot" of the Item at a particular point in time, including the state of the metadata, and all Manifestations and Content Files. One version of the Item is the most recent.

An Item may be changed, such as by changing metadata, adding new Manifestations, and changing Content File. When these changes are complete a new Version is created. This new Version becomes the most recent revision of the Item, although previous revisions may be retained. We expect that previous Versions and their data would indeed be retained over time in most cases, but that individual DSpace repositories could decide on alternative policies to retain or purge old Versions.

Each Version gets a "revision identifier". This could be as simple as an incremental sequence number. A particular Version of an Item, Manifestation, Content File or metadata record at any level will then have a unique identifier.

Our earlier identifier proposal can be easily extended to handle Versions. For example, consider the case where item `123.456/4` is created and has two Content Files, as shown in the "Identifiers" section above. The most current revision of the second Content File will always be:

```
info:dspace/site/123.456/item/123.456/4/manifestation1/contentfile2
```

The first Version of that content file will *always* have the identifier:

```
info:dspace/site/123.456/item/123.456/4/r1/manifestation1/contentfile2
```

Note the `r1` path component of the identifier. This identifier will *always* point to the same exact Version of the Content File, even if that Content File is changed in later Versions.

Now suppose that the Content File has been changed, perhaps along with some metadata changes, and a new revision of the Item has been created.

`info:dspace/site/123.456/item/123.456/4/manifestation1/contentfile2` will now point to the updated Version's corresponding Content File.

`info:dspace/site/123.456/item/123.456/4/r1/manifestation1/contentfile2` will continue to point to the original version Content File.

`info:dspace/site/123.456/item/123.456/4/r2/manifestation1/contentfile2` will always now point to this new revision of the file (notice the `r2`), even if the Content File is again changed in later Versions.

In other words, accessing

`info:dspace/site/123.456/item/123.456/4/XXX/YYY`

will always identify the *latest* revision of a particular Item or constituent Manifestation, Content File or Metadata Record within.

`info:dspace/site/123.456/item/123.456/4/`**`rX`**`/XXX/YYY`

will always identify a *specific* revision of a particular Item or constituent Manifestation, Content File or Metadata Record within.

Of course, components within an Item may be added or deleted between Versions, so it will be very possible for e.g.

`info:dspace/site/123.456/item/123.456/4/`**`rX`**`/manifestation3`

to exist but

`info:dspace/site/123.456/item/123.456/4/manifestation3`

to be non-existent, and vice versa.

Item aggregation

> **Recommendation**: DSpace's should generalize its aggregation model to better exploit commonalities between Communities and Collections.

The group noted that Communities and Collections are very similar. They have slightly different metadata in the current implementation, and currently Collections can only contain Items, while Communities cannot contain Items. In many ways, however, both Collections and Communities can be modeled by a more general Container concept. A fully general Container could potentially include both Containers and Items, as is possible in some other repositories (and by analogy in filesystems). However, we did not observe significant demand for this functionality from the present-day DSpace community, and were not sure whether removing the constraints on current DSpace container concepts would break existing applications in ways that would be difficult to address.

We therefore have not recommended replacing Collections and Communities with a single Container concept at this time. However, it would still be useful for the DSpace implementation to introduce a virtual Container interface that captures as much as can be

reasonably generalized from the Collection and Community concepts. If a merger of the concepts seems called for in the future DSpace core, or in customized DSpace installations, this construct would ease the transition. It may also be useful to support Collections transforming into Communities or vice versa, or to a less constrained Container class if that becomes desirable in the future.

Concrete data model

The abstract data model described above must be represented in DSpace storage somehow. We refer to this representation as the 'concrete data model'. In DSpace 1.x, this representation was a bespoke relational database schema. Parts that did not comfortably fit into the schema were stored in specially named Bundles and Bitstreams. Our new model, in contrast, has a cleaner separation of concerns.

> **Recommendation**: DSpace's "concrete" data model should be considered separately from the abstract data model described above.

Thus, metadata may be stored in a file (say, XML) somewhere, but this does not make it a Content File; it is still a Metadata Record in the abstract data model.

This approach also allows more flexibility in creating storage systems for DSpace. APIs can operate on the abstract data model, while masking the specific storage mechanisms and concrete data model.

Metadata plays an increasingly important role in the revised information architecture described above for DSpace, and it appears in more places in the information model. It therefore becomes increasingly important, especially as preservation activities become more significant in an ongoing DSpace implementation, to manage the metadata in a robust, sustainable manner.

> **Recommendation:** Metadata should be maintained in the persistent store, and extractable in serializable form.

It is insufficient to have metadata maintained in transient database tables with predefined table fields for specified metadata attributes, if we want to make it easy to preserve metadata and to support a wide range of metadata schemas.

The system will need to maintain default metadata schemas for Items, Manifestations, and Content files, but should allow other metadata to be added, and should attempt to accommodate changes in the default schemas, as DSpace 1.4 allows today.

Efficient table-based representation of metadata is important for efficiency of some DSpace functions. Therefore, the system may need to maintain relational views of the canonical metadata that can be efficiently and intelligibly accessed by the user interface and other applications. These views do not necessarily need to include all metadata maintained by the system, however. For instance, search and browse functionality may

only need certain descriptive metadata fields to function. Views that act as crosswalks for parts of DSpace's metadata to basic Dublin Core may be sufficient for functions like basic OAI export.

We do not prescribe a particular strategy for maintaining views of metadata in the general case. However, we propose that the Event mechanism described later in this report may prove a useful mechanism to maintain consistency between DSpace metadata and content, and its views. It may also be useful to have such mechanisms capable of generating specialized views for DSpace extensions and applications.

In physical storage, we propose that a METS profile be developed that allows the DSpace abstract data model and default schemas to be serialized in a standard way.

EXTENSION FRAMEWORK

> **Recommendation:** DSpace should adopt a general framework for adding extensions to its core.
>
> **Recommendation:** DSpace should adopt an existing open source extension framework, rather than build a new one from scratch.

The DSpace community has been developing an increasing variety of extensions, customizations, and applications of the basic DSpace platform. This extended development effort of the community has done much to make DSpace widely useful and relevant for its users, and to help it interoperate with other systems. To promote further development and extension as the DSpace core evolves, integrating new extensions to the DSpace core needs to be easier. Implementation frameworks have been developed inside and outside the open source community to ease such integration. Some of these frameworks have been developed ad-hoc for a particular need, such as the simple framework now used to integrate Manakin with DSpace. Others, however, are more general purpose. While general-purpose frameworks may involve more complexity than a simple, made-to-order integration framework, they also can make it easier to integrate a variety of extensions to DSpace that are not limited to current applications, but that can be adapted as new needs and ideas arise.

We therefore recommend that DSpace 2 adopt a general-purpose open source extension framework. Adopting a pre-existing framework, furthermore, avoids unnecessary effort in developing a new framework, and also gives DSpace 2 developers ideas and possibly working examples of how the DSpace core can interoperate with other modules and applications using the framework.

A sub-group of the architecture review group has been commissioned to review existing open source extension frameworks, and recommend a system for DSpace 2 to adopt, and to describe benefits and drawback of this system and other potentially promising candidate extension frameworks. Before this sub-group began its work, the architectural

review group as a whole came up with a set of requirements and desired features for the framework. These include the following:

- Fine-grained support for dependencies between various versions of core and extension system components, to ensure compatibility between components
- Support for separate update of different components
- Compatibility with the DSpace open source license
- Access to DSpace persistent storage, including extension-specific persistent storage
- Compatibility with distributed environments, such as are used in some DSpace systems
- Java support
- Ability to reconfigure a system without recompilation. (Run-time reconfiguration of a running system would also be useful if supported, but this is not a requirement)
- Support of relevant and applicable open standards
- Ease of use and configuration. (It should be easy to integrate with Manakin, for instance.)
- It would also be desirable if the framework is already being used or considered in related library, archiving, or higher education applications

Converting DSpace core modules to a framework structure would require re-engineering of their interfaces. The form of the resulting interfaces work might be quite different depending on the framework choice, since different frameworks use different integration paradigms, such as Inversion of Control (IOC) patterns, or registration systems. Evaluation of frameworks should carefully consider the conversion costs of moving to a particular implementation framework, and who would bear those costs.

At present, the subgroup is not ready to make a definite recommendation, but a Wiki discussion of possible candidate frameworks, currently focusing primarily on OSGi and Spring as possible candidates, is underway at

http://wiki.dspace.org/index.php/ArchReviewFrameworks

We hope that further discussion and analysis, with the involvement of the wider DSpace developer community, will lead to a specific recommendation of framework choice.

While DSpace lacks a general extension framework, simple add-on mechanisms can be used to integrate third-party components. Scott Phillips at Texas A&M has recently released a simple add-on mechanism for integrating Manakin and other packages with DSpace at build time. His prototype system can be downloaded from

http://di.tamu.edu/~scott/

USER INTERFACE

Recommendation: DSpace 2 should shift away from the current JSP-based user interface in favor of systems that allow more decoupled composition of business logic and user interface, such as Manakin.

Manakin is an alternative user interface to the current JSP user interface. It was developed at Texas A&M University, and is based on XML translated into web pages via stylesheet transformations. Although Manakin is not yet part of the standard interface, and requires a special mechanism to make it interface with standard DSpace code, it has attracted enough interest to be used in several DSpace sites outside Texas A&M. It can be used to create a wider variety of user interfaces than the JSP model supports, and interfaces that can be much more loosely coupled with the underlying DSpace business logic. Manakin is not yet part of the official DSpace release, but there is significant interest in integrating it, or a similar XML-based user interface, with DSpace.

Integrating Manakin into DSpace requires the use of an extension framework. While we have recommended adopting a general-purpose extension framework above, it will not be ready for use immediately. However, Manakin is already showing increasing use and interest, even with its current ad-hoc add-on mechanism. We recommend that developers of future DSpace 1 releases include a lightweight add-on mechanism capable of integrating Manakin into DSpace 1, and provide out of the box support for Manakin in future DSpace releases, until DSpace 2 is ready. Then DSpace 2 could integrate Manakin via its general purpose extension framework.

EVENT MECHANISM

Recommendation: DSpace 2 should include an event notification system as a central component to its core.

Event notification systems have long been used to provide loose integration of components in complex systems. System components register as listeners to respond to different kinds of events raised by the core, which include reports of changes to content and metadata as well as other significant phenomena. When an event occurs, applicable listeners are activated and provided a copy of the event data. (This pattern is sometimes known as a publish-subscribe model.) For example, DSpace's history mechanism would listen for significant events involving changes to data, and record the event details in history logs. Event mechanisms could also be used to keep database views of repository metadata consistent, and to coordinate activities between the data store and the user interface, or other applications and third party components.

Events can be implemented in a variety of ways, and some component integration frameworks and workflow systems include their own event model. Work is underway at MIT, led by Larry Stone and others, on a prototype event model for DSpace. We recommend that development of this model continue, so that developers get experience with the types of events, and publish-subscribe patterns, that will be most useful in the DSpace context. We then recommend, to avoid unnecessary complexity and duplication

of function, that the implementation of the event mechanism be transitioned to the framework provided in DSpace 2, if that framework provides a suitable event implementation.

WORKFLOW

> **Recommendation:** DSpace should adopt a generic workflow system to manage all phases of the information lifecycle.
>
> **Recommendation:** DSpace should adopt an existing open source workflow framework.
>
> **Recommendation:** DSpace should support user interfaces to make it easy to adapt the generic workflow interface for specific workflow needs.

The current version of DSpace includes a workflow manager for ingest. While this manager has worked reasonably well for bringing content into DSpace, large-scale repositories will need to be able to accommodate a wider variety of workflows, not just in the ingest phase of information management, but also for later phases, including migration, versioning, and export.

Just as there has been much work in the open source community on general-purpose extension integration frameworks, so too has there been notable work on general-purpose workflow managers. For reasons similar to those given in our extension framework recommendations, we recommend that DSpace adopt an existing third-party system to manage its workflows.

Workflow requirements are likely to vary significantly between different DSpace repositories, and may change over time at any given location. Therefore, it is particularly important that DSpace repository maintainers be able to configure, review, and reconfigure the workflows for their repositories without needing to dive deeply into the internal interfaces (or worse yet, implementation code) of DSpace. DSpace's standard distribution, then, should support user interfaces for viewing and configuration of a repository's workflow. This user interface does not need to be provided by the workflow manager itself, but if the manager provides suitably documented APIs and/or XML views, then Manakin or other applications could provide appropriate user interfaces on top of them.

As we did with extension frameworks, the architecture review group drew up a set of requirements and desired features for a workflow manager, and then formed a sub-group to review and recommend possible candidate systems. Our requirements and desirable features include the following:

- Ability to support the full information curation lifecycle (from ingest through deaccession)
- Compatibility with the DSpace open source license.
- Support for changing workflows at run time

- Ability to specify different workflows for different communities and collections
- Compatibility with distributed environments
- Ability to receive and react to DSpace Events
- Compatibility with DSpace's new modularity/extension framework
- A well-defined API that will support user interfaces based on it. (Depending on the circumstances, users might use the workflow manager's own user interface if provided, or a user interface designed specially for DSpace built on top of the workflow manager's API.)

The subgroup conducted a survey of candidate workflow engines, and prepared a descriptive table available online at

http://wiki.dspace.org/index.php/ArchReviewWorkflowEngines

Although the subgroup did not produce a single endorsement, it did find that Open WFE, and OpenSymphony's OSWorkflow, and JBoss jBPM appeared to be the three most promising systems, meriting further evaluation. These Java-based products have a strong API for back-end use, instead of depending on an all-in-one vertical interface (though there are also graphical front ends for tools like jBPM), and they show promising flexibility and expressiveness. They also have open source licenses compatible with DSpace. They are therefore likely to be easier than other workflow systems to integrate into DSpace to support its workflow needs. Implementation experimentation with the systems' interfaces is needed to determine which system better meets the needs of DSpace 2. It is also worth considering which workflow systems are in use, or planning to be used, by related systems.

CORE INTERFACE REVIEW AND DOCUMENTATION

> **Recommendation:** DSpace 2 should include a fully documented core API as part of its initial release.

Our manifesto calls for a stable core on which a variety of applications can be built. For this core to meet our requirements, it is essential that full documentation be provided for the entire core interface. This includes, among other things, public object APIs, the events raised by the core (and the conditions that raise them) and the data model. While such documentation is usually seen as desirable in systems as complex as DSpace, in practice it is often left as an afterthought. For DSpace to satisfy the terms of our manifesto, however, documentation must be an integral deliverable of the DSpace 2 releases, for this documentation provides the working contract between the DSpace core and its applications and extensions. Having this contract explicit and well-understood promotes the stability of the DSpace core, ensures that the interfaces are carefully reviewed (with the opportunity to find undesirable application dependencies or design limitations) and makes it easier for third parties to develop a wide variety of decoupled applications, in the knowledge that they can rely on the promises of the DSpace core documentation.

In order to fulfill this recommendation, implementation plans for DSpace 2 need to include sufficient time and resources to produce and maintain this documentation. We do not expect that these interfaces will be designed from scratch. Some interfaces, and implementations, will be carried over from DSpace 1, with minimal changes. Others will need to be revised or reworked to fit the new data model and implementation architecture of DSpace 2. The choice of integration framework and workflow manager may affect the exact composition of the interface as well, and may provide APIs that can be used "out of the box" for certain aspects of the system.

The road to DSpace 2

Designing robust software architecture requires a careful balancing of desirability and feasibility. What one might imagine as the perfect system may not be attainable; indeed, for any system that has the community the size of DSpace, there is no one system that would be perfect for all of its users. At the same time, limiting our ambitions to incremental and easily-implemented architectural changes would mean missing important opportunities to improve the stability, capabilities, and extensibility of the system, and to keep it useful and relevant as repository needs and applications evolve. We have attempted to chart a sensible middle course. Our proposed architecture includes some significant changes from DSpace 1, particularly to the data model and the integration framework, but changes that are realistic to implement within a two-year time period given appropriate support, and a feasible migration path from DSpace 1 to DSpace 2.

Developing DSpace 2 will require significant resources and community support. While this group has outlined the general architectural directions and redesigns that we recommend for DSpace 2, we have not written detailed interface specifications. These would need to be written by a core developer group. While the DSpace 2 architecture does not require a complete reimplementation, many existing modules may need to be rewritten or refitted for the new architecture, and some other modules (such as search and browse) may need to be rewritten not for architectural reasons, but to satisfy scalability and performance requirements of DSpace 2 users. Along with detailed specification and reimplementation work, interface documentation also is an essential task, as we have described above.

The work of specifying, implementing, and documenting the core of DSpace 2, based on our recommendations and the existing code and support base for DSpace 1, could probably be completed by three suitably-skilled developers over a two-year time period. This projection assumes that they would not be working in a vacuum, but would be adopting third party extension frameworks and workflow managers, publishing detailed specifications of data structures and APIs, and reworking or reimplementing core modules as needed to fit the new architecture. They would not be responsible for the entire DSpace standard distribution, but would supply detailed enough specifications and documentation such that other members of the DSpace community could implement important non-core components (such as the Manakin user interface), write extensions that fit the integration framework, or provide new implementations of particular modules under the core APIs to meet community needs (such as better performance for user

discovery, or distributed storage mechanisms). These contributions from the wider DSpace community will be crucial to the success of DSpace 2.

While detailed designs are often best made by small groups rather than large committees, larger groups can play a useful role in vetting designs. To ensure that DSpace 2 design and development addresses the needs of the DSpace community and can be completed in a timely fashion, we recommend that a follow-on architecture oversight committee, consisting of selected DSpace committers and other appropriate experts, should review and check the detailed designs and specifications produced by the core DSpace 2 implementers based on the recommendations of this group.

While DSpace 2 is under development, much work can be done in DSpace 1 to prepare for the new architecture. We have recommended, for instance, that Manakin become part of the standard distribution, and be integrated with DSpace 1 using a simple extension mechanism, until the more general extension mechanism for DSpace 2 is ready. We have also mentioned that a prototype event mechanism has been implemented for DSpace, and could be tested and exercised prior to the rollout of DSpace 2. There is also already work in progress, and partly realized in DSpace 1.4, to support a wider variety of metadata schemes, which we hope will pave the way for fully generalizable metadata schemes, and mappings from them to simplified views, in the DSpace 2 architecture.

If resources are scarce and ambitions need to be cut short, a smaller core group could implement a subset of our recommendations. If implementers need to choose among recommendations, support for a more robust data model, along the lines we have advocated above, may well be the most important thing to establish and get right in the new architecture. Applications come and go, and get update or replaced by new ones; the needs and uses of repositories will evolve over time. But the information stored in a repository may need to last decades or even centuries, far longer than any given computer technology will last. It will be much easier to maintain and preserve this information if it can fully expressed in a well-supported data model than if it can only be partially supported in a limited data model, and imperfectly transitioned to the next system. Our manifesto goals of supporting a wide variety of applications, to support an exit strategy for content, and to allow DSpace to continue to evolve, require the best data model we can manage.

As we have indicated above, the success of DSpace 2 will depend heavily on the support of the community as well as the central core developers and their immediate oversight. DSpace 2 will need in-kind support, such as development of new modules and extensions for the standard distribution, as well as financial support to fund the core developers. The means for garnering the necessary support for DSpace 2 is not in the scope of this group, but we are hopeful that the demonstrated success of DSpace as the basis of hundreds of repositories in hundreds of institutions will make it possible for appropriate support to be found from the DSpace community and other interested funders.

As the knowledge base of the world's institutions of research and learning become increasingly digitally based, it is more important than ever that these institutions are able

to take control of their own digital destiny, without being forced to rely on the whims and fortunes of commercial software developers and outside publishers. They need to be able to manage their digital endowments without large expenditures for local repository development and customization, while still being able to extend, adapt, and build on their repositories to meet the most pressing needs of their local communities. DSpace, and the community that has grown up around it, has already done much to meet the repository needs of these institutions. We hope that investment in the second major version of DSpace, along the lines we have described in this document, will deliver similarly large returns in years to come.

Participants in the architectural review group

The architectural review group had the following members:

John Mark Ockerbloom, University of Pennsylvania, chair
Tim DiLauro, Johns Hopkins
Mark Diggory, MIT
John Erickson, Hewlett Packard
Jim Downing, Cambridge University
Henry Jerez, CNRI
Richard Jones, Imperial College
Gabriela Mircea, University of Toronto
Scott Phillips, Texas A&M University
Richard Rodgers, MIT
Mackenzie Smith, MIT
Robert Tansley, Google
Graham Triggs, Biomed Central

Larry Stone also participated in part of the meeting at MIT, presenting some of his DSpace event system work and taking part in follow-up discussion. Numerous other people gave useful suggestions in the online discussions, and in they survey we conducted. We are thankful for the contributions and support of the DSpace community.