ARTICLES

D-Lib Magazine
April 2003

Volume 9 Number 4

ISSN 1082-9873

The Fedora Project

An Open-source Digital Object Repository Management System

Thornton Staples
Digital Library Research and Development
University of Virginia
<tls@virginia.edu>

Ross Wayland
Digital Library Research and Development
University of Virginia
<rlw@virginia.edu>

Sandra Payette
Computing and Information Science
Cornell University
<payette@cs.cornell.edu>

Introduction

In September 2001, the University of Virginia was awarded a grant from the Andrew W. Mellon Foundation to develop the first digital object repository management system based on the Flexible Extensible Digital Object and Repository Architecture (Fedora) [1]. The Digital Library Research Group at Cornell University originally developed Fedora under a National Science Foundation Grant [1-3]. Fedora is one of a number of repository architectures that have been proposed over recent years for use in digital libraries. Related architectures have included those described in Kahn and Wilensky [4], Arms, et.al. [5], Mönch [6], and Nelson and Maly [7]. The Digital Library Research and Development Department at the University of Virginia Library reinterpreted the architecture and built a prototype that demonstrated the feasibility of the architecture with their large and diverse digital collections [8]. Together, the two groups make up the development team for the project.

The project also includes a deployment team made up of representatives from groups at eight institutions in the US and the UK. This team will build testbeds drawn from their own collections that they will use to evaluate the software and provide feedback to the development team. Descriptions of the active deployment team projects are available [9].

The first phase of the project is aimed at developing version 1.0 of the system, which is described in this article. That software will be made available under a GNU public license on May 1, 2003 at the project web site [10]. Later phases of the three-year project will focus on adding functionality relating to security and policy enforcement, building on research done by Cornell [3], and for interoperability among repositories, building on experiments carried out by Cornell and the Corporation for National Research

Initiatives (CNRI) [2]. Future work by the development team will be aimed at producing software to support very large-scale, very efficient repositories of digital information.

The basic Fedora architectural model

To review the basic Fedora architectural abstractions, an updated, slightly modified, vocabulary is used to facilitate the discussion of four scenarios for the use of Fedora repositories. For the most complete description of the current details of the architecture and software, see Payette/Staples [11].

The Fedora architecture is based on object models that by definition are templates for units of content, called data objects, which can include digital resources, metadata about the resources, and linkages to software tools and services that have been configured to deliver the content in desired ways. These software connections are provided as methods encoded into two kinds of inter-related behavior objects as described below. A Fedora repository provides access to the data objects by leveraging tools and services that are described by the behavior objects. The behavior objects store metadata that describes the operations of the tool/service and the runtime bindings for running the operations. The Web Services Description Language (WSDL) is used to describe the tool/service bindings.

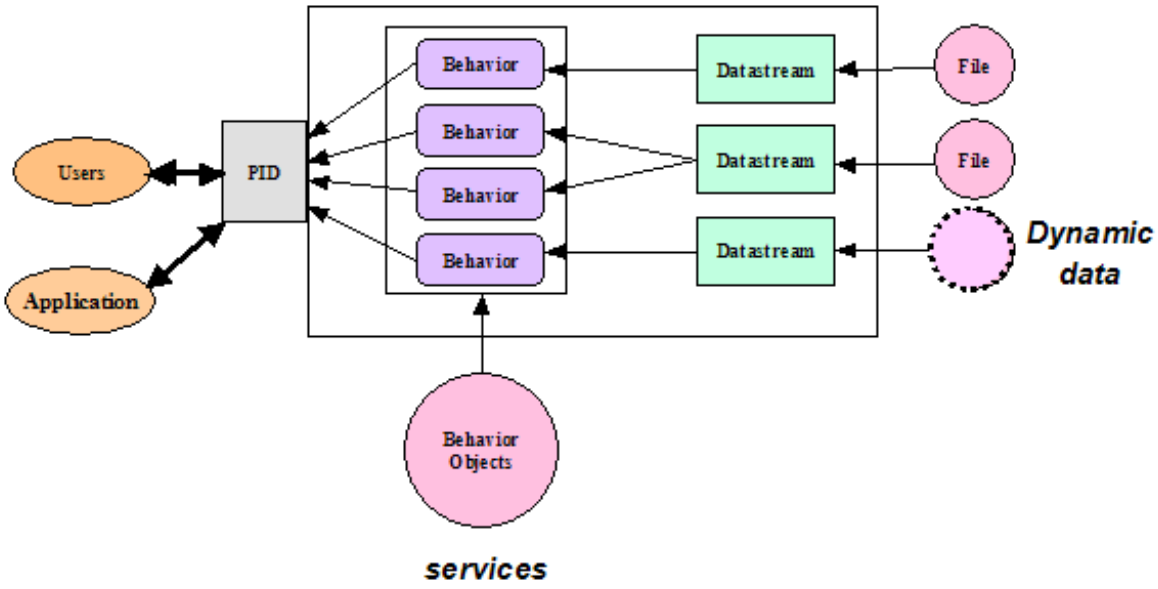

Figure 1. The object model

The digital resources and the metadata are *datastreams* in an object model, definitions of which connect the content model either to internal content under the direct control of the repository or to external content that is delivered via HTTP servers. The content of a datastream is identified using a URL. When an object is ingested into a Fedora repository, a URL for a *managed* datastream is used by the repository system to retrieve the content and store it in the file space under its control; the datastream in the object is updated to be this internal address. When an object contains a datastream defined as *external*, the URL is stored in the datastream and used by the repository to access the data whenever necessary. An *in-line metadata* datastream is a bytestream that is name-spaced XML encoded data stored in the XML instantiation of the object directly, rather than as remote or managed content.

From the user's point of view, the linkages to software tools and services (via *disseminators*) are seen as behaviors upon the units of content. These behaviors can be exploited to deliver varieties of prepared content directly to a web browser. They can also be used to prepare or configure content to be used through some external software application. In a sense, these object models can be thought of as containers that give a useful shape to information poured into them; if the information fits the container, it can immediately be used in predefined ways.

Fedora makes it possible to describe abstract sets of behaviors that constrain a corresponding set of specific processes or mechanisms delivering the behavior described for a given unit of content. One abstract set of behaviors, a behavior definition (*bdef*)

object, can be used to constrain many mechanism sets, or behavior mechanism (*bmech*) objects, ensuring a standardization of behaviors for different units of content that are equivalent in type, but differing in format. A *bdef* object formally defines the terms of a behavior contract that must be upheld by any *bmech* object to be paired with it. In turn, the *bmech* object contains a data contract, the terms of which any data object model subscribing to it must meet. *Bdef* objects and *bmech* objects are analogous to interfaces and implementations in object-oriented programming.

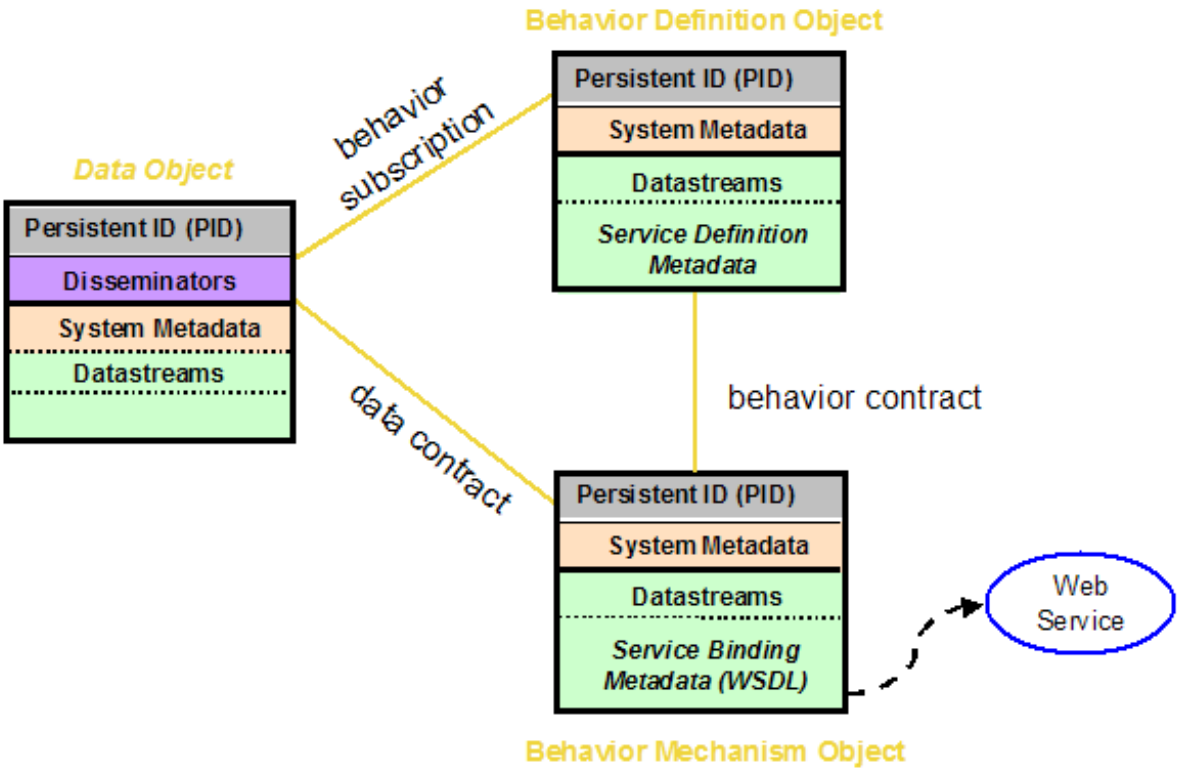

Figure 2. Behavior object contracts

A data object model subscribes to a set of behaviors by linking to a *bdef* object and pairing it with a link to an appropriate *bmech* object. This pair of links defines a disseminator; an object model can contain any number of disseminators. In practical terms, this means a specific data object conforming to the model can have sets of behaviors for a variety of purposes, or sets of behaviors equivalent in purpose but that prepare the object's content to be delivered to applications with different format requirements. In summary, a data object model specifies the number and types of datastreams as well as the set of disseminators every conforming data object will have.

The Software

The Fedora repository system is an open source, digital object repository system using public APIs exposed as web services. As depicted in Figure 1, the Fedora repository system consists of three layers: the Web Services Exposure Layer, the Core Subsystem Layer, and the Storage Layer. The Web Services Exposure Layer is comprised of three related web services described using Web Services Definition Language (WSDL).

- Management Service (API-M) - Defines an open interface for administering the repository. It includes operations necessary for clients to create and maintain digital objects and their components. API-M is implemented as a SOAP-enabled web service.

- Access Service (API-A) - Defines an open interface for accessing digital objects stored in the repository. It includes operations necessary for clients to perform disseminations on objects in the repository (i.e., to access an object's content) and to discover information about an object using object reflection. API-A is implemented as a SOAP-enabled web service.

- Access-Lite Service (API-A-Lite) - Defines a streamlined version of the Fedora Access Service that is implemented as an HTTP-enabled web service.

The WSDL definitions for these services are available at the Fedora site [12].

The Core Subsystem Layer implements the management and access subsystems, defined as a group of related subsystems by the Web Services Layer. The Management subsystem implements the operations necessary for creating, modifying, deleting, importing, exporting, and maintaining digital objects. The management subsystem also includes modules for validation and object integrity to ensure that imported, newly created, and modified objects are valid from both an XML Schema perspective and also from the set of Fedora-specific integrity rules. The PID generation module is responsible for providing a unique persistent identifier (PID) for each digital object. The Access subsystem implements the operations necessary for disseminating the content of digital objects and digital object reflection. Digital object reflection provides a means of discovering additional information about a digital object and its behaviors.

The Storage Layer implements the storage subsystem that handles reading, writing, and removal of data from the repository. Digital objects are stored as XML-encoded files that conform to an extension of the Metadata Encoding and Transmission Standard (METS) schema [13]. For a more detailed description of the software, see Payette/Staples [7].

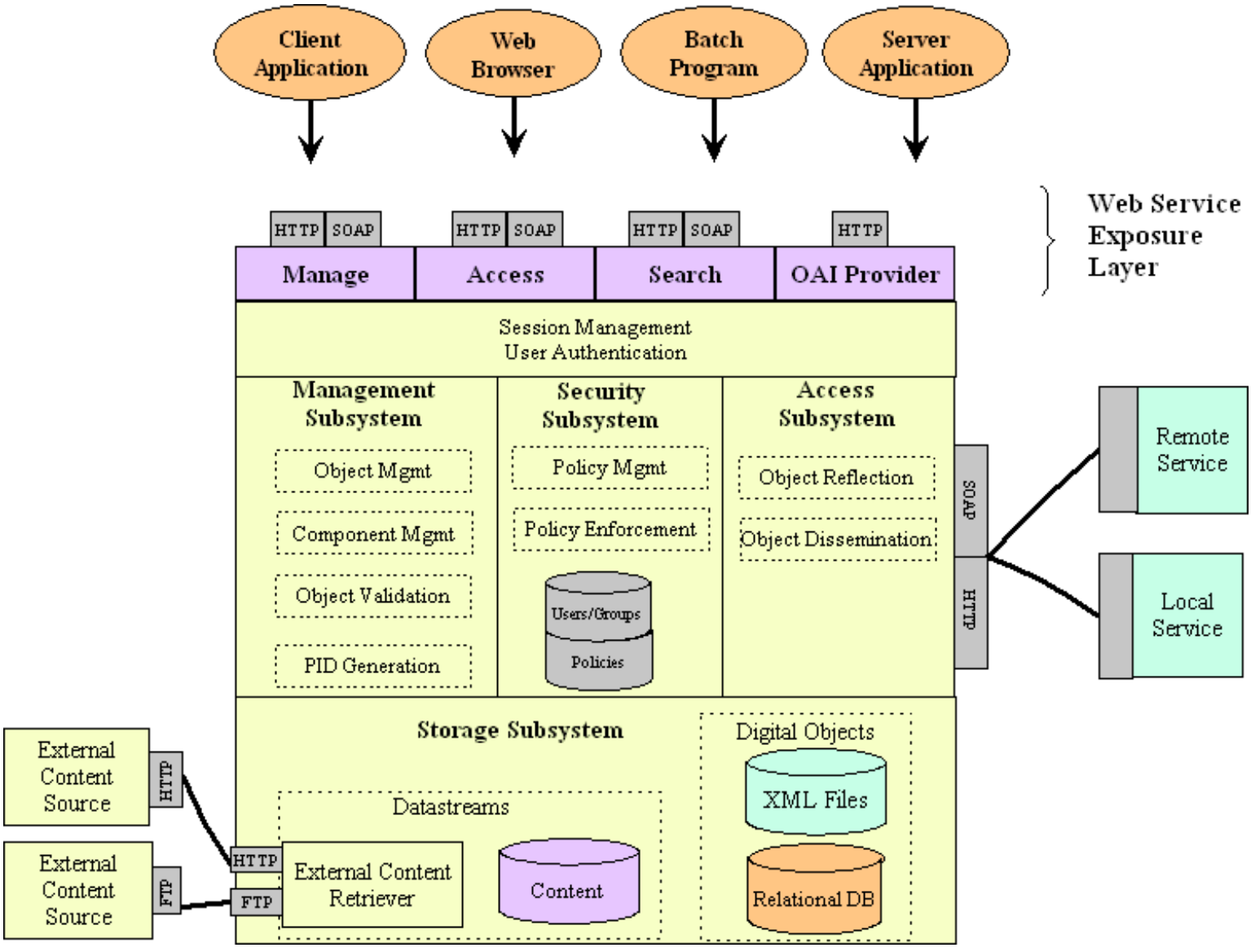

Figure 3. Fedora Repository System

The Fedora architecture provides a flexible object model that offers several key features. In addition to the basic management and access services, Phase I of the Fedora repository system includes a number of additional noteworthy features:

- *XML submission and storage* - Digital objects are stored as XML-encoded files that conform to an extension of the Metadata Encoding and Transmission Standard (METS) schema [9].

- *Parameterized disseminators* - Behaviors defined for an object support user-supplied options that are handled at dissemination time.

- *Access Control and Authentication* - Although Advanced Access Control and Authentication are not scheduled until Phase II of the project, a simple form of access control has been added in Phase I of the project to provide access restrictions based on IP address. IP range restriction is supported in both the Management and Access APIs. In addition, the Management API is protected by HTTP Basic Authentication.

- *Default Disseminator* - The Default Disseminator is a built-in internal disseminator on every object that provides a system-defined behavior mechanism for disseminating the basic contents of an object.

- *Searching* - Selected system metadata fields are indexed along with the primary Dublin Core record for each object. The Fedora repository system provides a search interface for both full text and field-specific queries across these metadata fields.

- *OAI Metadata Harvesting* - The OAI Protocol for Metadata Harvesting [14] is a standard for sharing metadata across repositories. Every Fedora digital object has a primary Dublin Core record that conforms to the schema [15]. This metadata is accessible using the OAI Protocol for Metadata Harvesting, v2.0.

- *Batch Utility* - The Fedora repository system includes a Batch Utility as part of the Management client that enables the mass creation and loading of data objects.

Release 1.0 of the Fedora repository system completes Phase I of a three-year project. Although development plans for Phase II and Phase III are not yet finalized, several additional features and refinements are currently planned including:

- *Component Management* - The component management module provides methods necessary for directly maintaining object components (i.e., datastreams and disseminatiors) once an object has been ingested or created in the repository.

- *Advanced Access Control* - The primary focus of Phase II of the project will be developing an XML-oriented policy expression that can be used to enforce fine-grained object-level policies.

- *Versioning* - Although the architectural infrastructure for versioning of object components is present in Phase I, the complete implementation of versioning is planned for Phase III of the project.

- *Preservation Service* - The Fedora architecture provides the infrastructure upon which a variety of reporting and control mechanisms can be built. Later phases of the project will investigate ways that system metadata about an object and its associated services can be leveraged to include additional reporting tools to monitor objects, provide alerts to vulnerabilities, and perform corrective actions.

- *R2R Repository Federation* - Later phases of the project will also explore the potential for distributed Fedora repositories. To accommodate interoperation among repositories some form of external PID (publication identifier) resolution service will be necessary to resolve the appropriate Fedora repository server given a Fedora PID.

- *Reliability* - Additional work is also planned to enhance the reliability of the core Fedora server to provide improved fault tolerance and to enable mirroring and replication.

- *Performance tuning* - Phase II of the project will also focus on fine-tuning repository performance based on feedback received from the deployment group. Planned additions in this area include caching, load balancing, and enhancing storage

scalability.

Four Use Cases for Fedora Repositories

There are many different ways of implementing Fedora repositories within information management schemes. Decisions about the design of content models for different kinds of objects, particularly the patterns of datastreams included and the number and type of disseminators developed, are basic to such schemes. It is very possible to design relatively simple Fedora repositories that depend on robust applications on the front end, as well as schemes where the same kind of functionality is distributed among a very rich and powerful array of disseminators. Four scenarios for Fedora repositories are presented below as illustrative examples of the possibilities. In reality, most serious repository implementations will probably be hybrids of those described below.

Use Case 1: Fedora "out-of-the-box".

> *The most elementary use of Fedora is management and access for simple content objects. This case is the "low barrier to entry" scenario for Fedora. It takes advantage of the default capabilities of the system, and does not require the creation of any special behavior definitions and behavior mechanisms. Digital Objects are simple containers for data and metadata. Digital object content can be accessed via "built-in" object behaviors.*

Although one of the powerful features of Fedora is the ability to associate specialized disseminators with objects to provide services for content transformation and presentation, this is not required to run a Fedora repository. An individual or institution can use Fedora "out-of-the-box" as a system for managing and delivering simple content entities such as stand-alone images, electronic documents, audio/video files, and other media types.

Used in its most basic way, the Fedora digital object model provides a means of aggregating one or more MIME-typed content streams and associating metadata with the content. For example, an object can be created to house a JPEG image along with technical and descriptive metadata for that image. The Fedora object model can also be used to represent an electronic publication where multiple formats of a paper are stored in the object (e.g., PDF, Postscript, HTML) along with descriptive metadata about the paper (e.g., a Dublin Core record). These simple content models can be the basis for quickly building a variety of digital collections in Fedora.

In terms of management of such collections, Fedora provides a flexible object model and an environment to store digital content and metadata in a secure, extensible repository system. By default, the repository will accept objects encoded in the Metadata Encoding and Transmission Standard (METS), with a few Fedora-specific requirements. Alternatively, objects can be built via client tools (both interactive and batch) that interact with the Fedora Management Web service. In either case, the repository will store digital objects as XML and protect the content datastreams.

In terms of providing access to such collections, Fedora automatically exposes a set of generic disseminations for digital objects. Every digital object stored in a Fedora repository will be assigned a virtual disseminator known as the "Default Disseminator." Whether or not a digital object has custom disseminators associated with it, *every* digital object will have a set of out-of-the-box disseminations via the Default Disseminator. The Default Disseminator provides methods to:

1. Get a profile of the object (Figure 4)
2. List all the disseminators in the object and access them directly (Figure 5)
3. List all the datastreams in the object and access them directly (Figure 6)
4. Get a Dublin Core record for the object.

The methods of the Default Disseminator can be discovered by a user through the Fedora Access Web service. In this simple Fedora use case, a user can search the repository to locate objects and then request views of individual objects. This can be done via a web browser with simple URLs (see syntax defined in API-A-LITE). Fedora provides a default "look-and-feel" for object item lists, object profiles, and Dublin Core records. However, client applications can request that this information be returned as XML and

apply custom rendering. Also, repository system managers can configure their own XSLT stylesheets in the repository to change the default object displays.

This all adds up to a simple way to get started with Fedora. Any objects created in this elementary manner can later be augmented with custom disseminators that provide additional functionality beyond the generic access behavior described above. The next set of use cases will describe a number of interesting ways the Fedora object model and service architecture can be exploited beyond this simple baseline.

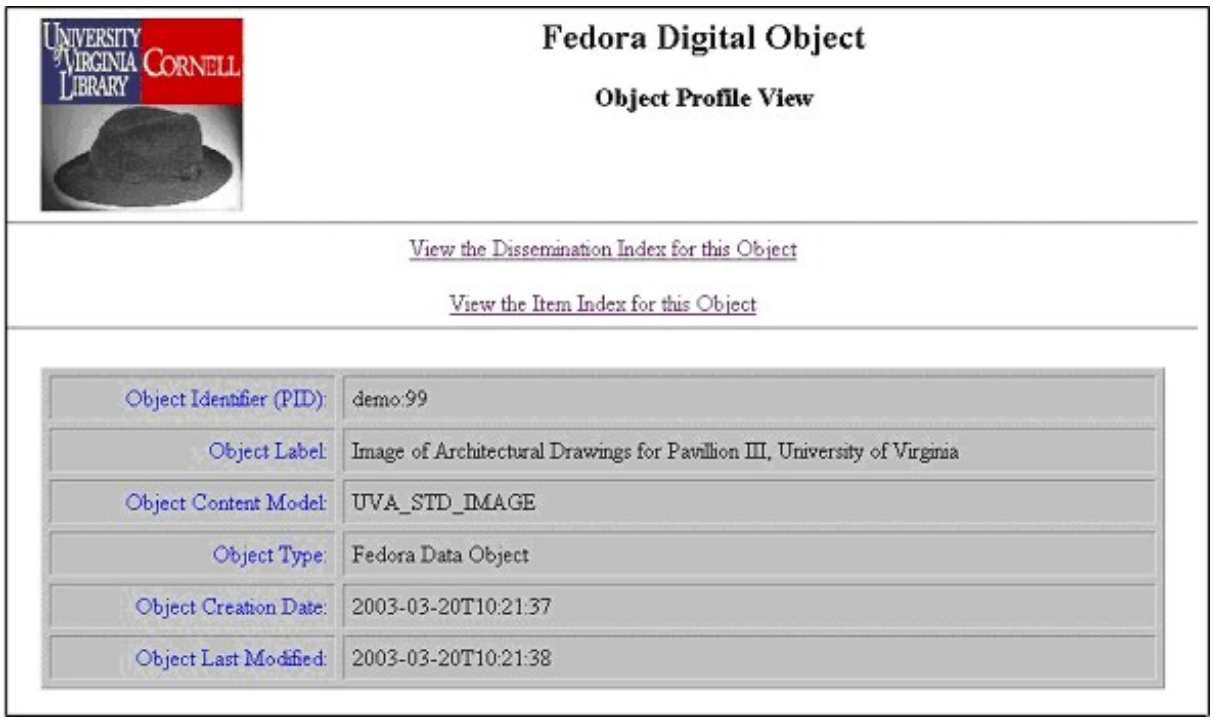

Figure 4. Object Profile

Figure 5. Dissemination Index View

Figure 6. Datastream Index View

Use Case 2: A digital asset management system

> *The Fedora repository is the system used to manage a set of digital assets accessed by one or more client applications. Disseminators on the asset objects are relatively simple and are used to organize the use of assets by applications having specific needs with respect to size and format of the content, and the format of metadata. All relationships among the assets are assumed to be known and exploited through client applications.*

The digital asset management system described in this scenario manages digital images, videos and audio recordings as three different types of data objects, each potentially with more than one object model. Client-side applications with flexible, configurable interfaces depend on the repository to provide very standardized views of the assets. Each object contains metadata that is specific to it. Metadata about relationships among the assets is assumed to be located in external databases, keyed to the repository by the persistent identifier of the object. Metadata about the assets is extracted for the objects and combined with data from external databases to feed the appropriate data to each application. This makes it possible for one digital asset management system to support multiple applications with differing demands.

Every object model used in the repository has one descriptive metadata datastream and one administrative datastream, both of which are defined in a format native to the repository, and subscribes to the same two sets of behaviors, each of which contains methods associated with the respective datastream. The descriptive metadata behaviors include three methods; one to get the native datastream directly, one to convert it to Dublin Core format and one to convert it to MARC format. The administrative metadata behaviors include two methods; one to get the native datastream directly and one that extracts the part of the administrative metadata pertaining to policies associated with the use of the object.

There are multiple object models for each of the three types of data, each with a different combination of content datastreams. All the object models for each media type subscribe to the same behavior definition object that acts as a contract to define and constrain behaviors for those objects. Each model pairs this definition to a different behavior mechanism object that carries out the terms of the contract. Different combinations of datastreams can be balanced by different service definitions in the behavior mechanism objects, making it easy to change the details of the content management scheme while preserving a stable interface to client applications.

Three possible image object models include:

- One datastream that contains a group-4 fax compressed bitonal TIFF image
- Two datastreams that contain a gif thumbnail and a MrSid image
- Three datastreams that contain a gif thumbnail, a jpeg screen size, and a jpeg max size.

All three of these object models subscribe to a behavior definition object that provides four behavior methods:

- *get_thumbnail* - gets a version of the image which fits in a 120x120 pixel box;
- *get_screensize* - gets a version of the image optimized for a 600x800 pixel screen;
- *get_max* - gets the largest size available;
- *get_sized_jpeg* and *get_sized_gif* - accept pixel dimensions as parameters and get the desired size and format of the image on demand.

Object models for audio and video objects would be developed along the same lines. Versions of the content in the most used formats would be included as pre-calculated datastreams, making the behavior methods very simple to preserve processor cycles. Little-used or very easy-to-derive formats would be included using behavior methods that derive them from existing datastreams. From the client's point of view, all the objects have the same formats and resolutions available.

Use Case 3: A digital library for a research university

> *A complex digital library scheme can be developed that depends on disseminators to provide most or all of the functionality needed. In this scenario, software tools for delivery and analysis are added to those described in the previous case. Data objects, which include primary content referencing other data objects, are exploited to associate behaviors with relationships among digital objects. Higher levels of aggregation, with associated behaviors, are provided by both implicit and explicit collection objects.*

This scheme builds on the one described in use case 2, including the same media objects and associated behaviors. Methods would be added to the basic behaviors for those objects to provide links to tools for manipulation or analysis along with the content. For example, a *manipulate_image* method is added to the image objects that delivers an image via Java applet allowing the user to zoom in on the object and change its contrast and brightness. For audio and video objects, as well, an applet can be passed that gives more control to the user than is usually available in a browser.

The major difference in this scenario is that the repository uses no client-side applications other than web browsers; all end-user interactions with the repository are done through web pages created on the fly by disseminations of the objects. Every object has one *web_default* method that gives a default view of the content plus provides the user with links to invoke other disseminations of the object.

For example, a book object has one datastream representing an XML-marked up textual transcription of a book and a *web_book* disseminator. The disseminator provides a variety of ways to render the XML to give views of the text, in addition to methods allowing the users to search the text, count occurrences of words or phrases, or download the book as raw XML, a PDF file and an ebook. The *web_default* method for this object would render a two-frame web page with a title frame at the top and the title page rendered in the frame below. The bottom frame could include javascript menus as buttons. A "Table of Contents" menu could provide a dynamic list of chapters and sections of the book, an "Analyze" menu could provide search and text analysis functions, and a "Download" menu could provide formatting options. All items on each of these menus would be rendered as "<a>" tags with the appropriate dissemination calls in the "href" attribute.

This all becomes more interesting when the repository includes objects whose contents contain references to other objects. The book

object above and electronic finding aids (parent objects) often contain references to pages (child objects). Collections of art and architecture information are represented as one object for each building or artwork that contains an XML file describing the work and associates any number of other objects with it, including images and video or audio clips, texts of letters describing or referring to it, or virtual reality objects representing it. In all of these cases, disseminations of the parent object renders references to child objects as links that disseminate the child.

In an artwork example, a web page is dynamically created that displays descriptive information about the artwork and presents thumbnail images of each of the child image objects. On the web page, the "<img>" tags for the thumbnails have "src" attributes that contain dissemination calls to the image object to produce a thumbnail; these images and tags are then themselves enclosed in "<a>" tags that have an "href" attribute containing the dissemination call to get a larger version of the image wrapped in the Java applet described above. When the page is sent back to the browser, the image object is asked to disseminate the thumbnail; when the user clicks on the thumbnail, the large image is delivered within a manipulation tool via another disseminator.

All the parent objects are examples of explicit collection objects, in addition to being primary data objects themselves; all the members of the collection are explicitly mentioned as child objects. Collection objects can be defined to create explicit collections of collections, containing information about the collection itself, references to its children and, potentially, information about how to classify the children in the context of the collection. Disseminators of these collection objects provide behaviors at the collection level, depending on the disseminators specific to its child objects.

Other collection objects in this repository are implicit. In these objects, XML datastreams contain information about the collection and information about how to classify members of the collection, but rather than explicit references to its child objects, it contains rules to dynamically assemble the members of the collection. These rules refer to patterns of information in the child objects. The value of these types of collection objects is that they do not need to be updated each time a new child is added. For example, all book objects would have one or more collection identifiers in their headers. A collection object would have an Xpath rule describing where to look in the XML and what value to look for to identify children of the collection. A geographically organized collection object would contain a geographic boundary box definition in an Xpath rule for how to find the latitude and longitude in each child object. Disseminators on both objects would use this information to assemble the members of the collection in different ways.

In this repository, discovery search services across the whole repository are provided by the repository's built-in search of the Dublin core metadata datastreams for each of the objects. Full resource searches of particular large collections are built around implicit collection objects. Each of these collection objects contains a datastream representing the index for the collection. Disseminators on the collection object contain methods for building a new index, disseminating search pages for different audiences, and describing the collection. Collection objects for sub-collections of these indexed collections only contain a reference to the parent collection and an Xpath rule for recognizing children of the sub-collection in the larger index.

Use Case 4: Fedora for distributed content objects

> *Fedora can be used to build a widely distributed repository where digital objects contain a mixture of repository-managed content and content that is completely outside of the direct custody of the repository. The repository system provides the ability to deliver these objects in interesting ways. It also provides a means of asserting managerial control over these kinds of objects.*

One important feature of Fedora is that the object model does not require datastreams to contain content actually stored in the repository system. Complex digital objects can reference remote content. In fact, the Fedora object model supports a spectrum of possible configurations from objects where all content is stored in the repository, to "hybrid" objects where some content is managed by the repository and some is stored remotely, to "surrogate" objects whose content is external to the repository.

There are many scenarios that require this type of flexibility. A prime example occurs in the area of learning objects and courseware. Many educators create modules that integrate primary source resources from the web with their own instructional materials. Often this is done in a one-up manner in the form of custom web pages. Another approach is to employ toolkits such as Instructional Architect [16] and CREATE [17], which enable educators to create web-based instructional modules with minimal programming experience.

The Fedora repository system can offer strong architectural underpinnings to these and similar endeavors. The Fedora object model

provides the flexibility to aggregate references to web resources along with locally stored content such as lesson documents, interactive simulations (e.g., Java applets), and educational videos. It also enables educators to conceive of appropriate behavior definitions for delivering these objects and to associate presentation or transformation services with the objects. Interestingly, different disseminators can be created to provide customized views of the content for different audiences. For example, one disseminator may present an object's content in a manner appropriate for the K-12 audience, while another disseminator may provide an exhibit for the general public.

A key aspect of this use case is the ability to combine content directly controlled by the repository with content that is located elsewhere. As institutions increasingly rely on distributed web resources, as they enter into relationships where resources are shared among several institutions, or as they license resources from remote content providers, a hybrid and distributed digital object model becomes a necessity for an institutional repository system. Where licensed content is made available in a flexible, open manner, a Fedora repository can integrate it locally in interesting ways.

For example, if a library subscribes to a service that aggregates multiple journal subscriptions into one license, an object could be created to represent that service. The object would contain metadata about the service and about the library's subscription to it, as well as possibly containing locally written instructions for its use. Disseminators could give users different views of the metadata and instructions, and then direct users to the service. If the license also made Z39.50 access to the service possible, each of the included journals could be represented by separate objects that could be integrated in different ways, disseminating locally developed search interfaces as well as information about the specific journals.

Acknowledgements

We would like to thank Don Waters and the Andrew W. Mellon Foundation for their help in making this project possible. We would also like to thank Ronda Grizzle, Leslie Johnston, Carl Lagoze, Bill Niebel, Tim Sigmon and Chris Wilper for their help on this paper and on the project.

References

[1] Payette, Sandra, and Carl Lagoze: Flexible and Extensible Digital Object and Repository Architecture (FEDORA). Second European Conference on Research and Advanced Technology for Digital Libraries. *Lecture Notes in Computer Science*, Vol. 1513. Springer-Verlag, Berlin Heidelberg New York (1998) 41-59. <http://www.cs.cornell.edu/payette/papers/ECDL98/FEDORA.html>.

[2] Payette, Sandra, Christophe Blanchi, Carl Lagoze, and Edward Overly: Interoperability for Digital Objects and Repositories: The Cornell/CNRI Experiments. *D-Lib Magazine.* (May 1999) <doi:10.1045/may99-payette>.

[3] Payette, Sandra, and Carl Lagoze: Policy-Carrying, Policy-Enforcing Digital Objects. Fourth European Conference on Research and Advanced Technology for Digital Libraries. *Lecture Notes in Computer Science*, Vol. 1923. Springer-Verlag, Berlin Heidelberg New York (2000) 144-157. <http://www.cs.cornell.edu/payette/papers/ECDL2000/pcpe-draft.ps>.

[4] Kahn, Robert and Robert Wilensky, "A Framework for Distributed Digital Object Services," Corporation for National Research Initiatives, 1995, <http://www.cnri.reston.va.us/k-w.html>.

[5] Arms, William Y., Christophe Blanchi, and Edward A. Overly, "An Architecture for Information in Digital Libraries," *D-Lib Magazine*, February 1997, <doi:10.1045/february97-arms>.

[6] Mönch, Christian, "INDIGO - An Approach to Infrastructures for Digital Libraries," *Fourth European Conference on Research and Advanced Technology for Digital Libraries*, Portugal, Springer, 2000, Lecture Notes in Computer Science, Vol. 1923.

[7] Nelson, Michael L. and Kurt Maly, "Buckets: Smart Objects for Digital Libraries," *Communications of the ACM*, 44(5), May 2001, pp. 60-62.

[8] Staples, Thornton, and Ross Wayland: Virginia Dons FEDORA: A prototype for a digital object repository. *D-Lib Magazine* (July 2000) <doi:10.1045/july2000-staples>.

[9] The Fedora Project: Developing An Open-Source Digital Repository Management System, <http://www.fedora.info/testbed.shtml>.

[10] The Fedora Project: Developing An Open-Source Digital Repository Management System, Information Page <http://www.fedora.info>.

[11] Payette, Sandra and Thornton Staples, "The Mellon Fedora Project: Digital Library Architecture Meets XML and Web Services," Sixth European Conference on Research and Advanced Technology for Digital Libraries. *Lecture Notes in Computer Science*, Vol. 2459. Springer-Verlag, Berlin Heidelberg New York (2002) 406-421. <http://www.fedora.info/documents/ecdl2002final.pdf>.

[12] Web Services Definition Language (WSDL), <http://www.fedora.info/definitions/1/0/api/>.

[13] METS (Metadata Encoding and Transmission Standard), <http://www.loc.gov/standards/mets/>.

[14] OAI Protocol for Metadata Harvesting, <http://www.openarchives.org/>.

[15] XML Schema 2002-03-18 by Pete Johnston, <http://www.openarchives.org/OAI/2.0/oai_dc.xsd>.

[16] Instructional Architect toolkit, <http://ia.usu.edu>.

[17] CREATE (Component Repository and Environment for Assembly of Teaching Environments), <http://ir.chem.cmu.edu/create>.

Top | Contents
Search | Author Index | Title Index | Back Issues
Previous Article | Next Article
Home | E-mail the Editor

DOI: 10.1045/april2003-staples