Fedora Tutorial #1

Introduction to Fedora

Fedora 3.0 Beta 2
6/13/2008 13:01 PM

FedoraCommons™

Author: The Fedora Development Team

Purpose: This tutorial introduces the basic development questions, design concepts and project goals of the Flexible Extensible Digital Object Repository Architecture (Fedora).

Audience: This tutorial is intended for anyone who will be using the Fedora software in any capacity, or who is generally interested in Fedora and its development.

Table of Contents

Table of Contents 3
Figures 4
Tables 4
Section 1: What is Fedora? 5
What is Fedora? 5
Fedora History 5
Section 2: Motivation 6
The Problem of Digital Content 6
Key Research Questions 6
Fedora Goals 7
Design Advantages—Where the Rubber Hits the Road 7
Fedora’s Digital Object Model 7
Distributed Repositories 8
Preservation & Archiving 8
Content Repurposing 8
Web Services 9
Easy Integration with Other Applications and Systems 9
Section 3: Digital Object Model 9
Fedora Digital Object Overview 9
Datastreams 11
Digital Object Model – An Access Perspective 14
Section 4: What are Digital Object Relationships? 15
Why are Fedora Digital Object Relationships Important? 16
Where is Digital Object Relationship Metadata Stored? 17
How is Digital Object Relationship Metadata Encoded? 17
Resource Index - RDF-based Indexing for Digital Objects 19
Section 4: The Content Model Architecture 19
Introduction 19
Content Model Architecture Overview 21
Specializing Digital Objects 24
Section 4: Fedora Repository Server 27
Fedora Server Architecture 27
Client and Web Service Interactions 28

Figures

Figure 1: Fedora Digital Object Data Model 10
Figure 2: Fedora Digital Object Datastreams 12
Figure 3: Fedora Digital Object Access Perspective 14
Figure 4: A Network of Digital Objects 16
Figure 5: Fundamental CMA Relationships 26
Figure 6: Fedora System Architecture (simplified) 27
Figure 7: Client and Web Services Interaction 28

Tables

Table 1: Fundamental Fedora Object Types 24
Table 2: Relationships Between the Fedora Object Types 25

Section 1: What is Fedora?

What is Fedora?

Fedora is an acronym for Flexible Extensible Digital Object Repository Architecture. Fedora's flexibility makes it capable of serving as a digital repository for a variety of use cases. Among these are digital asset management, institutional repositories, digital archives, content management systems, scholarly publishing enterprises, and digital libraries. Fedora is open-source software licensed under the Educational Community License 2.0.

Fedora History

Fedora began in 1997 as a DARPA and NSF funded research project at Cornell University, where the initial reference implementation was developed by Sandra Payette, Carl Lagoze, and Naomi Dushay. Work at Cornell included a CORBA-based technical implementation, work on policy enforcement, and extensive interoperability testing with CNRI.

The first practical application of Fedora was the digital library prototype developed at University of Virginia by Thornton Staples and Ross Wayland in 1999, where the software was adapted to the web and an RDBMS was added to improve performance. The initial work done on the Fedora prototype included scalability testing for 10 million objects.

A full scale development project was begun in 2002 with grant funding from the Andrew W. Mellon Foundation. This project's charge was to create a production quality Fedora system, using XML and web services to deliver digital content. Fedora 1.0 was released in May 2003, with subsequent releases following approximately every quarter which have added functionality and corrected bugs discovered by users and the Fedora development team.

In June 2004, the Andrew W. Mellon Foundation funded Fedora Phase 2 for an additional 3 year project. Phase 2 development is underway as of this writing. In 2008 the Moore Foundation provided a grant that permitted the creation of Fedora Commons, a 501(c)3 non-profit corporation, to act as the home for the sustainable development of the Fedora software and with a mission to develop technology to build the technologies that enable durable access to digital content containing our cultural and scientific heritage.

Section 2: Motivation

This section provides information on the issues and questions that have motivated the development of Fedora.

The Problem of Digital Content

Digital content is not just documents, nor is it made up exclusively of the content from digital versions of currently owned non-digital content.

- **Conventional Objects:** books and other text objects, geospatial data, images, maps
- **Complex, Compound, Dynamic Objects:** video, numeric data sets and their associated code books, timed audio

As users become more sophisticated at creating and using complex digital content, digital repositories must also become more sophisticated. As digital collections grow, and are made use of in previously unconsidered ways, repository managers are faced with management tasks of increasing complexity. Collections are being built which contain multiple data types, and organizations have discovered a need to archive and preserve complex objects like those listed above, as well as web sites and other complex, multi-part documents. And finally, as collections grow in both size and complexity, the need to establish relationships between data objects in a repository becomes more and more apparent.

Key Research Questions

After considering the complexities of digital content, the original developers of Fedora at Cornell University developed five key research questions:

1. How can clients interact with heterogeneous collections of complex objects in a simple and interoperable manner?
2. How can complex objects be designed to be both generic and genre-specific at the same time?
3. How can we associate services and tools with objects to provide different presentations or transformations of the object content?
4. How can we associate specialized, fine-grained access control policies with specific objects, or with groups of objects?
5. How can we facilitate the long-term management and preservation of objects?

Fedora Goals

These five key research questions led logically to ten goals for Fedora's development.

- **Identifiers:** provision of persistent identifiers; unique names for all resources without respect for machine address
- **Relationships:** support for relationships between objects
- **Tame Content:** normalization of heterogeneous content and metadata based on an extensible object model
- **Integrated Management:** efficient management by repository administrators not only of the data and metadata in a repository, but also of the supporting programs, services and tools that make presentation of that data and metadata possible
- **Interoperable Access:** provision of interoperable access by means of a standard protocol to information about objects and for access to object content; discovery and execution of extensible service operations for digital objects
- **Scalability:** provision of support for >10 million objects
- **Security:** provision of flexible authentication and policy enforcement
- **Preservation:** provision to features to support longevity and archiving, including XML serialization of objects and content versioning
- **Content Recon:** reuse of objects including object content being present in any number of contexts within a repository; repurposing of objects allowing dynamic content transformations to fit new presentations requirements
- **Self-Actualizing Objects:** ability of digital objects to disseminate launch-pads or tools for end-user interaction with content

Design Advantages—Where the Rubber Hits the Road

Fedora's design offers many advantages to repository developers and administrators.

Fedora's Digital Object Model

The digital object model offers the strengths and advantages of

- **Abstraction:** The object model is the same whether the object is data, behavior definitions, or behavior mechanism. It also does not matter what kind of data the digital objects is representing—text, images, maps, audio, video, geospatial data are all the same to Fedora.
- **Flexibility:** Implementers of Fedora can design their content models to best represent their data and the presentation requirements of their specific use case.
- **Generic:** Metadata and content are tightly linked within the digital object.

- **Aggregation:** Fedora objects can refer to data that is stored locally or that is stored on any web accessible server.
- **Extensibility:** Fedora's behavior interfaces are extensible because services are directly associated with data within a Fedora object. As the services change, the objects change along with them.

Distributed Repositories

The Fedora Architecture, as originally designed by Payette and Lagoze, was intended to support distributed repositories. This vision is described in the Fedora Specification and placeholders exist in the current software to build this functionality. Repository federation is important for several reasons. First, federation is a natural requirement for delivering integrated access to digital resources that are owned or managed by several institutions. Second, federation makes it easy for digital library and other applications to interface with multiple information sources in a seamless manner. Third, federation can help with scalability or performance issues for very large repositories. Specifically, a local federation of repositories can be established as a means of distributing load and object storage among several running repository instances, so that together these separate instances can be treated as one 'virtual repository.'

Preservation & Archiving

Fedora's archival and preservation capabilities include:

- **XML:** Fedora objects' XML and the schema upon which they are based are preserved at ingest, during storage, and at export.
- **Content Versioning:** Fedora repositories offer implementers the option of versioning data objects. When a data object is versioned, the object's audit trail is updated to reflect the changes made to the object, when the change was made and by whom and a new version of the modified data is added to the object's XML. This new Datastream cascades from the original and is numbered to show the relationship between original and version. This allows users to retrieve older versions of a data object by performing a date/time search and retrieval, or the most current version if the date/time criteria are not included in the search.
- **Object to Object Relationships:** Relationships between objects can be stored via the metadata included in the objects. This allows implementers to link together related objects into parent/child relationships.
- **Event History:** Every object in a Fedora repository contains an audit trail, which preserves a record of every change made to the object.

Content Repurposing

Application of different stylesheets to the data and metadata of a Fedora object allows multiple views of the object's content and metadata; for example, one view for a domain scholar and another for a K-12 audience. Additionally, content referenced in a Fedora data object can be dynamically transformed as it is called by a user by use of custom services. Because of this inherent strength and flexibility, new views and data transformations are simple to add over time as the implementer's and user's requirements change.

Web Services

Fedora is exposed via web services and can interact with other web services. The interfaces and XML transmission are defined in WSDL.

Easy Integration with Other Applications and Systems

Fedora is not architected in such a way that any particular workflow or end-user application is assumed. Thus Fedora is able to function as a generic repository substrate upon which many kinds of applications can be created and to take advantage, over time, of advances in web services. This ability to grow and change builds longevity into a digital library system built on the Fedora architecture.

Section 3: Digital Object Model

Fedora Digital Object Overview

Fedora uses a "compound digital object" design which aggregates one or more content items into the same digital object. Content items can be of any format and can either be stored locally in the repository, or stored externally and just referenced by the digital object. The Fedora digital object model is simple and flexible so that many different kinds of digital objects can be created, yet the generic nature of the Fedora digital object allows all objects to be managed in a consistent manner in a Fedora repository.

A good discussion of the Fedora digital object model (for Fedora 2 and prior versions) exists in a recent paper (draft) published in the International Journal of Digital Libraries. While some details of this paper have been made obsolete (e.g. Disseminators) by the Content Model Architecture, a refinement of the original Fedora concepts introduced in Version 3.0, the core principles of the model remain the same. The Fedora digital object model is defined in XML schema language (see The Fedora Object XML - FOXML). For more information, also see the Introduction to FOXML in the Fedora System Documentation. A data object in a Fedora repository describes content (data and metadata) and a set of associated behaviors or services that can be applied to that content. Data objects comprise the bulk of a repository.

Figure 1 below shows the basic data model of a Fedora digital object.

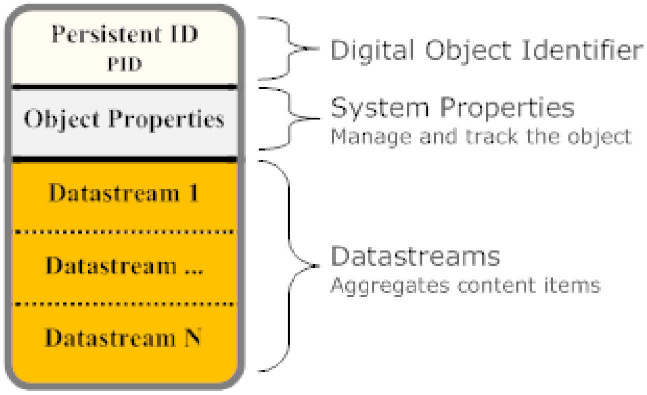

Figure 1: Fedora Digital Object Data Model

A Fedora digital object consists of three parts:

1. **Digital Object Identifier:** A unique, persistent identifier for the digital object.
2. **System Properties:** A set of system-defined descriptive properties that is necessary to manage and track the object in the repository.
3. **Datastream(s):** The element in a Fedora digital object that represents a content item.

The FOXML metadata for a digital object is the metadata that must be recorded with every digital object to facilitate the management of that object. FOXML metadata is distinct from other metadata that is stored in the digital object as content. This type of metadata is the metadata that is *required* by the Fedora repository architecture. All other metadata (e.g., descriptive metadata, technical metadata) is considered optional from the repository standpoint, and is treated as a Datastream in a digital object.

Object Properties describe the object's type, its state, the content model to which it subscribes, the created and last modified dates of the object, and its label.

Datastreams represent the digital content that is the essence of the digital object (e.g., digital images, encoded texts, audio recordings). All forms of metadata, except system metadata, are also treated as content, and are therefore represented as Datastreams in a digital object. All Datastreams have the potential to be disseminated from a digital object. A Datastream can reference any type of content, and that content can be stored either locally or remotely to the repository system.

Datastreams

A Datastream is the element of a Fedora digital object that represents a content item. A Fedora digital object can have one or more Datastreams. Each Datastream records useful attributes about the content it represents such as the MIME-type (for Web compatibility) and, optionally, the URI identifying the content's format (from a format registry). The content represented by a Datastream is treated as an opaque bit stream; it is up to the user to determine how to interpret the content (i.e. data or metadata). The content can either be stored internally in the Fedora repository, or stored remotely (in which case Fedora holds a pointer to the content in the form of a URL). The Fedora digital object model also supports versioning of Datastream content (see the Fedora Versioning Guide for more information).

Each Datastream is given a Datastream Identifier which is unigue within the digital object's scope. Fedora reserves three Datastream Identifiers for its use, "DC", "AUDIT" and "RELS-EXT". Every Fedora digital object has one "DC" (Dublin Core) Datastream by default which is used to contain metadata about the object (and will be created automatically if one is not provided). Fedora also maintains a special Datastream, "AUDIT", that records an audit trail of all changes made to the object, and cannot be edited since only the system controls it. The "RELS-EXT" Datastream is primarily used to provide a consistent place to describe relationships to other digital objects. In addition, a Fedora digital object may contain any number of custom Datastreams to represent user-defined content.

Decisions about what to include in a Fedora digital object and how to configure its Datastreams are choices as you develop content for your repository. The examples in this tutorial demonstrate some common models that you may find useful as you develop your application. Different patterns of Datastream designed around particular "genre" of digital object (e.g., article, book, dataset, museum image, learning object) are known as "content models" in Fedora.

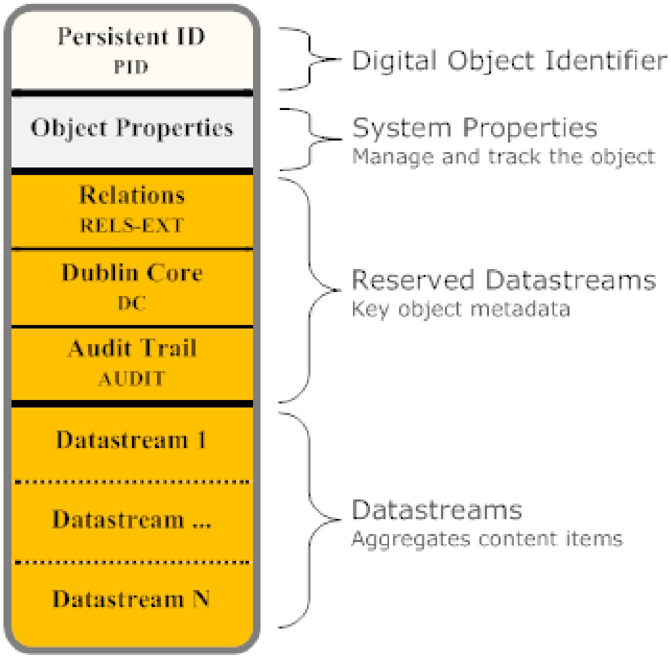

Figure 2: Fedora Digital Object Datastreams

The basic properties that the Fedora object model defines for a Datastream are as follows:

- **Datastream Identifier:** an identifier for the Datastream that is unique within the digital object (but not necessarily globally unique)
- **State:** the Datastream state of Active, Inactive, or Deleted
- **Created Date:** the date/time that the Datastream was created (assigned by the repository service)
- **Modified Date:** the date/time that the Datastream was modified (assigned by the repository service)
- **Versionable:** an indicator (true/false) as to whether the repository service should version the Datastream. By default the repository versions all Datastreams.
- **Label:** a descriptive label for the Datastream
- **MIME Type:** the MIME type of the Datastream (required)
- **Format Identifier:** an optional format identifier for the Datastream. Examples of emerging schemes are PRONOM and the Global Digital Format Registry (GDRF).
- **Alternate Identifiers:** one or more alternate identifiers for the Datastream. Such identifiers could be local identifiers or global identifiers such as Handles or DOI.

- **Checksum:** an integrity stamp for the Datastream which can be calculate using one of many standard algorithms (MD5, SHA-1, etc.)
- **Bytestream Content:** the "stuff" of the Datastream is about (such as a document, digital image, video, metadata record)
- **Control Group:** pertaining the the bytestream content, a new Datastream can be defined as one of four types, or control groups, as follows:
    - ***Internal XML Metadata*** - In this case, the Datastream will be stored as XML that is actually stored inline within the digital object XML file. The user may enter text directly into the editing window or data may imported from a file by clicking Import and selecting or browsing to the location of the XML metadata file.
    - ***Managed Content*** - In this case, the Datastream content will be stored in the Fedora repository and the digital object XML file will store an internal identifier to that Datastream. To get content, click Import and select or browse to the file location of the import file. Once import is complete, you will see the imported file in a preview box on the screen.
    - ***External Referenced Content*** - In this case, the Datastream content will be stored outside of the Fedora repository, and the digital object will store a URL to that Datastream. The Datastream is "by reference" since it is not actually stored inside the Fedora repository. While the Datastream content is stored outside of the Fedora repository, at runtime, when an access request for this type of Datastream is made, the Fedora repository will use this URL to get the content from its remote location, and the Fedora repository will mediate access to the content. This means that behind the scenes, Fedora will grab the content and stream in out the the client requesting the content as if it were served up directly by Fedora. This is a good way to create digital objects that point to distributed content, but still have the repository in charge of serving it up. To create this type of Datastream, specify the URL for the Datastream content in the Location URL text box.
    - ***Redirect Referenced Content*** - In this case, the Datastream content is also stored outside the repository and the digital object points to its URL ("by-reference"). However, unlike the External Referenced Content scenario, the Redirect scenario signals the repository to redirect to the URL when access requests are made for this Datastream. This means that the Datastream will not be streamed through the Fedora repository when it is served up. This is beneficial when you want a digital object to have a Datastream that is stored and served up by some external service, and you want the repository to get out of the way when it comes time to serve the content up. A good example is when you want a Datastream to be content that is stored and served by a streaming media server. In such a case, you would want to pass control to the media server to actually stream the content to a client (e.g., video streaming), rather than have Fedora in the middle re-streaming

the content out. To create a Redirect Datastream, specify the URL for the content in the Location text box.

Digital Object Model – An Access Perspective

Below is an alternative view of a Fedora digital object that shows the object from an access perspective. The digital object contains Datastreams and a set of object properties (simplified for depiction) as described above. A set of access points are defined for the object using the methods described below. Each access point is capable of disseminating a "representation" of the digital object. A representation may be considered a defined expression of part or all of the essential characteristics of the content. In many cases, direct dissemination of a bit stream is the only required access method; in most repository products this is only supported access method. However, Fedora also supports disseminating virtual representations based on the choices of content modelers and presenters using a full range of information and processing resources. The diagram shows all the access points defined for our example object.

For the access perspective, it would be best if the internal structure of digital object is ignored and treated as being encapsulated by its access points. Each access point is identified by a URI that conforms to the Fedora "info" URI scheme. These URIs can be easily converted to the URL syntax for the Fedora REST-based access service (API-A-LITE). It should be noted that Fedora provides a several protocol-based APIs to access digital objects. These protocols can be used both to access the representation and to obtain associated metadata at the same access point.

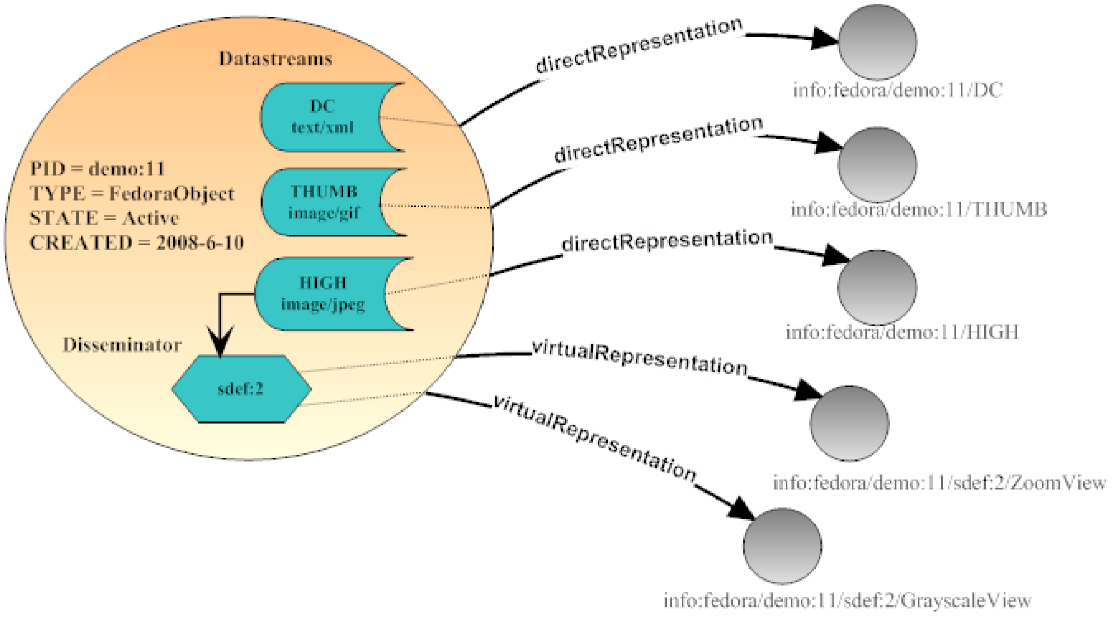

Figure 3: Fedora Digital Object Access Perspective

By default, Fedora creates one access point for each Datastream to use for direct dissemination of its content. The diagram shows how these access points map to the Datastreams. The example object aggregates three Datastreams: a Dublin Core metadata record, a thumbnail image, and a high resolution image. As shown, each Datastream is accessed from a separate URI.

Custom access points are created using the Content Model Architecture by defining control objects as described below. Behind the scenes, custom access points connect to services that are called on by the repository to produce representations of the object. Custom access points are capable of producing both virtual and direct representations (though they are likely to provide slower performance). Content in the Datastreams may be used as input as well as caller-provided parameters. A "virtual representation" is produced at runtime using any resource the service can access in conjunction with content generated in its code. In this example, there is one service that contains two operations, one for producing zoomable images and one for producing grayscale images. These operations both require a jpeg image as input, therefore the Datastream labeled "HIGH" is used by this service. Fedora will generate one access point for each operation defined by the service. Control objects contain enough information so that a Fedora repository can automatically mediate all interactions with the associated service. The Fedora repository uses this information to make appropriate service calls at run time to produce the virtual representation. From a client perspective this is transparent; the client just requests dissemination from the desired access point.

Section 4: What are Digital Object Relationships?

Fedora digital objects can be related to other Fedora objects in many ways. For example there may be a Fedora object that represents a collection and other objects that are members of that collection. Also, it may be the case that one object is considered a part of another object, a derivation of another object, a description of another object, or even equivalent to another object. For example, consider a network of digital objects pertaining to Thomas Jefferson, in which scholarly works are stored as digital objects, which are related to other digital objects representing primary source materials in libraries or museums. The composite scholarly objects can be considered a graph of related digital objects. Other types of objects can also be related to the scholarly object over time, for instance annotations about the scholarly object can be created by others and related to the original object. Also, digital objects can be created to act as "surrogates" or "proxies" for dynamically produced web content such as an Amazon page for a book relevant to the scholarly object. Such a network of digital objects can be created using Fedora, which in the abstract, would look like Figure 4.

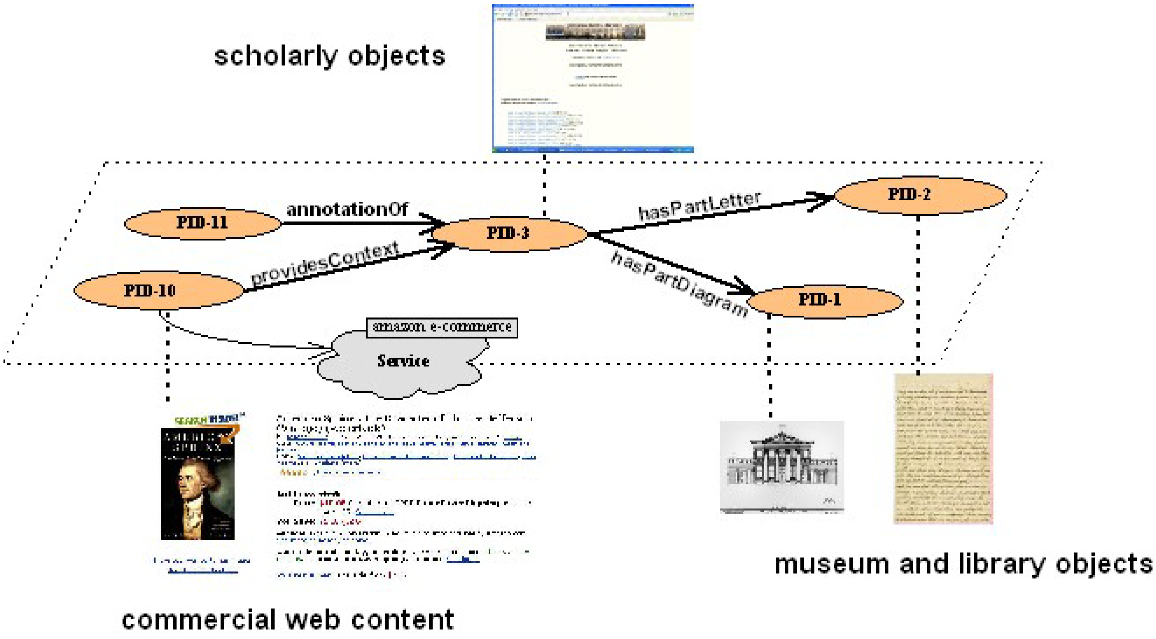

Figure 4: A Network of Digital Objects

Digital object relationship metadata is a way of asserting these various kinds of relationships for Fedora objects. A default set of common relationships is defined in the Fedora relationship ontology (actually, a simple RDF schema) which defines a set of common generic relationships useful in creating digital object networks. These relationships can be refined or extended. Also, communities can define their own ontologies to encode relationships among Fedora digital objects. Relationships are asserted from the perspective of one object to another object as in the following general pattern:

[<subjectFedoraObject> <relationshipProperty> <targetFedoraObject>]

The first Fedora object is considered the "subject" of the relationship assertion. The relationship, itself, is considered a property of the subject. The target Fedora object is the related object. Thus, a valid relationship assertion as an English-language sentence might be:

[<MyCatVideo> <is a member of the collection> <GreatCatVideos>]

Why are Fedora Digital Object Relationships Important?

The creation of Fedora digital object relationship metadata is the basis for enabling advanced access and management functionality driven from metadata that is managed within the repository. Examples of the uses of relationship metadata include:

- Organize objects into collections to support management, OAI harvesting, and user search/browse
- Define bibliographic relationships among objects such as those defined in Functional Requirements for Bibliographic Records
- Define semantic relationships among resources to record how objects relate to some external taxonomy or set of standards
- Model a network overlay where resources are linked together based on contextual information (for example citation links or collaborative annotations)
- Encode natural hierarchies of objects
- Make cross-collection linkages among objects (for example show that a particular document in one collection can also be considered part another collection)

Where is Digital Object Relationship Metadata Stored?

Object-to-Object relationships are stored as metadata in digital objects within a special Datastream. This Datastream is known by the reserved Datastream identifier of "RELS-EXT" (which stands for "Relationships-External"). Each digital object can have one RELS-EXT Datastream which is used exclusively for asserting digital object relationships.

A RELS-EXT Datastream can be provided as part of a Fedora ingest file. Alternatively, it can be added to an existing digital object via component operations of the Fedora management service interface (i.e., addDatastream). Refer to the FOXML reference example to see an example of the RELS-EXT Datastream in context. Modifications to the RELS-EXT Datastream are made via the Fedora management interface (i.e., modifyDatastream).

The RELS-EXT Datastream is encoded as an Inline XML Datastream, meaning that the relationships metadata is expressed directly as XML within the digital object XML file (as opposed the relationship metadata existing in a separate XML file that the digital object points to by reference).

How is Digital Object Relationship Metadata Encoded?

Fedora object-to-object metadata is encoded in XML using the Resource Description Framework (RDF). The relationship metadata must follow a prescribed RDF/XML authoring style where the subject is encoded using <rdf:Description>, the relationship is a property of the subject, and the target object is bound to the relationship property using the rdf:resource attribute. The subject and target of a relationship assertion must be URIs that identify Fedora digital objects. These URIs are based on Fedora object PIDs and conform to the syntax described for the fedora "info" URI scheme. The syntax for asserting relationships in RDF is as follows:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:fedora="info:fedora/fedora-system:def/relations-external#"
  xmlns:myns="http://www.nsdl.org/ontologies/relationships#">

  <rdf:Description rdf:about="info:fedora/demo:99">
          <fedora:isMemberOfCollection rdf:resource="info:fedora/demo:c1"/>
          <myns:isPartOf rdf:resource="info:fedora/mystuff:100"/>
          <myns:owner>Jane Doe<myns:owner/>
  </rdf:Description>
</rdf:RDF>
```

The above RDF fragment indicates that the Fedora digital object identified as "info:fedora/demo:99" is the subject that is being described (as specified by the rdf:about attribute). This object has two relationships to other digital objects. It has an "isMemberOfCollection" relationship to the object identified as "info:fedora/demo:c1" which is a Fedora object that represents a collection. This collection object is considered the target of the relationship assertion. The second relationship assertion that is just like the first one, except that it asserts that the subject is a part of another Fedora digital object, "info:fedora/mystuff:100". The third relationship asserts that the digital object has owner "Jane Doe". Note that the object of this relationship is a text literal, not a URI.

To ensure the integrity of relationship metadata so that it can be properly indexed by Fedora, the Fedora repository service validates all RELS-EXT Datastreams and enforces the following constraints on the RDF:

1. The subject must be encoded as an <rdf:Description> element, with an "rdf:about" attribute containing the URI of the digital object in which the RELS-EXT Datastream resides. Thus, relationships are asserted about this object only. Relationship directionality is from this object to other objects.
2. The relationship assertions must be RDF properties associated with the <rdf:Description>. Relationship assertions can be properties defined in the default Fedora relationship ontology, or properties from other namespaces.
3. Prior to 2.1, the objects of relationships were restricted to other Fedora digital object URIs. This has since been relaxed so that a relationship property may reference any URI or literal, with the following exception: a relationship may not be self-referential, rdf:resource attribute must not point to the URI of the digital object that is the subject of the relationship.

4. There must be only one <rdf:Description> in the RELS-EXT Datastream. One description can have as many relationship property assertions as necessary.
5. There must be no nesting of assertions. Specifically, there cannot be an <rdf:Description> within an <rdf:Description>. In terms of XML "depth," the RDF root is considered at the depth of zero. The must be one <rdf:Description> element that must exist at the depth of one. The relationship assertions are RDF properties of the <rdf:Description> that exist at a depth of two.
6. Assertions of properties from certain namespaces for forbidden in RELS-EXT. There must NOT be any assertion of properties from the Dublin Core namespace or from the FOXML namespace. This is because these assertions exist elsewhere in Fedora objects and may conflict if asserted in two places. The RELS-EXT Datastream is intended to be dedicated to solely object-to-object relationships and not used to make general descriptive assertions about objects.

Resource Index - RDF-based Indexing for Digital Objects

Yes! The Fedora repository service automatically indexes the RELS-EXT Datastreams for all objects as part of the RDF-based Resource Index.

This provides a unified "graph" of all the objects in the repository and their relationships to each other. The Resource Index graph can be queried using RDQL or ITQL which are SQL-like query languages for RDF. The Fedora repository service exposes a web service interface to search the Resource Index. Please refer to the Resource Index documentation for details.

Section 4: The Content Model Architecture

Introduction

A major goal of the Fedora architecture has been to provide a simple, flexible and evolvable approach to deliver the "essential characteristics" for enduring digital content. Whenever we work with digital content, it is with an established set of expectations for how an intellectual work may be expressed. With experience we develop "patterns of expression" that are the best compromise we can craft between the capabilities of our digital tools and the intellectual works we create in digital form. We store our digital content with the expectation that all the important characteristics of our intellectual works will be intact each and every time we return to access them, whether it has been a few minutes or many years.

We also want to communicate our intellectual works effectively to others. To do this, we have an expectation that the important aspects of our digital works are delivered accurately to users accessing them. Often we teach the same patterns of expression used to create our works to aid our users in comprehending them. And the same patterns may be used once again to enable collaboration for creating new or derived works.

Librarians, archivists, records managers, media producers and the myriad other author and publishers of intellectual works have long used content patterns calling them, for example: books, journals, articles, collections, installations. The term "content model" originated with the publishing community to describe the physical structure of the intellectual work. Users and collectors of intellectual works often add value by organizing works, annotating them, and producing ways to find them or information within them.

With the development of digital technology, publishers, users and collectors have applied these same patterns to this new media with some initial success. But like many advancements, digital technology enables ways to create and use content in ways that cannot be achieved with physical media. While it is beyond the scope of this document to discuss the many emerging aspects of digital content technology and its use, there are two key subjects that help in understanding the requirements of the architecture presented here.

First, reusable patterns such as content models can reduce the effort to create or capture, ingest, store, manage, preserve, transform, and access digital content. For example, content models can be used for content classification to facilitate discovery. Other uses include content validation usually at ingest or modification. Content models can be, for example, used as a template when content is created to generate user interfaces, drive workflows, describe content components, or to manage policy enforcement.

Second, digital content is not defined by its format or technology, and may also incorporate functions as a part of its nature. For example, when we access a digital picture we experience its resolution and color fidelity; the technology that delivers that experience is irrelevant. A spreadsheet or a video game is hardly the same thing if there is no software to make it function. Those features which must be present to provide an authentic experience of the digital content are called the "essential characteristics" and they can be captured in a content model to ensure durability of the experience as format and technology changes over time. Those same characteristics also facilitate sharing by describing the nature of the content and the ways it can be used.

Similar needs have been noted in the software engineering community for the development of complex computer systems. Often, organizational information outlasts the technology used to create and access it. Corporate mergers and breakups raise havoc with the integration of company information technology infrastructures. The same concepts that have been developed to satisfy agile IT infrastructures can help provide solutions for creating, accessing and preserving content. In this document, we introduce the fundamental concepts of Fedora's Content Model Architecture or CMA. The CMA builds upon the basic building blocks established by previous versions of the Fedora architecture, restructuring them to both simplify use while unlocking their potential.

We use the term "content model" to mean both:

1. Content structure as used by publishers and other traditional content-related professions
2. A computer model describing an information representation and processing architecture

By combining these very different views, CMA has the potential to provide a way to build an interoperable repository for integrated information access in our organizations and to provide durable access to our intellectual works.

As we introduce CMA concepts, we will discuss the rationale behind the design decisions. This is only the first generation of the CMA and, like the rest of Fedora, we expect it to evolve. An understanding of design decisions behind this "first-generation" CMA is a key element for community participation in future generations of CMA development. Most important is an understanding of three significant and interrelated developments in software engineering: (1) object-oriented programming, (2) design patterns, and (3) model-driven architectures. It is beyond the scope of this document to discuss any of these developments in detail but we will make reference in this document to aspects of them which inform the design of the CMA.

Content Model Architecture Overview

The Content Model Architecture (CMA) describes an integrated structure for persisting and delivering the essential characteristics of digital objects in Fedora. In this section we will describe the key elements of the architecture, how they relate, and the manner in which they function. The original motivation for the CMA was to provide a looser binding for Disseminators, an element of the Fedora architecture used to stream a representation to a client. However, the CMA as described in this document has encompassed a far greater role in the Fedora architecture, in many ways forming the over-arching conceptual framework for future development of the Fedora Repository.

The Fedora application community developed a number of clever approaches to add content model-like capabilities to Fedora. The CMA formalizes some of the "best of breed" techniques gained from further research and from our application community. Two primary ways of thinking about content models have emerged and both are supported by the CMA. The first approach is focused on using complex single-object models and is commonly called "compound." The second approach is to use multi-object models and is commonly called "atomistic" or "linked."

Prior implementations of the Fedora Repository utilized a set of specialized digital objects as a functional and persistence framework. All of these objects conform to the same basic object model. Digital objects in CMA are conceptually similar in prior versions of Fedora though some important implementation details have changed. Fedora still implements a compound digital object design consisting of an XML encapsulation (now FOXML 1.1) and a set of bitstreams identified by the "Datastream" XML element. We can also assemble multi-object groups of related digital objects as before using semantic technologies.

In the CMA, the "content model" is defined as a formal model that describes the characteristics of one or more digital objects. A digital object may be said to conform to a content model. In the CMA, the concept of the content model is comprehensive, including all possible characteristics which are needed to enable persistence and delivery of the content. This can include structural, behavioral and semantic information. It can also include a description of the permitted, excluded, and required relationships to other digital objects or identifiable entities.

The content model is expressed in a modeling language and stored in one or more bitstreams like any other kind of content. It may be useful to think of the content model as one or more closely related documents that contain a machine processable model.

There will likely be more than one content modeling language; the CMA does not specify that there be only a single standardized one. One of the key aspects of Fedora is the expectation that over time all things will change and the same will be true of content modeling languages. There are many valid approaches to content modeling and we wish to enable innovation in this area. Guided by these observations, the CMA is designed to be a framework for developing and deploying content model-driven repository infrastructures. However, while it is a Fedora first principle to minimize required elements, there are a small set of features, described below, that the content modeling language must provide in order for Fedora to function.

Since we anticipate that a large number of digital objects will conform to the same content model, they may be treated as a class. While we need to be careful about analogies between "class" in CMA and "class" in object oriented programming languages or in semantic technologies, exploiting the notion of "class" is a major objective of CMA.

We must have a unique, unambiguous method to identify the class. For this purpose, in the CMA we have defined the "Content Model object," a digital object that both represents the notion of the class and can contain the content model. We use the identifier of the content model object (or CModel) as the class identifier. Following the rules of Fedora identifiers, the identifier of the CModel object can be encoded within a URI. We will describe the rationale for this decision in a later section but this approach provides two immediate benefits: (1) it provides a scheme which works within the Fedora architecture with minimal impact, and (2) it is compatible with the Web architecture, RDF and OWL. We can even build functionality using just the knowledge of the identifier without creating a content model. Having a uniform method for identifying a digital object's class maximizes interoperability.

The CMA does not require that digital objects explicitly conform to its architecture or explicitly declare any of its metadata elements beyond providing well-formed Fedora digital objects - unless you want to use the advanced features provided by the Fedora repository. The CMA uses a "descriptive" approach where the Fedora Repository will issue a run-time error for any operation it cannot perform. In most cases, you should still be able to disseminate bitstreams exactly as they are stored. CMA's dissemination approach is more consistent with the Web architecture, and provides a better balance between durable access to content and future innovations.

The minimum requirement to participate in the CMA is for a digital object to assert a relation to record its class' identity. Digital objects that do not explicitly identify their class are assumed to belong to a system-defined "Basic Content Model" which has a repository-defined reserved identifier. In CMA, you should use a digital object (see CModel below) as a way to register a "class" in a repository federation.

The remaining functionality is enabled by creating the content model and storing it within Fedora digital objects like any other content. This permits applications or the repository itself to disseminate the content model, interpret it and use it to provide functionality for the set of objects it describes. Content designers are free to develop content models (or even content modeling languages). Content Models and Content Model objects are designed to be shared. Digital objects having the same content model automatically form an interoperable information community. If the whole Content Model object is shared, keeping the object's identifier the same across multiple repositories, the community can easily extend across multiple organizations.

There are two basic approaches to using content models. First, the content and content models can be disseminated to applications able to interpret them to deliver the essential characteristics of the content. Disseminating the content model to external services can be also be used when creating new digital objects for ingest, validation, transformation and replication.

Alternately, the Fedora Repository can be used as a content mediation layer which interprets the content model to disseminate the content correctly. To accomplish this, the Fedora Repository must have access to code compatible with the Content Model. The compatible code may be added to the Fedora Repository using its plug-in architecture in combination with service mechanisms. While all the tools needed to plug in your own Content Model mediation are not complete in Fedora 3.0, this feature of the CMA will permit serving several generations of digital objects in the same repository reducing the need to update objects whenever there are new releases of Fedora. This will increase the durability of collections and enable longer periods between migrations of Fedora digital objects to new formats.

While the CMA does not force you to use a specific content modeling language, Fedora 3.0 contains a reference implementation that enables the Fedora Repository to operate much as it did in prior versions. The following sections describe CMA in more detail and provide instructions on how to use the reference content modeling language so you can create your own CMA compatible objects immediately. Over time Fedora Commons will support the development of one or more content modeling languages as part of solution bundles that may be used by the community with minimum effort.

Specializing Digital Objects

One of the basic elements of the Fedora architecture is the Fedora digital object. Every digital object stored in a Fedora repository is some variation of the same basic Fedora digital object model. In Fedora, digital objects containing data (Data object) utilize an XML element called the Datastream to describe the raw content (a bitstream or external content). In Fedora 2 and prior versions, digital objects containing data may also have contained Disseminators. The Disseminator is a metadata construct used by Fedora to describe how a client can access content within the digital object or remotely referenced by the digital object. If you only needed to access the raw content, default functionality was provided by the Fedora Repository which did not require that a Disseminator be explicitly added to the digital object. Unfortunately, the older design meant that the Disseminator was repeated in every Data object.

In Fedora 2 and prior versions, three specializations of the Fedora digital object were used: the Data object, the Behavior Definition (*BDef*) and the Behavior Mechanism (*BMech*). The BDef contained a description of the access interface which, in turn, was implemented by the BMech. In combination, the Fedora digital object model provides a uniquely flexible way to persist and access digital content. However, these features were not as simple to use as they could be. In particular, repeating the Disseminator in every Data object made changes to the access interface very difficult since every object had to be changed.

In the CMA, the Fedora digital object remains the model for all digital objects just as it was in prior versions of Fedora. However, for the CMA we define four specialized variations of the Fedora digital object (see Table 1).

Object Type	Code	Description
Data	Data	A container for content
Service Definition	SDef	A container for service definitions, an element of a content model
Service Deployment	SDep	A container for the service deployment bindings
Content Model	CModel	A container for content models

Table 1: Fundamental Fedora Object Types

Each of these specialized digital objects will be described in much greater detail below. The BDef and BMech have long been recognized as having reserved functions in Fedora and we often called them "control" objects because they contain data used to control object-specific functionality. CMA adds a new control object, the CModel. However, some of the data needed for object-specific functionality was located in the Data object (often just labeled generically as the digital object) in prior Fedora versions. In particular, Disseminators were defined in the Data object. The CMA eliminates the Disseminator and redistributes control data in a more logical and reusable fashion among the "control" objects.

Care should be taken in equating the names of object types in the Fedora 3.0 from the prior (Release 2 and earlier) Fedora architecture because there are some similarities to roles played by these objects in both the old and new architecture. However, the CMA is a major redesign so care should be taken not to automatically equate the roles of the control objects in the old and new implementations. In pre-release reviews with community members, we discovered that there were concerns about the potential of confusion so we have used a new naming scheme for Fedora 3.0.

Also, care should be taken not to equate the digital object as a container and any data (or metadata) it contains. Since the deployment, functional , persistence and information views of the Fedora architecture tend to intermingle more than in most architectures, it is easy to accidently conflate characteristics of the architectural elements across views. We will try to make the distinctions clear where needed.

Table 2 lists the supported relationships between fundamental object types in the CMA.

Origin	Target	Relation	Purpose
Data	CModel	hasModel	Identifies the class and, optionally, the object containing a model of the essential characteristics of the class
CModel	SDef	hasService	Identifies the object containing a model of the functional characteristics of class members
SDep	SDef	isDeploymentOf	Identifies the object containing a model of the functions being deployed
SDep	CModel	isContractorOf	Identifies the object containing a model of the information being deployed

Table 2: Relationships Between the Fedora Object Types

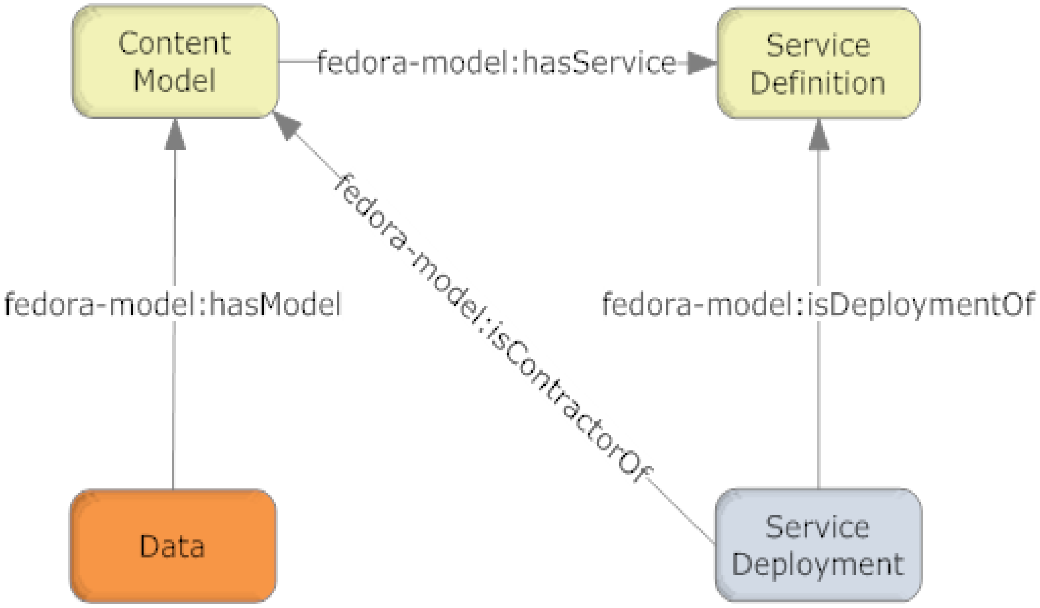

Figure 5: Fundamental CMA Relationships

Figure 5 illustrates the required relationships between fundamental object types in the CMA. In the CMA object serialization these relations are asserted as RDF statements in the digital objects' RELS-EXT Datastream. These relations are asserted only in the object at the origin of each arrow though typically these relations will be harvested and indexed within utilities such as Semantic Triplestores or relational databases to enable fast query over them, or into caches which permit rapid access to their functionality.

The "hasModel" relation identifies the class of the Data object. There may or may not be a Fedora digital object that corresponds to the identifier. If the identifier refers to an object it must be a CModel object and contain the base content model document. It is expected that many Data objects conform to a single Content Model (and have a relation asserted to the same CModel object). The Content Model characterizes the Data objects that conform to it.

The SDef object describes a Service and the Operations it performs. Defining a Service is the means by which content developers provide customized functionality for their Data objects. A Service consists of one or more Operations, each of which is an endpoint that may be called to execute the Operation. This approach is similar to techniques found in both object-oriented programming and in Web services. The CModel object uses the "hasService" relation to assert that its' class members provides a Service (and its associated Operations). A CModel is free to assert relations to more than one Service. A Service may be related to many CModels.

Deployment of a Service in a repository is accomplished by using the "isDeploymentOf" relation to the SDef object. The Service Deployment (SDep) object is local to a Fedora repository and represents how a Service is implemented by the repository.

Finally, the SDep object asserts the "isContractorOf" to indicate the CModel (effectively the class of Data objects) for which it deploys Services. This permits the SDep to access the Datastreams in the Data object and user parameters when an Operation is called for a Data object.

Section 4: Fedora Repository Server

Thus far, we have talked about the component parts of a Fedora repository, but the larger picture is also important. A repository is made up of digital objects, but in what context do those objects exist and how is it that users interact with them?

Fedora Server Architecture

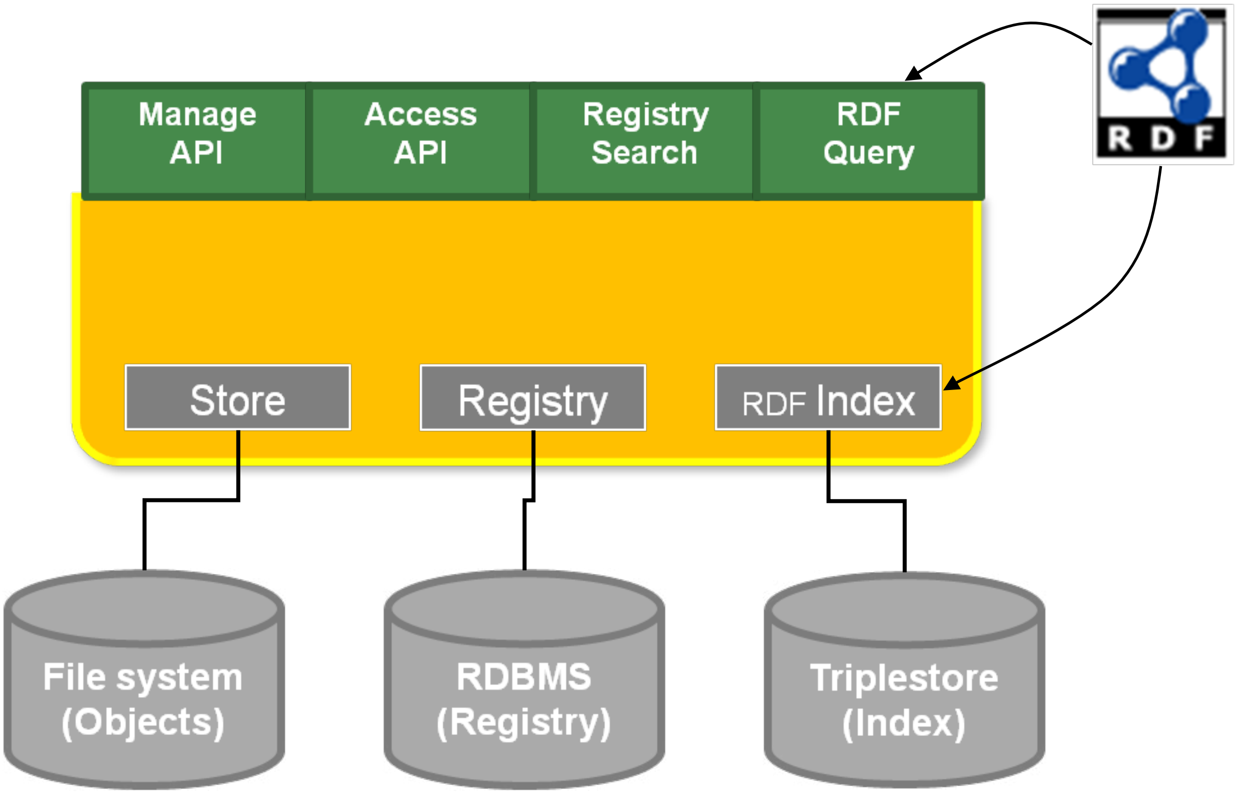

Figure 6: Fedora System Architecture (simplified)

This diagram shows in very general terms the structure of the entire repository. Users interact with the content of the repository by means of client applications, web browsers, batch programs, or server applications. These applications access the repository’s data by means of the four APIs by which Fedora is exposed: management, access, search, which are exposed via HTTP or SOAP, and the OAI provider API, which is exposed via HTTP.

Client and Web Service Interactions

This diagram gives another view of the larger context of a Fedora repository. Users perform common tasks such as ingesting objects, searching the repository, or accessing objects via client applications or a web browser. These client applications mediate this interaction with the repository via web services on the frontend, and on the backend, the repository interacts with web services to perform any data transformations that are requested by users. The transformed data is then passed back to the user via the frontend web services.

It is important to note that users only interact with the repository via the APIs, even though it may sometimes seem that they are interacting directly with an object, they are not.

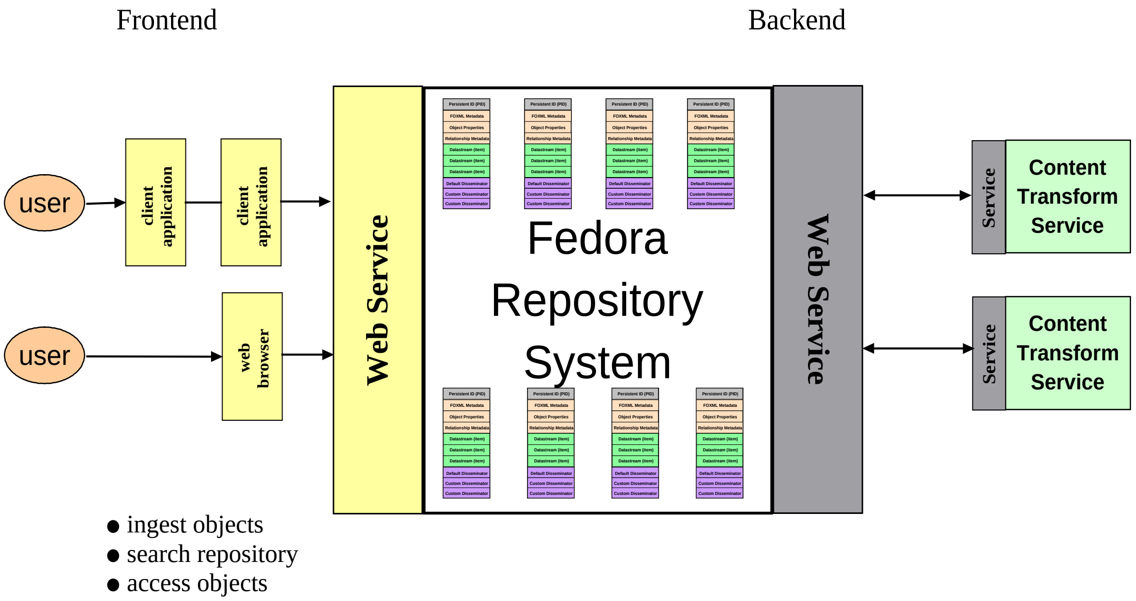

Figure 7: Client and Web Services Interaction