Good Afternoon. I'm Stefano Cossu, Director of Application Services at the Art Institute of Chicago.

This presentation wants to describe some use cases for a Digital Asset Management System that we are building for our Collections data.

The Art Institute of Chicago has a large collection of assets, which are accessed by many departments with very diverse needs. We have curators, conservators, Imaging staff etc.

In the near future we are planning to digitize the whole collection, thus increasing the volume and complexity of our data.

In order to manage the current data and build a solid workflow for the upcoming digitization project, we decided to set up a Digital Asset Management System.

This involves:

- Creating a repository which can scale up to large and complex datasets;
- Making this repository shared across departments and with outside clients with different access levels
- Move our application architecture from a large single system which performs many tasks to a message-oriented, asynchronous and distributed one
- Improve our current, custom-built Collection Management System to be at the front end of this architecture, to avoid disrupting the staff's user experience
- Improve the data flow across systems by standardizing the format of data exchanged.

After evaluating several commercial and Open Source DAM systems, we unanimously decided to use Fedora Commons.

The reasons are several:

- It has been around for a long time and has proven to be very stable even in very large and complex scenarios.
- It does not impose a User Interface. We have our front-end applications already and just want to plug them into a system with a good API.
- It accepts any content without making assumptions about what it is. We can model our content.
- It does few things and does them well. All other functionality can be delegated to external applications that can be added and replaced at any time.
- Most important, it is supported by a community of peer institutions with very similar goals as we.

After building a proof of concept in Fedora 3 and getting familiar with its features, we were presented with the hardest choice of this project so far: which major version of Fedora to choose.

It was especially hard because we are going to rely on it for a core system, not a side project.

Several major factors were on each side:

- Fedora 4 has some features that allow us a much greater freedom in building our architecture than Fedora 3
- On the other hand, Fedora 4 was still in early alpha when we had to make this decision and was a complete rewrite of the underlying code, so the stability of the system still had to be proved
- In addition, the first stable releases of Fedora 4 won't be as complete as Fedora 3, so we would have to hold off on certain features or find some workaround
- Finally, and this was the main selling point, we understood that going either way would involve a great effort on our end. Eventually Fedora 3 would be made obsolete, and we would have to transition to Fedora 4, and that would become another huge project.

- We decided to go with Fedora 4 with the condition that it would not be a dependency for any ongoing major project, so we can leave ourselves plenty of testing time.

The Fedora 4 features we are planning on relying on the most are:

- Federation of external resources: this will allow us to avoid migrating all our existing databases to Fedora
- Sequencers are a powerful tool to help make our workflow asynchronous
- The Fedora 3 REST API was excellent but the Fedora 4 one is even better!
- Scalability is greatly improved with clustered configurations
- Fedora speaks RDF natively, which makes it a great candidate as a linked data provider
- The storage engine relies on a solid JCR implementation which is much more powerful than the previous one

Some Fedora 4 use cases that I 'd like to present are:

- Building our DAMS architecture as a shared data pool – or better, a LAKE.
- Ingesting and processing large images in the background
- Content modeling for our Collections ontology

LAKE stands for Linked Asset and Knowledge Ecosystem and is planned to be at the core of our Collections information.

LAKE will contain shareable data, linked with pre-defined relationships that make up our Collections' ontology.

LAKE has no user interface of its own.

Instead, a number of pre-existing or newly built or acquired applications can be adapted to connect to the LAKE data via its API.

These applications make up the "gateways" for our staff or external users to access the data.

Different departments or special applications can access the data with different access levels.

Departments also have their own repository of non-shared data, which is related to the shared data.

Depending on their structure, some of these data can reside in the same repository, in a closed-off workspace that is only accessed by that department; or they can be stored in another system such as a relational database or an external service.

Also, we can federate external data and integrate them in our repository, or make our public data available through a query endpoint.

Another use case is the ingestion of batches of large files.

Imaging staff use a drop-box folder to upload their day's batch of high resolution images.

The drop-box is available as a mounted network filesystem on their workstations.

Once a batch is complete, the users fill in a form in their management application that creates a manifest file.

The manifest file, containing the list of images and their metadata, is ingested into Fedora.

Fedora parses the manifest file and starts a sequencer in the background, which creates nodes for the images and sets properties based on the metadata in the manifest file.

The high-resolution images are not ingested in Fedora, but moved from the drop-box to a permanent location in the imaging server.

High resolution images are made available in Fedora via filesystem federation

Another sequencer extracts metadata and sends the high-res files to an external image processor to generate derivatives. These are ingested in Fedora.

Another important feature of Fedora 4 is the underlying node structure.

This is built on JCR specifications and allows creating content models in a much simpler and flexible way than Fedora 3.

We can define “node types” using a specific syntax that allows multiple inheritance and mix-ins, effectively allowing to create a hierarchy that is very similar to an object-oriented architecture.

We can assign and un-assign mixin types to objects on the fly; using sequencers, we can also add behaviors to these operations.

For example, adding a “web published” mixin to a node can trigger the assignment of certain access policies, and the creation of “web” derivatives.

Node types also define the type of relationships that we can assign to a node.

This allows us to match the actual content model very closely to our ontology.

I hope this concise presentation was useful to give a brief insight of Fedora’s new capabilities.

We are still at an early stage of our implementation but with the valuable help of the Fedora development team we have been able to lay the foundations of a large and complex system.

Our team is open to discussing ideas, strengthening the Fedora community and promoting Fedora in the museum sector.

Thank you for listening.