# Flexible Access Control, Federated Identity and Heterogeneous Metadata Supports for Repositories

Chi Nguyen, James Dalziel
MELCOE,
Macquarie University
{chi,james}@melcoe.mq.edu.au

Steve Cassidy
Centre for Language Technology,
Macquarie University
steve.cassidy@mq.edu.au

## Abstract

*In this paper, we present a new framework complete with implementation, for a digital repository that will address some of the most difficult issues facing repository managers today: how to enable federated identity access, rapidly changing access control requirements, and the management of multiple metadata standards for different types of digital objects. Our work draws together leading industry standards in the area of authentication, authorization, and metadata management, and apply them in a new and innovative way to the repository landscape. As a demonstration, we apply our work to a speech annotation research project which makes use of a repository to manage its culturally sensitive data.*

## 1 Introduction

Repositories have become a critical component for institutions due to their role in the preservation and dissemination of information, both to the general public as well as restricted audiences. With their increased role, critical new requirements have emerged that must be addressed by repository developers. One is the need to enable restricted access across institutional boundaries. A common example is one where researchers from one institution working on a joint project with partners from different institutions, requiring access to restricted data at one or more of the partners' institutional repositories. In these situations, the researchers would like to be able to use their home institution login credentials, and not have to create a new username and password for each of the repositories that they require access to. This requirement is often referred to as support for federated identity for the repository, and single-sign-on for the end user.

Repository managers have also found that access control policies for their repositories need to change over time. Currently most repositories have hard-coded criteria upon which their access control is decided. If the new requirements happen to fall outside the scope of existing criteria, the repository will need to be re-designed. An example is a repository housing, amongst other things, student theses, and is then given a new requirement that all theses less than one year old are to be embargoed from the general public. If the repository authorization logic was not designed to take into consideration of the submission date of the thesis, its code, design and/or authorization database schema will need to change. This is a costly exercise, and in most case beyond the capability of most repository managers. A better approach is to define the authorization policies separate to the internal workings of a repository, but still have them enforced at the appropriate check points.

Another emerging requirement for repositories is how to support for multiple metadata standards, arising from the need to house more and more types of digital objects. Most of these would often have its own specific metadata schema. While most current repositories typically support one of the well known metadata standards such as Dublic Core (DC) [1], Metadata Object Description Schema (MODS) [2], or MARCXML [3], if the repository is required to support a new metadata standard (e.g. MIX [4] for digital images), or their own user-defined metadata schema for their particular research area, the repository manager would need to delve into the code of the repository, develop new input forms as well as the associated validation logic for the new metadata. Again, this would not be possible for most repository managers. What is needed instead, is a GUI framework whereby new metadata support can be added by the repository manager, and made available to the end-user with proper input forms with validation. It is critical that the framework should not require any code modification of the repository for this support.

To address the aforementioned requirements, we have developed a repository call Muradora [5]. It is built upon the Fedora [6] repository framework, which was chosen due to its service-oriented architecture (SOA), scalability, and world-wide popularity. With Muradora, we developed a new pluggable authorization module for Fedora which enables flexible access control policies to be enforced without any code change to Fedora itself. The module is based on the

OASIS XACML [7] standard that was created to address the issue of how to handle flexible access control requirements in such a way that it can be “de-coupled” from the end system's implementation. The new authorization module also has the added advantage in that if a single Fedora instance needs to be made available via multiple interfaces (e.g. different customized portals for different end users), one can still have the same consistent access control across all of these interfaces.

But most importantly, Muradora offers an enterprise-ready web GUI for Fedora; a critical component that is sorely missing from the current Fedora out-of-the-box offerings. A key feature of the GUI is an intuitive access control interface which allows flexible access control criteria to be made by the end-user – not just repository manager or librarians – and yet the user is never exposed to any of the complexities that accompany the management of XACML policy documents. Equally important is the new GUI framework that allows new metadata standards to be supported by Muradora in a pluggable and re-usable manner. The result is a generic repository interface that can be customized to handle many different use cases, ranging from a traditional document-centric repository to one that houses large research datasets.

The rest of the paper is organized as follows. Section 2 discusses how federated identity is made possible in Muradora. Section 3 addresses the issue of flexible access control. This is followed by an overview of Muradora web interface implementation in Section 4, focusing on the pluggable and re-usable metadata input and validation support. We examine the application of Muradora to a speech annotation project which has complex access control requirements for its culturally sensitive data in Section 5. We conclude in Section 6.

## 2 Federated Identity and Authentication with Shibboleth

The need for federated identity and single-sign-on capability for applications has led to many countries around the world to develop a trust fabric (also referred to as a federation) within their respective country. Institutional applications can rely upon the authentication servers within these federations to obtain trusted assertions about user attributes, and therefore make the appropriate authentication and authorization decisions. Some examples of the federations that have been set up are the Australian Access Federation (AAF) [8], InCommon in the US [9], and SWITCH AAI [10] in Switzerland. Most of these federations make use of Shibboleth [11] as the software for the exchange of user attributes.

Shibboleth, developed by the Internet 2 project, is a well-known open source implementation of the web-browser profile of the Security Assertion Markup Language (SAML) standard [12]. Its main goal is to provide Web Single SignOn (SSO) for end users within and across institutional boundaries, and the secure exchange of these users attributes between the trusted end systems.

However Shibboleth cannot be utilized directly by Fedora to provide federated access, since most calls to Fedora is typically done using the web service interface from a web GUI. As a result, most deployments which try to integrate Shibboleth with Fedora do it by only enabling Shibboleth on the web GUI – not Fedora. This implies that access control policies and their enforcement must also reside within the web GUI implementation. This would require that all accesses to the Fedora repository to be restricted to this single web GUI interface, limiting the usefulness of Fedora as a service-oriented architecture. The reasons for the restriction are:

- If a persistent opaque identifier for the user (such as eduPersonTargetedID [13]) is used by Fedora, it is not possible for different “shibbolized” GUIs to receive the same persistent opaque identifier and pass that on to Fedora.
- To enable a consistent access control enforcement, if multiple interfaces are used to access Fedora, they all would need to have the same synchronized access control policies for Fedora.

To overcome the above problems, we implemented pluggable modules that allow Shibboleth to be used without requiring code changes to Fedora. More importantly, they allows multiple interfaces to co-exist with the same instance of Fedora.

The two pluggable modules are call Delegated Attribute Retriever (DAR) and Authenticated State Manager (ASM). The DAR is a filter/interceptor which sits in front of the web GUI. It checks if a session has been established with the browser and if not, redirects the user to the ASM. It gets back from the ASM a username and password, unique to the current user session, which it uses to establish web service communication to Fedora.

The ASM is a “shibbolized” module on the Fedora server. Its job is to trigger the Shibboleth login process and gets from the user's identity provider their attributes. The attributes in conjunction with a generated random username and password are then stored in a local LDAP which Fedora itself uses to perform authentication. The random username and

password are communicated back to the DAR. A complete sequence diagram for the whole federated authentication process followed by a search operation is shown below:

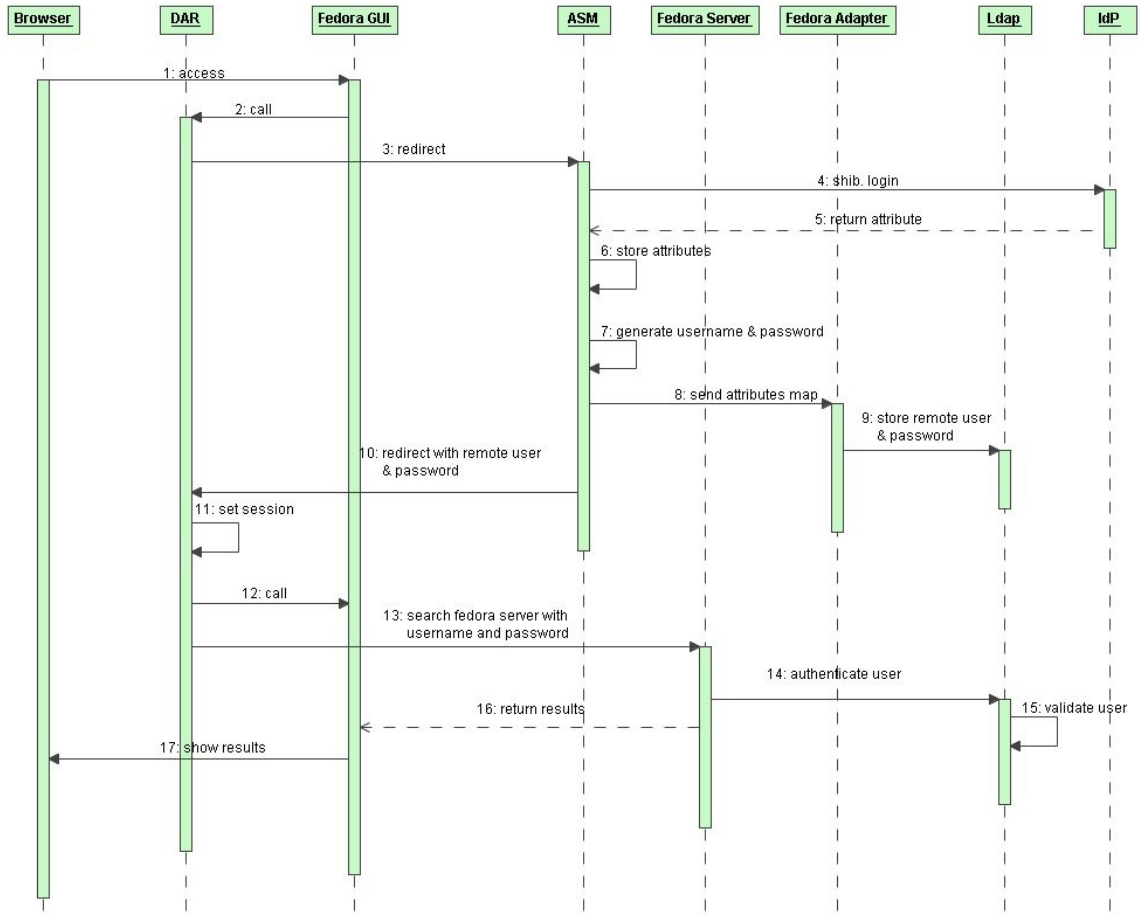


*Figure 1: Sequence diagram of the DAR and ASM modules for Shibboleth authentication*

The advantages that the new modules bring are:

- Federated identity is now supported along with the use of opaque persistent identifier for the end user.
- Only one Shibboleth component is needed per Fedora server, irrespective of the number of web GUIs talking to that server.
- The modules are fully pluggable and no code changes are needed to Fedora to support federated authentication. Fedora only needs to enable the "Basic Authentication" method for its web service interface; this is already supported natively by Fedora.

# 3 Flexible Access Control

## XACML Overview

According to OASIS, eXtensible Access Control Markup Language (XACML) was chartered *"to define a core schema and corresponding namespace for the expression of authorization policies in XML against objects that are themselves identified in XML. [...] XACML enables the use of arbitrary attributes in policies, role-based access control, security labels, time/date-based policies, indexable policies, 'deny' policies, and dynamic policies — all without requiring changes to the applications that use XACML."*

XACML policies are expressed in terms of *Actions* performed by *Subjects* on *Resources* within an *Environment*. *Subjects* are users of the system which can be specific individuals, or groups of individuals identified by a particular attribute (e.g. their roles). A *resource* is the item being protected which in a repository context can be anything from a

collection, a digital object or even a part of a digital object. The *action* is any action that the user might perform on the resource such as viewing, editing, accessing meta-data, deleting etc. The environment is additional information associated with the request that is needed by the XACML evaluation engine in determining the authorization decision, such as geospatial and date/time information.

Conditions expressed in an XACML policy consist of one or more rules each of which might allow or deny access to the action on the resource. The rules can be arbitrarily complex combinations of conditions expressed in terms of properties of the *subject*, the *resource*, the *action* and the *environment*. An overview of the key components for an XACML policy is shown in Figure Error: Reference source not found (note. *Environment* attributes is often expressed as part of a Condition hence is not show in the diagram), with an example policy that says *"allow read access to all resources if the current user is a staff member at Macquarie University",* broken down to the respective elements.

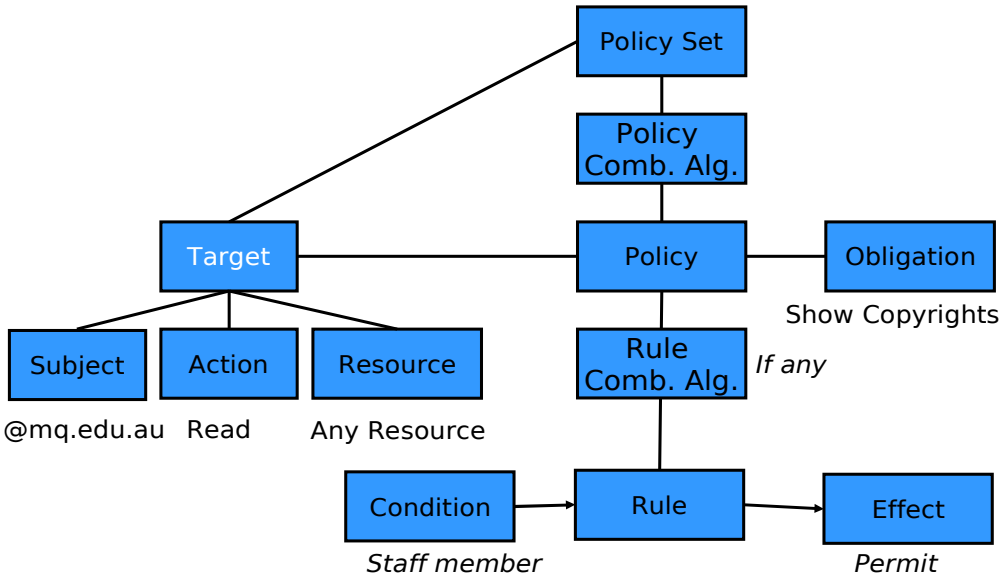


*Figure 2: Overview of key XACML Policy Components*

## Fedora and XACML

Recognizing the needs for flexible access control, Fedora has recently adopted XACML for specifying authorization policies. It uses the free SUN XACML implementation [14] as the evaluation engine. However, the current Fedora implementation of XACML authorization suffers from the following:

- There is no management interface for policy files which are kept on the file system. User must manually manage these policies. This is not scalable once the number of policies increases into the hundreds.
- Editing and creating of XACML policies must be done by hand. Even for users who understand the complex XACML standard, mistakes can still easily occur without sufficient validation and verification support.
- The enforcement points of the XACML policies are deeply embedded in Fedora code. This means that any requirement to support new access control criteria requires modification of Fedora's code itself.
- There is no hierarchical enforcement of policy. A policy for a given collection does not filter down to objects belonging to that collection. Therefore, one would need to maintain the link from each object in a collection, and its sub-collections, if one wants to apply the same access control policy to that collection. Again maintaining these links is also unmanageable even for a small size repository.
- When changing a policy, there is no way to know which objects it applies to without searching through the entire repository. This makes updating access control policies a very time consuming exercise.

Due to the limitations above, few Fedora deployments make use of the XACML feature; and even then its use is limited to administrators or repository managers, not end-users.

### Extended XACML Supports

We also make use of the SUN XACML engine for our implementation. However, it is clear that there needs to be a better management interface for policy files. To that end, we utilized DB XML [15] which is a free XML database from Oracle. By storing policies inside DB XML database rather than the file system, we now have a much more powerful interface through which to query, update and delete policies.

A new module for the XACML engine to interact with an XML database was also developed. Thus for a given XACML request, the XACML engine can query the database and get back only the applicable policies for that request. This is in contrast to the default XACML engine which requires all the policies to be kept in memory, thus limiting its scalability.

### Authorization Interceptor Pattern

We developed a new authorization architecture based on the interceptor pattern to replace the current Fedora's one. A simplified diagram of the architecture is shown below:

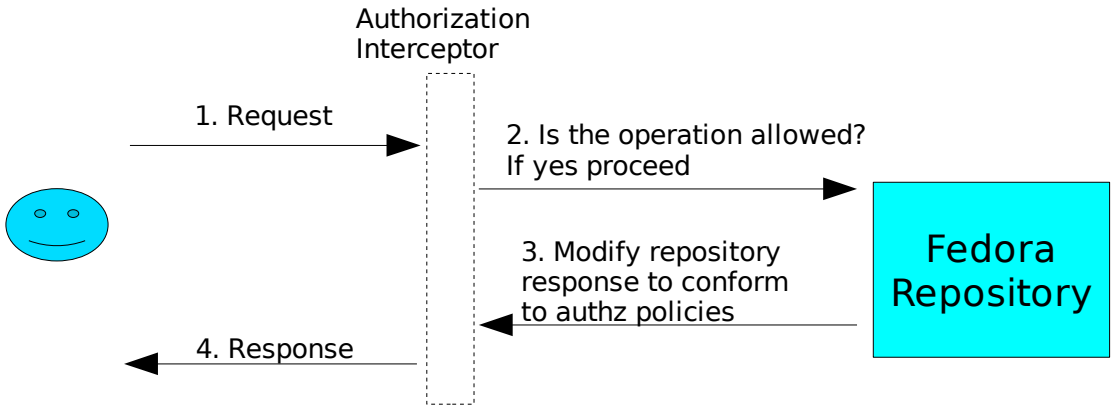


*Figure 3: Authorization interceptor pattern for Fedora*

With the interceptor pattern, no authorization is done inside Fedora itself. Instead incoming requests are intercepted and only authorized requests are forwarded to the repository. The interceptor can also intercept the response from the repository, perform authorization on the response before returning it to the user. A typical example where filtering the response is needed is the search operation where one might want to filter out search results that the user does not have permission to view, before returning it to the user's browser for display.

Currently there are three main interfaces to the Fedora repository: the REST interface, the RI-Search, and the web services interfaces (API-A and API-M). In the first and second instances, the interceptors were implemented as servlet filters. For web services, the interceptors take the form of AXIS handlers. The key point to note is that even though different interceptor implementations are needed for different interfaces to Fedora, they all share the same XACML engine and policy database, thus ensuring a consistent access control irrespective of the interface that is used to communicate with Fedora.

The interceptor is also where hierarchical enforcement of collection-based policies is performed. By supplying the XACML engine with the information of the parent collections, the XACML engine is able to retrieve all the applicable policies of the parents if they exist and apply them. We adopt a lowest-level precedent algorithm where policies at the datastream level takes precedent over policies at the object level, which in turn takes precedent over polices at the parent collections. This is an intuitive algorithm that is often employed in applications that have a hierarchical architecture, e.g. setting file permissions in Windows Explorer.

Finally, it should also be noted that the new interceptor implementation can be deployed on existing Fedora deployments without requiring any code change to Fedora itself.

# 4 Muradora

Muradora is a new web GUI for Fedora that has been developed to demonstrate the new federated identity and flexible access control modules described above. It is built on the well-known enterprise and extensible Java Spring framework. It has all the common features that one would expect of a repository such as basic search, full-text, faceted search (through the use of SOLR [16]), and versioning supports. What differentiates it from other repositories is the ability to specify complex access control criteria through an intuitive user interface, and the support for multiple heterogeneous metadata standards.

## Access Control GUI

An AJAX-based GUI was developed for end-users to specify access control policies for a datastream, object or collection. The GUI also provide the user with feedbacks on the current policies for a given datastream, object or collection. These feedbacks include inherited policies from the parent object and/or collections.

Behind the scene, when the user submit their changes, the GUI generates all the necessary XACML policies and stores them in the DB XML database via a web service interface. When providing feedbacks to the user, it queries the database for the necessary policies, then parse and present them to the user. Through this GUI, the end users never have to look at or edit any raw XACML policy. Figure 4 and 5 show two screen shots for the access control interface: one for specifying simple access control, and the other for a more complex access control policy with many different criteria. In both cases, the users is never exposed to the raw XML file of the XACML policies.

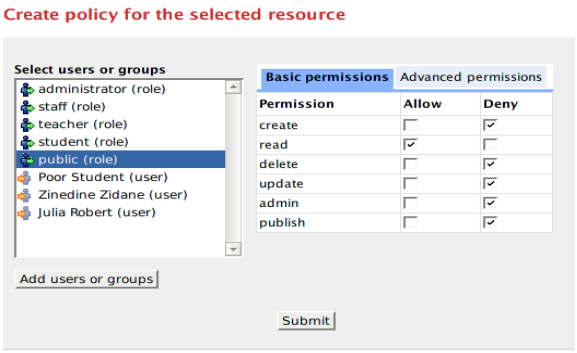


*Figure 4: Simple access control GUI*

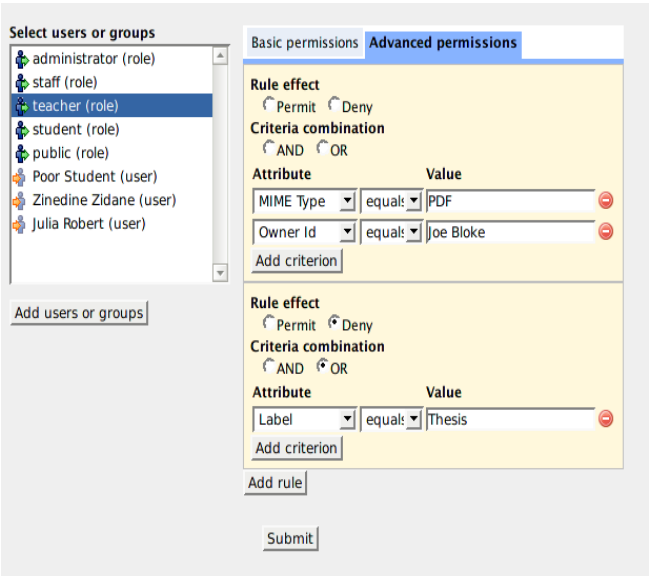


*Figure 5: Complex access control GUI*

## XForms Framework

XForms [17] is a W3C standard, designed to overcome the limitations with current HTML forms. In particular, it provides users with better dynamic feedbacks during form input, and validation of the input data according to the metadata schema.

By adopting this XForms technology, when there is a new metadata standard that needs to be supported by the repository, one can simply write the necessary XForms script, and plug that into the GUI framework without any code modification of the GUI itself. The reusability comes from the fact that if the metadata is a popular one, and someone has developed an XForms script for it, one can simply re-use that same script in Muradora.

The XForms engine will take care of form input and validation, and upon the successful completion of a metadata input, we are given an valid XML that can be stored as a metadata datastream for that object. To remove the need for any special browser plugin support for XForms, Muradora uses the freely available Orbeon Forms [18] Server-Side engine.

Figure 6 shows a screen shot of an XForms for the Dublin Core metadata. It should be noted that with the XForms technology, user can add additional repeating or nested elements without requiring any page refresh.

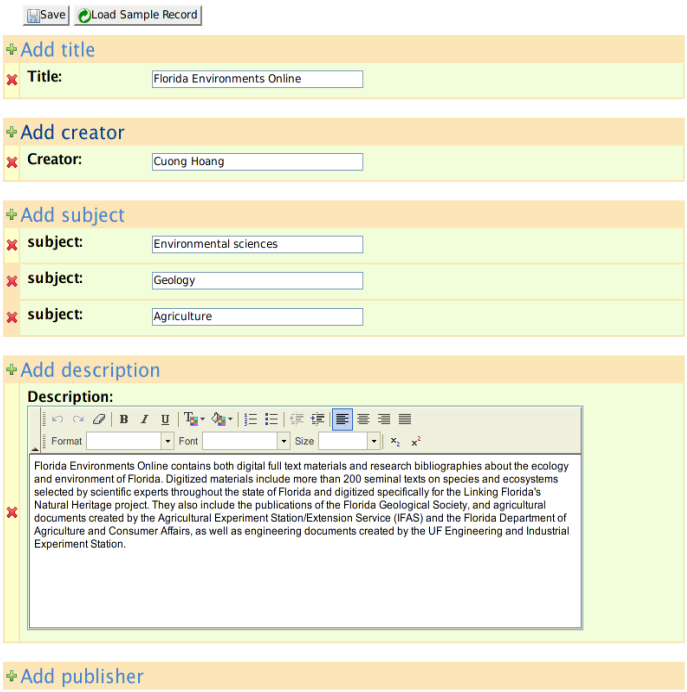


*Figure 6: A screenshot of an XForms for metadata input*

# 5 A Muradora Application: Managing Cultural Sensitive Data[1]

The Centre for Language Technology carries out research in the area of speech annotation. Often when dealing with collections of culturally sensitive language data, issues of access control are very important; researchers often take a conservative approach and refuse all access to data that contains some culturally sensitive material. This can place unnecessary restrictions on access to other parts of the data collection. What is required is a way to provide fine-grained access control to language resources such as annotations or parts of a data recording, and preserve the cultural sensitivity of the material.

## Data Example

As an example of data that would benefit from more fine-grained access control we consider a field recording of songs and dialogue recorded in 2007 by Linda Barwick from the PARADISEC project [20]. Some of the songs belong to a third person who is recently deceased (even though the person singing them on this recording continues to have rights to perform them). To observe local community restrictions on hearing or seeing cultural objects associated with the deceased person, these songs need to be restricted for an initial period of say 2 years, to be reviewed thereafter. This restriction will eventually be lifted after the deceased person's family has organized the required ceremony.

This data is annotated by the researcher to identify the start and end of each song and each song is transcribed in interlinear text. The goal is to be able to provide access to all parts of this annotation and data except those parts that are restricted which should only be made available at an appropriate time.

## Web Based Annotation Store

We have been developing a system for accessing and updating language data over the web using an HTTP based protocol. This annotation store makes use of an RDF triple store as the back-end data store for annotations and extends the Annotea and Vannotea annotation schema to support the kinds of annotations required on linguistic data. For the

1 This work is still in the development stage, but we expect to have it complete by the time of the conference.

purposes of this paper, the main feature of this system is that both the source data (audio, video and text) and the annotations are available over HTTP using distinct URIs. For example, an audio recording might be returned via the URI: “http://example.org/corpora/data/sample/interesting0012.wav”

In our system, access to this file might be mediated by a caching HTTP proxy to allow faster access to a local version of the file. Using server extensions, sections of this recording can be accessed using the URI for time fragments standard: “http://example.org/corpora/data/sample/interesting0012.wav?t=15.2/18.7”. This example would result in the section of the audio file from 15.2 to 18.7 seconds being returned as a result of a GET HTTP request to the server. The important point for this paper is that access to any segment of the file is possible via a simple GET request to a URI and that the URIs for the complete and partial files are related, but distinct.

Annotations on data are also available via HTTP at various levels of granularity. For example, the full set of annotations on a recording might be retrieved using the URL: “http://example.org/corpora/sample/interesting0012/”. Depending on the form of the request, the result might be in an XML format suitable for an annotation tool or in HTML format suitable for browsing the annotations. Furthermore, elements of the annotation are also addressable; the annotation of the song we are interested in might be: “http://example.org/corpora/sample/interesting0012/ann1234”

The result of this request could again be an XML fragment or an HTML page displaying the region annotated. The system also supports creating and modifying annotations using HTTP POST and DELETE requests to these URIs.

### Utilizing Muradora and Its Authentication and Authorization Modules

Muradora is used to store the metadata associated with the recordings. These metadata includes the Open Language Archives Community Standard (OLAC) [19], as well as PARADISEC's own metadata schema for audio processing and editing workflow. Muradora also contains links for each object to the actual data recordings (currently stored at the APAC Data Center located in Australian National University), and links to the annotations server.

As a first step, an XForms script was developed by the repository managers of the PARADISEC project. This allows Muradora to handle the metadata input and validation of metadata for PARADISEC data. A batch ingestion process was then developed to ingest the complete data from the current PARADISEC repository (an custom built repository that is no longer maintained) to a new instance of Muradora. Muradora was also configured with its Shibboleth components and registered with the Australian Access Federation to enable federated access to the repository. This is important as potential users of this repository would be spread out across many different institutions.

XACML access policies are then developed, driven by the annotations placed on language data by the researchers collecting or curating the data. The song data referred to above would be annotated with a special tag marking it as being restricted until a given date. When a particular region of the audio recording is requested via HTTP, the XACML policy engine will be queried for an allow/deny decision. The XACML engine will in turn query the annotation server for any restricted annotations on the region of interest; if the region requested overlaps with a restricted region, access will be denied.

This is only one kind of access policy that could be implemented in this architecture, but it serves to show how a non-standard access control problem might be addressed. Other more common schemes such as only allowing certain groups of users access to data or allowing read but not write access to annotations are also easily expressed as XACML policies.

The advantages of this architecture are:

- Access control policies can be expressed in terms of the annotation on data as well as on the identity of the data itself
- Restricting access to only part of an audio or video file means that the remainder of the data remains useful to the research community while respecting the traditions of the language community
- Providing access to data over HTTP leverages well defined and optimized infrastructure
- Access control is truly flexible, and not embedded in the code of the server. Thus in the case of cultural sensitive information where the access control requirements can change rapidly, we can cater for that by updating and modifying the XACML policies without having to modify the code for the server.

## 6 Summary

In this paper, we outlined the case why repositories need to support federated identity, flexible authorization, and multiple metadata standards. We showed how this can be achieved by first enhancing Fedora's underlying authentication and authorization architecture. The additional modules can be plugged on top of an existing Fedora's deployment without requiring any code change to Fedora itself. We presented Muradora which is a new web GUI for Fedora utilizing these new security features. Muradora also provides a standard-based framework whereby new metadata schema can be supported; again without any code modification to the GUI. Finally we apply Muradora to a current research project in the speech annotation area requiring complex access control policies for its cultural sensitive data. It demonstrates that Muradora repository can be used in a wide variety of situations, from a traditional document-centric repository to a dataset repository. The latter is especially important to researchers in the humanity areas that have traditionally been overlooked by repository developers.

## Bibliography


1. "Dublin Core Metadata Initiative", http://dublincore.org
2. "Metadata Object Description Schema", http://www.loc.gov/standards/mods
3. "MARXML", http://www.loc.gov/standards/marcxml/
4. "NISO Technical Metadata for Digital Still Images Standards (MIX)", http://www.loc.gov/standards/mix/
5. "Muradora", http://www.muradora.org
6. Lagoze, C., Payette, S., Shin, E., and Wilper, C. 2006. "Fedora: an architecture for complex objects and their relationships", *Int. J. Digit. Libr.* 6, 2 (Apr. 2006), 124-138.
7. "OASIS eXtensible Access Control Markup Language (XACML)", http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
8. "The Australian Access Federation", http://www.aaf.edu.au
9. "InCommon", http://www.incommonfederation.org/
10. "SwitchAAI", http://www.switch.ch/aai/
11. "Shibboleth Project" - Internet2 Middleware, http://shibboleth.internet2.edu
12. "OASIS Security Services (SAML) TC, SAML V2.0", http://tinyurl.com/a3h5l
13. "eduPerson Object Class", http://www.educause.edu/eduperson/949
14. "Sun's XACML Implementation", http://sunxacml.sourceforge.net/
15. "Oracle Berkely DB XML", http://www.oracle.com/database/berkeley-db/xml/index.html
16. "Apache SOLR", http://lucene.apache.org/solr/
17. XForms 1.0, W3C Recommendations, http://www.w3.org/TR/xforms/
18. Orbeon Forms, http://www.orbeon.com/
19. "OLAC: Open Language Archives Community", http://www.language-archives.org

20. “The Pacific And Regional Archive for Digital Sources in Endangered Cultures (PARADISEC) Project”, http://www.paradisec.org.au/home.html