Project Information			
Project Acronym	HiH (Hydrangea in Hull)		
Project Title	Hydrangea: letting the repository flower		
Start Date	01/02/2011	**End Date**	30/09/2011
Lead Institution	The University of Hull		
Project Director	Chris Awre, Head of Information Management		
Project Manager & contact details	Richard Green, Independent consultant r.green@hull.ac.uk		
Partner Institutions			
Project Web URL	http://hydrangeainhull.wordpress.com/		
Programme Name (and number)	Repositories take-up and embedding (15/10)		
Programme Manager	Balviar Notay		

Document Name			
Document Title	Project Final Report		
Reporting Period			
Author(s) & project role	Chris Awre, Project Director Richard Green, Project Manager		
Date	31/10/11	**Filename**	HiHFinalReport-v10.pdf
URL			
Access	☐ Project and JISC internal		☑ General dissemination

Document History		
Version	**Date**	**Comments**
1.0	28/11/11	Submitted to JISC and JISCrte coordinator

Hydra in Hull

Hydra in Hull Final Report

November 2011

The Hydra in Hull Project

Project Director: Chris Awre (c.awre@hull.ac.uk)
Project Manager: Richard Green (r.green@hull.ac.uk)
Lead software developer: Simon Lamb
Usability and testing coordinator: Diane Leeson

The Hydra in Hull Project was undertaken by Library and Learning Innovation (LLI), with support from the Information and Communications Technology Department (ICTD), at the University of Hull. It was funded by the JISC Information Environment Programme 'Repositories take-up and embedding' strand.

Contents

Acknowledgements

The project team gratefully acknowledges the support of the JISC Information Environment Programme in funding this project. We are also grateful to the partners in the Hydra Project (at the University of Hull, Stanford University and the University of Virginia) for their contributions to the project especially in the realm of Fedora repository content modelling and to the staff at MediaShelf LLC who provided professional advice and training.

Our thanks, too, to the staff and students of the University who took part in testing the project outputs.

Executive Summary

The Hydra in Hull Project set out to take software code developed for the Hydra Project's 'Hydrangea' demonstrator package and use it as the basis for a much-improved digital repository interface at the University of Hull. In undertaking the work, we wanted to document and feed back to the community the experiences of such implementation, particularly in regard to its use of Ruby on Rails and the use of Hydra content models. It was our intention that, alongside this development work, we would work with our local colleagues to help embed the new ideas and approaches that it might require. In addition, we wanted to feed back our development work to the Hydra community world-wide and to contribute ideas and code for the implementation of future Hydra 'heads', thereby hopefully encouraging wider take-up of the Hydra model and associated technology.

Our approach to the project followed a pattern that we have used successfully in past JISC-funded projects; one which places the users at the centre of technical development so that the work carried out is actually relevant to their needs. This was not a 'clean slate' project: the University has had a digital repository since 2008. Thus, a further consideration was the need to manage a gentle transition of user experience from the old to the new. In designing and developing code for the new system we tried a number of techniques that were new to members of the team involving self-testing code and the use of a continuous integration server that would run these checks on each new build of code.

As the project got under way, it quickly became clear that a number of the approaches we had described in out Project Plan were not going to work out well in practice; it transpired that we needed to rethink our approaches both to design and implementation. In doing so, we took a rather different path through our work than the one we had intended but, nevertheless, achieved most of our goals within the project's eight month time frame.

At the end of the project we have a working, Hydra-driven user interface for those wishing to browse the repository and download content (hydra.hull.ac.uk); the content that they are allowed to discover and download is limited by multi-level security. Behind the scenes, the interfaces for creating and editing the repository content are not yet so well developed, partly due to an unforeseen absence of our developer at a critical time, but these will be completed post-project. These create and edit interfaces introduce an element of workflow that was missing in our old system, namely that of putting potential new digital objects through a quality assurance stage.

In undertaking this work we have extended and enhanced Hydra's guidance for designing digital objects within a repository and have contributed a number of significant improvements and enhancements to the Hydra code. We find ourselves in a good position to offer advice and guidance to others who might consider the use of Hydra as a repository solution. Locally, we have involved users and colleagues in the development work; our end users seem to have taken to the new interface without significant problems and content managers are keen to have access to a fully functional create and edit implementation as soon as possible.

Overall, we are pleased with the way this project has gone and hope that others can learn from our experiences, documented here, and will want to implement their own Hydra heads in order to exploit this powerful technology solution.

Background

"If you want to go fast, go alone. If you want to go far, go together" African proverb

The Hydra project[1] is a collaboration initiated in 2008 by Fedora Commons, now part of DuraSpace,[2] to investigate and work towards a reusable framework for multipurpose, multifunction, multi-institutional repository-enabled solutions. It is based on two fundamental assumptions:

- No single institution can resource the development of a full range of digital content management solutions on its own,
  - …yet each needs the flexibility to tailor solutions to local demands and workflows.
- No single system can provide the full range of repository-based solutions for a given institution's needs,
  - …yet sustainable solutions require a common repository infrastructure

The founder partners in the project were Stanford University, the University of Virginia and the University of Hull. The purpose of coming together was in recognition specifically of the first of these assumptions, and realising that we were better placed working together on how to address our digital content management needs rather than trying to do this alone. From the beginning a key aim was to enable others to join the open source Hydra project as and when they wished, and to establish a framework for sustaining the community as much as any technical outputs that may emerge.

The common technical link between the founding Hydra partners is their use of Fedora as the repository infrastructure.[3] The Fedora repository architecture allows for highly flexible management of many types of digital content. Whilst acknowledging this real strength, a key issue for Fedora has been the lack of a regular user interface, with different repository implementations frequently developing them locally (a flexible strength of its own, but one that has sometimes prevented adoption due to the development effort required). Hydra set out to develop a model that would enable the building of easy to use interfaces and workflows over a sound technical architecture, with the scalable ability to apply this to different content types and use cases as required: the concept of different Hydra heads to the common underlying repository, the Hydra body.

Some of the use cases highlighted during the project thus far are as follows:

- ETD management: a single PDF with possible auxiliary files
- Digitisation workflow: potentially hundreds of files, of different types
- Open access research outputs: single PDFs with self-deposit
- Dataset management: a variety of datasets of different types and sizes
- Image/video management: accommodating various formats of the same content
- Digital archives: multiple content types with specific arrangement requirements
- Institutional repository: multiple content types from different sources

[1] Hydra project, https://wiki.duraspace.org/display/hydra/The+Hydra+Project
[2] DuraSpace, http://www.duraspace.org/
[3] NB. This is not an absolute requirement. Others are now looking at implementing Hydra over other environments, particularly curation micro-services.

The basis of the Hydra project has not, though, been technical development, but a focus on how the repository could be used to address these multiple use cases in a way that allowed for software implementation in different ways. As such, a full definition of a Hydra head is the use case plus the software stack used to implement this. At the heart of Hydra is the way that content and metadata is structured within the repository. Fedora's digital object model[4] allows great flexibility in how this can be achieved, although such modelling principles could be applied in other systems. A temptation is to be very detailed in order to provide a strong structure for the repository. Whilst recognising that individual repositories may wish to apply this detail, Hydra has adopted a simpler approach that seeks to allow different types of content to be modelled using common building blocks. This has provided the basis from which others can develop.

Hull's involvement in Hydra stems from work carried out through the JISC-funded RepoMMan[5] and REMAP[6] projects, which sought to enable upstream interaction with a repository through the use of workflow. Tools to manage deposit into a Fedora repository came out of this work, and we also made use of a user interface development from Macquarie University in Australia, Muradora, when we launched in October 2008. Development of this interface has now ceased, highlighting a danger of a community failing to be built around a very good piece of initial software development. From this experience and our project work came recognition that the best way to sustain the interfaces we need was to work with others. Hydra seeks to address this by working together to enable the full CRUD (create, read, update, delete) set of interfaces and interactions with the repository based on the firm modelling of content, offering the ability to replace multiple user interfaces with one integrated one.

Notwithstanding this primary user interface onto the repository, the work carried out in the JISC-funded CLIF project,[7] on the integration of Fedora with SharePoint and Sakai, was taken forward using the same Hydra modelling principles; these developments can be considered as separate Hydra heads – different views and points of access onto a common repository. Closely related to this is our current adoption of the Converis research information system[8] and its integration with the repository.

It is one thing to model content in a sustainable and scalable way. It is necessary to show how this can be implemented, though, to demonstrate its value. The Hydra project, specifically Stanford in conjunction with MediaShelf LLC,[9] thus developed a Ruby on Rails-based implementation, Hydrangea,[10] that encapsulated the modelling principles and enabled repository interaction for institutional repository use, with particular emphasis on open access articles and datasets. Whilst

[4] See also DuraSpace wiki page on Fedora's digital object model, https://wiki.duraspace.org/display/FCR30/Fedora+Digital+Object+Model
[5] RepoMMan project, http://www.hull.ac.uk/esig/repomman/
[6] REMAP project, http://www2.hull.ac.uk/discover/remap.aspx
[7] CLIF project, http://www2.hull.ac.uk/discover/clif.aspx
[8] Converis, http://www.avedas.com/en/converis.html
[9] MediaShelf LLC, http://yourmediashelf.com/
[10] Hydrangea, https://wiki.duraspace.org/display/hydra/Hydrangea

our project started out with the Hydrangea codebase, that demonstrator package is now deprecated in favour of more stable, core Hydra functionality. Accordingly, throughout the rest of this document we shall refer consistently to 'Hydra software', 'Hydra code' and the like; we shall not attempt to make unnecessary distinctions. This has also led to us increasingly calling the project 'Hydra in Hull' (HiH).

The Hydra software works in tandem with related components as shown in Figure 1. Further information on these components is available on the Hydra project website[11] and in the technical documentation for this project. The HiH project has implemented all of these components around our Fedora repository as part of the development and embedding of this new way to use Fedora.

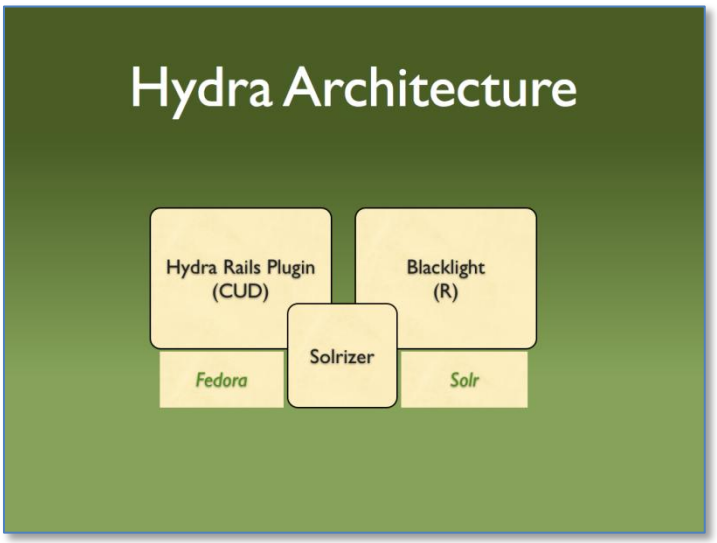

Figure 1: The outline Hydra architecture

Aims and Objectives

HiH set out to take the Hydrangea demonstrator package and underlying code offered by the Hydra Project[12] and to use it as the basis for a much improved repository implementation at the University of Hull which would replace the Muradora-based system in use since 2008.[13] Specifically we intended:

- To document and feed back the experiences and benefits of such implementation, including the use of Ruby on Rails for repository applications, and use of the Hydra models to the Hydra and wider repository communities

[11] Hydra technical components, http://projecthydra.org/technology/
[12] See http://projecthydra.org
[13] At http://edocs.hull.ac.uk until late December 2011

- To embed the use of Hydrangea within the local cataloguing team and other institutional users wishing to deposit content, and document changes to processes required
- To implement the Hydra models within systems integrated with the repository to aid consistency of content management using Hydrangea
- To make recommendations on the further development of Hydrangea
- To generate requirements for other Hydra heads

The Project Plan anticipated a number of clearly delineated stages to our work:

1. Preparation of a virtual machine environment
2. Conversion of existing repository content for Hydra compliance
3. Implementation of a 'read-only' repository using this converted content
4. Initial user testing and training
5. Process embedding
6. Extension of the new repository to full 'CRUD' capability
7. Further user testing and training
8. Roadmap development

This sequence was predicated upon the incremental development of the existing Hydra demonstrator package, 'Hydrangea'. Whilst this demonstrator was never offered as production-worthy code, it rapidly became apparent that the code fell well short of that standard and we have undertaken substantial work both to improve it and add to it and to feed back many of these enhancements to the Hydra community to become part of the subsequent 'HydraHead' core code.[14] It also quickly became apparent that our original plan to deal first with 'read-only' aspects of the new repository and then with other functionality later was, with hindsight, naive; in practice, the read-only code shares much in common with the create/update/delete code and the two needed to be developed side-by-side.

Another complication arose that we had intended at the outset that our Fedora digital objects should be structured according to guidelines already in place from the Hydra Project.[15] In the event, we decided to develop these guidelines yet further and this meant that the design was evolving until almost the end of the project and it was not appropriate to convert our old content until this design stabilised. Again, we believe that our developments in this area complement and enhance the Hydra guidelines and the work has been made available to the community.[16]

Despite this shifting and evolving landscape as Hydra matures, the project has largely achieved its aims, albeit the process undertaken was significantly different from that originally planned.

[14] We must acknowledge here a substantial contribution to this effort from MediaShelf LLC who have contributed code and training to the project under a separate University of Hull contract
[15] See: https://wiki.duraspace.org/display/hydra/Hydra+objects%2C+content+models+%28cModels%29+and+disseminators
[16] See: https://wiki.duraspace.org/display/hydra/University+of+Hull+implementation (in development)

Methodology

The overall methodology for HiH largely followed a pattern that has been used successfully in a number of previous JISC-funded projects undertaken at Hull. This approach places the user at the centre of technical development in order to ensure that the work carried out is actually relevant to the purpose at hand.

HiH was not a 'clean-slate' project; rather it built on the successful repository already operating at the University. Against this background we decided (unusually) that we should not undertake an initial user requirements survey as our users' likes and dislikes of the then current repository system were well understood. For this reason too, the initial implementation of Hydrangea was intended deliberately to echo some of the design of the existing system so that users would not experience a culture shock when coming to the new interface. It was then intended that, over time (largely beyond the period of this project), the design and functionality would evolve further to meet user needs and to incorporate appropriate enhancements provided by new technological developments.

Initial work undertaken revolved around two parallel aspects of development: one to establish a set of servers suited to enterprise delivery of the Hydra system (something that, at the start of the project, was still an evolving specification amongst the Hydra partners), and the other to decide how the Hydra guidelines for constructing Fedora digital objects should actually be implemented in Hull's new repository.

Phase one of the coding work had been intended to produce a read-only version of the new repository based on the code used in the Hydrangea demonstrator. In fact, quite a lengthy discussion took place before deciding to do this: an alternative Hydra codebase, developed at the University of Virginia, was also considered. The reader should bear in mind that at the time, in the spring of 2011, the Hydra partners were still working towards what might be described as 'definitive Hydra code'. Once our original plan was endorsed, work began to implement a local version of the Hydrangea code. The University was fortunate to be able to secure the services of MediaShelf LLC (through a separate contract) to help with this process. Our agreement was not just that MediaShelf should contribute to our software development process but that, as authors of much of the original Hydra code, they would simultaneously train our own developers in working with Hydra and with Ruby on Rails more generally. Weekly Skype calls were initiated – one for management purposes and one for the developers involved. Towards the end of the project a short, daily 'scrum' was also initiated via Skype to allow even closer contact between the developers enabling them to share out tasks on a much more frequent basis.

As noted elsewhere, it became quickly apparent that our intention to produce a read-only version of the repository divorced from its create-edit functionality was misguided; however we did wish to take advantage of some user testing as soon as possible. Thus the team concentrated on producing a first demonstration for staff and student end users centred on a very restricted set of content but showing a number of the features we felt would be desirable. This first user testing took place in June 2011 and produced very positive feedback.

Work then pressed ahead to add additional functionality to the system and to develop capability around further types of content. Another round of user testing and feedback was carried out in late August based on a partial export and conversion of content from the 'old' repository.

During the software development, which settled down to a series of weekly iterations, extensive use was made of Atlassian's JIRA project tracking tool[17] – a facility kindly made available to participants in the Hydra Project by DuraSpace. All code is held in a public Github repository for ease of management and sharing. In addition, the team used a Hudson (to become Jenkins) continuous integration server[18] to check the frequent builds. Our code contained integrated RSpec and Cucumber tests[19] for use with this system. Thus, code cannot make its way into the application until it has been tested and validated as working within the scope of the existing tests.

In the late summer, the team found itself in a position to begin the process of taking the entire content of the existing repository and transferring it to the new system. During this period our progress suffered a major setback due to an unplanned extended absence for our main software developer due to bereavement. Upon his return, in mid-September, we embarked on a period of intense activity in order to be able to launch the new repository for the start of the University semester on the 26th of the month. It is literally true to say that the task was completed with only a very few hours to spare.

As launched, and thus as at the end of this project, the new repository offers the end user a fairly comprehensive service across a wide range of content albeit there is a considerable number of, largely cosmetic, bugs still to address. However, behind the scenes, the provision for the creation of new content and editing content still lacks some significant elements and the level of differentiated provision by content type that we would wish. These issues are now being dealt with post-project as quickly as time allows. It is hoped that by Christmas 2011 the new repository will be fully operational in its role as a replacement for our Muradora-based system, although we anticipate that we shall be developing additional functionality and services for many years to come.

Implementation

Infrastructure design

One of the early elements of the project's work was to settle on an infrastructure design for Hull. At the time the project started there were very, very few Hydra heads offering production services and so we had very little experience elsewhere to draw on.

Following a number of conversations with technical staff at Stanford and Virginia, and in Hull's own Information and Communications Technology Department (ICTD) a basic configuration was

[17] http://www.atlassian.com/software/jira/overview
[18] http://hudson-ci.org/
[19] http://cukes.info/

identified. The architecture is implemented using virtual machines (VMs) rather than free-standing servers.

A three stage system has been set up: a development stage, a test stage and a production stage. Three software components of the Hydra stack are known to be resource-hungry and it was decided that these each merited its own VM although such division was unnecessary for the development server. Thus the production server group consists of three VMs:

- a VM for the primary Hydra code itself (Ruby on Rails and other delivery-related software)
- a VM to host the Fedora repository
- a VM to host the Solr indexing system

In addition use is made of a fourth VM in the University's centrally-managed MySQL cluster. The first of these (Ruby etc) runs Red Hat Enterprise Linux as an operating system whilst the other two run Windows Server 2008.

The test server has an identical configuration. The development server hosts the three components on a single VM. In all cases the digital content is held on the University's central storage area network.

Immediately prior to launching the production service for the Hydra repository the memory allocation was increased in the production cluster. The service was launched with 2GB memory for the Fedora test/production servers, 4GB for the Hydra test/production servers and 6GB for the corresponding Solr servers.

The technical configuration is more fully described in a separate project document.[20]

Digital object design

For good or bad, the 'F' in Fedora stands for 'flexible'. One of the places where this flexibility is manifest is in the building of digital objects in Fedora to hold content. The Hydra project has put a lot of time into developing a modular approach to building Fedora objects which retains much of this flexibility but yet allows adopters to follow common patterns and thus to adhere to a loose notion of standards compliance in order that the same UI software can be used across different repositories. The term 'Hydra-compliant' is gaining acceptability in the repositories community as an approach to structuring digital content.

Our approach to building Hydra in Hull was that we wanted our repository to adhere as closely as possible to the Hydra guidelines. We felt it important, as a founding partner institution for Hydra, to validate our own recommendations. This apparently simple decision had, though, some serious implications. Hull's institutional repository would comprise the most complex mix of content yet

[20] Green, Richard A. and Lamb, Simon W. (2011) *Hydra in Hull: Technical infrastructure and installation* Shortly to be available from the University of Hull digital repository at hydra.hull.ac.uk

attempted in a Hydra head. In attempting to deal with this complexity we found it necessary to adapt and further extend the Hydra guidelines. As with much else of our development work, these enhancements to the Hydra offering were fed back to the community and now form part of its central documentation.

In essence, current versions of Fedora anticipate that a digital object will subscribe to a content model, a 'cModel' in Fedora-speak. Hydra has suggested that these cModels be additive. Thus all content-bearing objects would subscribe to a cModel that said, in effect, the object had a single content datastream. So-called compound objects have multiple content bearing datastreams, but instead of having a completely different cModel Hydra recommends subscribing to two additive cModels: the declaration for the first (otherwise single) datastream, and a second that declares a largely arbitrary number of optional, additional datastreams (29 more in Hull's case). This has frequently been described as a 'Lego brick' approach with the additive cModels each a different brick.

This was well and good, but in implementing our code we needed to be able to distinguish between a single content datastream object that represented, for example, a presentation and one that represented, for example, a handbook in order that the information displayed to a user about each could be appropriately customised. To manage this we went down the road of 'coloured Lego bricks': a cModel to represent a presentation content datastream and a different one to represent a handbook content datastream. The models are identical except in name (colour) but that relatively minor difference allows us to treat them differently in the code. In addition, though, and accepting that the project would not have the time to produce the range of different cModels that we would ultimately want, we defined an underlying generic cModel which, for now, allows us to cope with a considerable range of our content types in a common fashion until such time as we can refine our approach further. If an object declares itself to be a 'newspaper', something for which we do not have specific capability at this point, the 'generic' code is used until such time as specific handling can be developed.

A second object design issue was that Fedora allows one to create compound objects, a single digital object containing one or more content datastreams – a simple, one content datastream object is a special case of this – or complex (often called atomistic) objects – a parent object with no digital content (only metadata) attached to one or more further objects, each containing a content datastream. This complex approach is particularly appropriate where the content objects might be re-used as elements of a different object; the compound approach is generally deemed appropriate where the content must be kept bound together for copyright reasons and/or the multiple content datastreams are the same intellectual content but represented in different ways. Hull needed to implement both approaches as appropriate to particular forms of content.

Other elements of the Hydra 'Lego-brick' approach deal with datastreams for metadata of one sort or another. Two such datastreams should be mentioned here:

MODS metadata

Although it is fairly straightforward to implement other schemas, Hydra generally recommends the use of MODS as the basic descriptive metadata for content. At the project's outset Hull intended to do this, moving away from the much more restrictive Dublin Core used previously, and anticipated supplementing it with specialist metadata where necessary: for instance to allow ETDs to be described using the UKETD_DC schema. Having taken this approach for some months we realised that it was easier to keep all the required information in MODS and then to create from it a UKETD_DC representation 'on-the-fly' should someone request it. With this approach, only one set of descriptive metadata needs to be maintained.

Content metadata

Each Hydra-compliant object contains a content metadata datastream which is a 'one-stop-shop' for the UI software to find a range of non-descriptive items that are needed for the display. Hydra's content metadata schema was developed originally at Stanford and owes elements of its structure to their particular, and highly evolved, information architecture. Hull has produced and implemented a modified version of this that should have wider applicability; this revised schema has been fed back to the Hydra community.

The results of our design work are that, in principle, we can create Hydra-compliant compound and complex objects, and deal with them on screen in customised ways appropriate to their particular content: thus the display screens for electronic theses and dissertations (ETDs) are significantly different from those for student handbooks. Equally, we have generic pages that can be used where this specialisation has not yet taken place or when we receive a new form of content that will subsequently require some coding work done to accommodate its particular needs.

Software implementation

Alongside work on the digital object design, work commenced on a software implementation. In practice there was a very dynamic interaction between the two development strands but they have been separated here for clarity.

An iterative approach was taken to this; initially the iterations were generally about a week long but towards the end of the project they became more frequent. As noted in the previous section, this process involved intensive use of JIRA project tracking software, a Hudson continuous integration server and a Github repository for handing code development. UI layout was largely handled using the Balsamiq software for screen design (which could be integrated with JIRA to associate designs with tickets).

The first period of work gave us rudimentary functionality around ETDs and presentations (complex and simple-compound content respectively) and this was built out to encompass a range of other content types (including the datasets promised in the Project Proposal). It became apparent that the project did not offer the time to develop customised behaviours for the full range of content types that the repository holds and thus, mid-summer, a switch was made to develop a generic handler –

effectively coping with a superset of metadata – that could be used where specialisation had not yet been possible.

The Hydra code developed for Hull as part of this project has been produced in such a way that our web pages do not use JavaScript. This ensures that at the base level we can address accessibility issues. A later phase of work (a move to HydraHead 3, see below) will allow us to layer over this base JavaScript-enabled pages that can offer a better user experience for those wanting to take advantage of it whilst maintaining accessibility through use of HTML5.

As part of the implementation, two rounds of user testing took place in which volunteers were asked to undertake a guided exploration of the developing system, from the point of view of academic end users, and to comment on it. The results of this testing were largely positive and the feedback from it influenced a number of changes.

Whilst we were working on Hydra in Hull, the rest of the world was also working hard and at the end of the project we find ourselves needing to update our software stack. Rails 2 has been superseded by Rails 3 and Hydra now has an official 'HydraHead 3' Ruby gem containing the essential elements of its code; a number of enhancements developed in Hull form part of this offering. This combination, which will be installed post-project, will allow us over a period of time to enhance our local Hydra implementation's functionality, including a move to HTML5 compliance.

Enhancements

We should note here, in brief, a number of enhancements to the Hydra code developed as part of this project but also fed back to the community.

Managed content

There are several ways to manage the storage of metadata and content in a Fedora repository. At the outset of this project the Hydrangea code could only exploit a very limited range of these; Hull's work has extended this. In particular, some of the Fedora metadata datastreams can now be managed independently in filestore (rather than being an inherent part of the XML that comprises a Fedora digital object) and this has reduced the size of our objects leading to lower memory requirements and better response times.

JMS listener

Hydrangea's code implementation, made the assumption that it was the only source of traffic to the underlying Fedora repository. In Hull this is not the case as objects can come from one of a number of other sources. The Ruby gem, Solrizer, that maintains the Solr index associated with Fedora is now triggered by a Java Messaging Service output from Fedora itself rather than directly by the Hydra code. This means that the Solr index, and thus the Hydra UI, reacts to changes in Fedora no matter where they originate: an object can be deposited in Fedora by a non-Hydra system but the Hydra UI will immediately make it available to users (rights permitting).

Sets

When this project started, Hydrangea did not support the notion of grouping objects into sets either for management purposes or for display purposes. Hull's work has added provision for both. Structural sets can be used to group together like objects for management purposes (say all the Physics PhD theses) whilst display sets allow users to identify groups of objects (say a particular image collection) and can provide a page explaining the context of this collection as well as providing access to it. Whilst not yet fed back into the Hydra core, this work is available for others to build on.

Workflow

Hull's initial work with repositories was rooted in the notion that deposit of content in a repository was part of a wider workflow. In implementing the 'create' elements of Hydra in Hull we have developed the UI around a two stage process – initial creation of a repository object which might take place within the Hydra environment (but which could occur elsewhere) and a quality assurance stage, only after which is the content made available to appropriate users. Again, we hope that this initial work might encourage others to adopt similar, workflow-driven approaches.

Full text indexing

Hull's 'old' repository provided full-text indexing of pdf documents. Hydra's Solrizer gem did not, initially, provide this facility but Hull has had it enhanced and the additional functionality is now available to the Hydra community.

Digital object conversion

Towards the end of the summer, following many tests, adjustments and refinements, we considered our Fedora content models stable enough that we could export all our content from the 'old' Muradora-based repository (running over Fedora 2.4.2) and convert it to be Hydra-compliant and able to run in a Hydra environment based on Fedora 3.4. This involved using conversion scripts which had been following the same cycles of revision and refinement as the cModels themselves.

Conversion was actually a two part process. The first part involved a small piece of bespoke software that would export objects in a particular part of the Muradora management tree. Thus one could, in principle, export all the ETDs, just the Physics ETDs, or whatever grouping was supported. It was thus possible to export content in batches that shared the same form of content and also, crucially, shared the same security settings governing who had what sort of access (if any) to them.

The second stage involved a further piece of bespoke software that used an XSLT script to take an exported object and reconfigure it in the new format, embedding links that would allow the actual digital content to be retrieved and transferred to the control of its new Fedora system when the converted object was imported. This second stage involved the operator in specifying exactly the type of content represented in a batch of files (so that the appropriate Fedora cModel could be assigned) and indicating the required security settings for use in the new repository. These new objects were then imported to Hydra's Fedora instance using Fedora's own administration client. By

virtue of the JMS listener service described earlier, they were then indexed by Solr and appeared in the Hydra UI.

These two bespoke tools are currently held in a closed Github repository because they contain administrative IDs and passwords; they will shortly be purged of this information and placed in our public Github with the other Hydra in Hull code.

Embedding

The Hydra in Hull Project was not solely about producing code and running a repository. An essential part of our approach involved ensuring that we were taking our users (both 'end users' and repository management staff) on the journey with us. In addition, we were keen to ensure that our colleagues in ICTD understood why we had insisted on adopting a repository solution that did not sit wholly within a Windows framework (their preferred approach). Thus a number of meetings were held during the course of the project to ensure that key players in our repository service were kept up-to-date and involved.

Outputs and Results

For the academic end-user

The Hydra in Hull project was always seen as the first steps in a somewhat longer process to launch a fully-functional, self-supporting Hydra-based repository for the University of Hull. That repository entered service at the end of September and has supplanted the Muradora-based for our academic users. (As we shall see in the next section, the 'old' repository is still running as a temporary management solution.)

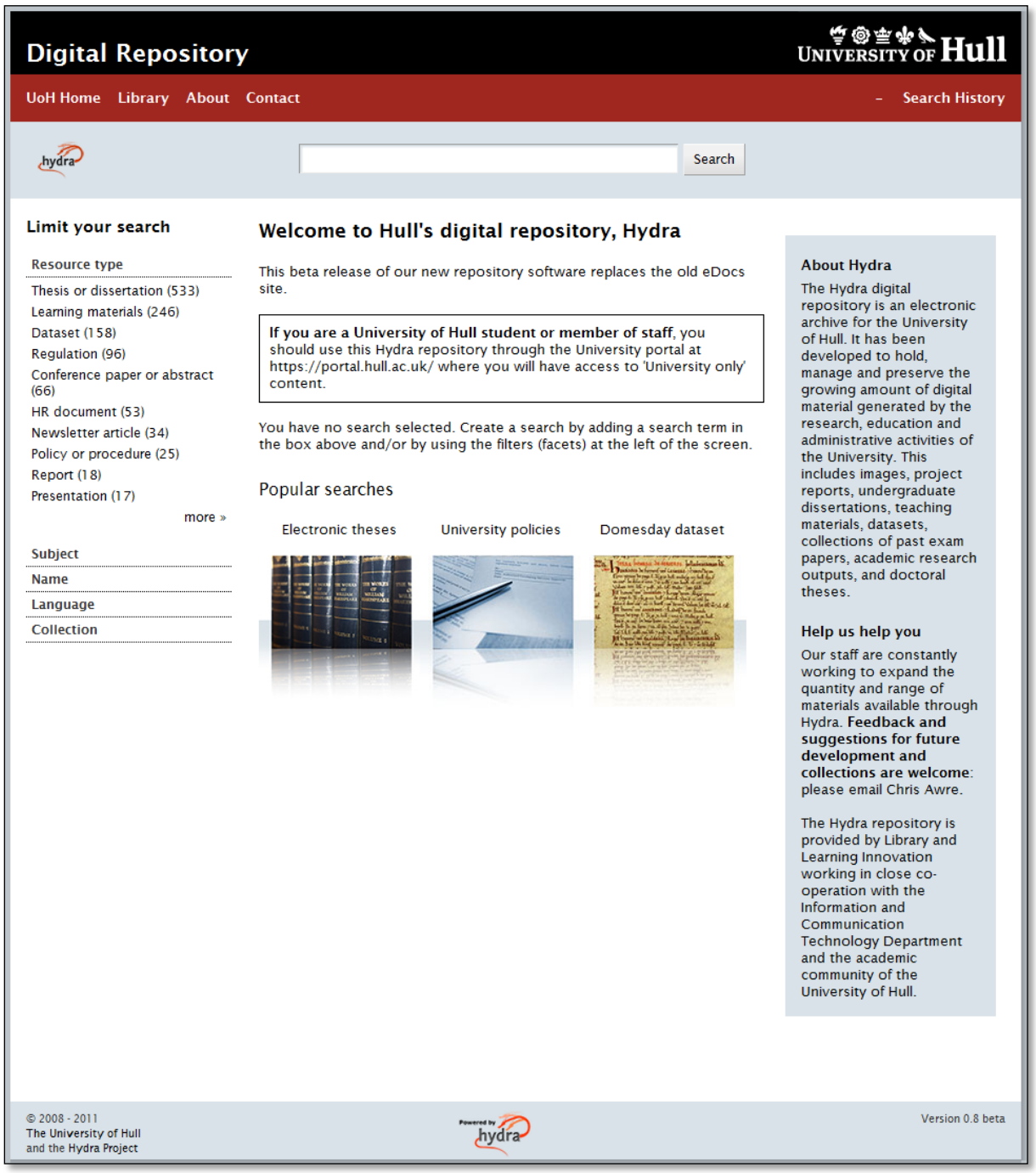

Figure 2: Hydra repository home page

As a member of the public, the Hydra repository at Hull can be accessed via hydra.hull.ac.uk This enables such a user to search and access the repository's public content; more restricted content is not accessible to them. (University users access the repository through our portal which, depending whether they are a member of staff or a student, gives them greater security privileges.) The user is able to use the facets at the left of the screen to narrow down the content to their area of interest (here the 'resource type' facet is shown open) or to use a search box at the top of the screen. The result of such a process is a search return (here the result of choosing the 'dataset' resource type):

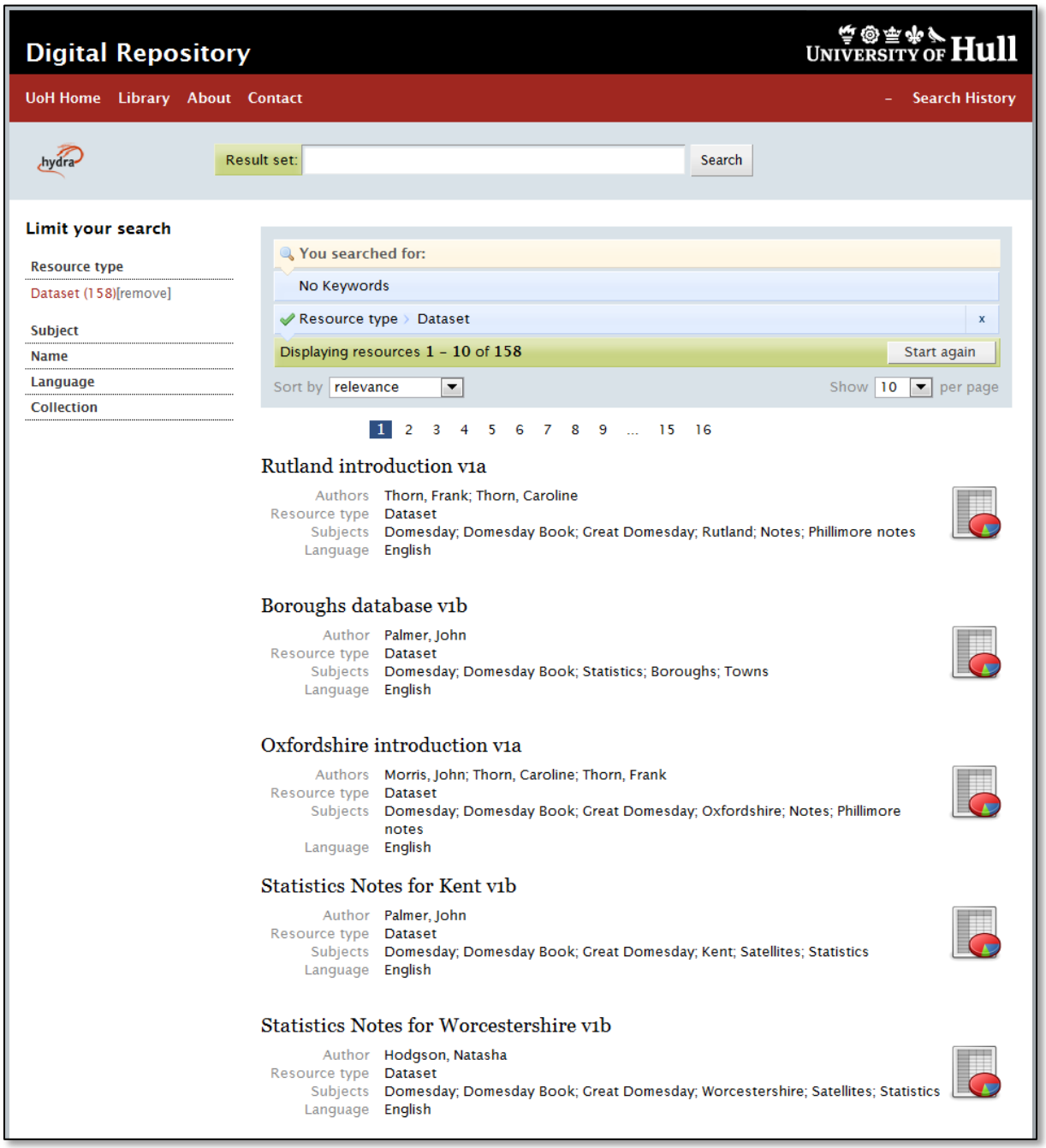

Figure 3: Hydra search return for dataset resources

The user can scroll through the results of the search or further refine it. Clicking on a resource produces a 'splash page' describing the resource more fully and offering links to initiate content download:

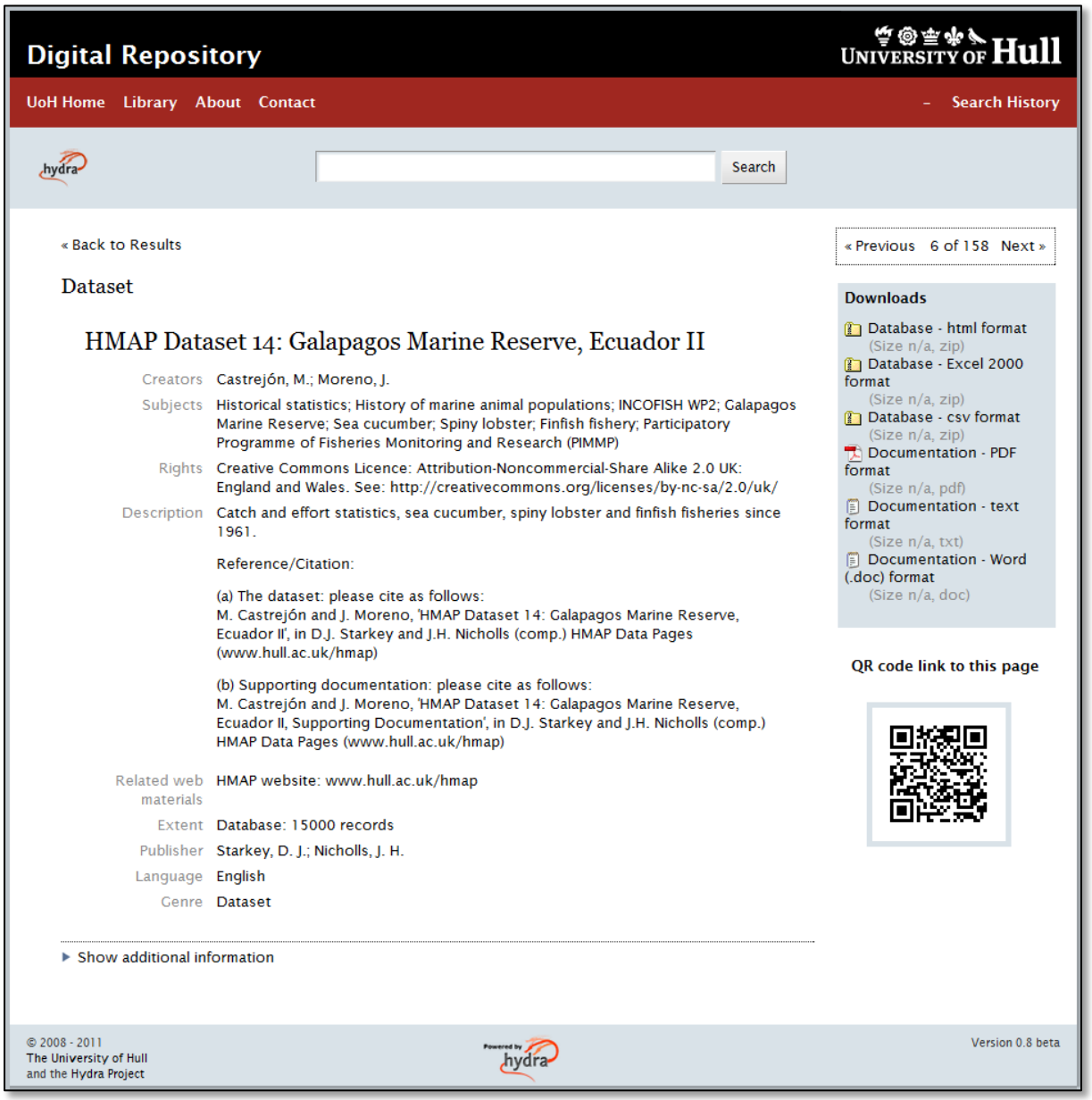

Figure 4: Hydra splash page for a dataset

Here the user is presented with the descriptive metadata appropriate to a dataset held in the MODS datastream. Had they chosen to access a publication, the page would have been somewhat different:

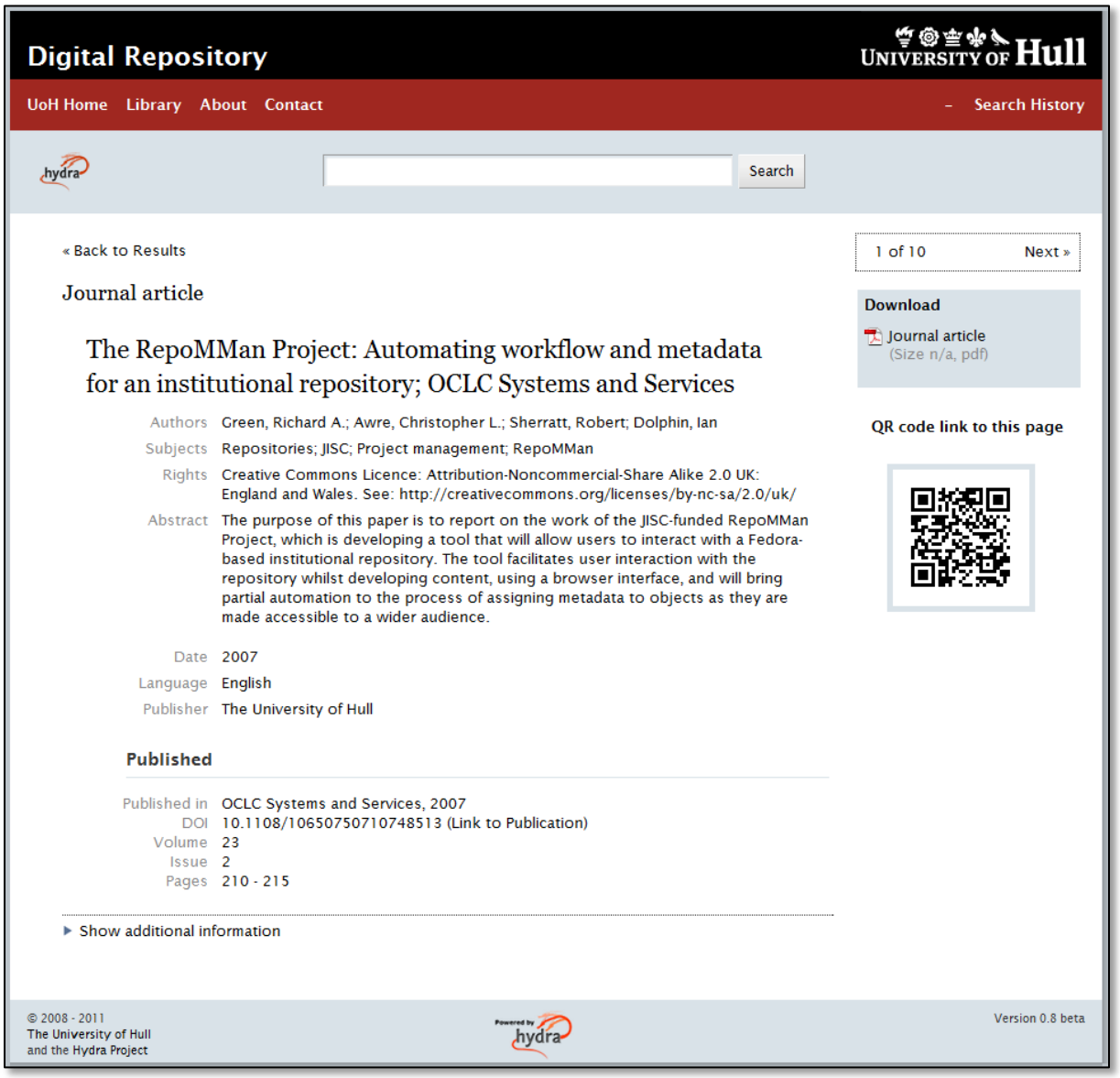

Figure 5: Hydra splash page for a publication

Here the splash page for a presentation shows rather different metadata and has a link to the formally published article as well as the ability to download the University's 'local' copy.

Opening the 'show additional information' panel allows the user to view the actual MODS metadata and the ability to see it represented in other schemas – DC in all cases and additionally UKETD_DC for electronic theses and dissertations. For all but the oldest content in the repository (by a quirk represented in both splash pages above) the information under each download link comprises the file size (in KB or MB as most appropriate) and the file type.

For repository contributors

Approved repository contributors are able to log in to the Hydra repository in order to manipulate content in various ways. Depending on their level of authorisation they may be able to create and/or update and/or delete content.

As noted above, the pages available for these purposes at the end of the project are quite basic forms, deliberately (as yet) devoid of JavaScript. We took the same approach to repository management that we took to providing read access: that, ideally, each page should be customised to the requirements of a particular content type. In other words, management pages should show those fields that were required in a given situation and hide others.

Content creation

A logged-in manager (here a member of the Library 'Content and Access Team') has two key additional functional links on the pages (s)he sees: one of which allows the creation of new content and the other that allows a toggle between 'view' and 'edit' modes. If they choose to create new content they must first identify the type (book, examination paper, ETD etc) and they are then presented with the appropriate, largely blank, form – here for an examination paper:

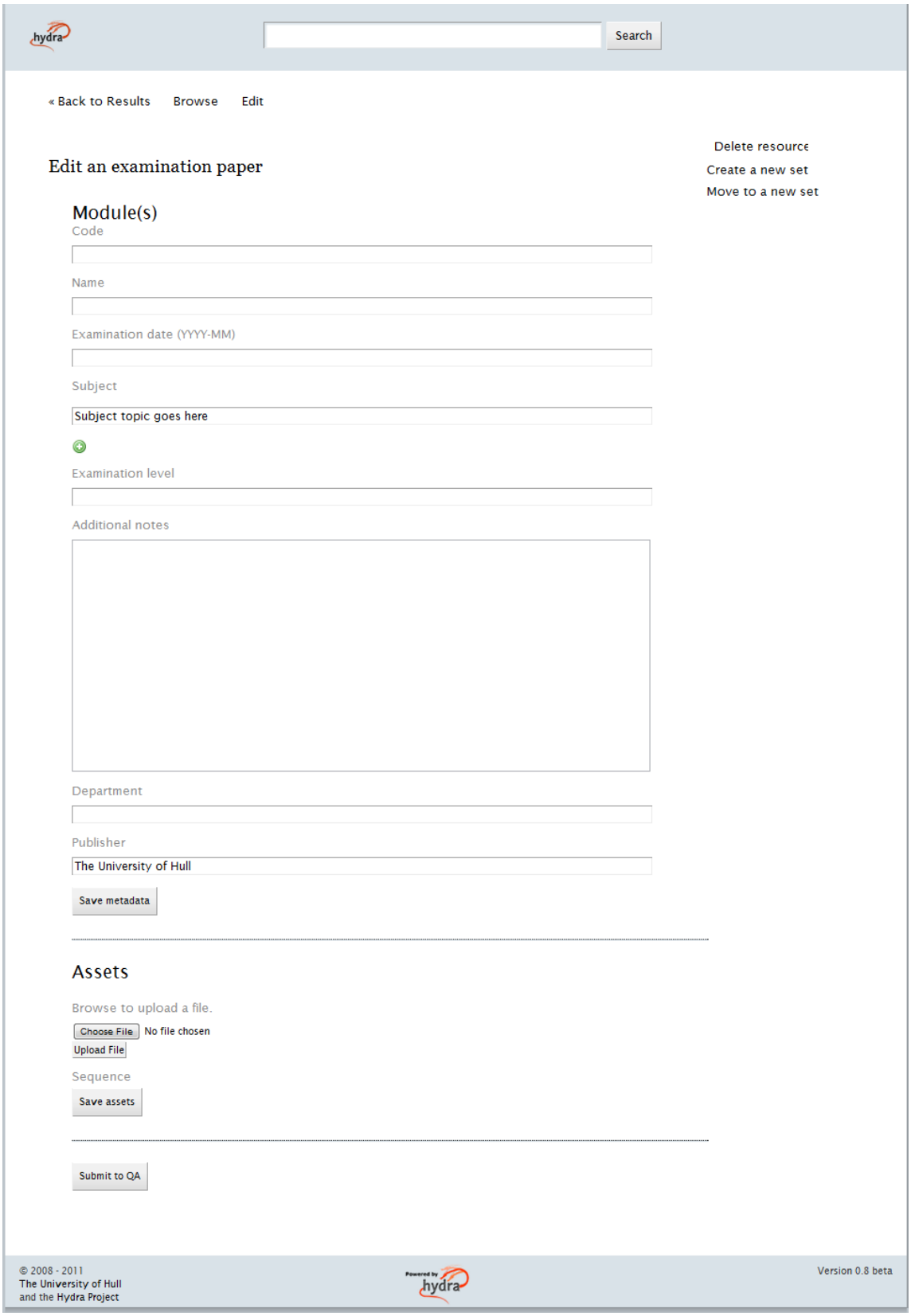

Figure 6: Create page for an examination paper

Note that the fields available to the user are those appropriate to examination papers; other potential MODS fields do not show here. Most of these fields are, by definition, singletons – which is to say that they cannot have multiple values. Note, however, that the 'Subject' field has a green button displaying a '+' sign: this allows the addition of additional subject fields should they be required. The 'Publisher' field has been pre-filled.

Once the metadata has been created and saved, appropriate file(s) can be uploaded and attached by browsing the user's computer drives. An appropriate display label can be created for each file and, behind the scenes, the system captures additional metadata such as the file type and size. If more than one file is attached, the sequence in which they should be displayed can be specified.

When these assets have also been saved, the bottom button 'Submit to QA' is used. If all necessary fields have been filled in, the object is passed to the second level of workflow for quality assurance by another member of the team.

The top of the QA page is identical to the first, but the lower section is considerably different:

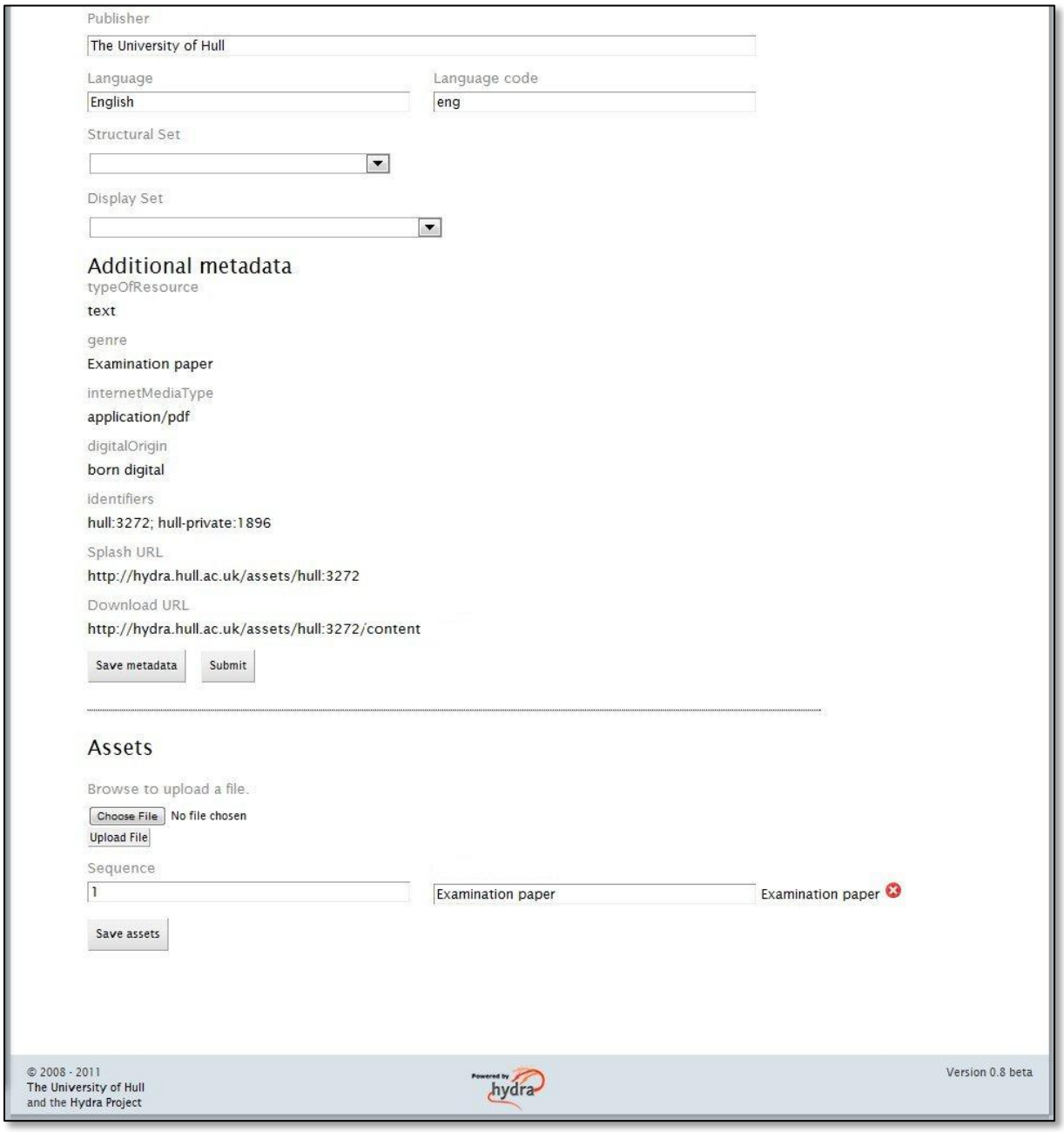

Figure 7: Part of exam paper edit page

Here, some of the automatically generated metadata is displayed and assets information is shown. In addition, there are drop-down menus that allow the object to be allocated to a 'structural set' for management purposes (this also determines the appropriate access permissions to use) and possibly a 'display set' to group it for end-users with other objects (optional). Once the QA work is done, the 'Submit' button will expose the object to appropriate repository users.

Editing

Should there be a need to edit an object that has been exposed in the repository, a generic form is generally used that covers a superset of the metadata (there is a small range of specialised content types that do have their own edit pages here).

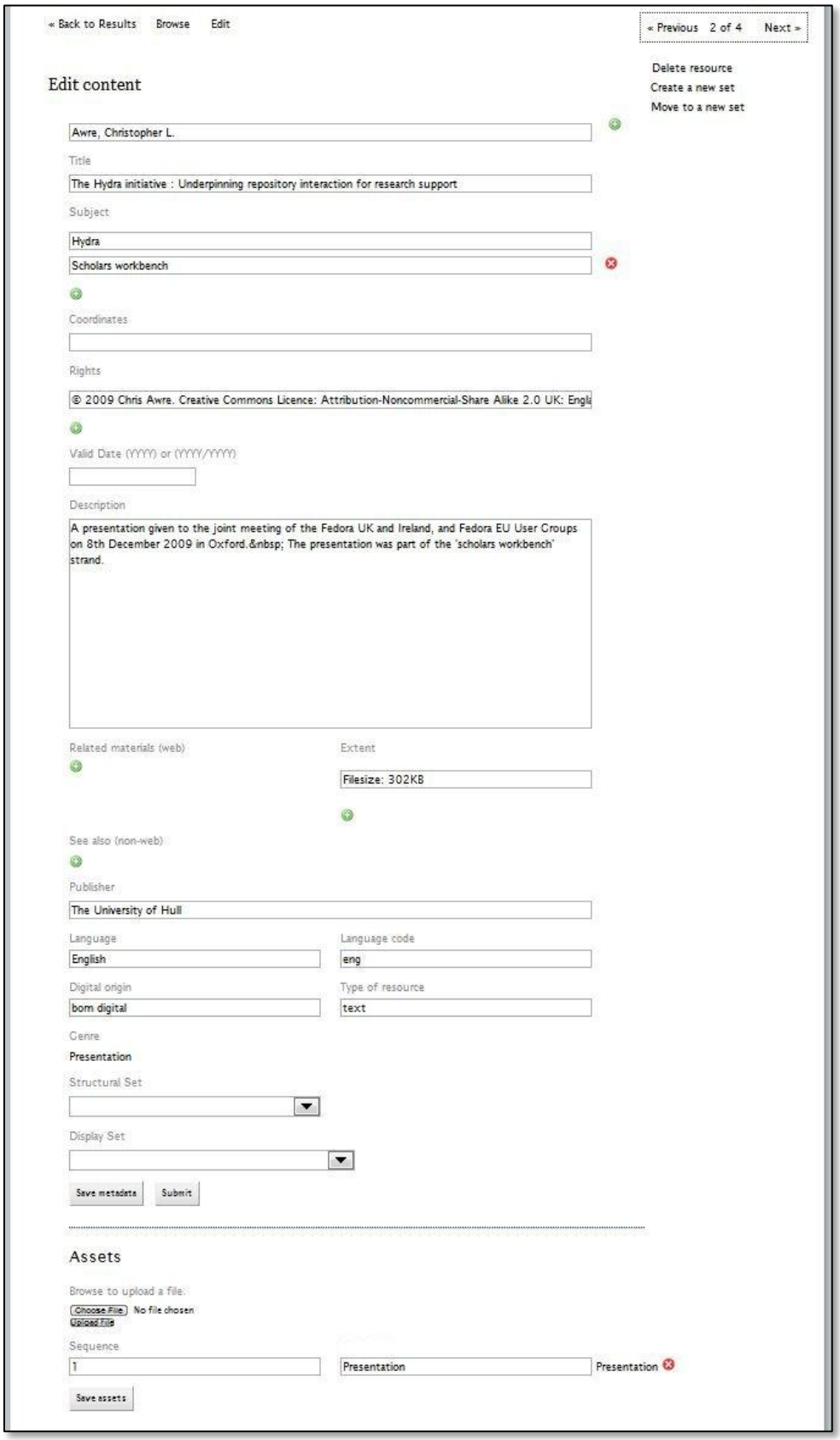

Figure 8: Edit page for general, published content

The provision of such an extensive form was a deliberate choice in order to give our professional library colleagues full range over the metadata that could be altered here. Potentially, this allows them to deal adequately with unusual items.

Deletion

Towards the top right in figures 6 and 8 is a 'delete' button. Until the object is exposed in the repository this button does, indeed, totally delete an object from the repository. Once the object has been 'published', the delete button merely renders an object inactive so that academic end-users cannot access it in any way.

Embedding

As previously noted, members of the Library Content and Access Team have been involved in the project at regular intervals and they seem eager to use the new system. Even with the flux the ongoing development has produced, they seem to feel that, once completed, the new system will be a great improvement over the old.

Dissemination

During the course of the project, we have actively been disseminating its work:

1. The project contributed a presentation to two events at the 2011 Open Repositories conference in Austin, Texas (OR11): the first a workshop around Hydra, the second a conference session on Hydra in the Fedora specialist track.[21]

2. The project team hosted a presentation to the University's Information and Communication Technology (ICT) Department outlining the purpose of the institutional repository and specifically detailing the reason for the move from the old Muradora-based repository to Hydra. The Hydra technology stack was explained at some length.[22]

3. The project contributed to the showcase of JISC RTE projects at the Kultivate conference on 15th July and also presented at the Kultivate sustainability conference on 30th September.

4. The project formed part of the JISC RTE event at the Edinburgh Repository Fringe on 4th August.

5. The project gave an update to the Fedora UK and Ireland User group meeting held in Manchester in mid-September.

6. Hull's Hydra implementation (and thus by inference the HiH project) forms a part of two papers about the JISC-funded CLIF Project:

[21] At https://wiki.duraspace.org/download/attachments/11502264/OR1124-7--v04-110610.pdf?version=1&modificationDate=1307969239535

[22] The technology explanation was based on a presentation given by Matt Zumwalt, a member of the Hydra Steering Group, at OR11. Matt's presentation is at: http://prezi.com/1lmhfhcvjhmm/hydra-technical-framework/

a) an article which has been accepted for Ariadne[23] No.68 (date tba), and

b) a paper submitted to the Journal of Digital Information[24]

and was referenced frequently in a presentation about CLIF to the European Sakai conference in September.

7. The Hydra in Hull implementation is one of the key installations referenced in detail on the Hydra Project's own website.[25]

Outcomes

The first aim of our project was to document and feed back the experiences and benefits of our move to Hydra, including the use of Ruby on Rails for repository applications, and use of the Hydra models to the Hydra and wider repository communities. We have taken many opportunities to do this and will take many more in the future. A major aspect of the project was to help the University of Hull implement Hydra as a new repository interface and to go about it in such a way that we could act as ambassadors and as a focus of guidance and assistance to those in the UK (and potentially also mainland Europe) who may wish to investigate a similar undertaking; we believe that we are in a position to do this. Indeed there are indications that others in the UK are seeking to follow suit and we understand that a university in the USA is considering adopting our customised Hull code as the basis for their own repository implementation. Hull's Hydra repository offers such potential adopters a reference implementation.

Secondly, we aimed to embed the use of Hydra within the local cataloguing (Content and Access) team and other institutional users wishing to deposit content, and to document changes to processes required. As regards the Content and Access Team, we have largely achieved this, albeit there remains work to be completed for some of their management screens; unavoidable delays to the project in the late summer meant that we have not yet involved other institutional users but this will happen very soon. Modified documentation will become available at that point. It is clear that there is considerable enthusiasm for the new system and we will build on that interest.

In order to improve the consistency of repository content management we had the aim of adopting the Hydra content modelling strategy. This has been done successfully and in such a way that we can now provide customised views both for reading and managing the different types of content. This has been a significant area of development and has led to the extension of the core Hydra guidance.

During and after the project we intended to contribute further to the development of Hydra and to influence the development path that it takes. We have done this and, as members of the steering

[23] See: http://www.ariadne.ac.uk
[24] See: http://journals.tdl.org/jodi
[25] See: http://projecthydra.org

committee, the Hull team will continue to do this. Our own implementation of the Hydra stack has led to a number of improvements and additions to core Hydra code.

Finally, our last aim was possibly to generate requirements for other Hydra heads. The University has been successful in bidding for further JISC funding in order to allow the development of a Data Management Plan for the Department of History. As part of this new project the repository team will have the opportunity to enhance the provision for datasets in Hydra both locally and internationally. Whilst it is probable that Hull's history work (and later the extension of this to other departments) will not lead to the creation of a new, free-standing Hydra head – rather enhanced provision within the existing one – it could form the basis of Hydra heads elsewhere.

Overall we are pleased with the outcomes of this project and feel that it will allow us to help and encourage other Hydra adopters in the UK, mainland Europe and beyond.

Conclusions

The following conclusions can be made on the basis of undertaking the HiH project:

- Whilst realising that the Hydra software was continuing to evolve as the project took place, the number of changes that resulted, whilst all being positive steps forward in its development, had a complicating impact on progress within the project. Many of these changes were necessary for our implementation, and as such we needed to take them on. It may have been practical at times, though, to acknowledge that a change would take place, but to also move ahead without that change at that time, returning to it later. We did do this on occasion, but later in the project than was perhaps practical. Whilst this may have resulted in a less up-to-date implementation, it may also have allowed smoother progress.
- The process of working in community on the development and implementation of Hydra has been a very powerful and enabling feature of the work. Hull would never have been able to pursue this on its own. The JISC project provided an opportunity to benefit from the community effort in a way that implementing a commercial solution would not have mirrored. It has encouraged us to continue our involvement in the Hydra community and similar initiatives as appropriate for our long-term strategy.
- The Ruby on Rails technology base for Hydra has been one we have had to learn in depth as part of this project, notwithstanding earlier preliminary activity. This technology does not seem to be widely used within UK HE, and has had its critics over the years in terms of its capability for large systems. Nevertheless, the flexibility it has offered us in being able to both implement Hydra and build on this beyond the project has suggested it will be a valuable asset and set of skills to have locally.
- The feedback from library staff and end-users to the interface has been very useful in guiding our implementation, and will no doubt influence ongoing and future changes as it is used more. We will thus continue to capture this feedback where we can to inform further development of the repository. Additionally, involving the local ICT Department was

essential in making development resource available and aiding understanding of what we wished to achieve.

Implications

The end of this project is far from being the end of work on Hydra in Hull. There remains much to do:

- The autumn of 2011 will see work in hand to complete full CRUD functionality for our major content types and generic provision for the rest; we hope also to deal with most of the known bugs during this time. Completion of this work will allow us finally to turn the old repository off.
- Specialist support will be added for further content types, particularly images
- During November, enhancements will take place to the underlying core Hydra code (gems) so that we can take advantage of the new Rails 3 release. Part of this enhancement will enable the repository to be HTML5 compliant (thus addressing some accessibility issues) and it will also allow us to add JavaScript functionality to pages more easily than at present
- Hydra management functionality will be offered to contributors beyond our Library professionals and the necessary documentation and training will be provided. Within a short time this may include key academic staff in the Department of History, a result of their involvement in the newly JISC-funded History DMP Project.
- The 2011/12 strategic plan for Hull's Library and Learning Innovation unit identifies work that will integrate the repository interface with the main library catalogue (OPAC). This work is anticipated during the spring of 2012.

These many developments, and others, will form part of a process and policy document describing the use of Hydra in Hull. This document was intended as a deliverable for our project here but the scope of it is now such that it will be slightly delayed and will be made available, via the repository, post-project.

Recommendations

There are two main recommendations from the HiH project:

- That institutions/libraries, etc. should be open to getting involved and contributing, in whichever way they can, to community initiatives to develop systems that meet common needs. We have had periodic concerns about our inability at times to contribute much in the way of technical development. Nevertheless, we have fed back what we can, and it is this

willingness to participate that is at the heart of the success of the community that is behind Hydra.

- Those managing digital content should give careful attention to how it is structured and organised, in order to facilitate its long-term curation. This is not simply a case of structuring it to fit with the system being used to hold it, but a need to understand how a digital file and the components associated with it (e.g., metadata, surrogates, etc.) should be related so that the context around digital content is kept alongside the content itself. Fedora aids this approach through its digital object model, but this is not a system-specific attribute and can be applied regardless of technology.