SWORD: Simple Web-service Offering Repository Deposit

Julie Allinson, Sebastien François and Stuart Lewis describe the JISC-funded SWORD Project which has produced a lightweight protocol for repository deposit.

Publication Date: 30-January-2008 Publication: Ariadne Issue 54
Originating URL: http://www.ariadne.ac.uk/issue54/allinson-et-al/

Introduction

This article offers a twofold introduction to the JISC-funded SWORD [1] Project which ran for eight months in mid-2007. Firstly it presents an overview of the methods and madness that led us to where we currently are, including a timeline of how this work moved through an informal working group to a lightweight, distributed project. Secondly, it offers an explanation of the outputs produced for the SWORD Project and their potential benefits for the repositories community.

SWORD, which stands for Simple Web service Offering Repository Deposit, came into being in March 2007 but was preceded by a series of discussions and activities which have contributed much to the project, known as the 'Deposit API'. The project itself was funded under the JISC Repositories and Preservation Programme, Tools and Innovation strand [2], with the over-arching aim of scoping, defining, developing and testing a standard mechanism for depositing into repositories and other systems. The motivation was that there was no standard way of doing this currently and increasingly scenarios were arising that might usefully leverage such a standard.

Overview of SWORD's Output

Before outlining the history and inner workings of SWORD, here's a brief overview of what SWORD has produced (to be further expanded in the following sections):

1. A profile of the Atom Publishing Protocol which can be used by implementers to create SWORD-compliant deposit clients or SWORD interfaces into repositories, where the client will perform the deposit and the interface will accept it. [3]
2. Test implementations of the SWORD interface in DSpace, EPrints, IntraLibrary and Fedora to demonstrate the efficacy of the SWORD approach. [4]
3. Two demonstration clients which can be used to deposit into the implementations at (2) or into any other SWORD-compliant implementations. [5]
4. Open source code for DSpace, Fedora, EPrints and the demonstration client. [6]

Background to SWORD : The Deposit API

Discussions about the lack of a standard mechanism for deposit have surfaced in a number of places, notably: in a session on repository reference models at the JISC CETIS Conference in 2005 led by Lorna Campbell and Phil Barker [7]; in Andy Powell's 'A service-oriented view of the JISC Information Environment' in which a set of discrete services that a repository should offer included 'deposit' [8]; in the 'add' service identified by the eFramework [9]; and in presentations about the new Open Archives Initiative-Object Reuse and Exchange (OAI-ORE) Project where 'register' has been used synonymously for 'deposit' [10].

Drawing on these indications that a deposit standard was needed within the repositories space, the newly formed Repositories Research Team, a collaboration between UKOLN and CETIS, led by Rachel Heery, contacted a group of repository developers in late 2005 and asked firstly whether they agreed that there was an identifiable need for a standard deposit protocol and secondly whether they would be willing to join a small 'Deposit API' working group of repository developers to kick off discussion in this area. Among the repository platforms represented in the discussions were ARNO, OCLC, KaiNao, Harvestroad, Intrallect, EPrints, Fedora, DSpace and aDORE. Many of those contacted were able to attend meetings in March and July 2006 where the requirements and scenarios around 'deposit' were explored and the beginnings of a serialisation were laid down [11].

Rationale for a Standard Deposit Mechanism

There are various scenarios which illustrate why a standard deposit mechanism would be useful. A standard deposit interface to repositories could allow more services to be built which can offer functionality such as deposit from multiple locations, e.g. disparate repositories, desktop drag&drop tools or from within standard office applications. Illustrations of this are shown in figures 1 and 2.

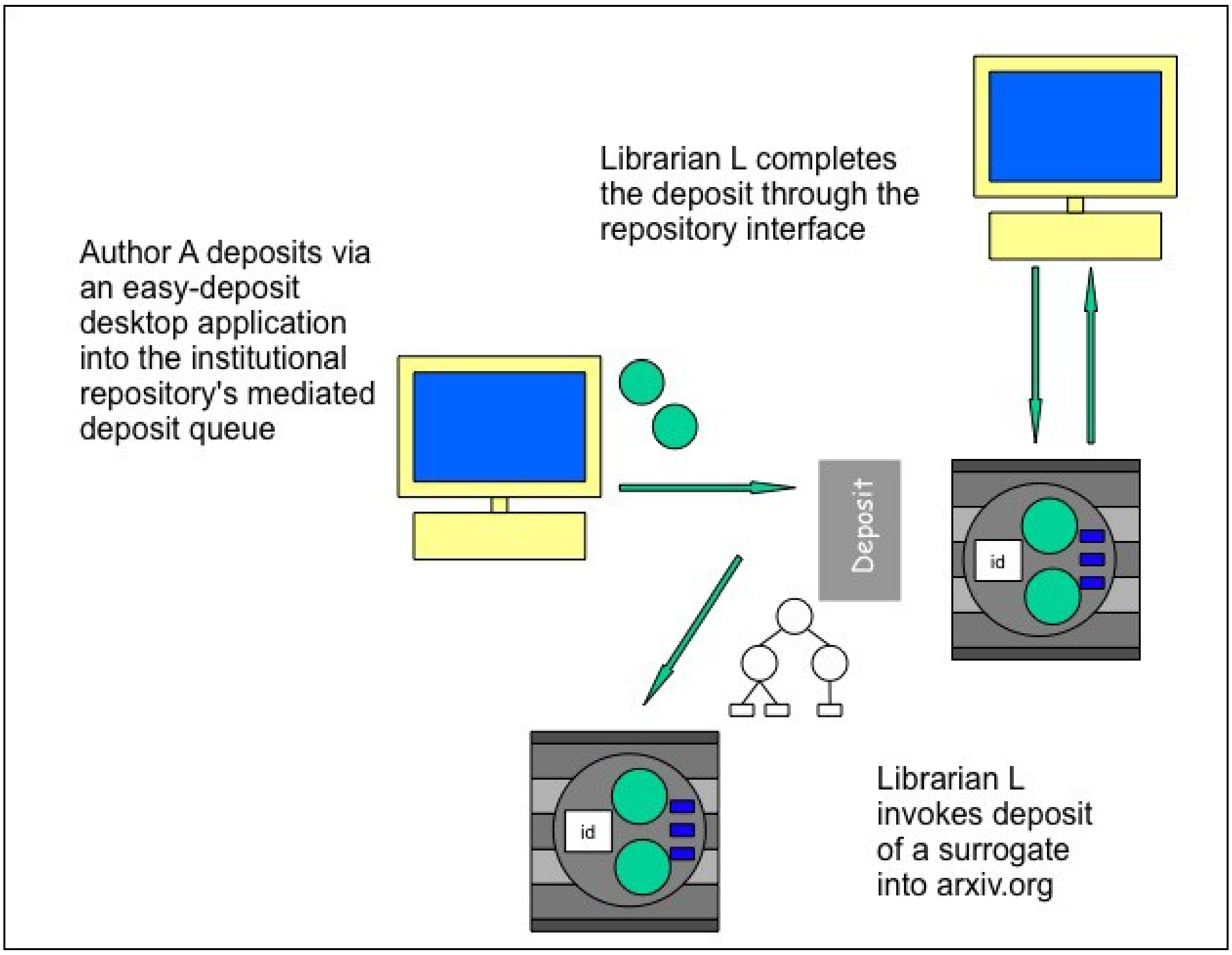

Librarian L completes the deposit through the repository interface
Author A deposits via an easy-deposit desktop application into the institutional repository's mediated deposit queue
Deposit
id
id
Librarian L invokes deposit of a surrogate into arxiv.org

Figure 1: Author deposits using a desktop authoring system to a mediated multiple deposit service

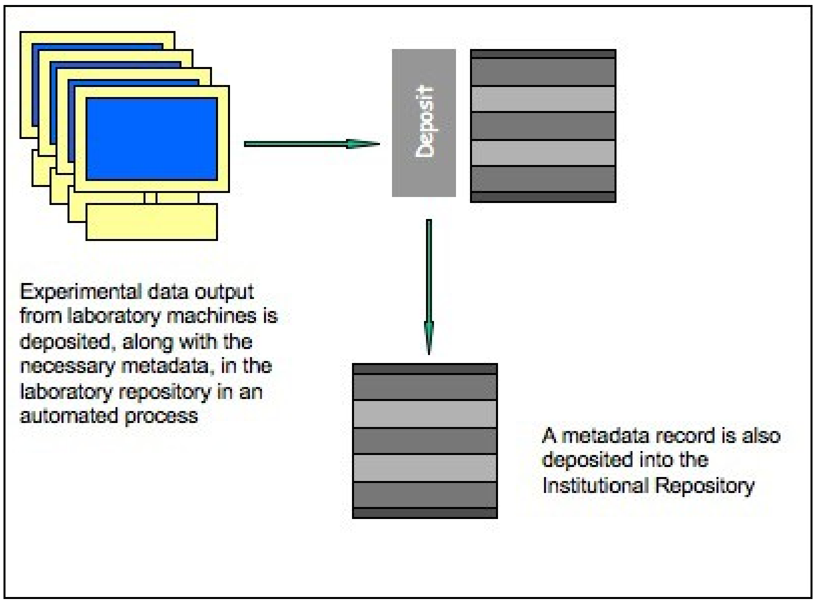

Deposit
Experimental data output from laboratory machines is deposited, along with the necessary metadata, in the laboratory repository in an automated process
A metadata record is also deposited into the Institutional Repository

Figure 2: Deposit of experimental data output from a laboratory machine is initiated by a 'save as' function

It could also facilitate deposit to multiple repositories, increasingly important for depositors who wish to deposit to funder, institutional or subject repositories (see figure 3). Other possibilities include migration of content between repositories, transfer to preservation services and many more (see figure 4).

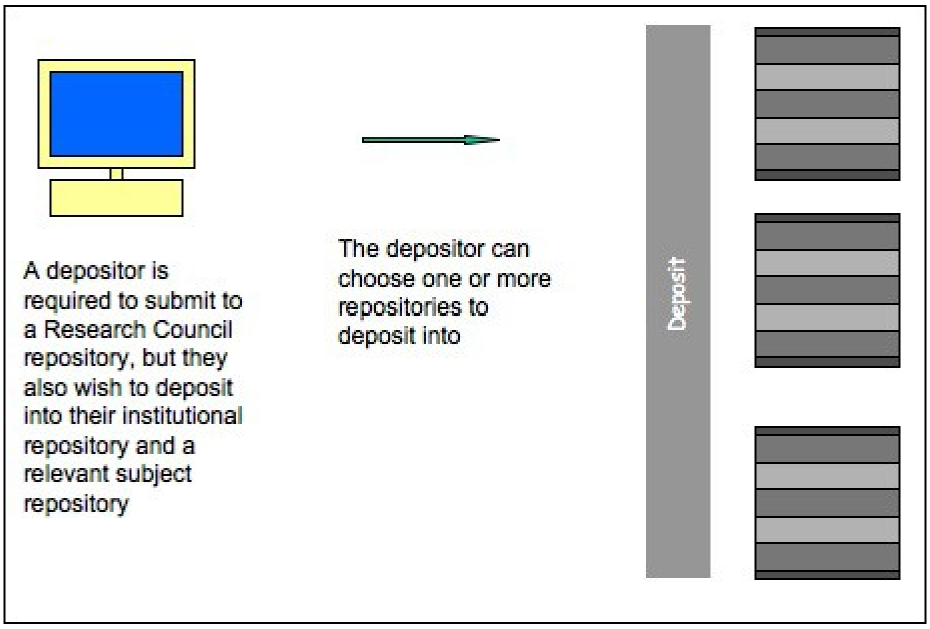

A depositor is required to submit to a Research Council repository, but they also wish to deposit into their institutional repository and a relevant subject repository
The depositor can choose one or more repositories to deposit into
Deposit

Figure 3: Deposit to multiple repositories is achieved from a single deposit point

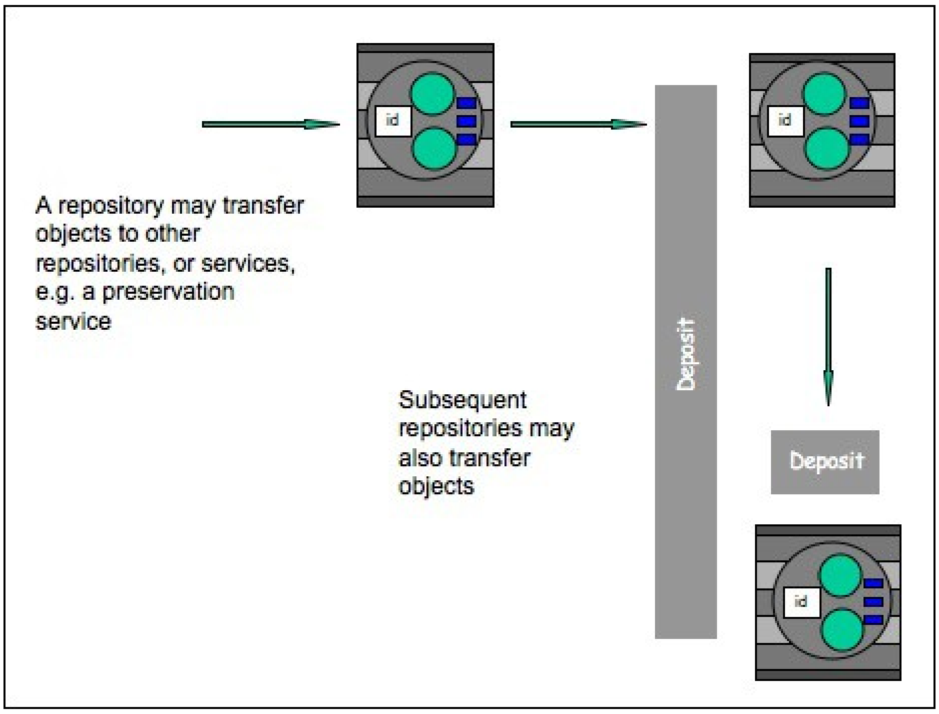

A repository may transfer objects to other repositories, or services, e.g. a preservation service
id
Subsequent repositories may also transfer objects
Deposit
Deposit

Figure 4: A repository transfers data to another repository or preservation service

SWORD: The Project

SWORD was funded to take the deposit API activity into a more formally funded project, to ensure that the ideas and enthusiasm already captured could be used to produce concrete outputs. Led by UKOLN, the project was a partnership between CASIS at Aberystwyth University, the University of Southampton and Intrallect. The project aims were simple – to agree on a protocol or specification for deposit, to implement a deposit interface into DSpace [12], Fedora [13], EPrints [14] and IntraLibrary [15] and to produce a prototype 'smart' deposit client for testing the implementations.

Requirements and Parameters

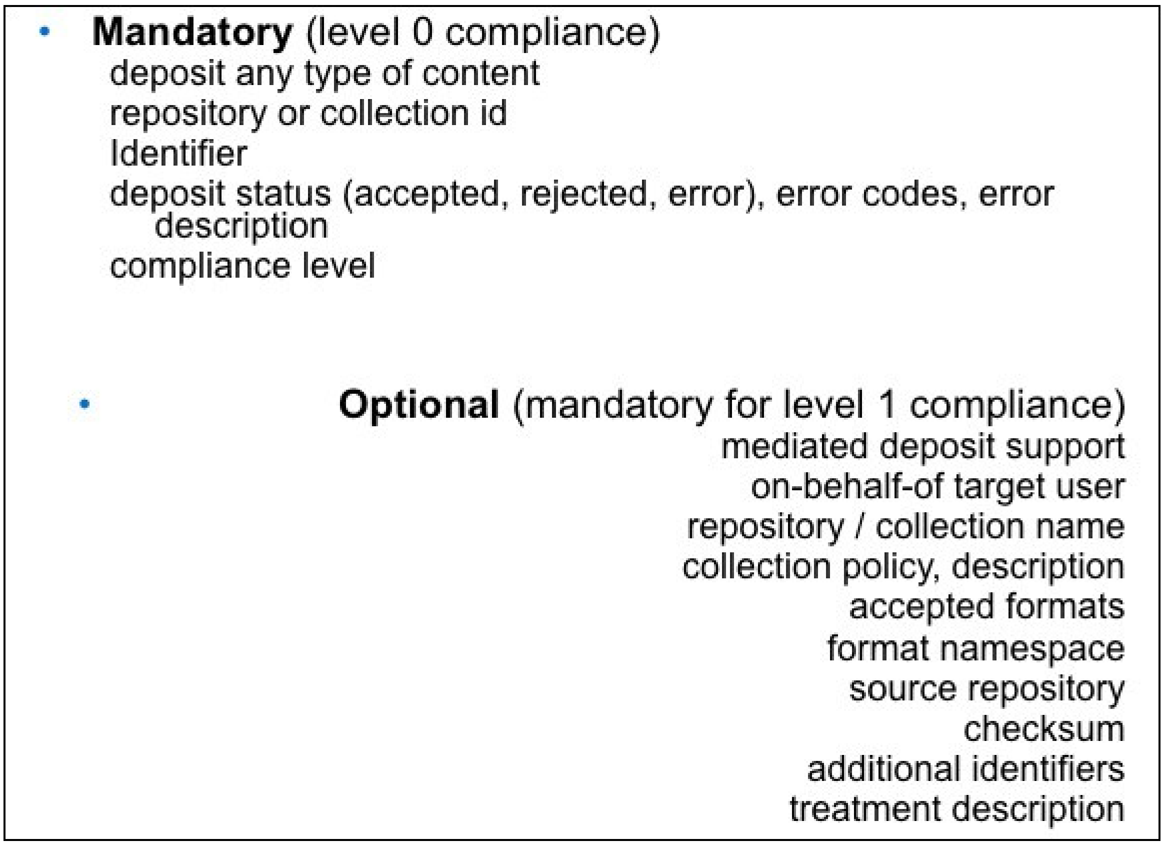
Before agreeing a specification for deposit, the next step was to turn our scenarios into a set of concrete requirements from which we could derive a service definition and set of parameters. Paramount in such a short project was the need to have a set scope. For example, we agreed that authentication and authorisation, although important, could not be mandated or settled by SWORD. SWORD concerns itself with the point of deposit, so metadata creation, again essential, should arrive at the deposit point in an agreed standard format such as that defined by the Scholarly Work Application Profile (SWAP) [16], but is not created by the deposit interface. Issues surrounding standard metadata, data integrity, data formats and packaging are not trivial and it is easy to see that a standard deposit mechanism could fail if the deposits cannot be read and understood by different repositories. Nonetheless, attempting to fulfil too big a scope would have made the work of the SWORD Project unmanageable within our time frame. A selection of the requirements which emerged from our discussions were the need:

- to support a wide range of heterogeneous repositories, e.g. scholarly publications, data, learning objects, images, etc.
- to accept submission of different digital object types in a consistent way.
- to accept data and/or metadata in the form of complex objects or content packages
- to support different workflows for deposit, e.g. user to multiple repositories via intermediate client; user to repository, repository to additional repositories; user-triggered and machine-triggered deposit.
- to accept large-scale deposits, e.g. scientific datasets.
- to support collections and changes in policy and permissions.

- to support non-instantaneous processes, e.g. deposit pending mediation.
- to support more complex, authenticated and mediated deposit.

Defining the Service

From these requirements we moved onto discussions about what a deposit service would comprise. Two distinct services were identified: deposit and explain. The explain service would allow a repository to provide information about its policy, whilst the deposit service would allow remote users to put data into the repository, with an accompanying 'receipt' indicating success or failure. Additionally, a layered approach was outlined, whereby basic level '0' compliance could easily be reached, with an extra level '1' including additional parameters necessary for some repository transactions. An added benefit of this layered approach was that it allowed additional layers to be added in the future. This is illustrated in figure 5.



- **Mandatory** (level 0 compliance)
  deposit any type of content
  repository or collection id
  Identifier
  deposit status (accepted, rejected, error), error codes, error description
  compliance level

- **Optional** (mandatory for level 1 compliance)
  mediated deposit support
  on-behalf-of target user
  repository / collection name
  collection policy, description
  accepted formats
  format namespace
  source repository
  checksum
  additional identifiers
  treatment description

Figure 5: Parameters and levels of compliance [alternative format]

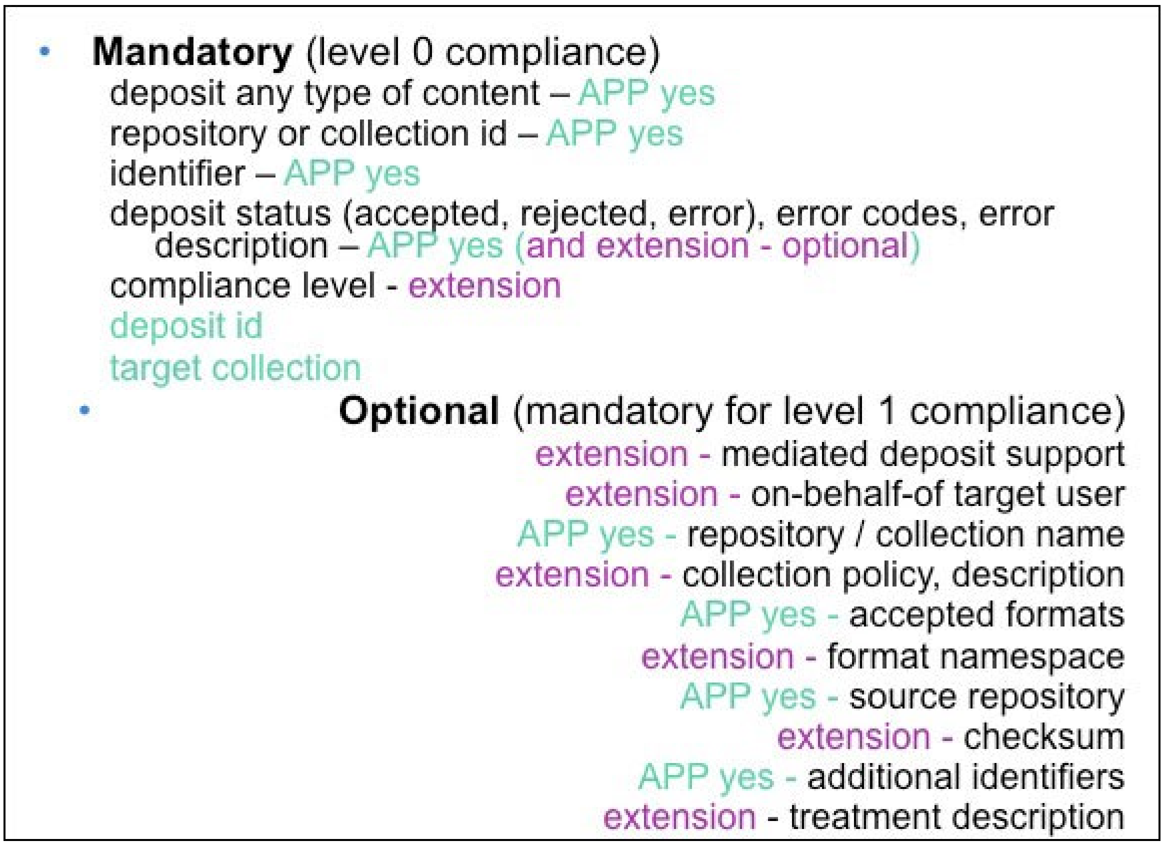
Protocol and Profile

From the outset of the Deposit API work, it was acknowledged that standards and specifications already existed which might be used to provide the basis of a deposit service. A shortlist of candidate deposit specifications was drawn up, including WebDav, the OKI OSID (Open Knowledge Initiative Open Source Interface Definitions) and SRU/W (Search and Retrieve via URL or Web Service). These were reviewed in a project brainstorming meeting and it was the Atom Publishing Protocol (APP or ATOMPUB) [17] that captured the interest of the group. Lightweight, relatively easy to implement, already growing in use for remote blog posts and closely tied to the widely used Atom Syndication format (ATOM) [18], APP had much in its favour. But would this lightweight standard be sufficient to fulfil the needs of repositories?

The Atom Publishing Protocol (APP) is 'an application-level protocol for publishing and editing Web resources'. It is based on the HTTP transfer of Atom-formatted representations and is designed to support the deposit (POST) of

ATOM documents and datastreams such as images. In addition, it also facilitates update and delete. Because of its scope, SWORD has focussed on two key aspects of the protocol - the deposit of files, rather than Atom documents, and the extension mechanism for specifying additional deposit parameters. SWORD does not specify the implementation of all of the functionality of APP since it supports deposit only, yet this is not meant to constrain implementers who want to support the fullness of APP. Such extended functionality is a potential next step for SWORD.

By adding a small number of extensions to APP, we were able to support the list of parameters identified in figure 5 above as illustrated in figure 6:



- **Mandatory** (level 0 compliance)
  deposit any type of content – APP yes
  repository or collection id – APP yes
  identifier – APP yes
  deposit status (accepted, rejected, error), error codes, error description – APP yes (and extension - optional)
  compliance level - extension
  deposit id
  target collection
- **Optional** (mandatory for level 1 compliance)
  extension - mediated deposit support
  extension - on-behalf-of target user
  APP yes - repository / collection name
  extension - collection policy, description
  APP yes - accepted formats
  extension - format namespace
  APP yes - source repository
  extension - checksum
  APP yes - additional identifiers
  extension - treatment description

Figure 6: Parameters and levels of compliance [alternative format]

Technical Outputs and Proof of Concept

Within the boundaries of the SWORD Project we have produced demo SWORD deposit interfaces in EPrints, Fedora, DSpace and IntraLibrary. Prototype clients in desktop, command-line and Web form allow for testing of SWORD interfaces, and sourceforge code for all of these permit reuse and wider dissemination. The following sub-sections provide some technical detail about the EPrints, DSpace, Fedora and reference client Java implementations.

Generic Java Server Implementation

A generic Java Servlet Implementation of a SWORD server was created to allow any Web system written in Java to add a SWORD interface easily. The SWORD Servlet works by handling the SWORD interactions with the user, but delegates the handling of authentication, service document requests and package deposits to the repository. The repository passes messages back to the SWORD Servlet to forward to the user in an appropriate format. Since these are standard actions a repository will perform, integrating it with the SWORD Servlet is very easy.

DSpace Implementation

The DSpace SWORD implementation has been written as an 'add-on' to DSpace making use of the new modular build system introduced in DSpace version 1.5. This allows the SWORD module to be installed with ease alongside a current DSpace installation. Once installed, it uses the generic Java server implementation of SWORD, and interfaces this with DSpace. Out-of-the-box, it supports the use of zipped METS (Metadata Encoding and Transmission Standard) packages (zip files), with a METS manifest containing SWAP encoded metadata.

Due to the hierarchical community and collection structure used by DSpace, responses to service document requests are easy to collate. Any collection that a user can deposit to within DSpace is included in the list of collections returned to the user in the service document. As each collection within DSpace has its own licence, this is used in the SWORD collection policy of each collection.

Deposited packages are ingested into DSpace via the METS plugin, with the SWAP metadata converted (crosswalked) into DSpace's intermediate, XML-based metadata encoding 'DIM' [19]. This is then ingested into DSpace, along with any files in the package. Authentication is delegated to DSpace, which then uses its configured authentication mechanism, for example LDAP (Lightweight Directory Access Protocol), or its in-built password system.

Fedora Implementation

As with the DSpace implementation, Fedora made use of the generic Java SWORD implementation. The implementation is very flexible in terms of the object types it accepts for deposit. It can accept single image files (gif or jpeg) which it stores in a new object with a single datastream, or it can accept zip files where it stores each file contained in the zip file in a separate datastream. Alternatively it can accept METS documents where it separates each dmdSec (descriptive metadata section) into a datastream, or a zip file with a METS manifest which it can handle in a similar manner.

Eprints 3 Implementation

Installation

EPrints introduced a Plugin hierarchy in version 3, allowing the system to be easily customised. Installing SWORD on EPrints 3 is simply a matter of copying some Perl files to the EPrints directory. SWORD can then be easily enabled or disabled by editing an Apache configuration file.

Software Implementation

The SWORD protocol was implemented as a two-step mechanism: the first step deals with unwrapping the metadata (eg. when metadata is contained within a ZIP file) while the second step performs the actual import. As such, two new classes of plugins were created on EPrints: the Unpacker class and the Importer class. A MIME type is associated with each Unpacker plugin, while a Namespace is associated with each Importer plugin. For example when a user deposits EPrints XML metadata embedded in a ZIP file, EPrints will first call the ZIP Unpacker module which decompresses the XML file. Then the system calls the XML Importer plugin to perform the actual import of metadata. Error messages will be returned to the client (especially the HTTP/400 Bad Request message) when the plugins fail respectively to decompress or import the data. Otherwise, the newly deposited item will appear in the target collection. (It is also possible to import raw files without embedded metadata eg. a ZIP file containing a number of pictures or PDF documents.)

This design allows (advanced) users to support new formats easily, according to the need of the repository, and without interfering with the SWORD implementation. Adding a new type (either MIME or Namespace) requires writing a module which handles the format and adding a single line to the SWORD configuration file.

Collections

Items in an EPrints repository belong to one of three datasets: the “user inbox” (only visible to the depositing user), the “buffer” (items awaiting for approval) and the “live archive” (items visible to the outside world). These datasets are the target collections when using SWORD. Depending on the policy of the repository, it is possible to disable any of these collections (eg. users are not usually allowed to deposit directly to the live archive on institutional repositories).

Authentication

Since EPrints prevents its modules from returning raw HTTP headers, the authentication is carried out by the SWORD core. This also makes it easier to define who can deposit on behalf of other users without modifying the conceptual model of EPrints. Authentication over HTTPS is known to work but it is the EPrints administrators who would have to set this up by themselves via Apache.

Future

EPrints hopes to support the full APP stack to enable more complex operations on existing items (updating, removing items or metadata, etc) and to develop extra tools for clients such as Microsoft plugins to deposit directly from Office applications.

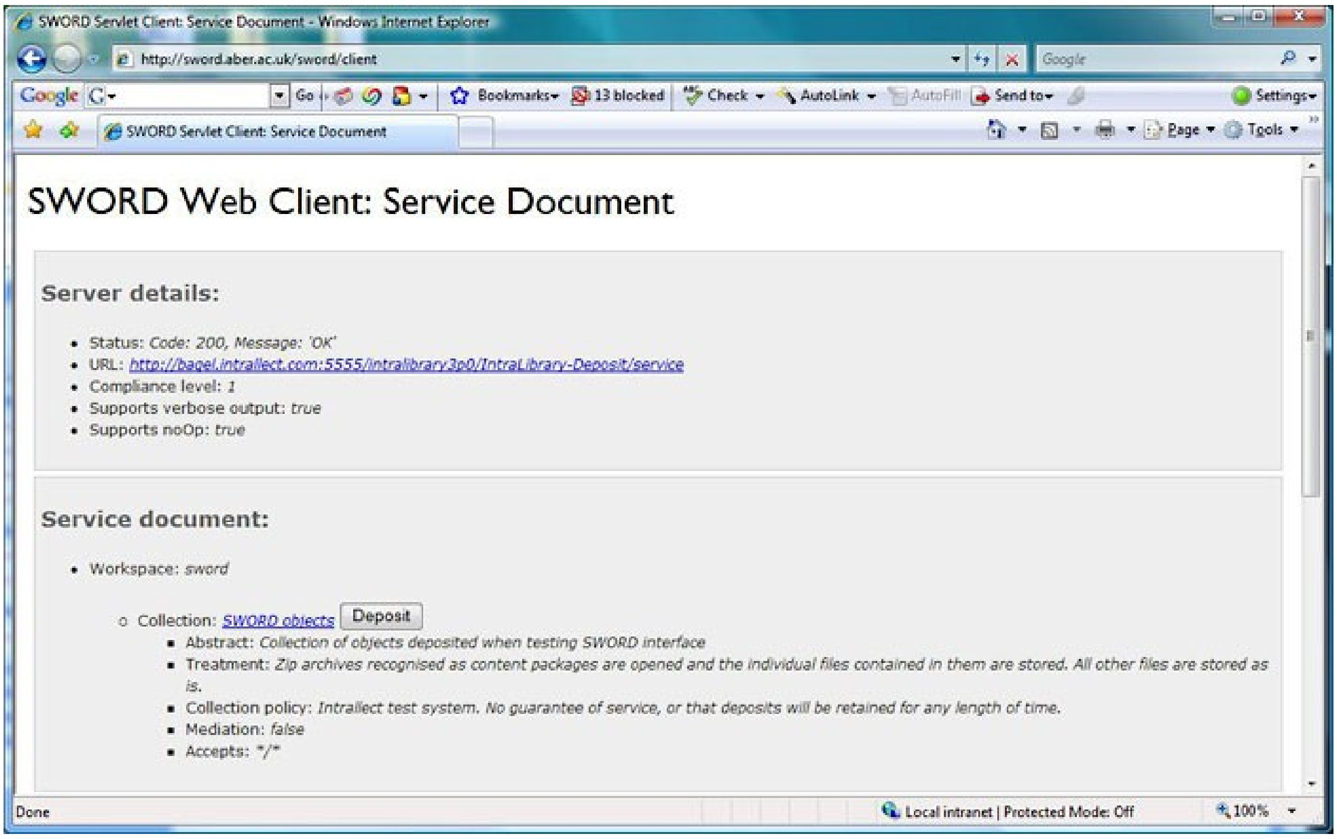
Clients

In order to test a new service such as SWORD, example clients are required. As part of the SWORD Project, a core client library was commissioned and created. In addition to this, three example clients were built using the client library:

- A command-line client: Useful for running automated scripts for testing. Also useful for batch deposit of items by a computer (e.g. laboratory equipment).
- A GUI (graphical user interface) interface: This is the sort of interface a human user might employ.
- A Web-based client: An online alternative to the GUI client, which has the benefit of not requiring any software to be installed.

The core client library was written in Java for portability and can easily be extended to create new clients. The library and example clients are all published with an Open Source licence and can be downloaded and extended.

The clients can all either request a service document, or deposit to a collection. The GUI and Web-based clients both make this easy by allowing a user to select from a preconfigured list of repositories and their service document locations. Once a service document has been retrieved, the user is presented with the list of collections, and deposit buttons to deposit to each collection easily.


SWORD Web Client: Service Document

Server details:

- Status: *Code: 200, Message: 'OK'*
- URL: *http://bagel.intrallect.com:5555/intralibrary3p0/IntraLibrary-Deposit/service*
- Compliance level: *1*
- Supports verbose output: *true*
- Supports noOp: *true*

Service document:

- Workspace: *sword*
    - Collection: *SWORD objects* Deposit
        - Abstract: *Collection of objects deposited when testing SWORD interface*
        - Treatment: *Zip archives recognised as content packages are opened and the individual files contained in them are stored. All other files are stored as is.*
        - Collection policy: *Intrallect test system. No guarantee of service, or that deposits will be retained for any length of time.*
        - Mediation: *false*
        - Accepts: **/**

Figure 7 : A screenshot of the Web-based SWORD client

As well as normal functionality, the clients can inject faults into deposits to allow SWORD server implementers to test their servers. For example the client can corrupt the MD5 checksum of the file being sent, to ensure that the server correctly identifies the fault, and reports it back to the client through the correct use of HTTP response error codes.

A Facebook SWORD client is also currently being developed, along with a PHP and CURL- based client library. Further details will be available from the SWORD wiki [20] when it becomes available.

Conclusion

SWORD has achieved what it set out to do, to facilitate deposit into repositories from remote locations in a standard way and to enable the scenarios identified earlier to become a reality. Now is the time to encourage uptake and implementation of both SWORD and APP. This is perhaps the most difficult part, but beginning from the foundations outlined above, with the backing and input of repository developers we hope that SWORD really can help interoperability of deposit become a reality. Not only that, but the process that has led us to where we are now can, it is hoped, help future development activity funded by JISC. Already SWORD is being implemented by a number of projects and repositories, including arXiv [21], ICE [22], ROAD [23] and the Curriculum Online Web Service and Tagging Tool; and with case studies detailing on-the-ground experience in implementing SWORD in repositories and deposit clients to follow, further discussion and dissemination is inevitable.

References

1. SWORD http://www.ukoln.ac.uk/repositories/digirep/index/SWORD
2. JISC Repositories and Preservation Programme http://www.jisc.ac.uk/whatwedo/programmes/programme_rep_pres.aspx
3. SWORD Profile http://www.ukoln.ac.uk/repositories/digirep/index/SWORD_APP_Profile
4. SWORD Implemetations http://www.ukoln.ac.uk/repositories/digirep/index/SWORD_access
5. SWORD clients http://www.ukoln.ac.uk/repositories/digirep/index/SWORD#SWORD_demonstration_clients
6. SWORD Sourceforge site http://sourceforge.net/projects/sword-app/
7. Barker, Phil and Campbell, Lorna, *JISC CETIS Conference 2005 Repsoitories Theme Strand Report* http://metadata.cetis.ac.uk/files/novconf2005.repositories.doc
8. Powell, Andy, A service-oriented view of the JISC Information Environment, UKOLN, November 2005 http://www.ukoln.ac.uk/distributed-systems/jisc-ie/arch/soa/jisc-ie-soa.pdf
9. eFramework 'Add' service genre http://www.e-framework.org/Services/ServiceGenres/ServiceGenreRegistry/Add11/tabid/843/Default.aspx
10. Object Reuse and Exchange http://www.openarchives.org/ore/
11. Deposit API http://www.ukoln.ac.uk/repositories/digirep/index/Deposit_API
12. DSpace http://www.dspace.org/
13. Fedora http://www.fedora-commons.org/
14. EPrints http://www.eprints.org/
15. Intrallect IntraLibrary http://www.intrallect.com/index.php/intrallect/products/
16. Scholarly Works Application Profile http://www.ukoln.ac.uk/repositories/digirep/index/Eprints_Application_Profile
17. Gregario, J. and B. de hOra, "The Atom Publishing Protocol", RFC 5023, October 2007. http://www.ietf.org/rfc/rfc5023.txt
18. Nottingham, M. and R. Sayre, "The Atom Syndication Format", RFC 4287, December 2005. [RFC4287]. http://www.ietf.org/rfc/rfc4287.txt
19. DSpace 'DIM' (DSpace Intermediate Metadata) Format http://wiki.dspace.org/DspaceIntermediateMetadata
20. SWORD wiki http://www.ukoln.ac.uk/repositories/digirep/index/SWORD
21. arXiv http://arxiv.org/
22. Integrated Content Environment (ICE) Project http://ice.usq.edu.au/
23. Robot-generated Open Access data (ROAD) http://www.jisc.ac.uk/whatwedo/programmes/programme_rep_pres/tools/road.aspx

Author Details

Julie Allinson
Digital Library Manager
University of York
and SWORD Project Manager for UKOLN, University of Bath)
Email: j546@york.ac.uk

Sebastien François
EPrints Developer
University of Southampton
Email: sf03r@ecs.soton.ac.uk

Stuart Lewis
Team Leader of the Web Applications and Repository Projects Team
Information Services
Aberystwyth University
Email: stuart.lewis@aber.ac.uk