SWORD V3 BoF

Implmentation, Sustainability & Community

Agenda

- Introductions (10 mins)
  - Name, Affiliation, Experience with SWORD (any version)
- Current Status and Overview (30 mins)
- Development Plans (15 mins)
  - Validation Suite
  - Reference Implementation
- Discussion (25 mins)
- Next Steps (10 mins)

Introductions

- **Oxford/D2P**
  - **Neil Jefferies (Community Lead)**
- Jisc
  - Dom Fripp, John Kaye (Funder)
- Cottage Labs
  - Richard Jones (Technical Lead)

Current Status

- Website: http://swordapp.org
- Alpha release (funded by Jisc)
  - Responses and reviews still welcome
  - Mailing list (contact details on website)
- Final Release only after working code
  - Validation tests
  - Reference Implementation
- Timeline mid-late 2019

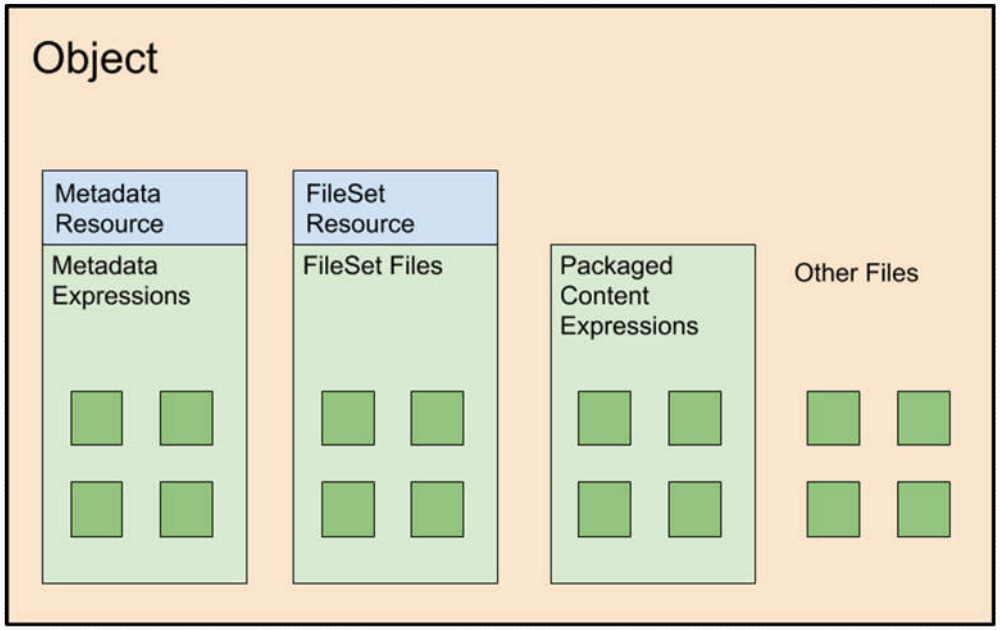
SWORD Overview

- Part of the COAR Next Generation Repository roadmap
- SWORD 3.0 is a protocol enabling clients and servers to communicate around complex digital objects
- Complex digital objects consist of both Metadata and File content, where the Files may be in a variety of formats, there may be many files, and some may be very large.
- It defines semantics for creating, appending, replacing, deleting, and retrieving information about these complex resources.
- It also enables servers to communicate regarding the status of treatment of deposited content, such as exposing ingest workflow information.

A Little History

- The first major version of SWORD (1.3) built upon the Resource creation aspects of AtomPub to enable fire-and-forget package deposit to a server.
- There are use-cases where this is insufficient: e.g. that the depositor wishes to construct a digital artifact file by file over a period of time before deciding that it is time to archive it.
- SWORD 2.0 was developed to service these use cases.
- SWORD 3.0 is a radical departure from SWORD 2.0 moving to a much stricter REST+JSON approach, utilising JSON-LD for alignment with Linked Data, and supporting Research Data Management use cases.

SWORD Object Structure



Key features

- Concurrency Control
- Continued Deposit
- Metadata Deposit
- Package Deposit
- Segmented File Upload
- By Reference Deposit

Concurrency Control

- Servers MAY implement Concurrency Control, to prevent clients from unintentionally overwriting data.
- The Server provides the `ETag` header on every response, which contains a unique version number for the Object.
- The client must then provide the `If-Match` header with every request to change data, which reflects the latest `ETag`

Continued Deposit

- Clients can indicate to a server that there is more content coming, and the item shouldn't be injected into any post-submission workflows by providing the `In-Progress` header.

Metadata Deposit

- SWORD allows the client to deposit arbitrary metadata onto the server through agnostic support for metadata formats.
- SWORD has a default format which MUST be supported by the server, which consists of the set of DCMI Terms expressed as JSON
- During deposit, the client specifies a `Metadata-Format` header which contains the identifier for the format.

Package Deposit

- SWORD allows you to deposit both Files and Metadata simultaneously through support of Packaged Content.
- When depositing Packaged Content, the client indicates information about the format using the `Packaging` header.
- Explicit BagIT support

Segmented File Upload (replaces multipart)

- If a client has a very large file that it wishes to transfer to the server by value, then in may be beneficial to do this in several small operations, rather than as a single large operation.
- In order to transfer a large file, the client can break it down into a number of equally sized segments of binary data (the final segment may be a different size to the rest). It can then initialise a Segmented File Upload with the server, and then transfer the segments. The server will reconstitute these segments into a single file, and then the client may deposit this file by-reference.

By-Reference Deposit

- By-Reference Deposit is when the client provides the server with URLs for Files which it would like the server to retrieve asynchronously to the deposit request itself.
- This could be useful in a number of contexts, such as when the files are very large and are stored on specialist staging hardware, or where the files are already readily available elsewhere.

More detailed overview…

- OR Workshop slides
- https://swordapp.github.io/swordv3/workshop

Validation suite

- Jisc (Funder)
  - Dom Fripp, John Kaye
- Cottage Labs (Development)
  - Richard Jones (Technical Lead)

- Validation for client and server implementations
- Testing for client and server implmentations
- In parallel, but slightly ahead of, reference implementation

Reference Implementation

- NII/WEKO (Pilot User, Funder)
  - Kazu Yamaji, Masaharu Hayashi
- CERN/Invenio (Development)
  - Lars Holm-Nielsen

- Python
- Modular for maximum re-usability
- Feeds back into final spec

Timescale

- …still being worked out
- …mid-late 2019 for completion
- …code availability before then

Discussion Topics

- Role of a potential RDA WG
- Development Partners
- Community

Role of a potential WG

- Under Repository Interoperability IG
  - Logical follow of from BagIT work
  - SWORD provides a transport for Bags
- RDA Endorsement of Spec -> EU Tech Spec
  - Encourages adoption
- Setup of community mechanisms

Development Partners

- https://github.com/swordapp/
- Code available when core functionality complete
  - Alternative implementations
  - Issues/commits for reference implementation
  - Testing
  - Documentation
- ArXiv and CADRE in the pipeline

Community

- Community mechanisms
  - Roles and process
  - RDA Maintenance Group?
  - CLiR
- Stewardship of spec/code/harness
- Recognition/aggregation of other work
  - Links to alternative implementations/possibilities/projects
  - Documentation

Next Steps

- Dev partners
- WG partners
- Community partners