Aligning the eARK4All Archival Information Package and Oxford Common File Layout Specifications

Complementary rather than competing approaches

Neil Jefferies
Bodleian Libraries
University of Oxford, UK
neil.jefferies@bodleian.ox.ac.uk
orcid.org/0000-0003-3311-3741

Karin Bredenberg
Swedish National Archives
Sweden
karin.bredenberg@riksarkivet.se
orcid.org/0000-0003-1627-2361

Angela Dappert
Springer Nature
UK
angela.dappert@springernature.com
orcid.org/0000-0003-2614-6676

Abstract – The E-ARK4ALL[1] project released an alpha version of its Common Specification for Information Packages (CSIP)[2] to be used in the eArchiving Building Block[3] for review at the end of November 2018. Slightly earlier, the Oxford Common File Layout (OCFL)[4] initiative had also released an alpha version of its software independent preservation file organisation specification. While, at first sight, these would appear to attempt to do similar things, they are in fact, largely complementary approaches. While the eARK specification aims to define the logical structure and content of Open Archival Information Systems (OAIS)[5] Information Packages, the OCFL describes how to map any logical digital object layout onto a physical file system in a preservation-friendly manner, as well as identifying the fundamental operations required to manage such objects. This paper provides a brief introduction to the two specifications and then describes how the OCFL can be applied to an E-ARK IP.

Keywords – Preservation, OAIS, AIP, file system, specification

Conference Topics: The Cutting Edge: Technical Infrastructure and Implementation;

Collaboration: a Necessity, an Opportunity or a Luxury?

I. Introduction

The E-ARK4ALL project released an alpha version of its Common Specification for Information Packages (CSIP) to be used in the eArchiving Building Block for review at the end of November 2018. In September, the Oxford Common File Layout (OCFL) initiative had also released an alpha version of its software independent preservation file organisation specification. While, at first sight, these would appear to attempt to do similar things, they are in fact, largely complementary approaches.

While neither specification is completely finalised at the time of writing, they are largely complete so it is an opportune time to examine how, in practice, they might be aligned. The examination presented here is quite high level since it is based on members of each of the respective communities reading of the other's specification while completing work on their own documents. However, it is possible to usefully identify some basic workable principles and potential areas for further discussion. As always, the fine detail will only emerge when code comes to be written and systems to be built.

II. The Oxford Common File Layout

he Oxford Common File Layout (OCFL) initiative began as a discussion among digital repository practitioners about the ideal layout and characteristics for persisted objects, from a computational and conceptual point of view. It is named, as with

[1] E-ARK4ALL Project, http://e-ark4all.eu/

[2] E-ARK Common Specification for Information Packages, http://earkcsip.dilcis.eu/

[3] eArchving Building Block, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eArchiving

[4] Oxford Common File Layout, https://ocfl.io/

[5] Open Archival Information System, http://www.oais.info/

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

a number of other projects[1], for the location of this initial discussion. It has since grown into an open community effort defining an application independent way of storing versioned digital objects with a focus on long term digital preservation.

The approach is informed by three simple observations:

1) Archived objects change relatively slowly compared to archival software, and are rarely deleted.
2) Migration by export and re-ingest is generally slow and error-prone. Data is most at risk of loss or corruption when it is moved or migrated, rather than at-rest.
3) File systems, in particular POSIX-style[2] file systems, have been the most consistently implemented and widely tested Application Programming Interfaces (API's) for accessing storage in any form.

- *Objectives*

The OCFL also builds on practical experience gained from previous work on related initiatives, such as Stanford's MOAB[3] and BagIt[4], both in order to avoid some of their pitfalls and bottlenecks, but also with a view towards interoperability and easy migration.

Consequently, the OCFL is constructed with five main objectives, most of which readily map to the more hardware focussed elements of the emerging Digital Preservation Storage Criteria[5].

- *Completeness*

All the data and metadata required to understand and render or execute an object should be stored within the directory that represents the object on the filesystem. This ensures that a repository can be rebuilt from scratch given just the files on storage. It also aligns very well with the construction of an E-ARK AIP.

- *Parsability*

The structure of content stored using the OCFL should be easy to access and interpret by humans and machines. This ensures that the content can be understood in the absence of the original systems and software. To this end, as with some parts of the E-ARK CSIP, the OCFL allows for embedded documentation. This is crucial since the OCFL does not mandate the internal structure of the objects that it stores.

- *Versioning*

The OCFL is designed with the expectation that digital objects will change over time, even if only as a result of preservation activity. It therefore supports object versioning, provides a mechanism for recording version history and allows access to previous versions.

- *Robustness*

Robustness against errors, corruption, and migration between storage technologies is a basic requirement of any preservation storage system. The OCFL uses SHA256 or SHA512 for content addressing and, consequently, for default fixity provision, which operates at both a file and object version level.

- *Portability*

The ability to store content on different storage infrastructures and migrate between them is essential for maintaining diversity as a hedge against both obsolescence and systemic technological failure. The OCFL requires a minimal set of file system features to operate, and proscribes the use of additional features if they have the potential to affect portability.

To these five criteria we can also add *efficiency* as an additional consideration, which is manifest in several ways. The OCFL is designed to support forward-delta differencing between object versions so that components that do not change between versions are only stored once, reducing the storage overhead, and hence cost, for versioning. It is also constructed to minimise the number of file system

[1] e.g. Dublin Core Metadata Initiative http://dublincore.org/; Portland Common Data Model https://pcdm.org/2016/04/18/models

[2] Posix - The Open Group Library, http://pubs.opengroup.org/onlinepubs/9699919799/

[3] The Moab Design for Digital Object Versioning, https://journal.code4lib.org/articles/8482

[4] The BagIt File Packaging Format (V1.0), https://tools.ietf.org/html/rfc8493

[5] Digital Preservation Storage Criteria (Version 2.0), https://osf.io/sjc6u/

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

operations that are involved in scanning OCFL structures for validation or rebuilding purposes. This had emerged as a key bottleneck with the design of the MOAB file layout.

Specific Features

Without going into too much technical detail on the specification[1], several features of OCFL should be highlighted with respect to the implementation of E-ARK AIP's.

- *The OCFL Storage Root*

While the CSIP specification deals purely with the internal structure of the object, the OCFL also describes how collections of objects should be managed in a file system, as a necessary feature for building archival systems and repositories. However, this is done in a way that is consistent with the CSIP design principles.

The "OCFL Storage Root" is the top level directory for storing OCFL Objects and should contain both a copy of the OCFL specification and a conformance declaration that indicates which version is implemented - as a check that the correct documentation is present, if nothing else!

In addition, the Storage Root should also contain documentation that describes the scheme used for distributing OCFL Objects across directories on the file system in a balanced way that maintains the efficiency of file system operations. Unfortunately, there is no single scheme that is optimal for all use cases, so various options and their relative merits are discussed in the Implementation Notes[2]. As a general principle, this "File System Disposition" should programmatically derive the path of an OCFL Object from its unique identifier.

- *OCFL Objects*

An OCFL Object is completely contained within one directory termed the "OCFL Object Root". At the top level of the directory there must be an object version conformance declaration, an object inventory, which is discussed further in the next section, and a digest for the inventory for validation purposes. Importantly, the OCFL only requires the version conformance to apply to the top level inventory and the most recent version of the object. This permits legacy versions to be included in an OCFL object without rewriting or otherwise tampering with them.

The content of the object is contained in sequentially numbered version directories within the Object Root, with all but the most recent version considered immutable. No content is stored outside the version directories. An optional *Logs* directory may exist in the Object Root to store information that does not affect the content of the object - for example, records of periodic fixity checks that identify no problems.

- *The OCFL Inventory*

The Inventory is the principal metadata structure mandated by the OCFL specification and is the primary mechanism through which most of its functionality is realised. Its primary function is to map between *content paths*, which point to physical files on storage, and *logical paths*, which indicate where these files appear in the logical representation of a version of an OCFL object. This distinction is important for a number of reasons:

1. *Deduplication* - the OCFL supports deduplication within an object, so that once a file exists in storage, with a given content path, all references to that particular content, regardless of filename are merely different logical paths that reference the single content path.
2. Filesystem Limitations - File systems may have limits on paths (such as length or restricted character sets[3]) that may mean that the object structure cannot be represented accurately on the file system. However, logical paths are not restricted in this way and thus object structure can be preserved regardless of file system restrictions.
3. *Efficiency* - Complex directory structures can be quite inefficient to traverse. For complex objects, the OCFL Inventory allows content paths to exist in a simplified hierarchy while retaining complexity at the logical level.

[1] *Oxford Common File Layout Specification 0.2,* https://ocfl.io/0.2/spec/

[2] *Oxford Common File Layout Implementation Notes 0.2,* https://ocfl.io/0.2/implementation-notes/

[3] *Comparison of file systems,* https://en.wikipedia.org/wiki/Comparison_of_file_systems

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

4. *Future Proofing* - Longer term, storage systems, such as content addressable object stores, are appearing that do not have a hierarchical file system. The OCFL Inventory is designed to be a functional object description even in this case.
5. *Optimisation* - Storage systems that handle large numbers of small files well tend to handle very large files poorly, and vice versa. While it is not the default behaviour, there is no reason content paths cannot point to separate storage locations for problematic files. This is a more robust approach than file segmentation or stressing unsuitable file systems.

The Inventory is formatted using JavaScript Object Notation (JSON)[1] because it is compact and easy to read for computers and humans. It has three main sections:

1. A preamble section that, most importantly, includes at least one unique identifier for the object.
2. A *manifest* section that lists every file in the object along with its digest. These are the *content paths*.
3. One *version* section for each version of the object that exists. Within each version section, a *state* section lists the digests for each of the files in the version (which must exist in the manifest section) alongside the *logical path(s)* for that file.

There is also an optional *fixity* section for additional fixity digests that is formatted in the same way as the manifest section.

Additionally, it is recommended that each version directory holds a copy of the inventory as it was at the time of its creation. This has the effect of the current version providing an additional copy of the inventory and allowing rapid rollback of the entire object state to an earlier version in the event of errors during updates.

Basic Operations

In addition to specifying how files should be organised, the OCFL Implementation Notes go further and define how basic operations on OCFL objects should be implemented with respect to inventory maintenance and the requirement for previous versions of objects to be immutable.

- *Inheritance*

By default, a new version of an OCFL Object inherits all the filenames (logical paths) and file contents (content paths) from the previous version. This serves as the basis against which changes are applied to create a new version. A newly created OCFL Object, obviously, inherits nothing and is populated by file additions.

- *Addition*

Adds a new logical path and new content with a content path in the current version. The logical path cannot exist in the previous version of the object, and the content cannot have existed in any earlier versions of the object.

- *Updating*

Changes the content pointed to by a logical path, which must exist in the previous version of the OCFL Object. A new content path is created in the current version of the object. The content cannot have existed in any earlier versions of the object.

- *Renaming*

Changes the logical path of existing content. The logical path cannot exist in the previous version of the OCFL Object.

- *Deletion*

Removes a logical path from the current version of an OCFL Object. The logical path and content remain available in earlier versions of the object.

- *Reinstatement*

Makes content from a version earlier than the previous version available in the current version of an OCFL Object. The content must exist in an earlier version, and not the previous version. The logical path may exist in the previous version, effectively updating the file path with older content, or it may not, effectively adding the older content as a new file.

- *Purging*

Purging, as distinct from deletion, covers the complete removal of content from all versions of an OCFL Object. This is a special case that is not

[1] *Introducing JSON,* https://www.json.org/

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

supported as part of regular OCFL versioning operations since it breaks the previous version immutability requirement. Ideally, a new OCFL Object with an amended version history should be created.

Community

The OCFL Community Google Group[1] is where discussion takes place and meeting announcements are made. At the time of writing, community conference calls are scheduled monthly. The specification and implementation notes are managed on Github2 and everyone is welcome to raise issues or even submit pull requests.

III. The e-ARK Common Specification for Information Packages

In 2017 the European Archival Records and Knowledge Preservation Project (E-ARK project[3]) delivered its draft common specifications for information packages to the Digital information LifeCycle Interoperability Standards Board (DILCIS Board[4]) . The board is responsible for the enhancement, maintenance, continuous development and endorsement of specifications. Specifications concern information packages as well as Content Information Types. The information package specifications describe OAIS reference model packages for archival transfer, but can also be used for other types of transfer. Content Information Type Specifications (CITS) describe the content itself as well as its structure within the package in order to facilitate easier content validation.

The DILCIS Board specifications are the core specifications in the Connecting Europe Facility Building Block eArchiving[5].

The drafts for information packages are:

1. Common Specification for Information Packages
2. Specification for SIP
3. Specification for AIP
4. Specification for DIP

These draft specifications have been updated, enhanced and published in version 2.0 during spring 2019. The specifications are available as pdf at the DILCIS board's webpage and as markdown in GitHub[6] accompanied with METS profiles and XML-schemas. Questions and issues are handled in each specification's GitHub repository issue tracker. GitHub has been chosen as the transparent platform in which users can follow progress, see notes and comment on the current work.

- *Common Specification for Information Packages (CSIP)*[7]

The core package specification describes general principles and requirements for an information package, that are shared by all types of information package in the OAIS reference model.

The principles present a conceptual view of an Information Package, including an overall IP data model, and use of data and metadata. These principles could be implemented with a physical directory structure and the requirements are expressed with the Metadata Encoding and Transmission Standard (METS)[8].

The principles describe:

- General requirements for the use of the specification;
- Identification requirements ranging from identification of the package to identification of the transferred digital files;
- Structural requirements for the content in the package, for example how different kinds of metadata should be structured and added;
- Metadata requirements outlining the use of standards for describing data.

[1] Oxford Common File Layout Community, https://groups.google.com/forum/#!forum/ocfl-community

[2] The OCFL Specifications (WIP), https://github.com/OCFL/spec

[3] European Archival Records and Knowledge Preservation, http://eark-project.com/

[4] The Digital Information LifeCycle Interoperability Standards Board, http://dilcis.eu/

[5] CEF eArchiving BB, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eArchiving

[6] DILCIS Board in GitHub https://github.com/DILCISBoard

[7] E-ARK CSIP http://earkcsip.dilcis.eu/

[8] Metadata Encoding and Transmission Standards http://www.loc.gov/standards/mets/

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

The requirements are expressed using METS and PREservation Metadata Implementation Strategies (PREMIS)[1]. METS describes the requirements on the package level and PREMIS defines the preservation metadata needed, especially those for the AIP. The METS specification available both as a METS Profile and in text form in the specification expresses the requirements for how each part of METS is to be used and how it fulfills the CSIP principles. A validation tool has been created to support automatic metadata validation.

In summary, the requirements specify:

- how to identify the package ;
- how to describe the type of content;
- how to link descriptive, technical and provenance metadata;
- which files are to be contained in the package, where each file is described by its :
  - File name
  - Path
  - Mime Type
  - File size
  - Checksum
  - Creation date
- The mandatory METS structural map which describes the package structure on a high level.

Sometimes an IP is large, reaching tera bytes in size. This is cumbersome to handle, both for the submitter and the receiver of the IP. An example is an IP that contains a whole database from an electronic records management system with records comprising over a year. Therefore, CSIP contains a section that discusses how to split large IP's. In a coming version of CSIP this section will be extended and give guidance on splitting large packages. Splitting leads to more than one IP being created. The full IP is established by creating package referencing connections between the split IP packages. Draft specification texts are currently being written that describe how to carry out the splitting, as well as to how to describe the different parts and their relationships. These updates will be published after a review period.

[1] PREservation Metadata Implementation Strategies, http://www.loc.gov/standards/premis/

E-ARK profiles building upon CSIP

E-ARK SIP[2], E-ARK AIP[3] and E-ARK DIP[4] profiles all use CSIP as their basis and extend the CSIP requirements with requirements for their specific type of information package. Some requirements extend existing specific CSIP elements, for example by requiring the value for the element describing the OAIS Reference Model type of the package being set to the appropriate value (SIP/AIP/DIP). The focus in this paper is the AIP. To learn more about the E-ARK SIP and E-ARK DIP please refer to their available specifications.

E-ARK AIP

- The objectives for the E-ARK AIP are as follows:
- To define a generic structure of the AIP format in a way that it is suitable for a wide variety of data types, such as document and image collections, archival records, databases or geographical data.
- To recommend a set of metadata standards related to the structural and the preservation aspects of the AIP.
- To ensure the format is suitable for storing large quantities of data.
- To mitigate the potential preservation risk of repository obsolescence by implementing a repository succession strategy.

The purpose of defining a standard format for the archival information package is to pave the way for simplified repository migration. Given the increasing amount of digital content archives need to safeguard, changing the repository solution should be based on a standard exchange format. This is to say that a data repository solution provider does not necessarily have to implement this format as the internal storage format, but it should at least support exporting AIPs. By this way, the costly procedure of exporting AIP data as Dissemination Information Packages (DIPs), producing SIPs for the new repository solution, and ingesting them again in the new repository can be simplified. Data repository solution providers know what kind of data they can expect if they choose to replace an existing

[2] E-ARK SIP, https://earksip.dilcis.eu/

[3] E-ARK AIP, https://earkaip.dilcis.eu/

[4] E-ARK DIP, https://earkdip.dilcis.eu/

repository solution. An E-ARK compliant repository solution should be able to immediately analyse and incorporate existing data in the form of AIPs without the need of applying data transformations or having to fulfil varying SIP creation requirements.

Generally, a variety of repository systems are being developed by different providers. The way the AIP is stored often depends on specific requirements which have been addressed according to the needs of their respective customers. For this reason, the purpose of the E-ARK AIP format is not to impose a common storage format that all repository systems need to implement. While it can be used as an archival storage format, it can also be seen as a format that makes system migration easier.

IV. Alignment of The E-ARK AIP and The OCFL

The OCFL is engineered from the viewpoint that a digital object should be considered a greater whole, comprising several streams of information that can be arbitrarily labelled data or metadata but all of which contribute to the intellectual content of the object. Consequently, it does not make any assumptions about the internal structure or composition of a digital object, which is key to the alignment between the E-ARK and OCFL specifications. In this respect, the CSIP specification and the extension profile for E-ARK AIP can be considered as filling this intentional gap in the OCFL for a number of use cases, to provide a more complete approach.

A very simplistic implementation could therefore just encapsulate an entire eARK AIP structure within an OCFL object. However, since the OCFL provides file mechanisms for fixity, versioning, deduplication and logging that are optimised for simplicity and computational efficiency, a more nuanced and functional approach would be to consider where these could interoperate with corresponding elements within the AIP structure.

The essential part of the alignment of the two approaches is that the AIP structure is implemented at the *logical* level within the OCFL. The OCFL client software can then handle versioning, deduplication and other features transparently but present the AIP structure when queried by other software.

Fixity

As stated previously, the OCFL supports SHA512 or SHA256 as the default digests for its content addressability, however other algorithms are permitted and the *fixity* section of the inventory allows storage of hashes generated by additional algorithms. These could be extracted from an AIP (by parsing METS files or manifest.txt, if it exists) as part of object creation or version updating. Using a SHA256-based implementation of OCFL obviously aligns well with the E-ARK AIP since these values can be shared.

The OCFL can technically support the use of other hash functions for *manifest* content addressing, but validity checks will generate errors for fixity algorithms that are considered broken/deprecated (e.g. MD5[1]). As a result, using the other fixity algorithms in place of SHA512 or SHA256 is not advised.

Copying digests from the OCFL inventory into the AIP is also possible but requires a little more care, since OCFL includes digests for every part of the AIP. Such a process would therefore need to exclude the METS and/or manifest files that would be updated.

Versioning and Deduplication

The OCFL differs from the AIP specification in the way that versions are treated, since it makes no assumptions about the types of changes that may occur. It also makes the version history explicit in the manifest with *state sections for every version.*

The E-ARK AIP versioning model is, in some respects, analogous to the OCFL model, in that the parent AIP can be seen as equivalent to the OCFL Object Root with child AIP's equivalent to OCFL versions. However, the AIP model is somewhat encumbered by the requirement for the parent to be compliant with the CSIP which results in additional complexity. In addition, using the AIP model can require multiple file parsing operations to determine version differences whereas the OCFL requires minimal processing.

This can become a significant overhead when objects are referenced externally, since, for

[1] G. Ramirez, MD5: The broken algorithm, https://blog.avira.com/md5-the-broken-algorithm/

referential integrity, the version of an object current at the time of citation should be readily accessible, using a protocol such as Memento[1]. Being able to easily identify the difference between any two versions is also essential for the efficient synchronisation of distributed storage systems that are maintained asynchronously. This was a bottleneck encountered, in practice, with systems that use the MOAB layout.

Thus, while it is perfectly possible to implement the parent-child AIP versioning model as distinct AIP's in the OCFL, a more efficient approach would be to create new versions of an AIP within a single OCFL Object, allowing the OCFL client to deduplicate the common elements between versions and providing quick access to the version history. This also eliminates the duplication of information between parent and child IP's, along with the consequent maintenance overheads.

The OCFL is constructed so that all changes to an OCFL Object are additions to its contents. This allows AIP's to be updatable but, at the same time, forces each version to be immutable but without incurring undue storage overheads. Using the *reinstatement* mechanism described earlier, it also allows rollback of failed DP actions such as migration at any point after the event[2].

Logging

The OCFL expects new versions to be created when a meaningful change is made to an object. A periodic virus scan or fixity check with a null result thus does not automatically result in the generation of a new version. In practice, there are a wide variety of events that may impact storage but which are largely invisible to preservation systems without explicit action. Examples would include operating system file system maintenance, and hard drive replacement and subsequent array rebuilding operations.

In practice, then, these can be potentially numerous and creating a new AIP each time would not make sense either. However, there is merit in capturing this information for recovery and audit purposes. In the OCFL, these can be captured in the *logs* directory which is outside the object version structure. PREMIS is suggested, but not mandated for this purpose in the OCFL, but it would be sensible to do so if using E-ARK AIP's. If desired, these logs could then be periodically added to a new version of the AIP to embed this audit trail without undue AIP version proliferation.

In the E-ARK AIP the use of PREMIS is mandatory, including the use of events. The full description of the PREMIS use in the specifications and the eArchiving Building Block is not ready at the time of the writing. The work is ongoing and the use of the semantic units of PREMIS will be described in its own document to allow it to be used in all the different IP's easily.

Pathname Mappings

Complex objects can contain paths that are not supported by the file system being used for preservation, especially if they have been imported from another system. This can be for reasons of length, number of directory levels or character encodings, amongst others. The OCFL handles this by allowing long Unicode *logical paths* while implementing *content paths* on storage that may be shortened or use different character encodings. No specific algorithm is mandated since the mappings are explicit in the OCFL inventory.

If the AIP is implemented at the OCFL logical level, then complex AIP structures need not be subject to such file system limits.

V. Conclusion

The choices that can be made when creating a digital archiving approach are numerous, starting with what you consider to be the first AIP. Should it be the SIP that has just been transferred and put directly into preservation storage so you can go back if everything gets demolished through a "bad decision in migration" further down the preservation journey?

Are we concerned more with the preservation of bitstreams as standalone entities or with the

[1] HTTP Framework for Time-Based Access to Resource States -- Memento, https://mementoweb.org/guide/rfc/

[2] You will thanks us for this, believe me! (Neil Jefferies)

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.

preservation of *knowledge*, where the meaning of an object can be largely determined by its relationships to other objects - relationships that necessarily change over time as a result of human discourse? Thus we need to consider how to design systems to capture and preserve this metadata and when and how often to capture this in new AIP versions.

At a technical level, we need to create systems that support the curatorial requirements of digital preservation yet also address the unavoidable limitations of the underlying computational and storage technologies.

Both OCFL and the E-ARK AIP standards go some way to addressing these issues, whilst accepting that not everyone will necessarily make the same decisions about their approaches, for entirely logical reasons. This preliminary analysis shows that, in many respects, the standards are largely complementary in that their primary foci are differ

ent aspects of the broader digital preservation problem space - the structure of preserved digital objects, and the efficient storage and management of them, respectively. This, somewhat fortuitous, "separation of concerns" is considered good practice in terms of systems design.

It can be seen that abstracting the logical structure of an object from the storage structure with the simple logic embodied in the OCFL inventory permits the E-ARK AIP to be realised over a broader range of platforms, very much in keeping with its purpose. It even has the potential to allow the use of object stores which do not implement hierarchical path systems at all.

Both efforts are still in the development phase and more work is required to bring them to fruition. However, this paper shows that there is value in working together, learning and contributing to each other. One early recommendation from the OCFL community to the E-ARK CSIP community is to look further into the selection of checksum algorithms. An area that probably requires further discussion on both sides is the issue of object/AIP dependencies - where one object, such as a collection, depends on the existence of others.

More recommendations and comments will most certainly pass between the groups as we move forward, particularly once we begin to write code and develop systems. Collaboration between efforts can only be beneficial!

...Diversity and choice is always good for Digital Preservation - as is discourse and alignment between concerns and communities.

16th International Conference on Digital Preservation
iPRES 2019, Amsterdam, The Netherlands.