The Archive and Package (arcp) URI scheme

1st Stian Soiland-Reyes
School of Computer Science
The University of Manchester
Manchester, UK
https://orcid.org/0000-0001-9842-9718

2nd Marcos Cáceres
Mozilla Corporation
Melbourne, Australia
https://marcosc.com/

***Abstract*—The arcp URI scheme is introduced for location-independent identifiers to consume or reference hypermedia and linked data resources bundled inside a file archive, as well as to resolve archived resources within programmatic frameworks for Research Objects. The Research Object for this article is available at http://s11.no/2018/arcp.html#ro**

***Index Terms*—Uniform resource locators, Semantic Web, Persistent identifiers, Identity management systems, Data compression, Hypertext systems, Distributed information systems, Content-based retrieval**

I. Background

Archive formats like BagIt [1] have been recognized as important for preservation and transferring of datasets and other digital resources [2]. More specific examples include COMBINE archives [3] for systems biology, CDF [4] for astronomy data, as well as the more general HDF5 [5] which is also used for meteorological data. For the purpose of this article an *archive* is a collection of data files with related metadata, typically packaged in a compressed file format like *.zip* or *.tar.gz*.

One challenge with regards to embedding Linked Data in such archives is how to reliably generate and resolve internal URLs, for instance `<dataset13.zip>` may contain an RDF Turtle file `<metadata/description.ttl>` to describe the CSV file `<data/survey.csv>` — but in order to correctly reference that file it will either have to use a relative path `<../data/survey.csv>` or some pre-existing Web URL like `<http://example.com/dataset13/survey.csv>`.

The *Research Object Bundle* [6] format suggested re-using the app URI scheme for minting absolute URIs from relative paths of resources within a ZIP file. The *app URL scheme* [7] was originally intended for packaged web applications, where each application would get their own namespace like `<app://c6179148-3cde-4435-8e66-304453f89d59/>` with paths resolved from the corresponding application package ZIP file. However the app URL scheme did not progress further on the W3C Recommendation track, and this approach was abandoned in favour of the combination of Web App Manifest [8] and Service Workers [9]. Together these technologies reuse the http/https origin URL of a downloaded application manifest together with relative links, while also allowing a web application to work offline.

This work has been done as part of the BioExcel CoE, a project funded by the European Commision (H2020-EINFRA-2015-1-675728).

https://doi.org/10.1109/eScience.2018.00018

II. The Archive and Package (arcp) URI scheme

Inspired by the app URL scheme we defined the Archive and Package (arcp) URI scheme [10], an IETF Internet-Draft which specifies how to mint URIs to reference resources within any archive or package, independent of archive format or location.

The primary use case for arcp is for consuming applications, which may receive an archive through various ways, like file upload from a web browser or by reference to a dataset in a repository like Zenodo or FigShare. In order to parse Linked Data resources (say to expose them for SPARQL queries), they will need to generate a base URL for the root of the archive.

It should be clear that using local file URIs [10] for extracted archives like `<file:///tmp/tmp.cUK6ERfdBe/>` do not serve well for this purpose, as they are not universally unique, are difficult to create consistently, and may introduce security risks of attacks like `<../../etc/passwd>`. Similarly it may be inappropriate to mint new web based URIs like `<http://repo.example.com/cUK6ERfdBe/>` as web presence should not be a requirement to process a linked data archive, in particular as processing may occur on a laptop or a cloud node with no public IP address.

A. Identifier structure

By definition an arcp identifier is an URI [12] with three parts, as shown in figure 1.

`<arcp://prefix,namespace/path>`

Fig. 1. Structure of arcp identifier

The arcp Internet-Draft specifies three initial `prefix` values: `uuid`, `ni` and `name`, each which defines how to identify a particular archive by a corresponding `namespace`. These namespaces are not intended to be directly resolvable without prior knowledge of the corresponding archive.

The `path` is the folder and file path within the archive, represented as an URI path [12] e.g. `/file.txt` or

`/my%20project/about/intro.doc` — using percent-escaping if needed. The root folder `/` represent the archive itself.

B. UUID-based identifiers

The simplest case for temporary sandbox processing of an archive with arcp is to generate a new random UUIDv4 [13], e.g.:

```
c6179148-3cde-4435-8e66-304453f89d59
```

From this the corresponding arcp URI is:

```
<arcp://uuid,c6179148-3cde-4435-8e66-304453f89d59/>
```

This *base URI* can be used when resolving relative URI references, e.g. if `<metadata/description.ttl>` references `<../data/survey.csv>` we find the absolute URIs:

```
<arcp://uuid,c6179148-3cde-4435-8e66-304453f89d59
  /metadata/description.ttl>
<arcp://uuid,c6179148-3cde-4435-8e66-304453f89d59
  /data/survey.csv>
```

The application is then able to do translation from arcp to local paths using URI parsing libraries to select the URI path, and augment that to the locally extracted path. Such arcp identifiers are temporary in nature, but the application can maintain a mapping from the UUID to the archive and perform extraction on demand, or the archive can self-declare its UUID, such as the `External-Identifier` header in BagIt [1].

arcp also suggests how a UUID can be reliably created from the URL location of an archive. For instance, an application may be processing a file from:

```
http://example.com/download/archive13.zip>
```

The application can calculate the *name-based UUIDv5* [13] by SHA1 hashing the URL string and mint:

```
<arcp://d9f0b57d-0504-5e9a-abae-f5f2b8c49b94/>
```

With this method anyone processing that archive URL will always get the same arcp base URI, however the application will still need to maintain a mapping to find the original archive URL. Location-based arcp identifiers may also not be ideal for preservation purposes, as the archive might change upstream or move to a different location.

C. Hash-based identifiers

For this arcp defines a hash-based method, where the bytes of the archive file is used to find a checksum-based identifier based on the Naming Things With Hashes (ni) URI scheme [14]. For instance if the sha-256 checksum of a Zip file is in hexadecimal:

```
7f83b1657ff1fc53b92dc18148a1d65d
fc2d4b1fa3d677284addd200126d9069
```

After base64 encoding the `ni:` uri would be:

```
<ni:///sha-256;
  f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk>
```

The corresponding arcp base URIs for resources within the archive is thus:

```
<arcp://ni,sha-256;
  f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk/>
```

With this method, anyone processing the byte-wise equal archive (using the same hash method) will get the same identifier.

Another advantage is that hash-identified archives can be retrieved from a NI resolver [14] using well known paths [15]:

```
<http://repo.example.com/.well-known/ni/sha-256
  /f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk>
```

Clients can verify the checksum of the downloaded archive, so any accepting resolver endpoint can be used.

D. Name-based identifiers

Finally, paying homage to its origin in app URLs, arcp can use a system-based app name. This is a suggested mechanism for resolving resources of an application package installed in a runtime system like Android applicationId or Java package name, where an application identifier can be directly reused in arcp for URIs within that runtime system, e.g. to reference the resource `styles/resource1.css` within the installed package `com.example.myapp` one can use the URI:

```
<arcp://name,com.example.myapp/styles/resource1.css>
```

As application package content do not necessarily correspond to archive file listings, it is open-ended how name-based arcp identifiers can be resolved, and indeed package content may vary per operating system, device type or application version, and so name-based arcp identifiers should be treated as system-local identifiers similar to `file:///` URIs [11], but within a particular programming framework.

III. Related work

A. Archive fragments

Without using arcp one could in theory still reference files within archives at an URL with `#` fragments:

```
<http://example.com/download
  /archive13.zip#data/survey.csv>
```

Unlike formats like *text/html* or *application/pdf*, most archive media formats like *application/zip* unfortunately do not define a fragment syntax, and some major types like *tar.gz* are not even listed in the IANA media types registry. Therefore this would be an ad-hoc approach which still needs to clarify details in order to be interoperable, for instance character escaping, if the root is `#` or `#/`, and how to reference nested fragment identifiers in hypermedia within archived resources.

B. File URIs

As argued above, file URLs [11] that represent local directories are fragile and not globally unique. It is perhaps less known that file URLs can specify a host name:

```
<file://host.example.com
  /home/alice/extracted/archive13/>
```

The above references a file path on the machine with the fully qualified domain name (FQDN) `host.example.com`. The usually empty hostname is equivalent to `localhost`.

This approach may be used if both the hostname and extracted path are stable (e.g. a repository file server), but this faces the same challenges as minting http/https URLs, which in many cases would be preferable as they are also globally resolvable.

An ad-hoc possibility here could be to use a UUID [13] as "hostname" to represent an archive's internal file system:

```
file://8f26cb8c-617e-46b4-bc48-e650bf70f33d
  /data/survey.csv/>
```

This is technically permittable as the `file:` URL scheme [11] do not define any particular connection protocols, and an UUID is unlikely to be a valid hostname in DNS. Such `file:` URIs could however cause confusion against file paths on `localhost`, for instance Firefox 62.0 opens file://8cd4ce0d-4a41-4b4e-bfdd-1e2d0495f714/ to browse the local file system.

C. JAR URLs

If we restrict usage to ZIP files at a known URL, then they are in theory also valid JAR files, and we can address files with the *jar URL scheme*:

```
<jar:http://example.com
  /download/archive13.zip!/data/survey.csv>
```

Here relative URIs may not parse well, as it is easy to accidentally climb out of `!/`, and technically the JAR URI scheme is missing the familiar `://` to indicate for URI parser libraries that it is indeed an hierarchical URI scheme [12].

D. Object Reuse and Exchange proxies

OAI-ORE [16] defines proxies to represent a resource as aggregated in a collection; these can be used to model archives [17], but ORE proxies face two problems: How to represent the file path, and how to identify the proxy so it can be used as a reference in Linked Data. The resource must be identified using two triples of `ore:proxyFor` (the archived file) and `ore:proxyIn` (the archive); but this reduces to the same problem of identifying the file. The ni URI [14] for the file bytes can in theory be used to identify the file, but the other missing information is the file path and name, which usually convey meaning for users.

The Research Object ontology's `FolderEntry` specializes the `ore:Proxy` to add a property `ro:entryName` to indicate the filename, as exemplified in figure 2, but to find the full archive file path one would have to traverse the parent folder's `ro:entryName`. In either case there is no defined method to predictably generate unique identifiers for the ORE proxies themselves, although the RO Bundle specification recommend they should be randomly generated `urn:uuid` URIs, which would not be compatible with relative URIs within an archive.

E. Publishing file systems as Linked Data

F2R [18], using the Nepomuk File Ontology [19], defines a way to publish file systems as Linked Data, where a server endpoint exposes the files and their file system metadata.

F2R URIs are localized to an endpoint and an free-text named file system, e.g. `mysource`, and files are identified with UUIDs:

```
<http://f2r.example.com
  /mysource/09b205be-bj80{4ab9{8ddc-802be95220bb>
```

Using the same example as for OAI-ORE we can combine F2R with PAV [20], as shown in figure 3.

The F2R approach have similar disadvantages as JAR and OAI-ORE; in that the URIs do not support relative path resolution, that a web endpoint must be set up, and that the file paths are hidden through multiple steps. In addition one would need to assigned a corresponding file system name like `mysource`, although one may use a single file system as exemplified above and use `belongsToContainer` to treat archive files as if they are folders.

F. EPUB canonical fragment identifiers

EPUB is a standard for hypermedia eBooks. RO Bundle [6] is based on the EPUB Open Container Format [21]. EPUB Canonical Fragment Identifiers [22] can link to nested XML elements of an publication using a variation of XPath with doubled indexes:

```
<http://example.com/book.epub
  #epubcfi(/6/4[chap01ref]!/4[body01]/10[para05])>
```

The above example show an example to a paragraph with an ePub book. Here `/6` refer to the 3rd element of the root manifest's `<package>` element (which in ePub is always `<spine>`), then `/4[chap01ref]` is the second element `<itemref>` with `xml:id="chap01ref"`.

The `!` character means the element's reference is followed to open the corresponding XML file, where `/4[body01]` is the 2nd element with id `body01`, traversed to find the 5th element with id `para05`.

While this is quite a powerful construct that can refer to any XML element of nested documents, even sentences or words, it seems rather contrived and inflexible. The major limitation is that ePub archive resources are not identified by file paths, but must be addressable through rather rigid XML structures (order can't change), thus this approach is not appropriate for archives without an XML manifest. Even if using a RDF/XML manifest it would be inadvisable to assume a fixed order of it's XML elements. It seems however an appropriate reference scheme for ePub documents, which generallyhave a fixed reading order.

IV. ARCP IMPLEMENTATIONS

The **arcp Python library** [23] was developed to help creating, parsing and validating arcp URIs. In particular it can generate arcp based on random UUIDs, URL locations, names and hashing archive bytes. The arcp parser recognize the arcp prefix and can extract UUIDs or hashes, and can generate

```
@prefix ore: <http://www.openarchives.org/ore/terms/> .
@prefix ro: <http://purl.org/wf4ever/ro#> .
<urn:uuid:c5971b62-72e6-4a8f-8b0b-944065e0d5c8> a ore:Proxy, ro:FolderEntry ;
    ore:proxyFor <ni:///sha-256;f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk> ;
    ore:proxyIn <urn:uuid:efb14c0a-3cd5-4d78-a168-f246d18bde39> ;
    ro:entryName "survey.csv" .
<urn:uuid:efb14c0a-3cd5-4d78-a168-f246d18bde39> a ore:Aggregation, ro:Folder .
<urn:uuid:24b34ecb-e46b-46ec-be36-a18dbba90247> a ore:Proxy, ro:FolderEntry ;
    ore:proxyFor <urn:uuid:efb14c0a-3cd5-4d78-a168-f246d18bde39> ;
    ore:proxyIn <http://example.com/download/archive13.zip> ;
    ro:entryName "data/" .
```

Fig. 2. RDF Turtle example of how a file with the sha256 checksum `7f83b1...6d9069` could be described using RO folders and ORE proxies to belong to `<data/survey.csv>` within the archive downloaded from `<http://example.com/download/archive13.zip>`

```
@base <http://f2r.example.com/mysource/> .
@prefix nfo: <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#> .
<c5971b62-72e6-4a8f-8b0b-944065e0d5c8> a nfo:ArchiveItem;
    nfo:fileName "survey.csv" ;
    nfo:belongsToContainer <24b34ecb-e46b-46ec-be36-a18dbba90247> .
<24b34ecb-e46b-46ec-be36-a18dbba90247> a nfo:ArchiveItem;
    nfo:fileName "data" ;
    nfo:belongsToContainer <5d0a538a-ef00-48b6-bcb2-f561effe9fe5> .
<5d0a538a-ef00-48b6-bcb2-f561effe9fe5> a nfo:ArchiveItem:
    nfo:fileName "archive13.zip" ;
    nfo:belongsToContainer <http://f2r.example.com/mysource/> ;
    pav:retrievedFrom <http://example.com/download/archive13.zip> .
<http://f2r.example.com/mysource/> a nfo:Filesystem .
```

Fig. 3. RDF Turtle description of a file `<data/survey.csv>` within an archive `<http://example.com/download/archive13.zip>`, using Nepomuk File Ontology [19], PAV [20] and F2R [18] identifiers.

the corresponding `.well_known/ni` URI for retrieving the archive. This library is meant to complement the Python 3 *urlparse* library, and so it is deemed out of scope for this library to do resolution of arcp based on archive or network access.

The **Research Object Bundle library**, part of Apache Taverna (incubating), is adding support for arcp URIs in its opening and creation of RO bundles, initially using the arcp UUID format as a replacement for app URIs, with planned support also for hash-based identifiers and opening RO Bundles from a `.well-known/ni` endpoint.

The **CWLProv** [24] approach for capturing provenance of executing Common Workflow Language is using arcp in its BagIt metadata `bag-info.txt` using `External-Identifier` to identify its research object:

```
External-Identifier:
  arcp://uuid,d47d3d43-4830-44f0-aa32-4cda74849c63/
```

For CWLProv the use of arcp is crucial, as it assigns global identifiers for use across resources in the RO bag, including the RO manifest itself and in W3C PROV file formats like PROV-N and N-Triples, as neither format support relative URIs.

In this approach the UUID component of the RO arcp identifier `d47d3d43-4830-44f0-aa32-4cda74849c63` also appears in the workflow provenance as the identifier of the top-level workflow run (a PROV Activity):

```
prefix id <urn:uuid:>
activity(id:d47d3d43-4830-44f0-aa32-4cda74849c63,
  2018-08-21T17:26:24.467636, -,
  [prov:type='wfprov:WorkflowRun',
  prov:label="Run of workflow/packed.cwl#main"])
```

This is showcasing how an RO that is the primary representation of a *non-information resource* (e.g. a process) can be identified using a directly derived arcp URI. While this could in theory also been achieved with an arcp UUIDv5 derived from hashing the URI "location" of the activity, that would be a confusing hack, as `urn:uuid:` references by design are not resolvable, and hence technically not URLs. UUIDv5 hashing could however be appropriate for non-information resource if they have a resolvable http/https permalink.

V. Conclusion

This article propose the arcp identifier scheme for resources within archives using formats like ZIP, tar and BagIt, and suggest arcp is useful for identifying standalone Research Objects and for processing Linked Data embedded in archives. The Internet-Draft `draft-soilandreyes-arcp` [10] is under consideration by IETF's Applications and Real-Time Area to progress towards Informational RFC status.

References

[1] J.A. Kunze, J. Littman, L. Madden, J. Scancella, C. Adams (2018): **The BagIt File Packaging Format (V1.0)**. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-kunze-bagit-16

[2] Research Data Repository Interoperability WG (2018): **Research Data Repository Interoperability WG Final Recommendations**. Research Data Alliance. https://doi.org/10.15497/RDA00025

[3] F.T. Bergmann, R. Adams, S. Moodie, J. Cooper, M. Glont, M. Golebiewski, et al.,(2014): **COMBINE archive and OMEX format: one file to share all information to reproduce a modeling project**. BMC Bioinformatics. 15 369. https://doi.org/10.1186/s12859-014-0369-z

[4] Space Physics Data Facility (2016): **CDF Internal Format Description, 3.6**. NASA / Goddard Space Flight Center. https://spdf.gsfc.nasa.gov/pub/software/cdf/doc/cdf364/cdf36ifd.pdf

[5] The HDF Group (2016): **HDF5 File Format Specification Version 3.0**. The HDF Group. https://support.hdfgroup.org/HDF5/doc/H5.format.html

[6] S. Soiland-Reyes, M. Gamble, R. Haines (2014): **Research Object Bundle 1.0**. researchobject.org Recommendation, Zenodo. https://w3id.org/bundle/2014-11-05/ https://doi.org/10.5281/zenodo.12586

[7] System Applications Working Group (2015): **The app: URL Scheme**. W3C Working Group Note 23 July 2015, World Wide Web Consortium. https://www.w3.org/TR/2015/NOTE-app-uri-20150723/

[8] M. Cáceres, K.R. Christiansen, M. Lamouri, A. Kostiainen, R. Dolin, M. Giuca (eds.) (2018): **Web App Manifest**. W3C Working Draft 04 July 2018, World Wide Web Consortium. https://www.w3.org/TR/2018/WD-appmanifest-20180704/

[9] A. Russel, J. Song, J. Archibald, M. Kruisselbrink (2017): **Service Workers 1**. W3C Working Draft 2 November 2017, World Wide Web Consortium. https://www.w3.org/TR/2017/WD-service-workers-1-20171102/

[10] S. Soiland-Reyes, M. Cáceres (2018): **The Archive and Package (arcp) URI scheme**. Internet-Draft draft-soilandreyes-arcp, Internet Engineering Task Force. https://tools.ietf.org/html/draft-soilandreyes-arcp-03

[11] M. Kerwin (2017): **The "file" URI scheme**. RFC Editor. RFC 8089 https://doi.org/10.17487/RFC8089

[12] T. Berners-Lee, R. Fielding, L. Masinter (2005): **Uniform Resource Identifier (URI): Generic Syntax**. RFC Editor. RFC 3986 https://doi.org/10.17487/rfc3986

[13] P. Leach, M. Mealling, R. Salz (2005): **A universally unique identifier (UUID) URN namespace**. RFC Editor. RFC 4122 https://doi.org/10.17487/rfc4122

[14] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, P. Hallam-Baker (2013): **Naming Things with Hashes**. RFC Editor. RFC 6920 https://doi.org/10.17487/rfc6920

[15] M. Nottingham, E. Hammer-Lahav (2010): **Defining Well-Known Uniform Resource Identifiers (URIs)**, RFC Editor. RFC 5785 https://doi.org/10.17487/rfc5785

[16] C. Lynch, S. Parastatidis, N. Jacobs, H. Van de Sompel, C. Lagoze (2007): **The OAI-ORE effort: Progress, challenges, synergies**. Proceedings of the 2007 Conference on Digital Libraries - JCDL '07. https://doi.org/10.1145/1255175.1255190

[17] N. Ferro, G. Silvello (2013): **Modeling Archives by Means of OAI-ORE**, IRCDL 2012: Digital Libraries and Archives, pp 216–227. https://doi.org/10.1007/978-3-642-35834-0_22

[18] Shaopeng He, Jianhui Li, Zhihong Shen (2013): **F2R: Publishing file systems as Linked Data**. *10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pp. 767–772. https:///doi.org/10.1109/FSKD.2013.6816297

[19] Ansgar Bernardi, Gunnar Aastrand Grimnes, Tudor Groza, Simon Scerri (2011): **The NEPOMUK Semantic Desktop**. *Context and Semantics for Knowledge Management* pp 255-273. https://doi.org/10.1007/978-3-642-19510-5_13

[20] P. Ciccarese, S. Soiland-Reyes, K. Belhajjame, A.J. Gray, C. Goble, T. Clark (2013): **PAV ontology: provenance, authoring and versioning**. *Journal of Biomedical Semantics* **4**:37. https://doi.org/10.1186/2041-1480-4-37

[21] James Pritchett, Markus Gylling (eds) (2017): **EPUB Open Container Format (OCF) 3.1**. W3C Member Submission 25 jan 2017. World Wide Web Consortium. https://www.w3.org/Submission/2017/SUBM-epub-ocf-20170125/

[22] Peter Sorotokin, Garth Conboy, Brady Duga, John Rivlin, Don Beaver, Kevin Ballard, Alastair Fettes, Daniel Weck (eds) (2017): **EPUB Canonical Fragment Identifiers 1.1**. Recommended Specification 5 January 2017. International Digital Publishing Forum. http://www.idpf.org/epub/linking/cfi/epub-cfi-20170105.html

[23] S. Soiland-Reyes (2018): **stain/arcp-py: arcp 0.2.0**. Zenodo software http://arcp.readthedocs.io/en/0.2.0/ https://doi.org/10.5281/zenodo.1165986

[24] F.Z. Khan, S. Soiland-Reyes, M.R. Crusoe, A. Lonie, R. Sinnott (2018): **CWLProv - Interoperable Retrospective Provenance capture and its challenges**. In preparation. Zenodo preprint: https://doi.org/10.5281/zenodo.1215611