PID Service – an advanced persistent identifier management service for the Semantic Web

P. Golodoniuc [a], N. J. Car [b], S. J. D. Cox [c], and R. A. Atkinson [d]

[a] *CSIRO Mineral Resources Flagship, Kensington, WA, Australia*
[b] *CSIRO Land & Water Flagship, Dutton Park, QLD, Australia*
[c] *CSIRO Land & Waters Flagship, Highett, VIC, Australia*
[d] *Metalinkage, Wollongong, NSW, Australia*

Email: pavel.golodoniuc@csiro.au

Abstract: Persistent identifiers are an integral part of the Semantic Web and Linked Data applications: they enable the stable identification of digital objects and may be used as a top-level application programming interface (API) to bind multiple representations of digital objects into a single, coherent, data model. In addition to these technical tasks, persistent identifiers and their management are of prime concern for the governance of Domain Name System (DNS)-based domains containing output from multiple parties that need to ensure identifier uniqueness as their first order of operation.

Our contribution to solutions for the technical and governance challenges posed by identity management is the PID Service – a persistent identity service – which is a web service offering advanced persistent identifier management with features not found in proxy servers and other web redirection products. The PID Service is able to store and implement large numbers of complex Uniform Resource Indicator (URI) redirection rules and handle related sets of rules according to rule hierarchies. This, combined with a web-based graphical user interface and database rule storage, allows users of the PID Service to far more easily manage large numbers of complex rules within a domain avoiding rule collision and specialised partial URI delegation. The Application Programming Interface allows programmatic access to all features of the service that, in turn, provides boundless integration possibilities with other applications and services. These possibilities include, but are not limited to, applications such as automatic data harvesting and digital entity identification.

The PID Service is being used for a number of operational Semantic Web and Linked Data applications including the 'environment' portion of the Australian Commonwealth Government's data.gov.au project operating at environment.data.gov.au. There the PID Service handles the identifiers for a range of Linked Data products including the large and complex national Australian Hydrological Geospatial Fabric. In addition to handling current products within the domain, the design of the PID Service is such that it will be able to cope with large increases in number of persistent identifiers, which is important given the rising popularity of open government data and the use of services such as environment.data.gov.au. The PID Service has also formed part of the key service infrastructure in the Spatial Identifier Reference Framework (SIRF) (Atkinson et al., 2013) – a scalable linked data infrastructure that aims to improve the supply of open, spatially enabled and linked information. SIRF provides means to reliably cross-reference identifiers for the real-world locations and encodes spatial relationships between features (i.e. containment and adjacency). This framework of spatial identifiers is used to link together information (e.g., socio-economic statistics) about locations, stored in multiple distributed systems.

In this paper we outline the motivation for the PID Service including the limitations of other proxy and redirect technologies. We provide an overview of the system design and describe both its technical functionalities and use cases. Finally, we describe the aforementioned installation of the PID Service at environment.data.gov.au and discuss how it impacts domain governance.

Keywords: *Persistent identifier, Uniform Resource Identifier (URI), redirection service, web proxy, Semantic Web*

1. INTRODUCTION

1.1. Identifiers

Identifiers for digital objects and the digital representation of non-digital objects are critical for the functioning of many information systems. Much as relational databases (Codd, 1970) use unique keys for finding data in tables, the World Wide Web uses Uniform Resource Identifiers (URIs) (Berners-Lee, 2005) to identify and resolve resources – web pages and other digital data. In order to reduce 'dangling links' – web page URIs that no longer resolve – URI design patterns have been proposed (Berners-Lee, 1998; Sauermann and Cyganiak, 2008). Organisations such as the Australian National Data Service (ANDS) provide local URI guidelines[1]. Persistent identifiers do not have to contain any metadata about the resource they identify or provide a mechanism for a user to resolve the resource however in the best case they do provide these properties which we summarise as:

- **Uniqueness** – identifiers imply the lack of ambiguity and one must provide means to identify the same information resource even if it is presented in different formats;
- **Persistency** – once minted, an identifier must remain active over the life of the information resource and beyond, even if that resource, its particular format or representation ceases to exist;
- **Resolvability** – identifier system must provide mechanisms to resolve identifiers to information resources and their associated formats and representations.

Some of these properties are exhibited by identifiers from different system, including non-digital systems.

1.2. Identifier schemes

Duerr et al. (2011) assessed and compared nine technologies and systems for assigning persistent identifiers for their applicability to earth science data (ARK, DOI, XRI, Handle, LSID, OID, PURL, URI/URN/URL, and UUID) depending on their technical, user and archival values. The case for 'pure' HTTP-based URIs for universal use was made by the W3C's Technical Architecture Group (Thompson and Orchard, 2006) and summarised in Cox (2011). It rests on three points: 1. HTTP URIs can serve as better protocol-neutral identifiers than systems such as DOI; 2. URIs are resolvable using standard Internet technologies (Domain Naming System & Internet Protocol); and 3. management regimes may be implemented through HTTP URIs without establishing a new protocol. For these reasons, our consideration of identifier management in this paper and the design of the PID Service was limited to HTTP URIs.

1.3. HTTP URI Identifier technologies

The management of HTTP-based URIs requires mappings between URIs used as persistent identifiers and 'internal' resource locators that give access to the resources in various forms. The most basic of web server function, web page resolution, exhibited by all web servers (e.g., Apache[2] with no additional functional modules implemented) follows this method by relating a Uniform Resource Locator (URL – a subset of URIs used for web pages) to a file stored on a server's disk. A higher-order HTTP URI management method is to map persistent identifier URIs to other, less persistent URIs and files. Such an 'indirection layer' allows resources' locations – given by less persistent URIs – to change while the persistent identifiers are maintained.

We present a short review of available technology for mapping-based HTTP URI management based on many usability aspects and the completeness with which they implement the HTTP 1.1 standard (IETF, 1999). Specifically we considered:

- Availability of a browser-based graphical user interface (GUI) to allow remote configuration;
- Availability of an Application Programming Interface (API);
- Database backend;
- Pattern-based URI mappings;
- One-to-one mappings for identifiers unable to be managed by a pattern;
- MIME-type content negotiation (e.g., CSV, GML, RDF/XML, etc.);
- Fine-grained MIME-type content negotiation (e.g., subtype): application/gml+xml; subtype=gml/3.1.1; schema=GeoSciML 2.0
- Content negotiation via the use of URI query string parameters;
- URI decommissioning (allowing resolution of those with resources no longer available);
- URI mapping rule inheritance (i.e. specialisations of broader rules);

[1] http://ands.org.au/guides/persistent-identifiers-working.html. Accessed 31 June 2015.
[2] http://httpd.apache.org/

- Redirects and proxying requests;
- Scalability (ability to handle many patterns and 1:1 mappings);
- Resource version handling;
- Multilingual resource representations.

The technologies/systems we review are Persistent Uniform Resource Locator (PURL), URL Shorteners, Apache mod_rewrite, and the Django Framework's URL Dispatcher. We indicate their abilities with respect to the criteria above and compared to the PID Service in Table 1 as well as providing some notes below.

Table 1. Results of a comparison of HTTP URI identifier management technologies.

Feature	PURL	URL Shorteners	Apache mod_rewrite	URL Dispatcher	PID Service
Web-based GUI	X	X			X
API	X			X	X
Database backend	X			X	X
Pattern matching			X	X	X
1:1 mapping	X	X	X	X	X
MIME Content Negotiation			X	Possible	X
Fine-graining MIME			X	Possible	X
QSA Content Negotiation				Possible	X
Redirects	X	X	X	X	X
Proxying			X	With other systems	X
Scalability	?	?		X	X
Resource versions				Possible	X
Multilingual resources			X	Possible	X
URI decommissioning		X	X	X	X
Memorable URIs	X		X	X	X
Live updates (i.e. no server restart required)	X	X		X	X

PURL

URLs in PURL describe a persistent location that redirects to the current location of a web resource or tree of resources. Many reviews of PURL have been conducted over time, such as The Library of Congress (1997), Weibel (2007) and Duerr et al. (2011). Persistent Uniform Resource Locator (PURL) technologies have existed for several decades (Lynch, 1997) and are still in active use, and they can be considered highly successful. PURL provides an exemplary web-based UI for remote management of persistent identifiers. In practice this is used more commonly than the API which is much more limited, particularly in terms of diagnostics. However, PURL does not support expression-based mapping rules, and this gap was the initial prompt for the development of a new solution.

URL Shorteners

URL Shortening provides 1:1 mappings between a short URI and a longer resource URL, for example the URI for MODSIM 2015's "Instructions for Authors" web page, http://mssanz.org.au/modsim2015/instructions.html, is now redirected to from the shorter http://bit.ly/1DUZBpC. The 'shortened' URL almost always ends in an opaque non-memorable string. These services have become very popular, especially for use with social media (Wortham, 2009), within search engine optimisation communities (SEO), and by spammers (Curtis, 2014). There are now tens of different implementations available worldwide. While URL shorteners do provide a redirection, no service provider guarantees the persistence of identifier over a long time frame (multiple years). This provides a limited capability to address the requirements.

Apache mod_rewrite

The Apache server *mod_rewrite* extension is a rule-based URL rewriting engine, based on a regular expression parser, to rewrite URLs at run time. It provides a powerful and flexible way to manipulate URLs based on a number of conditions, including patterns, server variables, HTTP headers, query string parameters, etc., see Figure 1. Through mod_rewrite rules, the user may choose to resolve the incoming URL request to an information resource locally stored on a file system, respond with an HTTP redirect, or proxy request to a third-party data service. Being based on the PCRE regular expression parser, the mod_rewrite module provides powerful pattern recognition. Its functionality also provides a fine-grained control over the actions that need to be taken to resolve an URI. A full list of its technical capabilities may be obtained from Apache online documentation[3].

[3] http://httpd.apache.org/docs/current/mod/mod_rewrite.html, accessed 24 June 2015.

```
ServerName data.gov
RewriteEngine On

# a. Simple URI pattern recognition
RewriteRule "^/rock/(.+)$"         "http://myorg.org/data/rock/$1" [L]

# b. Simple URI pattern recognition with a condition
RewriteCond "%{HTTP_USER_AGENT}"   "(iPhone|Blackberry|Android)"
RewriteRule "^/soil/(.+)$"         "http://mobile.myorg.org/data/soil/$1" [L]
```

Figure 1. Apache *mod_rewrite* configuration file for (a) an unconditioned, simple, URI pattern using regular expressions, and (b) URI pattern recognition with a condition applied to the HTTP_USER_AGENT server variable to ensure handling of requests from some mobile devices only.

While mod_rewrite provides many technological solutions for URI redirection issues, it does not address their manageability and governance aspects. Being a server-side technology, it does not intend to provide either a user-friendly GUI or an API for managing identifiers programmatically. Each change to the rule-set requires a server re-start. These limitations confine the technology to use by system administrators, which makes it difficult to use in environments with distributed governance, for example, multiple agency's use of a single DNS domain.

Many of the features, but not all, available in mod_rewrite are also available in Apache's sister web server, Tomcat, through the UrlRewriteFilter[4].

URL Dispatcher

The Django Framework's URL Dispatcher[5], while being a single technology product, exemplifies a range of framework integrated redirection systems such as those present in the popular Drupal Framework[6]. URL Dispatcher specifically aims to create 'Cool URIs' as described by Berners-Lee (1998). Being part of a web server programming framework that uses the comprehensive Python programming language, URL Dispatcher can perform any action required of it, with appropriate programming development effort. URL Dispatcher does not come with an 'out of the box' GUI nor is it intended for use by non-programmers. For these reasons, like Apache's mod_rewrite, it is unsuitable for situations where distributed governance or other issues require non-technical staff to maintain URI mappings.

2. THE PID SERVICE

The PID Service[7] was developed in an effort to address the technical and governance requirements for identifier management not addressed in a single package by any existing systems. Implementation took into account findings, requirements and observations from the technology review conducted to construct Table 1.

2.1. Functionality

The PID Service intercepts HTTP requests and attempts to match all parts of them – the URI, any query string arguments and all HTTP headers – to patterns and other logic stored in a persistent data store. The service then performs a set of user-defined actions, such as redirects, proxying, and also HTTP header manipulation, delegating resolution to another service, etc. (Figure 2). When redirection is chosen, any of the HTTP standard status codes may be set. It features extensible architecture for future improvements and supports multiple control interfaces and a web-based graphical user interface (GUI), see Figure 3, for non-programmatic management of URIs as well as for automated management of URIs via an API.

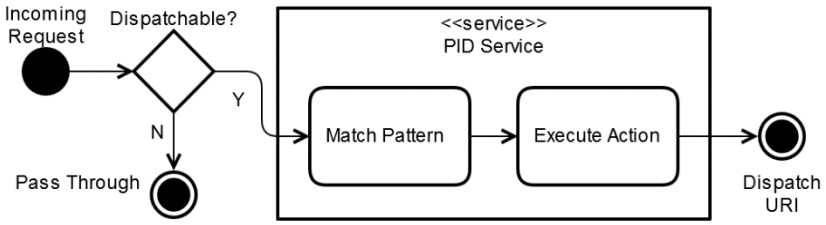

Figure 2. PID Service core principle activity diagram.

[4] http://tuckey.org/urlrewrite/, accessed 31 June 2015.
[5] https://docs.djangoproject.com/en/1.4/topics/http/urls/, accessed 31 June 2015.
[6] http://drupal.org for the framework, https://www.drupal.org/project/redirect for the common redirect module.
[7] https://www.seegrid.csiro.au/wiki/Siss/PIDService, accessed 14 September 2015.

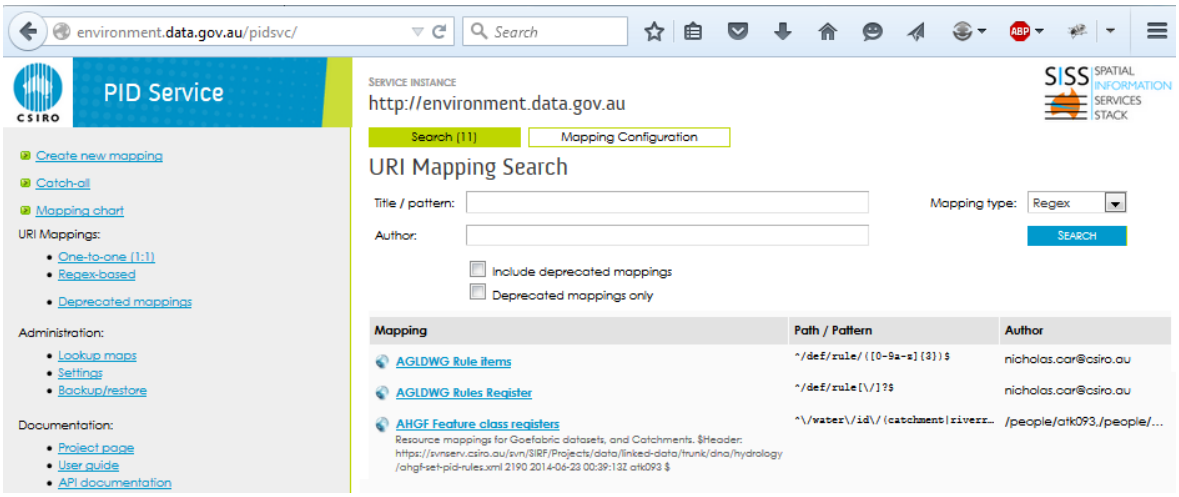

Figure 3. PID Service web-based Graphical User Interface (GUI) showing patterns in place for URI matching within the environment.data.gov.au subdomain.

The PID Service was designed to help manage persistent identifiers for the Australian earth science community. This required fine-grained content negotiation; mapping rule inheritance and the ability to handle exceptions and/or specialisations within collections of identifiers; scalability – handling large collections of identifiers with millions of individual 1:1 mapped identifiers; and handling of multiple resources representations, including language variants.

One of the core design principles of the PID Service was to ensure that identifiers remained persistent. This was realised by ensuring that once a URI for a resource or a collection of resources is created it cannot be deleted from the system or re-used by another resource although it may be suppressed, if a resource ceases its existence. The service keeps track of all modifications performed on an URI mapping, including suppression of rules – placing them into a, so-called, "tombstoned" state. Tombstoned rules are ignored by the dispatcher, but are still discoverable by users using the GUI. Tombstoned rules may be reinstated and brought back to life.

2.2. Rule inheritance and prioritisation

The PID Service implements a straightforward prioritisation system for resolution of identifiers: an exact match of a whole URI (a 1:1 mapping) takes precedence over pattern-based URI mapping. This gives a significant performance boost when handling large numbers of identifiers as it allows the system to use high-performance database indexing, potentially on a dedicated database server, rather than application code to find mappings. If an exact match is not found, the system will retrieve all pattern-based mapping rules and iterate through them to find a match following the principles of inheritance described further.

Rule inheritance is based on a tree-like hierarchy of mapping rules, where the topmost rule is “Catch all”. All mappings not explicitly defined as being a subset of a more generic rule will automatically inherit from the “Catch all” rule. The inheritance mechanism is built on a single inheritance principle, where any particular rule may be a subset of one and only one parent rule. The hierarchical structure allows visual representation of the mapping dependency tree for better understanding of rules from a user perspective. The tree is organised in such a way that all rules ultimately inherit from the “Catch all” mapping rule, which is also being used when no rules match the incoming request. The inheritance tree is searched for a matching pattern starting from the deepest level up until it matches the incoming request or reaches “Catch all” mapping at the top of the tree.

Rule inheritance provides mechanisms to control the order in which pattern-based mapping rules are asserted by the system. Due to the nature of regular expressions, it is possible to have multiple pattern-based mappings matching the same URI and residing at the same level in the tree hierarchy. This forms an exception, where the order in which URI mapping rules are processed is undefined. This, however, should not be an issue as there should be no overlap between different regular expressions at the same level within any particular logical branch. The following rules apply within the rule inheritance principle:

1. By default, all rules are inheriting from the top level "Catch all" rule unless a specific parent is set by the user. No orphaned mapping rules are allowed;
2. One-to-one mapping rules may only inherit from pattern based mapping rules, assuming the URI matches the regular expression of the ancestor;
3. Pattern-based mapping rules only inherit from other pattern-based rule with the following conditions:
   a. The validity of inheritance is not checked at the time of mapping rule creation due to complexities associated with regular expressions;
   b. When the condition search logic progresses up in the hierarchy to the parent pattern-based mapping, the URI must match the regular expression of the parent rule. If it does not, it then falls back to the top level "Catch all" mapping rule.
4. Single inheritance principle means that any one rule may only inherit from one parent only.

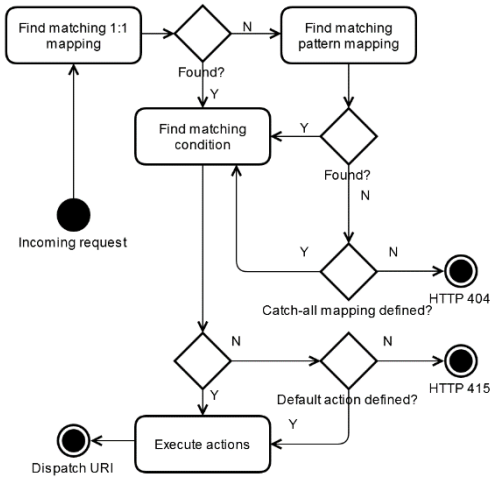

Figure 4. URI dispatching workflow.

Figure 4 provides a high level perspective on the workflow used for dispatching HTTP URIs. Once a matching mapping rule is found, either as an exact one-to-one mapping or a matching pattern based mapping rule, it will proceed to selection of a matching condition. At this stage the PID Service will iterate through ordered list of conditions defined for a matching mapping rule and, if none is found, it will proceed to the parent mapping rule's set of conditions. It will keep going through parents' conditions until either one of the conditions matched the incoming request or it reaches the "Catch all" mapping rule. If none of the conditions match the request, the PID Service will trigger the first defined default action starting from the initial matching mapping rule and progressing through the chain of ancestors up to the "Catch all" rule.

3. PID SERVICE IMPLEMENTATION EXAMPLE

The Australian Government Linked Data Working Group (AGLDWG)[8] uses an instance of the PID Service to manage URIs within the environment.data.gov.au subdomain. This subdomain acts as a demonstrator for the governance of a domain delivering URIs for multiple Semantic Web products and systems that are related only by their URIs and are otherwise heterogeneous regarding ownership, technology, URI granularity or the types of patterns implemented. A key product is the Australian Hydrological Geospatial Fabric (Geofabric)[9] which implements several tens of 1:1 URIs for Geofabric product Web Service endpoints and multiple patterns for many thousands of geospatial features, which resolve to Web Service requests. An example pattern request is given in Figure 5. Products of similar types are maintained within registers using the PID Service's hierarchical rule inheritance, for example several ontologies are hosted within the `def/` register with both hash and slash URIs for ontology items mapped via next-level rules, for instance the *Observable Properties Ontology's*[10] (Cox

```
[1] ^\/water\/id\/(catchment|riverregion|drainagedivision|stream|waterbody)\/([0-9]+)$ → [2]

[2] http://sirf.csiro.au/LdaIdService/resource?uri=${RAW:${ENV:FULL_REQUEST_URI}}&_format=html

[3] _view=(SimpleFeatures|ahgf:AHGFContractedCatchment)&_format=(csv|kml) → [4]

[4] http://geofabric.bom.gov.au/simplefeatures/ows?request=GetFeature&service=WFS&version=1.1.
0&typeName=ahgf_hrc:AHGFContractedCatchment&Filter=<Filter><PropertyIsEqualTo><PropertyName>ah
gf_hrc:ConCatID</PropertyName><Literal>$2</Literal></PropertyIsEqualTo></Filter>&outputFormat=
${C:_format}

[5] _view=listaltids&_format? →[6] http://sirf.csiro.au/sissvoc/lu/listall?uri=
                        ${RAW:${ENV:FULL_REQUEST_URI}}&_format=${ISNULL: ${C:_format:0}: html}
```

Figure 5. Geofabric patterns implemented in at environment.data.gov.au. [1] is the base URI pattern matching URIs such as http://environment.data.gov.au/water/id/catchment/104792 – a particular catchment. [1] redirects (HTTP 303) to a CSIRO Web Service [2]. [3] is an alternate *view* of a resource matched by [1] specified by the query string arguments which then redirects (HTTP 303) to a Bureau of Meteorology OGC Web Feature Service query, given in [4]. [5] specifies another *view* redirecting to a second CSIRO Web Service with response formats chosen by a user-specified `format` query string argument.

[8] See the group's information web page at http://linked.data.gov.au.
[9] http://www.bom.gov.au/water/geofabric/
[10] http://environment.data.gov.au/def/op

et al., 2014) definition of a 'substance' is given at http://environment.data.gov.au/def/op#Substance. All the products and their corresponding top-level URI patterns are listed at http://environment.data.gov.au/.

4. FUTURE WORK

The basic functionality has been implemented in a number of live deployments, but the capabilities around fine grained content negotiation require further tuning as well as mechanisms for the delivery of different resource representations. Versioning concepts of persistent identifiers are largely untested. The existing identifier systems resort to using different identifiers for different versions of the same information entity, which inherently causes identifiers to lose critical provenance information. Various aspects of identifiers versioning are yet to be investigated and tested.

5. CONCLUSIONS

The PID Service provides a useful component to assist in the deployment of linked data architectures. It offers a superset of the functionality of other persistent identifiers technologies, but adds significant additional capability, particularly concerning pattern matching within URIs, and all aspects of HTTP header requests. It is scalable and allows non-technical use through a GUI, as well as programmatic control using an API, as exemplified by `environment.data.gov.au`.

REFERENCES

Atkinson R., Box P. and Kostanski L., 2013, Spatial Identifier Reference Framework (SIRF), 26th International Cartographic Conference, Dresden, Germany, August 2013. http://www.icc2013.org/_contxt/_medien/_upload/_proceeding/417_proceeding.pdf. Accessed 31/07/2015.

Berners-Lee T., 1998, Cool URIs don't change. Design Issues. http://www.w3.org/Provider/Style/URI.

Codd E. F., 1970, A relational model of data for large shared data banks. Communications of the ACM 13 (6): 377. doi:10.1145/362384.362685.

Cox S. J. D., Simons B. A., and Yu J., 2014, A harmonized vocabulary for water quality, International Conference on Hydroinformatics. http://academicworks.cuny.edu/cc_conf_hic/179. Accessed 14/09/2015.

Cox S. J. D., 2011, Identifiers for Water Features: Requirements and Current Best Practices. In Water for a Healthy Country Flagship Report Series ISSN: 1835-095X.

Curtis S., 2014, Twitter's t.co URL shortener used to spread spam, The Telegraph. Telegraph Media Group. http://www.telegraph.co.uk/technology/internet-security/11020760/Twitters-t.co-URL-shortener-used-to-spread-spam.html, Accessed 24/06/2015.

Duerr R. E., Downs R. R., Tilmes C., Barkstrom B., Lenhardt W. C., Glassy J., Bermudez L. E. and Slaughter P., 2011, On the utility of identification schemes for digital earth science data: an assessment and recommendations. Earth Science Informatics, 4:139–160, doi:10.1007/s12145-011-0083-6.

Internet Society, The (IETF), 1999, Hypertext Transfer Protocol – HTTP/1.1. IETF RFC document online. http://www.w3.org/Protocols/rfc2616/rfc2616.html. Accessed 31/06/2015.

Library of Congress, 1997, The relationship between URNs, Handles, and PURLs. http://memory.loc.gov/ammem/award/docs/PURL-handle.html. Accessed 24/06/2015.

Lynch C., 1997, Identifiers and their role in networked information applications. http://www.cni.org/wp-content/uploads/1997/10/identifier.pdf. Accessed 24/06/2015.

Sauermann L. and Cyganiak R., 2008, Cool URIs for the Semantic Web. W3C Interest Group Note 03 December 2008. http://www.w3.org/TR/cooluris/. Accessed 31/06/2015.

Thompson H.S. and Orchard D. (Eds.), 2006, URNs, Namespaces and Registries. W3C Technical Architecture Group Finding. http://www.w3.org/2001/tag/doc/URNsAndRegistries-50. Accessed 31/06/2015.

Weibel S., 2007, PURLy gates and gift horses. Six Apart, Ltd., TypePad. http://weibel-lines.typepad.com/weibelines/2007/11/purly-gates-and.html. Accessed 24/06/2015.

Willet P., 2010, NOID: Nice Opaque Identifier (Minter and Name Resolver), https://confluence.ucop.edu/display/Curation/NOID, Accessed 28/06/2014.

Wortham J., 2009, Goo.gl Challenges Bit.ly as King of the Short, New York Times. http://bits.blogs.nytimes.com/2009/12/14/googl-challenges-bitly-as-king-of-the-short/, Accessed 9/6/2015.