20th International Congress on Modelling and Simulation, Adelaide, Australia, 1–6 December 2013
www.mssanz.org.au/modsim2013

A method and example system for managing provenance information in a heterogeneous process environment – a provenance architecture containing the Provenance Management System (PROMS)

Nicholas J. Car[a]

[a] *CSIRO Land & Water, Environmental Information Systems*
Email: nicholas.car@csiro.au

Abstract: For large distributed environmental information systems built for projects of national significance, such as eReefs and Bioregional Assessments, it is crucial they be able to capture information about their data processing in order to be able to trace the lineage of their data products. Policy and decision makers may need to know great detail about the processes that created data products in order to trust them as they influence crucial and high profile political decisions. Scientists may even need to be able to recreate modelling or data manipulation processes long after their original implementation in order to verify results.

There are a series of tools that help in this task: there is a W3C Recommendation for a 2nd generation provenance data model and Semantic Web ontology that has been developed by an international team of provenance researchers, known as PROV-O (Moreau and Missier, 2013). It is intended that it be used to represent the provenance of generic processes and can therefore be used as a common information format for projects that contain heterogeneous processes. Researchers in the CSIRO have developed a Persistent ID Service (PID) that helps to manage the identity of 'things' (information resources and representations of real world features) (Golodoniuc, 2013) for which, when those things are generated by human processes, provenance can be recorded. Provenance information can be very complex and different for every single process recorded, thus making storage difficult. Even though there are mechanisms (triplestores) built to store data of the PROV-O type, they are not massively scalable so we have used a recently popular, schema-less, databases that is able to store large collections of data 'documents' without forcing structural constraints on them. Additionally, the use of Linked Data through a variant of the Linked Data API (Epimorphics, 2013), also by CSIRO members, can be used to provide access to different forms, or views, of the 'things' stored. Putting these four developments together allows us to represent, manage, store and provide access to provenance data in novel ways. Implementing this is the focus of this paper.

A *provenance architecture* using the PROV-O ontology, PID Services, variant Linked Data APIs and several support services has been tested with several automated workflows, notably the Bureau of Meteorology's Australian Hydrological Geospatial Fabric's Contracted Catchments production workflow. We describe the development of this architecture and detail the componentry it uses – in particular the new Provenance Management System PROMS.

The methodology and architecture described here, more than the specific tools detailed, are this paper's contribution to large, multi-part, information systems' provenance handling so we present this information in order to demonstrate and approach, not to evangelise the use of a specific tool.

Keywords: *Provenance, data management, cultural change, semantic web, metadata*

1. INTRODUCTION

1.1. Data management in large science projects

The Murray Darling Basin Sustainable Yields project (MDBSY) was a large, multidisciplinary project that informed national policy regarding the Murray Darling Basin's water resources. Due to this high profile, a data management team was included in the project's staff to ensure adequate data and metadata management procedures (DM) were implemented to allow in-depth inspection of the project's processes and possibly, the testing of them by reimplementation. The project's DM foci were to ensure that simulation model data (inputs and outputs) were stored in a systematic way on enterprise-level storage systems and that all datasets had metadata entries placed in a centralised catalogue. This allowed dataset audit trails to be constructed by linking metadata entries upwards to their parent processes' records. (Hartcher & Lemon, 2009).

A similar approach is scheduled to be used for the Bioregional Assessments project (BA) (SEWPaC, 2012). This project is similar to the MDBSY in terms of complexity, scale and likely scrutiny. Unlike the MDBSY, the BA project will be supported by an *information platform* that aims to deliver the BA project's datasets through standardised web services and to standardise the metadata used for them. The aim is to allow consistent and transparent access to quality-assured BA project information. (Simons *et. al.*, 2012).

Even when DM is implemented within a project, it is difficult to ensure that team participants actually enter metadata of a sufficient quality to allow valuable audit trails to be created due to the extra effort required of staff that don't see DM as a core part of their role (Hartcher & Lemon, 2009). One partial solution to this is to implement automated project procedures, where possible, that handle metadata reporting thus removing the need for project staff to deliberately do so. While this is not applicable to many situations in projects such as the MDBSY & BA and initial cost and effort are required where it is, the approach is nonetheless attractive, given the longer term effort savings and enhances metadata harvesting.

If automation can be applied to project processes, several issues need to be overcome in order to facilitate in-depth inspection and reimplementation of processes. For the first, the issues surround the representation, management, storage and access to the audit trail information of processes' datasets' and for the second, effective archiving of the input, output and configuration data of automated process' is important.

This paper provides background to the provenance representation which deals with datasets' and process' audit trail information and presents the PROMS system as a component in a provenance architecture that attempts to allow process in-depth inspection and reimplementation. We draw on our implementation of provenance data management systems for the Bureau of Meteorology's Australian Hydrological Geospatial Fabric[1]'s Contracted Catchments generation process and from other systems currently under test utilising PROMS and provenance architecture.

2. PROVENANCE AND 'PROV-O'

The Oxford English Dictionary definition of 'provenance' is "a record of ownership of a work of art or an antique, used as a guide to authenticity or quality"[2]. When applied to computer data, 'provenance' carries the same notions relating to the guarantee of authenticity or quality. Data provenance requires knowledge of the data's generation processes, the inputs to those processes and details of the configuration of systems used in generation; anything that has affected the current form of the data.

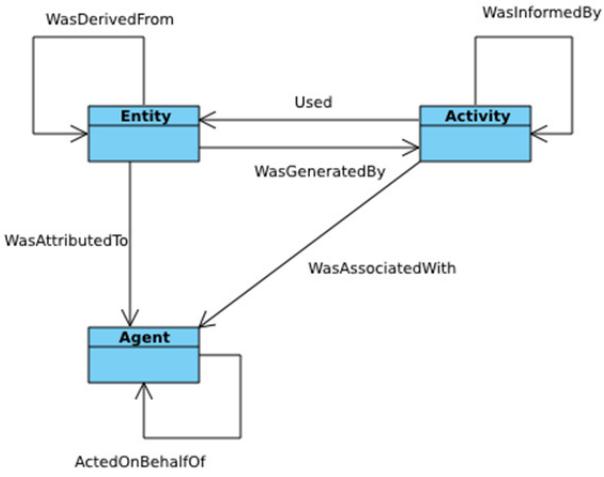

Figure 1 from Moreau & Missier (2013) UML diagram of PROV's Data Model showing 3 classes ('Entity' 'Activity' and 'Agent') and their relationships, Agents run activities which both use and produce Entities.

To record data provenance information, several mark-up languages have been developed over the past decade with the Open Provenance Model (Moreau *et. al.*, 2010). Proof Mark-up Language (Pinheiro da Silva, 2004) and its derivative World Wide Web Consortium standard PROV-O (Moreau & Missier, 2013) among them. PROV-O aims to "enable the inter-operable interchange of provenance information in heterogeneous environments such as the Web" (*ibid.*) so is suited

[1] http://www.bom.gov.au/water/geofabric
[2] http://oxforddictionaries.com/definition/english/provenance?q=provenance

to projects where provenance data from heterogeneous processes needs representation. PROV-O has a very simple data model show in Figure 1. that is deliberately generic and thus flexible.

If all the entities (datasets and configurations) used by a process and all their sub-processes could be recorded in PROV-O, very high levels of in-depth inspection of that process could take place. Typically PROV-O records (*reports*) do not store enough information about processes to allow their reimplementation. For example, a report about the use of a model, Model X, may tell you what input data was used by it, what its external settings were which/what machine it was run on, its execution time and results but this won't give you a copy of the input data itself or Model X itself which will be needed for an identical rerun of the process.

3. EXTENSIONS TO PROV-O FOR REIMPLEMENTATION CAPABILITY

3.1. PROV-O ontology properties

Holding utility for original content access in Entity and Activity class instances in PROV-O reports is difficult. Consider the example PROV-O report shown in Figure 2: standard Entity properties include *alternateOf*, *specializationOf*, *atLocation*, *value*, *wasAttributedTo*, *atTime* and many others. These properties

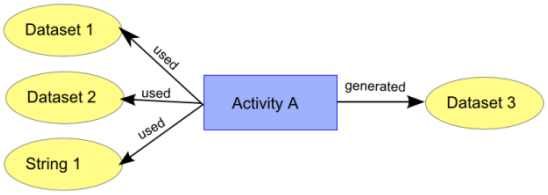

Figure 2: A basic, generic PROV graph showing an Activity using and producing Entities (the Datasets and a String).

tell you

things about the dataset but do not give you access to non-RDF representations of it (such as the original binary file formats) unless the entirety of the Entity is stored as a string in *value*. Likewise, properties of Activity include *startTime, endTime*, *wasAssocatedWith* and *wasStartedBy* which give you metadata but not the substance of the Activity. For Activity, unlike Entity, there is no *value* for attribute that could be used if relevant.

```
:activity_a
   a prov:Activity;
   dcterms:title        "Activity A"^^xsd:string;
   proms:machine        "butterfree-bu.nexus.csiro.au"^^xsd:string;
   prov:startedAtTime   "2013-06-13T12:30:01+11:00"^^xsd:dateTime;
   prov:wasStartedBy    :org_agent;
   prov:endedAtTime     "2013-06-13T12:31:01+11:00"^^xsd:dateTime;
   prov:wasEndedBy      :org_agent;
   prov:used            <http://dids.it.csiro.au/data/51b1665836cff90b9d00004c>;
   prov:used            <http://dids.it.csiro.au/data/51b1435a78cff90b9d000055>;
   prov:used            :string_1;
   prov:generated       <http://dids.it.csiro.au/data/51b1790836cff90b9d00001b>; .

<http://dids.it.csiro.au/data/51b1665836cff90b9d00004c>
   a                    prov:Entity, void:Dataset;
   dcterms:title        "Dataset 1"^^xsd:string;
   proms:usedAtTime     "2013-06-13T12:31:01+11:00"^^xsd:dateTime; .

:string_1
   a                    prov:Entity;
   dcterms:title        "String 1"^^xsd:string;
   prov:value           "The quick brown fox"^^xsd:string;
   proms:usedAtTime     "2013-06-13T12:31:02+11:00"^^xsd:dateTime; .

<http://dids.it.csiro.au/data/51b1790836cff90b9d00001b>
   a                    prov:Entity, void:Dataset;
   dcterms:title        "Dataset 3"^^xsd:string;
   prov:generatedAtTime "2013-06-13T12:31:05+11:00"^^xsd:dateTime; .
```

Figure 3: Partial turtle serialisation of Figure 2's provenance graph. Namespace declarations and Dataset 2 are not shown.

Such properties may be sufficient for the in-depth inspection of Activities and Entities but cannot allow reimplementation without further manual discovery of copies of the original Entity and Activity items.

3.2. PROMS ontology properties as usage approach

Figure 3 shows a Turtle partial serialisation of Figure 2. In the case of the input Entity *string_1*, everything we need to know about the Entity's value for recreation is contained within the PROV-O *value* attribute and it's type declaration (*xsd:string*). For the input entities *dataset_1*, *dataset_2* (not shown in Figure 3) the attribute *usedAtTime* has been added using the *proms* namespace to indicate when an externally hosted datasets was actually used. The use of the *void:Dataset* class as an additional *rdf:type* for certain activities indicates not that they are RDF datasets but that they are a *prov:Entity* with additional information (representation) elsewhere, given by their URI which must be an absolute URI, not relative to the report's URI. Thus for an information resource Entity, its non-RDF form may be dereferenced from its URI and thus a further step towards recreation has been made. Where Activities are computer software processes, a similar mechanism can be used to refer to a representation of its executable code.

The result of these additions is that an RDF interpreter could collect not only the structure of a process which has been reported from its expression as Entities and Activities but also copies of the files used, executed and

Table 1: Example URIs and URI patterns for a single dataset for use in a provenance architecture

produced by the process. Instantiating the non-RDF Entity URI endpoints meaningful data requires work outside of provenance generation *per se* but must still be handled within a provenance architecture.

3.3. Persistent Identities

In order to create URIs for use by Entities tagged with *void:Dataset* that persist for very long periods of time, IT system infrastructure changes must be handled. If not, when a person or a machine dereferences an such an Entity's URI, it could be broken, thus preventing dereferencing and thus retrieval.

Table 2: Example URIs and URI patterns for a single dataset for use in a provenance architecture

URI Type	URI
Generic URI pattern for a data item	http://{domain}/{data_registery}[/{sub_registry}]+/{datum_ID}
externalURI as used in a PROV-O report document	http://doc.csiro.au/dataitems/workflow-y/123456
Underlying URI used by a particular dataset storage mechanism	http://server-x.it.csiro.au/data/123456

Several systems have been built to provide persistent URIs with the most well-known probably being PURL[3] and DOI[4] however none of these systems have been implemented by the authors' provenance architecture for reasons given in Cox (2011). These reasons can be paraphrased by saying none of the other systems are as widely supported by internet software as plain URIs and the governance regimes that such systems offer contain no benefits beyond those conferred by a self-managed URI-based system if well maintained however they may pose significant constraints such as cost or bottlenecking resolver services.

The system chosen by the authors is the Persistent Identifier Service (PID Service) (Golodoniuc, 2013) that adds simple governance to Apache's *mod_rewrite* by handling changes – through a dedicated interface and tracking them – as well as scalability – through database lookups – for large mapping lists. For the purposes of provenance Entity URIs, it allows a URI to be created and then mapped to an underlying URL and for that mapping to be updated if and when the underlying URL is required to change, thus preserving the public URI and thus the link integrity of provenance reports that contain it.

4. PROMS DESIGN

4.1. History of PROMS

PROMS was designed as a successor to the Central Provenance Store (CPS) (Kloppers, 2012). It was built by the CSIRO's Sustainable Water Information Models (SWIM)[5] and the Australian Water Resources Assessments (AWRA) (Stenson *et. al.*, 2012) projects to capture provenance information from heterogeneous, automated, workflow systems. The system relied on harvester software components that collected provenance information from automated workflows and other software systems and then stored the results in a single database. It used the Proof Mark-up Language (PML) as its provenance interlingua and provided a RESTful API and desktop tools visualisation tools for trace access.

The CPS was tested using dummy complex geoprocessing workflows for the Australian Hydrological Geospatial Framework project[6] and real model chaining workflows for AWRA however it was never put into operation. The project suffered from complex interactions between system components and an inability for simple provenance data inspection. It also provided a single methodology which one had to use in order to report provenance information; one had to allow a harvester that reported to the CPS to access a system's own provenance repository at fixed intervals in order for the CPS to record provenance.

4.2. PROMS developments from the CPS

PROMS retains the idea of using a single, central, database to store representations of provenance from multiple processes. It also retains the idea of using a single provenance representation interlingua, although a change has been made from PML to PROV-O to further compliance with international provenance work. Further, it uses the CPS method of storing provenance reports as documents in a document database.

[3] http://purl.org/docs/index.html
[4] http://www.doi.org/
[5] http://www.csiro.au/Organisation-Structure/Flagships/Water-for-a-Healthy-Country-Flagship/WIRADA_WFHC_ResearchProfile/SWIM.aspx
[6] http://www.csiro.au/~/media/CSIROau/Flagships/Water%20for%20a%20Healthy%20 Country%20Flagship/WIRADA/WIRADA_GeofabricFactsheet_WfHC.ashx

PROMS dispenses with the idea that a CPS or equivalent need run harvesters with access to processes' own provenance repositories in order to collect provenance data, instead it requires that processes wishing to have their provenance reported must implement exporters that deliver provenance information to the PROMS provenance store when a process is run. This obviates the need for processes to store their own provenance data in a local repository and thus greatly expands the number of systems able to report provenance.

Through the use of a PID Service and extra provenance report elements (see Section 3.3 for an explanation) PROMS can be used to access the input and executable files of software processes stored in a file store, something not possible in the CPS.

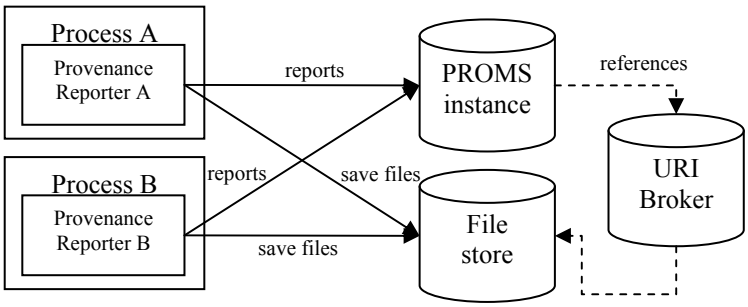

Figure 4: Overall conceptual architecture allowing two heterogeneous process to report provenance reports and store files for reimplementation

4.3. PROMS within a broader Provenance Architecture

Communication standards are used at every component interface within the provenance architecture given in Figure 4 so that a system wishing to store provenance information and files from multiple processes may use any file store, PID Service or even provenance storage system in place of PROMS as long as certain performance requirements are met. The requirements for the file store component within this PROMS provenance architecture are only that it provide a stable URI which can be dereferenced to an instance of the dataset identical to that used in the original workflow. It must do this for each dataset that it stores that is to be used by the architecture. Examples of systems that can fulfil this role are generic document management systems such as Microsoft's SharePoint[7] (most appropriate for documents), version control systems such as Subversion[8] and Git[9] (most appropriate for computer code) and DIDS, the Data ID System[10] developed by the author (most appropriate for datasets with multiple representations).

For the URI Broker component, any URI lookup system may be used which provides governance over pattern and 1:1 URI translation. If a large number of datasets are to be used, good scaling will be required. Apache[11] with the *mod_redirect* module could be used or better still, a content management system, such as Drupal[12] which provides database/application layer URI aliasing, could be. The PID Service was chosen for this work due to its controlled governance and scalability as described in Section 3.3.

In place of the PROMS compo nent itself, any system that can store and manage PROV-O reports as outlined in Section 5 could be used. The author is not aware of the existence of such systems however, based on the experience of creating PROMS, it would not be hard to make an equivalent system for a particular purpose.

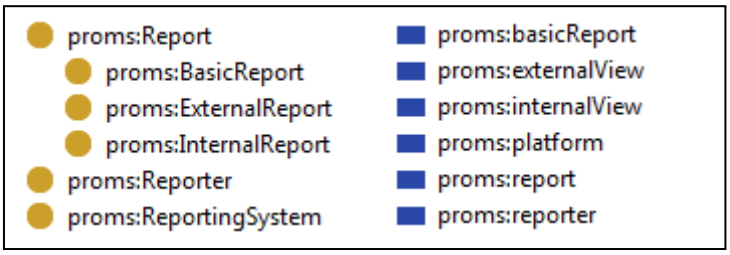

Figure 5: *proms* ontology Classes (golden circles) and properties (blue rectangles).

4.4. PROMS design

The PROMS system consists of an application layer that exposes a RESTful API to users allowing them to both deliver provenance data to it and query the data it has stored. Python is used for the application layer's code with the Python Flask framework[13] used to support HTTP interactions. PROMS may be run as a request handling application directly on a server however the author has found it more useful to install PROMS behind another web server instance, namely nginx[14]. nginx is to be preferred over servers such as Apache due to its non-blocking nature which allows Python's Flask, and thus PROMS, to operate in a similar fashion.

[7] http://office.microsoft.com/en-au/sharepoint-server-help/what-is-sharepoint-HA010378184.aspx
[8] http://subversion.tigris.org
[9] http://git-scm.com
[10] https://wiki.csiro.au/display/proms/Data+ID+System
[11] http://www.apache.org
[12] https://drupal.org
[13] http://flask.pocoo.org
[14] http://nginx.org

4.5. The PROMS Ontology

Communicating with PROMS requires the use of the *proms* ontology[15] to populate provenance reports. It relies on PROV-O ontology classes and defines a *Reporter, ReportingSystem* and *Report* Class that are used to represent provenance contributors (see Figure 5), the provenance reporting systems and the reports that contain individual traces. The *proms* ontology has three subclasses for *Report*: *BasicReport, ExternalReport* and *InternalReport.* The first allows reports containing only a title, a start time & end time and a free text description field. This type of *Report* allows for minimal effort in reporting to PROMS but, as a result, little useful provenance information is captured beyond event occurrence. The *ExternalReport* subclass requires processes reporting provenance be regarded as a single *prov:Activity* with input and output *prov:Entities* (*prov:used* & *prov:generated* properties). A process' inner workings are not captured in a *ExternalReport* instance which is used to chain multiple processes together through linking outputs of one to inputs of another. The *InternalReport* subclass requires as much detail about a processes' constituent sub-processes and internal datasets as possible be reported using the *prov:Activity* and *prov:Entity* Classes. An *ExternalReport* instance is inferable from a *InternalReport* instance.

5. PROMS OPERATION

proms:Reporters need to ensure a *proms*:*Report* is generated for each run of a *proms:ReportingSystem*. *Reporter prov:Elements* (input and output files) must firstly be registered with a file store able to generate URIs for them. *ReportingSystems* need to be registered by a PROMS instance in order to have an end-point URL for them to send provenance documents to via HTTP POSTs. Any project's system or process which can deliver POST messages, extract the data necessary to form valid PROV-O documents[16], represent it in one of several formats (XML RDF or Turtle) and communicate via HTTP may act as a Reporting System.

5.1. Storing PROV-O reports

The Mongo database[17] is used as the persistence layer in PROMS, as it was in the CPS. Unlike the CPS, PROMS uses JSON-LD[18] to store the PROV-O portions of reports and a conversion between a PROV-O exchange format (turtle, XML RDF etc.) and JSON-LD is carried out by the application layer (the *rdflib*[19] and *librdf-jsonld*[20] Python modules) when information is moved in or out of PROMS. Future versions of PROMS will utilise the same Python modules to generate RDF triples from multiple provenance reports on the fly. Mongo was chosen as it supports massive scalability. It is possible that current triplestores would not be able to efficiently scale to the likely future requirements of PROMS instances with multiple processes posting very large reports to them but this is yet to be tested. Relational databases, although able to scale massively, would not efficiently store report documents as their contents do not conform to a schema.

5.2. Accessing PROMS reports

Reports stored in PROMS can be accessed via a RESTful API. At a minimum, Turtle, RDF XML and HTML formats are available for every point in the API via Linked Data principles meaning a machine reading of a PROMS data store is straight forward. SPARQL endpoints for RDF querying are available for each *ReportingSystem* and will soon (late 2013) be able to query across multiple *ReportingSystems.* Figure 6 shows a screenshot from a PROMS instance giving the HTML view of a report at the *External* level.

6. CONCLUSION

A provenance architecture including PROMS has enabled the in-depth inspection and reimplementation of automated processes that are able to incorporate reporter elements. Its various elements can be implemented in a number of ways as long as the overall approach is maintained. This provides great flexibility to system architects and should allow for implementations in many scientific projects.

The specifics of the a provenance architecture using PROMS are not all able to be related in a paper of this length however the architecture's method of linking provenance representation and information resources,

[15] http://promsns.org/ns#
[16] See Moreau & Missier, 2013 for the PROV data model rules that PROV-O uses.
[17] http://www.mongodb.org
[18] http://json-ld.org
[19] https://github.com/RDFLib
[20] https://github.com/RDFLib/rdflib-jsonld

the use of a Linked Data API & novel RDF data storage by PROMS and the flexibility of the approach regarding the capture of provenance data will all help manage provenance from heterogeneous processes.

Imminent use of this system for several large projects will test assumptions made here and also generate a large volume of provenance data in the RDF format which will provide provenance researchers with an interesting dataset on which to perform inferencing and other RDF reasoning activities.

Figure 6: An HTML format view of an instance of a *ExternalReport* as generated by a PROMS page view. Turtle and RDF XML formats of this view are also. The *dcterms:title* of the *prov:Activity* is 'test', while the Activity, being also a *void:Dataset,* has a URL given under the *title* in white text. The single input (*prov:used*) and the top-most output (*prov:generated*) is presented as a hyperlink with *title* set to the visible text and the hyperlink's underlying href element set to the *prov:Entity's* URL. The 'ID' and 'Full filepath' outputs do not present as hyperlinks since they can be entirely represented as title/value pairs. For the ID output, the title is ID and the value is '5201d4be80e9fc2afb23b066'.

REFERENCES

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

Epimorphics Pty Ltd, (2013) ELDA – Epimorphics Linked Data API. Downloadable open source code base, online at https://code.google.com/p/elda/. Last accessed, 23/09/2013.

Golodoniuc, P. (2013) Persistent Identifier Service (PID Service). Web page retrieved 12 Aug 2013 from https://www.seegrid.csiro.au/wiki/Siss/PIDService.

Hartcher, M.G. and Lemon, D. (2009), Developing data audit trails for the CSIRO Sustainable Yields projects. In MODSIM09 conference. MSSANZ, July 2009, pp. 2377-2383. ISBN: 978-0-9758400-7-8. http://www.mssanz.org.au/modsim09/J4/hartcher.pdf

Kloppers C, Liu Q, Taylor K, Walker G. 2012. WDTS provenance. Internal Project Report. CSIRO Water for a Healthy Country Flagship. 30 pp.

Lee, B. and Box, P. (2012), Generation of data product provenance information from HWB. CSIRO Water for a Healthy Country Flagship, Australia.

Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, E., Simmhan, Y., Stephan, E. and Van den Bussche, J. The open provenance model core specification (v1.1). Fut. Gen. Com. Sys., July 2010. (doi: 10.1016/j.future.2010.07.005).

Moreau, L. and Missier, P. eds. (2013) PROV-DM: The PROV Data Model. W3C Recommendation. Web page retrieved on 30 April 2013 from http://www.w3.org/TR/2013/REC-prov-dm-20130430/.

Pinheiro da Silva, P.; McGuinness, D. L. & Fikes, R. E. A Proof Markup Language for Semantic Web Services Knowledge Systems Laboratory, Stanford University Technical Report, 2004.

DSEWPaC (2012) The framework for bioregional assessments of coal seam gas and coal mining development. A report of the Independent Expert Scientific Committee on Coal Seam Gas and Coal Mining through the Department of Sustainability, Environment, Water, Population and Communities.

Simons, B.A., Lemon, D., Cox, S.D.J. and Woodcock, R., (2012) Information Requirements for Bioregional Assessments. Water for a Healthy Country Flagship Report series ISSN: 1835-095X

Stenson, M.P., Fitch, P., Vleeshouwer, J., Frost, A., Bai, Q., Lerat, J., Leighton, B., Knapp, S., Warren, G., Van Dijk, A., Bacon, D., Pena Arancibia, J., Manser, P. and Shoesmith, J (2012). Operationalising the Australian Water Resources Assessment (AWRA) system. In: WIRADA: Science Symposium Proceedings; 1-5 August 2011; Melbourne. CSIRO; 2012. 36-45.