# Prov4J: A Semantic Web Framework for Generic Provenance Management

André Freitas, Arnaud Legendre, Seán O'Riain, Edward Curry
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
Galway, Ireland

***Abstract*—Provenance is a cornerstone element in the process of enabling quality assessment for the Web of Data. Applications consuming or generating Linked Data will need to become provenance-aware, i.e., being able to capture and consume provenance information associated with the data. This will bring provenance as a key requirement for a wide spectrum of applications. This work describes Prov4J, a framework which uses Semantic Web tools and standards to address the core challenges in the construction of a generic provenance management system. The work discusses key software engineering aspects for provenance capture and consumption and analyzes the suitability of the framework under the deployment of a real-world scenario.**

***Keywords- provenance management; semantic web.***

## I. INTRODUCTION

The Web is evolving into a complex information space where users have access to an unprecedent volume of information. The advent of Linked Data in the last years as the de-facto standard to publish data on the Web, and its uptake by early adopters[1][2], defines a clear trend towards a Web where users will be able to easily aggregate, consume and republish data. With Linked Data, Web information can be repurposed with a new level of granularity and scale. In this scenario, tracking the provenance of an information artifact will play a fundamental role on the Web, enabling users to determine the suitability and quality of a piece of information.

As a direct consequence, Linked Data applications will demand mechanisms to track and manage provenance information. This new common requirement is inherent to the level of data integration provided by Linked Data and it is not found in most systems consuming information from 'data silos', where the relationship among data sources and applications is, in general, more rigid.

Until now, provenance management has been a wide concern in the domain of scientific workflow systems [1, 2], enabling understandability and reproducibility in scientific experiments. Provenance on the Web introduces new and broader requirements for representing and managing provenance[3], as different communities are represented under the same space.

This work discusses provenance management from the perspective of this larger audience, describing Prov4J[4], a general-purpose open source provenance management system. The framework uses Semantic Web standards and tools to deploy a generic and standards-based solution. The paper also discusses key software engineering aspects in the process of designing the framework.

The central goal behind the design of the framework is to provide a set of core functionalities that enable users to develop provenance-aware applications, both from the consumption (discovery/query/access) and from the capture (logging/ publishing) perspectives.

The paper is structured as follows: section II introduces a motivational scenario; sections III and IV describe general aspects of provenance management and the architecture behind Prov4J; sections V and VI cover the consumption and capture cycles of provenance management, discussing the application of Semantic Web standards and tools in the construction of the framework. Section VII provides a brief analysis of the framework using a real world scenario based on the motivational scenario; section VIII present related work and section IX conclusions and future work.

This work concentrates its contributions: (1) in the description and analysis of a generic provenance framework for the Web using Semantic Web standards and tools; (2) in the analysis of the suitability of these standards and tools in the process of building this framework.

## II. MOTIVATIONAL SCENARIO

Financial analysts in an investment company are using information from the Web to help make investment decisions. Business related data aggregated from different Web sources is filtered, curated and analyzed, and financial reports about companies or investment areas are generated. Each report is a data mash-up and the provenance of each statement in the report should be tracked to its sources. The ecosystem of Web applications used for aggregating, filtering, curating, analyzing and visualizing the data should be provenance-aware, i.e. the

[1] http://data.gov.uk, UK Government Data
[2] http://data.nytimes.com/, NYT Open Data
[3] http://www.w3.org/2005/Incubator/prov/wiki/User_Requirements, W3C Provenance Incubator Group
[4] http://prov4j.org

historical trail of all the entities and processes behind the transformation of the original data need to be recorded and users should be able to access the provenance of data.

## III. A Generic Provenance Framework for the Web

The core goal behind Prov4J is the provision of a provenance management mechanism for the large set of applications which will increasingly need to capture and consume provenance information. As a result, Prov4J is targeted towards an application developer which needs to build provenance-aware applications.

According to Freire et al. [2], provenance management frameworks typically consist of three main components: a capture mechanism, a representational model and an infrastructure for storage, access and queries (provenance consumption). In Prov4J, the representational model is covered by the W3P provenance ontology[5], the capture mechanism is covered by *ProvLogger*, the component which is responsible for logging and publishing, and the provenance consumption is done by the *ProvClient* component.

W3P is a generic provenance ontology for tracking provenance on the Web. W3P is designed to be a lightweight provenance ontology, complementing and integrating vocabularies such as Dublin Core[6], and the ChangeSet[7] vocabulary. Other key features of W3P include the coverage of social provenance [3] and the maximization of the compatibility with the Open Provenance Model (OPM). Prov4J uses W3P as its default provenance model. Sections V and VI approach the consumption and capture cycles in Prov4J.

## IV. Architecture

In most applications, provenance represents a cross-cutting concern where the functionalities to capture and consume provenance are a complementary requirement to the core functionalities of an application. A cross-cutting concern is a common feature that is typically spread across objects in the application, being difficult to decompose from other parts of the system. Prov4J adopts a provenance architecture which reflects the separation between the core concerns of the application and the cross-cutting concern of provenance. The architecture maximizes the encapsulation of provenance capture and consumption functionalities in a separate layer (figure 1). The architecture behind Prov4J contains many elements in common with the general architecture proposed by Groth [4].

However, differently from classical examples of cross-cutting concerns (e.g. message logging and user authentication/access control), provenance capture and consumption is typically more tightly coupled with the logic structure of the calling application (process documentation perspective [2]) or to the data used in the application (data provenance perspective [5]), bringing challenges and practical limits to the isolation of the provenance concern inside the calling application.

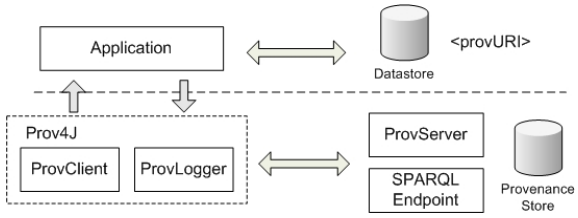


**Figure 1: Generic provenance management architecture.**

A common provenance scenario is the association of the information present in a procedural or object-oriented application to a data artifact in a generic data store (e.g. relational databases, XML or RDF data). The strategy used in Prov4J is to use RDF to represent provenance data and URIs to associate the described information resource in the core application layer with its provenance descriptor. This allows Prov4J to cope with both *data representation independency* and *separation of concerns*, important requirements for a generic provenance framework. A <*provURI*> is a connection point between the core application layer and the provenance layer, being an entry point into the provenance store. This allows an abstraction over the artifact type, which can be a relational tuple, RDF triples, a named graph, a XML element, a HTML element, etc.

The <*provURI*> mechanism also allows Prov4J to partially track data provenance. Data provenance is defined as the process of tracking the origins of data and its movement between databases [6]. Compared to the perspective of workflow provenance, data provenance approaches the problem under a database perspective, focusing on the relationships between data artifacts. A typical problem in this perspective is the representation of dependencies between data artifacts (i.e. on which artifacts a specific piece of data depends upon). Despite the fact that a complete data provenance tracking solution is highly dependent on the storage mechanism, Prov4J provides a basic functionality for mapping dependencies across data artifacts. This discussion is briefly detailed in section VI.

Prov4J also allows the discovery and consumption of provenance descriptors associated with different types of resources, including HTML pages, SPARQL endpoints, and RDF published as Linked Data. This allows Prov4J to respond to an important use case where an application is consuming provenance from third-party Web resources. Prov4J consists of two core components: *ProvClient* and *ProvLogger* (figure 2). ProvClient is responsible for the consumption cycle of the application, while ProvLogger provides an interface for provenance capture. A third element, *ProvServer*, is introduced in order to allow high performance provenance capture.

## V. Provenance Consumption

### A. Description

The consumption cycle inside Prov4J starts with the specification of the information sources which will be

[5] http://prov4j.org/w3p/schema#
[6] http://dublincore.org
[7] http://vocab.org/changeset/schema.html

consumed: users can specify the location (URIs) of information resources that have associated provenance descriptors or the URIs of provenance data sources. There are three types of supported provenance sources: provenance stores (which are SPARQL endpoints), linked provenance data (RDF published using the Linked Data principles [8]) and provenance descriptors (RDF data embedded in different formats).

Each type of provenance data source has a different consumption approach. Data in provenance stores are consumed after a user query is defined over the API. Linked provenance data is consumed using a navigational approach [7], where provenance is queried by successive navigation over the provenance graph (de-referencing each of the provenance entities and loading the returned RDF into a memory model). Figure 2 shows the basic components inside the framework including the components for provenance discovery and RDF extraction under different publication protocols (Provenance Discovery and Parsers), the components for Linked Provenance Data navigation (Linked Data Navigator) and provenance store data consumption (Client).

The provenance graphs collected from different sources are then loaded into a memory model. The framework uses two basic internal provenance structures: a provenance graph (*ProvGraph*) and a provenance view (*ProvView*). A ProvGraph represents the basic fragment of provenance information associated with a data source. One or more different ProvGraphs can be loaded into a single model by using a ProvView. The ProvView is the model where users have a consolidated provenance view over a set of different provenance data sources.

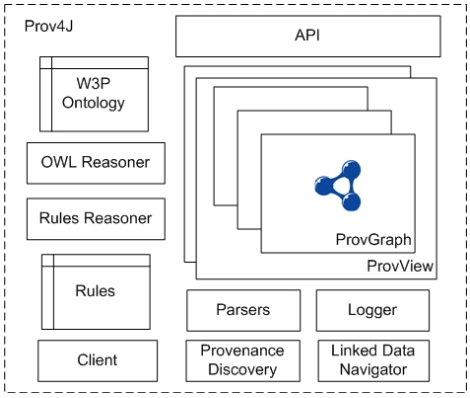


**Figure 2: Prov4J key components.**

After all provenance graphs are merged into provenance views, elements from different vocabularies are mapped into W3P entities using rules reasoning. Rules provide an expressive mechanism which allows complex mappings between different vocabularies which cannot be addressed by *owl:equivalentClass* or *owl:equivalentProperty*. Examples of more complex mappings across different provenance representations can be found in Miles [8], which defines OPM Profiles for Dublin Core vocabulary elements. The use of rules for vocabulary mappings also allows the representation of the mappings in standardized representations such as SWRL[9].

Once the vocabularies are mapped into W3P elements, the framework applies RDFS/OWL reasoning over the provenance model. *owl:TransitiveProperty* and *owl:InverseProperty* are used in W3P to improve the number of provenance queries answered by the framework (*subsection B* in this section). The consumption process is dependent on the type of provenance data source: for provenance stores and linked provenance data, the reasoning is done at query time, while for descriptors the *discovery-parsing-reasoning* is done during the definition of data sources on the interface. Prov4J uses the Jena framework [10] in its core and Pellet [9] is used for both OWL and Rules reasoning. Users can disable both types of reasoning from the API.

### *B. Provenance Queries*

The provenance consumption API (ProvClient) provides the core operations over the provenance views. The ProvClient API contains key interface methods for a set of provenance queries, minimizing the interaction from users with SPARQL. A SPARQL query interface is also exposed to allow non-predefined types of queries over the model. The framework supports five query categories:

**SPARQL based queries:** Provenance queries supported by the elements of the SPARQL specification [11] are accessible by using the direct query over the provenance model or by using API methods for common queries. Prov4J SPARQL also includes syntactic extensions: GROUP BY, HAVING and aggregation. ARQ with syntactic extensions [12] is the core query engine behind Prov4J.

**Queries supported by reasoning:** Some key provenance queries over W3P can be addressed by applying OWL reasoning over provenance data. Examples of queries of this type involve the determination of indirect relationships/dependencies in a workflow chain, such as "*list all artifacts which were used directly or indirectly in artifact X*". Similarly, rules can be applied to improve query expressivity.

**Path queries:** One important feature for provenance queries is the ability to query paths over provenance trails. A typical path query is "*show all the processes between artifacts A and B*" or more specifically "*list all the trails containing a process which uses artifact C between artifacts A and B*". Regular expressions queries over RDF elements can be used for expressing provenance path patterns. Prov4J uses the Gleen SPARQL extension described in [10] for path queries. Prov4J users can launch their own path queries or can access some of the functionalities provided by Gleen through API methods.

[8] http://www.w3.org/DesignIssues/LinkedData.html

[9] http://www.w3.org/Submission/SWRL/

[10] http://jena.sourceforge.net

[11] http://www.w3.org/TR/rdf-sparql-query/

[12] http://jena.sourceforge.net/ARQ/extension.html

**Navigational queries:** In some scenarios the primary way to consume provenance information is through RDF published as Linked Data. In this case Prov4J provides two interfaces: one for users browsing provenance data and the other for navigational queries. In the first case, the first level provenance descriptor of an artifact is available as a de-referentiable URI. The RDF provenance data can be consumed by the application and further de-referentiations are directed by user input (in this case Prov4J provides a simple interface for node de-referentiations). A second type of navigation provided by the framework is through the provision of iterators to provenance nodes where provenance properties are used to determine which provenance nodes to de-reference. For example, the iterator defined by the property *w3p:used* can be used to navigate through a chain of artifact dependencies. The third functionality is defined by the idea of navigational queries, which are mechanisms to query Linked Data by launching a SPARQL query over a collection of RDF graphs collected from a de-referentiable URI entry point [7]. In the case of Prov4J, a simple de-referentiation algorithm follows the provenance links until it reaches a pre-configured limit.

**Similarity queries:** One type of provenance query refers to the similarity analysis between two provenance graphs. This type of comparison can be used in the determination of similar workflow conditions and have potential applications in quality assessment scenarios. A user may trust a specific workflow and may want to query for similar or identical conditions. The matching process used in Prov4J is based on the approach described by Oldakowsky & Bizer [11] adapted to the W3P provenance model.

### C. Provenance Discovery

Provenance Discovery consists in automatically discovering the provenance given an information resource and it is an important requirement for a generic provenance management framework for consuming provenance data on the Web. Information resources can be HTML pages, elements inside the page, SPARQL endpoints, RDF files or de-referentiable URIs. A provenance discovery mechanism should not rely on centralized crawled provenance repositories: it should always be possible to navigate from the artifact to its provenance descriptor. Prov4J supports four mechanisms to discover provenance on the Web:

**Semantic Sitemaps + robots.txt:** Used to discover the provenance descriptor of a dataset having as a starting point a domain name. As covered in [12], the mechanism used by voiD [13], using *robots.txt* and the *semantic sitemaps extension*, can be used to discover dataset provenance descriptors.

**Linked Provenance Data:** Provenance descriptors can be published as Linked Data in two ways: (1) the URI represents an artifact and links directly to other provenance properties, (2) a provenance property such as *w3p:provenance* links the URI to the starting point of a provenance descriptor (a mirror to the provenance layer representation of the artifact).

**Embedded RDFa:** Provenance data can be embedded as RDFa in HTML pages.

**POWDER:** POWDER (Protocol for Web Description Resources) [13] is a W3C recommendation which provides a standard for describing general Web resources. Provenance descriptors can be embedded as RDF payloads in POWDER files.

## VI. PROVENANCE CAPTURE

One key challenge in the process of building a generic provenance capture framework is the process of providing a simple yet expressive provenance interface. The ability to express provenance accurately and with the mimimum amount of intervention in the application is a fundamental feature in the process of introducing the provenance functionality in existing applications. In order to achieve this objective, the provenance capture engine was built using the following principles:

**Pushback capture:** Provenance capture or logging can be implemented as pushback operation, where, from the capture interface perspective, new provenance information is inserted but never deleted or updated. This assumption is consistent with the fact that provenance maps to the actual temporal execution flow of the application. Instead of allowing a full interaction with the provenance store, the ProvLogger interface is primarily designed for pushing back fragmented provenance logs, which are reconstructed in the provenance store (concept present in [4] and [14]).

**Minimization of adaptations:** Prov4J capture interface can be used to implement *adaptations*, a concept defined by Munroe [14], in a software engineering methodology designed for the development provenance-aware applications (PrIMe). Adaptations allow actors to record process documentation, adding the provenance functionality to the application. Relations among *entities* in the provenance model can be determined based on the *execution scope* of these elements. Temporal relations, order relations, relationships between agents, processes and artifacts in the same execution scope are examples of provenance data which can be determined without explicit *adaptations*. The ProvLogger component minimizes the user input in the construction of the provenance model, *'*filling the gaps*'* in the provenance model. Figure 3 shows examples of adaptations.

**Provenance URIs:** In some cases, provenance entities can be interconnected with elements in different parts of the workflow (e.g. a process consuming an artifact that was generated by another process at a different time). The logger interface provides a mechanism to interconnect provenance entities in different execution scopes. Users can associate different provenance entities by using internally the concept of ApplicationId-URI mapping, which associates Ids inside the application to provenance URIs. These associations can also be done directly by referencing directly provenance URIs. To

[13] http://www.w3.org/TR/powder-dr/

minimize the performance impact, this mechanism relies in the construction of a provenance URI cache in the capture mechanism.

**Annotations:** Java Annotations provide a mechanism to map the structure of an application to provenance elements. Annotations also allow users to provide provenance relationships valid in a specific scope. Provenance entities inside the scope of a method may be directly associated with an entity represented in the annotation, depending upon their relationship (figure 3). The design of ProvLogger allows users to express provenance information by maximizing the mapping between the application object structure and the provenance elements. In this case, classes, methods and member variables can directly map to provenance artifacts, processes and agents by using annotations.

The ProvLogger interface uses the concepts of aspect oriented programming (AOP) and Java annotations to maximize the isolation between cross-cutting concerns, allowing users to separate distinct functionalities of a software. AOP combined with Java Annotations can provide a powerful mechanism to implement the separation of concerns in provenance capture.

```
@Host("DERI")                                                    (1)
@Owner("DERI")
@Process("Report_Creation")
public void createReport() throws URISyntaxException, IOException {
        // … method operations
        ProvLog.generates(m_report);
        ProvLog.triggeredBy(m_username);
        // Process, Artifact and Agent are associated behind the scenes
}
```

```
@Host("DERI")                                                    (2)
@Process("News_Aggregation")
public void aggregate() throws URISyntaxException, IOException {
        String newsId = …                    // query news data here
        ProvLog.artifact(newsId);            // artifact gets a provenance id
        String newsProvURI = ProvLog.getURI(newsId);
        ProvLog.consumed(newsProvURI);
        ProvLog.triggeredBy(m_username);
        // timestamps are set in background
}
```

**Figure 3: Examples of adaptations using the capture interface.**

After the information is collected in the capture interface, the provenance log information is sent to the *ProvServer* and translated into a SPARQL/Update query to the provenance store. Prov4J relies on Scribe [14], a high-performance logging mechanism for the communication and distribution of provenance logs.

## VII. Framework Analysis

The framework was analyzed using the scenario described in section II. A provenance dataset was generated using real financial data aggregated from multiple data sources, which focused on news and opinions about businesses collected from the Web. These data elements defined the ground artifacts which were further aggregated, curated and analyzed in a financial analysis workflow simulator. The output of the workflow is a report for a specific company, which is a mash-up of business data[15]. The provenance of the final report and each of its artifacts is tracked down to their original sources. In the experiment, business reports were generated with data collected for 100/500/1000 companies with up to 100/500/1000 news respectively for each company. Details about the experiment are outlined in table I. The experiment sought to determine the performance of the framework in a realistic scenario.

| *Data set* | *Reasoning level* | *# triples* | *min query (ms)* | *max query (ms)* | *Reasoning (ms)* |
|---|---|---|---|---|---|
| 1000 | voc | 674.786 | 1,2 | 680,6 | 2.717,9 |
| | voc+owl+rules | 686.829 | 2,4 | > 90.000 | 314.846,2 |
| 500 | voc | 231.217 | 1,2 | 246,1 | 959,8 |
| | voc+owl+rules | 234.572 | 1,1 | 22.445,4 | 44.536,9 |
| 100 | voc | 84.520 | 1,2 | 217,9 | 602,9 |
| | voc+owl+rules | 87.204 | 1,4 | 5.180,7 | 16.246,9 |

**Table I: Prov4J performance metrics.**

The experiment was run in a 2.53 GHz Intel Core 2 Duo computer with 4 GB of memory. The minimum level of reasoning enabled was vocabulary rules mapping (*voc*) and the maximum added OWL features and 5 additional rules (*voc+owl+rules*). The input provenance data was consumed from 2 RDF files which were merged inside the framework. In the experiment path-based queries showed the highest execution time (*max query*), being highly sensitive to reasoning. SPARQL queries over basic elements of the ontology accounted for the lowest execution time (*min query*). Most of the queries (aggregate included) showed low execution time increase after the reasoning was enabled. Navigational and similarity queries were not tested. The 1000/1000 dataset counted 53.844 processes, 20.179 artifacts and 30 agents. The framework was able to do reasoning and answer the majority of provenance queries present in the API with acceptable runtime latencies. The scalability of reasoning could, however, represent a problem for larger provenance datasets.

## VIII. Related Work

Different approaches for provenance management have been described in the literature. Pegasus [15] is a workflow-based system that uses both OWL and relational databases to represent provenance. ES3 [16] is an OS-based provenance system that represents provenance data in XML and provides query support through XQuery. PReServ [17] is a process-based provenance recorder that allows the integration of provenance into third-party applications. PReServ uses XML to persist provenance data; queries are provided through a Java query API and XQuery. In [18] Bochner et al. describe a python client library for provenance recording and querying in a PReServ store. Taverna [19] is a workflow-based system which represents provenance in both Scufl Model and prospective provenance in RDF. SPARQL is used as a query language. In [20], Sahoo et al. present PrOM, a Semantic Web provenance management framework focused on scalable querying for eScience. The reader is referred to [1, 2] for comprehensive surveys and analysis of existing provenance management systems.

[14] http://sourceforge.net/projects/scribeserver/

[15] http://prov4j.org/w3p/scenario/financial.html

Compared to existing works, Prov4J stands out as most heavily leveraging Semantic Web standards and tools, using RDF as its core provenance representation and both OWL and rules reasoning over provenance data. In addition, Prov4J extends existing SPARQL query capabilities: a Java API, SPARQL aggregate functions, regular expression path queries and similarity queries together with reasoning provide additional query expressivity. Prov4J also incorporates important requirements for the Web: provenance discovery and Linked Data navigation. Compared to PrOM, which is targeted towards the provision of a scalable query mechanism for an eScience scenario, Prov4J focuses on generic provenance management for the Web including both provenance capture and discovery. Similarly to the combination PreServ + Python Provenance Client Library, Prov4J is designed as an independent provenance layer, being designed for the provision provenance-awareness to generic applications.

In [12], Hartig and Zhao covers the main aspects of publishing and consuming provenance in the Web of Data using the Provenance Vocabulary. The provenance discovery mechanism behind Prov4J shares common aspects with the publication methodology described in their work. Additionally, the mapping mechanism behind Prov4J allows the framework to consume and query Provenance Vocabulary descriptors.

## IX. Conclusion & Future Work

This work described Prov4J, a generic provenance management framework. The design of the framework focused on the following features: (1) the provision of expressive provenance queries; (2) the maximization of the use of Semantic Web standards to address the challenges of managing provenance data; (3) software engineering aspects for provenance capture; (4) discovery mechanisms for provenance descriptors on the Web. The use of Semantic Web tools and standards to address these challenges played a fundamental role in the construction of the framework. Prov4J benefited largely from: the use SWRL-like rules to map and align different provenance vocabularies; OWL reasoning to address a subset of provenance queries; use of different publishing protocols (POWDER, semantic sitemaps, etc) for provenance discovery on the Web; SWRL-like rules applied to the enrichment of the provenance structure; RDF to represent the bulk of provenance data, SPARQL as a query mechanism and Linked Data as a publication mechanism for the Web. The use of non-standardized extensions over existing standards such as aggregate SPARQL queries, SPARQL/Update and path queries provided important features for the framework. The ensemble of these technologies proved to achieve a good performance under a realistic provenance scenario.

From the software engineering perspective, Prov4J orchestrates different strategies to maximize the separation between provenance aspects and core concerns and to reduce the number of application adaptations for provenance capture.

Future work will include a detailed analysis of the query expressivity and query performance of the framework. A mapping mechanism from W3P to OPM profiles using rules is planned. One current limitation of the framework is related to the deployment of security and integrity mechanisms. In addition, an in-depth comparative study across existing provenance management systems is planned. Improvements over the framework to transform Prov4J from an experimental to a robust provenance solution are set as a priority.

## Acknowledgment

The work presented in this article has been funded substantially by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2). The authors want to express their gratitude to Tomas Knap and Micah Herstand for their support in the elaboration of this work.

## References

[1] Y. Simmhan, B. Plale, and D. Gannon, “A survey of data provenance in e-science,” *SIGMOD Record*, vol. 34, 2005, pp. 31-36.

[2] J. Freire, D. Koop, E. Santos, and C.T. Silva, “Provenance for Computational Tasks: A Survey,” *Computing in Science Engineering*, vol. 10, no. 3, 2008, pp. 11 -21.

[3] A. Harth, A. Polleres, and S. Decker, “Towards A Social Provenance Model for the Web,”, 2007.

[4] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau, “ An Architecture for Provenance Systems,”, ECS, University of Southampton., 2006.

[5] P. Buneman, S. Khanna, and W.C. Tan, “Why and Where: A Characterization of Data Provenance,” *ICDT*, pp. 316-330.

[6] P. Buneman, and W.C. Tan, “Provenance in databases,” *SIGMOD Conference*, pp. 1171-1173.

[7] P. Bouquet, C. Ghidini, and L. Serafini, “A Formal Model of Queries on Interlinked RDF Graphs,” *AAAI Spring Symposium Series*.

[8] S. Miles, L. Moreau, and J. Futrelle, “OPM Profile for Dublin Core Terms,” *Book OPM Profile for Dublin Core Terms*, Series OPM Profile for Dublin Core Terms, 2009.

[9] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, “Pellet: A practical OWL-DL reasoner,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, 2007, pp. 51-53.

[10] L.T. Detwiler, D. Suciu, and J.F. Brinkley, “Regular paths in SparQL: querying the NCI Thesaurus,” *AMIA Annual Symposium proceedings / AMIA Symposium. AMIA Symposium*, 2008, pp. 161-165.

[11] R. Oldakowski, and C. Bizer, “SemMF: A Framework for Calculating Semantic Similarity of Objects Represented as RDF Graphs (Poster).”, 2005.

[12] O. Hartig, and J. Zhao, “Publishing and Consuming Provenance Metadata on the Web of Linked Data,” *Book Publishing and Consuming Provenance Metadata on the Web of Linked Data*, Series Publishing and Consuming Provenance Metadata on the Web of Linked Data, 2010.

[13] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao, “Describing Linked Datasets,” *Proceedings of the Linked Data on the Web Workshop LDOW 2009*.

[14] S. Munroe, S. Miles, L. Moreau, and J. Vazquez-Salceda, “PrIMe:A software engineering methodology for developing provenance-aware applications,” *Foundations of Software Engineering*, 2006, pp. 39-39.

[15] J. Kim, E. Deelman, A. Gil, G. Mehta, and V. Ratnakar, “Provenance Trails in the Wings/Pegasus System”.

[16] J. Freire, D. Koop, L. Moreau, J. Frew, and P. Slaughter, “ES3: A Demonstration of Transparent Provenance for Scientific Computation,” *Provenance and Annotation of Data and Processes*, Lecture Notes in Computer Science 5272, Springer Berlin / Heidelberg, 2008, pp. 200-207.

[17] P. Groth, S. Miles, and L. Moreau, “PReServ: Provenance Recording for Service,”, 2005.

[18] J. Freire, D. Koop, L. Moreau, C. Bochner, R. Gude, and A. Schreiber, “A Python Library for Provenance Recording and Querying,” *Provenance and Annotation of Data and Processes*, Lecture Notes in Computer Science 5272, Springer Berlin / Heidelberg, 2008, pp. 229-240.

[19] T. Oinn et al., “Taverna: lessons in creating a workflow environment for the life sciences:Â Research Articles,” *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, 2006, pp. 1067-1067.

[20] S.S. Sahoo, R. Barga, A. Sheth, K. Thirunarayan, and P. Hitzler, “PrOM: A Semantic Web Framework for Provenance Management in Science,”, 2009.