Published in final edited form as:

Proc ACM/IEEE Joint Conf Digit Libr. 2017 June ; 2017: .

Automating data citation: the eagle-i experience

Abdussalam Alawini,
University of Pennsylvania, Philadelphia, PA

Leshang Chen,
University of Pennsylvania, Philadelphia, PA

Susan B. Davidson,
University of Pennsylvania, Philadelphia, PA

Natan Portilho Da Silva, and
Feevale University, Novo Hamburgo, Brazil

Gianmaria Silvello
University of Padua, Padua, Italy

Abstract

Data citation is of growing concern for owners of curated databases, who wish to give credit to the contributors and curators responsible for portions of the dataset and enable the data retrieved by a query to be later examined. While several databases specify how data should be cited, they leave it to users to manually construct the citations and do not generate them automatically.

We report our experiences in automating data citation for an RDF dataset called eagle-i, and discuss how to generalize this to a citation framework that can work across a variety of different types of databases (e.g. relational, XML, and RDF). We also describe how a database administrator would use this framework to automate citation for a particular dataset.

1. INTRODUCTION

An increasing amount of information is being stored in structured databases and retrieved using queries. Since much of the information in these databases is contributed by members of the community and curated by experts, there is increasing interest in understanding how to cite the result of a query and give credit to the people or organizations who were responsible for it. Currently, several databases describe (in English) what “snippets” of information are to be included in a citation for information displayed as a web page (e.g. the Reactome Pathway database [1] and eagle-i [2]), but few databases generate the citation automatically. A notable exception to this is the IUPHAR/BPS Guide to Pharmacology [3] in which the citations for specific web-page views of the database are hard-coded in the web

[1] http://www.reactome.org/pages/documentation/citingreactome-publications/
[2] https://www.eagle-i.net/get-involved/forresearchers/citing-an-eagle-i-resource/
[3] http://www.guidetopharmacology.org/

page results. However, none provide a citation for *general queries* over the database, i.e. those which do not correspond to web page views of the database.

This problem is especially hard because, unlike traditional publications which have a fixed granularity to which citations can be attached (e.g. a paper in a conference proceedings, or chapter in a book), the granularity of data varies when retrieved by a query over a database. Since there are a potentially infinite number of queries, each accessing and generating different subsets of the database, it is impossible to explicitly attach a citation to every possible result set and/or query. Instead, we must find ways of specifying citations for portions of the database which represent frequent queries (e.g. web page views), and use these to automatically construct citations for data returned by more general queries. There is therefore an interesting connection between data citation and the problem of query rewriting using views [20], as observed in [9].

A first step in obtaining a solution to this problem is to develop an approach for *specifying* the frequent queries as well as how to construct citations to those queries. In [13] we introduce a notion of *citation views* in the context of relational databases, and show how they can be used to specify frequent queries as well as how to obtain the snippets of information that are to be included in the citation. In this paper, we discuss how citation views can be specified in the context of *RDF databases*[4], and describe the implementation of an automatic citation generator for frequent queries against the eagle-i database.

The eagle-i database is a “resource discovery” tool built to facilitate translational science research which allows researchers to share information about resources. Resources are added to the database by project participants through data entry screens, with required and optional fields which depend on the type of the resource (e.g. cell lines, software, …). A resource is later retrieved in response to a query on the eagle-i id of the resource or via keyword search, and the information displayed as a web page. Embedded in the web page is a button saying “Cite this resource”. When this button is clicked, the system returns the eagle-i id of the resource (which is the http address of the web page), and describes (through yet another click yielding a web page) what snippets of information on the original web page should be put in a citation for the resource. Users must then construct citations manually. Clearly, since eagle-i is stored as an RDF dataset, the snippets of information to be included in the citation for a resource can be easily retrieved using SPARQL [7], the W3C standard query language for RDF data [5]. We therefore developed a prototype which, given a resource identified by its eagle-i id, obtains the necessary snippets and renders the citation in a number of different formats, e.g. human readable, BibTex, XML or RIS. Users can then copy-paste the appropriate format to the bibliography generator of their choice.

However, SPARQL is too complex to use as a specification language for citation views, if the ultimate goal is to be able to generate citations for general queries. In this paper, we therefore propose to use (positive) Datalog [3] as the model and specification language for RDF citation views, and show that they can be easily implemented using SPARQL.

[4] https://www.w3.org/RDF/
[5] https://www.w3.org/TR/rdf-sparql-query/

Unfortunately, since eagle-i is not versioned, when a citation is "dereferenced" the latest version of the resource will be obtained rather than the version seen when the citation was generated; there is no guarantee of *fixity*. We therefore developed a versioning system for eagle-i which, given an eagle-i id and time of access, returns the appropriate version of the resource.

In this paper, we discuss our experiences in automating citation for eagle-i resources (see Figure 1). When the "Cite this resource" button of an eagle-i resource is clicked, the eagle-i id of the resource is passed to the citation model module, which gathers the necessary snippets of information from the underlying eagle-i (RDF) triple-store and generates a citation in JSON format. The JSON object can then be rendered in a variety of formats (e.g. human readable, XML, RIS, BibTex…). When a citation is "dereferenced", the correct version of the eagle-i resource is obtained from the versioning system and returned to the user.

We also discuss how this framework can be reused for automating data citation across a variety of different database models, e.g. RDF, relational and XML. This is enabled by using Datalog [3] as the specification language for citation views. Datalog is an elegant formalization which can be used to capture the core of many query languages for relational, graph-based, and semi-structured data. In particular, it has been used to model and study the expressive power of SPARQL [7].

We start in Section 2 by presenting our model of citation views, showing how to specify eagle-i citations in Datalog, and discussing the translation to SPARQL. Section 3 discusses the problem of fixity, and presents our RDF versioning system. Section 4 discusses problems encountered in using standards such as BibTex, RIS or DataCite's XML for eagle-i citations, and argues the need for flexible formats. After discussing related work in Section 5, we describe a general framework for automating data citation in Section 6, discussing the steps that database owners must take to use the framework. Finally, we conclude and outline future work in Section 7.

2. MODEL AND IMPLEMENTATION

Eagle-i is a resource discovery tool which allows users to search for resources by keywords, and displays information about a resource of interest (identified by its eagle-i id) as a web page. Resources fall into about 20 top-level categories, such as software, mice, cell lines, and core facilities. Each resource is a citable object, and the content of the citation depends on the category of the resource. Thus the frequent queries to the database are selecting a resource based on its eagle-i id and displaying the web-page view according to the category of the resource. The citation views for eagle-i are formed around these frequent queries, one for each top-level category of resource.

In this section, we show how to use Datalog as the specification language for citations views in eagle-i, discuss how they were then implemented in our prototype system, and discuss how Datalog can be used to reason about citations for *general* queries, i.e. those that go beyond the frequent queries used to specify citation views.

2.1 Model

We start by describing how to capture an RDF database as a set of Datalog *facts* and then use Datalog *rules* to specify citations views. A Datalog *rule* is an expression of form

$$R_1(u_1) :- R_2(u_2), \ldots, R_n(u_n)$$

where $n \geq 1$, the R_i's are predicates, and the u_i's are tuples of appropriate arities. The *head* of the rule, R_1, is on the left of the "if" sign : –, and the *body* of the rule, composed of one or more subgoals, is on the right side. The interpretation of such a rule is that whenever there is a binding of values to variables in u_2, …, u_n that makes each R_i true (conjunctive semantics), then $R_1(u_1)$ must also be true.

2.1.1 RDF facts and inference—An RDF graph is a set of subject-predicate-object triples, where the elements may be IRIs (Internationalized Resource Identifiers), blank nodes, or data-typed literals. A fragment of RDF related to an eagle-i resource is shown below, where `rdfs:subClassOf`, `rdfs:type`, and `rdfs:label` are built-in predicates.

```
<ont: t1> <rdfs:label> <"Software">
<ont: t2> <rdfs:label> <"Software component">
<ont: t3> <rdfs:label> <"Algorithmic component">
<ont: t2> <rdfs:subClassOf> <ont: t1>
<ont: t3> <rdfs:subClassOf> <ont: t2>
<eagle-id: e1> <rdfs:type> <ont: t3>
<eagle-id: e1> <Name> <Significance Tester>
<eagle-id: e1> <Developer> <Grant, G.>
<eagle-id: e1> <Developer> <Lazar, M.l>
<eagle-id: e1> <Developer> <Manduchi, E.>
<eagle-id: e1> <Org> <UPenn>
<eagle-id: e1> <URL> <http://www.cbil.upenn.edu/STAR/>
```

The first five triples define (a portion of) the resource type ontology related to software. The next seven triples define the eagle-i resource with id "e1", which is an "Algorithmic component" (which is, more generally, "Software"). This dataset can be represented as a set of facts, where the subject and object become arguments to the predicate. For example,

```
<ont: t2> <rdfs:subClassOf> <ont: t1>
```

would become

```
rdfs:subClassOf('ont: t2', 'ont: t1')
```

By convention, the built-in predicate `rdfs:subClassOf` is transitive, i.e. if *x* is a subclass of *y* and *y* is a subclass of *z*, then *x* can be inferred to be a subclass of *z*. This can be captured using Datalog by defining an intensional database predicate `subClassOf*` which is recursively derived from `rdfs:subClassOf` as follows:

```
subClassOf*(x, y):- rdfs:subClassOf(x,y).
subClassOf*(x, z):- subClassOf*(x,y),
                                subClassOf*(y,z).
```

Using these two rules on the RDF instance given above, we could infer the following:

```
subClassOf*('ont: t2', 'ont: t1').
subClassOf*('ont: t3', 'ont: t2').
subClassOf*('ont: t3', 'ont: t1').
```

Additional information may also be needed for inclusion in a citation. For example, the database as a whole may have a standard reference to the literature (e.g. [34]), or there may be a current version number or timestamp included to ensure fixity. We model this using the predicate MetaData(x, y), a "key-value" store in which x is the key and y is the value.

2.1.2 Citation Views—Citation views have three components: a view query, a citation query and a citation function. *View queries* define citable portions of the database; collectively, they represent some portion of the expected query workload. Each view has an associated *citation query*, which obtains the snippets of information to be used in a citation. The output of the citation query is then used by a *citation function* to format the snippets to create the final citation.

Both the view and citation query are expressed as Datalog queries, optionally *parameterized* by one or more variables. The effect of the parameters is to define separate views for each binding of parameters to values, and therefore generating different citations. For example, a citation in eagle-i is associated with an *individual* resource, and therefore the view definitions (and corresponding citation queries) are parameterized by the resource identifier (the eagle-i id). The snippets of information captured in a citation depends on the *category* of the resource, therefore a different view is defined for each category of resource.

As an example, views for software (SW) and cell line (CL) would be defined as follows. The views are parameterized by the resource identifier, indicated by the term λ `(r)`:

```
λ(r)V_SW(r) :- rdfs:Type(r,t), SubClassOf*(t,s),
                                    rdfs:Label(s,"Software")
λ(r)V_CL(r) :- rdfs:Type(r,t), SubClassOf*(t,s),
                                    rdfs:Label(s,"Cell Line")
```

The view query tests for the top-level class of the category of the resource `r` (`SubClassOf*`). If it is “Software” then V_{SW} is used; similarly, if it is “Cell Line” then V_{CL} is used (and so on for the other categories).

Given the view that the resource matches, a different citation is specified. The citation for a software resource includes the software name, optional software version number, optional developer name(s), optional team name, and optional owning organization name. We extend this to include the version number of the database to enable fixity. The citation query for software obtains the necessary snippets as follows:

```
λ(r)C_SW(r,s,v,d,t,o,u,y) :- Name(r,s), SoftVersion(r,v)?,
                                                    Developer(r,d)?,
Team(r,t)?,
                                                    Org(r,o)?, URL(r,u),
                                                    MetaData(“Version”, y)
```

Note that we have extended the syntax of Datalog slightly to reflect the *optional* snippets (indicated by ?), adopting an outer-join semantics rather than a join semantics.[6] The result of C_{SW} for the RDF fragment above, assuming the current database version to be V12, is shown in Table 1.

The citation function could then format this information in JSON as follows:

```
{eagle-id: ”eagle-id: e1”, db-version: “V12”,
  name: ”Significance Tester”,
  developers: {“Grant, G.”,”Lazar, M.l”, “Manduchi, E.”},
  url: ”http://www.cbil.upenn.edu/STAR/”}
```

The citation query of a cell line resource would be slightly different to reflect the necessary snippets for that resource category, e.g. the cell line name, resource type, optional other resource identifiers, and owning organization name.

Figure 2 illustrates how the three components of the citation model interact with one another. In this RDF graph, nodes are resources, other IRI’s, or literals, and edges are properties. The two parameterized views described above, V_{SW} and V_{CL}, partition the RDF graph into subgraphs representing individual resources; for sake of illustration, we have extended the original RDF dataset to include a cell line resource, e_4. Given an input eagle-i id e_1, we check the category of the resource and find that it matches the view $\lambda(e_1)V_{SW}$, with associated citation query $\lambda(e_1)C_{SW}$; note that C_{SW} is parametrized by the resource e_1. Finally, the query is evaluated and the result set is processed by the citation function, which produces a citation in JSON.

[6]This can be modeled as a set of Datalog clauses, one for each optional predicate. Details are omitted for simplicity.

2.2 Implementation

There is an easy translation from this extension of Datalog into SPARQL queries. For example, the citation query C_{SW} shown above, would be translated into the following SPARQL query to retrieve the necessary snippets of information from the underlying eagle-i dataset:

```
SELECT ?name ?version ?developer ?team ?org ?url
WHERE {
<eID> rdfs: label ?name.
<eID> owl: has_url ?url.
OPTIONAL{<eID> owl: has_version ?version}
OPTIONAL{<eID> owl: has_developer ?developerURI.
            ?developerURI rdfs: label ?developer}
OPTIONAL{<eID> owl: has_team ?teamURI. ?teamURI
            rdfs: label ?team}
OPTIONAL{<eID> owl: located_in ?organizationURI.
            ?organizationURI rdfs: label ?org}
}
```

In our eagle-i citation framework, the view and citation queries stored as SPARQL queries. The framework is shown in Figure 1, and consists of three components: Citation Generator, Citation Dereferencer and Versioning Manager.

Given an eagle-i id `<eID>` (an instance of the term λ `(r)` described above), the RDF Citation Model issues the stored SPARQL view query to determine the category of the resource. The stored citation query for that category of resource is then executed with parameter `<eID>` to retrieve the necessary snippets of information from the underlying eagle-i dataset. Finally, the Citation Model generates the citation as a JSON object (as shown in Section 2.1.2), and passes it to the Citation Formatter module. Based on the user's preference, the citation formatter can then render the JSON object as a human readable, XML, RIS or BibTex object.

The remaining components of the framework, Citation Dereferencer and Versioning Manager, will be discussed in Section 3.

2.3 Discussion

We chose Datalog as a specification language for citation views since it captures the core of many query languages and can therefore be used with a variety of different types of database systems. Our experience with eagel-i and several other biomedical databases we have worked with also shows that it is sufficiently expressive to capture the type of views to which citations are attached: conjunctive queries, also known as select-project-join-union (SPJU) queries. However, from a user perspective, a more intuitive, user-friendly API could easily be developed which compiles into Datalog as an intermediate language.

An added advantage of Datalog is that it has a very well-developed theory, and has been extensively used in the context of problems such as query optimization, maintenance of physical data independence, and data integration [14, 20, 23]. In particular, the notion of *query rewriting using views*, which is at the center of these problems, can be used to construct citations for *general queries*. Here the problem is: Given a set of view queries V_1, …, V_n and a general query Q, can Q be rewritten to obtain an equivalent query Q' in terms

of some subset of $V_1, \ldots, V_n$? And if so, how can the associated citation views $C_1, \ldots, C_n$ be used to create a citation for Q?

The problem is especially interesting since there may be several alternate rewritings of Q, say $Q_1, \ldots, Q_n$, and each rewriting Q_i may jointly use more than one view query. The citation for Q, Cite(Q), can therefore be expressed in terms of the citation views as:

$$\text{Cite}(Q) = \text{Cite}(Q_1) + \ldots + \text{Cite}(Q_n)$$

where + represents *alternate* use. In turn, Cite(Q_i) can be expressed in terms of its component view queries (call these $\{V_{i1}, \ldots, V_{in_i}\}$) as:

$$\text{Cite}(Q_i) = \text{Cite}(V_{i1}) * \ldots * \text{Cite}(V_{in_i})$$

where * represents *joint* use. Examples of interpretations for + include union or the least-upper bound in some ordering over views. Similarly, interpretations for * include union or some sort of join. The interpretation of + and * are *policies* that must be specified by the database owner.

In future work, we will study what the common interpretations of + and * are, and how to optimize query rewriting according to common policies.

3. FIXITY

Fixity is a crucial requirement for accurate data citation. It ensures that when a citation is dereferenced, the version of the data as of the time of citation becomes available to the reader. However, eagle-i does not currently guarantee fixity since only the latest version of the database is visible; cited data may therefore have changed or become unavailable when the citation is dereferenced. We therefore developed a web service for versioning eagle-i datasets called *eagle-iV*.

In this section, we describe how eagle-iV is implemented, what happens when a citation is dereferenced, and discuss performance considerations.

3.1 eagle-i Versioning Service

Although there has been quite a bit of research on versioning RDF data sources, we were unable to find an RDF versioning tool that could be easily integrated with eagle-i. The difficulties were that: i) most systems do not provide a standard interface for integrating with existing repositories; ii) some systems, such as SemVersion [35], are discontinued; and, iii) several systems, including R&Wbase [29] and R43ples [17], do not support time queries [24], which we discuss below. We therefore decided to implement a new versioning system for eagle-i, building on previous ideas. The prototype system is currently limited to a subset of the eagle-i institutions (Dartmouth University, University of Pennsylvania, Harvard University and Howard University), although it is straightforward to expand it to include all institutions.

When a new version becomes available, it is downloaded by our service. The simplest idea would be to maintain a complete copy of each version; when a citation is dereferenced, the appropriate version would be retrieved and the data for the eagle-i resource accessed. However, although citation dereferencing can be done very quickly using this approach, it is very space inefficient.

Our eagle-iV service therefore maintains a single version of an RDF dataset as a database table, and tracks changes in terms of the RDF triples that are inserted or deleted between consecutive versions. This reduces storage space, as it only captures the changes between each version, with the potential for increasing the cost of dereferencing a citation since the log of all changes to the resource must be examined to get the correct version. Our experience, however, is that eagle-i changes very slowly and therefore the deferencing cost is negligible.

Furthermore, since eagle-iV maintains a timestamp for each recorded change, we can answer time-based queries, such as *what triples were added or deleted in the period between* t_1 *and* t_2*? What was the object of triple X at time* t_1*?* And *when was triple Y first added (deleted)?* Supporting such time queries is crucial for understanding how datasets change.

An overview of our approach is shown in Figure 3, which depicts the process of versioning two RDF triples from the eagle-i dataset. When we version the dataset for the first time, eagle-iV creates a database table (*triple store*), which contains all the unique RDF triples in the dataset. For each triple, eagle-iV also creates a log record in the log table (*triple log*), which is used to track revisions. For each new version downloaded, our service checks if there have been any changes in the triples. It does so by comparing the downloaded version with the previously recorded version, and then recording any added or deleted triples in the *triple log* table.

We discuss our versioning approach in more detail by using a simplified example that depicts the versioning of the following two triples:

```
<eagle-id: e2> <Developer> <Grant, G.>
<ont: t1> <rdfs:label> <“Software”>
```

Versioning e2—First, eagle-iV searches for e2 in the *triple store* table (1). Since e2 does not exist, eagle-iV inserts it into the *triple store* table (2), and inserts a new log record for e2 (using its generated ID) into the *triple log* table (3). Because e2 is being versioned for the first time, eagle-iV assigns today’s date (12/31/2016) to both the start and the end version date (*version start date* and *version end date*) of e2’s log record.

Versioning t1—eagle-iV first checks if t1 exists in the *triple store* table (1). Because t1 matches a triple with the ID 2, eagle-iV then searches the *triple log* table for t1’s most up-to-date log record (2). It does so by searching the *triple log* table for the record with ID 2 that has the maximum end version date. eagle-iV then checks (3) if the end version date of t1’s most up-to-date log record is equal to the last version date (12/30/2016). If so (4), it updates

the triple's *version end date* to today's date (12/31/2016), i.e., t1 has not been changed since the last version. Otherwise, our service would have inserted a new log record for triple t1 into the log table, indicating that triple t1 has been previously deleted and is now being reinserted.

3.2 Citation Dereferencing

To dereference an eagle-i citation, the eagle-i id and date of access are used to retrieve the RDF instance as of the time the resource was cited. Figure 4 depicts the approach, which consists of three stages: 1) Parse Citation, 2) Identify Resource Triples, and 3) Retrieve Versioned Data. We discuss our approach in more detail using the following example, which is a simplified version of the citation presented in Section 2.1.2:

```
{eagle-id: "eagle-id: e1", db-version: "V12",
  name: "Significance Tester",
  developers: {"Manduchi, E."},
  url: "http://www.cbil.upenn.edu/STAR/"}
```

First, "Parse Citation" extracts the information needed to retrieve the RDF data of the cited resource, including its eagle-i id (*e1*) and access timestamp (10/15/2016). Second, "Identify Resource Triples" uses the eagle-i id to query the *triple store* table for the set of RDF triple IDs (TIDs: {1, 2, 3}) that are part of *e1* (i.e., RDF triples with eagle-i id *e1*). Third, "Retrieve Versioned Data" checks the *triple log* table to determine which of the RDF triples were active when the citation was generated (10/15/2016). Note that only triple 1 and 2 were active on 10/15/2016; triple 3 was added on 10/31/2016. Finally, "Retrieve Versioned Data" returns triples 1 and 2 from *triple store* table. Notice that we have omitted the url triple from this example for the sake of simplicity.

3.3 Performance Considerations

To understand how eagle-i changes, we have been downloading the working version of four eagle-i institutional datasets – Dartmouth, Penn, Harvard and Howard – on a daily basis since August, 4th 2016. During this period, we observed that only two datasets were modified: Harvard and Dartmouth. We also observed that these changes were minor. For instance, only four RDF triples, each representing a cell line, were updated in Dartmouth dataset. Based on these observations, it is clear that versioning on a daily basis is not efficient. The question becomes: How often should eagle-iV version datasets, and how can our service perform versioning more efficiently for citation purposes?

Since the primary use of versioning is to dereference citations, we can perform fine-grained versioning that 1) is triggered when a user cites an eagle-i resource; and 2) only records changes on the resource to be cited. By applying this technique (inspired by work by [27]), we can further reduce the space required to store deltas between consecutive versions: If a version of a resource is not cited, it does not have to be stored. As a result of reducing the set of triples to be versioned, dereferencing performance can also be significantly improved.

However, time-based queries will only reflect changes with respect to citations rather than all changes. We plan to implement this technique in the next version of eagle-iV.

In our implementation, we also create a separate *triple store* table for each institution (e.g. *Penn triple store*, *Harvard triple store*, etc.) to improve dereferencing performance.

4. FORMAT

One goal in automating citations for eagle-i is to structure them so that they can be formatted according to predefined styles – e.g., BibTex [7], RIS [8], or XML – and included in standard bibliography management tools.

Several metadata formats for data citations [5,18,33] share a common subset of elements [8]: `author`, `publication date`, `title`, `edition`, `version`, `URI`, `resource type`, `publisher`, `unique number fingerprint` (a form of hash of the data), a persistent `URL`and `location`. DataCite [2], the most recent and widely recognized metadata format proposal for citing data, expands this common set of fields by adding other ones, such as `subject`, `contributor`, `format`, `size`, `description`, `language`, `rights` and `funding reference`.

While DataCite is a good fit for many datasets, there is no easy translation between the eagle-i fields and those in the DataCite format. As shown in Table 2 there are some eagle-i fields that find no match in the DataCite format – i.e., `usedBy`, `inventoryNumber`, `researchProvider` and `serviceProvider`. Other eagle-i fields, such as `developerName`, `manufacturerName`, `performedBy` and `author`, find a match to a single DataCite field (`contributor`). This is a problem because by mapping to a single field we lose the individual semantics of these fields, and thereby the different nuances of contributions in eagle-i.

This problem is shared by other datasets, such as the Earth Science Information Partners (ESIP)[9]: There is no match in the DataCite format for the `access date and time` field, and `archive` and `distributor` are both mapped into the `publisher` field. There have also been several requests to extend DataCite to accommodate software version as an attribute of the `format` field [10] or to provide ad-hoc fields for describing data catalogs [11]. These requests are hard to meet without enclosing metadata fields to address specific dataset needs that may be too narrow for a general-purpose metadata schema such as DataCite.

It is evident that one size does not fit all when it comes to metadata formats for data citation. Metadata formats for “traditional” resources in the digital library domain such as the well-known Dublin Core [12] adopted a dynamic solution that we think could be a viable possibility also in the data citation context. Indeed, the Dublin Core metadata standard is

[7] http://www.bibtex.org/
[8] http://referencemanager.com/sites/rm/files/m/direct_export_ris.pdf
[9] http://wiki.esipfed.org/index.php/Interagency_Data_Stewardship/Citations/provider_guidelines#Detailed_Citation_Content
[10] https://groups.google.com/forum/#!topic/datacite-metadata/7y2Zbzr0YTY
[11] https://groups.google.com/forum/#!topic/datacite-metadata/nHhFue3FnoU
[12] http://www.dublincore.org/

composed of 15 base elements – i.e., the Simple Dublin Core – that can be extended by using the so-called "application profiles" that adapt the standard to the requirements of specific application domains. Note, however, that an application profile is not only a set of extra metadata fields, but also consists of policies and guidelines defined for a particular application, implementation, or object type. Some examples of this are the Dublin Core Application Profile for ScholarlyWorks designed by [4] and the MountainWest Digital Library Dublin Core developed by the Utah Academic Library Consortium [36]. A viable solution for data citation that would also work well for eagle-i is to use the DataCite metadata format as a base reference and extend it from time to time on the basis of specific dataset requirements using the idea of Dublin Core application profiles.

It is also not surprising that there is no easy translation between the eagle-i nomenclature and fields in BibTex and RIS, since these formats were developed for more traditional publications such as books and journals (see Table 2). There are two obvious solutions to this problem: i) encourage the owners of databases who wish them to be cited to conform to the developed standards whenever possible; and ii) facilitate extensions to existing standards. For example, in BibTex it is possible (with some work) to add new fields and define new entry types so that the citation is properly rendered accordingly to the specific application profiles adopted as citation format. However, currently the overhead of doing this would discourage many database owners from doing so and a community effort in this direction would therefore be desirable.

5. RELATED WORK

Core principles

Two major international initiatives have focused on defining the core principles for data citation: CODATA (the International Council for Science: Committee on Data for Science and Technology), which published a report on data citation principles [1]; and FORCE 11 (The Future of Research Communications and e-Scholarship) which published a list of principles synthesizing the ideas of a number of working groups [15].

In addition to highlighting the idea that data is a research object that should be citable, giving credit to data creators and curators, these principles state a number of criteria that a citation should guarantee:

- the *identification and access* to the cited data;
- the *persistence* of data identifiers as well as related metadata (i.e. the notion of *fixity*);
- the *completeness* of the reference, meaning that a data citation should contain all the necessary information to interpret and understand the data even beyond the lifespan of the data they describe;
- the *interoperability* of citations, meaning that they should be usable both by humans and machines.

Several organizations have also proposed standards for data citation, specifying snippets of information that should be included [6,11]; in particular, DataCite provides an XML metadata schema [12].

Computational solutions

Computational solutions for data citation often rely on persistent identifiers such as Digital Object Identifiers (DOI), Persistent Uniform Resource Locator (PURL) and the Archival Resource Key (ARK) [22, 32]. While persistent identifiers enable the data to be located and have associated guarantees of persistence (fixity), they do not constitute a full-fledged solution for data citation. In particular, they do not include snippets of information that are useful for human understanding (such as author/contributor/curator), nor do they address the issue of how to manage the variable granularity of data to be cited. For this reason, the use of persistent identifiers in the context of publishing research data is intended to provide a handle for subsequent citation purposes rather than being the data citation solution itself [22, 25].

Three main proposals target the problem of citing eXtensible Markup Language (XML) data at different levels of granularity. The first is a *rule-based citation system* that exploits the hierarchical structure of XML to provide humanand machine-readable citations to XML elements [10]. The second solution uses the idea of *database views* to define citable units as a key to specifying and generating citation of XML elements [9]; this solution was then extended to handle relational databases in [13]. The third uses a *machine learning approach* that learns a model from a training set of existing citations to generate citations for previously unseen XML elements [31]. None of these solutions can be straightforwardly adopted for citing Resource Description Framework (RDF) data.

For citing RDF datasets, there are two main contributions. The first proposes a *nano-publication model* where a single statement (expressed as an RDF triple) is made citable in its own right; the idea is to enrich a statement via annotations adding context information such as time, authority and provenance [19]. The model does not specifically address how to cite RDF sub-graphs with variable granularity and the automatic creation of citation snippets. The second defines a methodology based on *named metagraphs* to cite RDF sub-graphs, and proposes an approach to create human-readable and machine-actionable data citations [30]. Although the approach addresses the variable granularity problem, the snippets of information desired for a citation are not automatically selected. These solutions only marginally satisfy the completeness requirement mentioned above. None of the solutions mentioned above explicitly deal with the issue of fixity.

In contrast, the approach of [27] deals with both identification and fixity, and shows an implementation in the context of relational databases. In this approach, a query against the database returns a result set as well as a stable identifier; the identifier includes the version number of the database when queried, and serves as a proxy for the data to be cited. The database is versioned, so that when the stable identifier is later used (dereferenced) the data can be recovered as of the query time rather than the current version.

Versioning RDF

[21] proposes a relational database-based version management framework for RDF stores. To reduce space, they store the original version and the deltas between two consecutive versions in a database. They also propose *aggregated deltas*, a compression technique for deltas that creates a logical version directly rather than executing a sequence of deltas. We borrow Im's idea of reducing versioning storage overhead by maintaining one original version and the deltas between consecutive versions.

[16] introduces an RDF-based approach that supports versioning of RDF datasets and blank nodes, and uses an RDF vocabulary to describe changes to an RDF dataset. For each SPARQL update query, this system creates a patch containing the added and deleted triples and their corresponding RDF graphs. We also record changes as sets of added and deleted RDF triples, but store the information in a relational database.

Papavasileiou et al. [26] propose a methodology for handling change management for RDFS data maintained by large communities. They define a formal language of change for RDFS knowledge bases, and develop a change detection and application algorithm based on this language of change. Papavasileiou's work focuses on detecting high-level (i.e, human readable) changes from low (RDF triple) level deltas.

SemVersion [35] is an RDF-based approach, inspired by version control system (CVS). It provides structural and semantic versioning for RDF and RDF-based ontology languages (e.g., RDFS and OWL), and provides a hierarchical data model for versioning data.

R43ples [17] is another RDF-based approach that uses named graphs to semantically store groups of addition and deletion deltas between revisions. A version is restored by reconstructing a graph from the head revision, then undoing changes stored in the addition and deletion graphs. However, both R&Wbase and R43ples do not support time queries, whereas our versioning service supports them, as we discussed above.

6. TOWARDS A GENERIC CITATION FRAMEWORK

Although the citation framework described in this paper is specific to RDF datasets, in future work we plan to develop a generic citation framework and test it on citable databases using different models (in particular IUPHAR/BPS, which is a relational database). An overview of this framework is shown in Figure 5.

In our proposed framework, the database administrator (DBA) must first specify citation views and policies for how they are to be used in constructing citations for general queries such as joint- and alternate-use policies. When a user in the role of *Author* issues a query in the host database language (e.g. SQL, XQuery, or SPARQL), the query is translated into Datalog and rewritten using the (citation) view queries by means of the "Query Rewriting" component. Joint- and alternate-use policies are then used by the "Citation Generator" component to determine which citation views are to be used and how they are to be combined. The chosen citation queries are then executed in the host database language to retrieve the appropriate snippets of information, and the information combined using the

specified policy. The citation is then returned to the Author along with the data. When a user in the role of *Reader* later dereferences the citation, the "Citation Dereferencing" module parses the citation and extract the information required to re-issue the original query to the database in order to obtain the originally cited data.

To use the citation framework within a database instance, the database administrator (DBA) must do the following:

1. *Understand what information must be captured in the database to populate the citations.* In the case of eagle-i, this is done by requesting the information on the data-entry screens used to register a resource; in the case of IUPHAR/BPS, additional relations are added to the schema to capture information about the people responsible for certain subsets of information. There may also be additional *meta-data* required to construct the citation, such as the version number (or timestamp) at which the data was retrieved, or a reference to the standard literature for the database as a whole.
2. *Specify the citation views for the database.* Frequently, web-page views of the database represent the commonly executed queries and form the basis for the citation views. However, usage of the database may change over time, and the DBA may want to specify additional views in response to those changes.
3. *Specify joint- and alternate-use policies.* Although defaults (such as union) can be used, the DBA may wish to specify an ordering relationship over the views to define the "best" view for a query. For example, in IUPHAR/BPS the web page views of the underlying relational database are hierarchically organized, which imposes a natural ordering over the views: a child page view is more specific than a parent page.
4. *Ensure that the system is versioned and enable dereferencing.* Several approaches could be used to enable dereferencing: One is to assign an identifier to the author's query, store it, and attach the query id to the citation as proposed by [28]; another is to attach the query to the citation.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we present a citation framework developed for the eagle-i dataset. The architecture of this system is shown in Figure 1. When viewing an eagle-i resource, a user may click on the "Cite this resource button". As described in Section 2, rather than merely returning the eagle-i id for the resource, the system will use the id to first determine the type of the resource (i.e. which citation view to use) and then use the associated citation query to retrieve the appropriate snippets of information to construct the citation for that type of resource. The citation can then be served up in human-readable, RIS, XML or BibTex format. At a later point in time, a user may dereference the citation to retrieve the RDF instance for that resource as of the time that it was cited, as described in Section 3.

The citation model developed for eagle-i is *generic*, and could be used for other citable databases using a variety of different data models. This is enabled since Datalog captures the

core of common query languages for relational, graph-based, and semi-structured data models: A wrapper can be written for each model which translates a (citation) query written in Datalog into a query in the language for the model (e.g. SPARQL for RDF or SQL for relational). The wrapper then translates the query result into the format required as input to the citation function (e.g. JSON). Note that since Datalog is not very user-friendly, it can be used as the *intermediate language* for a graphical interface for specifying citation views.

The citation framework is also *extensible* beyond the fixed set of web-form views of the citable databases. For example, in IUPHAR/BPS citations for web-page views of the underlying relational database are well understood, and can be easily encoded using citation views. However, in the future the owners of the database would like to enable general queries over the relational database and be able to automatically serve up citation. As discussed in Section 2.3, using *query rewriting* we can rewrite a general query Q in terms of set of the view queries $V_1, \ldots, V_n$, and use their associated citation queries to construct a citation for Q. Note that the policies for how to combine the citation views used in the rewriting(s) must be specified by the database owner. We have recently explored this idea in the context of relational databases [13], and our preliminary results seem promising.

An important side effect of the citation process is that the *impact* of different components of the database can be automatically measured. In the future, we would like to explore how to use these impact measurements to enable fine-grained scientometrics. We would also like to work with owners of citable databases and developers of standards to ensure that appropriate information is included in the database schemas, and that the formats are flexible enough to capture the desired information.

Acknowledgments

The authors would like to thank Greg Grant and Faith Coldren from the eagle-i team for many fruitful discussions related to this work. This work has been partially funded by NSF IIS 1302212, NSF ACI 1547360, and NIH 3-U01-EB-020954-02S1.

References

1. Out of Cite, Out of Mind: The Current State of Practice, Polocy, and Technology for the Citation of Data. Vol. 12. CODATA-ICSTI Task Group on Data Citation Standards and Practices; Sep. 2013
2. Technical Report. DataCite Metadata Working Group; 2016. DataCite Metadata Schema Documentation for the Publication and Citation of Research Data, Version 4.0.
3. Abiteboul, S., Hull, R., Vianu, V. Foundations of Databases. Addison-Wesley; 1995.
4. Allinson J, Johnston P, Powell A. A Dublin Core Application Profile for Scholarly Works. Ariadne. 2007
5. Altman M, King G. A Proposed Standard for the Scholarly Citation of Quantitative Data. D-Lib Magazine. 2007; 13(3/4)
6. American Meteorological Society. [accessed Nov 2016] Data archiving and citation. http://www2.ametsoc.org/ams/index.cfm/publications/authors/journal-and-bams-authors/journal-and-bams-authors-guide/data-archiving-and-citation/
7. Angles, R., Gutierrez, C. The Expressive Power of SPARQL. Proceedings of the 7th International Semantic Web Conference (ISWC); 2008. p. 114-129.
8. Ball, A., Duke, M. Technical Report. Edinburgh: Digital Curation Centre; 2015. How to Cite Datasets and Link to Publications.

9. Buneman P, Davidson SB, Frew J. Why data citation is a computational problem. Communications of the ACM (CACM). 2016; 59(9):50–57.
10. Buneman P, Silvello G. A Rule-Based Citation System for Structured and Evolving Datasets. IEEE Data Eng Bull. 2010; 33(3):33–41.
11. Data Observation Network for Earth (DataONE). [accessed Nov 2016] Data citation and attribution. https://www.dataone.org/citing-dataone
12. DataCite. [accessed Nov 2016] DataCite metadata schema for the publication and citation of research data. Oct. 2014 http://schema.datacite.org/meta/kernel-3/doc/DataCite-MetadataKernel_v3.1.pdf
13. Davidson SB, Deutsch D, Milo T, Silvello G. A model for fine-grained data citation. The biennial Conference on Innovative Data Systems Research (CIDR 2017). 2017 accepted for publication.
14. Deutsch A, Popa L, Tannen V. Query reformulation with constraints. SIGMOD Record. 2006; 35(1):65–73.
15. FORCE-11. Data Citation Synthesis Group: Joint Declaration of Data Citation Principles. FORCE11; San Diego, CA, USA: 2014.
16. Frommhold, M., Piris, RN., Arndt, N., Tramp, S., Petersen, N., Martin, M. Towards Versioning of Arbitrary RDF Data. 12th International Conference on Semantic Systems Proceedings; 2016.
17. Graube M, Hensel S, Urbas L. R43ples: Revisions for triples an approach for version control in the semantic web. CEUR Workshop Proceedings. 2014; 1215
18. Green, T. Technical report. OECD Publishing; 2010. We need publishing standards for datasets and data tables.
19. Groth P, Gibson A, Velterop J. The Anatomy of a Nanopublication. Inf Serv Use. 2010; 30(1–2): 51–56.
20. Halevy AY. Answering queries using views: A survey. VLDB J. 2001; 10(4):270–294.
21. Im D-H, S-WL, H-JK. A version management framework for rdf triple stores. International Journal of Software Engineering and Knowledge Engineering. 2012; 22(01):85–106.
22. Klump J, Huber R, Diepenbroek M. DOI for Geoscience Data – How Early Practices Shape Present Perceptions. Earth Science Informatics. 2015:1–14.
23. Lenzerini, M. Data integration: A theoretical perspective. Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '02; New York, NY, USA: ACM Press; 2002. p. 233-246.
24. Meinhardt, P., Knuth, M., Sack, H. Tailr: A platform for preserving history on the web of data. Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS '15; New York, NY, USA. ACM; 2015. p. 57-64.
25. Mooney H, Newton MP. The Anatomy of a Data Citation: Discovery, Reuse, and Credit. Journal of Librarianship and Scholarly Communication. 2012; 1(1)
26. Papavasileiou V, Flouris G, Fundulaki I, Kotzinos D, Christophides V. High-level Change Detection in RDF(S) KBs. ACM Trans Database Syst. Apr; 2013 38(1):1:1–1:42.
27. Pröll, S., Rauber, A. Scalable data citation in dynamic, large databases: Model and reference implementation. Proceedings of the 2013 IEEE International Conference on Big Data; 6–9 October 2013; Santa Clara, CA, USA. 2013. p. 307-312.
28. Rauber A, Ari A, van Uytvanck D, Pröll S. Identification of Reproducible Subsets for Data Citation, Sharing and Re-Use. Bulletin of IEEE Technical Committee on Digital Libraries, SpecialIssue on Data Citation. May; 2016 12(1):6–15.
29. Sande MV, Colpaert P, Verborgh R, Coppens S, Mannens E, Walle RVD. R&Wbase: Git for triples. Proceedings of the WWW2013 Workshop on Linked Data on the Web. 2013:1–5.
30. Silvello G. A Methodology for Citing Linked Open Data Subsets. D-Lib Magazine. 2015; 21(1/2)
31. Silvello G. Learning to Cite Framework: How to Automatically Construct Citations for Hierarchical Data. Journal of the American Society for Information Science and Technology (JASIST). 2016:1–28. in print.
32. Simons N. Implementing DOIs for Research Data. D-Lib Magazine. 2012; 18(5/6)
33. Starr J, Gastl A. isCitedBy: A metadata scheme for DataCite. D-Lib Magazine. 2011; 17(1/2)
34. Torniai C, Bourges-Waldegg D, Hoffmann S. Semantic Web. 2015; 6:139–146.

35. Völkel M, Groza T. Semversion: An rdf-based ontology versioning system. Proceedings of the IADIS international conference WWW/Internet. 2006; 2006(44)

36. Walters, CD. Technical report. Utah Academic Library Consortium. Digitization Committee. Metadata Task Force; 2010. Mountain west digital library dublin core application profile.

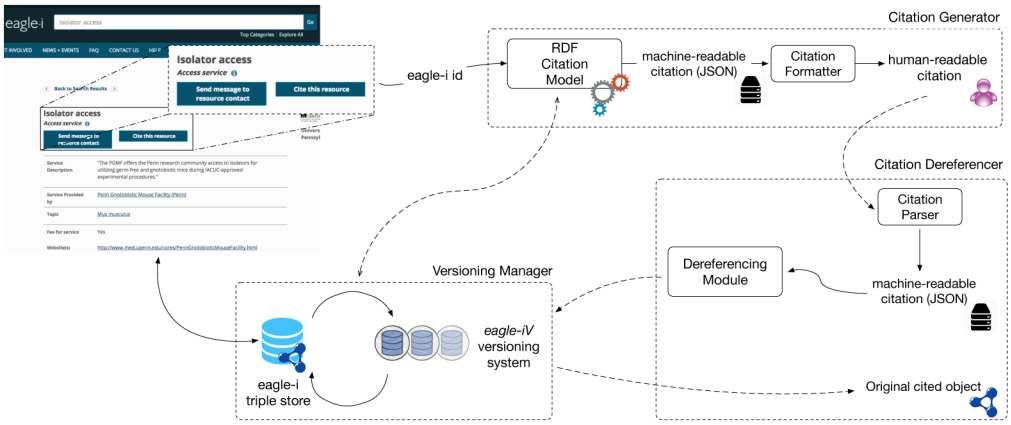

Figure 1.
Citation framework for eagle-i.

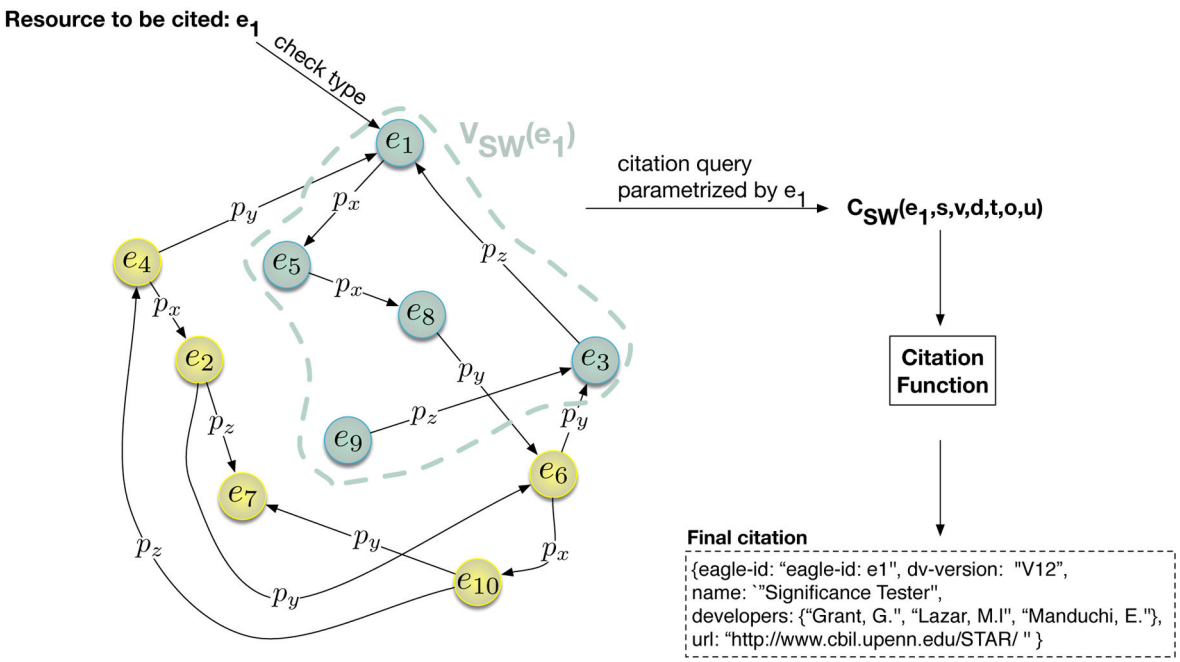

Figure 2.

Components of the citation model and example of citation for resource e_1.

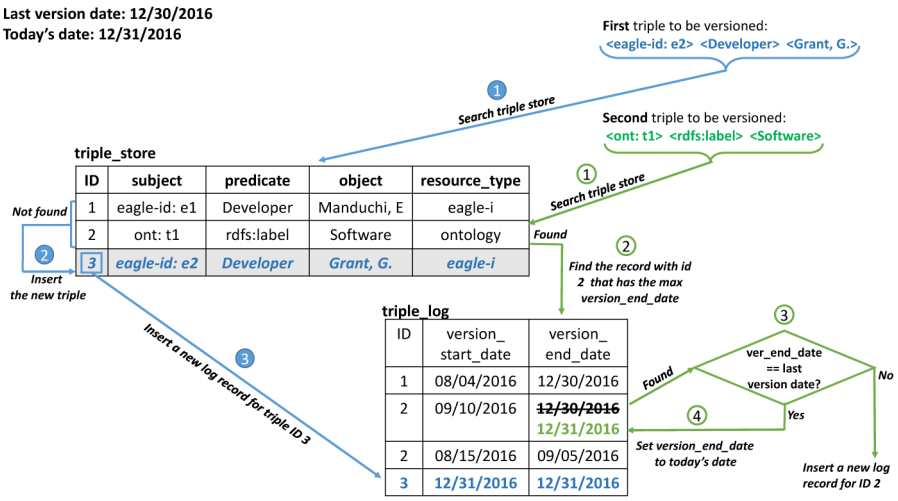

Figure 3.
An example of versioning two RDF triples using eagle-iV

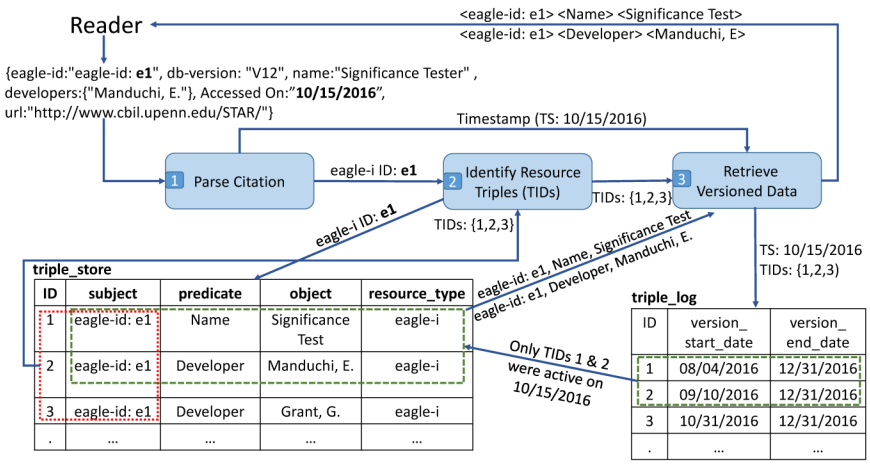

Figure 4.
An example of dereferencing the citation of the "Significance Tester" eagle-i resource (e1)

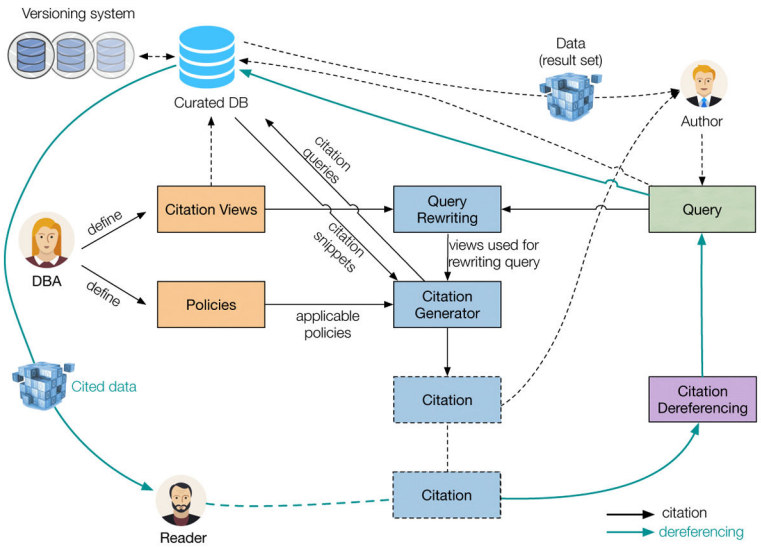

Figure 5.
Overview of general citation framework.

Table 1

Result of C_{SW}

r	s	v	d	t	o	u	y
eagle-id: e_1	Significance Tester		Grant, G		UPenn	http://www.cbil.upenn.edu/STAR/	V12
eagle-id: e_1	Significance Tester		Lazar, M.l		UPenn	http://www.cbil.upenn.edu/STAR/	V12
eagle-id: e_1	Significance Tester		Manduchi, E.		UPenn	http://www.cbil.upenn.edu/STAR/	V12

Proc ACM/IEEE Joint Conf Digit Libr. Author manuscript; available in PMC 2018 July 27.

Table 2

Mappings between eagle-i, BibTex, RIS and DataCite

eagle-i	BibTex	RIS	DataCite
developerName	author	AU	contributor
manufacturerName	author	AU	contributor
usedBy	organization	DP	?
name	title	TI	title(s)
version	version	M2?	version
URL	how published	UR	related Identifier
location	organization	DP	geo Location
resource Type	type	M3?	resource Type
performedBy	author	AU	contributor
author	author	AU	contributor
inventoryNumber	version?	M1?	?
researchProvider	organization	AU	?
serviceProvider	organization	AU	?
eagle-i id	URL	ID	identifier