Precisely and Persistently Identifying and Citing Arbitrary Subsets of Dynamic Data

Andreas Rauber[1], Bernhard Gößwein[1,3], Carlo Maria Zwölf[4], Chris Schubert[5], Florina Wörister[1], James Duncan[6], Katharina Flicker[1], Koji Zettsu[7], Kristof Meixner[1], Leslie D. McIntosh[8], Reyna Jenkyns[9], Stefan Pröll[10], Tomasz Miksa[1,11] Mark A. Parsons[2]

1 TU Wien, Vienna, Austria
2 University of Alabama in Huntsville, AL, USA
3 Earth Observation Data Centre, Vienna, Austria
4 LERMA, Observatoire de Paris, PSL Research University, CNRS, Sorbonne University, UPMC Univ Paris, Meudon, France
5 Climate Change Centre Austria, Vienna, Austria
6 Forest Ecosystem Monitoring Cooperative, University of Vermont, Burlington, VT, USA
7 National Institute of Information and Communications Technology, Tokyo, Japan
8 Ripeta, Saint Louis, MO, USA
9 Ocean Networks Canada, University of Victoria, Victoria, BC, Canada
10 Cropster, Innsbruck, Austria
11 SBA Research, Austria

Abstract

Precisely identifying arbitrary subsets of data so that these can be reproduced is a daunting challenge in data-driven science, the more so if the underlying data source is dynamically evolving. Yet, an increasing number of settings exhibit exactly those characteristics: larger amounts of data being continuously ingested from a range of sources (be it sensor values, [online] questionnaires, documents, etc.), with error correction and quality improvement processes adding to the dynamics. Yet, for studies to be reproducible, for decision-making to be transparent, and for meta studies to be performed conveniently, having a precise identification mechanism to reference, retrieve, and work with such data is essential. The Research Data Alliance (RDA) Working Group on Dynamic Data Citation has published 14 recommendations that are centered around time-stamping and versioning evolving data sources and identifying subsets dynamically via persistent identifiers that are assigned to the queries selecting the respective subsets. These principles are generic and work for virtually any kind of data. In the past few years numerous repositories around the globe have implemented these recommendations and deployed solutions. We provide an overview of the recommendations, reference implementations, and pilot systems deployed and then analyze lessons learned from these implementations. This article provides a basis for institutions and data stewards considering adding this functionality to their data systems.

1 Introduction

Accountability and transparency in automated decisions (ACM US Public Policy Council, 2017) have important implications on the way we perform studies, analyze data, and prepare the basis for data-driven decision making. Specifically, reproducibility in various forms, that is, the ability to recompute analyses and arrive at the same conclusions or insights is gaining importance. This has impact on the way analyses are being performed, requiring processes to be documented and code to be shared. More critically, data–being the basis of such analyses and thus likely the most relevant ingredienable in any data-driven, decision-making process–needs to be findable and accessible if any result is to be verified. Yet, identifying precisely which data were used in a specific analysis is a nontrivial challenge in most settings: Rather than relying on static, archived data collected and frozen in time for analysis, today's decision-making processes rely increasingly on continuous data streams that should be available and usable on a continuous basis. Working on last year's (or last week's) data is not an acceptable alternative in many settings. Data undergo complex preprocessing routines, are recalibrated, and data quality is continually improved by correcting error. Thus, data are often in a constant state of flux.

Additionally, data are getting 'big': Enormous volumes of data are being collected, of which specific subsets are selected for analysis, be they a small number of individual values to massive subsets of even bigger data sets. Describing which subset was actually being used–and trying to re-create the exact same subset later based on that description–may constitute a daunting challenge due to the complexity of subset selection processes (such

as marking an area on an image) and the ambiguities of natural language (e.g., do the measurements in the time period from January 7 to June 12 include or exclude the respective start and end dates?).

At the same time, expectations of data provisioning are increasing:

- Data should be timely, that is, available for analysis immediately after they have been collected and quality-controlled, not at weekly, quarterly, or annual intervals.
- Data management and storage should be efficient, avoiding storing overlapping duplicates of increasingly large data subsets.
- Data identification and citation processes should be transparent to users.
- Solutions should be generic, working across different types of data, to be acceptable in an increasingly interdisciplinary work environment, and the need to combine and work with varying types of data in each domain.

The Research Data Alliance (RDA) Working Group on Dynamic Data Citation set off to tackle these challenges in March 2014. In September 2015 it released its recommendations in the form of a compact 2-page flyer (Rauber, Asmi, van Uytvanck, & Pröll, 2015) together with a more extensive report (Rauber, Asmi, van Uytvanck, & Pröll, 2016) providing some background information on the rationale for certain decisions. The recommendations specifically address principle 7 on Specificity and Verifiability of the Joint Declaration of Data Citation Principles (Data Citation Synthesis Group, 2014) and aim specifically at providing an automated, machine-provided, and machine-actionable solution in addition to being human-readable.

The recommendations center around the principle of ensuring evolving data is timestamped and versioned, with subsets being identified via time-stamped queries that are stored. By assigning a persistent identifier (PID, e.g., a Digital Object Identifier, DOI) to these queries they become resolvable and can be reexecuted transparently against the time-stamped database to re-create the exact same subset that was initially selected. This eliminates the need for predefined subsets that are frozen at predefined intervals, avoids data duplication, and is transparent to the researcher, while at the same time being applicable to virtually all types of data, such as databases, spreadsheets, collections of files, or an individual image. Similarly, queries can come in any form, ranging from SQL database queries to marking an area on an image.

After a one-year intensive evaluation, both conceptually, as well as via reference implementations and actual operational deployments, these recommendations were officially endorsed by the RDA in September 2016 and are currently listed to become a European Technical Specification. They are further being referred to in an increasing number of standards and reference guidelines such as ISO 690:2010, Information and documentation — Guidelines for bibliographic references and citations to information resources (International Organization for Standardization (ISO), 2010) and the Earth Science Information Partners (ESIP) Data Citation Guidelines for Earth Science Data Version 2 (ESIP Data Preservation and Stewardship Committee, 2019).

Given the experiences gained from several further implementations and deployments, as well as the discussions taking place at a variety of workshops and meetings of the Working Group, it seems timely to review the various approaches that different institutions have taken to implement a data identification and citation service, compiling the lessons learned in order to provide guidance and good-practice examples to other repositories wishing to offer such services to their customers.

The remainder of this article is structured as follows. Section 2 provides a brief review of the recommendations, followed by descriptions of several proof-of-concept reference implementations in Section 3 that show the feasibility of the approach in a range of different data settings. Section 4 describes several operational implementations of the recommendations from a range of disciplines (e.g., Medicine, Earth Sciences, Astrophysics), and a range of different data types (relational databases, NetCDF files, repositories of image data) that vary in size and data dynamics. Section 5 discusses the lessons learned and identifies open issues given the current state of the recommendations and their deployment. Finally, Section 6 summarizes the current state of affairs and provides an outlook onto future development.

2 RDA Recommendations on Dynamic Data Citation

In order to identify reproducible subsets for data citation, sharing and reuse of data 14 recommendations were formulated by the Working Group on Data Citation (WGDC) of the Research Data Alliance (RDA) (see Figure 1).

They are grouped in four areas (Rauber et al., 2015, 2016):

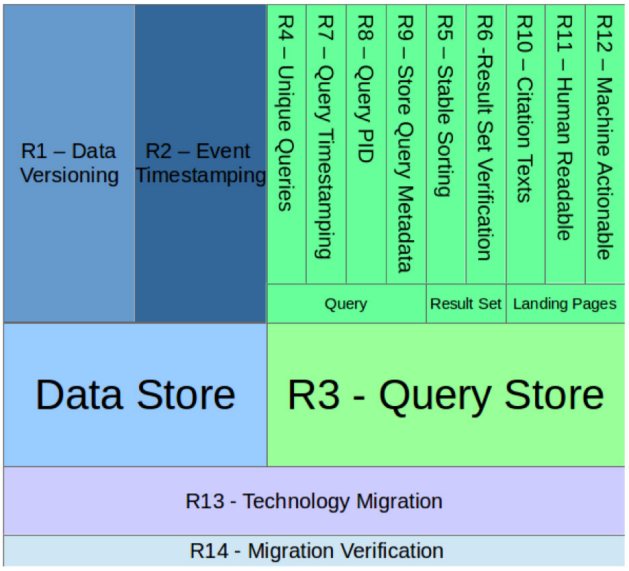

Figure 1: RDA WGDC Recommendations for Data Citation (Rauber et al., 2016).

- *Preparing the Data and the Query Store*
  - **R1 - Data Versioning**: Apply versioning to ensure earlier states of data sets can be retrieved.
  - **R2 - Timestamping**: Ensure that operations on data are timestamped, i.e., any additions, deletions are marked with a timestamp.
  - **R3 - Query Store Facilities**: Provide means for storing queries and the associated metadata in order to reexecute them in the future.
- *Persistently Identifying Specific Data Sets*
  - **R4 - Query Uniqueness**: Rewrite the query to a normalized form so that identical queries can be detected. Compute a checksum of the normalized query to efficiently detect identical queries.
  - **R5 - Stable Sorting**: Ensure that the sorting of the records in the data set is unambiguous and reproducible.
  - **R6 - Result Set Verification**: Compute fixity information (checksum) of the query result set to enable verification of the correctness of a result upon re-execution.
  - **R7 - Query Timestamping**: Assign a timestamp to the query based on the last update to the entire database (or the last update to the selection of data affected by the query or the query execution time). This allows retrieving the data as it existed at the time a user issued a query.
  - **R8 - Query PID**: Assign a new PID to the query if either the query is new or if the result set returned from an earlier identical query is different due to the changes in the data. Otherwise, return the existing PID.
  - **R9 - Store Query**: Store query and metadata (e.g., PID, original and normalized query, query and result set checksum, timestamp, super-set PID, data set description, and other) in the query store.
  - **R10 - Automated Citation Texts**: Generate citation texts in the format prevalent in the designated community for lowering the barrier for citing the data. Include the PID into the citation text snippet. For details see (Silvello, 2017).
- *Resolving PIDs and Retrieving the Data*
  - **R11 - Landing Page**: Make the PIDs resolve to a human readable landing page that provides the data (via query reexecution) and metadata, including a link to the super-set (PID of the data source) and citation text snippet.
  - **R12 - Machine Actionability**: Provide an API / machine actionable landing page to access metadata and data via query reexecution.

- *Upon modifications to the Data Infrastructure*
  - **R13 - Technology Migration**: When data is migrated to a new representation (e.g., new database system, a new schema or a completely different technology), migrate also the queries and associated fixity information.
  - **R14 - Migration Verification**: Verify successful data and query migration, ensuring that queries can be reexecuted correctly.

The recommendations are applicable for all types of data representation, be it individual images or comma-separated value (CSV) files, collections of files such as PDF documents, source code or NetCDF files, structured data in relational databases, semistructured data in XML files, linked data in triple stores, and more. The recommendations also apply for all kinds of data characteristics, be they massive repositories of satellite imagery or small collections of aggregated statistical information, highly dynamic data and continuous sensor data streams or entirely static data that never changes. The recommendations also work for all kinds of subsetting paradigms, be they classical SQL queries against a relational data base, SPARQL queries against an RDF endpoint, selecting one or more files from a repository or directory structure or identifying a subgraph in a network structure, marking an area on an image by drawing boundary lines, slicing and dicing multidimensional data files, or running specific scripts that select subsets of some form of data structure.

This approach allows a system to precisely identify any arbitrary subset of data without requiring any additional effort from the user because the actual subsetting and resolution process is completely transparent. The user need not know anything about the actual subsetting process to retrieve the subset. (In discussions the approach has been described as analogous to the transparency provided by file systems: when opening a file for reading / writing on common file systems, the identifier (toward the user front-end usually a combination of path and filename) is resolved and a query is passed to the hard disc to retrieve and assemble the file that is actually stored in a series of blocks distributed across a disc or, in the case of RAID system, even distributed across several discs. A file usually is not materialized as a contiguous sequence of bytes at a specific address, but assembled dynamically after retrieving a set of distributed blocks due to a query being processed by the file system.)

Reexecuting the query with the original timestamp against the time-stamped data source allows a user to retrieve exactly the data as it was initially selected. Additional safeguards such as checksums support verification of data, while automatically derived citation texts ease referencing data in publications.

However, the approach even extends beyond retrieving the data in its original state. Reexecuting the query against the current status of a data source (basically like using the current timestamp as part of the query), also allows the retrieval of the current status of the semantically identical data. It returns the semantically same subset of data including all additions and corrections made since that are covered by the query. Having the queries stored allows them to be reexecuted against the status of the data source at any given point in time, allowing to track the effects of data evolution.

Prior to releasing these recommendations, they were tested both conceptually and in practice via several proof-of-concept implementations apparently confirming their viability and versatility. It is only through actual deployment in fully operational settings, however, that the validity and practicality of the recommendations can truly be tested. In the following sections we will thus take a look at the most pertinent pilot implementations, analyze the concrete approaches that have been taken to implement the various elements, and analyze the lessons learned to provide guidance to other institutions wanting to deploy the same services.

Both the proof-of-concept and operational implementations present a broadly representative but not exhaustive set of different data types and disciplines. Note that the applicability of these principles is independent of the actual need for implementing the solution. For example, if data sets are entirely static, then obviously no versioning needs to be implemented. If, at any time later, a change to the data set does occur, then such a change should be versioned and time-stamped if reproducibility of results obtained from the preceding version of the data is required. Several of the pilot adopters chose to only implement a subset of the recommendations. While the basic principles always apply, adopters of these recommendations will always need to consider the trade-offs between the cost and value of implementing the solution, the efficiency of the implementation, and the functionality currently supported by the data repository. The following examples and discussion can inform those sorts of considerations. Table 1 provides an overview of current and ongoing implementations and indicates at which RDA Plenary they were being discussed. The corresponding slide decks are publicly available in the WGDC repository[1].

[1] `https://www.rd-alliance.org/node/141/file-repository`

Table 1: **Overview of adoption of the recommendations**

Standards and Reference Guidelines	RDA Plenary #
Joint Declaration of Data Citation Principles: Principle 7: Specificity and Verifiability	
ESIP:Data Citation Guidelines for Earth Science Data Vers. 2	P14
ISO 690, Information and documentation - Guidelines for bibliographic references and citations to information resources	P13
EC ICT TS5 Technical Specification (pending)	P12
Reference Implementations	
MySQL/PostgreSQL	P5, P6
CSV files: MySQL, Git	P5, P6, P8
XML	P5
CKAN Data Repository	P13
Adoptions deployed	
CBMI: Center for Biomedical Informatics, WUSTL	P8
FEMC: Forest Ecosystem Monitoring Cooperative	P8
CCCA: Climate Change Center Austria	P10, P11, P12
EODC: Earth Observation Data Center	P14
VAMDC: Virtual Atomic and Molecular Data Center	P8, P10, P12
NICT: Smart Data Platform	P10, P14
Deep Carbon Observatory	P12
Ocean Networks Canada	P12
In Progress	
Dendro System	P13

3 Proof-of-Concept Implementations

We developed four proof-of-concept reference implementations for different data storage scenarios. These cover (1) relational databases in a solution relying on MySQL (Pröll & Rauber, 2013; Pröll, Meixner, & Rauber, 2016), (2) a Git-based approach to deal with file-based data, for example, CSV files (Pröll et al., 2016), (3) two different approaches to XML databases (Huber, 2015), and (4) an approach integrating a NoSQL database into the CKAN repository system. We review these briefly to provide some context for the operational pilot deployments described thereafter, including pointers to source code.

3.1 Relational Databases

Relational database management systems (RDBMS) are a well-established and well-understood technology based on the strong theoretical foundation of set theory. The four fundamental operations known as CRUD allow basic data manipulation by offering functions for creating (INSERT), reading (SELECT), updating (UPDATE), and deleting (DELETE) records. These operations are essential for interacting with the structured data that is exposed as tables by database management systems.

RDBMS use transactions for ensuring the Atomicity, Consistency, Isolation, Durability (ACID) principle. While this powerful feature ensures that changes are visible to other transactions in a consistent way, it does not provide any temporal information that would allow reconstructing the state of a data set without additional metadata.

The SQL 2011 standard (International Organization for Standardization & International Electrotechnical Commission, 2011) introduces features for temporal data support, automatically creating history tables and adding time periods of record validity, thus taking care of versioning and time-stamping 'out-of-the-box.' As support for these temporal tables is still limited, we describe how a data citation mechanism can be implemented regardless of the SQL standard compliance and independent from vendor implementations (Pröll & Rauber, 2013).

To establish data citation as an accepted practice, the implementation of a data citation model should be

as convenient and noninvasive as possible, but often database queries are deeply coupled with the application code base or external tools. Thus, changing queries and interfaces is cumbersome and often not feasible. Here we describe the benefits and limitations of three approaches toward addressing these challenges and handling the metadata required for data citation: (1) integrated, (2) history table, and (3) hybrid.

Integrating the temporal data: To store the records in a versioned fashion, the existing tables need to be extended by temporal metadata columns, that is, two columns for *timestamp-added* and *timestamp-deleted* are added to each table. In order to maintain the uniqueness of the records, the primary key column needs to be extended by the *timestamp-added* column. This method is intrusive, because tables and application interfaces need to be adapted to insert and process the new timestamp columns. Also, the queries need to be adapted in a way that per default only the latest version of the records is obtained. This modification can have effects on the performance, as queries require filtering outdated data. Nevertheless, this method can be used in scenarios where deletions and updates occur rarely or where processing times of regular queries can easily be handled over the entire versioned data set. This approach requires a major redesign of the database, as well as the interfaces of programs accessing it. As an advantage, the retrieval of the earlier data state then requires a simple query that selects the latest version per record and omits deleted records. From a storage perspective, this approach only produces low overhead as records only append timing and versioning data.

Dedicated history table: The second approach implements a full history using a dedicated history table. Records that are inserted into the source table are immediately copied into the history table, which is extended by *timestamp-added* and *timestamp-deleted* columns. Deleted records are removed from the original table and only marked as deleted in the history table. An advantage of this approach is that the original tables remain unchanged, incurring no changes to their interfaces. There is also no impact on the database performance as the source table only store the live data. All requests for data citation are handled by the history table.

Hybrid: For the hybrid approach, a history table has to be added for all tables. The history table, extended again by *timestamp-added* and *timestamp-deleted* columns, is used for storing all records that are updated or deleted from the source table, thus keeping only the latest version in the source tables. The original table always reflects the latest version, whereas the history table records the evolution of the data.

The advantage of this approach is a minimal storage overhead, especially for append-only scenarios. A disadvantage is a more complex query structure for retrieving historical data because the system needs to check whether updates exist and then retrieve the records either from the original source or history tables. Yet, in settings where the reexecution of historic queries is a low-frequency event, this might be the preferred solution. This is thus also the solution specified in the SQL2011 standard and rolled out as temporal tables by several widely used database engines, including DB2, PostgreSQL, Teradata, Microsoft SQL Server, Oracle, and others.

A crucial aspect of result sets is their sorting. The results returned by a database query are commonly fed into subsequent processes in their order of appearance in the result set. In situations where the order of processing has some effect on the outcome (e.g., in machine learning processes), we need to ensure that consistent sorting of the result sets is provided. Hence, the order in which records are returned from the database needs to be maintained upon a query's reexecution. This is challenging as relational databases are inherently set based. According to the SQL standard, if no ORDER BY clause is specified, the sorting is implementation specific. Even if an ORDER BY clause is provided, the exact ordering may not be defined if there are several rows having the same values for the specified sorting criteria. For this reason, queries need to be rewritten to include a sort on the primary key prior to applying any user-defined sort. The reference implementation was developed for the MySQL database engine[2] but will work for any relational database management system supporting the SQL standard. The Posgres database engine natively features support for temporal tables.

3.2 File-based Data via Git

Many data sets are represented in a variety of file formats, including structured data stored, for example, in comma-separated value (CSV) files, various spreadsheet program formats, XML files, software source code, and questionnaires and papers represented in plain text or word processing formats. Specifically, in settings for long-tail data, the diversity of formats, sometimes combined with a low update frequency, makes it difficult to devise a data-type-specific solution as presented for relational databases above. In such settings, a variety of versioning systems such as Git[3] and Subversion (SVN)[4], versioning file systems (e.g., NILFS[5]), or the versioning

[2] `www.datacitation.eu`
[3] `https://git-scm.com/`
[4] `https://subversion.apache.org/`
[5] `https://nilfs.sourceforge.io/en/`

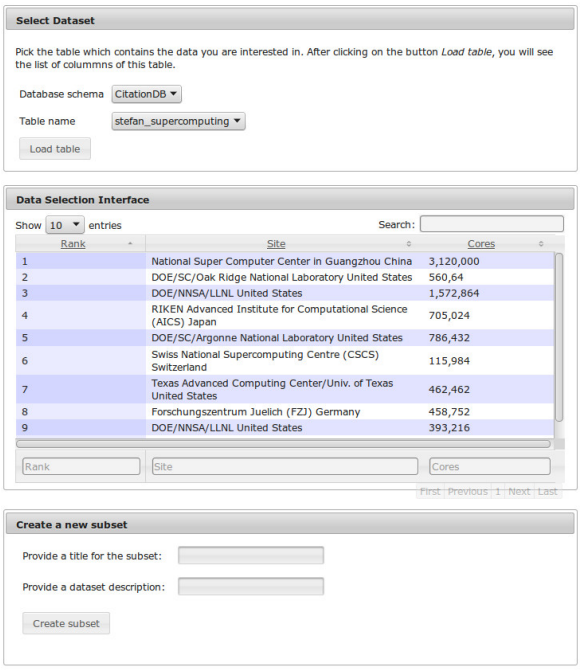

Figure 2: Interface of prototype for citable and reproducable data subsets from CSV files via Git versioning (Pröll et al., 2016)

.

features offered by cloud storage providers such as Amazon's S3[6] can be used to take care of versioning the files stored.

Depending on the access mechanisms used, queries can be issued in multiple ways against data stored as files. Frequently, dedicated scripts or access programs are being used to identify subsets, such as scripts identifying a certain time span in a video stream, or via dedicated libraries that allow interacting with CSV files as if they were relational databases supporting SQL queries (e.g., CSV2JDBC[7]). Any such scripts, representing the 'query language' of such a repository, can be maintained, allowing a low-effort implementation and deployment of the recommendations for, specifically, long-tail data settings. It is important to note that the implementation and deployment of the functionality supporting (subset) citation for (static or evolving) file-based data can be completely transparent to the user, making it a useful tool in many long-tail data settings (Pröll et al., 2016).

As a demonstrator, we present such an implementation supporting subset queries (or, of course, entire data set downloads) for CSV files. To realize this we use Git as a versioning system, and optionally add CSV2JDBC as a query interface to support subset queries.

Git is a distributed version control system for text-based artifacts, like source code. Extensions exist that allow it to be also used for binary data content such as media files, for example, GIT-LFS[8]. In the last decade, Git has evolved to the de-facto standard for managing source code, allowing for a variety of contribution processes (German, Adams, & Hassan, 2016).

This offers a simple solution to storing reproducible data sets within Git repositories. However, the repository has to ensure that the scripting language used to identify the subset is maintained. In addition to any script-like interfaces, support for SQL-like query languages can be provided as well. Since both CSV and SQL are based on a tabular view of data, fully harmonized CSV data can be easily mapped into relational database tables and accessed via library functions supporting SQL-like queries on CSV files. Thus we are able to provide support for subsetting using an SQL-like query language that can be executed on CSV files with the help of additional libraries like CSV2JDBC. These queries, in turn, can be created through the user interface providing faceted browsing functionalities: users can select the fields they want to filter by, provide selection values, and identify by which column they want their data to be sorted in ascending or descending order.

When a researcher wants to create an identifiable subset, the selected columns, the filter parameters, and

[6] `https://docs.aws.amazon.com/AmazonS3/latest/userguide/Versioning.html`
[7] http://csvjdbc.sourceforge.net/
[8] `https://git-lfs.github.com/`

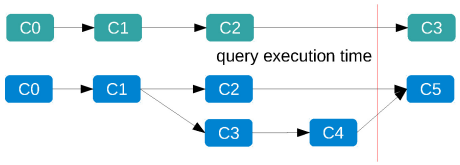

Figure 3: Citable and reproducible CSV data subsets using Git without and with branching (Pröll et al., 2016)

.

the sorting information are stored in the (again Git-based) query store as query metadata files together with the CSV file name and location and the execution timestamp. As each query metadata file has the unique PID as file name, the query can be reexecuted based on the versioned CSV data set.

Listing 1 provides a simple example for creating such a subset of CSV data using the mathematical software R[9] on the Top500[10] data set, which is updated periodically in the Git repository. Listing 2 shows the execution of the script in a Linux shell.

Listing 1: R Script for Subsetting

```
# Create a subset of the top 5 of the Top500 list
args <- commandArgs(trailingOnly = TRUE)
inputdata setPath=args[1]
outputSubset=args[2]
data set <- read.csv(inputdata setPath, header=TRUE)
subset <- subset(data set, Rank<=5,select=c(Rank,Site,Cores))
write.csv(subset, file=outputSubset)
```

Listing 2: Executing the Script

```
# Execute the R script and obtain a subset
# from the provided CSV file
/usr/bin/Rscript top5-subset.r /media/Data/Git-repository/supercomputing/supercomputer.csv \
/media/Data/Git-repository/supercomputing/supercomputer-top5.csv
```

Such scripts are versioned in Git and are used to retrieve the same data set by reexecuting the script. This is done by assigning a PID to each query, serving as an identifier in the Git repository, which is resolved during data retrieval. Revisions of the data set are committed to the repository, where Git stores a commit hash and the timestamp of the update. This first method is a simple way of storing reproducible data sets within Git repositories in a working environment where data keep a clear history.

An extension to the basic model presented above brings several advantages to collaborative work environments. It is based on the concept of branching, which is a simple and straightforward task in Git. Branching enables multiple researchers to work with different states of the data or files at the same time and is considered a 'best practice' in software development. Figure 3 shows a commit history in a repository with (a) a straight commit line in the upper part of the figure and (b) a repository with history where a developer created a branch base on commit *C1* and later on merged the commit into the main branch. The data or files can be merged later on with the main line or other branches again.

To retrieve the queries and reexecute them on the correct data set, the user provides the PID, which is hashed to get the file name of the query metadata file. Next, the demonstrator checks out the query branch and reads the query metadata file identified by the hashed PID. The demonstrator extracts and checks out the commit hash and file name of the CSV data file from the read query metadata. This restores the CSV data file from when the query was executed, and the system reexecutes the query script against the data file. The application demonstrator, shown in Figure 2, is available on Github[11].

3.3 XML Databases

A challenge when dealing with XML data is its inherent tree-like structure. Thus, (Huber, 2015) identified an approach to make XML data sets stored in native XML databases citable. This method also makes use of timestamps, versioning, PIDs, and query stores similar to the methods previously described.

[9] https://www.r-project.org/
[10] www.top500.org
[11] https://github.com/Mercynary/recitable

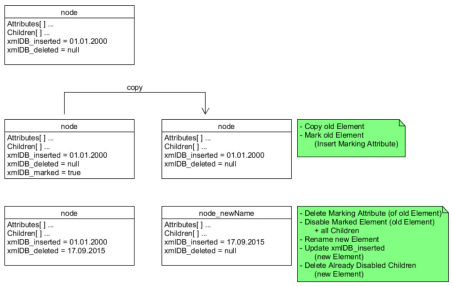

Figure 4: Process to version a *rename* operation in the branch-copy approach for XML data (Huber, 2015)

.

The reference implementation works with two different open source native XML database engines, namely, (1) BaseX[12], and eXist-db[13], both supporting XPath and XQuery but differing in their syntax and behavior. Furthermore, two different approaches to versioning and timestamping have been realized and evaluated, one relying on branch copying, the other on capturing the versions via parent-child relationships. Both approaches suffer from the complexity induced by the fact that a range of different scenarios need to be covered to support operations both on the hierarchy (element nodes) as well as attribute levels. The CRUD operations comprise of (1) three types of Insert (element into element, attribute into element, text into element), (2) Replace (element with element, attribute with attribute, text with text), (3) Replace Value (element with text, attribute with text, text with text), (4) Delete (element, attribute, tetx), and two types of (5) Rename (element as text, attribute as text).

In the first approach, each element node has two additional attributes capturing the add and delete timestamps, the latter being set to NULL upon insert. If a node is edited, the entire branch originating from that element is copied, with the original being marked with a deleted timestamp, and the copied branch being set active. Figure 4 shows an example of this process for the *rename* operation applied to an element. All other operations (inserts, deletes, and replacements) of both nodes and attributes are handled accordingly.

For the second approach, only *insert* timestamp attributes are added for the top-level node that is added to the hierarchy. *Delete* attributes are only added to the element node being deleted, marking the entire branch as invalid. Attributes and text nodes are versioned via versioning blocks, which are children of their respective element nodes. The existence of such a block indicates that edits happened to any of its elements.

While the first approach is conceptually simpler to implement, it obviously carries a massive storage overhead, especially when the XML data set represents deep hierarchies. Also, the query execution times are lower for the second approach, mostly due to the inefficiency of copying large branches in deep hierarchies, that make up for the more complex processing required for the individual updates, with eXist-db showing significantly lower performance than BaseX (9-15 sec for updates as opposed to 0.5 sec for flat hierarchies).

Another solution to archive, version, and query XML data is provided by XArch[14] (Müller, Buneman, & Koltsidas, 2008). This differs from the approach presented above in that it merges individual versions of XML files and uses a dedicated query language to identify subsets rather than using XPath and XQuery constructs. On the other hand, a performance evaluation demonstrates good performance and scalability due to the optimizations possible by deviating from a native XML storage infrastructure.

Each data set has its own query store, within which each query element represents a query having several child nodes storing metadata, such as the PID, execution timestamps, original and rewritten queries, MD5 hashes, and so on.

3.4 NoSQL-based Data Citation Support Added to CKAN

This reference implementation demonstrates a solution for NoSQL databases, in this case MongoDB[15], a popular open-source NoSQL database engine. It is integrated into CKAN, a prominent Open Source data portal platform[16], but also integrates links to a source code repository to connect and serve both code and data. By also integrating a handle service, it provides a full solution for supporting dynamic data citation within a data

[12] `http://basex.org`
[13] `http://exist-db.org`
[14] `http://sourceforge.net/projects/xarch`
[15] urlhttps://github.com/mongodb/mongo
[16] `https://ckan.org/`

portal. It consists of four independent applications, a CKAN instance, a CKAN datapusher service, a Source Code Repository, and a Handle registry service. Furthermore, there are four different storage containers. The CKAN instance requires a storage to store all uploaded files and a database to maintain application-specific data (e.g., credentials) and meta data of resources. These two containers are required by a default CKAN installation, the remaining two storage containers, data and query store, are specific to this solution.

The CKAN extension consists of two plugins, the *mongodatastore* and the *reclinecitationview* plugin. The first one is a novel DataStore implementation that aligns with the RDA Recommendations for Data Citation. The second plugin was needed to additionally add the citation feature to the view where data sets can be queried.

Data in MongoDB is organized in collections that consist of several JSON objects, which are referred to as documents. Using the terminology of relational databases, the document corresponds to one row of a database table and each attribute represents a column and its associated value. Every document has an *_id* attribute by default, which is of type ObjectId and maintained by the database. ObjectIds are 12-byte values consisting of a 4-byte timestamp, a 5-byte random value, and a 3-byte incrementing counter. In order to maintain older versions of a data set, the data store must not overwrite data records once they are updated. Therefore, updates are added as new documents to the collection, while the older version of the record remains unchanged. In other words, one document in the collection represents the state of a data record for a certain period of time. This requires another ID in order to define the relation between data records and its states. In order to define which point of time a certain document was valid, two additional attributes, *_created* and *_valid_to*, are added to the documents. The first attribute indicates when the document was added to the collection and the second attribute indicates until when it was valid. The *VersionedDataStoreController* encapsulates access to the MongoDB in a way that all changes (inserts, updates, and deletions) to the data set are tracked.

MongoDB queries are represented as JSON documents. Each attribute defines a filter that is applied on all documents. A query extension approach was chosen to restore a historic state of the data set on query-time. The idea is that for each query, conditions are added that only apply to the records that were valid at a certain point of interest. In order to detect semantically equal queries, they have to be transformed to a normalized form. As for the queries, the order of their attributes has no impact on the query result, they are sorted alphabetically by key before they are processed by the database.

In order to reexecute queries that were submitted to the data store in the past, all queries, including their metadata, are stored into a relational database. It contains all data that is required to reexecute a historic query and subsequently validate the result set, that is, hash values for the query, result set, and assigned record fields to identify query duplicates. As different scientific communities may require different information for citing data sets, the information on record fields is stored as key-value entries in a separate table.

The entire software package, including the plugins as well as test data and evaluation scripts, is available as OpenSource software on Github[17].

4 Pilot Adopters and Deployments

Any technology can ultimately only prove itself by adoption and use, leading to the creation of value for the user community. In this section we take a closer look at eight specific operational deployments and the variety of design decisions taken. These pilots are (1) the Center of Biomedical Informatics (CBMI) implementing the solution in an RDBMS setting with i2b2 as a repository for medical data; (2) the Virtual Atomic and Molecular Data Centre (VAMDC), which is a network of distributed repositories operating via an XML-based exchange protocol; (3) the Climate Change Centre Austria (CCCA), operating a repository of NetCDF files queried via a Thredds server; (4) the Forest Ecosystem Monitoring Cooperative (FEMC, formerly the Vermont Monitoring Cooperative, VMC), processing heterogeneous data ranging from databases to images; (5) the Earth Observation Data Center (EODC), running a massive infrastructure serving Sentinel satellite images; (6) the Deep Carbon Observatory (DCO), providing centrally managed object identification and access services for a global community; (7) the xData platform operated by NICT for real time event monitoring and predictions; and (8) Ocean Networks Canada, providing a comprehensive solution across the wide range of data types from more than 400 different types of instruments, including real-time streaming data as well as from autonomous platforms with specific transmission modes. While the actual code for each solution is tightly integrated with each specific data repository setting, all adopters decided to release their implementations as open source modules for others to adopt and adapt. Links to these are provided in the respective sections. Several other implementations are being deployed in a variety of settings, for example, allowing citizen scientists' contributions to data being

[17] `https://doi.org/10.5281/zenodo.4015614`

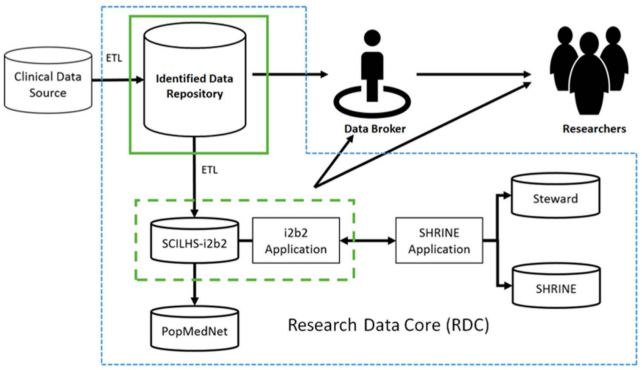

Figure 5: Adoption of RDA WGDC Recommendation to Electronic Health Records (McIntosh & Vitale, 2017).

acknowledged via dynamic query resolution (Hunter & Hsu, 2015), or integration into the DENDRO data sharing platform (Costa & da Silva, 2019).

4.1 Center for Biomedical Informatics (CBMI)

The Center for Biomedical Informatics (CBMI) at the Washington University in St. Louis (WU, now known as the Informatics Institute) implemented several of the recommendations in an initial set-up of its i2b2-based[18] repository of biomedical data. This repository contained electronic health record data and other biomedical data extracted from primary systems serving as a data warehouse for studies. This means that when a patient comes into hospital, data on normal care, diagnosis, procedures, medications, or any sorts of allergies are recorded for operational purposes. This data is replicated into the biomedical repository to be used for research (Center for Biomedical Informatics, Washington University, 2017). As part of a learning health system, this system supported a number of research data resources. The heart of the system is the Research Data Core (RDC), which is a confederation of data resources (see Figure 5) including i2b2. It comprised multiple database instances for collecting and storing data as well as a variety of web applications that interact with one or more of these databases. Collectively, these databases housed over six million patient records comprised of over 48 million visits, 114 million laboratory results, and 122 million text documents with daily updates. The initial system has since been replaced, but the pilot experience still provides some interesting lessons.

The implemented recommendations included **R1** - Data Versioning, **R2** - Timestamping, **R3** - Query Store Facilities, **R7** - Query Timestamping, **R8** - Query PID, **R9** - Store Query and **R10** - Automated Citation Texts, and have been described in greater detail in a prior publication(Gupta et al., 2017). Through a gap analysis and further investigation, it became clear that it was necessary to implement the enhancements at the source repository, which was built using PostgreSQL database management system–essentially a replicate of the clinical data.

To employ the changes the approach took the work almost entirely to the DB level, with only minor adaptations being required at the user interface level: In order to implement Data Versioning (**R1**) and Timestamping (**R2**) each live data table in need of versioning was given an additional column holding a timestamp. A corresponding 'history' table to hold historical data was created and is structured in the same way as live data tables but without primary keys and unique constraints. A trigger to be called before insert, update, or delete transactions was added to the live data tables. To allow a user to query all live and historical records in a given table or tables as they are or were, views were created for each table. To limit the results to a certain timestamp, a new limited access schema was built.

To incorporate Query Store Facilities (**R3**), Query Timestamping (**R7**), Query PIDs (**R8**) and to store queries (**R9**), a function serving three purposes was implemented. It a) ensured the currently executing query has been logged, b) returned a date and time value as well as c) a preformatted citation line. Hence, a query ID was assigned to each new query, which was then stored in the DB and included in the citation text. The Query Citations (**R10**) were implemented as a functionality within data broker views. When a new query was received via the standard Web interface and run using these views, the data broker would get a) the data set for the query and b) a preformatted citation line. The citation text always included the query's PID. As a result of these implementations, a previous data set as it existed at a specific point-in-time was successfully reproduced

[18] Informatics for Integrating Biology to the Bedside, Partners Healthcare Systems `www.i2b2.org`

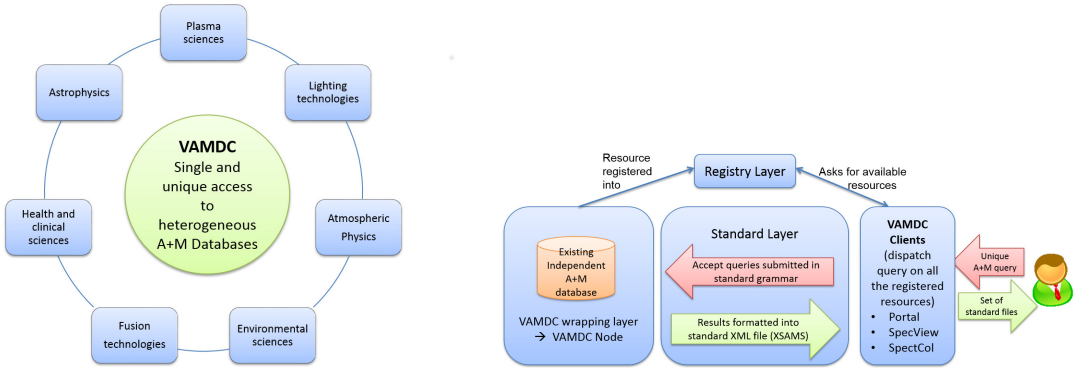

Figure 6: VAMDC Adoption: (a) Infrastructure set-up, (b) Query processing (Zwölf et al., 2017)

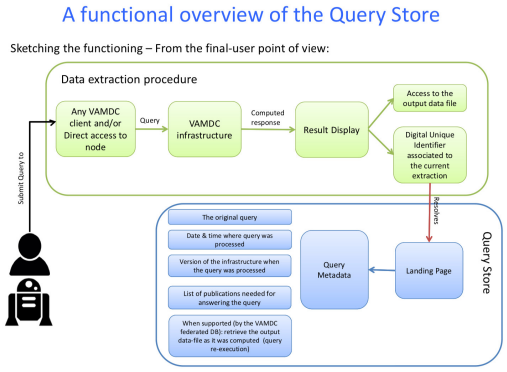

Figure 7: Functional overview of the query store at the VAMDC (Zwölf et al., 2017).

using only the PID as provided in a citation. Not all queries were immediately assigned a PID. They were first assigned a temporary ID in a kind of 'shopping cart,' from which the user could select those that were produced in the data subsets to be used in a study, whereas the others were purged after some time.

These implemented improvements simultaneously increased efficiency while decreasing the cost of queries. The average time of completing a data request was 20 hours. If this request was needed to be repeated (e.g., to check data, to add parameters), it typically took almost the same amount of time. Assuming an unsubsidized cost of 150USD/hr and an average of 20 hours per research request, 1 study (new or replication) would cost 3000USD. If a request needed to be replicated, the new request could be fulfilled on average in 3 hours saving 17 hours of time and 2,550USD. This work was initially funded through a 40,000USD grant, thus in less than 16 research requests for further data exploration, the investment will have paid off.

4.2 Virtual Atomic and Molecular Data Centre (VAMDC)

The Virtual Atomic and Molecular Data Centre (VAMDC) is a political and technical framework for operating and sustaining a worldwide digital research infrastructure built over two European projects (Dubernet et al., 2010; Albert et al., 2020). The e-infrastructure federates about 30 heterogeneous atomic and molecular databases that are used for the interpretation of astronomical spectra and for the modeling in media of many fields of astrophysics. Other application fields, cf. Figure 6a, include atmospheric physics, plasmas, fusion, radiation damage. VAMDC offers a common entry point to all federated databases through the VAMDC portal [19], providing a set of tools to retrieve and handle the data (Moreau et al., 2018)). Each node may use different technologies and tools for storing data and is responsible for its data. New data may be added or new versions of existing data may be provided. An ad hoc generic wrapping software, called the node-software, transforms an autonomous database into a VAMDC federated database, called a *data-node*. Each data-node accepts queries submitted in a standard grammar (VAMDC SQL Subset) and, by implementing an interoperable data access protocol, provides output formatted into a standard XML file (VAMDC XML Schema for Atomic Molecular

[19] `http://portal.vamdc.eu`

and Solid Data, VAMDC-XSAMS), shown in Figure 6b. The VAMDC data-citation implementation is based over the *Node Software technology* (Zwölf, Moreau, Ba, & Dubernet, 2019).

Concerning the data versioning and time-stamping (R1 and R2), VAMDC has two different mechanisms:

- coarse-grained: a modification of any publicly available data at a given data-node induces an increment in the version of the data-node. A notification mechanism informs that something has changed on a given data-node, indicating that the result of an identical query may be different from one version to the other.
- fine-grained: The information contained in the *Version* element in the VAMDC-XSAMS standard. (Zwölf, Moreau, & Dubernet, 2016) indicates which data have changed between two different data-node versions.

For extracting data from VAMDC, users may query directly a given known data-node or use one of the centralized query-clients (e.g., the already mentioned VAMDC portal). In the latter case, the centralized client software asks the registries what are the data-nodes able to answer and dispatches to them the query. Any centralized client acts as a relay. This is completely transparent from the data-node perspective and a data-node acts in the same way regardless the source of the query it is serving; when a data-node receives a query:

- it generates a unique query-token (this can be seen as a session token associated to the incoming query);
- it answers the query by producing the VAMDC-XSAMS output file, which is returned to the user together with the generated query-token. The stable sorting (R5) is guaranteed by the *node-software*. The token is copied both in the header of the answer and in the output file;
- it notifies the Query-Store (R3), providing the query-token, the content of the query, the version of the node and the version of the standards used for formatting the output. It is worth noting that this process is not blocking and has no impact on the existing infrastructure whatsoever: the data extraction process is not slowed down. If the Query Store cannot be reached, the user will still receive the VAMDC-XSAMS output file.

When the Query-Store service (R3) receives a notification from the data node, it stores the received information and reduces the query to a standard form, using the VAMDC SQL-comparator library[20] (R4), and it checks if a semantically identical query has already been submitted to the same data-node, having the same node version and working with the same version of the standards.

- If there is no such a query, the Query-Store service attributes a unique UUID and a timestamp to the new query (R8 and R7), downloads the data, that is, the VAMDC-XSAMS output file from the data-node and processes this file in order to extract the bibliographic information (each VAMDC-XSAMS file produced by the VAMDC infrastructure includes the references to the articles used for compiling the data) as well as metadata. The relevant metadata are stored and associated with the generated UUID. These metadata are kept permanently (R9): the Query-Store service permanently keeps the mapping between the UUID and the set of query-tokens assigned to a given query.
- If such a query is already stored in the Query-Store service, the new tuple (query time-stamp, query token) is added to the lists of the other time-stamps already associated with the query.

The functioning of the Query-Store is asynchronous. This was a mandatory constraint in order to avoid slowing down the VAMDC-infrastructure with a central bottleneck service. The unique identifier assigned to each query is resolvable (R11), and is both human and machine actionable (R12). The associated landing page provides the metadata associated with the query, as well as the access to the queried data and a BibTex citation snippet (R10).

The VAMDC Query Store is interconnected to Zenodo: from the landing page displaying the information about a given query, the user may with a simple 'click' trigger the replication of the data on Zenodo, together with all the metadata and bibliographic references. This procedure also generates a DOI as alternative PID for the current query. The interconnection with Zenodo[21] provides the Query Store with Scholix[22] functionalities and with lifetime access to the query-generated data.

This query store's benefits include a) its transparent usage for users, b) the live monitoring of queries and users, so that data providers can measure their impact, c) easy and automatized data citing and d) minimal

[20] `https://github.com/VAMDC/VamdcSqlRequestComparator`
[21] `https://zenodo.org/`
[22] `http://www.scholix.org/`

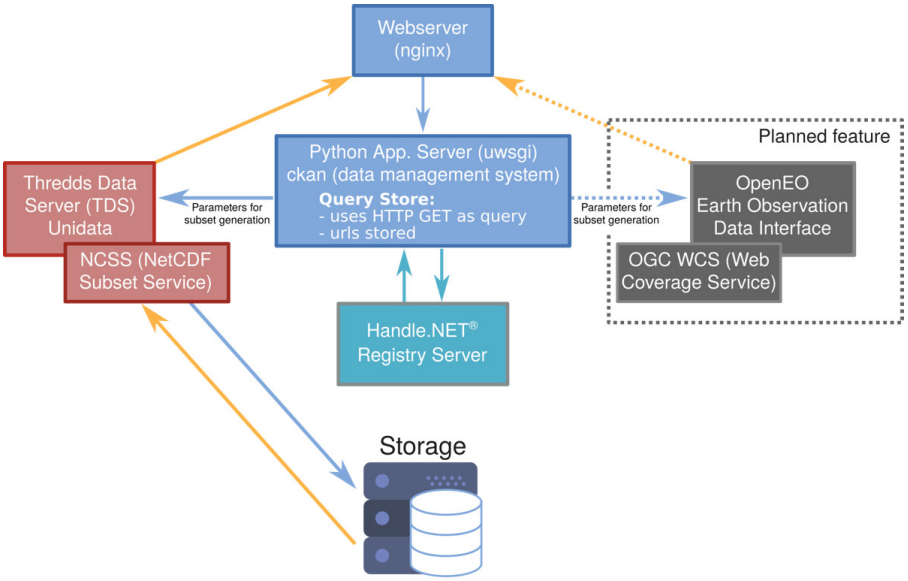

Figure 8: CCCA Data Centre: Simplified structure of implemented server and hardware components for the CCCA Data Centre environment: (i) CKAN web server, (ii) the application server for access, data management used as query store, (iii) Handle.NET®Registry Server for PID allocation, and (iv) the Unidata Thredds Data Server (TDS), NCSS Subset Service and features on Open EO support bedded with the RDA Adopotion Grant Framework

.

impact on existing infrastructures. DB owners only have to install the latest VAMDC wrapping software version. DB providers just have to fill a 'version,' field which is the label of a version. The source code of the implemented system is available on Github[23]. The effort required to design/implement the overall solution was about 20 person-months.

4.3 Climate Change Centre Austria (CCCA)

The Climate Change Centre Austria (CCCA) is a research network promoting climate research and climate impact research. The CCCA—Data Centre, as one of the three CCCA departments, operates a research data infrastructure for Austria with storage capacity of more than 700 TB embedded in a highly available Linux Server Cluster and linked to the high-performance computing facilities of the Vienna Scientific Cluster and the Central Institute for Meteorology and Geodynamics (ZAMG) as the national weather service. The service portfolio includes a central access point for storing and distributing scientific data and information in an open and interoperable manner. Starting with the initial tasks of setting up a data management framework for heterogeneous climate information, the focus changed in 2016 to highly resolved regional climate scenarios. These Austrian Climate Scenarios include climate parameters like surface temperature, precipitation, radiation, and so on, and various derived climate indices, for example, summer days. Calculated records are available on daily basis from 1970 up to 2100 in 1×1 km gridded data as multiple single files. The calculation process include different 'representative concentration pathways' (RCPs), ensembles of GCM (general circulation models) and RCM (regional climate model) runs, which are combined with statistical methods for the integration of in-situ observations for high-resolution conclusions. The open accessible entire data package for Austria includes over 1,200 files with a size up to 16 GB per file. These dependencies of different model ensembles and methods, as well as uncertainties as statistical down-scaling effects, force a continuous correction of some single data files. The interval between updates or new versions usually depends on the time frame of the funding schemes and their research projects and happens approximately once per year.

The main motivation for setting up a web-based tool for dynamic data citation and its fragments was to have a technical solution to align a persistent identifier with an automatically generated citation text.

[23] `https://github.com/VAMDC/QueryStore/`

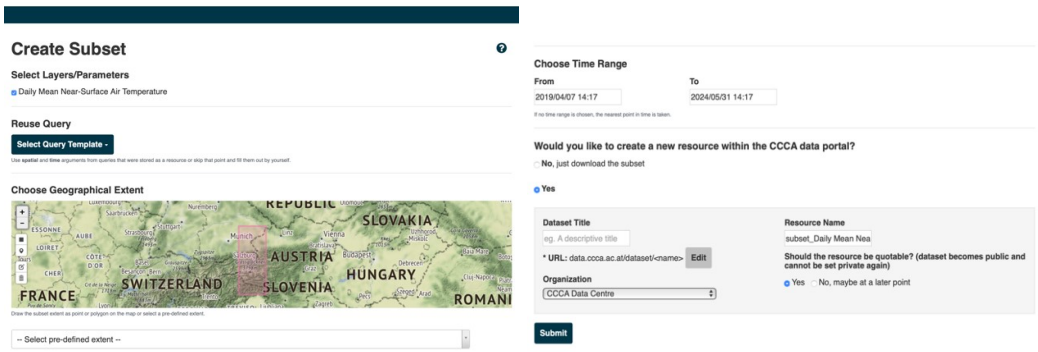

Figure 9: GUI of the subset creation function: (a) The upper part defines the parameter, or reuse an existing query, defining a bounding box either by polygon or predefined administrative units, (b) allows choosing the time range

.

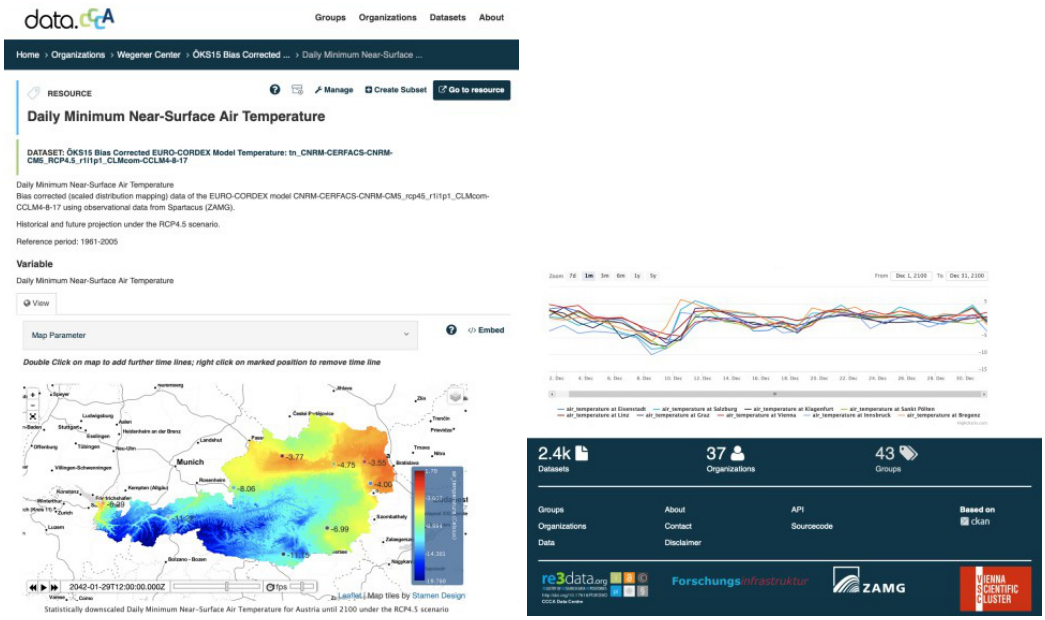

Figure 10: Landing page of a data resource where the subset can be created. a) The visualization is a view service (WMS), created by Thredds, and allows by activating the time control to visualize each time step up to 2100. b) A 2D timeline at a point of interest is processed on demand.

The core component for the set up within the CCCA environment depicted in Figure 8 (Schubert, Seyerl, & Sack, 2019) is the Handle.NET®Registry Server for PID assignment. For processing and creating data fragments, the Unidata Thredds Data Server (TDS) in combination with NCDF Subset Services (NCSS) is embedded. NCSS provides a catalog of parameters that allows the description of data fragments while retaining the original characteristics of the data. These are geographic coordinates, date ranges, multidimensional variables. NCSS uses "HTTP GET" request in the following structure:

Listing 3: NCSS Query via HTTP Get request

```
http://{host}/{context}/{service}/{data set}[/data set.xml |
        /data set.html | {?query}]
where
{host}                          = server name
{context}                       = "thredds" (usually)
{service}                       = "ncss" (always)
{data set}                      = logical path for the data set obtained from the catalog
data set.xml                    = to get the data set description in xml
data set.html                   = to get the human-readable web form
data setBoundaries.xml = to get a human-readable description of the bounding boxes
{?query}                                = subset requested
```

The subsetting element {*?query*} allows a combination of different parameters, like the name of variables, the location points or bounding box, arguments that specify a time range, the vertical levels, and the returned format. Figure 8 provides an overview of the relationships between requests (blue arrows) and responses (orange arrows) between the server, plus (aqua) alignment with PID Register. The application server takes the requests via the Web server and generates URL-based (HTTP GET) requests with the subsetting parameters (subset requests). These requests are stored in the query store and are assigned with the Handle identifier. Figure 9 illustrates the implemented components and gives an overview about the relationships between requests (blue arrows) and responses (orange arrows) between the server. The application server takes the requests via the Web server and generates URL-based (HTTP GET) requests with the subsetting parameters (subset requests). These requests are stored in the query store and are assigned with the Handle identifier.

A rough estimate puts the effort required to implement the above processes and tools at around 1.5 person months over a three months time period.

4.4 Forest Ecosystem Monitoring Cooperative (FEMC, formerly VMC)

The Forest Ecosystem Monitoring Cooperative (FEMC, formerly the Vermont Monitoring Cooperative, VMC) is a collaborative network that monitors forest ecosystems. The FEMC provides a data archive and an access and integration portal for the network to save data and to make them more available for assessment and research. Understanding complex ecosystem processes requires cross-disciplinary work. Thus, the data comes from many different fields in natural resources science and from diverse contributors, from citizen scientists to monitoring professionals to researchers.

Much of the data in the system are highly dynamic as data are added or updated frequently, necessitating versioning in order to cite the data. These data include observations of tree canopy condition, soil chemistry, high elevation bird counts, and photomonitoring of alpine vegetation quadrants. When possible, data are stored as tables in a relational database system that includes a metadata documentation workflow. Data that cannot be stored as a database table–such as raster, vector, and image formats–are stored in a file system. Users manage and access data through a web interface. Because the FEMC archive supports both monitoring and research use cases, the ability to cite an evolving data set in a way that allows others to access that exact state of the data used in a particular analysis is critical. FEMC sought to implement R1 – Data Versioning, R2 – Timestamping, R3 – Query Store Facilities, and R7 – Query Timestamping.

The workflow that emerged to implement the recommendations covers three different steps: data addition and editing, subsetting, and recovery. The data editing allows for provenance tracking so that a data set can be modified without affecting the previous iterations of the data set. Users add data to the system, at which point they can commit this to a version, preventing additional changes to the stored content. Modifications or additions to the data set are tracked as subsequent versions of the same data resource, preserving the evolution of the data set over time. When tabular data are updated to a new version, the database table storing the data is updated to the new state, and the database operations required to roll back the database table to the previous state are recorded. The system creates a result hash and a query hash to accompany the version identifying

information as a way to check the validity of any rollback operations. For file-based data, the file is given a unique name, registered in the version table, and stored in the file system, but the result hash is not created. A URL and, if requested, a digital object identifier (DOI) are assigned to the new version as well. Researchers can choose not to version the data they are providing, though they cannot switch off versioning once they have created at least one version.

The subsetting workflow enables data managers to build a specific state of the data set by using a builder or typing in SQL to work against a given version. It then shows the users the subset records matching that query. Once satisfied, users can commit this subset as a version, and a unique URL and, if requested, a DOI are be assigned in order to track the correct provenance. The recovery system restores previous versions by creating a new version table from the current data table state, compiling query steps and walking the table back to the prior state using stored SQL, or retrieving the appropriate file when stored in the file system.

For both subsetting and editing, timestamps of version commits are stored, tracked, and displayed to both managers and researchers. While FEMC did not implement R4 – Query Uniqueness and R6 – Result Set Verification, the current integration of query hashing and result set hashing in the versioning steps means that implementation would not be a significant burden in the future. FEMC stores the original subsetting queries, their associated metadata, unique URLs, and, when requested by the user, DOIs. No normalization is performed on the queries, thus for the time being two semantically equivalent queries formulated in different ways will get two different DOIs. However, identity of result sets can be determined by the result set hashes that are computed (Duncan & Pontius, 2017).

Implementing these recommendations required several person-months of time, primarily due to the need to adjust the historical data management workflow used by FEMC to be able to implement data set versioning. FEMC sought a parsimonious solution that did not require storing copies of data in every situation, leading to a lot of work to differentiate between additions and replacements. The implementation of the query store and the initial steps toward verification of query and result sets was relatively straightforward, requiring less than a person month of time. However, this upgrade solved several other issues in the FEMC archive, and thus taking these steps to lay the groundwork for the recommendations was welcome. The biggest improvement aside from addition of DDC capacity was the structure built to version data sets themselves. Previously, the same data were being uploaded as entirely new data sets every time a change was made, and users then couldn't figure out which to use, or the previous data was being overwritten to provide the one authoritative data set, breaking previous uses of the data. FEMC's long-term monitoring reports[24] were produced as a snapshot of the previous year's data in over a dozen key metrics. As monitoring data sets evolved, our site held only one representation and thus FEMC was invalidating links in its own publications by subsequently updating data. It also improved the culture of data management within the organization for FEMC's own monitoring work, by providing a clear point where FEMC staff certify the data produced as final and ready for distribution, such as our annual forest health monitoring work[25]. Researchers have appreciated the certainty that comes from versioning, and the dynamic data citation capabilities are now fully integrated into the data processing workflow in our forest indicators dashboard[26].

4.5 Earth Observation Data Centre (EODC)

The Earth Observation Data Centre for Water Resources Monitoring (EODC) is a processing and data backend founded in 2014 and located in Vienna, Austria. It operates a multi-petabyte, scaled, storage infrastructure connected to the Vienna Scientific Cluster (VSC) high-performance computing (HPC) system. It obtains Sentinel 1-3 data from the European Space Agency (ESA) Program Copernicus. New data are added in up to daily increments, depending on the satellite and sensor type. ESA releases data updates and corrections in cases when one of the instruments used for the observation was wrongly calibrated or broken. These are rare and never happened in the history of EODC so far.

To increase reproducibility of studies and support precise data citation, the existing systems were modified to support precise data identification. The implemented recommendations include **R3** - Query Store Facilities, **R4**-Query Uniqueness, **R6**-Result Set Verification, **R7**-Query Timestamping, and **R8**-Query PID, with versioning and timestamping (**R1** and **R2**) already in place via the standard storage infrastructure.

Researchers do not run their experiments locally but on the environment of the central server. Therefore, a definition of the processing steps and the input data is transmitted, for example, following the openEO standard (openEO Consortium, 2020). The backend creates a new job following the definition and waits for the

[24] `https://www.uvm.edu/femc/products/reports`
[25] `https://www.uvm.edu/femc/data/archive/project/forest-health-monitoring/dataset`
[26] `(https://www.uvm.edu/femc/indicators/vt`

Figure 11: Subsetting process in EODC

.

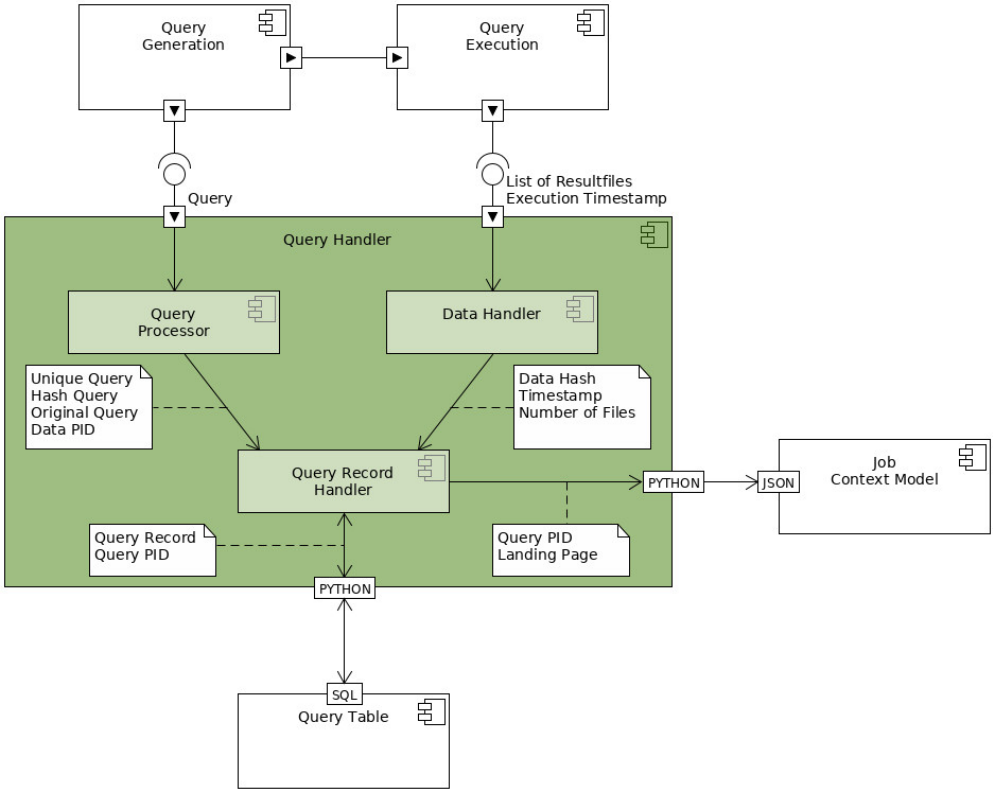

Figure 12: System architecture of EODC implementation

.

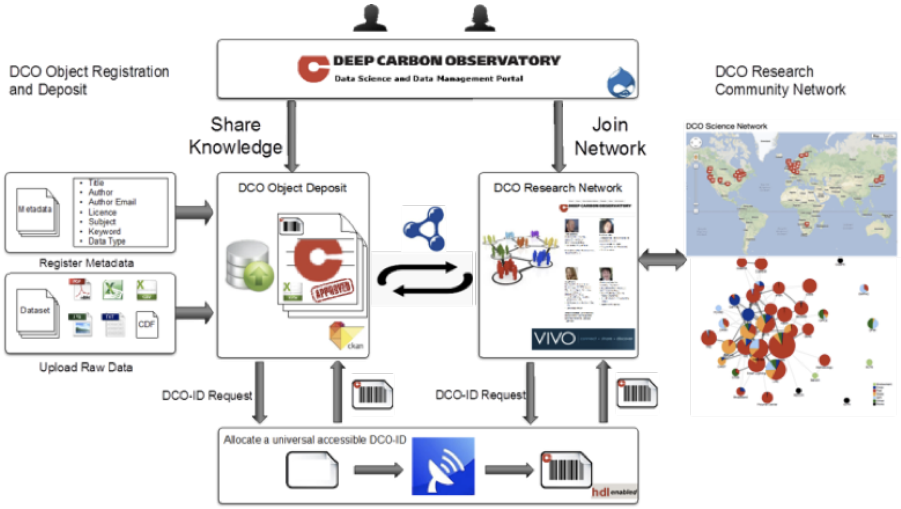

Figure 13: Deep Carbon Observatory Architecture

researcher to start it. After doing so, the backend starts the processing, providing the researcher with status information on request. When the backend finishes the processing, the researcher can request the result by following the provided download link. After adding our data citation extension to the backend, the workflow was unchanged, so researchers do not have to change the way to work with the backend.

EODC is a file-based earth observation backend, which uses a PostgreSQL (incl. PostGIS extension) meta-database with an Open Geospatial Consortium (OGC) compliant Catalogue Service for the Web (CSW) interface for querying. It represents a central service for data-driven applications of EODC using queries via XML requests and responses on a publicly available endpoint[27]. The unique path to a file is the identifier and used for the versioning, since every data update results in a new path. The creation timestamp is persisted in the meta-database, which we use to query for data versions that were available at a particular time (see red marker in CSW Query excerpt provided in figure 11). The query store is implemented as an additional table in the meta-database for the job executions. The unique query is defined as an alphabetically sorted JSON object of the filter arguments since the order of the filters makes no difference in the outcome. The filter arguments consist of the satellite identifier, the spatial extent, the temporal extent, and the spectral bands of the satellite. The result of the query is a list of files with a fixed order defined by the CSW standard (Gößwein, Miksa, Rauber, & Wagner, 2019). For fast comparisons, the hash of the resulting file list and the hash of the unique query are stored in the query table. During every job execution the query PID of the input data is added to the job meta-data, either by generating a new query PID or by using an existing query PID if the same query was executed before. Figure 12 provides an overview of the system architecture of the EODC extension. The *Query Record Handler* is responsible for the uniqueness of the query entries and adds the query PID to the job meta-data (*Job Context Model*).

The effort for implementing the solution was roughly one person month. The most challenging part was the integration with the standards underlying the system of EODC (OGC, CSW, and PostGIS) and to set up a test instance of the actual server. The implementation was relatively straightforward, since the versioning and timestamping was already in place for the GeoGIS database. It was limited to extending the query execution part of the backend and creating the human-readable as well as the machine-actionable landing page. Additionally, the citation and referencing functionality was integrated into the job definition part, so that other users can use the exact input data of others, by providing the data PID.

4.6 Deep Carbon Observatory

The Deep Carbon Observatory (DCO) is a global community of multidisciplinary researchers unlocking the inner secrets of Earth through investigations into life, energy, and the fundamentally unique chemistry of carbon. It started as a 10-year initiative that produced significant data and scientific results.[28] DCO organized a Data Portal that provides a centrally managed digital object identification, object registration, and metadata

[27] https://csw.eodc.eu/
[28] `https://deepcarbon.net/vivo`

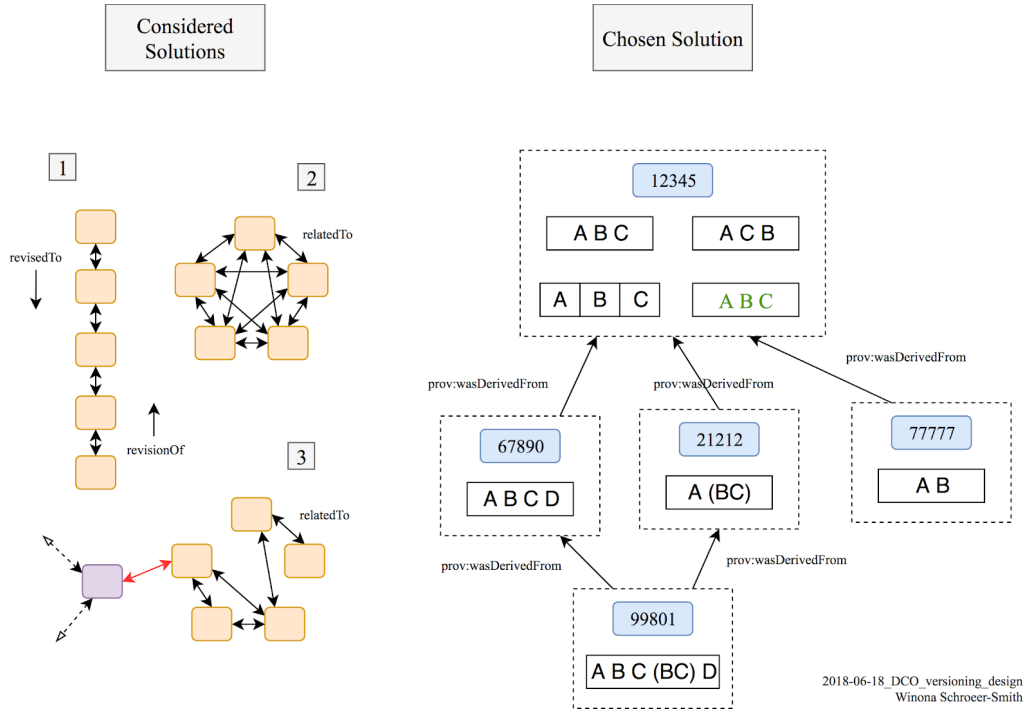

Figure 14: DCO Subsetting Process

.

management service that provides discovery and access to diverse data for the DCO community.

The DCO Data Science Team at the Tetherless World Constellation of Rensselaer Polytechnic Institute maintains the DCO Data Portal outlined in Figure 13. The Portal makes extensive use of Persistent Identifiers, most notably something called the DCO-ID. The DCO-ID is a Handle and is similar to the Digital Object Identifier (DOI) for publications, but it extends the scope to many more types of objects, including publications, people, organizations, instruments, data sets, sample collections, keywords, conferences, and so on. Each DCO-ID can redirect to the Web profile (often a landing page) of an object where detailed metadata can be found. In the DCO Data Portal, each object is the instance of a class. The metadata items describing an instance are properties. All those classes and properties are organized by the DCO ontology (Ma et al., 2017; Prabhu et al., 2020).

In implementing the WGDC Recommendation, the team essentially walked through the 14 recommendations, assessed where they were in terms of compliance, and then made adjustments accordingly. Most of the work was conducted by a summer undergraduate student hired as part of an RDA/U.S. adoption grant. She required expert guidance, however, from graduate students well-versed in the portal as well as senior staff able to advise on changes in policies and workflows. A number of the 14 recommendations were already implemented, but several required some thought and adaptation. In particular:

R1 Versioning. After considering several options, the team decided the best option was to add the PROV property prov:wasDerivedFrom to the DCO ontology and knowledge graph (Figure 14).

R3 – 9 Query store and management. The DCO portal only provides queries of and access to collection level data. It does not allow querying subsets of the collections. Nonetheless, the team added a mechanism that allows portal users to store a query that is a URI. They also updated DCO-ID instances with a prov:generatedAtTime relationship from their Handle records. When a user selects to store a query, the particulars of the current query, that is, search keywords, values of filter facets, ordering, and so on are stored along with a standard representation of the current date/time for future recall. When a stored query is rerun, the query is executed along with the recorded original date/time such that it only records those DCO IDs minted prior to the query date are returned. This required tighter coupling between the portal and the Handle server and additional interface options to save a query and repeat with a specific timestamp.

This query management process added functionality to the portal, but it did not enable citation of subsets of collections (although it does capture relevant versions). That said, the team was able to reapply the portal technologies to enable search and access to a large collection, the Global Earth Mineral Inventory. This allows

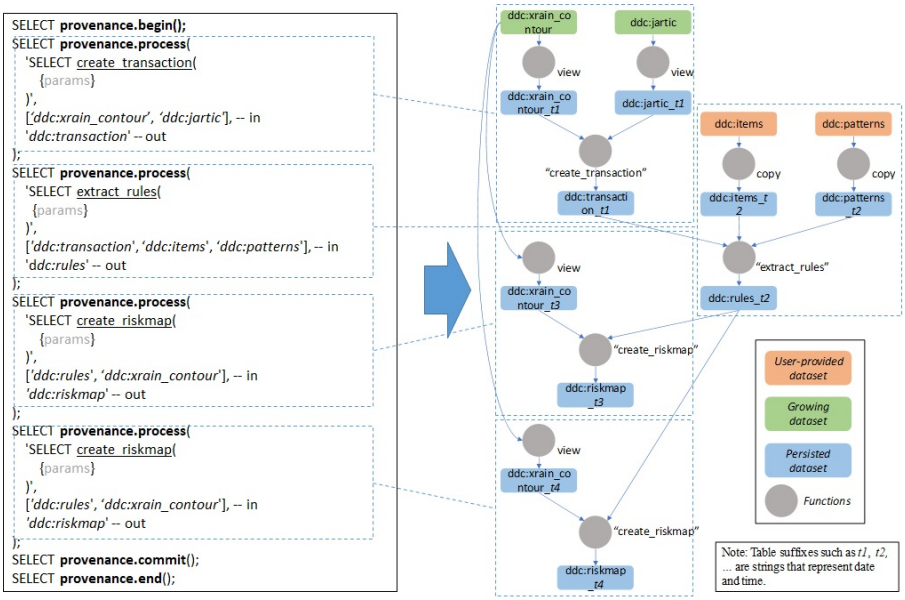

Figure 15: Data provenance using dynamic data citation on xData Platform.

Dynamic data citation text	View
ddc: xrain_contour_201910140946.215344	CREATE VIEW provenance.f362aee8AS SELECT *FROM xrain_contour WHERE datetime <='2019-10-14 09:46';
ddc: xrain_ contour_ 201910160952.876604	CREATE VIEW provenance.43bd2c64AS SELECT * FROM xrain_contour WHERE datetime <= '2019-10-16 09:52';

Figure 16: Example of dynamic data citations on xData Platform.

citation of subsets of that collection (Prabhu et al., 2020).

R13 Technology Migration. This was a policy effort not a technical effort. A graduate student with a background in business and policy developed a draft migration plan, which was reviewed and approved by the overall DCO Portal team. The critical issue will be **R14** actual implementation and verification, which may need to happen soon.

Overall, the robustness and semantic flexibility of the DCO portal architecture allowed for a straightforward implementation of almost all of the WGDC Recommendations. The project also demonstrated that the technology can readily be transferred to other data collections. The challenge will be in sustaining and eventually migrating the system. This has emerged as a critical issue with the formal end of the DCO project. This is a larger issue than dynamic citation, however, and illustrates the challenges of sustaining research data infrastructure in general.

4.7 xData Platform at NICT

The Big Data Integration Research Center at the National Institute of Information and Communications Technology in Tokyo (NICT) develops a data analysis platform, called **xData Platform**, on NICT Integrated Testbed. It aims at collecting heterogeneous sensing data from various data sources, then discovering and predicting associations of complex events in the real world for providing actionable information (Zettsu, 2019). For example, it enables predictive modeling of traffic obstructions caused by extraordinary weather based on association mining between weather observation data and traffic monitoring data for route navigation applications. Multiple domains of data sets are being collected by platform users such as weather observation data (precipitation radar, meteorological stations), atmospheric observation data (air pollution observations, personal

environment sensors), traffic monitoring data (congestion, prove car), and lifelog data (fitness sensors, camera sensors).

The xData Platform consists of a database server, called Event Data Warehouse, and APIs for collecting, associating, predicting and distributing data sets. The Event Data Warehouse transforms the data sets into a common format, called event data, for storing heterogeneous sensing data in an interoperable manner. Through the APIs, a transaction data table is created by joining multiple event data onto spatial and temporal attributes. Cleaning and tailoring the transaction data, predictive modeling of associative events is processed based on data mining and machine learning methods like frequent itemset discovery and deep neural networks. The prediction results are then distributed in application-friendly formats like JSON. For the purpose of tracing and verifying the cross-data analysis process, the xData Platform provides a **data provenance** function. As shown in Figure 15, the data provenance function captures a workflow showing how data sets are selected, processed and generated by the APIs during execution of an analysis process. A programming library is provided for capturing the provenance information (e.g., *provenance.process(API)*).

Dynamic data citation is a key for realizing the data provenance function for sensing data. Adopting the **R3** (Query Store), a dynamic data citation is implemented based on a 'view' of a data table in Event Data Warehouse, which is a database object containing a query to a database table for selecting a target data set dynamically. Concerning the **R1** (Data Versioning) and **R2** (Timestamping), dynamic data citation is represented by combination of a table name and a timestamp of view creation. Figure 16 shows an example of dynamic data citation, where two different views are created from a growing archive table of rainfall sensing data "xrain_contour" for citing the data set available at two different execution times of a workflow. The view-based dynamic data creation is generated automatically by the provenance library (**R10** - Automated Citation Text). When *provenance.process(API)* is invoked for an API taking a growing archive table as an input or updating an existing table, a dynamic data citation is created for a (materialized) view selecting a snapshot of currently available data set from the table (**R6** - Result Set Verification, **R7** - Query Timestamping).

Tracing provenance information enables users to resolve derivations between the dynamic data citations. Though a provenance visualization tool integrated with an IDE, xData Platform users can use it for verifying credibility of a cross-data analysis result using individually created data sets. It is also used for managing different combinations of inputs, outputs, and parameters of a prediction model for its fine-tuning based on different hypotheses in data science work. We are also extending it to a distributed collaborative environment by introducing a location-identification mechanism like URI to the dynamic data citation.

4.8 Ocean Network Canada

Ocean Networks Canada (ONC) operates observatories and platforms in coastal, deep-ocean, and polar environments. The majority of data streams are real-time from cabled observatories such as the NEPTUNE observatory in the North East Pacific, but there are also autonomous and mobile platforms with other data transmission modes. In addition to serving these data management needs, ONC fulfills a repository role for partners spanning government, nonprofit, industry, and First Nations. Over 400 instrument types are supported, representing thousands of instruments and deployments. Data sets hosted at ONC are highly dynamic, changing over time as new records are added and as errors are corrected. In order to introduce data citations within ONC's digital infrastructure, known as Oceans 2.0, the MINTED (Making Identifiers Necessary to Track Evolving Data) project (Ocean Network Canada (ONC), 2020) was awarded funding through CANARIE's Research Data Management program. Figure 17 shows the relevant relationships between Oceans 2.0 and external entities.

Important considerations when assigning data set identifiers include data set granularity conventions, partner recognition, and geospatial metadata. After carefully considering data set boundary options (e.g., time, geography, instrument type, platform, data processing level) and constraints (DataCite metadata kernel, contributor attributions, repository architecture), it was decided that one deployment of one device would represent one data set, that is, a DOI registered at DataCite. Attributions to organizational data partners are included in the DataCite entries, including Research Organization Registry identifiers when available. Although geospatial extent metadata is not required by DataCite, the latitude and longitude range is deemed necessary by ONC since location is an essential aspect of ocean data discovery. The implementation used at ONC supports fixed-position, mobile, and remote sensing instruments.

The query store–related recommendations were mostly well-aligned with existing infrastructure. The data discovery interfaces within Oceans 2.0 allow researchers to access subsets of data sets based on their selected criteria of time, variables, formats, and data product processing parameters. These query details are stored within a relational database and are assigned a resolvable internal identifier, but were not normalized for

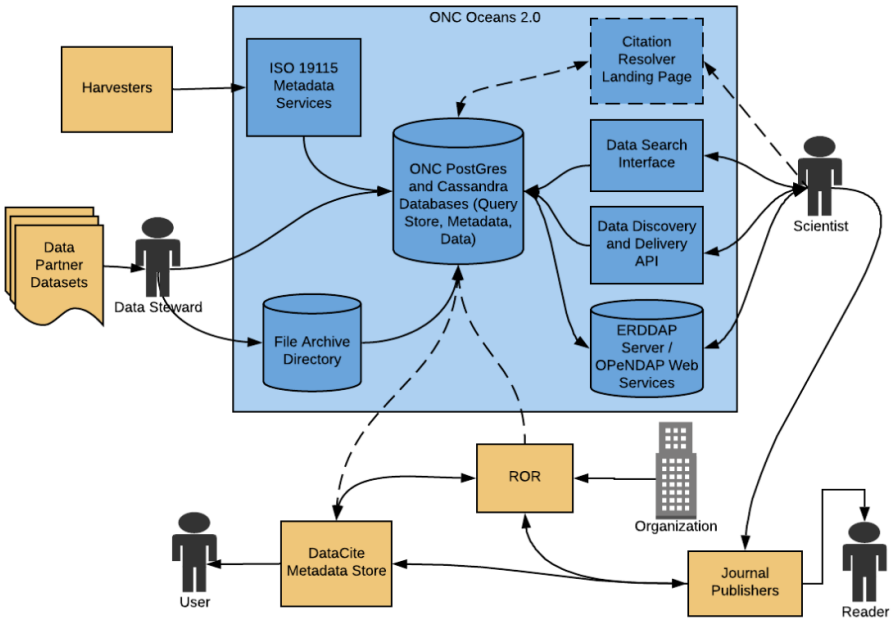

Figure 17: The ONC Oceans 2.0 system (in blue), and third party sources and applications (in orange). Dotted lines indicate aspects that were added, while all ONC components were modified. Modifications included an extended data model, additional web services, integration of third party APIs and data citation features.

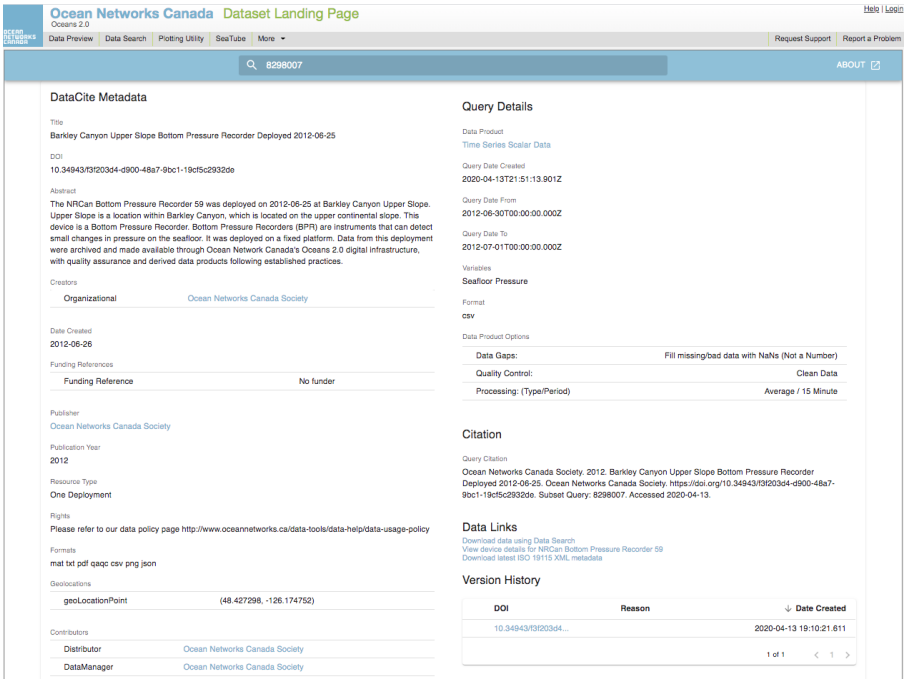

Figure 18: Landing page of a data set, including subset query details shown on the right hand side. (https://data.oceannetworks.ca/DatasetLandingPage?queryPid=8298007)

uniqueness (this was deemed low priority due to low likelihood of exactly the same queries, and since it can be reconsidered in the future due to saving all queries).

Landing pages and web services provide metadata and citation text for both full and subset data sets (accounting for requirements R10, R11, and R12). An example landing page is shown in Figure 18. The citation text is following conventions from the ESIP Data Citation Guidelines for Earth Science Data, Version 2 (ESIP Data Preservation and Stewardship Committee, 2019). A Technology Migration Policy was established to ensure sustainable resolution of data sets (related to R13 and 14), including mitigating measures like unit tests, regression tests, workflows, and comprehensive documentation. Data stewards use tools in Oceans 2.0 that automate DOI minting using DataCite services, assuming that the necessary metadata exists and using algorithms to construct data set titles and abstracts. Data stewards verify the data citation as part of their device workflow commissioning phase, in case any of the metadata was not correct at the time of minting. Data set versioning (R1) results in a new DOI that is associated with its predecessor, using a framework that can capture and display provenance details. The ONC implementation is based on a batch system, which aggregates versioning triggers (e.g., calibration formula change), data versioning tasks (e.g., reprocessing), and DataCite DOI updates (e.g., new DOI minting, populating related identifier fields using relationship types "isPreviousVersionOf" and "isNewVersionOf"). In the future, it is intended to reformulate the provenance information in terms of W3C PROV ontology (based on agents, activities, and entities). The versioning history is also displayed on the landing page. Additional work planned includes more explicit end-user support, version notification services, ORCID integration, citation metrics and more. The initial work using these RDA recommendations provided the foundation upon which enhancing services and features can be added.

5 Discussion and Lessons Learned

Data citation is still an emergent practice. While there is broad acceptance in the information science community, as evidenced through the Joint Declaration of Data Citation Principles (Data Citation Synthesis Group, 2014), the actual practice is still evolving, especially for citing dynamic data (Parsons, Duerr, & Jones, 2019) Nonetheless, multiple implementations both conceptual and in practice, especially those briefly presented in this article, suggest that the RDA Recommendations present a valid, viable, and adaptable approach that may be emerging as a community standard. It is also clear that the specific implementation within a repository is highly contextual. The fourteen recommendations serve as guiding principles that inform specific technical decisions for a particular data management system. In this sense the recommendations do, indeed, work for all kinds of data and in a diversity of settings.

Despite this apparent success, repositories can still find it daunting to implement the recommendations. Indeed, most of the pilots received additional funding for implementation. This is understandable. Data stewardship is an unending (underfunded) and increasingly complex process. We have found that certain questions or concerns frequently arise for which we now have answers based on real-world experience. Therefore, the rest of this section is a sort of 'FAQ' that may help address the concerns of future adopters.

Do the recommendations work for any kind of data? Yes, it appears so.

The solutions presented include small-scale textual data, relational databases (MySQL, PostgreSQL), NoSQL Databases (MongoDB), native XML Databases (BaseX, eXist-db), filesystems managed standalone or in combination with distributed database systems such as GeoGIS, or multidimensional data cubes such as, for example, NetCDF files. Queries include dedicated scripts in R or using Java libraries mimicking database functionality over CSV files, SQL, XPATH, dedicated query languages for NoSQL databases, bounding boxes drawn on a map, and specific interfaces for data cubes such as NCDF subset services for NetCDF files. Neither conceptually nor in practice have we found a data type or query structure where the recommendations would not be feasible to implement.

Do all updates need to be versioned? Ideally, yes. In practice, probably not.

Settings that see extremely high-frequency updates over massive amounts of data may face a challenge in maintaining ALL states that ever occurred. We propose that in settings where not all states of data that ever existed need to be documented (e.g., for accountability reasons) or where states that were never read (i.e., updates to the database without any intermittent read operations to that data, data could be overwritten without versioning). Alternatively, versioning at lower frequencies that serves the needs of the respective domains may be a solution as well. In such cases, it should be made clear that only queries selecting data from certain 'stable states' are reproducible, clearly separating data for live-tracking and monitoring from research data serving as a basis for studies and decision-making. Similarly, when massive data volumes create economic challenges in maintaining multiple versions, such as reprocessing large numbers of satellite image data, respective

trade-offs must be considered and some versions must be deprecated. In all cases, a recurrent query should result in a meaningful result even if it is to state that the particular version of the data is no longer available (see next question).

May data be deleted?: Yes, with caution and documentation.

The recommendations do not prohibit deletion. States may be overwritten and earlier versions of data may be deleted. However, as with any action impacting reproducibility and transparency of experiments, this should follow a well-designed and well-documented process. In accordance with standard citation guidelines, the metadata describing any subset rendered irreproducible due to deletions, should remain available, that is, the meta-information provided by the query store should be maintained.

What types of queries are permitted?: Any that a repository can support over time.

Queries can be of any type as long as the repository can assure their identical reexecution across technology migrations over time. Query constructs that may see changing semantics or may face numeric inaccuracies (e.g., by integrating more complex processing or mathematical constructs as part of queries) should be avoided. We recommend a clear separation of data subset selection processes and data processing and analysis steps. We assume that certain queries are sufficiently well-defined and precise in their computation that they can be migrated across a range of technological platform changes. These may include counts, min and max value determinations, ranges in defined spaces of time, geography, and space; and type queries on well-established categories.

Does the system need to store every query?: No, just the relevant queries.

Several pilots allow the user to decide when a query should persist. It is important to allow users to explore, refine, and revise their queries. For example, CBMI used a 'shopping cart' approach and DCO allows users to decide when they want to 'share' a PID for a specific query.

Which PID system should be used?: The one that works best for your situation.

The recommendations are neutral to the actual PID system being used. However, it is highly recommended to adopt a system that is widely used within the community, and that allows references to be easily and transparently resolved. Some of the pilots use established systems like DOIs or Handle.net, while others have chosen to resolve the query identifiers themselves. Other aspects of PID systems, such as external visibility, use by aggregators, or cost aspects may also influence decision-making. This is, for example, demonstrated by the VAMDC implementation pushing the information to Zenodo to issue a DOI to connect to the Scholix initiative.

When multiple distributed repositories are queried, do we need complex time synchronization protocols?: No, not if the local repositories maintain timestamps.

As the VAMDC deployment shows, no stringent time synchronization is required. As query stores are local with the data provider, only local timestamps are relevant. Queries distributed over a network of nodes are stored with one local timestamp at the node that answered the query, which, in turn, distributes the query and receives answers from the distributed nodes with their local timestamps of execution.

How does this support giving credit and attribution?: By including a reference to the overall data set as well as the subset. The recommendations foresee two PIDs to be listed: one for the immutable, specific subset identified by a query at a given point in time, and one for the evolving data source. This is similar to conventional references in the paper world, where both a specific paper (immutable) is cited within the context of an (evolving) journal or conference series. By having these dual identifiers, attribution can be traced at the institutional level while supporting precise identification for reuse. The recommendations were originally geared toward the reference aspect of citation rather than the credit aspect, but they have also been repurposed to provide fine-grained credit as well. For example, Hunter et al. (Hunter & Hsu, 2015) demonstrate how the recommendations can be adopted as a mechanism to identify and credit individual, volunteer contributors to a large citizen-science data set of bird observations.

How does this support reproducibility and science?: By providing a reference to the exact data used in a study. The information required to precisely identify any arbitrary subset of data (including even the empty set, i.e., a query that returns no result!) comes 'for free' in a very precise manner as actually executed by a subset selection process. Storing this information provides a more precise definition than natural language descriptions in the methods section of a paper may be able to provide. It also allows the PID of that data subset to be used as an input parameter into other processes, thus easing the creation of meta-studies or the continuous monitoring of specific analyses.

Does this data citation imply that the underlying data is publicly accessible and shared?: No. The citation should, as usual, lead to a landing page providing relevant but nonsensitive metadata as well as information on access regulations and process to request access if possible. These should be both human as well as machine processable to support automation.

Why should timestamps be used instead of semantic versioning concepts?: Because there is no standard mechanism for determining what constitutes a 'version.' Contrary to the software world, where semantic versioning is widely adopted, it is hard to apply stringent protocols to define the difference between major and minor updates to data. While in the software world, major releases are often those that break certain interfaces or introduce significant new functionality, whereas minor releases comprise mostly bug fixes and other nonfunctional improvements, such differences hardly exist with data. Fixing a spelling mistake in a data record will make a difference to data sets extracted from the database with that record suddenly being (no longer) found after the correction. Translating attribute names, changing encodings, or increasing the precision of numeric representation will have an impact on subsequent computations and may thus lead to artifacts, rather than referring to small and large differences in data; it thus seems recommended to refer to 'the state of the world at a given point in time,' which—as a concept—works across all disciplines and data types.

How complex is it to implement the recommendations?: It depends on the setting. The difficulty of implementation obviously depends on the complexity of the data infrastructure, the type and volume of data and changes of the data, as well as the query processing load, among other aspects. Most pilot adopters have reported effort in the order of a few person months spread over a period of about six months to to a year. The effort was greater for the complex distributed environment in VAMDC. The ONC effort was quite significant, but one could argue that the recommendations may have saved effort by providing a guiding framework for their system upgrade. Note also, the recommendations do not need to be rolled out in a big-bag scenario across all data sources at once; incremental approaches deploying different recommendations over time or for individual data products can also provide benefit.

Why should I implement this solutions if my researchers are not asking for it or are not citing data?: Because it's the right thing for science. This is a kind of chicken-and-egg problem. Solid scientific practice requests researchers to meticulously describe the data used in any study. This is currently extremely cumbersome, with researchers spending effort on precise descriptions in their methods section. The complexity as well as lack in precision discourages reuse of data and lowers reproducibility of scientific research. If we are able to provide mechanisms that make citing data as easy (or even easier) than citing papers, researchers will be more willing to do it, although any such cultural change will require time in addition to the functionality actually being available. This is all part of creating the necessary infrastructure for open, robust, and reproducible science. It will be an incremental process. It is also worth noting, as discussed below, that all the pilots gained additional value through improved data stewardship as well.

6 Conclusions

Since the RDA Recommendations on Dynamic Data Citation were released 5 years ago, they have been successfully implemented by a number of institutions in a variety of settings around the world, several of which have been illustrated here. Not all of the pilots described have implemented all 14 of the recommendations (arguably R13 and R14, Technology Migration and Verification, remain somewhat untested), but all the pilots found benefit in implementing even a subset of the recommendations in terms of improved data processes, data-quality aspects, and policy decisions. All the implementations required nontrivial and sometimes major work, but all found it to be a worthwhile effort. In some cases, such as CBMI, the ability to easily repeat past queries clearly saved the repositories and their users time and effort. In other cases, such as FEMC and ONC, the pilots reported positive user feedback and extension to other tools. In all cases, repository systems were made more robust and trustworthy: ONC was guided through a major system upgrade; DCO and NICT improved their versioning and provenance; CCCA and FEMC improved their data selection processes and GUIs; most pilots enhanced the information on data landing pages; and all pilots have a clearer and more-documented understanding of their data management processes as well as a clear statement on data citation.

While the actual technical solutions differ, the principles were feasible across all settings. We have not identified a setting so far where they would not work, neither in practice nor conceptually in numerous discussions at the biannual meetings of the WGDC and at several other workshops. Despite the variance in technical solutions, the effort is primarily technical and can be implemented in a gradual or phased approach. It is the cultural and policy aspects of citation that remain the most challenging.

First off, citation is still not a cultural norm in most of the scientific community, and even when journals request or require data citation, there is little, if any, consideration of the type of dynamic citation we describe here. Nonetheless, data citation is a growing concern that is rapidly being implemented in some disciplines such as the Geosciences (Stall et al., 2019), and there is growing expectations around scientific validity and reproduciblity (Fanelli, 2018). At the same time, the community recognizes the critical importance of PIDs in

making data "FAIR" — findable, accessible, interoperable, and reusable (Wilkinson et al., 2016). We hope that making precise reference easy and transparent for the researcher will help address these issues.

We also found it important to consider the multiple concerns of citation. Data citation is defined as "a reference to data for the purpose of credit attribution and facilitation of access to the data" (Task Group on Data Citation Standards and Practices, CODATA-ICSTI, 2013). In this work, we have been primarily focused on the reference and access aspects of this definition. We have paid less attention to credit attribution. As discussed, we maintain the same high-level, author-style credit that comes from citing the whole data set, and at least one project has used the approach to provide fine-grained credit to individual data contributors (Hunter & Hsu, 2015), but credit has not been our focus. Indeed it is unclear as to whether citation is the best or primary way to credit data contributors and stewards despite a growing recognition that they *should* be credited (Dosso & Silvello, 2020; Parsons et al., 2019; C. Borgman, 2016). Regardless of how credit schemes for data evolve, it will be useful to precisely reference a contribution.

Secondly, it is the policy considerations embedded in the 14 recommendations that tend to be the most difficult. These emerge primarily in data versioning (R1) and in technology migration and verification (R13 and R14).While versioning is, in principle, a pretty well-defined concept, the best way to implement it for any given data source is a complex problem in its own right. A dedicated RDA working group[29] is investigating different approaches to versioning. Guidelines for what is considered a meaningful difference or relevant time-granularity as well as retention policies will require discipline-specific agreement. Yet if reproducibility is a goal to be met, and data is evolving, then ensuring that previous states of a data collection can be reconstructed is an unavoidable requirement.

Technology migration is also a well-defined concept with great complexities. The ISO standard Open Archival Information System Reference Model (International Organization for Standardization, 2003) defines 'long-term' as "a period of time long enough for there to be concern about the impacts of changing technologies." In other words, it is a very contemporary concern. Many repositories now see media and other technical migrations as operational concerns. Data infrastructures require perpetual maintenance, and this work, while critical, is often invisible, undervalued, and underfunded (Olson et al., 2019; C. L. Borgman, Sands, Darch, & Golshan, 2016). Maintaining precise identification of data may be cumbersome, but it is clearly an essential aspect of archiving. Indeed, one might consider the maintenance of reference schemes almost as essential as maintaining the data. Data are worthless unless you know what they are and where they are. This is why libraries are some of the longest living institutions on the planet.

Overall (while we are admittedly biased), we find that the RDA Recommendations on Dynamic Data Citation have shown to be a robust and viable approach to precisely identifying arbitrary subsets of data so that they can be reproduced. This has multiple scientific benefits. Indeed, the highest benefits are not yet fully realized. PIDs provide machine-actionable, precise specifications of input data and may serve as input parameters in analytical processes and models. This greatly simplifies automation and enables automatic study reexecutions when, for example, code in a library has changed, or reexecuting the same analysis on the same semantic definition of a data subset but at a newer state. If machines can be unambiguously and repeatedly told exactly the data in question, it could lead to dramatic improvements in the quality and efficiency of data processing, data analysis pipelines, and modeling.

Acknowledgments

This work benefited from the contribution of numerous experts participating in discussions during the WGDC meetings at the RDA plenaries, as well as the input and feedback from developers to policy makers in the various implementation projects. Their feedback was extremely valuable in initially shaping the recommendations and providing guidance to repositories embarking on the project to adopt them as well. The framework provided by RDA for these discussions was invaluable. RDA also helped secure pilot funding from the European Commission and the U.S. National Science Foundation to support institutions via adopters' grants, which supported several of the pilots. Overall, the gravitas and reach of RDA helped many projects acquire a bit of additional funds to implement the recommendations. Support for the work described in this paper includes but is not limited to support from several programs funded by:

- The Austrian COMET program
- The European Commission

[29] `https://www.rd-alliance.org/groups/data-versioning-wg`

- The US National Science Foundation
- CANARIE - The Canadian Network for the Advancement of Research, Industry and Education
- The AP Sloan Foundation
- National Aeronautic and Space Administration

References

ACM US Public Policy Council. (2017, January). *Statement on Algorithmic Transparency and Accountability.* ACM. Retrieved from `https://www.acm.org/articles/bulletins/2017/january/usacm-statement-algorithmic-accountability`

Albert, D., Antony, B. K., Ba, Y. A., Babikov, Y. L., Bollard, P., Boudon, V., ... Zwölf, C. M. (2020, October). A Decade with VAMDC: Results and Ambitions. *Atoms*, *8*(4), 76. doi: https://doi.org/10.3390/atoms8040076

Borgman, C. (2016). Data citation as a bibliometric oxymoron. In C. R. Sugimoto (Ed.), *Theories of informetrics and scholarly communication* (p. 93-115). Berlin & Boston: Walter de Gruyter GmbH & Co KG.

Borgman, C. L., Sands, A. E., Darch, P. T., & Golshan, M. S. (2016). The durability and fragility of knowledge infrastructures: Lessons learned from astronomy. *Proceedings of the Association for Information Science and Technology*, *53*(1), 1–10.

Center for Biomedical Informatics, Washington University. (2017, January). *Adoption of the RDA Data Citation of Evolving Data Recommendation to Electronic Health Records.* YouTube. Retrieved from `https://www.youtube.com/watch?v=SYIOm9_j2J4` (Accessed: 2019-08-05)

Costa, L., & da Silva, J. R. (2019). A fair, open-source data sharing platform. In *Proceedings of the 23rd International Conference on Theory and Practice of Digital Libraries, (TPDL 2019)* (p. 384-387). Springer.

Data Citation Synthesis Group. (2014). *Joint Declaration of Data Citation Principles.* Martone M. (ed.) San Diego CA: FORCE11. Retrieved from `https://www.force11.org/datacitationprinciples` doi: https://doi.org/10.25490/a97f-egyk

Dosso, D., & Silvello, G. (2020). Data credit distribution: A new method to estimate databases impact. *Journal of Informetrics*, *14*(4), 101080.

Dubernet, M., Boudon, V., Culhane, J., Dimitrijevic, M., Fazliev, A., Joblin, C., ... Zeippen, C. (2010). Virtual atomic and molecular data centre. *Journal of Quantitative Spectroscopy and Radiative Transfer*, *111*(15), 2151 - 2159. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0022407310001731` (XVIth Symposium on High Resolution Molecular Spectroscopy (HighRus-2009)) doi: https://doi.org/10.1016/j.jqsrt.2010.05.004

Duncan, J., & Pontius, J. (2017, February). *Implementation of Dynamic Data Citation at the Vermont Monitoring Cooperative.* Retrieved from `https://www.rd-alliance.org/system/files/documents/170213_RDA_WGDC_Webinar_James_Duncan_Adoption_VermontMonitoringCooperative.pdf`

ESIP Data Preservation and Stewardship Committee. (2019, July). *Data Citation Guideline for Earth Science Data.* Retrieved from `https://esip.figshare.com/articles/Data_Citation_Guidelines_for_Earth_Science_Data_Version_2/8441816` doi: https://doi.org/10.6084/m9.figshare.8441816.v1

Fanelli, D. (2018). Opinion: Is science really facing a reproducibility crisis, and do we need it to. *Proceedings of the National Academy of Science*, *115*(11), 2628–2631.

German, D. M., Adams, B., & Hassan, A. E. (2016). Continuously mining distributed version control systems: an empirical study of how Linux uses Git. *Empirical Software Engineering*, *21*(1), 260–299.

Gößwein, B., Miksa, T., Rauber, A., & Wagner, W. (2019, September). *Data identification and process monitoring for reproducible earth observation research.* Proceeding of the IEEE Conference on eScience (EScience 2019). doi: https://doi.org/10.1109/eScience.2019.00011

Gupta, S., Zabarovskaya, C., Romine, B., Vianello, D. A., Vitale, C. H., & McIntosh, L. D. (2017, March). *Incorporating Data Citation in a Biomedical Repository: An Implementation Use Case.* AMIA Joint Summits 2017. Retrieved from `https://www.rd-alliance.org/system/files/documents/AMIASummit2017_BiomedicalDataCitation_Article_Final.pdf`

Huber, P. (2015). *Enabling Data Citation for XML Data* (Unpublished master's thesis). Faculty of Informatics at the Vienna University of Technology.

Hunter, J., & Hsu, C.-H. (2015). Formal acknowledgement of citizen scientists' contributions via dynamic data citations. In *Proceedings of the International Conference Asian Digital Libraries (ICADL 2015)* (p. 64-75). Springer.

International Organization for Standardization. (2003). *ISO Standard 14721:2003, space data and information transfer systems—a reference model for an open archival information system (OAIS).* International Organization for Standardization.

International Organization for Standardization, & International Electrotechnical Commission. (2011). *ISO/IEC 9075-2:2011, Information Technology — Database Languages—SQL—Part 2: Foundation (SQL/Foundation).* ISO Standard.

International Organization for Standardization (ISO). (2010, June). *Information and documentation - Guidelines for bibliographic references and citations to information resources.* Retrieved from `http://www.cmaph.org/attachment/201364/1370309271657.pdf`

Ma, X., West, P., Zednik, S., Erickson, J., Eleish, A., Chen, Y., ... Fox, P. (2017). Weaving a knowledge network for deep carbon science. *Frontiers in Earth Science*, *5*. doi: https://doi.org/10.3389/feart.2017.00036

McIntosh, L., & Vitale, C. H. (2017, January). *Adoption of the RDA Data Citation of Evolving Data Recommendation to Electronic Health Records.* Retrieved from `https://www.rd-alliance.org/system/files/documents/170117_RDA_WGDC_Webinar_Leslie_McIntosh_Adoption_WUSTL_Biomedical.pdf`

Moreau, N., Zwölf, C.-M., Ba, Y.-A., Richard, C., Boudon, V., & Dubernet, M.-L. (2018, Oct). The VAMDC portal as a major enabler of atomic and molecular data citation. *Galaxies*, *6*(4), 105. doi: https://doi.org/10.3390/galaxies6040105

Müller, H., Buneman, P., & Koltsidas, I. (2008, May). XArch: Archiving scientific and reference data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 1295–1298). ACM. doi: https://doi.org/10.1145/1376616.1376758

Ocean Network Canada (ONC). (2020, April). *Coming soon: MINTED dynamic citation tool.* Website. Retrieved from `https://www.oceannetworks.ca/coming-soon-minted-dynamic-citation-tool`

Olson, D., Meyerson, J., Parsons, M. A., Castro, J., Lassere, M., Wright, D. J., ... Acker, A. (2019). *Information maintenance as a practice of care.* Report. doi: https://doi.org/10.5281/zenodo.3236409

openEO Consortium. (2020). *openEO API.* API Specification, version 1.0.1, https://api.openeo.org. Retrieved from `https://api.openeo.org`

Parsons, M. A., Duerr, R. E., & Jones, M. B. (2019). The history and future of data citation in practice. *Data Science Journal*, *18*.

Prabhu, A., Morrison, S., Eleish, A., Zhong, H., Huang, F., Golden, J., ... Fox, P. (2020). Global earth mineral inventory: A data legacy. *Geoscience Data Journal*. doi: https://doi.org/10.1002/gdj3.106

Pröll, S., Meixner, C., & Rauber, A. (2016, October). *Precise Data Identification Services for Long Tail Research Data.* iPRES 2016. Retrieved from `https://www.rd-alliance.org/system/files/documents/iPRES2016-Proell.pdf`

Pröll, S., & Rauber, A. (2013, October). Scalable data citation in dynamic, large databases: Model and reference implementation. In *Proceedings of the IEEE International Conference on Big Data.* Retrieved from `https://ieeexplore.ieee.org/abstract/document/6691588` doi: https://doi.org/10.1109/BigData.2013.6691588

Rauber, A., Asmi, A., van Uytvanck, D., & Pröll, S. (2015, October). *Data Citation of Evolving Data: Recommendations of the Working Group on Data Citation (WGDC).* Retrieved from `https://zenodo.org/record/1406002#.XWKXzi1XZsM` doi: http://doi.org/10.15497/RDA00016

Rauber, A., Asmi, A., van Uytvanck, D., & Pröll, S. (2016, May). Identification of Reproducible Subsets for Data Citation, Sharing and Re-Use. *Bulletin of the IEEE Technical Committe on Digital Libraries (TCDL)*, *12*(1). doi: https://doi.org/10.5281/zenodo.4048304

Schubert, C., Seyerl, G., & Sack, K. (2019, August). Dynamic Data Citation Service - Subset Tool for Operational Data Management. *MDPI*, *4*. Retrieved from `https://www.mdpi.com/2306-5729/4/3/115` doi: https://doi.org/10.3390/data4030115

Silvello, G. (2017, November). *Automatically generating citation text from queries (Recommendation 10) - RDA Data Citation WG Webinar.* YouTube. Retrieved from `https://www.youtube.com/watch?v=Sf2bhx0kChA`

Stall, S., Yarmey, L., Cutcher-Gershenfeld, J., Hanson, B., Lehnert, K., Nosek, B., ... Wyborn, L. (2019). Make scientific data FAIR. *Nature*, *570*(7759), 27–29.

Task Group on Data Citation Standards and Practices, CODATA-ICSTI. (2013). Out of cite, out of mind: The current state of practice, policy, and technology for the citation of data. *Data Science Journal*, *12*(0), CIDCR1–CIDCR75.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., ... Mons, B. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific Data*, *3*, 160018–160018.

Zettsu, K. (2019). Transforming sensing data into smart data for smart sustainable cities. In *Proceedings of the 7th International Conference on Big Data Analytics (BDA2019).* Springer. doi: https://doi.org/10.1007/978-3-030-37188-3_1

Zwölf, C. M., Moreau, N., Ba, Y.-A., & Dubernet, M.-L. (2019). Implementing in the VAMDC the new paradigms for data citation from the research data alliance'. *Data Science Journal*, *18*. doi: http://doi.org/10.5334/dsj-2019-004'

Zwölf, C. M., Moreau, N., & consortium, V. (2017, March). *Implementing the RDA Data Citation Recommendations in the Distributed Infrastructure of the Virtual and Atomic Molecular Data Center (VAMDC).* Research Data Alliance (RDA). Retrieved from `https://www.rd-alliance.org/system/files/documents/170331_RDA_WGDC_Webinar_CarloMariaZwoelf_Adoption_VAMDCInfrastructure.pdf`

Zwölf, C. M., Moreau, N., & Dubernet, M.-L. (2016). New model for datasets citation and extraction reproducibility in VAMDC. *Journal of Molecular Spectroscopy*, *327*, 122 - 137. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0022285216300613` (New Visions of Spectroscopic Databases, Volume II) doi: https://doi.org/10.1016/j.jms.2016.04.009