Enabling the data FAIRness of version control systems

Cas Fahrenfort
casfahrenfort@gmail.com

November 26, 2019, 46 pages

Research supervisor: Zhiming Zhao, z.zhao@uva.nl
Host/Daily supervisor: Thijs Gillebaart, thijs@grasple.com
Host organisation/Research group: Grasple, https://www.grasple.com/

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Many kinds of different scientific data are being produced every day by research institutes across the globe. Scientists are interested in using this data, but often have difficulties when trying to obtain access to data that has been created and is stored by external organizations, due to incompatible data management standards. The Findability, Accessibility, Interoperability, Re-usability (FAIR) principles are guiding principles for scientific data management and stewardship, which have been developed to facilitate knowledge discovery by introducing common standards for human and machine interaction with data, utilizing Persistent Identifiers (PIDs) and metadata. Several technologies and services have been introduced which leverage these principles. However, all aforementioned standards, technologies, and services are intended for static data and do not provide adequate support for dynamic and evolutionary data, e.g. software source code, which is often managed by Version Control Systems (VCSs) such as Git and Subversion. This research investigated the current approaches to managing persistently identified data through VCSs and found them to be lacking in diversity of supported VCSs and persistent publishing systems, and proposed a novel system which allows for direct publishing of repositories from multiple VCSs to multiple, external publishing systems through a web-accessed interface. This initial idea has also been published as a poster in the 2019 eScience Proceedings [1], which originated from an industry problem posed by Grasple [2]. Additionally, at the end of the thesis, several assertions and conclusions about the state of the art of persistent publishing of evolutionary data, most notably software source code, are made which detail important problems that need additional solutions.

Contents

Chapter 1

Introduction

Environmental research infrastructures are often built on a large number of distributed observational or experimental sites, run by hundreds of scientists and technicians, financially supported and administrated by a large number of institutions [3]. Each one of these distinct Research Institutes (RIs) produces large amounts of important data used in research environments and publications, which are useful to many researchers across the world. For example, a researcher at an atmospheric observatory might need oceanographic data to create a prediction model for atmospheric movements. However, this required data is gathered by an oceanographic observatory and not easily accessible outside of the organization.

If such a situation occurs in which an environmental researcher requires data from a different RI for a study, they first need to find out whether the data exists, find out where it is stored, obtain access and/or access rights to it, transfer the data to their own system, and finally build an application workflow or data processing pipeline to utilize the data. Once their study is complete, they must publish some form of identification or citation for the data that they used, alongside the results. However, these steps are inherently time-intensive because they must be performed manually. Furthermore, they become even more time-intensive due to the lack of standardized data management approaches, leaving data consumers to repeat this entire process every time they require a data set from a distinct RI. Moreover, data structures are not standardized leading to interoperability problems when combining different data sets into a single workflow, requiring even more work to adjust the data to the desired format. Not only large research organizations but also the data industry is beginning to struggle with carrying out data-intensive tasks: studies have shown that up to 80% of the time of data scientists is wasted on *data-wrangling*, and up to 60% of industrial data projects fail [4]. Therefore, advanced data infrastructure and data management solutions are needed to alleviate these time-intensive tasks.

The Findability, Accessibility, Interoperability, Re-usability (FAIR) principles [5] [6] are guiding principles for scientific data management and stewardship, which have been developed to facilitate knowledge discovery by assisting humans and machines in their discovery of, access to, integration and analysis of, task-appropriate scientific data and their associated algorithms and workflows [7]. These principles have become increasingly important for the advancement of scientific research and industrial innovation because they are aimed at reducing time spent on data management and therefore enable researchers to focus their efforts on improving the research itself.

The FAIR principles specify ways to make data sets more Findable, Accessible, Interoperable and Re-usable. They emphasize machine-actionability since researchers are increasingly more reliant on computational support when dealing with data due to the constant increases in size, amount and creation speed. Making data easier to process by machines also reduces time spent on data management by data consumers. Increased amounts of data FAIRness make it easier to work with data from third parties, both for researchers and industrial workers. Applying these principles to existing legacy data storage systems can reduce the amount of time spent on locating the data (*Findability)*, provide clarity on how to access the data (*Accessibility)*, ease the combination of multiple datasets to obtain new insights (*Interoperability)*, and enable machine-assisted 'pattern recognition' in the data so that workflows can be re-used (*Re-Usability)*.

1.1 Data FAIRness

To enable all of these actions, data should be FAIR [5]:

- To be *Findable*, data sets should be described by rich, contextual metadata, should be assigned a globally unique and persistent identifier and be indexed in a searchable resource. Rich, contextual metadata should describe exactly what the data represents, who created it, where it was created, when it was created and what methods were used to create it. For example, the environmental researcher from the example above might need a specific oceanographic data set created at a certain time under specific conditions by a distinct instrumentation set. Being able to filter all available data by these criteria using the metadata greatly reduces the search results and therefore time and effort required to locate the correct data.

  Being able to identify and access a specific data set at all times is crucial for open research and innovation. For example, replication studies for verification of results need access to the same data as the original study and therefore be able to identify and locate them. Furthermore, proper attribution and citation of data sets used in research are important to allow people who invest time into creating important data sets to receive credit for their work and stimulate others to do so as well. Moreover, grant attribution is often based on (among other things) how many citations a given work receives, which implies its importance. For these reasons, globally unique, persistent and resolvable identifiers must be used when publishing data sets. Without such an identifier, accessing the data sets from an organization from outside is very difficult, because data management practices could be inconsistent between organizations. For example, one oceanographic observatory might use a different metadata schema than another, leading to being unable to find two cited data sets when using the same search query for both institutions. Furthermore, file names could be similar between the two institutions, causing an inability to identify exactly which data has been cited. Therefore, globally unique identifiers are needed which can only ever resolve to a single data set.

  To be able to put the above into practical action, the data and metadata should be indexed in a searchable resource, such as a data repository, which also provides a solution for resolving of identifiers to the correct data.
- To be *Accessible*, (meta)data should be accessible using a standardized communications protocol, so the users know how they can be accessed (including any possible authentication/authorization). After identification of the correct data sets, acquiring and using the data itself is often difficult due to distribution limitations because of rights and licensing issues. Furthermore, obtaining the proper authentication credentials to physically access the data can often take long if it must be manually approved by the data distributor. Therefore, standardized protocols to handle these issues are needed to allow them to be performed by machines and speed up the processes involved. Furthermore, persistent identifiers are crucial to harnessing the full potential of such protocols, as they provide a globally resolvable identifier to reason about the data for the instances involved in the process.
- To be *Interoperable*, (meta)data should use a formal, accessible, shared and broadly applicable language for knowledge representation, to be able to interoperate with applications/workflows for analysis, storage, and processing. If data representations differ between two systems, making them communicate and work with the data from each other is difficult and time-consuming. By utilizing commonly used, controlled vocabularies, automatic findability, and interoperation of data sets can be ensured. For example, JSON-LD [8] is a standardized, machine-readable data representation of Linked Data across the Web. These machine actions can be further developed by utilizing globally unique, persistent and resolvable identifiers to access data sets from within scientific applications and workflows.
- To be *Re-usable*, (meta)data should be richly described with a plurality of accurate and relevant attributes so they can be replicated and/or combined in different settings, including clear and accessible data usage licenses as well as detailed provenance information. For others to reuse your data, they should know where the data came from (i.e. a clear story of origin and history), who to cite and/or how they wish to be acknowledged. This principle is similar to the Findability principle, however, this one focuses more on the *legal* side of metadata, as well as making sure that data is actually useful for the data consumer.

FAIRness is not only important for basic, numerical datasets but also many more types of complex and dynamic data which are used for research purposes, most notably [3]:

- Instrumentation and sensors
- Workflows
- Algorithms

- Data pipelines
- Data models
- Software/source code

Each of these digital assets is used in the creation and processing of scientific data and represents some form of the encoding of knowledge. However, at the moment it is difficult to refer to these digital assets in scientific works because there are no standardized mechanisms in place to do such a thing. This creates a huge limitation when verifying and reproducing such works. For example, a researcher performs a study on oceanographic data collected by a specific sensor. Sometime later, a different researcher wants to perform a comparative study using data from the same sensor. If there is no unambiguous way to identify the utilized sensor, it might be difficult, time-consuming and even impossible for the second researcher to locate the sensor data and obtain access to it. Furthermore, the second researcher requires access to the data processing workflow used by the first researcher, to verify if the method used was correct and to ensure the same method was used in both experiments. Therefore, applying the FAIR principles and philosophy to these kinds of data is needed to ensure these digital assets are referable in a globally unique and persistent way, are described with rich contextual metadata and are indexed in searchable resources.

It is important to note that FAIR in this context does not mean *open*, as in openly accessible to everyone, in the same way, that e.g. the source code for open-source software is viewable and editable by everyone. While FAIR specifies standardized ways to interact with digital assets, licensing and access restrictions could still apply to data which is published in a FAIR way, which offers legal restrictions to who is able to use it and in what ways.

1.2 Problem Statement

One of the most important properties of digital assets that evolve over time, such as algorithms, workflows, data models and software source code, is their versioning. Each version must be distinctly marked and be described with sufficiently rich metadata, to provide a clear history and changelog fully cataloging the evolution of the digital asset, as well as its provenance details. Every company that manages these kinds of digital assets utilizes versioning of data in some way, some even using complex versioning systems. Version Control System (VCS) repositories are such a complex system for managing versioning, provenance, distribution, quality assurance and transparency of data, as well as providing an environment for collaborative work among both research and industry organizations. Such repositories often contain the entirety of the data for a single project or research effort and the systems that control them are the main method of interaction with this data.

However, the FAIRness of such repositories is currently not fully enabled. Although open sharing of software is commonly accepted and celebrated within the community (namely the open-source culture), such sharing is generally not done using unique-identifier based or FAIR enabled systems. For example, the currently most popular repository sharing platform is GitHub [9], which does not persistently store data using globally unique identifiers, nor does it offer rich, contextual, searchable metadata. Furthermore, the current approaches to software development and common workflows used therein have not been designed with FAIR in mind. For example, in an open education platform like Grasple [2], community collaborators must provide content through proprietary systems built specifically for this purpose, which are not generic and do not provide any standardized communication options, making interoperation with existing systems used by the community collaborators difficult. Utilizing FAIR-based systems, content providers could use their preferred working space to publish created content to a generic publication space. Only a proprietary application space is needed to process and display the content to users, which is obtained using standardized communication protocols. Creating generic spaces as described here not only enables the FAIRness of this platform (allowing standardized searching of, access to and usage of the open education data) but also lowers the barrier for creating FAIR-enabled data platforms.

Therefore, to enable the FAIRness of VCS repositories, data management solutions are needed to effectively manage the dynamic evolution, i.e. versioning, of the data and other digital assets they contain. These solutions must allow VCS repositories to be assigned globally unique identifiers and rich, contextual metadata to comply with the FAIR principles. Currently, very few VCSs have (partly) integrated solutions for assigning persistent identifiers to data set versions and publishing them to publicly accessible publishing systems. This thesis will research what options exist for persistent publication of VCS repositories, what options are needed but missing and try to provide technical solutions for this problem, thereby introducing FAIRness into the world of version-controlled software development.

1.2.1 Research Questions

To tackle this issue, we try to answer the following research question:

How can existing Version Control Systems be connected to persistent identifier-based data management systems?

To answer this question, we first try to answer the following sub-questions:

1. *What are the state-of-the-art approaches to enable data FAIRness of Version Control Systems?*
2. *How can digital objects in VCS repositories be persistently published with Persistent Identifiers (PIDs) and metadata?*

1.3 Outline

In Chapter 2 we describe the background, including technical background and related work, of this thesis. Chapter 3 describes the development of an abstract architecture for persistent publishing of VCS repositories. Chapter 4 shows the implementation of this architecture. Results are shown in Chapter 5 and discussed in Chapter 6. Finally, we present our concluding remarks in Chapter 7 and future work in Chapter 8.

Chapter 2

Background

2.1 Technical Background

2.1.1 Persistent Identifiers

When the World Wide Web was first introduced, its creator Tim Berners-Lee proposed to the IETF to use the Uniform Resource Identifier (URI) as the naming scheme for describing identifiers for content on the Web. The use of URIs was rejected by the IETF because it would not allow users to change the identifiers of Web content when they moved to another location. Therefore, Uniform Resource Locators (URLs) were chosen as the main identifiers for Web content. Although this way of retrieving Web content worked well during the early stages of the Web, several studies have found that approximately 50% of the URLs in scholarly publications fail to retrieve the digital content after a period of seven to ten years [10]. This effect is known as "link rot", a situation in which a URL fails to retrieve digital content because its location has been changed, and can be caused by several reasons [11]:

- The digital content has been moved to a different server and is no longer accessible through the given URL
- The URL is accessible but now points to another digital object
- The website registration has expired
- The web server hosting the digital object has been changed to a different web address
- The website directory structure is rearranged

When a user is unable to resolve a URL to specific digital content, they are often not given any information on the underlying cause. For example, if digital content has been moved to a different location, and its URL now resolves to nothing, users are often presented with a 404 error ("Page not found"). However, this page does not give any information on whether the content was moved to a new location, what this new location is, where an alternative to the content can be found or whether the content was effectively deleted from the Web. The user, therefore, has no idea how to respond to the missing content. This problem led to the need for persistent identification of digital objects.

PIDs are in essence a permanent reference to a document, file, web page or another digital object. Generally, PIDs are assigned and resolved by a service that is external to the data provider. The service assigns a persistent identifier to a data object on the request of a user, after which the service guarantees that whenever this identifier is resolved, it will provide some information about the object. Usually, this information contains access methods and metadata about the digital object but could contain any other kind of information. PIDs do not guarantee digital objects will be available forever, as the object might be deleted by the data host due to e.g. data management costs, licensing issues, data inaccuracies or any number of other reasons. However, if the object is deleted, the identifier should still be resolvable (to what is known as a "tombstone page"). Information about the data set can still be learned from this resolution, such as provenance, contextual information, possible redirection to updated versions and reasons for deletion.

PIDs often have different "namespaces" which can be used to point to different sub-resolvers, much like how URLs can have different sub-domains. These namespaces can be used to allow a single resolver to resolve multiple top-level PIDs to several distinct resolvers that have their own PID resolution system and separate repositories for storing data. This structure allows multiple different organizations to resolve data using a single PID scheme, as long as they agree to uphold the persistence policies set by

the top-level resolver [4].

Because of this need for persistent identification, the first PID systems emerged shortly after the introduction of the World Wide Web itself. Over the years, several well known PID schemes have been developed, each using a different approach and structure [3] [12]:

- **The Handle System (HS)** One of the earliest created PID systems, providing governance for top-level namespaces. However, besides this, it does not provide any obligations concerning persistence policies
- **Digital Object Identifier (DOI)** One of the most well-known PID systems due to its usage for citing scholarly literature. It makes use of the Handle System and uses its namespace "10.[sub-namespace]/"
- **Uniform Resource Name (URN)** Unlike the Handle System, URNs do not use a common resolver system. Therefore it is up to the user to know which resolver system to use, which has been a severe impediment against the uptake of this system
- **Persistent URL (PURL)** PURL does not separate between the identifier and resolving mechanism and has no single global resolving mechanism.
- **Archival Resource Key (ARK)** Initially, ARK was planned to enable decentralized resolvers. However, ARK now relies on local resolvers which ARK-issuing archives have to provide and maintain.

Several services provide the creation of a variety of different persistent identifiers, by allowing users to upload content which is stored in repositories and assigned a PID. Some of the most notable are:

- **European Persistent Identifier Consortium (ePIC)** Based on Handles, ePIC PIDs can be used for any data objects and data collections. Consortium members share services and APIs, meaning that if one center is out of order, other centers can still resolve and distribute PIDs [3].
- **DataCite** A non-profit, community-driven organization that provides DOI services for research data, including generation and allocation of DOIs and related metadata, and discovery services for research data.

2.1.2 Digital Objects

Kahn and Wilensky [13] posed that after the introduction of the Internet something essential was missing: the Internet Protocol specifies exactly how messages are sent between machines but do not specify what kind of meaning these messages have. They say that humans only want to exchange "meaningful entities", meaning entities that contain data that people want to talk about, work with, process, refer to, cite, etc. Therefore, Digital Objects (DOs) as meaningful entities being subject of exchange and processing needed to be developed since they are central for human and machine communication and the need to identify and describe them to enable interpretations by both humans and machines [4].

PIDs and metadata were the basic building blocks that started the discussions that resulted in the FAIR principles, and, as shown in Section 1.1, are a core part of FAIRness and its implementation. After the formulation of the FAIR principles in 2014, the Research Data Alliance (RDA) Data Foundation & Terminology Group started work on a core data model, which moved a step ahead of the FAIR principles by writing a model of how the different aspects need to be related, known as DOs [4]. Shown in Figure 2.1, a DO is represented by a structured bit sequence (data) being stored in a repository, is referenced by a PID and described by metadata.

2.1.3 Digital Object Architecture

Enabling the exchange of DOs and their processing faces some challenges. Data repositories are all different and do not have a unified method of storing data. They could use files, clouds, SQL, NoSQL or any number of storage solutions. Furthermore, the way data and metadata are related together can differ wildly between repositories. Therefore, proposals were made for a digital object access protocol [14] (Figure 2.2), which enables the exchange and processing of DOs in a way that is independent of how repositories organize their data, thereby bridging these differences and achieving interoperability at the level of data organizations, considerably reducing data-consuming costs [4]. For example, a researcher wanting to process data from two independent oceanographic observatories will only have to write a single data retrieval and processing workflow because these interactions are standardized across both organizations.

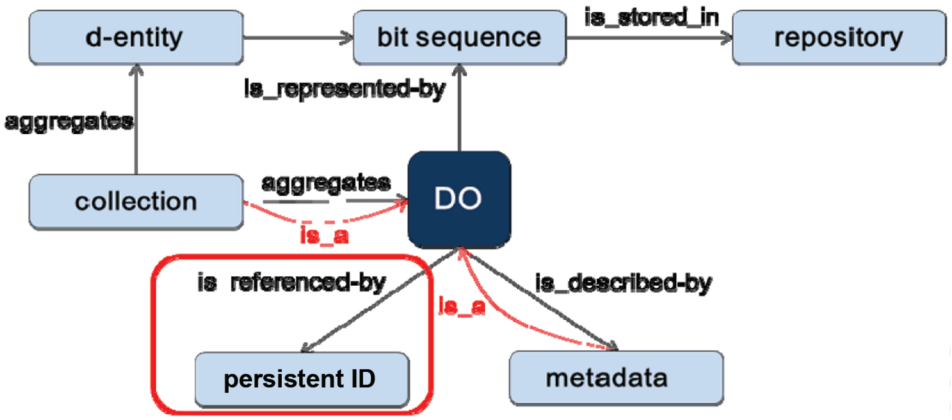


Figure 2.1: The notion of a Digital Object [4]

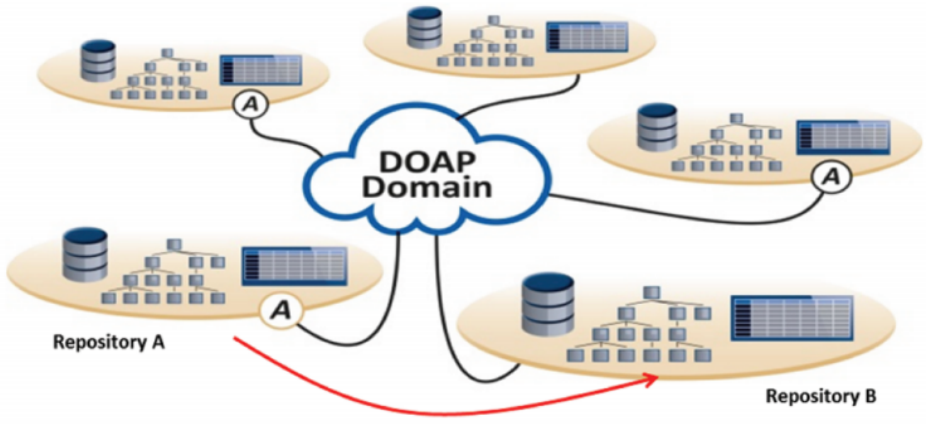


Figure 2.2: The concept of a Digital Object Access Protocol [4]

To enable a logical extension of the Internet architecture that addresses the need to support information management more generally than just conveying information in digital form from one location to another, thereby enabling interoperability across heterogeneous information systems, the Digital Object Numbering Authority Foundation [15] has developed the Digital Object Architecture (DOA) [16]. The DOA specifies two core protocols for communication and three system components for managing Digital Objects:

The **Identifier/Resolution Protocol (IRP)** is used for creating, updating, deleting and resolving DO identifiers. Each identifier is of the form prefix/suffix and is associated with an identifier record containing relevant "state information" clients can resolve to. The identifier's prefix is resolved to locate the specific identifier/resolution service to be used (the suffix may be any bit sequence). An organization may run its resolution system by having a prefix allotted to it.

The **Digital Object Interface Protocol (DOIP)** specifies a standard way for clients to interact with DOs. The DOIP makes use of the IRP for associating identifiers with different elements of the protocol. DOIP enables the provision of security by validating digital objects as well as ensuring integrity via signatures. Basic operation clients may invoke on the DO services are defined; the addition of operations is supported. Each DO must specify its type (which is extensible), enabling DOIP services to identify allowable operations. Each type is associated with an identifier record that can be accessed by the use of the IRP.

The **Identifier/Resolution System** provides the following services:

- allotment of unique identifiers to information structured as digital objects
- rapid resolution of identifiers to current state information of the digital object
- administration of the identifier records containing state information

The **Repository System** provides the provision of access to digital objects based on identifiers (integrated security), as well as abstracting away the details of storage technologies from the client, enabling a long-lived mechanism for depositing and accessing digital objects.

The **Registry System** stores metadata about digital objects which are managed by repository systems.

From a data consumer perspective, these systems and protocols combine into a single, virtualized layer on top of existing data storage systems, known as the Global Digital Object Cloud (Figure 2.3). Consumers only interact with logical representations of DOs and any organizational complexity is hidden away [4]. This means that, while data storage systems could become as complex as they desired, user interaction using standardized access methods and standardized DO representations, which include PIDs and metadata, does not change. Such a structure enables high levels of accessibility, interoperability, and re-usability of stored data. Combining this with the registry and repository systems of the DOIP, a high degree of findability is obtained as well, due to the standardized methods of accessing and searching repositories using standardized communication protocols and metadata. Therefore, a high degree of FAIRness can be achieved by combining these technologies.

2.1.4 ENVRI-FAIR

Several projects are now underway to combine the previously mentioned technologies and utilize them for practical research applications. For example, ENVRI-FAIR is a recently launched project of the European Union's Horizon 2020 program (EU H2020) that connects the cluster of European Environmental Research Infrastructures (ENVRI) to the European Open Science Cloud (EOSC), and aims to provide a set of interoperable FAIR data services for all participating Environmental Research Institutes that enhance the efficiency and productivity of researchers, support innovation and enable data- and knowledge-based decisions [17]. They make use of a roadmap defined by the European Strategy Forum on Research Infrastructures (ESFRI). This forum maintains a roadmap for the need for additional RIs in Europe, as well as the needs of already existing ones, for the next 10-20 years, stimulating implementation and improvements of these facilities [18].

ENVRI-FAIR wants to enable the FAIRness of all RIs mentioned in the ESFRI roadmap by utilizing and combining all previously mentioned technologies, standards, and philosophies. Their methodology includes [17]:

1. Defining community policies and standards across all stages of the data life cycle, aligned with the wider European policies and with international developments.
2. Creating sustainable, transparent and auditable data services for each stage of the data life cycle, following the FAIR principles, for all participating RIs.

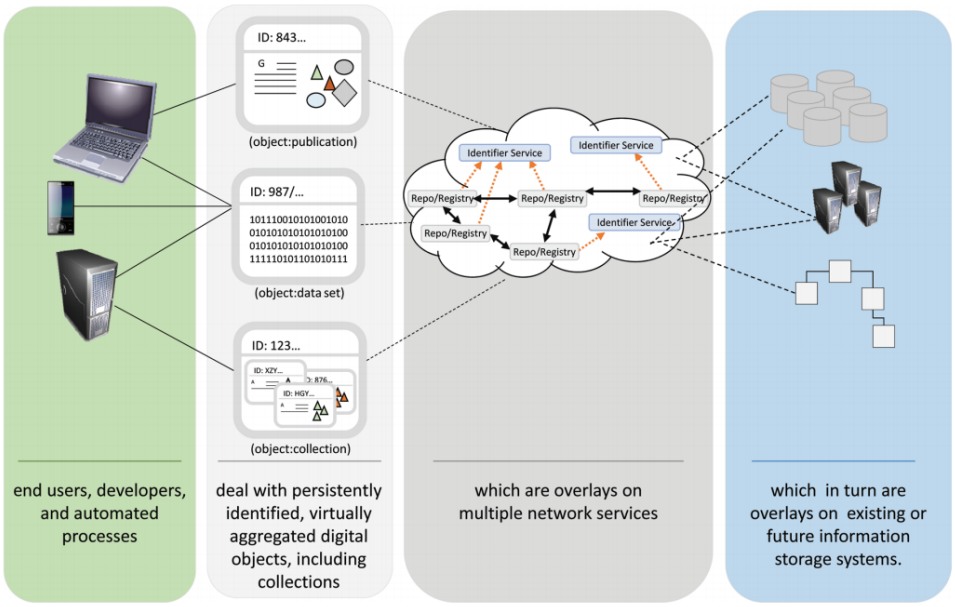


Figure 2.3: The Global Digital Object Cloud [4]

3. Implementing prototypes for testing pre-production services at each RI, leading to a catalog of prepared services.
4. Exposing the complete set of thematic data services and tools of the ENVRI cluster to the EOSC catalog of services.

The ultimate ambition of ENVRI-FAIR is establishing the foundations of a virtual, federated machine-to-machine interface that provides access to environmental data and services from contributing RIs. This interface (known as the ENVRI-hub, see Figure 2.4) is to be realized as the services across RIs and even between environmental sub-domains become progressively more integrated. By following a modular design, existing services can be re-used across RIs when developing new ones. EOSC services such as Authentication and Authorization Infrastructure, data storage solutions and catalog of services are available to all users to enable the seamless integration of ENVRI data and services into the EOSC. For users that require a broader or full spectrum of environmental parameters, the ENVRI-hub will offer a single platform that hides the complexity and diversity of the ENVRI landscape, while preserving the structure of the individual RI and continuing to fulfill the requirements of their designated communities [17].

2.1.5 Named Data Networking

Another technology that is being developed concerning the FAIRness of digital objects is Named Data Networking (NDN). NDN [19] is an implementation of the Information Centric Networking (ICN) paradigm, which is founded upon the idea that users are interested in accessing data irrespective of their locations. Many data-intensive applications are reliant on data that are either distributed over a large number of different data providers (e.g. composing and collaborative editing of online education or training material using community-contributed media) or being collected by distributed sources (e.g. environmental observations from different geo-regions). When the sizes of these data objects increase, the performance of the object sharing is critical for user experience.

While the FAIR principles specify a lot about data storage and management, they do not talk about the distribution of digital objects. Centralized storage of data objects often creates performance bottlenecks when a large number of distributed users tries to access the objects simultaneously, or when a single user tries to access a large number of data objects. Object transfer speed is not only limited by the services providing the data, but also by the network it is transferred over. During processing and analyzing of data objects, data movements are frequent, especially when an application uses multiple resources across multiple locations. Delivering large data volumes over traditional IP networks causes severe delays to the execution of such scientific workflows because the Internet's point-to-point underlay is poorly matched to the data-centric overlay needed by data-intensive applications [20]. Therefore,

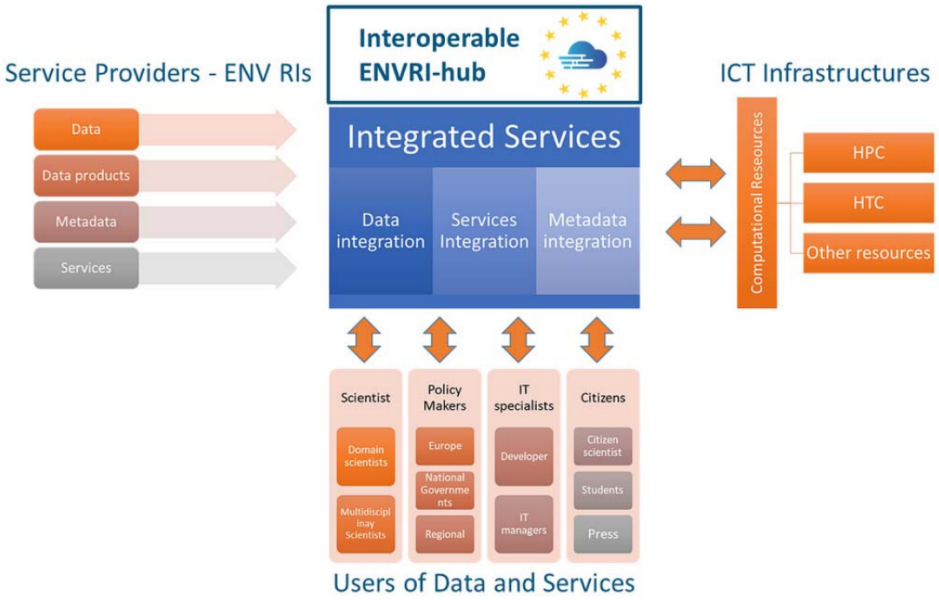


Figure 2.4: The ENVRI-hub Architecture [17]

technologies that allow such efficient data transfer are needed in combination with FAIR-enabling data management techniques to fully enable scientific progress and innovation.

In an NDN network, data is routed between source and destination based on unique names of data objects and cached at intermediate locations to enable efficient distribution. Each digital object present in the network has a unique name, and when an object passes a network node, its name and content are stored in the cache, so that whenever the same object is requested again, it can be immediately delivered from the cache. If an object is not present in the cache, the network node it is requested from uses its internal routing protocol to find the next node on the path to the source of the digital object, until a node with the object in its cache or the object's source is found. The object is then returned to the original node by tracing its path backward. To integrate such a network with FAIR enabled technologies, authors of [20] have proposed an architecture for distributing DOs utilizing PIDs as defined in the DOA via NDN (Figure 2.5). This architecture consists of

1. PID2NDN gateway, which resolves PID names to NDN names
2. NDN4PID router image, NDN node implementing a virtualized NDN router
3. NDN4PID manager, automating the management of the virtualized NDN overlay on other cloud or network infrastructures

Figure 2.5 also illustrates how such an NDN network would be able to be combined with DOIP technologies, most notably the Repository and Registry systems. The PID2NDN interface acts as a bridge between the DOIP system and NDN network nodes, by translating PIDs to NDN names and vice-versa, thereby allowing the transfer of digital objects from the DOIP repositories to the NDN network. Different mapping architectures for resolving PIDs to NDN names and vice-versa are being developed to optimize the efficiency of such data distribution [21].

2.1.6 Version Control Systems

Data that are both changing frequently over time and being collaboratively worked on by multiple parties at the same time are often managed by a VCS. Popular systems include Git [22], Mercurial [23], Subversion (SVN) [24] and Concurrent Versions System (CVS) [25]. These systems allow users to edit and synchronize their files while preventing losing any work when the same files are edited by multiple people at the same time, by recording changes and employing different strategies to merge files when both parties are finished editing.

A VCS typically defines a repository in which the files are stored and maintained. This repository is often accessible through a web URL, and the VCS software allows for manipulation of the repository through commands such as *clone* (copy the repository from the remote server to a local machine), *push*

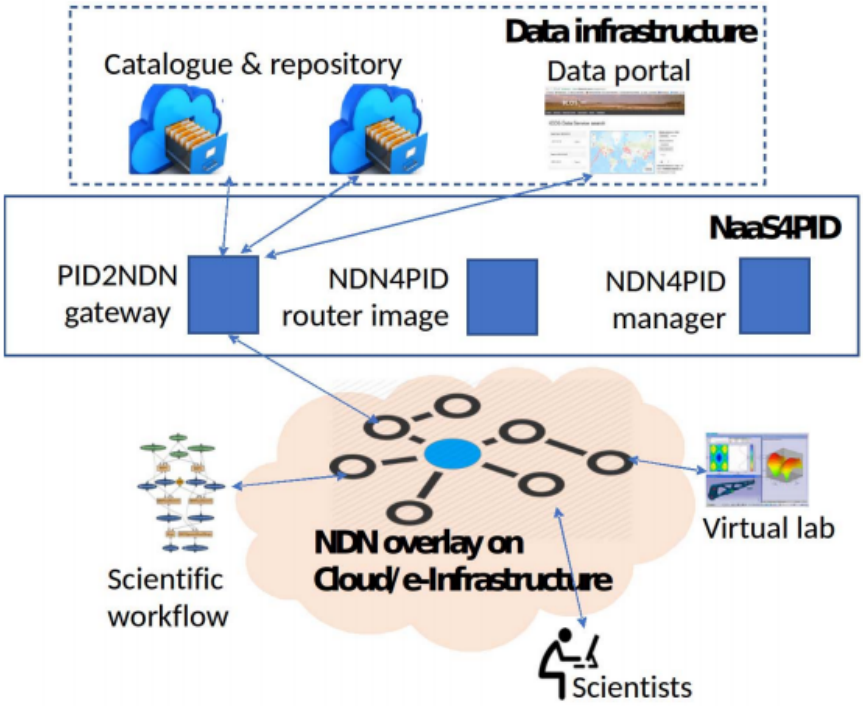


Figure 2.5: NDN-as-a-Service for PID architecture [20]

(send new changes to the remote server) and *pull* (check for and obtain new changes from the server), among others. Each VCS has its distinct terminology for different versions of the repository (e.g. *commit* in Git and *revision* in SVN). In this thesis, the term *snapshot* will be used to indicate such a repository version.

Online platforms such as GitHub [9] enable users of VCSs to easily share their open-source repositories with and receive contributions from other users. Such platforms often provide workflow integrations for these contributions, such as *pull requests* which enable a contributor to request their version of a repository to be merged into another one, and *issue trackers* which allow anyone to open discussions about any issues they experience with the contents of the repository, or other topics such as possible improvements. Other, more specialized, workflow integrations also exist, such as automatic building and deployment of source code contained within repositories (usually seen in DevOps-enabled platforms). However, at the moment it is not yet possible to assign a DOI or any other PID to software code or packages using solely GitHub features [3]. In general, interoperability between VCS repositories and persistent publishing technologies is very poor. Even though almost every piece of software being developed right now utilizes some form of version control, not only for industry but also for research efforts, standardized methods for enabling the FAIRness of software source code, or any other kind of data, developed in such a way are still in a state of infancy. However, a small number of FAIR data storage systems have developed some form of interoperation with existing VCSs and VCS platforms.

2.1.7 Data Storage Systems

Several persistent data storage systems that implement the ideals of the FAIR principles and the DOA in varying degrees have already been developed and are operational. These services are generally Web-based Application Programming Interfaces (APIs) that expose endpoints to upload binary file data along with metadata properties specified in JavaScript Object Notation (JSON) standard, after which a record will be created containing the file (meta)data and also an assigned PID. Furthermore, they provide both manual and machine-enabled searching for metadata. Each system also provides additional operations to be performed on the stored records, e.g. file downloads, record deletion, and record versioning. For the purposes of this thesis, several of these systems, which are freely accessible to everyone to both store and browse data, have been selected and will be discussed in more detail below.

EUDAT

EUDAT [26] addresses the needs of researchers, research communities and infrastructures who are producing or using very large data sets for research purposes ensuring that the data is managed and stored in a secure and professional and persistent manner. They provide an extensive service catalog for persistent data storage aimed at a variety of use cases, both at an individual and organization level. The platform is focused on being easily integrable with existing data and providing straightforward methods for the harvesting of uploaded (meta)data.

Their B2SHARE service is an open RESTful API that can be utilized for creating data records, supplying metadata and uploading binary files. Each record is assigned a unique, Handle-based ePIC identifier, and the long-term persistence of data is guaranteed.

Dataverse

Dataverse [27] is an open-source web application and data repository which allows researchers to "share, preserve, cite, explore, and analyze research data" [3]. It is open to researchers in all disciplines throughout the world. Both DOIs and Handle system PIDs are supported and can be provided by either DataCite or EZID (a service for minting DOIs and ARKs offered by the California Digital Library). The open-source status allows any organization to host a Dataverse repository.

A Dataverse repository is the software installation, which then hosts multiple virtual archives called Dataverses. Each dataverse contains datasets, and each dataset contains descriptive metadata and data files (including documentation and code that accompany the data). As an organizing method, dataverses may also contain other dataverses [27]. The Dataverse software exposes an HTTP API that can be used to create and manage datasets. The Harvard Dataverse is open to all researchers worldwide in all disciplines [28].

Zenodo

Zenodo [29] is an online open science platform and data repository developed within the OpenAIREplus project, with CERN as the major developer [3]. They accept any research data from any discipline in any kind of format, and anything uploaded is assigned a DOI, making data sets easily and uniquely citeable. Uploaded material is curated for display on the front page, to further facilitate openness of data. They also allow for the creation of data collections and accepting or rejecting uploads submitted to it, making the maintenance of these data sets a community effort.

Zenodo offers to link a Zenodo and a GitHub account, allowing users to easily upload anything hosted in a Git repository, whether it be data or software. This further facilitates finding data and source code that has been and should be used together [3].

figshare

figshare [30] is a repository for sharing a wide range of research output types, including datasets, software code, and posters. They guarantee at least 10 years of persistent availability of uploaded resources, as well as the assignment of a DOI to each upload. In the spirit of linked open data, GitHub and figshare accounts can be connected to facilitate finding data and the software that has been used to generate process, or analyze the data, and might be needed to understand and use the data. The linking of the two repositories was an early initiative to link resources stored in different repositories and was a collaborative project between GitHub, Mozilla Science Lab, and figshare [3].

2.2 Related Work

As evident from Section 1.1, persistent identifiers are one of the core aspects of enabling data FAIRness. The RDA has published many sets of recommendations for the management of FAIRly published data and best practices regarding this topic. Although a lot of work has been done regarding FAIR data management, PIDs and metadata, this is still an area of research that is in early stages and therefore many aspects are still not clearly defined or do not have an overwhelming amount of agreement among community members. This section will highlight some of these recommendations regarding PID usage, which is particularly relevant when examining the transfer of data resulting from common VCS repository actions, including versioning, releasing data subsets and managing dependencies, to FAIR-enabled data storage systems.

2.2.1 Versioning of PID-enabled DOs

One of these RDA publications is a set of consolidated assertions about the nature, creation, and usage of PIDs, compiled by experts from 47 European research infrastructure initiatives [31]. The main purpose of this publication is to identify agreements across existing documents that have been suggested to be included by experts. It states that PID service providers have different practices when assigning PIDs regarding the issues of granularity and versioning, and most do not make clear which policies they apply, particularly in respect to the recommended timing of assignment and granularity needed. A few of the collected assertions are:

1. Repositories need to clearly state which policies they follow in terms of granularity, versioning, time of PID assignment, binding, etc.
2. Previous and subsequent versions can be indicated as machine-readable types in the PID record. If standardized, typed attributes are used to refer to previous and subsequent DO versions, machines can interpret and process this information to increase efficiency.
3. Implement a version-management policy: either the change history of the digital object needs to be documented, or the identifier should be versioned, or both should be done. If a resource is removed, the PID should resolve to a "tombstone" page, containing still relevant information about the object and reasons for deletion.
4. Manage complex life cycles without deletion (of PIDs). Generated and publicly advertised PIDs must never be reassigned to a different resource or be deleted. There exist differences between static records such as experimental data and dynamically evolving records such as software source code or concept descriptions (which are likely to be versioned).
5. PIDs should be assigned to the smallest subsets of scientifically meaningful data that is practical to refer to, to optimally support citation and later re-use. This has a direct consequence on the granularity of data versions.

Furthermore, this publication states that there is an ongoing debate on the best solution for the DO versioning problem and how PIDs could be used efficiently. Some will assign PIDs for each new version of a digital object, in which case the PID registry should contain the history of a given digital object, allowing end-users (humans or machines) to traverse the DO version history if needed. Others allow versioning of DOs but do not assign a new PID, and instead append a version indicator the end of the PID or utilize some other form of DO version management.

Another RDA recommendation provides a discussion on PID information types [32], and it also examines what should happen when a new version of a DO is created, whether to assign a new PID and how to connect the new and old PID. They specify three different sub-cases for this situation:

1. Old data "A" must be replaced by new data "B". In this case, the old, no longer useful data A must be erased and its new version, data B must be stored. The RDA recommends to assign a new PID to B, and link the PIDs of A and B in some way (e.g. via metadata). A can optionally be deleted and replaced with a tombstone page.
2. A user wants to access the latest version of the object "A". In this case, the user tries to access object A via its PID and is interested in retrieving the latest version (as the object is versioned in a way similar to sub-case 1). The system accesses A and determines its latest PID to object "Z" (through the links made by the system in sub-case 1). It then responds to the user with Z.
3. A user wants to access a specific version of object "A". They request the system to retrieve A, and the system either replies with A if it is available or with the fate of A if it is not. Optionally, PIDs for predecessors or successors are returned. Also optional, the system may enquire from the user whether a newer version should be returned if available.

ENVRIPlus, an EU H2020 project (like ENVRI-FAIR mentioned in Section 2.1.4), published a document [3] bringing together Environmental and Earth Systems Research Infrastructures in order to 1) promote cross-fertilization between infrastructures, 2) implement innovative concepts and devices across RIs, and 3) facilitate research and innovation in the field of environmental science for an increasing number of users outside the RIs. This document details the state of the art regarding data practices among environmental RIs and defines requirements and technologies which will be needed in the coming years. From a two-to-five year analysis of the state of the art, this document concludes that a majority of (starting-up) RIs adapt data curation strategies that are fully capable of handling dynamic data (both versioned static files and truly dynamic databases), centered around persistent identifiers for both data

and metadata objects and queries. However, versionable databases are needed to support "time machine" retrieval of large datasets (also sensor data) that are dynamic. Some solutions in this area already exist, but in general, well-planned and optimized data schemas are needed for storing transactions and their timestamps to prevent significant losses in performance when generating and retrieving many versions of stored data sets which are dynamic. For data stored as flat files, it is mainly the metadata which must be stored in a database supporting versioning functionalities, to allow identification of what files represent the current state at the time.

Another RDA recommendation has been published on the citation of evolving data [33], focused on ways to create identification mechanisms that (a) allow identifying and citing arbitrary views of data, from a single record to an entire data set in a precise, machine-actionable manner; (b) allow citing and retrieving that data as it existed at a certain point in time, whether the database is static or highly dynamic; and (c) is stable across different technologies and technological changes. This publication recommends to build a data store which also stores the queries that are executed against it. All data stored inside should be versioned, and operations on this data must be timestamped. The stored queries are also timestamped and assigned PIDs. This approach allows identifying, retrieving and citing the precise data set with minimal storage overhead by only storing the versioned data and the queries used for creating the data set.

2.2.2 Compositing multiple PID-enabled DOs

Generally, a PID is assigned to a single DO. However, in some cases, it might be desirable to assign a PID to a collection of objects or create an object which consists of multiple sub-objects (a composite object). The RDA recommendation on PID information types states: No consensus exists regarding the level of granularity at which PIDs are assigned to data objects. Different usage scenarios require different granularities, and thus PIDs must become hierarchically structurable. If both individual objects and the larger composite receive PIDs, then these implicit relations should be discoverable for humans but also machine agents that for example copy or analyze objects [32].

An example use case is provided in which a user wants to create a composite object. In this recommendation, users provide a list of PIDs to the system which then creates a new object that is assigned a new PID and contains references to the objects that are composited. Should this composite object be retrieved by a user, the list of composited PIDs is returned. If a regular object is retrieved, a list of all super objects is returned to the data consumer. It is noted that the example use case assumes that there is no previously existing super object with an already assigned PID and that the PID created for the composite object does not point to a distinct object itself. This may not be the case for every data community. Furthermore, some communities may have data collections with a large number of elements, rendering the discoverability of this kind of super object impractical due to the number of required links and PID operations. Moreover, hierarchies with multiple layers, where an object is simultaneously an element and a super object can further complicate things.

2.3 Gap Analysis

Many issues regarding generation and assignment of PIDs and storage of DOs do not have a clear consensus among field experts. For example, as discussed in Section 2.2, one RDA recommendation states that PIDs should be assigned to the smallest subset of scientifically meaningful data [31]. However, another recommendation states that different usage scenarios require different granularities, and there can be no pre-defined level of granularity [32]. Because of these lacking points of consensus, these problems concerning versioning of DOs, which is an area of work with little exploration done so far, do not have clear, viable solutions or paths towards one. To create such solutions and thereby getting closer to achieving consensus about these topics, more exploratory system design needs to be done.

While several tools and services for creating and managing persistently identified datasets have been discussed in Section 2.1.7, options for FAIR-enabled publishing of VCS repositories are very limited. Only two of these tools, Zenodo and figshare, allow users to directly publish VCS repositories, assigning a PID to them. Both services allow users to link their GitHub account to instantaneously publish a repository from its GitHub page. However, GitHub only hosts Git repositories, and no other VCS is supported. If a different VCS is used, or the repository is not hosted on GitHub, the services are unable to be utilized at all. This is a major limitation for many researchers wanting to use the systems, as currently many different VCSs are appearing which are specialized for specific research disciplines, e.g. DVC [34] for machine learning and Pachyderm [35] for data science. Therefore, to facilitate as many data

consumers from varying fields of research to assign PIDs to data sets from version control repositories, multiple Version Control Systems need to be supported by such tools.

Furthermore, Zenodo and figshare both publish all data objects to their repository and make it accessible through their platform. However, this platform is certainly not the most fitting for every dataset which is desired for publication. For example, datasets containing long series of high-frequency sensor data can benefit from being stored using database solutions that support versioning of database records, for "time-machine" like extraction of data without much loss in performance. For data stored as flat files, it is mainly the metadata that must be stored in a database supporting versioning, to allow identification of what file(s) represent the "current state" of the data at a given point in time [3]. Benefits in both performance, as well as ease of discovery and usability of the data objects, can be gained by storing such datasets using specialized data storage services. Therefore, solutions for assigning PIDs to version control repositories should not rely on a single form or provider of data storage, especially because of the varying range of research disciplines, and therefore data storage requirements, that make use of these tools.

Chapter 3

Architecture

3.1 Requirements

Section 2.3 details current problems regarding FAIR-enabled publishing of VCS repositories. To take an initial step towards a definitive consensus about and solution for this problem, this thesis proposes a system which allows publishing of repositories from multiple VCSs to multiple data publishing systems, to enable as many researchers as possible to persistently publish, share and cite their research data and software. Users of such a service will have certain expectations and needs regarding the provided functionalities. From literature and sources discussed in Chapter 2, the following key requirements for this service have been gathered:

1. **Interactive VCS Content Composition** The solution should provide a means for users to interactively compose their desired content for publication, using an existing VCS repository, which should be provided by the user. The users should be able to manage all content composition within the system itself.
2. **Multiple Publishing Service Support** The system must offer means to publish the desired repository content to a number of different persistent data publishing services, as well as handle accessibility and authorization of the desired publishing system.
3. **Metadata Standard** Data citation methods must be flexible, which implies some variability in standards and practices across different scientific communities. However, to support interoperability and facilitate interpretation, the citation should preferably contain several metadata elements that make the dataset discoverable, including author, title, publisher, publication date, resource type, edition, version, feature name, and location. Finding standards for citing subsets of potentially very large and complex datasets poses a special problem, as e.g., granularity, formats and parameter names can differ widely across disciplines [3].

   As the goal of this tool is to provide researchers from many disciplines with an easy method of persistently publishing version control repositories, the metadata should reflect this. Every data publishing system has its own metadata standard, and although there is a lot of overlap between standards, the tool still needs to bridge the gaps between them. When publishing a repository to one of these data publishing systems, their metadata standard needs to be respected. Therefore, a single metadata standard is needed which can be mapped to each of the target metadata standards while losing as little information granularity as possible. Alternatively, support for different metadata schemas based on the targeted publishing system could be given.

Some of the most important additional requirements are:

4. **Data Guarantees** Persistent publication of digital objects is to a large extent dependent on the publication service provider. Because the published data can only be accessed through the publication service, it is important to consider what kind of guarantees data publishers give about published data before making a publication. Not only guarantees about the retention period of the data are important, but also aspects such as functional preservation, fixity, authenticity and (meta)data access should be taken into consideration. Since the proposed tool offers persistent data publication, users will expect to find similar guarantees.
5. **Data Subsets** A majority of the RIs feel it is necessary to allow unambiguous references to be made to specified subsets of datasets, preferably in the citation, while few find the ability to create,

identify, and later cite collections of individual datasets is important [3] [31]. This highly specific attribution is needed because performance indicators that can be obtained from these citations are used by funding agencies and other stakeholders. Therefore it is important for the tool to make possible not only the publishing of full repositories but also individual subsets.

6. **Data Dependencies** Datasets and the software used to generate them are often maintained in separate repositories. However, when citing such data, it is often desired to include the software used, for verification and reproduction of the datasets. Therefore, the tool should provide an easy method for linking multiple publications together.
7. **Transactional Nature** When dealing with any process which has permanent effects, it is important to consider the effects of each step of the process, and what happens when the process does not complete in its entirety. The process of publishing a repository through the tool involves multiple steps, among others the creation of a new record, setting metadata, uploading files and finalizing the publication. Should any step of this process fail, any lasting side effects are undesirable. Therefore, each operation performed by the tool should be transactional, i.e. treated as a single unit of work.
8. **User Interface** Surveys have indicated that the reasons for the slow adoption rate include a lack of knowledge about the existing opportunities, confusion over their relative differences and merits, and difficulties related to the identifier creation process (especially when it needs to be performed on a large scale, as is often the case for data). The latter problem is to a large extent due to the large variety in the design and functionality of PID registry user interfaces and APIs [3].

   Because the target audience of the tool are researchers from varied disciplines, and different publishing systems and their APIs are targeted, it is critical to have an easy to understand interface that clarifies each step of the process in as basic terms as possible.
9. **Performance** For any data management system, performance is a cornerstone of effectiveness, usability and general user experience, especially when dealing with large datasets. Performance in this situation is not trivial to define and can have many different characteristics. However, it will be important to compare the performance to existing systems like Zenodo and figshare to ensure that, to those users to whom the features of the service are more applicable than existing systems do not stay with their current solution because of lack of performance.

3.2 Architecture

3.2.1 Components

To establish the necessary components for such a publication tool, a possible workflow (seen in Figure 3.1) for interaction with the tool is examined. This workflow shows the need for the following components:

- Interface component to handle user interaction with the tool
- Abstract component to obtain snapshots from multiple Version Control Systems
- Abstract component to publish data to multiple data publishing systems
- Connecting component to process publication request using the other components

The proposed architecture including these components is shown in Figure 3.2. In step 1, a user supplies repository information (repository location, choice of VCS and metadata), publishing environment information (which publishing system to use, access information) and publication metadata. In step 2, the publication request is supplied to the main processing component. In step 3, the abstract snapshot component obtains the snapshot from the external VCS repository. In step 4, the obtained snapshot data is published to the external publishing system. The PID obtained from the publishing system is then returned to the user.

An additional component, the publication database, is present in the system. This database holds a record for each snapshot that has been published by the tool. Between steps 3 and 4, the obtained snapshot is checked against the database to see if that particular snapshot has been published before. If so, the location of the previously published digital object is returned to the user immediately. If the snapshot has never been published before, a record is inserted into the database after step 4. This component is added for performance considerations (avoiding having to do unnecessary work multiple times) and to reduce unnecessary clutter in the publishing system's repository.

With these components, the following requirements from Section 3.1 are (partly) satisfied:

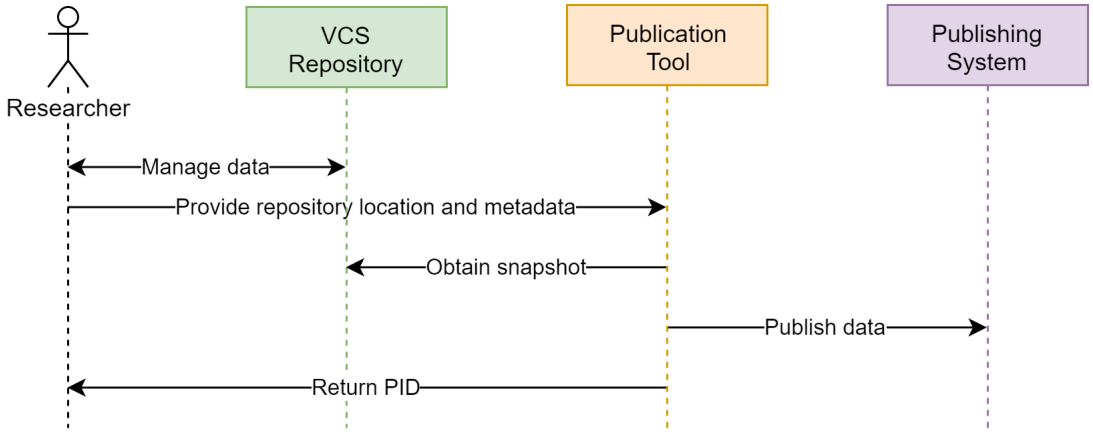


Figure 3.1: Data publication workflow using the proposed tool

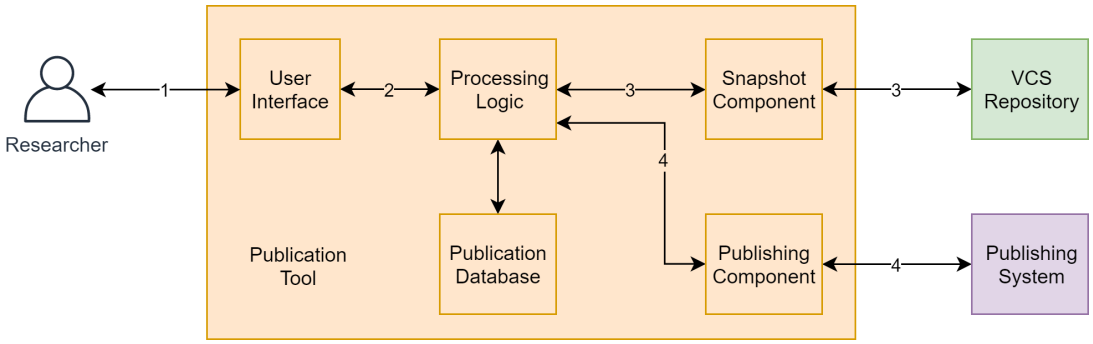


Figure 3.2: Architecture for a prototype tool for persistently publishing version control repositories

1. **Interactive VCS Content Composition** During step 1, the user can provide details about their repository and interactively compose the desired data subset and possible repository dependencies to be published.
2. **Multiple Publishing Service Support** During step 4, the abstract publishing component can be adapted to interface with any possible publishing system that provides HTTP communication.
3. **Metadata Standard** The user can provide metadata at a single point which can be mapped to all possible publishing systems during step 4.
4. **Data Subsets** During step 3, the snapshot component can obtain any arbitrary amount of data from the VCS repository, and therefore it is possible to obtain any possible data subset. However, the granularity of the possible subsets is determined by the options provided by the user interface.
5. **Data Dependencies** If enabled by the user interface, the user could provide information about multiple repositories, and the processing component could combine the snapshots into a single publication.
6. **User Interface** The user interface is an integral component of the proposed architecture.
7. **Performance** The publication database is a performance-enhancing aspect of the proposed architecture. However, most performance characteristics will come from the implementation rather than the architecture.

Persistent digital object publishing systems, as defined in the DOA [16], have two core components: the registry system and the repository system. In the architecture proposed above, the external publishing system contains both the registry and repository systems. However, should the implementation of this architecture gain enough popularity to warrant its evolution into a full-fledged service utilizing its own storage and identifier solutions, it could be achieved by simply replacing step 4 with proprietary components.

3.2.2 Constraints

There are several constraints which must be kept in mind while implementing the proposed architecture:

1. **VCS Abstraction** The intention of the architecture is to allow the system to interoperate with multiple Version Control Systems. To do this effectively, the VCS needs to provide an interface exposing commands to allow for repository manipulation. As a result, only VCSs offering a sufficient abstract interface that makes the required operations possible can be supported.
2. **Publishing System Interoperability** The second main intention of the architecture is to allow the system to interoperate with a multitude of external publishing systems. Therefore, the technology chosen to implement the architecture should be able to communicate with all or most publishing systems.
3. **Accessibility** The tool must be easily accessible by a large part of the target user group. Its effectiveness will in large part be determined by how and by whom it can be accessed. This is an important consideration when deciding what technologies to use for implementing the system.

Chapter 4

Prototype

To further analyze and explore the requirements of the architecture described in Section 3.2, a prototype has been developed. This chapter will cover the design choices and implementation details of the prototype.

4.1 Technology Considerations

As described in Section 3.2.2, it is important to consider communication standards for existing publishing systems. When looking for such systems, nearly all of the publicly accessible ones communicate over HTTP using RESTful API standards. Therefore, to enable seamless integration, the same technology has been used for building the prototype. Furthermore, such APIs are easily made available over the Internet, allowing anyone in the world to reach them, enabling high degrees of accessibility. Moreover, exposing a user interface over the Internet can easily be done using HTML and JavaScript, which also allows for communication with the API over HTTP. Using these technologies, the interface can be hosted using the same cloud technology as the API. By separating the user interface and processing logic/data storage concerns, the portability of the user interface across multiple platforms is improved and scalability is increased by simplifying the server components.

An alternative option would have been to distribute the tool as a stand-alone software package, including both the interface and the underlying processing logic. However, the extra steps needed to download and install the software are unnecessary, as all functionality can easily be implemented on cloud-based systems. Furthermore, cloud technology allows (in most cases) for better performance than user machines, both in processing power as well as scalability. Moreover, there are no performance benefits to distributing the service this way, as the web-based user interface runs in the browser on the user machine, meaning no extra delay is introduced when interacting with it.

4.1.1 Platforms

The API itself has been developed using ASP .NET Core, a cross-platform, high-performance, open-source framework for building cloud-based, Internet-connected applications [36]. ASP .NET Core targets the .NET Core framework and is developed in the C# programming language. This framework has been chosen because it contains many user-created code libraries for executing commands on a multitude of VCS repositories.

The user interface and API communication have been implemented using Angular, an open-source framework for developing cross-platform, highly scalable web applications [37]. Angular is developed in the TypeScript programming language, which is a strict syntactical superset of JavaScript. This framework has many pre-conceived features to make developing interfaces and HTTP interactions fast and simple.

Both the API and user interface are hosted using the Azure platform [38]. Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. Azure has been chosen because it allows both hosting of static webpages (the user interface) simply by uploading the files and because applications developed in ASP .NET Core can be published to the Azure App Services platform with a single button click. These features allowed for the rapid development of the prototype.

4.2 Implementation

4.2.1 Version Control Systems

The prototype supports two version control systems: Git and SVN. These systems have been chosen because, according to the 2018 StackOverflow Developer Survey [39], they are the two most popular version control systems in use today. Furthermore, they both expose interfaces with sufficient commands for repository manipulations. A .NET Core package is available for Git, which enables quick manipulation of repositories. SVN repositories accept commands over HTTP which allows for even faster manipulation without the need to transfer all of the files first.

To increase the efficiency of data transfer and storage space usage, the actual files that are stored are reduced to a minimally needed set of data to be able to use the repository. Git is a distributed VCS, meaning that the full contents of the entire repository are stored in each distribution. The .git subfolder contains all files necessary to maintain and manipulate the repository [40], including all objects that have been committed to the repository at any point in time, stored in a special database file. Therefore, only publishing this sub-folder is enough to allow the full reconstruction of the repository. SVN is a centralized VCS, meaning that there is a single host server, called the "remote repository", which each contributor commits changes to and pulls changes from. To be able to keep a repository intact without any corruption, this entire remote repository has to be maintained. However, any local SVN repositories (called "working copies"), which contributors have to maintain on their own systems to be able to communicate with the remote repository, and which do not contain information about every snapshot, do not have to be stored at all.

To allow for the publishing of data subsets, the prototype includes a feature that, upon entering a valid repository URL, retrieves a list of all known snapshots for this repository. The user is then provided with a user interface detailing each snapshot and its file contents. They can then select the desired snapshot to be published. In case the snapshot is from a Git repository, the snapshot is reconstructed, restoring all files as they were. However, this reconstruction still contains a .git folder, as this is where the snapshot is reconstructed from. This folder contains all data from the entire repository, most of which is not needed to be published. Furthermore, the data which is needed to be published has already been reconstructed into proper file formats (instead of being stored inside a Git database file). Therefore, the .git folder can be dropped completely and is not published. If the snapshot is from an SVN repository, the specific snapshot is checked out from the remote repository, creating a local working copy. This working copy contains a .svn folder including some metadata about the snapshot. However, this data is not desired or required to view or use the file contents of the snapshot. Therefore, it is also omitted during publication.

4.2.2 Publishing Systems

Three publishing systems, EUDAT - B2SHARE, figshare, and Harvard Dataverse, are supported by the prototype. Each of these systems exposes an HTTP API and is freely accessible to anyone. Therefore, any user can obtain credentials for making use of the services. Furthermore, they each offer API support for creating publications and additional operations which allow for quick and easy integration into the system. Moreover, they each provide options for metadata (although not all with a similar granularity or richness), as well as automatic PID assignment (figshare and Dataverse both assign DOIs when publishing, while EUDAT uses its own identifier scheme).

4.2.3 Metadata Mapping

B2FIND is a service which harvests metadata from a multitude of Research Institutions and creates a mapping between the harvested metadata schema and a self-defined, homogeneous B2FIND schema [41]. B2FIND then allows users to search and find research data across scientific disciplines and research areas - thus enabling an interdisciplinary perspective. The mappings between the RI and B2FIND schemas is done manually, by opening a dialogue with the RI and coming to a mutually satisfiable conclusion. Further research indicates that in the current state-of-the-art there is no tool to perform automated metadata mappings, as each one requires human input. Therefore, a metadata mapping has to be manually created for the prototype.

To create such a mapping, the different metadata standards from the three publishing systems have been outlined in Table 4.1. For brevity, properties that are differently named but similar in interpretation have been combined. As seen in the table, not all three publishing systems offer support for each metadata property, and which metadata properties are required and optional also differs. Note that B2SHARE

contains two metadata fields which none of the other systems do: Community and Open Access. The Community field is intended for a specific B2SHARE feature which allows users to publish data to different categories in the B2SHARE database, called communities, each regarding a specific topic. As the data submitted to the tool will be of varying disciplines, the default community which is suitable for any type of data will be used for all publications. The Open Access field is required to denote whether the data can be openly accessed by anyone. Both other systems also contain such a feature, however, the accessibility is not set through metadata but a distinct API function. Therefore, the Open Access field will be available in the tool's metadata but handled separately for each system.

Currently, the most up-to-date, cross-discipline metadata standard is the one proposed by DataCite [42]. It has been in constant iterative development since 2011 and is created in cooperation with the entire research community. Table 4.2 shows the properties currently proposed by DataCite and their usage by B2SHARE, figshare, and Harvard Dataverse. Some properties which are marked as shared exist under a different name in the publishing system metadata properties, but their interpretation is similar. Based on this table, a final metadata standard has been created which covers all three publishing systems, seen in Table 4.3. This table shows, for each of the three publishing systems, which metadata property has been mapped to which DataCite property. Most properties that are not shared between all three services have been removed from the finalized standard. However, some properties (namely those concerning publication date, language, and licensing information) have been kept, even if not all systems support them. These properties are deemed to be of enough importance to remain included. When submitting data to a system not supporting these properties, they will be left out of the submission.

4.2.4 Duplicate Publications

To reduce the number of unnecessary publications, the tool keeps track of all published repositories and prevents duplicate publications. When the desired snapshot of a repository is obtained, a checksum is calculated for it. This checksum is then checked against a local database containing every publication that has been made, including their checksum and publication URL. When a duplicate checksum is found, the already existing publication URL is returned to the user to facilitate the reusability of the digital objects.

An obvious way to calculate such a checksum would be by, after cloning the repository snapshot to the local drive, calculating a checksum for all files in the repository directory. However, due to metadata in the files which are altered by the cloning process, the same repository will not obtain the same checksum after distinct cloning operations. For this reason, internal variables of the VCS have to be used (e.g. commit hashes in Git, and dump files in SVN). Unfortunately, this means the VCS is no longer completely treated as a black box.

Storing checksums can be done either as a part of the repository publication or in a database local to the tool. The first option would drastically decrease performance, as every single publication had to be retrieved from a remote server to run a checksum comparison. The second option has better performance, however, it requires the checksums to be stored at a location separate from the actual persistent publication. Separation of (meta)data in this way could lead to problems if either one of the data locations becomes compromised in some way. While the data publication system is generally (by definition) guaranteed to be always available, the same can not be guaranteed about this tool.

4.2.5 Composite Publication

Besides publishing single repositories, the tool allows users to publish multiple repositories simultaneously as a bundle, or a composite publication. This is useful for repositories which contain dependencies on other repositories, and are therefore always required to be obtained together.

The bundle itself is published as a separate digital object, containing references to the repositories' digital objects, which have been separately published. This separation enables composite publications to reuse existing repository publications (duplicates), simply by referencing the existing publication in the bundle instead of creating an entirely new publication.

4.2.6 Transactional Nature

To ensure the integrity of the publication repository, the tool's publication process must be of a *transactional* nature, meaning that either all steps of the process complete or none of them. If at any point during the publication process an error occurs, all previously completed steps must be undone. This is

Table 4.1: Metadata properties used by the B2SHARE, figshare and Harvard Dataverse publishing systems. Red properties indicate required metadata, black properties indicate optional metadata.

	B2SHARE	**figshare**	**Harvard Dataverse**
Community	X		
Title(s)	X	X	X
Subtitle			X
Alternative Title			X
Description	X	X	X
Author(s)	X	X	X
Contributors	X		X
Depositor			X
Open Access	X		
Embargo Date	X	X	
Publication Date	X	X	
Production Date			X
Distribution Date			X
Disciplines	X	X	X
Keywords	X	X	X
Tags		X	
References		X	X
Related Publications			X
License	X	X	
Contact	X		X
Resource Type	X	X	X
Alternate Identifiers	X		X
Version	X		
Publisher	X		
Distributor			X
Language	X		X
Custom Fields		X	
Funding		X	X

Table 4.2: Usage of DataCite metadata properties by the B2SHARE, figshare and Harvard Dataverse publishing systems. Red properties indicate required metadata, black properties indicate optional metadata.

	B2SHARE	figshare	Harvard Dataverse
Identifier			
Title	X	X	X
Author	X	X	X
Publisher	X		
Publication Year	X	X	
Resource Type	X	X	X
Subject	X	X	X
Contributor	X		X
Date	X	X	
Language	X		X
Alternate Identifier	X		X
Related Identifier			
Size			
Format			
Version	X		
Rights	X	X	X
Description	X	X	X
Geo Location			
Funding Reference		X	X

Table 4.3: Mapping of the finalized prototype metadata standard to the B2SHARE, figshare and Harvard Dataverse publishing systems. Red properties indicate required metadata, black properties indicate optional metadata.

	B2SHARE	figshare	Harvard Dataverse
Title	Title(s)	title	Title
Author	Creators	authors	Author
Publication Year	Publication Date	timeline	
Resource Type	Resource Type	defined_type	Kind of Data
Subject	Disciplines	categories	Subject
Contributor	Contributors	contributors	Contributor
Date	Publication Date	timeline	
Language	Language		Language
Related Identifier	Alternate identifiers	references	Related Publication
Rights	License	license	
Description	Description	description	Description
Keywords	Keywords	keywords	Keyword

easily doable for single publications, where the final publication step is not made until everything else is verified.

Difficulties occur when making a composite publication (publishing multiple repositories at the same time). If an error occurs when performing the final step of the publication process, all previously published repositories which are part of the current composite publication have to be unpublished. However, due to the persistent nature of the publication systems that are used, this is not possible. Therefore, it could be possible for the entire composite publication to fail while some of its repositories have still been published.

Chapter 5

Results

5.1 Requirements Results

Chapters 3 and 4 have described an architecture for persistent publishing of version control repositories to existing digital object publishing systems, and implementation of this architecture, meant to satisfy a list of requirements posed in Section 3.1. This section will discuss whether each of the posed requirements has been satisfied, and if so, to what extent.

1. **Data Guarantees** The heart of the proposed tool lies in the distribution of the version control repository data to different publishing systems, which all bring different guarantees about the data that is uploaded. Because the tool only acts as a bridge between the VCS and publishing system, it does not store any data itself, but only facilitates its movement. Because of this, any guarantees about the data after it has been moved are independent of the tool itself and therefore difficult to provide to the user.

   The only guarantees that can be established are those in the intersection of the guarantees provided by each distinct publishing system, meaning those guarantees which are provided in the same way by all of the publishing systems. However, listing these guarantees would be a difficult and time-consuming endeavor, as some are abstract in nature and dependent on varying conditions. Furthermore, providing these guarantees to the user would likely lead to confusion, as each publishing system has additional guarantees not covered by the listing of the tool. Moreover, not being able to provide a single, comprehensive list of all guarantees offered by the tool (because there are multiple possibilities) also impairs the functional clarity of the tool.
2. **Metadata Standard** As described in Section 4.2.3, a singular metadata standard has been implemented which covers all available publishing systems. Although a lot of overlapping metadata properties are included in the final mapping, many important and more detailed properties are excluded. As evident from this case, the different metadata standards for each publishing system introduce difficulty when trying to create tools that create bridges between such systems.
3. **Data Subsets** By allowing users to choose a specific snapshot to be published, the tool provides support for data subsets with a highly specified level of granularity, as the user themselves can decide exactly what data goes into the snapshot. However, due to lacking metadata support and versioning capabilities from the publishing systems, properly labeling these subsets becomes difficult. Because of these difficulties, machine identification, as well as human identification of these data subsets, is impaired.
4. **Data Dependencies** Due to the availability of composite publishing, labeling and storing dependencies is possible. However, these dependencies are often stored in a rudimentary way. None of the publishing systems used by the tool include a "dependency" metadata field, and therefore such dependencies have to either be included in the data itself (which is not always possible due to licensing issues) or be stored in another, somewhat related field (e.g. the "references" field in figshare).
5. **Transactional Nature** For single publications, the processes employed by the tool are transactional. However, when making composite publications, there exist certain conditions that breach the guarantee of a transactional process. While this is not necessarily a problem for users, it can lead to clutter and incomplete publications in the publishing system repositories (if these conditions would occur in very large numbers). This can, in turn, affect the effectiveness of the repositories

themselves, leading to a larger number of false positives when searching for data as well as longer search times.

6. **User Interface** The tool provides a single user interface that spans all supported VCSs and publishing systems. It is similar to many existing interfaces and forms found on the Internet, which should aid in user recognition and usability. Because there has been no user testing done specifically for the interface, it is impossible to make any conclusions regarding its effectiveness.
7. **Performance** Because the prototype is implemented using cloud technology, it can easily be scaled up to allow many concurrent users to utilize the service simultaneously. However, because the tool forms a bridge between two components, the data needs to bridge longer distances causing higher latency during the publication process. Therefore, the publication time (which could be used as a metric of performance for this tool) is longer than existing systems, for example, Zenodo, which has its own, internal repository readily available with very low latencies. However, several performance metrics can still be measured, which is shown in Section 5.2.

5.2 Performance Measuring

As stated in Section 3.1, performance for a tool such as this is not trivial to define and can include both technological and user interaction aspects. Because no proper user testing has been done for this prototype, only technological metrics have been measured. A series of load tests have been performed to measure how the application scales when multitudes of concurrent users are requesting to publish repositories. These tests have been completed using the Azure DevOps [43] platform, which offers a load testing feature that allows you to simulate a number of virtual users that send a specified web request to a service over a certain time span.

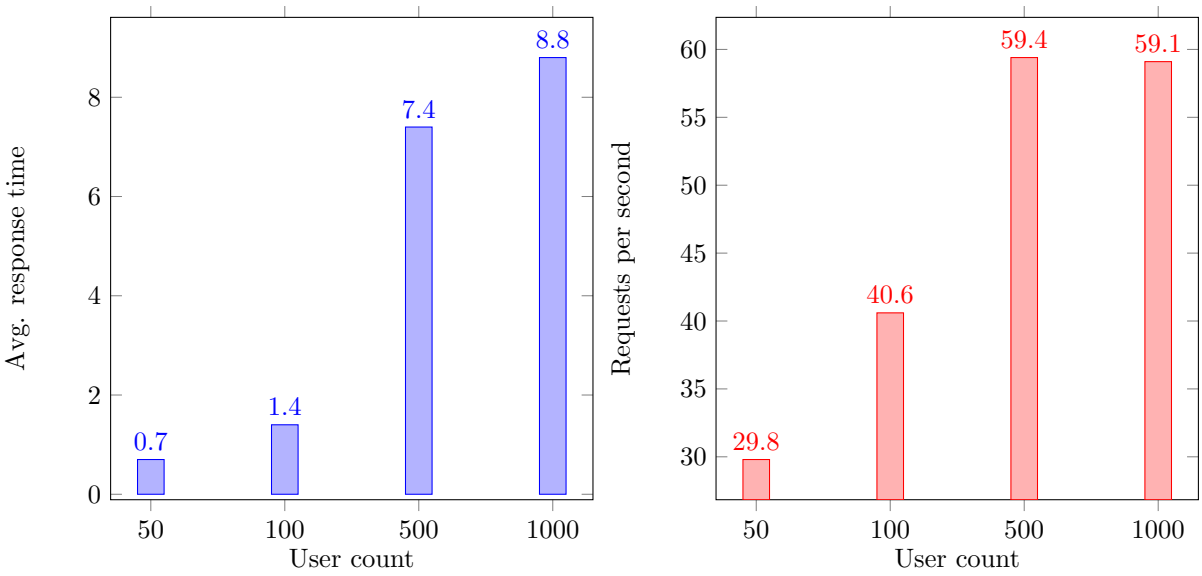
Many components of the prototype interface with other applications and services over the Internet. In order to perform load testing on the prototype, these interfaces had to be closed off, to avoid simultaneously load testing the external services. Therefore, the downloading of repositories, uploading to the publishing system and communication with the internal database have been disabled and replaced with dummy interfaces that always return a default result. However, all programming logic remains and is tested. Not only does this approach reduce the load on the external services, but it also increases the reliability of the load tests because there is no longer any external dependencies present which could influence the results.

To determine the number of users to be load tested, data from Zenodo has been obtained and analyzed to see how many users are publishing repositories through GitHub to Zenodo using its integrated feature. Unfortunately, this data could not be obtained using the Zenodo developer API, as it only allows users to access information about datasets that they have published themselves (a personal Access Key is required for each API call). Therefore, the data has been obtained using the Zenodo search feature along with web scraping techniques. By searching Zenodo for all publications with the "Software" Type metadata property, a list of the past 1850 publications has been obtained. This list was filtered to contain only publications that have been published through GitHub and then sorted by date (shown in Table 5.1). From these results, it is clear that this service is being used by only a small number of users. The load tests have therefore been performed with 50 concurrent users (roughly the maximum amount possible for any single day according to the data from Zenodo), 100, 500 and 1000 users (multiples of the lower bound up to a factor of 20). Table 5.2 shows the complete list of Azure DevOps Load Test settings and values that were used during these load tests.

To determine technological performance of the system, two metrics have been measured: average response time and requests per second. Average response time is measured by taking the time between when the request is sent by the (simulated) user browser and when the entirety of the response has been received. To obtain an average, the mathematical mean of the response time of each request is calculated. Requests per second is measured by the (average) amount of requests made to the service every second of the load test. The average response time metric is a good indicator of processing speed, whether the service can receive, process and respond to requests in a timely manner. Poor performance in these areas would mean that the average response time rises rapidly with many concurrent users, slowing down response times and degrading user experience and usability of the system overall. Requests per second is an indicator of the efficiency of the system in sending and receiving requests, i.e. whether or not unnecessary requests are being made.

Figures 5.1, 5.2 and 5.3 show the average response time as well as requests per second that are being processed simultaneously when publishing a single repository, a composite of 2 repositories and a

Figure 5.1: Average response time and requests per second for publishing a single repository



composite of 5 repositories, respectively, to the system. These results show that when publishing a single or a composite of 2 repositories, the response time scales linearly with the number of users between 100 and 500 (with about a factor 5 increase, which is expected, as the number of users also increases by that factor). However, between 500 and 1000 users, the scaling is much better, with only around a factor of about 1.25 (while a factor of 2 is expected). When publishing a composite of 5 repositories though, the average response time scales completely linearly, increasing with a factor very close to the one that increases the number of virtual users.

The statistics for the requests per second metric are much more variable and do not seem to follow a general pattern at all. One would assume that when the number of concurrent users goes up, the average response time increases and therefore the number of requests that can be handled simultaneously per second goes down. However, this only seems to be the case in the tests for publishing a composite of 5 repositories. In the other tests, the number of requests per second generally increases as the user load does. Although this is unexpected, a larger number of requests per second means more requests are being processed simultaneously, which indicates a higher degree of scalability and performance.

To assess the performance of the system using these statistics, the user activity data from Zenodo needs to be taken into account, as well as the general use case of the system. Zenodo only has a possible maximum concurrent user load of 50. The system, at 50 users, performs well (scaling with a factor that is less than linear to the number of repositories published). Furthermore, when examining the general use case of the system, users are unlikely to perform publications more often than the peak response time visible in the statistics. When writing source code through a VCS, commits can happen in large numbers and frequently. However, publications of source code generally only happen when a certain unit of work has been completed or milestone has reached. These things occur much less frequently and therefore there is less pressure on a system that provides a service for this task to perform it extremely quickly. Of course, should such a system be more properly implemented, for example in the ENVRI-hub environment (see Section 2.1.4), the number of concurrent users is likely to rise significantly. However, due to the nature of the use case, the current response times shown are still adequate should the user load increase by a factor of 20 of the currently expected users.

Additional performance data, which shows the increase and decrease of performance over the duration of the load tests, can be found in Appendix B. Although some spikes are seen both upwards and downwards, generally the plots are straight within an acceptable vertical range. This is the most desirable shape, as any spikes indicate a large difference in either average response time or requests handled per second, which implies a poor ability to handle a high user load over a long time period. Figures B.5 and B.6 however, do show consistently increasing numbers for both average response time and requests per second when publishing a composite of 5 repositories. This could indicate that the system does start to collapse under a consistently higher load, although, as shown by the averages that have been discussed in the previous paragraph, still within an acceptable margin.

Table 5.1: GitHub repositories published to Zenodo between September 5th, 2019 and October 17th, 2019

Date	Repositories published
2019-09-05	1
2019-09-06	14
2019-09-07	11
2019-09-08	22
2019-09-09	51
2019-09-10	48
2019-09-11	44
2019-09-12	45
2019-09-13	39
2019-09-14	16
2019-09-15	8
2019-09-16	39
2019-09-17	55
2019-09-18	52
2019-09-19	45
2019-09-20	48
2019-09-21	12
2019-09-22	25
2019-09-23	43
2019-09-24	43
2019-09-25	38
2019-09-26	48
2019-09-27	57
2019-09-28	16
2019-09-29	19
2019-09-30	40
2019-10-01	62
2019-10-02	42
2019-10-03	45
2019-10-04	50
2019-10-05	9
2019-10-06	27
2019-10-07	37
2019-10-08	61
2019-10-09	56
2019-10-10	46
2019-10-11	44
2019-10-12	18
2019-10-13	23
2019-10-14	63
2019-10-15	56
2019-10-16	50
2019-10-17	27

Table 5.2: Azure DevOps Load Test settings and values

Setting	Value
Duration	2 minutes
Load pattern	Constant
Browser mix	40% IE, 60% Chrome
Load agents Geo-location	West-Europe (Netherlands)
Warmup duration	0 seconds
Virtual users	50, 100, 500, 1000

Figure 5.2: Average response time and requests per second for publishing a composite of 2 repositories

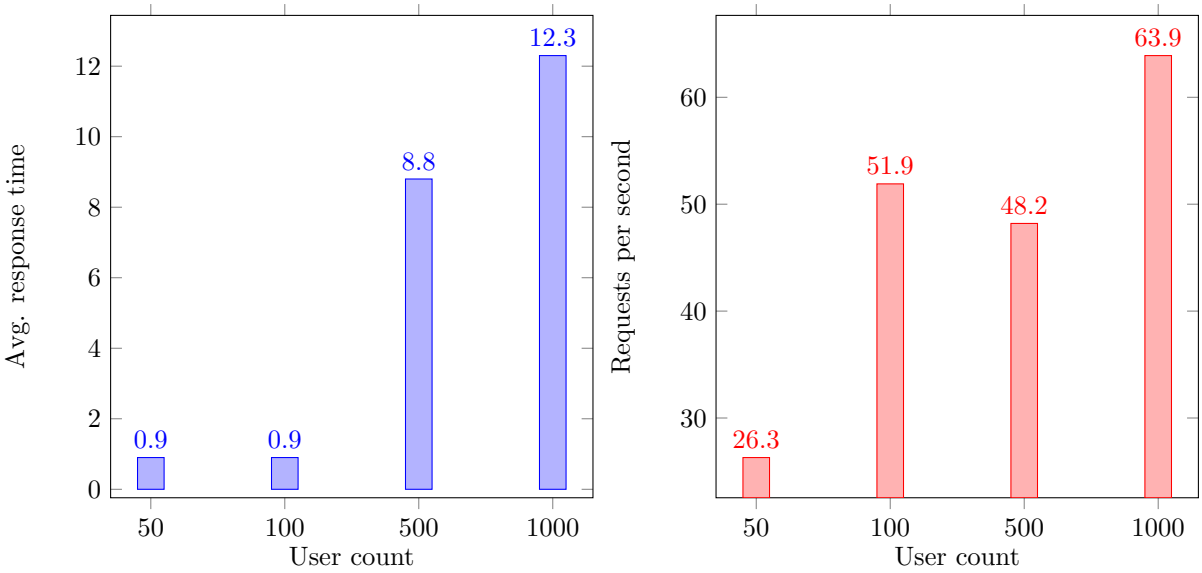
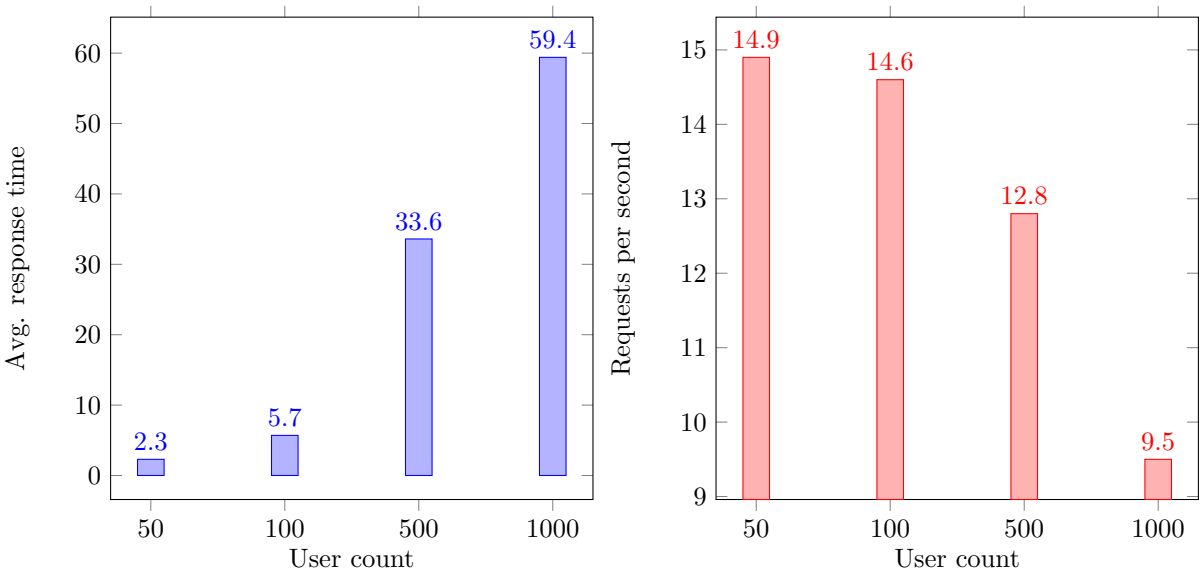


Figure 5.3: Average response time and requests per second for publishing a composite of 5 repositories



5.3 Other results

The prototype described in the previous chapter is the first step towards solving the problems posed in Section 1.2. It offers functionality for persistently publishing existing version control repositories, assigning PIDs to them and easing the transition to such persistent publishing systems for researchers or industry workers who are unfamiliar with these new technologies. Furthermore, it provides means to cite specific software versions and research data subsets. Moreover, it allows these processes from multiple different version control systems to multiple different publishing systems, with an abstract design leaving room to be extended to any number of additional systems in the future.

While several of the problem scenarios introduced in Chapter 1 could benefit from such a service, there are still many improvements that could be made to increase its effectiveness and interoperability with existing version control workflows. For example, the current prototype only allows manual manipulation for publishing data objects, and no machine manipulation. This is not as much of a problem when a single researcher is using the repository. However, when multiple researchers are using a repository and pushing data or code to it independently, being able to automatically create publications of snapshots based on certain criteria, and implement this into the software developer workflow, could help in the organization of a project.

Furthermore, the metadata standard provided by the tool is very limited due to the differences in metadata properties between the possible publishing systems. Because metadata and machine actionability are two of the most important aspects of the FAIR principles, these two factors pose a big limitation on the effectiveness of the tool.

Additionally, data provenance is not accounted for when publishing with the tool. Both figshare and Harvard Dataverse do not offer support for versioning of published digital objects, and B2SHARE only offers rudimentary support in the form of a metadata property. While it is difficult to provide such versioning with the tool itself, it is still paramount to the usefulness of online publishing services to provide these functionalities.

Moreover, from trying to implement this tool, it has become clear that current publishing systems are focused on data and not software. Even figshare, which has built-in GitHub integration, a platform that primarily hosts software, offers little metadata properties and other types of features specifically used for software rather than data. This makes it difficult to properly host software source code on these platforms.

Chapter 6

Discussion

In this chapter, the work, and results of said work, done in this thesis are discussed. First, the degree to which the research questions have been answered will be discussed. Next, several social challenges and quality control concerns are considered. Then, the novelty of the work and possible innovations will be examined, as well as additional contexts in which the work could be used.

6.1 Research Questions

Sub RQ1: What are the state-of-the-art approaches to enable data FAIRness of Version Control Systems?
We have seen many different persistent data management systems, and all of them share similar attributes. The general strategies for data management which they employ are similar, and following the FAIR principles (albeit to differing degrees). However, there are large differences in both metadata properties as well as other data management related features that impair the FAIRness of data and software which is published to these systems.

Two different systems which offer a service for persistent publishing of VCS repositories have been examined, namely Zenodo and figshare. Both these systems work via GitHub integration and do not offer any way for publishing repositories from either a different VCS or a different web host. Furthermore, both systems can only publish to their internal repository, meaning there is no flexibility in data host or PID type selection. Because of these two reasons, options for publishing VCS repositories are very limited.

Sub RQ2: How can digital objects in VCS repositories be persistently published with PIDs and metadata?
Section 3.1 details several requirements which have been obtained from literature study, previous experience within the field of software engineering and existing systems which have been examined as a part of this research. The listed requirements are only an initial suggestion of the complete list of requirements for persistent publishing of VCS repositories, and a more extensive requirements analysis is needed to develop such a list. Both interviewing of field experts as well as extensive user testing is needed to completely flesh out the final requirements.

RQ: How can existing Version Control Systems be connected to persistent identifier-based data management systems?
Chapter 3 describes an abstract architecture for connecting existing VCSs to persistent identifier-based data management systems. It is important to note that the architecture described and implemented in this thesis is only a proof of concept. It is meant to discover what kind of workflow is needed when persistently publishing VCS repositories, as well as any problems that arise when trying to create these workflows and how these problems should be managed. While the implementation of this architecture was successful, several problems arose which, at this stage, are not yet solved. The root cause of most problems lies in the incompatibility of the standards employed by the different publishing systems. While these systems are attempting to adhere to the FAIR principles, and are somewhat successful in this, the lack of actual standardized definitions, instead of mere principles or guidelines, still causes a hinder for both humans and machines interacting with these systems. Therefore, connecting VCSs and persistent publishing systems is practically possible, but far from theoretically optimized.

The conclusions that are made when answering this question have been made by examining purely

freely available online FAIR enabled publishing systems. A number of closed-off systems, only accessible to e.g. academic organizations or certain research institutes, might exist which, when examined, would lead to different answers to the research questions.

6.2 Social Challenges

Besides the technical challenges laid out in this thesis, there are several social challenges to consider as well. Many RIs and data-intensive organizations have a hesitancy to start utilizing persistent identifier systems in their data management cycles. Reasons for this hesitancy include a profound gap in knowledge about what persistent and unique identifiers are, what they can be used for, and what the best practices regarding their use are, perceived high investment costs (both for personnel, hardware and software), and a lack of support from the respective scientific communities to change engrained work practices. The use of persistent and unique identifiers for both data and metadata objects throughout the entire data life cycle needs to be encouraged, and RIs should become apt users of both tools for applying PIDs to their (digital) resources and data, as well as tools that use supported identification and citation mechanisms to facilitate research work - including data movement, method application to sets, provenance tracking, etc [3]. Therefore, solutions meant for transitioning organizations from legacy systems to persistent identifier systems need to be made available to (in part) provide such encouragement. A system such as described in this thesis could offer this encouragement, as it is available to use through an online service, without any extra organizational overhead. Furthermore, it is browser-based and does not require any additional expertise to integrate into existing workflows, therefore having a high degree of accessibility. Organizations that are hesitant on investing time and effort into enabling the FAIRness of their data management practices can utilize this system with their existing VCS repositories to experience the power of PIDs without any significant changes to data management structure or human resource investments.

Furthermore, due to the modular set up of the system and its components, as well as its nature as a Web service, there are opportunities for inclusion in projects like the ENVRI-hub (see Section 2.1.4). This could aid in the spread of the service to more potential users and popularizing the assignment of PIDs to VCS repositories, also easing the transition from traditional data management systems to PID based systems for hesitant data providers. However, to make this possible the system would have to be heavily modified to be able to provide a high standard of authentication, authorization, and integration with existing ENVRI services.

6.3 Quality Control

In the data management life cycle, quality control is an important aspect. Ideally, the quality of FAIRly published data is checked and approved before the publication is made final. Of course, methods for confirming the quality of data are highly dependent on the data and its context and can therefore (largely) not be implemented in a general sense. However, FAIRness is not about the quality of data, but rather the quality of metadata. The system proposed in this thesis demands a rigorous metadata schema, that is based on the latest, generally approved standards, to be filled out whenever a repository is published. Therefore, the tool provides at least the FAIRness aspect of quality control. Of course, quality control is only as good as the metadata schema. Should there be new developments from the data science community which bring forth a new consensus about the optimal metadata schema, the system should adjust accordingly.

Many platforms hosting version control repositories, such as GitHub or Azure DevOps, provide integrated quality control features (e.g. automated test and build pipelines when committing new content or issue boards allowing users to request changes). However, metadata is generally not something that is checked during these quality control processes. Therefore, should the proposed system be used in practice, e.g. for publishing repository contents after every software version is complete, it can be used in the quality control procedure to ascertain the correct metadata properties. However, the effectiveness and user experience would be much higher if the system was integrated with the existing pipelines and therefore could be managed from and automatically deployed by the platforms that are already being used for many other tasks.

6.4 Novelty and Innovation

While theoretical research has been done into exactly how digital objects should be formed to provide the greatest benefits to data consumers, practical research and solutions are still in their infancy. Such practical solutions for integrating Version Control Systems with persistent publishing systems do exist, but this thesis discusses a novel approach to this problem: providing a single solution for multiple VCSs and multiple publishing systems, while existing solutions only allow for a single VCS (which must be hosted on a specific platform) and a single publishing system.

While conducting the research, it has become clear that options for persistent software publishing are limited, both in service providers as well as provided functionalities and a lot of room is left to further explore these aspects of this problem space. Furthermore, it has been shown that, even though the FAIR principles are being adhered to by persistent publishing systems, they are still not very suitable for publishing software. This implies that the FAIR principles might not completely apply to software, or are not being implemented properly to deal with both software and data.

6.5 Additional Contexts

Some research is being done into creating practical applications dealing with DOs and PIDs. SeaDataNet [44], a distributed marine data infrastructure network for managing the large and diverse data sets collected by the oceanographic fleets and the automatic observation systems, maintains a project known as SeaDataCloud, which is a large database with millions of data records accessible by researchers. Because of the large data volumes that are being processed and transferred by this system, SeaDataNet is working with researchers to prevent network congestion when many concurrent users are active. One of the approaches to this problem is by utilizing a networking technique called NDN (see Section 2.1.5), which routes data objects based on (unique) names instead of a location. To integrate these two technologies, NDN must be able to interoperate with PIDs [20] [45]. When designing systems to integrate multiple technologies, all use cases from each technology must be accounted for. This thesis provides requirements and considerations for one such use case.

Expanding on this, techniques are being developed to optimize the fetching and sharing of PID identified objects over NDN [46]. Since PIDs are often used to fetch different data objects from multiple locations when reproducing a workflow published by community, an approach for integrating PIDs with NDN networks could be used to optimize the caching and distribution speeds of those data objects. Data repositories such as those discussed in this thesis can grow large in size, because all different versions of many, possibly large in size data objects are stored. Therefore, ensuring their efficient distribution is critical for user experience when utilizing services which store digital objects using PIDs.

Additionally, many RIs offer services not only for accessing and publishing data objects, but also for processing data based on user demands, e.g., via scientific workflows or third party virtual research environments. This data is almost universally stored using PIDs, providing a unique and persistent way to identify and cite stored DOs. However, efficiently retrieving and sharing digital objects in a shared data processing environment requires knowledge of application access patterns as well as the underlying network level distribution. Optimizing data discovery and access among distributed partners on shared infrastructure emerges as an important challenge for infrastructure operators to maintain quality of service and user experience [47]. Novel approaches utilizing ICN techniques to retrieve content based on PIDs are being developed to optimize such data access on shared infrastructure [47]. As mentioned before, optimizing efficient distribution of data repositories that can grow large in size is beneficial to users of version control systems.

Chapter 7

Conclusion

To enable open science and innovation in the industry, data sharing is critical. Current data management practices and solutions are not sustainable for the massive amounts of data that are being processed and transferred today, as well as for citation, interoperability and reproducibility needs. Therefore, new technologies and data management approaches need to be developed to deal with these problems. Several of these already exist, such as persistent identifiers, DOIP, and FAIR principles, but there are still many data management use cases that have not yet been covered.

This thesis has investigated one such use case, namely the integration of multiple Version Control Systems with multiple persistent data publishing systems. The state-of-the-art approaches to this case have been examined, and found that while solutions for this problem do exist, they are not diverse enough, and are lacking many useful features. For example, they only allow publishing repositories from a single kind of VCS, offer only a single type of PID and have no publishing database flexibility. Additionally, these solutions do not differentiate between general purpose data and software data. Therefore, many features that could be useful to software developers utilizing these systems are absent: metadata standards do not contain the necessary properties for software and APIs do not offer any specialized operations.

To expand this limited functionality, this thesis has proposed a novel solution that bridges this gap and creates a tighter integration between VCSs and publishing systems, by allowing publication of software repositories from multiple different VCSs to multiple different publishing systems. The requirements, implementation and workflow for this system detail how VCSs can be connected to persistent identifier-based data management systems and what must be kept in mind in future iterations of work on this topic. Although making this connection is practically possible, it is difficult and incomplete due to differing standards and practices between existing persistent data publishing services.

This thesis is only a preliminary study, and more research is needed for the persistent publishing of both version-controlled as well as non-version controlled software. Existing solutions for this problem do not yet include all features required by academic software developers, some of which have been laid out in this work. Software is a composite, live digital object and therefore has too many aspects that are subject to change to be directly submitted to persistent publishing systems that are targeted towards traditional data. More specific methods and tools need to be developed to facilitate the publishing of software and all of its specifics, such as dependencies and versioning.

Furthermore, the fact that all publishing systems which have been examined in this work follow the FAIR principles, yet are not suitable enough to be used to publish software suggests that the FAIR principles do not fully apply to software, or are only a part of the complete specification which is needed for software to be published. More work is needed which examines the FAIR principles in the context of software.

To improve the data management ecosystems that are used by scientists and industry, both theoretical standards, as well as practical applications in this field, need to be advanced. Not only to improve worker productivity and user experience but also to bring these issues to the attention of the public and increase the adoption rates among the scientific community.

Chapter 8

Future Work

Further VCS integration The prototype described in this thesis has focused on publishing VCS repositories to persistent publishing systems by manually selecting the required snapshots and manually entering the desired metadata. However, in many research and industry projects, large teams work together on single repositories. Therefore, manually publishing could cause problems because of a lack of communication between developers, resulting in a confusion of the publishing process. Integrated solutions into the VCS workflow and platforms offering VCS services which can automatically publish snapshots based on certain criteria could offer solutions here and should be considered for future research. Of course, there are many more complex scenarios that could be conceived regarding VCS workflow scenarios.

Software metadata and FAIR principles As discussed before, current metadata standards and by extension, the FAIR principles are possibly not completely suited for software. More research is needed to confirm this and create updated standards that can be used to persistently publish software.

Improved publishing systems As mentioned in the thesis, current publishing systems do not provide all required and/or desired functionality (e.g. composite publishing). Therefore, more research into possible requirements and implementing these requirements into existing and new publishing systems is needed.

Bibliography

[1] C. Fahrenfort and Z. Zhao, "Effective digital object access and sharing over a networked environment using DOIP and NDN", Sep. 2019. DOI: 10.1109/eScience.2019.00092. [Online]. Available: https://doi.org/10.1109/eScience.2019.00092.

[2] *Grasple webpage*, https://grasple.com/, Accessed: 2019-04-18.

[3] M. Hellström, M. Lassi, A. Vermeulen, R. Huber, M. Stocker, F. Toussaint, M. Atkinson, M. Fiebig, and Z. Zhao, "A system design for data identifier and citation services for environmental ris projects to prepare an envriplus strategy to negotiate with external organisations", ENVRIPLUS, Tech. Rep. D6.1, Jan. 2017. [Online]. Available: http://www.envriplus.eu/wp-content/uploads/2015/08/D6.1-A-system-design-for-data-identifier-and-citation-services-for-environmental-RIs.pdf.

[4] P. Wittenburg, "From persistent identifiers to digital objects to make data science more efficient", *Data Intelligence*, vol. 1, no. 1, pp. 6–21, 2019.

[5] *FAIR principles webpage*, https://www.go-fair.org/fair-principles/, Accessed: 2019-04-25.

[6] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, *et al.*, "The fair guiding principles for scientific data management and stewardship", *Scientific data*, vol. 3, 2016.

[7] FORCE11, "Guiding principles for findable, accessible, interoperable and re-usable data publishing version b1.0", Tech. Rep.

[8] *JSON for Linked Data webpage*, https://json-ld.org/, Accessed: 2019-10-08.

[9] *GitHub webpage*, https://github.com/, Accessed: 2019-08-05.

[10] F. McCown, S. Chan, M. L. Nelson, and J. Bollen, "The availability and persistence of web references in d-lib magazine", *arXiv preprint cs/0511077*, 2005.

[11] A. Karakannas and Z. Zhao, "Information centric networking for delivering big data with persistent identifiers", *University of Amsterdam*, 2014.

[12] J. Hakala *et al.*, "Persistent identifiers: An overview", *KIM Technology Watch Report*, 2010.

[13] R. Kahn and R. Wilensky, "A framework for distributed digital object services", *International Journal on Digital Libraries*, vol. 6, no. 2, pp. 115–123, 2006.

[14] P. Wittenburg and G. Strawn, "Common patterns in revolutionary infrastructures and data", *Draft manuscript*, 2018.

[15] *DONA Foundation webpage*, https://www.dona.net/, Accessed: 2019-10-09.

[16] DONA foundation, "Digital object interface protocol specification", Tech. Rep., 2018.

[17] B. MARIS, "Envri-fair-interoperable environmental fair data and services for society, innovation and research",

[18] *ESFRI webpage*, https://www.esfri.eu/forum, Accessed: 2019-10-21.

[19] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, *et al.*, "Named data networking (ndn) project", *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, vol. 157, p. 158, 2010.

[20] S. Koulouzis, R. Mousa, A. Karakannas, C. de Laat, and Z. Zhao, "Information centric networking for sharing and accessing digital objects with persistent identifiers on data infrastructures", in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, 2018, pp. 661–668.

[21] A. Karakannas, "Information Centric Networking for Delivering Big Data with Persistent Identifiers", PhD thesis, University of Amsterdam, Jul. 2014. DOI: 10.5281/zenodo.889603. [Online]. Available: https://doi.org/10.5281/zenodo.889603.

[22] *Git webpage*, https://git-scm.com/, Accessed: 2019-04-18.

[23] *Mercurial webpage*, https://www.mercurial-scm.org/, Accessed: 2019-08-27.

[24] *Subversion webpage*, https://subversion.apache.org/, Accessed: 2019-08-05.

[25] *Concurrent Versions System webpage*, https://www.nongnu.org/cvs/, Accessed: 2019-08-05.

[26] *EUDAT webpage*, https://eudat.eu, Accessed: 2019-08-23.

[27] *Dataverse webpage*, https://dataverse.org/, Accessed: 2019-08-26.

[28] *Harvard Dataverse webpage*, https://dataverse.harvard.edu/, Accessed: 2019-08-26.

[29] *Zenodo webpage*, https://zenodo.org/, Accessed: 2019-08-19.

[30] *figshare webpage*, https://figshare.com/, Accessed: 2019-08-21.

[31] P. Wittenburg, M. Hellström, C.-M. Zwölf, H. Abroshan, A. Asmi, G. D. Bernardo, D. Couvreur, T. Gaizer, P. Holub, R. Hooft, I. Häggström, M. Kohler, D. Koureas, W. Kuchinke, L. Milanesi, J. Padfield, A. Rosato, C. Staiger, D. van Uytvanck, and T. Weigel, "Persistent identifiers: Consolidated assertions", Tech. Rep., 2017. DOI: 10.15497/RDA00027. [Online]. Available: https://doi.org/10.15497/RDA00027.

[32] T. Weigel, T. DiLauro, and T. Zastrow, "Pid information types wg final deliverable", Tech. Rep., Oct. 2015. DOI: 10.15497/FDAA09D5-5ED0-403D-B97A-2675E1EBE786. [Online]. Available: https://doi.org/10.15497/FDAA09D5-5ED0-403D-B97A-2675E1EBE786.

[33] A. Rauber, A. Asmi, D. van Uytvanck, and S. Proell, "Data Citation of Evolving Data: Recommendations of the Working Group on Data Citation (WGDC)", Tech. Rep., Oct. 2015. DOI: 10.15497/RDA00016. [Online]. Available: https://doi.org/10.15497/RDA00016.

[34] *DVC webpage*, https://dvc.org/, Accessed: 2019-08-28.

[35] *Pachyderm webpage*, https://www.pachyderm.io/index.html, Accessed: 2019-08-28.

[36] *ASP .NET Core documentation webpage*, https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2, Accessed: 2019-09-02.

[37] *Angular webpage*, https://angular.io/, Accessed: 2019-09-03.

[38] *Azure webpage*, https://azure.microsoft.com/en-us/, Accessed: 2019-09-03.

[39] *Stack Overflow Developer Survey 2018*, https://insights.stackoverflow.com/survey/2018, Accessed: 2019-09-03.

[40] S. Chacon and B. Straub, *Pro git*. Apress, 2014.

[41] E. Quimbert, K. G. Jeffery, C. Martens, D. Boulanger, T. Carval, M. Hellström, H. Lankreijer, J. Peterseil, C. Pichot, and Z. Zhao, "Interoperable cataloguing and metadata harmonisation for environmental ris: Prototype", ENVRIPLUS, Tech. Rep. D8.4, Nov. 2018. [Online]. Available: http://www.envriplus.eu/wp-content/uploads/2015/08/D8.4-Interoperable-cataloguing-and-metadata-harmonisation-for-environmental-RIs-prototype-.pdf.

[42] DataCite Metadata Working Group, "Datacite metadata schema documentation for the publication and citation of research data version 4.3", DataCite e.V., Tech. Rep., 2019.

[43] *Azure DevOps webpage*, https://azure.microsoft.com/en-us/services/devops/, Accessed: 2019-10-18.

[44] *Seadatanet webpage*, https://www.seadatanet.org/, Accessed: 2019-09-24.

[45] K. de Jong and A. Younis, "Planning and scaling a named data network with persistent identifier interoperability", Master's thesis, University of Amsterdam, 2019.

[46] R. Mousa, "Application aware digital objects access and distribution using Named Data Networking (NDN)", PhD thesis, University of Amsterdam, Jul. 2017. DOI: 10.5281/zenodo.889740. [Online]. Available: https://doi.org/10.5281/zenodo.889740.

[47] S. Koulouzis, R. Mousa, A. Karakannas, C. de Laat, and Z. Zhao, "Information centric networking for sharing and accessing digital objects with persistent identifiers on data infrastructures", in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2018, pp. 661–668. DOI: 10.1109/CCGRID.2018.00098.

Appendix A

List of Acronyms

FAIR Findability, Accessibility, Interoperability, Re-usability
RI Research Institute
URI Uniform Resource Identifier
URL Uniform Resource Locator
PID Persistent Identifier
RDA Research Data Alliance
VCS Version Control System
HS The Handle System
DOI Digital Object Identifier
URN Uniform Resource Name
PURL Persistent URL
ARK Archival Resource Key
ePIC European Persistent Identifier Consortium
DO Digital Object
DOIP Digital Object Interface Protocol
IRP Identifier/Resolution Protocol
DOA Digital Object Architecture
EU H2020 European Union's Horizon 2020 program
ENVRI European Environmental Research Infrastructures
EOSC European Open Science Cloud
ESFRI European Strategy Forum on Research Infrastructures
SVN Subversion
CVS Concurrent Versions System
API Application Programming Interface
JSON JavaScript Object Notation
NDN Named Data Networking
ICN Information Centric Networking

Appendix B

Additional performance data

This appendix shows additional, more complete performance data to that which has been discussed in Chapter 5. The figures in this appendix show how the average response times and requests per second changed over the 2 minute duration of each load test.

Figure B.1: Average response time for publishing a single repository over 2 minutes

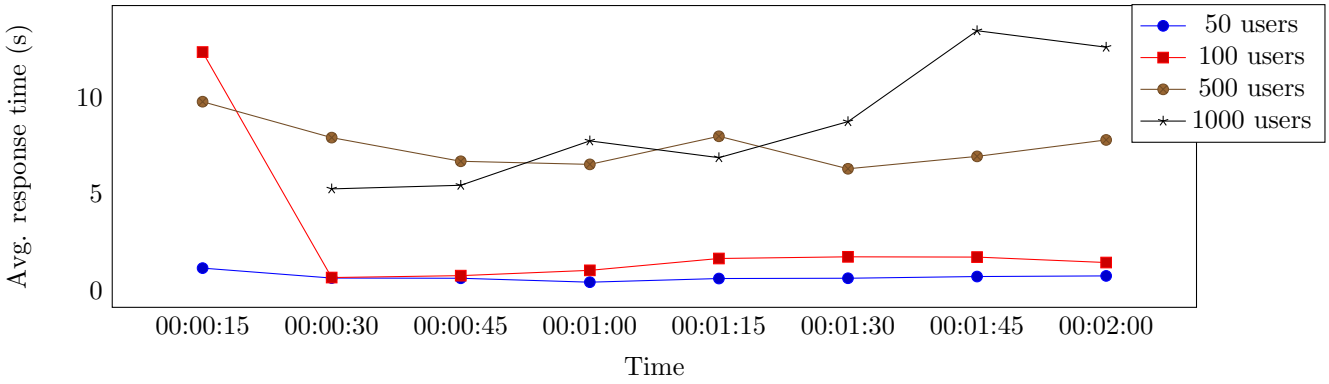


Figure B.2: Requests per second for publishing a single repository over 2 minutes

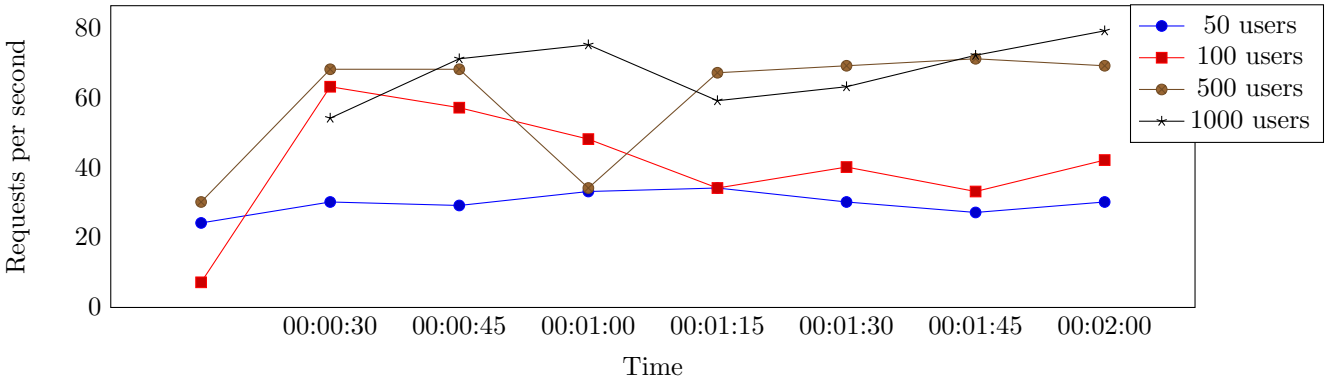


Figure B.3: Average response time for publishing a composite of 2 repositories over 2 minutes

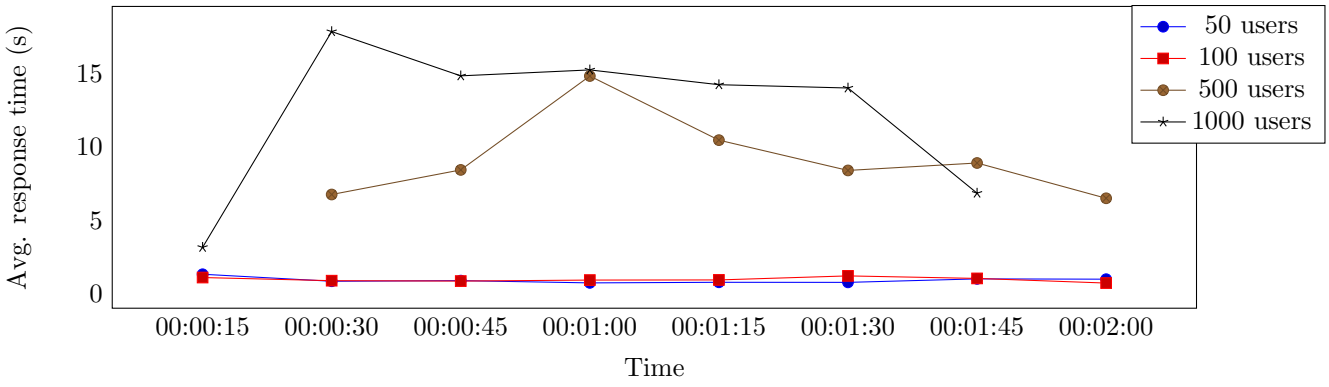


Figure B.4: Requests per second for publishing a composite of 2 repositories over 2 minutes

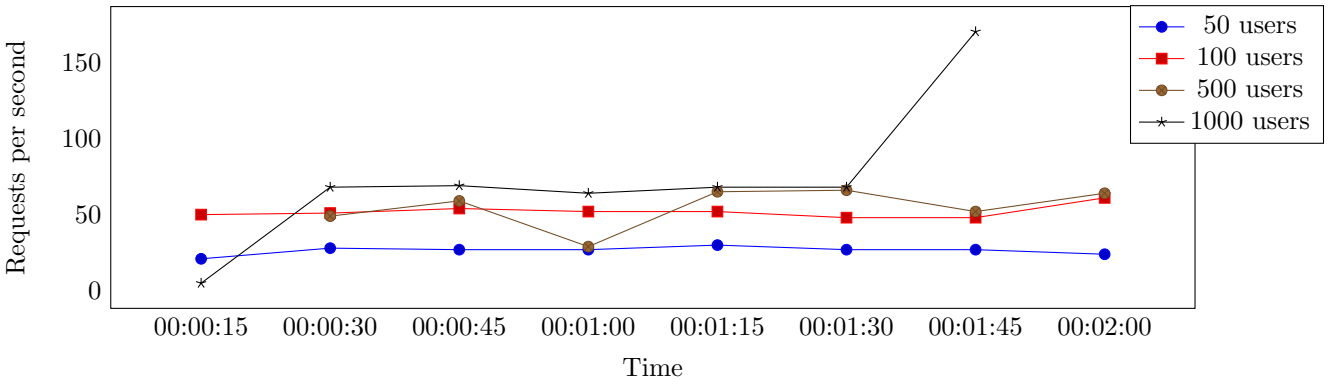


Figure B.5: Average response time for publishing a composite of 5 repositories over 2 minutes

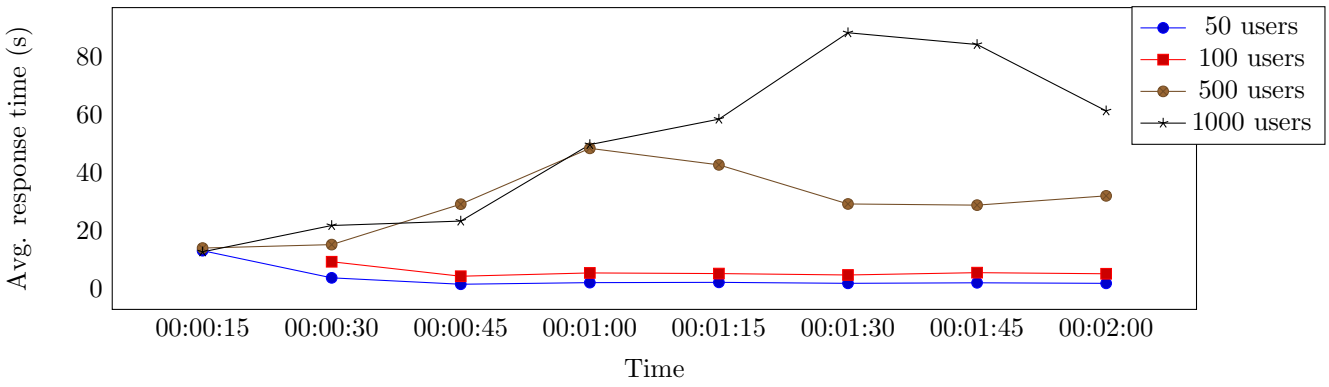


Figure B.6: Requests per second for publishing a composite of 5 repositories over 2 minutes