January 2015

DEVELOPING A METADATA REPOSITORY FOR DISTRIBUTED FILE ANNOTATION AND SHARING

Samuel Wilson
Purdue University

Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Samuel P Wilson

Entitled
DEVELOPING A METADATA REPOSITORY FOR DISTRIBUTED FILE ANNOTATION AND SHARING

For the degree of Master of Science

Is approved by the final examining committee:

J Eric Dietz
Chair

John Springer

Raymond Hansen

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): J Eric Dietz

Approved by: Jeffrey L Whitten 11/12/2015
Head of the Departmental Graduate Program Date

DEVELOPING A METADATA REPOSITORY FOR DISTRIBUTED FILE ANNOTATION AND SHARING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Samuel P. Wilson

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2015

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

I wish to first and foremost give praise to God for His saving and redeeming work in my life, without which master's degrees and life itself would be void.

I also wish to thank my wife for her enduring love and support. Without her joy and encouragement, I would not have considered a master's, let alone gotten this far.

I could not have done this particular study without the brilliance and insight of Nathan Deny. Thanks for showing me the need for a good hat rack.

Finally, but in no way less significantly, I wish to gratefully acknowledge my thesis committee for their insightful comments, guidance, and support. And a special thanks to Dr. Dietz, for taking me on as a student and giving me the flexibility to pursue my passions.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
DOI	Digital Object Identifier
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
NSF	National Science Foundation
OS	Operating System
POSIX	Portable Operating System Interface
RDF	Resource Description Framework
REST	Representational State Transfer
UML	Unified Modeling Language
W3C	World Wide Web Consortium

GLOSSARY

provenance	"The term provenance traditionally refers to the documented history of a work of art, which can be used as a guide toward the work's authenticity" (Godfrey, German, Davies, & Hindle, 2011, p. 65). In a similar manner, software provenance serves as the chain of custody of a given file.
reproducibility	Building upon provenance, reproducibility is the capability of researchers to arrive at scientifically similar results by means of repeating the processes and methods of the original authors (Gil et al., 2007).
REST API	Representational State Transfer APIs are a data-centric means of exposing semantic interfaces for interacting with external application data, utilizing web-based structures and protocols (Riva & Laitkorpi, 2009).

ABSTRACT

Wilson, Samuel P. M.S., Purdue University, December 2015. Developing a metadata repository for distributed file annotation and sharing. Major Professor: J. Eric Dietz.

Research data is being generated and modified at an increasingly accelerated rate. Iterations and derivations are being crafted at an almost equal velocity. With this increase comes a growing need to track the metadata about the data being generated. Where did this dataset originate? What exactly do the column headers mean? Who was the original publisher? Do I have the latest version of the data? This is to only name a few. As data is shared second or third-hand, or via alternative methods such as physical media or cloud based storage mechanisms, the veracity of the implicit metadata becomes circumstantial. This research quantified and contrasted existing file metadata management solutions, showing their inadequacy to solve the above stated problem, and highlighted the need for a new solution. The system subsequently established and developed by this research was designed to allow for arbitrary file metadata definitions across file systems in a collaborative manner, while facilitating platform independence and easy adoption.

CHAPTER 1. INTRODUCTION

Metadata, most simply put, is data about data. Without it, the file structures and operating systems known today would not exist. And yet, there is something oddly opaque and inaccessible about the metadata solutions currently available, specifically concerning user-defined metadata. In this chapter, the problems related to metadata will be identified, and the research that was conducted will be outlined. It will also quantify the scope of the research, and cast it within a concise domain framed by the research question. The chapter will then conclude by outlining the assumptions, limitations, and delimitations that were needed to effectively and feasibly conduct the study.

1.1 Background

Research data is being generated and modified at an increasingly accelerated rate. Iterations and derivations are being crafted at an almost equal velocity. With this increase comes a growing need to track the metadata about the data being generated. This data is typically housed in emails, Excel documents, and even handwritten notes. These storage and dissemination mechanisms, though reasonably effective at maintaining the integrity of the data itself, lack the necessary metadata and provenance over the long term. As time progresses and the origin of the dataset is lost from active thought, the user of the dataset may need to recall any one of several things. Where did this dataset originate? What exactly do the column headers mean? Who was the original publisher? Do I have the latest version of the data? This is to only name a few. If an original email can be found, one might be able to reconstruct several of these facts. But as data is shared second or

third-hand, or via alternative methods such as physical media or cloud based storage mechanisms, the veracity of the implicit metadata becomes circumstantial.

Many metadata systems exist on the market today. These systems fall into one of two primary categories: local file system metadata, and remote, web-based systems. But, as will be elaborated upon further in the following sections, these existing solutions are insufficient to meet the current needs. A better and more user-oriented metadata solution is required.

1.2 Scope

The primary purpose of this study was to determine if a system currently exists, or if one could be designed and prototyped, capable of spanning the gap between the traditional, local file system metadata, and purely remote, often web-based metadata systems. The spread of solutions in this space is wide, with differing problem statements spawning uniquely different mechanisms and feature sets. While local metadata systems offer convenience and operating system integration, they generally lack any awareness of the multiplicity of user environments, let alone distribution amongst users and organizations. Conversely, web-based systems typically offer communal solutions but make no contribution to assets that are not universally web accessible. These gaps will be further elaborated upon during the course of the review of relevant literature.

To understand whether or not a system existed to meet these needs, key features and factors of the metadata system needed were identified. Subsequently, an analysis of a wide range of existing solutions was conducted to determine what amount of coverage those solutions offered over the given feature set. Realizing that a sufficiently large gap existed between what was available and what was needed, a basic metadata system was designed and prototyped. The system includes the capability to store arbitrary metadata in the JavaScript Object Notation (JSON) standardized data format. The metadata server is open to queries from clients in a

standardized manner. Finally, the system focuses on the end user and the manner in which it will be used to ensure that it can be easily adapted and applied.

Given the core design elements as briefly described above, a system that can serve both local and distributed needs should have a significantly broader impact than merely metadata storage. Thus, beyond the basic system, other prototypes and conceptual models were considered. These provided proofs of concept or methodologies for expanded features, but were not all completely implemented in a production-ready manner. These features included a framework for versioning and provenance, as well as a security model. A hierarchical distribution system was also considered, allowing for the possibility of using the system locally, within a single organization, or in collaboration with other organizations.

Lastly, a sample command-line client was implemented. This demonstrated and facilitated the primary user interaction point with the system and should serve as a reference point for future clients.

1.3 Significance

The research and system that were produced out of this process will have its primary impact upon the research community, both academic and corporate. Given the current lack of unified, flexible, and transportable mechanisms for user file metadata definition, the introduction of a validated system such as this will allow for significant decreases in barriers to entry concerning the annotation of shared data and files. Additionally, with funding agencies such as The National Science Foundation (NSF) continuing to increase requirements for data transparency and availability, a system such as this will allow for increased compliance, openness, and accuracy of data after it has been shared.

That being said, this project could have ongoing and broader impact still, as any user wishing to describe their files in a platform independent manner could use it locally or in a distributed fashion, spanning both personal and collaborative

environments. This could facilitate a new wave of descriptive metadata sharing by users within their network of devices and beyond.

1.4 Research Question

Distilling the problems outlined above, this research was propelled by the following primary and secondary questions.

1.4.1 Primary Area

Can a new metadata system be proven necessary, designed, and prototyped that facilitates the sharing of arbitrary file metadata across file systems that is collaborative in nature, platform independent, and easily adopted?

1.4.2 Secondary Areas

- Can the system begin to facilitate such advanced operations as provenance tracking and versioning?
- Can the system be reasonably unrestrictive while still being cognizant of user context and security implications?
- Can the system be used locally, internally within an organization, or in collaboration with others in a potentially hierarchical manner?

1.5 Assumptions

The research herein was performed under the following assumptions:

- Files used during initial testing will not be larger than 1GB in size.

- Principles of software security will be considered, but the data stored within the application will be governed by open access and annotation policies unless the repository itself is secured.
- If applicable in a client-server environment, someone with some system administration experience will install the software.
- The metadata definitions are only as thorough and robust as the user choses to make them.
- A user wishing to use the application in a purely local configuration will require that the user have a broader system administration and installation background, given that they must essentially install the server as well as the client.

1.6 Limitations

The research herein was constrained by the following limitations:

- While the software developed can and should have a broad impact as noted above, the primary application of the initial research will be targeted at scientific inquiry producing data files shared within an undefined but finite group of researchers.
- The REST API will be the primary point of interaction with the server, and any significant abstractions of the basic GET, POST, PUT, and DELETE HTTP verbs will be incorporated into the clients.
- A command line client will be developed for testing, but will not fully realize the depth and abstraction of the user experience that is ultimately desired.

1.7 Delimitations

The research herein was conducted understanding the following delimitations:

- The software will not overcome any inherent security concerns found in the underlying applications (ex: MongoDB, PHP).
- The client prototype will not be production ready, nor will it be as user-friendly as ultimately desired.
- The developed repository is not intended to support, enforce, or enhance any specific metadata standard.
- The repository is not an archival solution and makes no claims to longevity.
- Clients may eventually become "smarter", but will not initially provide any additional metadata beyond that explicitly entered by the submitter.

1.8 Summary

This chapter established the background for this study by identifying the problem, scope, and significance of this research. Assumptions, limitations, and delimitations were delineated to ensure a focused and successful study. The next chapter will outline a review of the relevant literature in the field of file system metadata, looking at both local file system solutions as well as remote metadata repositories.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

This chapter seeks to elaborate on the existing literature surrounding file metadata systems in terms of their primary focus and functionality. Generically speaking, previous researchers have attempted to tackle the problem of file metadata from one of two broad angles, local file system metadata and remote metadata repositories. Both will be examined in turn. Though no system currently seeks to solve the exact combination of issues posed by this research, the following systems contain elements that position them in this general technological category.

2.1 Local File Metadata

The shortcomings of file metadata have been known for some time. Broadly speaking, file systems track relatively little meta-information concerning the files that they store. Primarily, these metadata items include system-centric information such as created and modified dates, ownership and permissions, and size and location of the files on the hard drive. Beyond that, specific file types occasionally offer built in standards for augmented metadata within the file itself. For example, MP3 audio files offer music and artist information stored in ID3 tags, and most photos have information concerning when the photo was taken and the specifications of the camera within its Exchangeable Image File Format (EXIF) data. But this is far from a general solution.

Cammarata, Kameny, Lender, and Replogle (1995) describe this issue when they note the following concerning their organization's internal data management structures:

> However, most of these databases have little documentation or other descriptional information to go along with them. The absence of such

> information leaves users at a loss for understanding the definitions, abbreviations, acronyms, and descriptions of the pieces of data stored and maintained in a DBMS (p. vii).

They go on to describe five significant problems being addressed by their proposed solution. They are, in essence, providing adequate documentation, establishing versioning, chronicling a history of structural changes, determining provenance, and governing standardization (Cammarata et al., 1995). These same principles and needs remain true today, and continue to be targeted by proposed file metadata solutions.

As S. Ames, Gokhale, and Maltzahn (2009) note in their introduction on the subject, there have been relatively no changes in the metadata management area of file systems in the last 10 years, even though significant improvements have been made in other areas of large-scale files systems. This leaves a sizable gap in users' ability to universally annotate any type of file that they have on their computer. Anecdotally speaking, people have tried many things, including keeping separate documents with file details, creating inline annotations within files, and file naming conventions (S. Ames et al., 2009).

2.1.1 Solutions

To combat this problem, using the paradigm of the local file system, several solutions have been constructed to offer augmented, metadata-rich, file systems. These applications position themselves directly on top of the file system for the purpose of offering greater metadata functionality. Several of these solutions will be examined to understand their purpose, features, and shortcomings, followed by an overview of the general advantages and disadvantages of the local solutions explored.

It is important to note that, though it is unlikely that a purely local metadata system will serve the purposes of this study (as one of the core requirements of this research is that the metadata would be communal), it is

important to begin here, as this lays the foundation for many of the established ideologies, and offers promising ideas and principles to govern the user's local interaction with the metadata.

2.1.1.1. Linking File System

One such local metadata system is that of the Linking File System (LiFS). Proposed and prototyped by A. Ames et al. (2005), it offers the specification of arbitrary user-defined keys and the ability to define relationships between files via links. Links offer the ability to associate two files, and attributes can be attached to both the links and the files themselves. The introduction of a file system level linking architecture and attributes offers extensive promise in the areas of "provenance, intended use, type, contents, creator, modification history, version, and other information that a user, application, or system may want to keep" (A. Ames et al., 2005, p. 50). By use of these extended attributes, context can be associated with files, something previously not possible, and critically important in many of the examples mentioned above. By linking files, one can begin to identify inputs or derivations. And by adding attributes, one could annotate where a file came from, or what its intended use is.

This serves as an example of the principle model for file system metadata through the use of low-level file system metadata architectures, or metadata integration into new file systems themselves. Unfortunately, though the Portable Operating System Interface (POSIX) style introduced by the LiFS is similar to that of existing Unix systems, the introduction of new and complex methods for interacting with the system seems somewhat complicated and inaccessible. Though a graphical user interface through the application would likely obscure some of these details from the average user, the likelihood of widespread integration of such a solution by a major developer such as Apple or Microsoft seems unlikely. This system also assumes the ability to retain the metadata in main memory for speed of

access and it does not offer the ability to be directly queried by the user or other applications.

2.1.1.2. Quasar File System

In slight contrast to the previous approach, S. Ames et al. (2009) demonstrated a unified file system approach where file system data and file metadata are both stored in a coordinated architecture called the Quasar File System (QFS). According to their research, the dichotomy between file systems for data storage and databases for querying extensibility was somewhat dated. S. Ames et al. (2009) further identified many of the shortcomings of the traditional model, namely in the use of relational databases at all, and a weak association between metadata and files identified by absolute paths. In their minds, the use of a relational database was not appropriate for file system metadata in the first place. Relational databases by definition require a structure for metadata fields. This leads to an overly generic system, or one that must be constantly updated for new fields required by additional file types or fields of study (S. Ames et al., 2009).

This analysis is apt, though other solutions now exist that might handle the metadata portion of the environment better, such as NoSQL databases. They also note that the link between file paths and metadata entries, as maintained by an absolute path, is a weak and expensive link to maintain (S. Ames et al., 2009). As files are moved and updated, it is consuming and expensive to track those changes, and errors and inconsistencies are more likely to be introduced.

The QFS works roughly the same way as LiFS and other similar implementations, but focuses more on the development of the Quasar query language, offering structured access to the underlying graph database. This emphasis on the query language poses interesting opportunities for searchable metadata. Though not inherently inaccessible in other implementations, the QFS places the database architecture at the forefront of their implementation.

2.1.1.3. File System as Linked Data

Though the local solutions mentioned above offer the rudimentary elements and functionality of interest in this research, they by definition lack many of the non-local benefits also being pursued. Other local solutions begin to bridge that gap by taking the local file system and connecting it to remote resources. He, Li, and Shen (2013) elaborate on this principle in their work on exposing file systems to the web as Linked Data through their application, F2R. This highlights several interesting principles in the transition from purely local metadata stores and feature sets, to shared and even distributed repositories and features.

He et al. (2013) propose the use of four key metadata sources to describe and even automate the metadata generation process. They are physical metadata, built-in metadata, user-defined metadata, and external data. In so doing, they offer flexibility, but also serve to aid the user in a potentially more robust metadata definition. Once described, structured definitions in the form of the Resource Description Framework (RDF) are available to be published to the semantic web for consumption.

There are, unfortunately, some shortcomings to the implementation. The interface itself is lacking, and the automatic metadata generation is dependent on a meaningful filename in a well-defined knowledge space. Additionally, automatic metadata generation has some value in back-porting or publishing existing data, but seems less relevant to the real time annotation of files being created and described.

2.1.2 Advantages

More generally speaking, each of the above-mentioned solutions share several key advantages. Namely, they are local. And, whether or not they had a fully developed graphical interface when conducting this research, they offer the potential for direct or low level integration into the users' file system environment and workflow. This, for example, could mean that a right-click menu option might allow

for definition or exploration of a file's metadata, rather than having to go to a browser to look up remotely stored information. This is a compelling factor in potential ease of use (though it is only potential for many of the solutions, as the interface is typically a follow-up concern of many of these initial reports).

Secondly, they offer the ability to be fast. Though network connectivity issues and latency may concern remote solutions, local solutions offer low-level integration and direct disk or memory-based access to file metadata.

2.1.3 Shortcomings

Many other similar solutions exist, but with all of these systems, the issues of adoption and technical requirements stand out as the primary barriers to success. They all lack either a graphical interface entirely or, at the very least, do not have one that would meet standard consumer expectation. They are also technically niche implementations. A solution that seeks to integrate at this level of the application hierarchy would likely require backing and integration from a major Operating System developer to become practically feasible to the end user (i.e., a non-technical user would not be able to install or operate many of the solutions in their current state). This also alludes to the issues of adoption and economies of scale that will be highlighted below.

Additionally, though many of these solutions show promise for local storage, the reality of the distributed nature of collaboration and social computing means these solutions miss the communal element of metadata altogether. And though F2R begins to hint at such a need, it falls short in several key areas, as mentioned specifically above. Remote and distributed systems must therefore be considered.

2.2 Remote Object Metadata

Metadata often has both a personal and communal function. Consider the modern publication scenario where an author publishes a paper with an associated

dataset. The paper is likely published in a printed journal and perhaps their online digital library as well. The paper may make reference to the generated dataset. That data is then stored in an online repository such as the one offered by the author's institution, the Purdue University Research Repository. The repository makes a place for the data to live for the long term. It also offers digital preservation. The digital object identification system described below can help with maintaining the appropriate location of that digital asset. But, the paper and dataset are often subsequently downloaded to a user's local hard drive, and all associations are lost between the file and the dataset, and even between the dataset and the location from which it was retrieved. Furthermore, as that dataset is shared directly with colleagues and collaborators, the origins and details of the dataset can be lost completely.

The local systems discussed in the previous section offer significant steps forward in terms of augmented file metadata. But, with the saturation of modern cloud-based file sharing mechanisms, a new set of issues has arisen. Not only do users need to annotate files on one machine, they now expect to share those annotations across all of their devices (or at the very least, all of their traditional mouse-based personal computing devices). Additionally, files shared between collaborators and colleagues also need to retain any additional metadata given them by the previous steward of the file.

2.2.1 Solutions

Particularly in the online space, metadata currently fulfills one of two primary purposes. The first is that of the digital preservation environment. In this scenario, metadata is principally concerned with identifying and augmenting the data necessary to maintain the primary files or datasets ad infinitum; as will be described below, this is a relatively common use case for iRODS. Though not mutually exclusive, metadata can also constitute augmented information about the

files themselves. This may mean descriptions of column headings, software packages used to derive the current data, provenance, versioning, and more. The later of these two is that with which this research is concerned.

It is important to understand which other contenders are situated in this space and what their focuses are, to better understand the gaps and identify strengths and weaknesses. There are several primary standards that contribute to the metadata domain. Though they may not be specifically targeting the same problems that this research has identified, they offer solutions to pieces of the problem, and pose important background to further expose the solution being pursued. These include the Digital Object Identifier (DOI) standard, the Open Annotation Data Model, and document collaboration solutions.

2.2.1.1. Digital Object Identifiers

The DOI system has been in establishment since 1997 and managed by the International DOI Foundation since 1998 (Chandrakar, 2006). DOIs offer a persistent pointer to a digital asset. Given the evolutionary nature of the Internet and developing technologies, this allows content to move digital locations and still have a consistent pointer that remains intact long after other content may have made its initial reference to it. As Wang (2007) notes, this seems to be the predominant perception, that the preeminent feature of the DOI system is persistence.

To exemplify this, suppose a researcher writes Paper A. They then publish that paper and make it available on their personal website, www.myresearch.com. Then another researcher uses that work to write their paper, Paper B. They give credit to the original author and say that her work can be found at www.myresearch.com. Now suppose that the original author purchases a new domain name and moves her work to that new location. The second author's reference to the deprecated domain is now invalid, and subsequent readers of Paper

B will no longer be able to reference the cited work. This is the problem resolved by the DOI system (no pun intended).

Instead of referencing www.myresearch.com, the second author could reference the DOI handle that the first researcher created. Initially, the handle would resolve to www.myresearch.com and the net result would be the same. But, after the first author changes domain names, she could also update the DOI to point to the new domain. And in so doing, the second author's citation remains valid.

In addition to maintaining the resolution between DOI and the digital asset location, the DOI system requires core metadata to be included with each DOI entry. Wang (2007) identifies this metadata as kernel metadata, and includes such basic attributes as title, type, mode, and other key fields. Though potentially perceived primarily as a persistence system, the DOI structure does offer a fairly robust metadata management scheme. According to the *DOI handbook* (2013), there is no restriction placed on the metadata to be included in the DOI, except that it meets the DOI kernel requirements. The kernel itself has two primary purposes, recognition and interoperability (*DOI handbook*, 2013).

That being said, the extended metadata model supported by the DOI system is still constrained by the standard of established and accepted metadata schemas (*DOI handbook*, 2013). According to Paskin (2004), DOIs that may have some use outside of the immediate DOI system itself are subject to certain metadata policies.

2.2.1.2. Open Annotation Data Model

Contrasting the persistence orientation of the DOI system, the Open Annotation Data Model seeks to create a framework for open and shared relationships and annotations amongst and about web objects. Annotations are an essential element of scholarly evolution and facilitate the organization of knowledge and the inception and dissemination of new knowledge (Sanderson & Van de Sompel, 2010). The Open Annotation Data Model is a World Wide Web

Consortium (W3C) community draft specification, and as such, is clearly defined as, and interesting in being, oriented to the structure of the World Wide Web (Sanderson, Ciccarese, & Van de Sompel, 2013). These annotations could be used, functionally speaking, to attach metadata to web resources. To put it another way, content metadata is a subset of the overall use case of open annotations.

In their initial work predating the formal introduction of the Open Annotation Data Model, Ciccarese, Ocana, Garcia Castro, Das, and Clark (2011) lay out the need for annotation, specifically in the scientific and academic realm. Here they describe the disassociation between well-defined domain-specific ontologies and the papers that cover such topics (Ciccarese et al., 2011). Though slightly different than the ultimate goals established by the later Open Annotation Data Model, it highlights the need to link, or describe, scientific literature by other established ontologies. From this, the need to openly annotate web resources with arbitrary data evolved.

To a certain extent, there is nothing revolutionary about web annotations. Consider an image posted to Facebook with several comments from friends and acquaintances. These are, in essence, annotations. They are textual content that is attached to that web resource (*Open annotation data model*, 2013). That being said, as the *Open annotation data model* (2013) notes, such annotations are often proprietary and non-portable, thus the need for an open and interoperable standard. But, to be clear, it is a just that, a standard, not an implementation. As the working draft of the data model explicitly states:

> The Open Annotation system does not prescribe a transport protocol for creating, managing and retrieving annotations. Instead it describes a web-centric method, promoting discovery and sharing of annotations without clients or servers having to agree on a particular set of network transactions to communicate those annotations (2013, Introduction section, para. 4).

Though several have begun experimenting with implementations of the standard, it remains a relatively immature field. Haslhofer, Simon, Sanderson, and van de Sompel (2011) describe the application of the model using several use cases in their work on the matter. They developed a JavaScript prototype for annotating images with SVG elements. Ironically enough, their paper contains a direct link to the prototype that is no longer valid.

In spite of the lack of concrete implementations, one function espoused by the Open Annotation Data Model is its intent and ability to scale from the incredibly simple use case to the far more complex examples (Sanderson et al., 2013). Cole and Han (2011) also note that the annotations are capable of referencing multiple entities, a feature of the Open Annotation Data Model that is critical in this environment.

Beyond the basic manifestation of the model, Sanderson and Van de Sompel (2010) indirectly highlight the convergence of the Open Annotation Data Model and the DOI system (though they propose different solutions) when they identify the issue of annotating a living and evolving entity (a dynamic web page) with content explicit to a specific version of that entity. This is an inherent architectural issue of the web, and the need to think about how an asset can be uniquely identified is something that will be addressed in the definition of a deterministic asset key in the following chapters.

In all of this, it is always important to evaluate the central purpose of the system. What is the key driving principal propelling the implementation and evolution of a given technology? The Open Annotation Data Model is ultimately concerned with creating and sharing annotations of web resources. This is critical in understanding whether or not it will meet the stated needs of this research.

2.2.1.3. Document Collaboration Solutions

In searching for solutions that offer a collaborative and communal metadata environment, several metadata-oriented services have been examined above, namely the DOI system and the Open Annotation Data Model. These solutions focus on associative functions such as metadata and relationships between web objects. Approaching the problem of metadata storage from the inverse perspective, that of the files themselves, also reveals several potential competitors. These solutions offer as their primary purpose, collaborative document development capabilities rather than metadata storage. The collaborative nature of these solutions is made manifest by features such as distribution of storage, versioning, and other similar mechanisms. Examples of products in this realm include the Integrated Rule-Oriented Data System (iRODS), Git, Dropbox, and Google Drive.

The beauty of these document collaboration solutions is that many of the tools in this space are already familiar to academics, professionals, and students alike. Whereas the average reader may be unaware of the implications of the Open Annotation Data Model, it is likely that the reader is actively using, or has used in the recent past, a Google Drive document to collaborate on the development of a proposal or spreadsheet, a Dropbox account to quickly and easily share a file, or GitHub to host an application or piece of code in a distributable and traceable manner. These solutions offer extensive collaboration and synchronization benefits. In some senses, they are the antithesis of the local solutions referenced earlier in this chapter. They offer all of the communal benefits of file development, with little to no explicit metadata value. These solutions will be briefly examined to better understand how they are positioned in this technological space, and what advantages and disadvantages they pose over the other above-examined solutions.

The descriptions below are not intended to be as thorough as other lesser known applications in this chapter, with the exception of iRODS, as these solutions are assumed to have achieved relative market saturation.

2.2.1.3.1. Integrated Rule-Oriented Data System

Though significantly larger in scope than a pure metadata management platform, iRODS offers a metadata catalog, called iCAT, for arbitrarily describing data objects (*iRODS technical overview*, 2014).

iRODS offers extensive features for data management, including data virtualization, file metadata, workflow automation, and collaborative tools (*iRODS technical overview*, 2014). Specifically related to file metadata, the iCAT uses a standard relational database to store internal metadata necessary to the functioning of the iRODS server, as well as arbitrary user defined metadata concerning data objects or even data collections (*iRODS technical overview*, 2014).

With such a rich feature set, though, comes an unfortunately complex environment. Assuming one is in need of the breadth of features offered, iRODS proves fruitful for advanced workflows and complex preservation environments. To this end, Hedges, Hasan, and Blanke (2007) highlight the use of iRODS and its metadata components in an archival preservation environment. But, for simple metadata management, it could be considered fairly cumbersome.

2.2.1.3.2. Google Docs/Drive

Google Docs offers online and collaborative creation of standard text documents, spreadsheets, presentations, and forms. The power of Google Docs can be found in its unique multi user editing environment, revision history, and automatic saving (*Google Docs*, n.d.). That being said, Google Docs does suffer in several capacities, including output formating and complexities, off-line support, and browser requirements (Dekeyser & Watson, 2006). As a compliment to Google Docs, Google Drive offers synchronized file storage space for native Google Docs or other standard file types.

2.2.1.3.3. Dropbox

Dropbox focuses on offering its users a safe and central location for file storage and sharing. It allows for the individualized distribution of files to collaborators and peers on projects or homework. Dropbox also offers enhanced functionality and support levels through its Dropbox for Business service (*Dropbox for Business*, n.d.). Unlike Google Docs, Dropbox functions primarily as a synchronization service, rather than an online editing environment (Marshall & Tang, 2012).

2.2.1.3.4. Git

Git is a slight divergence from the previous two examples. With its primary focus being on versioned code development, it offers a decentralized repository for document development (Chacon & Straub, 2014). And though its community is primarily composed of software teams and open source projects, it also has value for standard documents of a reasonable size. Git functions well for a variety of use cases, including, as a lab notebook, facilitating collaboration, preventing data loss, fostering exploratory freedom, soliciting feedback, increasing transparency, and lowering barriers to reuse (Ram, 2013).

By contrast to the previously discussed remote solutions, with the exception of iRODS, these services do not offer built-in features for the definition of arbitrary metadata. Using one of these solution, one sacrifices explicit metadata functionality for syncing and collaborative features. To make these solutions viable competitors in the collaborative metadata environment, one could include an ancillary metadata file and keep it stored within the collaborative space. Others have attempted to use folder structures and naming schemes for metadata purposes. As long as users are directly associated with the project, that implicit data will be included in the synced or downloaded versions shared across devices or between users. Also captured is standard file system level metadata, including ownership and modified dates.

2.2.2 Advantages

Considering the three types of remote solutions discussed, many advantages exist concerning the use of these systems, and are more or less the inverse of the disadvantages identified above regarding local metadata solutions. Broadly speaking, these solutions are communal in nature, offering online and shared visibility. They also require little to no end user requirements for installation and use, as they are managed by third party organizations and require a standard web browser for interaction. Some solutions also make simple and intuitive user client applications available that can be installed locally to emulate standard direct file system functions to which users are accustomed.

In addition to the generic advantages offered by solutions in this category, other individual advantages also emerge. For example, Git excels in the area of versioning. Git was developed for this purpose. Furthermore, particularly with the advent of service such as GitHub, GitLab, and Git desktop clients, Git has significantly decreased its barriers to entry for the average user.

Google Docs handles versioning and change management very well. According to the Google Help page entitled *View and manage file versions* (n.d.), Google retains file versions for Google and non-Google files alike, keeping track of native Google Doc changes indefinitely and cataloging up to 200 versions of non-Google files (i.e. files not created and edited through the Google Docs platform). In addition to the online editing functionality made available by Google Docs, Google also offers complementary services for file storage and syncing across devices through the use of their Google Drive service.

2.2.3 Shortcomings

In spite of the advantages of these remote systems, several shortcomings arise upon further examination of these solutions.

2.2.3.1. Usability

Issues surrounding usability include the lack of user customization and the absence of directly accessible mechanisms for metadata input. DOIs, practically speaking, have several layers of indirection from the average user. A third party issuer on behalf of a publication platform typically mints DOIs. The involvement of the publisher is often automatic, thus making it deterministic and not highly customizable. Tangential third parties are also unable to augment the annotation.

2.2.3.2. Functionality

As much as the document collaboration solutions excel in collaborative functionality, communal orientation, and user accessibility, they suffer significantly in terms of explicit functionality for metadata support. Forcing the user to capture extended metadata in an auxiliary location, while reasonable, is functionally no different than other existing file system solutions. While it may make it easy to share, if what is being shared is not well defined and managed, the problem persists.

2.2.3.3. Dependencies

Native to both open annotations and DOIs is the requirement that a publicly available web resource be accessible to which the annotations or DOIs can point. It is thus dependent on a public web entity. Though not a significant concern for published papers, private and developmental data should not have to come with the burden of maintaining a fixed public presence in order to be annotated. The document collaboration solutions do not suffer from the same problem, but conversely lack a truly public dissemination mechanism.

2.2.3.4. Persistence

None of the above-examined solutions is capable of retaining metadata once abstracted from the repository. By this, it is meant that files retain no persistent and inherent connection to anything in the original repository once removed - similar to the scenario about downloading a file and dataset, as described in the DOI section above. When in the repository, they are implicitly related, but once downloaded, that connection is lost.

To exemplify this, test implementations of the Open Annotation Data Model store annotations locally on the server where the annotated items live. This seems intuitive, but requires the user to travel to that site to discover (or rediscover) the annotations pertaining to that resource. This is to say, if the user downloads an item, those annotations are lost. And unless the users makes note of, or remembers where the file was retrieved from, those annotations remain detached.

Principally speaking, the DOI system suffers from the same issue. The metadata is stored remotely and the files are stored locally. By convention, the DOI is often included in the body of the paper, making it easier to retrieve. But, that is not a requirement.

2.2.3.5. Implementation

The Open Annotation Data Model does not actually offer an implementation; it is just a model. It offers an excellent framework for describing web-based resources, but there is still growth and evolution necessary before it becomes a commonly implemented standard. Until then, it is practically unapproachable.

Ultimately, none of these systems have metadata management as their primary purpose. In other words, though they can be construed to serve that purpose, it is not their intended goal. As such, they cannot be ensured to have metadata management as their principle driving requirement in the future. While

the DOI system and Open Annotation Data Model have gained some traction in the research community, as highlighted by Ciccarese, Soiland-Reyes, and Clark (2013), adoption continues to be a crucial issue in creating the economies of scale necessary to make these solutions succeed in the long term.

2.3 Summary

This chapter provided an overview of the existing literature surrounding metadata management systems. These systems are composed primarily of two types: local, file system based architectures; and remote, web-based asset annotation systems. While local systems offer direct user accessibility and customization, they lack the distributive functions necessary for multiple environments and shared annotation. And though remote systems counter on those points, they lack the user accessibility and non web-based features needed to fulfill the requirements presented by this research. Thus, this paper now turns to describe the methodology used to identify and quantify these existing solutions to determine if a new solution is needed.

CHAPTER 3. FRAMEWORK AND MEASUREMENT METHODOLOGY

Having identified the existing solutions and considered many of their benefits and shortcomings, it is important to now consider the way in which one will know whether or not there is potential to capitalize on these gaps through the creation of a new metadata solution. To do so, a measurement and evaluation mechanism was established that is independent of any of the particular solutions being described.

3.1 Research Question

Driving this research was the basic question: Can a new metadata system be proven necessary, designed, and prototyped that facilitates the sharing of arbitrary file metadata across file systems that is collaborative in nature, platform independent, and easily adopted?

As identified in the previous chapter, a significant gap exists in the user-defined file metadata space. Local file metadata systems appear simply inadequate to capture the necessary file metadata for modern applications and are isolated to a single environment. Distributed systems are often overly restrictive and may require a publicly web-accessible asset or large online file repository.

3.2 Hypothesis

Given these shortcomings, the hypotheses set forth by this research can be summarized as follows:

H_{o1} An improved metadata management solution is not needed to meet the needs established by this research.

H_{a1} An improved metadata management solution is needed to meet the needs established by this research.

H_{o2} An improved metadata management solution cannot be constructed that achieves 75% or greater coverage of requirements.

H_{a2} An improved metadata management solution can be constructed that achieves 75% or greater coverage of requirements.

3.3 Comparison Model

When comparing software applications, or even considering building a new one, it is important to quantify the existing competitive landscape. What else exists in that technical space and on what functional areas do they compete? Chapter 2 offered an overview of the metadata software landscape, along with initial observations of shortcomings and gaps. It then became important to compare the solutions to one another and evaluate how well they satisfied the problems outlined in chapter 1. According to Langer (2011), important factors in this decision include coverage, which is concerned with how well the product meets the needs of the customer, and direction, which includes the ability to customize existing solutions to best aligned with the user's core needs.

Software comparison methods offer several means to complete this task, including the Weighted Sum Model and other technical performance based models. Given the prototypal nature of many of the solutions analyzed, it became difficult to evaluate the applications based directly on technical performance. Many of the solutions were not available to be installed or analyzed. Additionally, it was important to be able to identify whether or not a better solution could and should be designed, prior to implementing it in a manner sufficient for prototyping, let alone a strenuous technical performance evaluation.

It thus became advantageous to use the Weighted Sum, or Weighted Scoring, Model for decision-making analysis. The question to be answered, from another perspective, was essentially that of a standard build versus buy analysis. To accomplish this task, the criteria for comparison were established. Those criteria were then ranked according to importance in the design specification. Finally, each solution was then considered in a matrix fashion against each criteria, rendering a final score for each product. This process was fulfilled in the creation of a Decision Analysis Spreadsheet, as outlined by Bandor (2006).

It is important to note that many of the solutions considered, especially in the remote realm, offer both a client and a server. Some criteria defined below may apply generically, irrelevant of the implementation architecture, and some may apply more specifically to either the client or the server (if applicable). When that is the case, this distinction was noted as the criteria were considered.

3.4 Evaluation Criteria

Based on the survey of existing solutions found in the prior chapter and the subsequent gap analysis, along with the research question put forth by this study, the following key criteria were identified for a successful and differentiated product to meet the requirements implicit in the problem statement found in chapter 1.

The criteria identified below fall into one of two primary categories. The first is that of functional needs, establishing whether or not the product does what it must do to meet the core needs of the application. The second set consists of implementation and adoption criteria, evaluating environmental and deployment factors.

3.4.1 Key/Value Definitions

To put it simply, this seeks to understand how well the solution actually offers services for storing metadata. Is the metadata storage explicit or implicit?

How flexible is the storage and is it built into the solution or a byproduct of the way it is used? Does it support only a defined set of metadata key/value pairs, or is it sufficiently flexible for new areas of interest or even unknown applications? This, along with the area of collaboration, were the key functional areas.

In this specific research inquest, there was a higher value placed on creating a sufficiently flexible environment for metadata definitions. This is to say that it is both capable of storing such standard elements as Dublin Core, as well as other domain-specific ontologies or even unknown applications in the future. Though to some areas of research, structured metadata is highly valued, in this application, a lack of rigidity was prized for the sake of flexibility.

3.4.2 Collaborative Nature

Collaboration was a critical aspect of this research. Being able to share metadata between users is the manifestation of that collaboration. This category looked at how well the solution facilitates input from its users. Is it limited to some externally defined group or all interested parties? How easily is the generated metadata shared with collaborators and external participants? How easily can users contribute back to the metadata definition?

In addition to the metadata definition group and collaboration sphere, it was important to also consider how transportable the metadata is. For example, once a file has been downloaded from its repository, does it retain its metadata or at least a reference to it, or is the metadata only applicable in the original context for which it was defined?

3.4.3 Platform Support

This criteria sought to understand on which different environments and operating systems the solution can be used. Is it constrained to primarily one type of environment, or is it flexible enough to support many different locales? This

question was particularly pertinent to clients, but could also apply to servers if the server must be self-hosted in order to use the solution. This ultimately sought to point to the reality that adoption and economies of scale hinge significantly on whether or not users can use the solution on a broad range of existing devices and products.

3.4.4 Adoption

According to Bonneau, Herley, Oorschot, and Stajano (2012) in their analysis of password mechanisms and security, they note that there is an important third category above and beyond the traditional dichotomy of usability and security: the category of deployability.

In keeping with this observation, widespread adoption is critical to success in a scale driven environment such as collaborative metadata. In order to facilitate adoption, the solution must be both platform independent, as noted by the previous category, and easily deployable. The issue of adoption is also fairly interconnected with the issue of cost, but considers specifically how hard it is to deploy the solution. Can an average user manage the deployment, or does it require extra or specialized training? This particular category was primarily concerned with the server but could have implications for the client as well.

3.4.5 Versioning and Provenance

This requirement sought to understand how well the solution can track changes within the files themselves. Versioning looks at the contents of the file, whereas provenance is concerning with context through external inputs, such as who changed the file, when, using what other applications, for what purpose, and other similar questions. The solution did not necessarily have to track the unique contents of the individual versions, but should be aware of the existence of other versions, and the relationships between them.

3.4.6 Security

In the context of this research, metadata definitions and collaborative features were targeted at trying to facilitate open annotation within a specific context or domain. Meaning, the definitions would likely be shared within the context of a scientific discipline or research group, and may actually be hierarchically distributed. As such, security may be both a function of the application or the domain implementation, depending on the target scope of the metadata service. That being said, how does it handle user identification and authentication? Does it require an authenticated user? Does it support some amount of anonymous functionality?

Security is also a function of the requirements of a given system. For example, saying that a system requires an authenticated user, though seemingly more secure, may be a disadvantage to the purpose of a particular project, especially if that project wants to support anonymous interaction.

Finally, it is important to note that this is not a discussion of the security of the application code itself and any susceptibility it may have to exploit. Given the nature of many of the solutions and prototypes being analyzed, understanding the actual security level of the software is impractical.

3.4.7 Distribution

Along with versioning and provenance, and security, distribution was the last of the extended models identified by the secondary research questions. Here, the analysis attempted to discern how the application is architecturally structured, whether it be centralized or decentralized. Furthermore, if possible, understanding whether the solution manifests any sort of distributed or hierarchical structure was also captured here. This ultimately points to how the application may work in such a way as to facilitate concentric circles of metadata coverage, context, and sharing.

3.4.8 Adaptability/Extensibility

As the build versus buy debate often considers, it is important to understand that even though an existing solution may not offer all needed features, it may offer the ability to easily extend its native functionality. Thus, how easy can the solution be customized or extended? Again, this is not necessarily for the average end user, but important when considering adoption by groups needing custom behavior or extended functionality.

3.4.9 Cost

While software may be free, it is not without cost. This criteria, in conjunction with adaptability and adoption, sought to understand those implicit and explicit costs (Langer, 2011). Does the software have a purchase price? How much does it cost to initially install? How much does it cost in ongoing maintenance? Does the user have to host both the client and the server, or are free hosting services available? The lower the initial and ongoing cost, the better score an application received in this category.

3.4.10 Risk

Risk can seem intangible, but it is extremely important to consider. Are users being asked to be an early adopter of the solution, or the user of a clearly established standard or service? And subsequently, if required to use some form of centralized server model, is that service clearly established and trustworthy?

3.5 Ranking

Having established the criteria of interest, those individual categories were ranked and weighted according to their overall importance to the project. The weighted sum of those rankings evaluates to 100% of the total possible weighted

score (Bandor, 2006). The above criteria have been defined in order of importance, and are weighted as shown in table 3.1.

Table 3.1. Weighted Ranking

Criterion	Weight
Key/Value Definitions	20%
Collaborative Nature	20%
Platform Support	10%
Adoption	10%
Versioning and Provenance	10%
Security	10%
Distribution	5%
Adaptability/Extensibility	5%
Cost	5%
Risk	5%

The criteria identified were ordered respective of their inclusion within the research questions of chapter 1. The first two elements, those of metadata definitions and collaboration, were ranked highest at 20%, as they were the core functional elements established and required by this research. The subsequent items of platform support and adoption were the non-functional requirements of the primary research question, and as such, were ranked just below the primary functional requirements, at 10%. The secondary research questions were those of versioning, security, and distribution. Distribution was ranked subtly less than the other two, as it is the least likely to significantly affect a successful initial implementation of a prototypal product. Lastly, adaptability, cost, and risk, were identified as critical aspects of any real-world analysis and adoption of a new product (Langer, 2011). Though they were important concerns and should weigh

into the discussion, they were weighted least, at 5%, for the purposes of this initial analysis as they are the most subjective based on the specific adopter.

3.6 Scoring

Each solution was evaluated on the established criteria and given a raw score. According to Bandor (2006), it is important to provide a scale from -1 to +1 for the definition of the raw score. This allowed for both the idea of negligence of a feature (a score of 0), as well as the explicit detraction of a feature (the score of -1). For example, one solution may only be available on one platform. This would have received a raw score of -1.0. On the contrary, if a solution were centrally hosted by a third party, platform support would be irrelevant, and received a 0.

The score scale is described graphically in table 3.2.

Table 3.2. Scoring Scale (Bandor, 2006)

Definition	Score
Perfect match	+1.0
Partially fulfills requirement	+0.5
Neglects requirement	0.0
Partially detracts from requirement	-0.5
Completely detracts from requirement	-1.0

3.7 Summary

This chapter established the methodology and criteria with which the existing metadata solutions can be compared. The weighted sum model was chosen,

with 10 criteria included in the analysis. With those criteria in mind, the competitors identified and described in chapter 2 will be evaluated.

CHAPTER 4. ANALYSIS

Utilizing the criteria established by the previous chapter, the competitors identified and described were evaluated to determine their strength against the criteria. A brief written description summarizing the solution and criteria will be given, followed by a numeric score. Each solution's scores will be summarized, and a collective summery will be given at the end of the chapter.

4.1 Linking File System

The Linking File System was the first of the local solutions considered as a comparable in this investigation. It offers the ability to link files and describe those links and the files themselves using key/value metadata pairs.

4.1.1 Key/Value Definitions

LiFS received a 1.0 for its ability to arbitrarily define key/value pairs and establish links between files. This core functionality of the application is a thorough implementation and serves to solve the metadata definition problem.

4.1.2 Collaborative Nature

By definition, local solutions all generally received a -1.0 for their collaborative nature. This was true of LiFS as well.

4.1.3 Platform Support

The prototype of LiFS was implemented for the Linux platform, and requires the FUSE userspace module to function (A. Ames et al., 2005). Due to these restrictions, it received a -0.5. Given the widespread adoption of Linux and the availability of FUSE within the Mac operating system it was not considered to completely detract from the criteria.

4.1.4 Adoption

As with many solutions in this space, their lack of maturity contributed to a difficulty in adoption. Some solutions, such as LiFS, were not actively available on the market, and thus received a -1.0. Solutions that are available, but only in a prototype phase received higher marks.

4.1.5 Versioning and Provenance

Though it offers no explicit promise of versioning or provenance, the ability to define links between files could function as a version history. Links could also be used to identify a history of contributing artifacts leading to a rudimentary understanding of provenance. Given these possibilities, LiFS received a 1.0 for versioning and provenance.

4.1.6 Security

In the context of all local file system solutions, security is not applicable as the data is not shared between hosts or other entities. For this reason, LiFS received a 0 for security.

Whereas all local solutions received a -1.0 for collaboration, they all received a 0 for security. The difference between the scoring of the two rests in the fact that

the lack of security is not applicable to a local solution, whereas a lack of collaborative nature is functionally lacking a core requirement of this research.

4.1.7 Distribution

As with security, local solutions do not have a potential application of distribution, as they do not conform to the client server architecture. For this reason a 0 was awarded.

4.1.8 Adaptability/Extensibility

The ability to adapt and extend the functionality of LiFS was unknown. For this reason, it received a 0 on this criteria.

4.1.9 Cost

For solutions only implemented in prototypal form, understanding cost was an impossible endeavor. In this context, given the academic nature of many of the available solutions, it was likely safe to assume that they would have been offered at little to no cost. For that reason, given the benefit of the doubt, LiFS received a 1.0 for cost.

4.1.10 Risk

For all products in the development and prototypal phases, risk is always high. There are concerns of maintainability and support over the long term, and risks related to unknown factors such as scaling and security. For this reason, LiFS received a -1.0 on the issue of risk.

A summary of the LiFS weighted sum results can be seen in table 4.1.

Table 4.1. Weighted Sum Model for LiFS

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	1.0	20%
Collaborative Nature	20%	-1.0	-20%
Platform Support	10%	-0.5	-5%
Adoption	10%	-1.0	-10%
Versioning and Provenance	10%	1.0	10%
Security	10%	0.0	0%
Distribution	5%	0.0	0%
Adaptability/Extensibility	5%	0.0	0%
Cost	5%	1.0	5%
Risk	5%	-1.0	-5%
Total	100%		-5%

4.2 Quasar File System

The Quasar File System is a functionally similar implementation to LiFS, offering comparable strengths and weaknesses, though it focuses more on unifying the location of both file data itself and its descriptive metadata. In so doing, it poses potential improvements in terms of performance, but no significant functional changes from that of LiFS. For this reason, no further analysis of QFS will be provided. The results of QFS are summarized in table 4.2.

4.3 File System as Linked Data

The last of the local solutions, the File System as Linked Data, or F2R, offers an interesting hybrid of the local perspectives with its attempt to bridge the gap between a strictly isolated solution and a more robust collaborative model.

4.3.1 Key/Value Definitions

As with LiFS and QFS, F2R offers as a core feature the ability to describe assets using key/value definitions. Furthermore, it proposes the idea of augmenting its user-defined metadata with certain automatically generated metadata elements. Though somewhat irrelevant to this discussion, this does pose interesting possibilities in this space of collaborative metadata generation. A 1.0 was given for this category.

4.3.2 Collaborative Nature

As noted above in the section on LiFS, all local solutions should inherently receive a -1.0. But, F2R introduces the ability to publish the contents of a file system to the web in a semantic context. Unfortunately, this does little to aid in the current endeavor, offering no practical value to the non-automated consumer of the shared metadata. A -1.0 was thus still awarded.

Table 4.2. Weighted Sum Model for QFS

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	1.0	20%
Collaborative Nature	20%	-1.0	-20%
Platform Support	10%	-0.5	-5%
Adoption	10%	-1.0	-10%
Versioning and Provenance	10%	1.0	10%
Security	10%	0.0	0%
Distribution	5%	0.0	0%
Adaptability/Extensibility	5%	0.0	0%
Cost	5%	1.0	5%
Risk	5%	-1.0	-5%
Total	100%		-5%

4.3.3 Platform Support

The prototype of F2R was implemented in Java, making it, in theory, cross platform (He et al., 2013). But, given the larger degradation in support of Java, especially for Mac, a 0.5 was given for this category.

4.3.4 Adoption

As with many solutions in this space, their lack of maturity contributes to a difficulty in adoption. Some solutions, such as F2R, were not actively available on the market, and thus received a -1.0.

4.3.5 Versioning and Provenance

Though it offers no explicit promise of versioning or provenance, the ability to define metadata about files could function as a version history. Given these possibilities, F2R received a 0.5 for versioning and provenance.

4.3.6 Security

In the context of all local file system solutions, security is not applicable as the data is not shared between hosts or other entities. Furthermore, assuming the choice to published metadata to the semantic (and open) web is a purely voluntary action, security still primarily remains irrelevant. For this reason, F2R received a 0 for security.

4.3.7 Distribution

F2R, in its server architecture, is not distributed. But, in publication, it could in theory allow for the distribution of file system metadata to multiple

destination. Though not defined, a 0.5 was awarded for the potential application of this solution in a distributed manner.

4.3.8 Adaptability/Extensibility

The ability to adapt and extend the functionality of F2R is relatively unknown. According to He et al. (2013), the application is developed in a modular manner, implying at least some amount of customization potential. For this reason, it received a 0.5 on this criteria.

4.3.9 Cost

For solutions only implemented in prototypal form, understanding cost is an impossible endeavor. In this context, given the academic nature of many of the available solutions, it is likely safe to assume that they would have been offered a little to no cost. For that reason, given the benefit of the doubt, F2R received a 1.0 for cost.

4.3.10 Risk

For all products in prototypal phase, risk is always high. There are concerns of maintainability and support over the long run, and risks related to unknown factors such as scaling and security. For this reason, F2R received a -1.0 on the issue of risk

A summary of the results of the F2R analysis can be seen in table 4.3.

4.4 Digital Object Identifiers

The Digital Object Identifier system is managed by the International DOI Foundation and allows for persistent handles to web objects. The DOI system is a

Table 4.3. Weighted Sum Model for F2R

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	1.0	20%
Collaborative Nature	20%	-1.0	-20%
Platform Support	10%	0.5	5%
Adoption	10%	-1.0	-10%
Versioning and Provenance	10%	0.5	5%
Security	10%	0.0	0%
Distribution	5%	0.5	2.5%
Adaptability/Extensibility	5%	0.5	2.5%
Cost	5%	1.0	5%
Risk	5%	-1.0	-5%
Total	100%		5%

client-server based architecture, where DOI registration is managed by decentralized and vetted Registration Agencies (RA), and the client is traditionally a web browser (*DOI handbook*, 2013).

4.4.1 Key/Value Definitions

The DOI scheme does support the definition of key/value metadata. That being said, the metadata standards are primarily intended to be based on the Dublin Core or an established domain ontology. For this reason, and the relative fixity of the metadata once published, it received a 0.5 in this category.

4.4.2 Collaborative Nature

The DOI system is intended to be collaborative and public in nature. That being said, it is not a living or collaborative metadata store to the extent required by this research. Metadata is only defined by the publisher at the time of publication. Additionally, metadata is loosely coupled to the object and can be easily lost. For these reasons, a -0.5 was given for collaboration.

4.4.3 Platform Support

Platform support for the server is less important, as it is not generally the responsibility of the user wishing to adopt the solution to set up their own server (though this may be considered a disadvantage to some). Given that the client is primarily a web browser, there is little to no platform dependence. But, understanding the lack of flexibility on the server side, it was only awarded a 0.5 for this criterion.

4.4.4 Adoption

Adoption is based on access to an existing service that supports the DOI. To use the DOI service, an appropriate Registration Agency must be identified. If one does not exist, a new one may have to be formed. It therefor may not be an applicable solution for all interested parties, depending on the availability of the assets being described (referring here to the requirement for publicly available assets as noted in the review of relevant literature). A -0.5 was thus awarded.

4.4.5 Versioning and Provenance

DOIs can and should be assigned to explicit versions of the objects they represent. In addition, those objects could in theory reference other objects, and thus fulfill a form of versioning and provenance. It thus warranted a 1.0 on this criterion.

4.4.6 Security

The security and validity of the metadata given for an object is managed by the registering agency. Security is therefor primarily a factor of the individual registration agency systems. In the absence of any other contrary information, a 1.0 was given for this category.

4.4.7 Distribution

The DOI system is by nature distributed, in that it is divided by registration domains identified by handle prefixes (*DOI handbook*, 2013). Though this does not necessarily constitute a hierarchical system, it is distributed, and a 1.0 was therefor given in this category.

4.4.8 Adaptability/Extensibility

There is little to no concept of user adaptability or extensibility in the DOI system, as it is strongly standardized in both use and implementation. A -1.0 was thus appropriate here.

4.4.9 Cost

Cost is dependent on whether or not an existing service entity is available and suitable to be used by the client. Furthermore, membership in the DOI Foundation may also include a cost (though is not required to utilize the service). Assuming the DOI system is used within the larger context of an existing RA, little to no cost will be assumed, and thus warranted a 1.0 in this category.

4.4.10 Risk

The DOI system is well established and has been in use since 1997 (*DOI handbook*, 2013). It has earned widespread acceptance and offers a valuable maturity when selecting a third party metadata service. It was given a 1.0.

A summary of the results of the DOI analysis can be seen in table 4.4.

4.5 Open Annotation Data Model

The Open Annotation Data Model, which is the second of the remote solutions discussed, posses high promise in the metadata realm. But, as discussed in the review of relevant literature, its lack of solid implementation is a significant hindrance to adoption and use.

Table 4.4. Weighted Sum Model for DOI

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	0.5	10%
Collaborative Nature	20%	-0.5	-10%
Platform Support	10%	0.5	5%
Adoption	10%	-0.5	-5%
Versioning and Provenance	10%	1.0	10%
Security	10%	1.0	10%
Distribution	5%	1.0	5%
Adaptability/Extensibility	5%	-1.0	-5%
Cost	5%	1.0	5%
Risk	5%	1.0	5%
Total	100%		30%

4.5.1 Key/Value Definitions

The Open Annotation Data Model takes a slightly different structural approach to the metadata definition aspect of the solution. Instead of explicitly allowing arbitrary key/value definitions, the data model defines a semantic language for describing and relating web objects. Though not exactly the same as many of the other solutions, this should prove sufficient to meet the requirements of this research. A 1.0 was therefor given.

4.5.2 Collaborative Nature

Though uniquely targeting a semantic definition to allow for the sharing of annotations, the Open Annotation Data Model does not actually advocated a user to user centric approach. For this reason, it only received a score of 0.5 for collaboration.

4.5.3 Platform Support

This issue cannot be addressed without proper implementations to analyze. Given, though, that this is intended to be a cross platform and cross domain specification, it was given a preemptive evaluation of 1.0.

4.5.4 Adoption

In a similar manner to the costing issue, the requirement for a home-grown implementation to mechanize the model poses significant hindrances to adoption. A -1.0 was given. This could be improved in the future by standards-compliant clients and servers being introduced to the market.

4.5.5 Versioning and Provenance

Understanding that the Open Annotation Data Model's intent is to describe web objects, versioning and provenance can pose a problem in this regard, given that the same web identifier may describe any number of items over a given period of time without semantic differentiation. That being said, assuming fixed objects such as publications and datasets should remained constant for a given resource identifier, versioning and provenance could apply for a certain subsection of potential objects. For this reason, a -0.5 was given.

4.5.6 Security

Security is not an explicit concern of the model, and as such, becomes a product of the implementation and specific use case of the adopter. It thus received a 0 in this analysis until specific clients or servers can be considered.

4.5.7 Distribution

The open standard has been defined to facilitate the distribution and interoperability of the web annotations created. This fits well within a distributed structure, and thus warranted a 1.0.

4.5.8 Adaptability/Extensibility

This criteria is specific to tangible implementation, and is thus given a 0.

4.5.9 Cost

There is no cost to the model itself, but as it is only a model, a solution must be implemented surrounding the use of the model for individual adopters. It thus

received a -1.0, assuming a reasonable amount of work would be required to implement both a server and cross-platform clients.

4.5.10 Risk

The risks are self-evident in the previous criterion. No solid implementation leads to significant development cost and risk. A -1.0 was thus awarded.

As a model, the Open Annotation Data Model offers meaningful insight and promise into the problem of shared annotation on the web. That being said, as can be seen above, without a tangible implementation, it was not a viable option for consideration in the current analysis.

The results of the Open Annotation Data Model are summarized in table 4.5.

4.6 Integrated Rule-Oriented Data System

Contrary to the previous two remote solutions, iRODS offers a large scale data management and virtualization platform with extended file metadata capabilities and a plugin-based system for file event management.

4.6.1 Key/Value Definitions

iRODS has built-in support for metadata key/value pairs. These pairs can describe data objects, resources, collections, or even users (*iRODS technical overview*, 2014). For this reason, iRODS received a 1.0 on the metadata definition requirement.

4.6.2 Collaborative Nature

iRODS is a centralized collaboration solution. Like other solutions presented, the value of the metadata and features are tied strictly to their existence within the

Table 4.5. Weighted Sum Model for Open Annotation Data Model

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	1.0	20%
Collaborative Nature	20%	0.5	10%
Platform Support	10%	1.0	10%
Adoption	10%	-1.0	-10%
Versioning and Provenance	10%	-0.5	-5%
Security	10%	0.0	0%
Distribution	5%	1.0	5%
Adaptability/Extensibility	5%	0.0	0%
Cost	5%	-1.0	-5%
Risk	5%	-1.0	-5%
Total	100%		20%

iRODS ecosystem. But, the iRODS platform does support user-based access levels and management. For this reason, a 0.5 was awarded.

4.6.3 Platform Support

The iRODS server is distributed for Linux-based systems, and is composed of several different packages. Specialized clients are less important in the iRODS environment, as it is often used by users as any other network mounted file system would be. A -0.5 was given for dependence on Linux. A -1.0 was not given in light of the client flexibility.

4.6.4 Adoption

Unfortunately, the barriers to entry, in terms of adoption, for iRODS are high, both in terms of installation and configuration, and knowledge required to properly manage the solution. The server must also be self-hosted, as it does not have a third party service implementation that clients can utilize. Thus, a -1.0 was given.

4.6.5 Versioning and Provenance

iRODS does not explicitly offer versioning functionality. But, through its use of metadata catalog, and extensive rule engine, iRODS offers strong provenance functionality (*iRODS FAQ*, n.d.). In fact, iRODS sees this preservation function as one of its core missions. This warranted a score of 1.0.

4.6.6 Security

The iRODS system has its own internal user management functionality. Additionally, the default installation of iRODS includes an extensive number of

features and submodules, some of which have been more closely examined than others. As is the case with many other solutions, the integrity of the whole is dependent on the weakest link. That being said, iRODS is a large and expansive solution with many features and interfaces, and should be examined by the adopting organization to ensure features are compatible with the desired level of security. In spite of this ambiguity, given its relative maturity, for the sake of this discussion, iRods received a 1.0 for security.

4.6.7 Distribution

iRods is an intentionally centralized environment. It is not necessarily intended to collaborate with other iRODS instances. A -1.0 was given.

4.6.8 Adaptability/Extensibility

To a certain extent, one of the primary purposes of iRODS is to be extensible. The ability to attach rule-based plugin events to the infrastructure is a core feature of the product. This thus warranted a 1.0 in this category.

4.6.9 Cost

iRODS is distributed as an open source project, but the server must be installed and maintained by the utilizing organization, which does incur its own set of costs. And, as noted in the section above on adoption, the installation and requirements of such an extensive system are much higher than average. Specialized experience and hardware may be required. A -1.0 was thus given for this category.

4.6.10 Risk

Again, contrary to the previous two solution, iRODS posses significantly less risk as a more mature and developed solution. Risk for this solution is primarily found in maintainability and configuration. In spite of potential risks due to complexity, a 1.0 was given for its maturity and flexibility.

A summary of the results of the iRODS analysis can be seen in table 4.6.

4.7 Google Docs/Drive

Google offers multiple solutions that, particularly when used in conjunction with one other, provide a compelling remote solution to the problem identified by this research. Google Docs and Drive are compliments to one another. Google Drive offers cloud based storage for files of any type, and Google Docs enables online editing and collaboration for standard document, spreadsheet, and presentation type files.

4.7.1 Key/Value Definitions

Unfortunately, in spite of Google's excellent offerings, they are not explicitly intended to be a metadata store. To achieve metadata definitions in this context, a separate file with information must be stored and distributed to describe any items available within the Google Drive. For this reason, a -1.0 was given for the criteria of metadata definitions.

4.7.2 Collaborative Nature

Google is a user-oriented company, and as such, offers well defined and intuitive products and interfaces. Their Docs and Drive offerings are highly collaborative and work well within small teams. But, generally speaking, they do not function as well in an openly collaborative environment, where unknown people

Table 4.6. Weighted Sum Model for iRODS

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	1.0	20%
Collaborative Nature	20%	0.5	10%
Platform Support	10%	-0.5	-5%
Adoption	10%	-1.0	-10%
Versioning and Provenance	10%	1.0	10%
Security	10%	1.0	10%
Distribution	5%	-1.0	-5%
Adaptability/Extensibility	5%	1.0	5%
Cost	5%	-1.0	-5%
Risk	5%	1.0	5%
Total	100%		35%

may need to collaborate, as explicit permissions must be granted in order to edit a document.

Furthermore, having a central location for files eliminates some questions (i.e. do I have the latest version of a given document), but once downloaded (perhaps in publishing the document to a research journal), the metadata is not transportable and is completely lost from its original context. It thus received a 0.5 in the collaborative category.

4.7.3 Platform Support

The server for this solution is hosted by Google, and is thus irrelevant. The client however is cross-platform by nature of being accessible through a web browser. Other clients have been developed, are well supported, and are available for multiple operating systems. A 1.0 was thus awarded.

4.7.4 Adoption

Google products are relatively easy to adopt given their proliferation in the market. Furthermore, having a centrally hosted server and available clients leads to little deployment burden. A 1.0 was therefor appropriate for this category.

It should be noted though, that licensing restrictions may be involved when considering the use of the platform. These should be addressed and reviewed on a case by case basis.

4.7.5 Versioning and Provenance

Google offers versioning mechanisms for their Docs enabled files. But again, that versioning is lost upon extraction of the files from the hosting platform. Additionally, no provenance mechanism is available, thus warranting a -0.5.

4.7.6 Security

By necessity, Google's security is high. Mechanisms are made available within Docs and Drive to allow individuals to explicitly define who can access certain assets within the space. For these reasons, Google received a 1.0 in the area of security.

That being said, though not explicitly related to security, it is important to note that there may be reasons to avoid using Google due to security restrictions for classified or otherwise restricted data. For those who may potentially find themselves in that category, it would be important to review the Google terms of service and licensing, and potentially consult with ones legal counsel prior to use.

4.7.7 Distribution

These two Google products are by definition centrally stored and not distributed. This contributes to the lack of metadata context once files are removed from the central repository. But, if using a provided desktop client, the illusion of a distributed environment can be given. This warranted a -0.5.

4.7.8 Adaptability/Extensibility

Google offers fairly robust and exhaustive APIs to facilitate extensibility within their products. Though this does not allow the core product to be changed (namely their browser-based interface), other products can be developed or augmented using their API. A 1.0 was awarded for adaptability and extensibility.

4.7.9 Cost

These products are free to use. But, as noted above, some cost may be incurred if business grade solutions are required. For this reason, a 0.5 was given.

4.7.10 Risk

The primary risk for any third party integration is that of sustainability. Will the product remain available and at relatively the same cost for an extended period of time? Google is a mature and user-centric company, which in theory limits the exposure of risk in this regard. Risk is therefore considered low, and in turn, warranted a 1.0.

This analysis is summarized in table 4.7.

4.8 Dropbox

Dropbox is another online document storage and sharing solution. It offers a similar compliment for online editing through the use of Microsoft Online. It is thus a functionally equivalent solution to Google and will not be analyzed further. Its scores are summarized in table 4.8.

4.9 Git

Git, the final solution in the remote object category, is similar to the above solutions of Google and Dropbox, with a few key variations. Git GUIs and desktop clients do exist, but the most powerful interface is the native command line tool. Additionally, whereas Dropbox and Google offer centralized storage servers, Git can be implemented locally and functions under a decentralized model. That being said, there are public service options for the use of Git, including the most popular and well known, GitHub.

4.9.1 Key/Value Definitions

Git, like Google and Dropbox above, does not offer an explicit metadata solution. Adding a file to the repository for defining this extended metadata would be the primary solution to this problem. This was given a score of -1.0

Table 4.7. Weighted Sum Model for Google Docs/Drive

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	-1.0	-20%
Collaborative Nature	20%	0.5	10%
Platform Support	10%	1.0	10%
Adoption	10%	1.0	10%
Versioning and Provenance	10%	-0.5	-5%
Security	10%	1.0	10%
Distribution	5%	-0.5	-2.5%
Adaptability/Extensibility	5%	1.0	5%
Cost	5%	0.5	2.5%
Risk	5%	1.0	5%
Total	100%		25%

Table 4.8. Weighted Sum Model for Dropbox

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	-1.0	-20%
Collaborative Nature	20%	0.5	10%
Platform Support	10%	1.0	10%
Adoption	10%	1.0	10%
Versioning and Provenance	10%	-0.5	-5%
Security	10%	1.0	10%
Distribution	5%	-0.5	-2.5%
Adaptability/Extensibility	5%	1.0	5%
Cost	5%	0.5	2.5%
Risk	5%	1.0	5%
Total	100%		25%

4.9.2 Collaborative Nature

Git, contrary to some of its version control predecessors, is intended to be decentralized and distributed, thus providing a highly collaborative and redundant environment. So long as the repository remains intact, the metadata file definitions remain available. But, like other solutions, once removed from the repository, the files have no explicit linkages back to their associated metadata. A 0.5 was thus awarded.

4.9.3 Platform Support

Git is built and distributed for all major platforms, and many desktop based GUI applications are available for multiple platforms. Use of online services such as GitHub can also be made available with a web browser. A 1.0 was therefor given for this category.

4.9.4 Adoption

The adoption of Git is slightly more involved than that of Google and Dropbox in that it may require management of the server as well as the clients, and may involve a slightly higher level of technical competency from its end users. Even still, it is a general accepted solution in the version control domain, and likely to be familiar to many users. A 0.5 was given.

4.9.5 Versioning and Provenance

Git's primary purpose is version control. It is especially designed for versioning text-based (i.e. non-binary) documents. Provenance, on the other hand, is not explicitly accounted for within the application. This, in conjunction with its lack of explicit metadata store, warranted a 0.5.

4.9.6 Security

Security in Git is a function of how and where it is implemented. For example, if the server is implemented locally within an organization or even to a single user, the credentials for access and authentication would be managed by access to the hosting server, not directly by Git. Similarly, access to GitHub is managed by account-based privileges on the site itself. It is thus considered not applicable and received a 0 in this category.

4.9.7 Distribution

As a decentralized versioning system, Git is by definition distributed. But, these distributed entities cannot interoperate unless they share a common file ancestry. A 0.5 was therefor given.

4.9.8 Adaptability/Extensibility

Git adheres to a hook-based system for extensibility, offering developers the ability to inject code at targeted points within the Git workflow. Some online Git-based server solutions also offer API endpoints and triggers for customizing workflows. This warranted a score of 1.0.

4.9.9 Cost

Git is free and open source. GitHub as well is free for many use cases. Under some circumstances, though, fees may be charged for those requiring private repositories. A 0.5 was thus given.

4.9.10 Risk

Risk for this solution will likely depend on the choice of server location, whether internally hosted or externally outsourced. Assuming the appropriate solution for the circumstance is chosen, a 1.0 was given.

A summary of the preceding Git analysis can be found in table 4.9, and a collective summary of all items can be found in table 4.10.

4.10 Summary

This chapter captured a concrete summary of the identified competitors in the realm of collaborative metadata management. Each solution has been summarized according to the key metrics of this research, and given a corresponding weight. With no solution scoring higher than a 35% feature coverage ratio, an alternative solution was sought. Given this information, the new application proposed by this research will now be defined and subsequently ranked using the same criteria identified in the previous chapter.

Table 4.9. Weighted Sum Model for Git

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	-1.0	-20%
Collaborative Nature	20%	0.5	10%
Platform Support	10%	1.0	10%
Adoption	10%	0.5	5%
Versioning and Provenance	10%	0.5	5%
Security	10%	0.0	0%
Distribution	5%	0.5	2.5%
Adaptability/Extensibility	5%	1.0	5%
Cost	5%	0.5	2.5%
Risk	5%	1.0	5%
Total	100%		25%

Table 4.10. Weighted Sum Model Comparison

	LiFS	QFS	F2R	OADM	DOI	iRODS	Google	Dropbox	Git
Key/Value Definitions	20%	20%	20%	20%	10%	20%	-20%	-20%	-20%
Collaborative Nature	-20%	-20%	-20%	10%	-10%	10%	10%	10%	10%
Platform Support	-5%	-5%	5%	10%	5%	-5%	10%	10%	10%
Adoption	-10%	-10%	-10%	-10%	-5%	-10%	10%	10%	5%
Versioning and Provenance	10%	10%	5%	-5%	10%	10%	-5%	-5%	5%
Security	0%	0%	0%	0%	10%	10%	10%	10%	0%
Distribution	0%	0%	2.5%	5%	5%	-5%	-2.5%	-2.5%	2.5%
Adaptability/Extensibility	0%	0%	2.5%	0%	-5%	5%	5%	5%	5%
Cost	5%	5%	5%	-5%	5%	-5%	2.5%	2.5%	2.5%
Risk	-5%	-5%	-5%	-5%	5%	5%	5%	5%	5%
	-5%	-5%	5%	20%	30%	35%	25%	25%	25%

CHAPTER 5. IMPLEMENTATION

As exemplified by the previous chapter, initially many seemingly viable solutions existed to handle the problem of file metadata. Unfortunately, many of those solutions also have key deficiencies, neglecting primary portions of functionality needed to create a more holistic solution. Thus, a new application had to be developed. This chapter will focus on that application, as a solution to uniquely and strategically solve the problems identified herein. In the subsequent chapter, the prototyped solution will be evaluated against the previous solutions to show whether or not there is sufficient reason to deem the prototyped metadata repository a success.

To fulfill the requirements of this research and support the hypothesis established in the previous chapter, a metadata system was designed and developed. The principle requirements and constraints of that system are as follows. The application itself was named, and will be henceforth referred to as Fez. The Fez name encompasses both the server and any clients described below.

5.1 Server

Two principle components make up the interaction of the system, the server and the client. This development focused primarily on the server, as it is the fundamental element to a functioning system. During development of the server, features from the research question established in chapter 1 were divided into core features (the primary research question), and features that were considered extended models (the secondary research questions). The core features were deemed essential and had to be fully functioning for the system to be deemed viable, while

extended models could be more conceptual in nature and may lack complete integration for the sake of the current analysis.

5.1.1 Core Components

The prototype server itself runs within the Linux Operating System (OS). Due to its establishment in the market and extensive feature-set, the MongoDB NoSQL database serves as the JSON data store for the server. Overlaying that rests the application itself. The application language was chosen from common web-ready, high level programming language, such as Ruby, PHP, or Python. The server application exposes a RESTful web API for accepting inquests of, and submissions to, the repository. The data architecture for the server is shown in figure 5.1 and further expounded upon in the following sections.

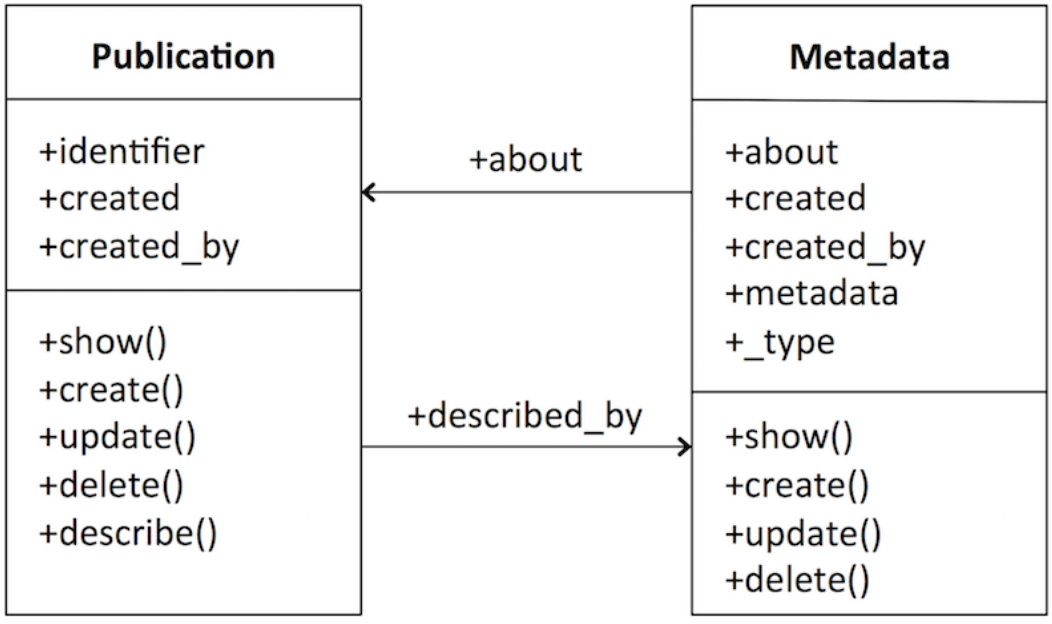

Figure 5.1. Unified Modeling Language diagram of the server.

5.1.1.1. The "Key"

Central to all of the solutions examined thus far is the need to have the files themselves, and the metadata about those files, stored in the same location. To eliminate the need to store all files in a centralized repository, a unique and deterministic key (labeled "identifier" in figure 5.1) was used to identify all published assets. This gives secondary recipients of files the ability to query the repository for metadata without having to learn the unique key from the original author or devise some universal mechanism for embedding that key in the file. The file, in essence, is the key. The key therefor travels with the file and allows for a unique separation of file contents and file metadata.

The key is composed of the file hash, suffixed with the file size, separated by a colon, in this manner:

```
1bec963c32050158e2c40f3f95ed62b55e926f918b9e2e2aa8e74ffb58d5d2e5:6546
```

The length of the hash, in combination with the file size makes this key, though deterministic, sufficiently unique.

5.1.1.2. The Foundation

The foundation of any solution is important. Choosing to construct the Fez server on the Linux architecture offered both an incredible amount of support and flexibility. It also strongly supported the expressed goal of cost effectiveness, as many distributions are available as free open source distributions.

That being said, the level at which this application is implemented allows the solution to run on other operating systems as well, including Windows and Mac. It is therefor not explicitly limited to Linux. The underlying applications required to install and run Fez include a modern version of Apache2, PHP, MySQL, MongoDB, and the PHP-MongoDB extension.

5.1.1.3. The REST API

A REST API offers a platform independent and semantic interface for any number of clients to communicate with the core metadata service. A proper REST API takes advantage of the central request methods of HTTP, namely GET, POST, PUT, and DELETE. This also facilitates the creation of clients across device types, including native phone apps, web browsers, and integrated desktops clients.

The use of REST-based APIs has gained significant momentum and acceptance, which further establishes it as the proper method for client-server interaction, as it is already familiar to most web and native app developers.

5.1.1.4. The NoSQL Data Structure

In contrast to traditional relational databases, arbitrary metadata fits more aptly into the emerging paradigm of NoSQL databases. It offers incredible flexibility for arbitrary data in the form of key/value definitions by the very nature of its underlying JSON data structure. MongoDB was selected for its strong feature set, community support and adoption, and maturity in this space. Additionally, as will be seen later, MongoDB's sharding capabilities are also advantageous for use in distributed applications.

5.1.1.5. The Language

The language chosen for the metadata server was that of PHP. This was due to it familiarity to the author and its prevalence in the web development community. According to *Usage statistics and market share of server-side programming languages for websites* (2015), PHP powers over 80% of websites whose server-side language is known.

5.1.1.6. The Installation

Having developed the server in a language such as PHP also gave the flexibility to have it function in both a standalone environment as well as alongside other existing frameworks. The basic server was constructed using the Laravel PHP framework. It is a well-adopted community driven PHP framework for developing powerful applications.

In addition to the Laravel Framework, the metadata server also integrates into the HUBzero Platform. HUBzero is a platform for scientific collaboration and community building and offers a natural and targeted integration point for working with predefined domain communities of people currently collaborated in the ways described by the problem statement. The HUBzero platform can also serve as a Fez client through the use of its collaborative projects technology.

Choosing to use strong existing frameworks such as Laravel and HUBzero also afforded the benefits of a surrounding architecture for authentication, as will be identified in the extended models below.

5.1.1.7. The Distribution

To disseminate an application to the community, it is important and beneficial to take advantage of existing solutions and mechanisms that are familiar to the community. Because of this, the source for the clients and the server will be distributed via the author's GitHub page. This is the de facto standard in the open source community for software exposure and distribution. It has built-in features for sharing, versioning, issue management, and community participation.

Beyond the source being made available on GitHub, the PHP community has adopted the package management solution known as Composer, coupled with the primary package repository called Packagist. These mechanisms coordinate with GitHub to offer a centralized and simple way for users to require, install, and keep the software updated.

5.1.2 Extended Models

In addition to the core server application, additional structures have been outlined by the secondary research questions of chapter 1. Though these items are not required to be fully functioning, paradigms were established within Fez for these extended features that should be compelling to the larger use case of Fez. As identified by the secondary research questions, these items were that of provenance tracking and versioning, security, and hierarchical distribution.

5.1.2.1. Versioning and Provenance

Provenance and versioning serve a complimentary purpose, and can be achieved by a relatively simple mechanism. The standard for a file link within the metadata can be established. For example, the child_of key (or similar), as shown below, denotes that this file is a subsequent version of another file known to Fez.

```
{
child_of :
1bec963c32050158e2c40f3f95ed62b55e926f918b9e2e2aa8e74ffb58d5d2e5:6546
}
```

Servers and clients alike could immediately recognize this as a file key and understand that this is a relationship between files. And though seemingly complex, clients interfaces can effectively obscure these details from the user, offering click-based or drag-and-drop mechanisms for establishing or identifying these relationships.

This does, however, raise the somewhat methodological question of rigidity versus flexibility. The goal of this particular project was not to develop a highly curated repository. Other systems exist that support specific metadata standards or ontologies. As previously discussed, they have the potential to suffer on account of their rigidity and the lack of adaptability in rapidly changing domains. But, that does mean that the data in Fez will ultimately only be as good as the user intends it

to be. Therefore, though the underlying structure of the application is as open as possible, clients may guide users toward certain paradigms, such as the child_of key identified above, while still offering flexibility for more advanced users.

5.1.2.2. Security

Discussions of security often attempt to understand two things, namely, authentication and authorization. Authentication seeks to know who the user is, while authorization is tasked with understanding whether or not the user can perform a given task. The goal of Fez is to provide a relatively open and unrestrictive platform for metadata definition and distribution. Taking advantage of the integration with the HUBzero framework, Fez can utilize the existing authentication mechanisms provided by the framework, including user accounts, group management, and OAuth2 token-based API requests.

Though not directly related to the functionality of Fez, it is also possible to impose limitations at the network level to completely restrict or limit interactions with the system to a closed group or domain.

5.1.2.3. Distribution

Lastly, the issue of hierarchical distribution was established to address the question of metadata scope. If a user wants to only store metadata items for consumption by their own devices, can that be supported? Additionally, can a hybrid model be established whereby a user might have a local repository, and then defer to a higher repository for locally unknown entities? To achieve such a structure, the concepts introduced by the Domain Name System (DNS) were examined to serve as a model for such a dynamic interaction.

Though not implemented, the ultimate goal of Fez is that it would be distributed in nature, allowing authoritative sources to preside over defined domains in a customizable fashion. This would follow a similar structure to the Domain

Name Service and be implemented via the MongoDB sharding mechanisms already available within the server application.

5.2 Client

With a clear understanding of the features of the Fez server, the client can now be considered to create a fuller picture of how the user will interact with the system. The beauty of building a client-server architecture atop a REST HTTP API is that any instantiation capable of implementing HTTP web requests can serve as a client to the system. This allows for incredible flexibility and customization. As it stands, native desktop clients would likely be the primary points of interaction with the system, but mobile apps and browser-based inquiries are also entirely feasible.

For the purposes of this research and testing, a platform independent command line utility was created, serving as both a complete client and as a base library that other application could leverage in constructing more advanced user interfaces.

The command line client, like one of the implementations of the server, is also based on the Laravel PHP framework, and currently supports the options shown in figure 5.2. Upon using, for example, the query command, the user can retrieve any metadata associated with an entity, as shown in the example in figure 5.3.

As mentioned above, this is not intended to be the primary, or even standard client. Other more user-friendly clients must follow to garner the support needed to scale the use of Fez. For example, the HUBzero platform file storage and collaboration feature, called Projects, has been expanded to function as a Fez client as shown in figure 5.4. Such a client is a prime example of how easily Fez can be integrated into new or existing solutions requiring extended file metadata capabilities. It also showcases how a given client may chose to guide the user in providing key metadata elements (such as Dublin Core), while still allowing for any number of arbitrary elements.

```
[fezclient]# fez
Fez console metadata management tool version 1.0.0

Usage:
  [options] command [arguments]

Options:
  --help           -h Display this help message
  --quiet          -q Do not output any message
  --verbose        -v|vv|vvv Increase the verbosity of messages: 1 for nor
mal output, 2 for more verbose output and 3 for debug
  --version        -V Display this application version
  --ansi              Force ANSI output
  --no-ansi           Disable ANSI output
  --no-interaction -n Do not ask any interactive question

Available commands:
  describe   Describes the given file with the given JSON metadata file
  help       Displays help for a command
  identify   Displays the unique ID for a given file|fguid
  list       Lists commands
  publish    Publishes the given file to the metadata repository
  query      Queries the metadata repository about the given file|fguid
```

Figure 5.2. Command line client.

```
[fezclient]# fez query file ~/Desktop/document.txt
{
    "publisher": "samwilson@purdue.edu",
    "metadata": [
        {
            "Contributor": "Sam Wilson",
            "Creator": "John Smith",
            "Date": "2015-11-04",
            "Description": "This is a sample document for testing purposes.",
            "Format": "text\/example",
            "Language": "en-US",
            "Publisher": "Purdue University Research Repository",
            "Subject": "Testing",
            "Title": "A Test Document",
            "Type": "Text",
            "child_of": "4159e6d9a1aa52749fccc981ab2c8d3917e3d3da98778b7745d799d1a1a36195:186887",
            "created_with": "TextEdit"
        }
    ]
}
```

Figure 5.3. Sample Fez query.

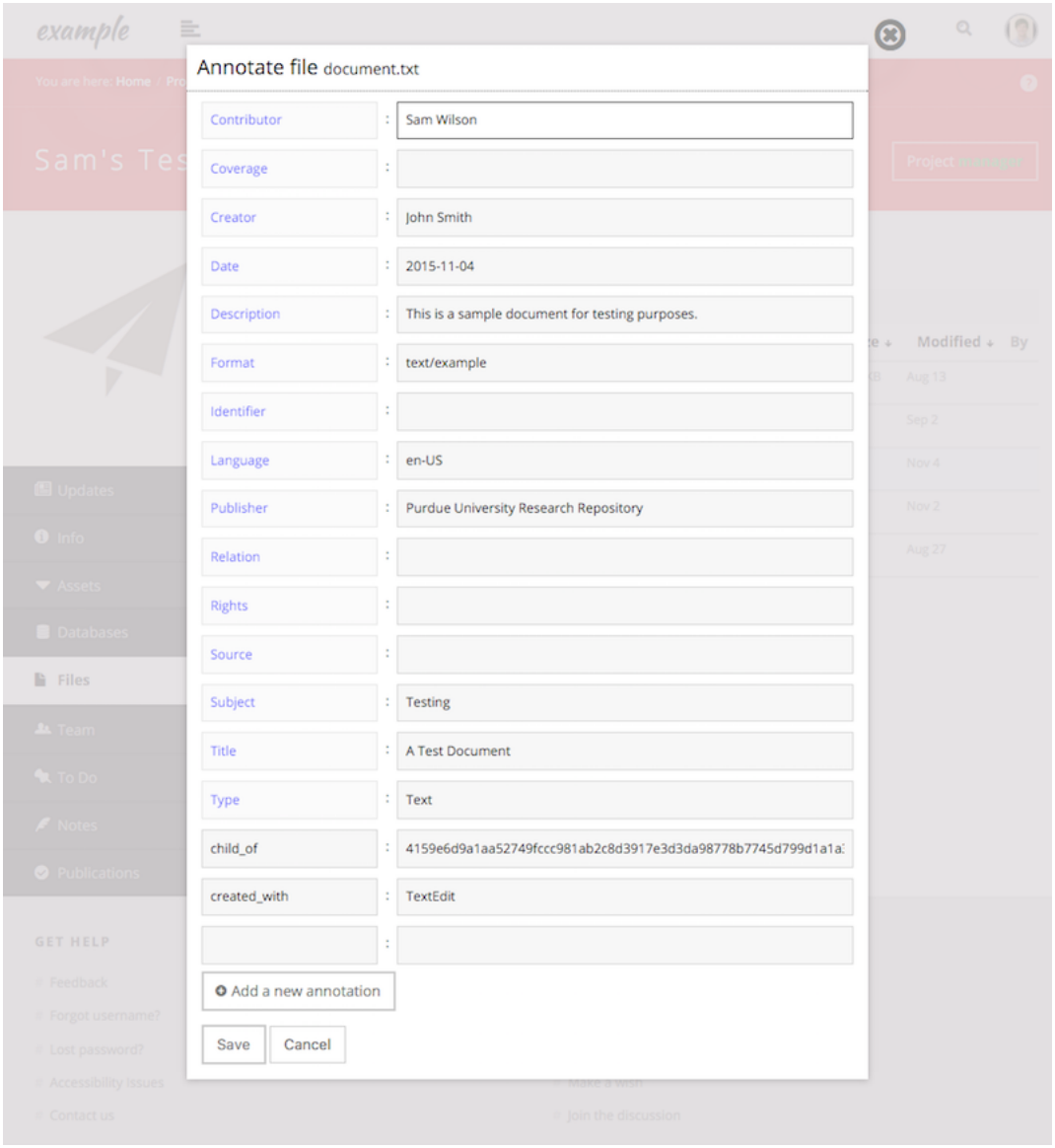

Figure 5.4. HUBzero projects Fez interface.

5.3 Summary

This chapter outlined the application developed, named Fez, in affirmation of the hypothesis, and in fulfillment of the research questions. With a functioning prototype and an understanding of its architecture and key features, it is now possible to evaluate Fez in light of the previously established criteria. This will determine whether or not Fez is sufficient to meet the needs and overturn the second hypothesis identified by this research.

CHAPTER 6. COMPARISON AND EVALUATION

Having clearly defined and identified the functionality of the developed solution, Fez, it is important to evaluate it against the same criteria as identified in the framework and measurement methodology chapter. Once complete, this grants a clear distinction as to whether or not there is sufficient cause to finalize development of the prototype solution. In addition to the justification of the build versus buy question, it may also prove beneficial for future marketing of the product, if and when it is released.

6.1 Criteria

The criteria identified in chapter 3 give a clear picture of the features and constraints necessary to sufficiently differentiate this research's solution from the existing products. These criteria will each be addressed to evaluate Fez, and to subsequently compute the weighted value of the solution.

6.1.1 Key/Value Definitions

This is the central feature of a metadata solution, and having been crafted for this purpose, Fez excels in this category. The MongoDB JSON storage offers a sufficiently flexibility data store while still giving the needed structure for the application to function. The Laravel Object Relational Mapping (ORM) and HUBzero equivalent offer the ability to define relationships between required fields, such as internal IDs and categorization constructs. Fez received a score of 1.0 for this category.

6.1.2 Collaborative Nature

This is the second key requirement of the application. And again, Fez was designed with the creation and sharing of extended metadata at its forefront. To do this, Fez has taken the approach of explicitly tying the metadata to a specific version of a file through the use of the computed file hash key. This allows the metadata to remain associated irrelevant of operating system or location. This also allows the metadata to seemingly travel with the file and to be accessible by anyone interesting in using a Fez client, irrespective of how the file was obtained. This is a significant improvement over other existing solutions, and thus warranted a 1.0.

6.1.3 Platform Support

Though the server was developed initially for a Linux distribution, such as the extremely popular Ubuntu, the use of PHP and other applicable applications can be supported on almost all modern systems, including Mac and Windows servers.

Clients themselves would ideally not be developed using a platform independent language such as Java, due to its security vulnerabilities and waining support. But, because of the implementation of a RESTful API, clients can be developed for any modern platform and language combination desired. The flexibility of both the server and the clients facilitated a 1.0 score for platform support.

6.1.4 Adoption

Ease of adoption will likely depend on whether or not those interested in using the service have any existing physical or virtual server capacity available for their use. Either way, the server and client, as mentioned in the previous chapter, are available via standard mechanisms. Concerning clients, that would ideally be the App Store for Mac users, or similar mechanisms for other platforms. The server will be distributed via Packagist, which though not likely familiar to all readers, is

very well known in the PHP community. This thus warranted a lesser score of 0.5, given the potential requirements for server acquisition.

6.1.5 Versioning and Provenance

In the Fez model, the concept of versioning and provenance is not an explicit construct. As described in the previous chapter, the client will promote the use of an association between files by way of the unique Fez key. It could also facilitate the logging of associated applications for provenantial reasons. That being said, it is a function of the user or the client, not the server itself, and thus received a score of 0.5.

6.1.6 Security

Security is not a central issue when considering the need for a relatively open system. But, for implementations limited to a certain organization, security may be implemented using network layer constructs, similar to the way that Git functions.

Additionally, the use of Laravel and HUBzero as frameworks does allow for easy integration with their existing authentication and OAuth structures. Using HUBzero as an OAuth server, for instance, would allow clients to request access to the server and facilitate authentication through users' existing usernames and passwords. This thus gave a score of 1.0 for function and flexibility.

6.1.7 Distribution

Given the proposed distributed model of hierarchical Fez repositories, the distribution score for Fez was a 1.0. This structure allows for smaller communities to define an authoritative source, while still deferring to other parent or peer Fez repositories for unknown metadata queries using the MongoDB sharding functionality and other application level protocols.

6.1.8 Adaptability/Extensibility

Often times, in a build versus buy decision, the ability to adopt an existing solution hinges on the adopter's capability to tweak and customize the existing solution to their needs. This is again the value in choosing a common language such as PHP and making the source open and available on GitHub. In-house development teams considering use of the Fez server can easily evaluate the source and consider whether it is reasonable for them to customize. They can also, using GitHub, potentially re-contribute any improvements they make back to the community for use by other interested parties. This gave the application a 1.0 on this criterion.

6.1.9 Cost

The intent of Fez, as both a server and client, is that it would be distributed as free, open source software. Thus, the only costs associated with the application are personnel costs needed to facilitate hosting and maintenance, or provisioning hardware. This will likely be relative to the size and scope of the implementation. But, given that the server is not an existing solution hosted by a third party (as is the case with solutions such as GitHub or Google Docs), Fez was awarded a score of 0.5 for cost.

6.1.10 Risk

Given the obvious immaturity of a brand new product, risk is high for early adopters. The success of solutions that hinge on widespread user adoption rely on economies of scale for added value to its users. That being said, risk is slightly lessened by the fact that this research is not proposing a completely new file type that is dependent on a Fez service to function. In other words, even if the metadata solution is disbanded, the files themselves are still intact and function normally. Even still, Fez received a -0.5 in terms of risk.

6.2 Analysis

These conclusions can be seen summarized in table 6.1.

6.2.1 Benefits

Fez excels in the key areas of metadata definition and collaboration, as well as flexibility and adoption. It also offers an open arena for a broad spectrum of client applications, including native OS apps and other frameworks or web services.

Fez offers a strong implicit tie between the extended file metadata and the file itself, even without the client being installed. The metadata definition does not require any additional standards to be implemented or operating systems to be changed. Furthermore, it does not inhibit anyone from using the files, even if they elect not to use Fez itself for extracting the augmented metadata.

6.2.2 Shortcomings

As with all solutions to a given problem, the shortcomings must be considered. Because Fez is implemented at the application level, it should be assumed to be less efficient than an application implemented directly at the file system level, as many of the local solutions analyzed in chapter 2 were designed to work. This is mainly due to network requirements and activity.

Additionally, some form of client is required to view the metadata. This may be a local native client, a web browser, or any other application to be developed. But, it will likely have to be installed on the user's system. That being said, even though the client is not built into the OS, many operating systems still allow the client to customize and even include features such as right click context menus for querying the extended metadata of a file.

Fez is easy to adopt, given the standardized distribution mechanisms mentioned above. But, as previously identified, applications of this nature require a certain scale of adoption within the community to be meaningful. This scale could

Table 6.1. Weighted Sum Model for Fez

Criterion	Weight	Score	Weighted
Key/Value Definitions	20%	1.0	20%
Collaborative Nature	20%	1.0	20%
Platform Support	10%	1.0	10%
Adoption	10%	0.5	5%
Versioning and Provenance	10%	0.5	5%
Security	10%	1.0	10%
Distribution	5%	1.0	5%
Adaptability/Extensibility	5%	1.0	5%
Cost	5%	0.5	2.5%
Risk	5%	-0.5	-2.5%
Total	100%		80%

be achieved over time in the larger community, or by necessity in smaller communities that may require the use of Fez to participate in their collaborative environment. Yet, at this point, adoption is not currently established or proven. Similarly, as identified in the criteria above, risk is high when adopting an immature solution.

6.3 Conclusions

As was seen in chapter 4, the existing solutions were insufficient to resolve the problems identified by this research. Therefore, the first hypothesis was overturned. An improved metadata management solution was needed to meet the needs established by this research. Fez was therefor created.

Having evaluated Fez as described above, it ultimately received a score of 80%, surpassing the desired level of 75%, and overturning the second hypothesis – namely, that an improved metadata management solution can be constructed that achieves 75% feature coverage. To highlight these results, a summary of all considered solutions, including Fez, is shown in figure 6.2.

In light of this analysis and success, there are two key differentiating marks that surface about this research and the solution that it proposes.

First, in relationship to the larger problem of metadata definition, this research takes a structural and user-centric approach. Others have opted for solving the problem theoretically, or by taking an overly restrictive, pre-defined ontological approach. While both are valid and needed, they have not led to many practical or well-rounded solutions to the problems identified herein. And though solutions such as Google Docs or Dropbox do have a user-centric design, their lack of extended metadata functionality causes them to suffer.

Secondly, the key is key. Fez does not rely on path-based or location-oriented identifiers. A user can annotate files through a browser or other client mechanism. Those annotations will immediately be available to a person who downloads, or

even has already downloaded, the file by virtue of the hashing key mechanism utilized in Fez. This is a significant improvement.

6.4 Summary

Ultimately, Fez promises to be a strong contender in the metadata market. Given the criteria established by this research, it is a significant improvement at performing metadata management in a collaborative environment. To show the significant improvements, this chapter analyzed Fez according to the criteria of chapter 3 and summarized the findings of this research. To conclude, the next chapter will highlight what remains to be done, and how this research should be taken from prototype to production.

Table 6.2. Weighted Sum Model Comparison Including Fez

	LiFS	QFS	F2R	OADM	DOI	iRODS	Google	Dropbox	Git	Fez
Key/Value Definitions	20%	20%	20%	20%	10%	20%	-20%	-20%	-20%	20%
Collaborative Nature	-20%	-20%	-20%	10%	-10%	10%	10%	10%	10%	20%
Platform Support	-5%	-5%	5%	10%	5%	-5%	10%	10%	10%	10%
Adoption	-10%	-10%	-10%	-10%	-5%	-10%	10%	10%	5%	5%
Versioning and Provenance	10%	10%	5%	-5%	10%	10%	-5%	-5%	5%	5%
Security	0%	0%	0%	0%	10%	10%	10%	10%	0%	10%
Distribution	0%	0%	2.5%	5%	5%	-5%	-2.5%	-2.5%	2.5%	5%
Adaptability/Extensibility	0%	0%	2.5%	0%	-5%	5%	5%	5%	5%	5%
Cost	5%	5%	5%	-5%	5%	-5%	2.5%	2.5%	2.5%	2.5%
Risk	-5%	-5%	-5%	-5%	5%	5%	5%	5%	5%	-2.5%
	-5%	-5%	5%	20%	30%	35%	25%	25%	25%	80%

CHAPTER 7. FUTURE RECOMMENDATIONS

Having crafted a viable solution to the problems identified in this research, it is also critical to identify adequate next steps for further growth and development of the solution. To do so, three primary categories of improvement should be considered, those of process validation, development, and user testing.

7.1 Process Validation

Given the scale and scope of this research, much of the work of identifying key functional requirements and weightings was performed by the author. To further validate this process, key groups of stakeholders should be established. Using the insight and expertise of those stakeholders, the requirements established and weighted herein should be validated.

Additionally, those stakeholders may also identify other technical requirements to impose upon Fez in order that optimal adoption may be achieved. This could include, for example, security audits to encourage governmental compliance and adoption. Ultimately, by incorporating key stakeholders, the potential for individual bias is lessened and practical applications of Fez are further expounded.

7.2 Development

Implementation has begun but, prototype and production-ready products are not the same. To move Fez from prototype to production, continued development and feature resolution is required.

In addition to the finalization of the server, to ensure initial adoption, several clients should be developed spanning multiple platforms. Though the client server architecture proposed offers the ability for any number of clients to function in this environment, several key clients should be established to seed the pool and aid initial adoption of the platform. Without these clients, buy-in may be difficult to establish with non-technical customers.

Lastly, more work needs to be done to finalize and implement the extended models identified. Of the three models identified, hierarchical distribution is the only one that remains at the conceptual level. Though not explicitly required, having this model implemented in the product would greatly enhance functionality and appeal.

7.3 Use Cases and Testing

In order to extend the value of the product to the end user, extensive use cases and testing should be established to further verify its product space and target audience. To do so, a sample community for initial testing and user evaluation should be established. The group would likely be a scientific community utilizing modeling, derivative data, and scientific workflows. Furthermore, user experience testing could be conducted to validate and verify any initial clients that are developed.

LIST OF REFERENCES

LIST OF REFERENCES

Ames, A., Bobb, N., Brandt, S., Hiatt, A., Maltzahn, C., Miller, E., ... Tuteja, D. (2005, April). Richer file system metadata using links and attributes. In *22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies, 2005. Proceedings* (pp. 49–60). doi: 10.1109/MSST.2005.28

Ames, S., Gokhale, M., & Maltzahn, C. (2009). *A metadata-rich file system* (Tech. Rep. No. LLNL-TR-409717). Washington, DC: United States Dept of Energy; Oak Ridge, Tenn.

Bandor, M. S. (2006, September). *Quantitative methods for software selection and evaluation* (Final No. CMU/SEI-2006-TN-026). Carnegie Mellon University. Retrieved from http://www.sei.cmu.edu/reports/06tn026.pdf

Bonneau, J., Herley, C., Oorschot, P. C. v., & Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (pp. 553–567). Washington, DC, USA: IEEE Computer Society. Retrieved 2015-01-05, from http://dx.doi.org/10.1109/SP.2012.44 doi: 10.1109/SP.2012.44

Cammarata, S., Kameny, I., Lender, J., & Replogle, C. (1995). *The RAND metadata management system (RMMS): a metadata storage facility to support data interoperability, reuse, and sharing* (Tech. Rep.). Santa Monica, CA: Rand.

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed. 2014 edition ed.). Berkeley, CA: Apress.

Chandrakar, R. (2006, July). Digital object identifier system: An overview. *The Electronic Library*, *24*(4), 445–452. doi: 10.1108/02640470610689151

Ciccarese, P., Ocana, M., Garcia Castro, L., Das, S., & Clark, T. (2011). An open annotation ontology for science on web 3.0. *Journal of Biomedical Semantics*, *2*(Suppl 2), S4. doi: 10.1186/2041-1480-2-S2-S4

Ciccarese, P., Soiland-Reyes, S., & Clark, T. (2013, December). Web annotation as a first-class object. *Internet Computing, IEEE*, *17*(6), 71–75. doi: 10.1109/MIC.2013.123

Cole, T. W., & Han, M.-J. (2011). The open annotation collaboration phase I: Towards a shared, interoperable data model for scholarly annotation. *Journal of the Chicago Colloquium on Digital Humanities and Computer Science*, *1*(3).

Dekeyser, S., & Watson, R. (2006). *Extending google docs to collaborate on research papers* (Tech. Rep.). Retrieved 2015-11-06, from http://www.sci.usq.edu.au/staff/dekeyser/googledocs.pdf

DOI handbook. (2013, August). Retrieved 2015-03-07, from https://www.doi.org/doi_handbook/

Dropbox for Business. (n.d.). Retrieved 2015-11-02, from https://www.dropbox.com/business/teamwork

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., ... Myers, J. (2007, December). Examining the challenges of scientific workflows. *IEEE Computer*, *40*(12), 26–34.

Godfrey, M. W., German, D. M., Davies, J., & Hindle, A. (2011). Determining the provenance of software artifacts. In *Proceedings of the 5th International Workshop on Software Clones* (pp. 65–66). Waikiki, Hawaii: ACM. doi: 10.1145/1985404.1985418

Google Docs. (n.d.). Retrieved 2015-11-02, from https://www.google.com/docs/about/

Haslhofer, B., Simon, R., Sanderson, R., & van de Sompel, H. (2011, June). The open annotation collaboration (OAC) model. In *Proceedings of the 2011 Workshop on Multimedia on the Web.* Graz, Austria: IEEE Computer Society. (arXiv: 1106.5178) doi: 10.1109/MMWeb.2011.21

He, S., Li, J., & Shen, Z. (2013, July). F2r: Publishing file systems as linked data. In *2013 10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)* (pp. 767–772). doi: 10.1109/FSKD.2013.6816297

Hedges, M., Hasan, A., & Blanke, T. (2007). Management and preservation of research data with iRODS. In *Proceedings of the ACM First Workshop on CyberInfrastructure: Information Management in eScience* (pp. 17–22). New York, NY, USA: ACM. doi: 10.1145/1317353.1317358

iRODS FAQ. (n.d.). Retrieved 2015-10-21, from http://irods.org/about/faq/

iRODS technical overview. (2014, November). Retrieved 2015-03-07, from http://irods.org/wp-content/uploads/2012/04/iRODS-Overview-November-2014.pdf

Langer, D. A. M. (2011). Build vs. buy. In *Guide to Software Development* (pp. 37–48). Springer London. Retrieved 2015-08-29, from http://link.springer.com/chapter/10.1007/978-1-4471-2300-2_3

Marshall, C., & Tang, J. C. (2012). That syncing feeling: Early user experiences with the cloud. In *Proceedings of the Designing Interactive Systems Conference* (pp. 544–553). New York, NY, USA: ACM. Retrieved 2015-11-06, from http://doi.acm.org/10.1145/2317956.2318038 doi: 10.1145/2317956.2318038

Open annotation data model. (2013, February). Retrieved 2015-03-07, from http://www.openannotation.org/spec/core/

Paskin, D. N. (2004). Digital object identifiers for scientific data. *Data Science Journal*, *4*, 12–20.

Ram, K. (2013, February). Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, *8*(1), 7. Retrieved 2015-11-06, from http://www.scfbm.org/content/8/1/7/abstract doi: 10.1186/1751-0473-8-7

Riva, C., & Laitkorpi, M. (2009). Designing web-based mobile services with REST. In E. D. Nitto & M. Ripeanu (Eds.), *Service-Oriented Computing - ICSOC 2007 Workshops* (pp. 439–450). Springer Berlin Heidelberg.

Sanderson, R., Ciccarese, P., & Van de Sompel, H. (2013, April). Designing the W3c open annotation data model. In *Proceedings of the 5th Annual ACM Web Science Conference.* Paris: ACM. (arXiv: 1304.6709) doi: 10.1145/2464464.2464474

Sanderson, R., & Van de Sompel, H. (2010, January). Making web annotations persistent over time. In *Proceedings of the 10th annual joint conference on digital libraries.* New York: ACM. doi: 10.1145/1816123.1816125

Usage statistics and market share of server-side programming languages for websites. (2015, November). Retrieved 2015-11-05, from http://w3techs.com/technologies/overview/programming_language/all

View and manage file versions. (n.d.). Retrieved 2015-11-02, from https://support.google.com/drive/answer/2409045?hl=en

Wang, J. (2007, September). Digital object identifiers and their use in libraries. *Serials Review*, *33*(3), 161–164. doi: 10.1016/j.serrev.2007.05.006