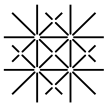

University of Basel

A Decentralized Wiki based on the Solid Ecosystem

Master Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Computer Networks Group
https://cn.dmi.unibas.ch

Examiner: Prof. Dr. Christian Tschudin
Supervisor: Claudio Marxer

Fabrizio Parrillo
fabrizio.parrillo@stud.unibas.ch
13-276-985

22 July 2020

Acknowledgments

During my master thesis, I spent many hours exploring and familiarizing with Solid. This opportunity would not have been possible without Prof. Dr. Christian Tschudin. I would specially thank you for your guidance and support during the thesis and especially for providing me with the opportunity to explore this new ecosystem and its surrounding technologies.

Special thanks also to Claudio Marxer, who supported me professionally and personally through the entire thesis. You always provided constructive feedback, which motivated me. Thank you.

I would also like to thank my family and girlfriend, Kim Buchmüller, who have always supported me during my studies. Special thanks also go to Florian Neaf, Niluka Piyasinghe, Normand Overney, and Stefan Karlin for proof-reading this master's thesis and helping me with suggestions and technical discussions.

Finally, thanks to the Solid Community and all the contributors to the Solid ecosystem, the public documentation and discussions helpt me a lot.

Abstract

Tim Berners-Lee invented the World Wide Web in 1989 at CERN. Today, almost three decades later after being released to the public in 1993, the Web has become an indispensable part of our today's society, which to some extent, influences how we communicate, think and act. During this time, companies that have shaped generations through their services, transforming the Web into a user-friendly environment and themselves into mega-corporations. Unfortunately, it is precisely their monopoly position and influence, which exposes the companies and downstream the users to the risks of centralization. The Solid ecosystem is one of many attempts to re-decentralize the Web, using Web technologies and Linked Data. It aims to restore the balance by separating the user's data from the service providers and consequently gives back the user full control over their data. This thesis explores how to implement Wiki functionalities in such a context. Since most mainstream Wikis build on a centralized architecture, we also aim to study the creation and use of 'owner-curated content' in different scenarios. In our thesis, we present our decentralized Wiki - *Solid Wiki* - incorporates a multi-synchronous collaboration environment that allows users to create, fork and merge pages. The Wiki provides versioning capabilities that let users commit and explore the evolution of pages. Moreover, Solid Wiki users can define a Wiki to be a composition of others. Finally, Solid Wiki relied on widely used technology allowing Solid Wiki pages to be presented by Web browsers, indexed by search engines and pulled by Git clients. Thus, the motivation for our thesis is to give Solid Wiki users full control over their data and the freedom to choose among Solid Apps which may be able to operate on their data.

Table of Contents

Acknowledgments **ii**

Abstract **iii**

1 Introduction **1**
1.1 Contribution . . . 2
1.2 Outline . . . 3

2 The Evolution of the World Wide Web **4**
2.1 Risks of Centralization . . . 5
2.2 Decentralization . . . 6
2.3 Current Web Developments . . . 7

3 Background and Related Work **8**
3.1 Linked Data . . . 8
3.2 Semantic Web . . . 9
3.3 Linked Data Platform . . . 9
3.4 Linked Data Notification . . . 10
3.5 Solid Ecosystem . . . 11
3.5.1 Solid Architecture . . . 11
3.5.2 Data Access Protocol . . . 11
3.5.3 Decentralized Identity and Authentication . . . 12
3.6 Wiki . . . 13
3.6.1 General Description . . . 13
3.6.2 Semantic Wikis . . . 15
3.6.3 Distributed Wikis . . . 16
3.6.4 Decentralized Wikis . . . 16
3.6.5 Federated Wikis . . . 16
3.7 Tree-Merge Algorithms . . . 17

4 Solid Wiki **18**
4.1 Functional Requirements . . . 18
4.1.1 Page Identifiers . . . 20
4.1.2 Linking . . . 20
4.1.3 Forking . . . 20
4.1.4 Merging . . . 20
4.1.5 Versioning . . . 21
4.2 Non-Functional Requirements . . . 21
4.2.1 Ownership . . . 21
4.2.2 Access Control . . . 21
4.2.3 Notifications . . . 22
4.2.4 Information Flow . . . 22

5 Scenarios **23**
5.1 Recipes . . . 23
5.2 Swiss Federal Popular Initiatives . . . 24
5.3 Vision . . . 24

6 Implementation **25**
6.1 Solid Wiki Architecture . . . 25
6.2 Domain-Driven Design . . . 26
6.3 Dependencies . . . 29
6.4 Data Model . . . 30
6.5 Procedures . . . 33
6.5.1 Create Wiki . . . 33
6.5.2 Create a Page . . . 33
6.5.3 Edit a Page . . . 33
6.5.4 Commit a Page Change . . . 33
6.5.5 Add a Page . . . 34
6.5.6 Delete a Page . . . 34
6.5.7 Subscribe and Unsubscribe to a Wiki or Page . . . 34
6.5.8 Forking . . . 34
6.5.9 Merging . . . 34
6.6 Protocols . . . 36
6.6.1 Notification of Page Changes . . . 36
6.6.2 Notification of Forking . . . 37
6.7 Solid Git . . . 38
6.7.1 Dumb Git Transfer Protocol . . . 38
6.7.2 Smart Git Transfer Protocol . . . 39
6.8 Tree-Merge Algorithm . . . 40

7 Evaluation and Results **43**
7.1 Solid Wiki . . . 43
7.2 Solid Git . . . 46
7.3 Tree-Merge Algorithm . . . 46
7.4 Recipes Wiki . . . 46

8 Discussion and Future Work **49**
8.1 Solid Wiki . . . 49
8.2 Solid Git . . . 51
8.3 Tree-Merge Algorithm . . . 52
8.4 Solid Ecosystem . . . 53

9 Conclusion **54**

Bibliography **55**

List of Figures **59**

Appendix A Turtle Files **62**
A.1 Wiki . . . 62
A.2 Config . . . 62
A.3 Empty Source List . . . 62
A.4 Page Index . . . 63
A.5 Page . . . 63
A.6 Page Title . . . 63

A.7 Git Commit Object 64

Appendix B Implementation **65**
B.1 Solid Wiki Ontology 65
B.2 Full Listing of the Implemented Domain Objects 70

Appendix C Evaluation **72**
C.1 Solid Wiki Benchmark 72
C.2 Solid Git Benchmark 74
C.3 Tree-Merge Unit Tests 76
C.4 Wiki Benchmark Test CPU-Usage 79

Declaration on Scientific Integrity **80**

Chapter One

Introduction

The digital age confronts us with an incredible amount of information every day. The Internet made it possible to share news and beliefs around the world. People can spread information instantly over social media such as Facebook and Twitter. Despite the benefit of spreading information, the Web has become filled with falsehood which is capable of swaying public opinion. The Facebook-Cambridge Analytica Scandal [1] and the interference of Russia in the US Election [2] are two prime examples which made that very clear. In Edward Bernays's early work 'Propaganda' he wrote, 'as civilization has become more complex, and as the need for invisible government has been increasingly demonstrated, the technical means have been invented and developed by which opinion may be regimented' [3]. Hence, Bernay's view on how information propagates is still valid almost a century after its publication in 1928.

Originally the term propaganda had a neutral or positive connotation, representing mainly the propagation of ideas or beliefs of a social, political and religious nature [4]. However, western democracies established new terminology in industries and academia after both World Wars demonstrated, the manipulative potential of propaganda [5, p.1][6, p.5]. Unfortunately, research thereby neglected to scrutinize the central role of propaganda in our societies [4]. For this reason and also because researchers working in the field of Communication and Media science try to bring back propaganda as a research topic [4], we refer to the natural notation of the term.

In social media, content is selected by algorithms according to a viewer's previous behaviours [7][8]. These digital footprints of users allow the service provider to place tailored propaganda [9].

Additionally, users have a hard time to curate information due to the wide variety of channels and service providers. This obstacle may originate by the lack of interoperability

between service providers and organizational tools. Furthermore, data within the providers' silos become unusable if the service provider stops a service. Users might get the chance to back up their data. Ideally, they can import the data to other software. If this alternative software does not exist, there still might be a chance that an individual or community efforts fill the gap. Otherwise, the data is unusable.

Within the scope of this thesis, we explore how Wikis could be used to tackle the described drawbacks we are facing today. We aim to provide a prototype based on Solid ecosystem started by Tim Berners Lee to (re-) decentralize the Web. Solid is one of many technologies emerged through the re-decentralize or DWeb movement [1] [2] [3] [4]. Our interest lays in exploring how to implement a decentralized Wiki based on the Solid ecosystem. The thesis also studies the production and use of 'owner-curated content' in different scenarios.

1.1 Contribution

This thesis lays a foundation for further research with Solid at the University of Basel. With our theoretical research, we explored the requirement of a decentralized Wiki. The most significant contribution is our implementation of a decentralized Wiki based on Solid - *Solid Wiki*. The prototypical implementation allows users to create federated Wikis and subscribe to Wikis or individual pages. Additionally, it incorporates versioning and a collaborative environment that let users commit, fork and merge changes within a user-friendly rich text editor. The prototypical implementation can be separated more precisely in four sub contributions:

First, we explored and applied a Domain-Driven Design (DDD) to Linked Data (LD). Second, we successfully integrated Git to the Solid ecosystem. Moreover, we were able to investigate and integrate two different Git transfer protocols. For these protocols, on one hand, we implemented a custom filesystem - *Minimal Solid Fs*. On the other hand, we implemented *Node Git HTTP Backend (NGHB)*, which enables Node.js HTTP servers to use the git-http-backend functionalities. Third, we explored how to reduce sender verification

[1] https://medium.com/fluence-network/decentralized-web-developer-report-2020-5b41a8d86789
[2] https://redecentralize.org/
[3] https://dci.mit.edu/research/the-decentralized-web
[4] https://dwebcamp.org/

for Linked Data Notification (LDN) with a pull-semantic synchronization approach. Fourth, we developed a novel Tree-Merge algorithm able to merge HTML documents in a three-way merge approach which integrates within rich-text HTML editors. Furthermore, we blended it seamlessly within the Git environment.

1.2 Outline

This thesis is structured as follows: Chapter 2 entails our motivation to investigate the development of a Wiki based on decentralized technology. Chapter 3 gives an overview of some specification upon Solid originates and builds on. Additionally, we give an overview of the evolution of Wikis that already exist and methods for digital collaboration. Chapter 4 explores and defines requirements for a decentralized Wiki. Chapter 5 constructs two particular scenarios and sketches a big picture based on the framework described in the previous chapters. Chapter 6 documents the implementation of *Solid Wiki*. Chapter 7 evaluates *Solid Wiki* and *Solid Git* by benchmarking the performance of the key procedures. Finally, we discuss our work in Chapter 8 and conclude our thesis in Chapter 9

Chapter Two

The Evolution of the World Wide Web

The initial Wew of Document (Web 1.0), invented in 1989, was a read-write environment. Back then, browsers were not only readers but also editors, which allowed users to edit the HyperText Markup Language (HTML) pages. Thereby the browsers applied the changes over the Hypertext Transfer Protocol (HTTP) [10]. This feature allowed users to edit websites directly if they had the write permission. However, this changed significantly until 2005, since most Websites were edited locally and transferred via the File Transfer Protocol (FTP) [11]. Interestingly, the Web was already a social place, where anybody had the right to use hyperlinks for linking to other hypertext documents and publish new ones.

The Web of People (Web 2.0) extended static websites with dynamic capabilities starting in 2005 [11]. Unfortunately, the initial idea that browsers were also editors did not last. Today it is not possible to edit pages directly using a browser as most websites are complex multi-layer applications these days. In general, users write content on the Web through social platforms such as Facebook, Twitter and other Web applications. Although users are still the editors of the Web, the mechanics behind the scenes have changed. Nowadays, the editing feature is handled on the application level and not directly via HTTP. Additionally, data is stored within databases and only selectively exposed via a Application Programming Interface (API). This design leads to isolation and centralization of data on these platforms. Although one can be not fond of centralization, it is not our biggest concern because it could be a natural consequence of decentralized systems. A more significant source of concern is the isolation of data. Allowing only users of an individual platform to retrieve and view its content, is entirely contrary to the idea of the Web, which should allow everybody to read and link to content they have access to.

The remainder of this Chapter first draws attention to potential risks for centralized systems. Then, we give a brief overview of decentralization before the Chapter ends with an outlook for potential future Web developments.

2.1 Risks of Centralization

According to Barbas et al. [12], centralization underlays the risk of *direct censorship*, *curatorial bias/indirect censorhip*, *abuse of curatorial power* and *exclusion*.

Services controlled by a single company are more vulnerable to direct censorship and surveillance pressure from the government than decentralized alternatives. Because in order to operate, companies must comply with local laws and regulations related to free speech and censorship, which mainly affects users in authoritarian states [13][14][15].

Indirect censorship emerges from the unintentional or intentional biases embedded in curation algorithms. These complex, obscure and ever-changing algorithms have the power to influence political decisions or the next hype by promoting or inhibiting content. Viewed from a different angle - with great power comes great responsibility. Nevertheless, corporations can exploit their curational power. Which, unsurprisingly, may primarily target content not complied with the ideological principle of the company [16][17].

Exclusion allows a platform or service provider to exclude certain users from the unprecedented opportunity to reach a global audience and engage in conversations with people from around the world. However, one should examine exclusion in the context of the local political system. In liberal democracies, exclusion can have a positive effect by targeting users that violate social or legal rules, for example. On the other hand, in an oppressed political regime exclusion mainly targets minorities which might not align with the social norms or ideology dictated by the regime [12].

In summary, all these risks of centralization can affect the filtering or amplification of content consumed by the user.

2.2 Decentralization

Decentralization is a widely used term in political science, in discussions surrounding the Internet and lately especially in blockchain technologies. Political scientists assume that decentralization takes place in hierarchical systems. The government pyramid is either federated or confederated, but in both cases, a head of state is assumed. In computer science, the manifestation of networks is far more diverse. This leads to the current state where systems are called decentralized far more often than it is theorized or consistently defined. Although much work has gone into exploring its different applications, there is no universal definition or understanding of decentralization [18].

Baran [19] [20] identified three basic topologies: centralized, decentralized and distributed, as shown in Figure 3.2. The decentralized network leads to the building of clusters and therefore increases the network's resilience to the failing of a single, central node. The distributed topology is the most resilient option since it has the highest redundancy level, which is a measure of connectivity. Interestingly, the behaviour of the decentralized network mentioned by Baran can also be observed in the short history of today's Internet.

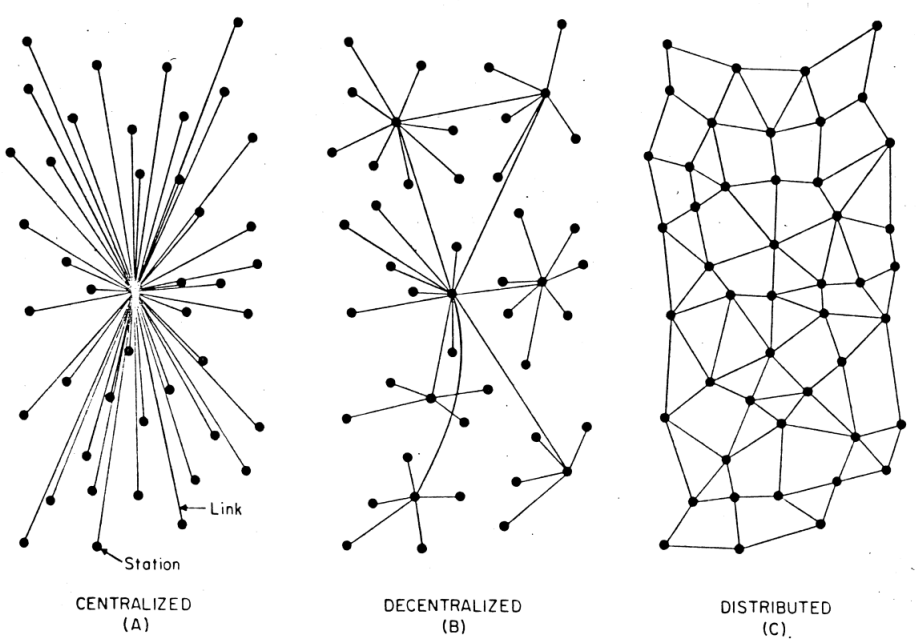

Figure 2.1. Centralized, Decentralized and Distributed Networks [20]: The study of networks can be traced back to the early sixties. Back then, the engineers had to design an available, reliable and resilient telethon network that could withstand the rigours of the Cold War.

In 1999, Tim Berners-Lee thought that the Web is so huge that there is no way any company could dominate it. Nearly two decades later, Facebook, Amazon, Apple, Microsoft, and Alphabet's Google disproved his statement by dominating the Web. Later, Tim Berners-Lee told a Vanity Fair reporter: 'For people who want to make sure the Web serves humanity, we have to concern ourselves with what people are building on top of it' [21].

Nowadays, Google and Microsoft dominated the decentralized protocol E-Mail. Even among software developers, Linus Trovards 'Git Version Control System' is primarily used through GitHub, instead of its distributed storage model [21]. Also, Peer-to-Peer (P2P) file sharing networks adopted server-like 'ultra-peers' and ' super-nodes' in order to ensure faster indexing, search [22] and to address a long tail of widespread free-riding [23]. Though the underlying protocols are decentralized, they gave birth to centralized pathogens, such as government censorship and monopolistic corporations [24].

2.3 Current Web Developments

The term Decentralized Web (DWeb) summarizes a recent movement which aims to decentralize or distribute certain centralized aspects of the current Web. The Decentralized Web Developer Report 2020 [25], published by Fluence [5] , gives a concise overview of ongoing development, technologies and the concerns of the community. In summary, the report reveals that most projects within the DWeb space emerged less than two years ago. The participants see technological infantility and P2P connectivity as the most significant technical challenges. The DWeb community, mostly driven by ideologies, expect the DWeb to strengthen data privacy, data sovereignty, and tech resistance to disruptive event or shutdowns. Among the existing internet protocols, Domain Name System (DNS) followed by HTTP worry the survey's participants the most. Among the underlying technologies, IPFS [6] and Ethereum [7] are the leaders, when it comes to developing DWeb application. Unfortunately, in general, business models are tricky to conceive for community-driven projects that do not monetize data or abuse privacy for advertising.

[5] https://fluence.network/
[6] https://ipfs.io/
[7] https://ethereum.org/

Chapter Three

Background and Related Work

This Chapter is structured as follows: First, we introduce the Solid and the leading technologies the ecosystem builds on. Then, we give a broad overview of the evolution and variations of Wikis. Finally, the Chapter ends with a brief introduction to Tree-Merge algorithms.

3.1 Linked Data

LD refers to a set of best practices for publishing and connecting structured data on the Web. Technically, LD is machine-readable data published on the Web, where its meaning is explicitly defined. Additionally, the data itself may link to external data and vice versa [26].

LD uses the Resource Description Framework (RDF) [27] to represent and model structured data. RDF is a framework for representing information on the Web. Within the framework, a graph, which is a set of subject-predicate-object describes a resource. The second data structure, RDF datasets, organize multiple resources/graphs. More precisely, a dataset comprises a default graph and zero or more named graphs [26].

RDF enables us to represent data, but does not by itself provide its schema. To model the schema, that describes the structure of the data, LD uses the Resource Description Framework Schema (RDFS) [28] and the Web Ontology Language (OWL) [29]. RDFS provides a vocabulary for basic data-modelling in RDF [28]. On the other hand, OWL provides a richer vocabulary to represent the meaning and the relationships between the terms in a vocabulary. The resulting data model that describes the representation of terms and their interrelationships is called an ontology [29].

Lately, search engines, such as Google, encourage publishers to embed additional LD withing their website. As a result, Google can interpret the data and present a rich-search-result to the user [30]

3.2 Semantic Web

The Semantic Web [31] [32] is a visionary Web presented in 2001 by Tim Berners-Lee, a Web where computers understand the meaning of a Web pages contents. Its goal to make machines understand the semantic meaning that humans interpret, is a challenging task by itself, achieving the same goal on a global scale amplifies and adds additional challenges. To achieve this ambitious goal, the Semantic Web, in addition to LD Standards, uses agents to filter and process information. Agents aim to reveal the full potential of the Semantic Web by collecting, processing and exchanging information from Web resources. The effectiveness of such software agents increases exponentially with the availability of machine-readable Web content.

Until today the described Web has not been implemented on a global scale. However, one can observe how individual industries are building on Semantic Web technologies, as an example in health care [33] [34] [35] but also in general for knowledge graphs [36] [37].

3.3 Linked Data Platform

The Linked Data Platform (LDP) specification specifies an ontology and a set of HTTP operations to enable a LD read-write architecture for Web resources. The operations allow clients to RESTfully create, read, update and delete (CRUD) resources. The basic concept of the ontology models is a kind of file system objects as LD. The model defines a file as Linked Data Platform Resource (LDPR) and a folder as a Linked Data Platform Container (LDPC).

LDPR is a resource that complies with simple patterns and conventions defined by LDP. A LDPR can either be RDF or a non-RDF resource. Generally, LDPRs are domain-specific resources that contain data from an entity in some domain.

LDPC is a container that contains LDPR. Additionally, LDPC are themselves LDPR, which allows to create hierarchical nested structures.

As an example a LDPC `../favorites/movies/`, which lists Alice's favorite movies contains the three movies (LDPR): `../movies/The_Matrix`, `../movies/Matrix_Reloaded` and `../movies/Matrix_Revolutions`.

3.4 Linked Data Notification

LDN [38] [39] is a protocol for decentralized notifications. Where the notifications are retrievable, reusable entries by their Uniform Resource Identifier (URI)s . LDN does not define any constrain regarding the content of the notification. The protocol allows senders, receivers and consumers to work together seamlessly. Moreover, this technology-independent recommendation allows implementations based on any software stack. Figure 3.1 illustrates the actors and resources involved in the LDN protocol.

Both, sender and consumer discover the receiver's inbox by requesting the target resource with an HTTP `GET` or `HEAD` request. The respond contains a RDF Statement stating the inbox location with `ldp:inbox` predicate from the LDP Ontology [8] .

A sender files a notification by a HTTP `POST` request to the receivers inbox Uniform Resource Locator (URL). The body of the request contains the application-specific notification represented as RDF.

Finally, a consumer retrieves an inbox or concrete notifications by a HTTP `GET` request on the resources URL.

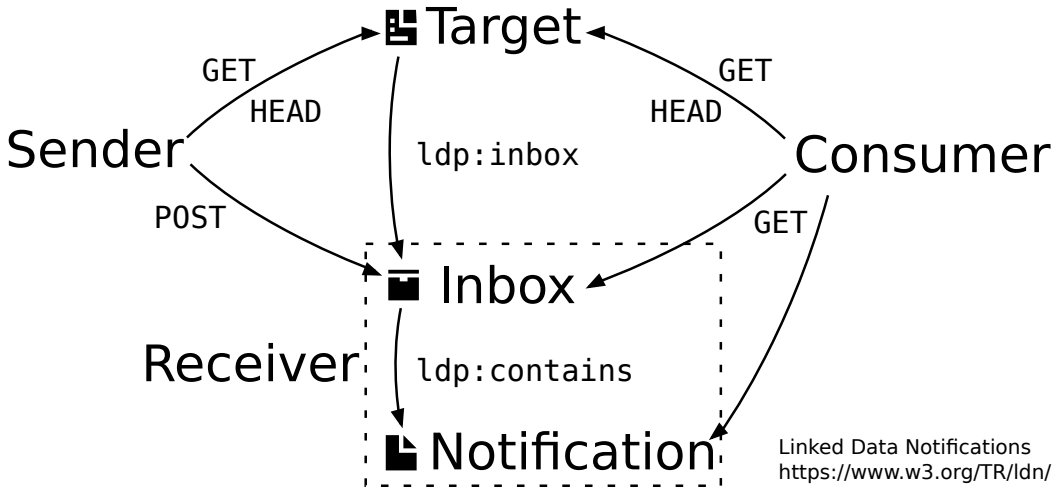

Figure 3.1. Overview of Linked Data Notification

[8] https://www.w3.org/ns/ldp

3.5 Solid Ecosystem

Solid[40] [41] is a decentralized platform for the social Web. The RDF and Semantic Web technologies build its foundation. Users store their data in a personal online data store (pod). Pods are Web-accessible storage services, either self-hosted or by a public pod provider. The used Semantic Web technologies decouple the user's data from the applications that use the data, allowing multiple applications to use the same data. Solid applications work independently of the pods' server location or service provider. Users can switch anytime between storage providers hosting the data. The platform heavily relies on existing W3C standards and protocols for reading, writing and access control of the contents of users' pods .

The following paragraphs give an overview of the Solid platform, highlight Solid's decentralized identity and authentication mechanism and presents the data access protocol in Solid.

3.5.1 Solid Architecture

The ecosystem specifies all the protocols required for application-to-pod and pod-to-pod communications. Authentication protocols are used to discover the user's identity, profile data and relevant links to the user's pod or application data. Decentralized authentication, a global ID space, and global single sign-on, is crucial for the ecosystem. WebID [42] is used to provide these features. A user registers with an identity provider, most likely being the users' pod provider. The identity provider stores the user's WebID profile document associated with a cryptographic key. Figure 3.2 shows how the described protocols and architectural components as application, client, identity provider and pods communicate [40].

3.5.2 Data Access Protocol

The Solid Data Access Protocols enable application-to-server and server-to-server communication. Currently, either RESTful methods or SPARQL Protocol and RDF Query Language (SPARQL) queries can be used to read or write data. Linked Data Platform (LDP) [43] based RESTful method, manipulate and retrieve single resources. SPARQL queries, on the other hand, perform complex data retrieval or link-following requests [40].

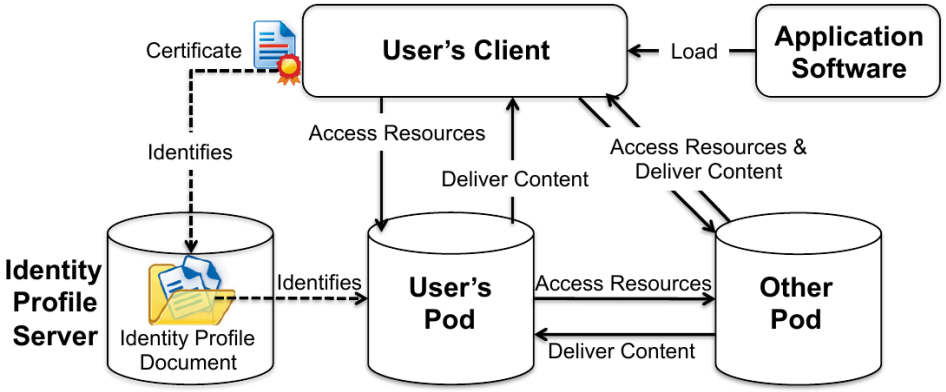

Figure 3.2. Solid Architecture [40]

3.5.3 Decentralized Identity and Authentication

Solid applications and users do not require to authenticate to the application provider. Instead, they may force a 'fake'authentication request to obtain the user's WebID from the client certificate. Once the application has access to the users WebID, the client (browser) authenticates directly with the user's pod. This practice saves users from typing their WebID during the login process.

To operate genuinely decentralized, it requires a global identity management space in which users can easily manage and extend their own identity and credentials. Solid uses WebID to implement this global identity space. Additionally, this concept of decentralized identities coupled with WebID-TLS [44] induces a Web-scale single sign-on [40].

3.6 Wiki

A Wiki is a collaborative and highly interactive communication tool that allows multiple users to view and edit web pages [45]. Ward Cunningham, the inventor of the first Wiki, describes it as 'the simplest online database that could possibly work' [46]. Therefore a Wiki can also be seen as a content management system, where any user who wants to, can edit it [47]. The acronym Wiki stands for 'what I know is' and translated to Hawaiian means fast [45][48][49]: a Wiki's core functionality is to make knowledge quickly available to the entire world. Unsurprisingly, this overlaps with the goal of Wikipedia to 'make all human knowledge available to anybody'.

This Section gives a general overview of traditional Wikis, continuing with the key design principles introduced by Cunningham. Finally, we address three Wiki properties interesting for this master's thesis, namely semantic, distribution and decentralization.

3.6.1 General Description

Wikis are not only used in non-professional communities, but also in the workspace and for education [49]. This variety of possible use-cases makes them universal text editing and content organization tools, which lead to their wide adoption over the past two decades. One of the main reasons they become so popular is because of their ease to use, also known as the 'Wiki way' , by adopting the convention over configuration software paradigm and bringing it to a broader audience. More precisely, this means that a Wiki lowers the barrier for a non-technical user to publish content as a Wiki page on the Web. Therefore, Wikis usually use an easy to use a custom markup language, which the Wiki engine converts to HTML. Thus, the more expressive the markup language becomes, the greater the complexity and consequently, the higher the entry barrier for a new user. Therefore, some Wikis abandon their custom markup language and give the users the ability to edit the content by a What You See Is What You Get (WYSIWYG) rich text editor directly. Atlassian Confluence, Google Documents or lately Slack[9] are only some examples of tools using such editors [50].

[9] https://api.slack.com/changelog/2019-09-what-they-see-is-what-you-get-and-more-and-less

Since the beginning of the first Wiki, Wikis tried to follow the design principle introduced by Cunningham, which are [51]:

- **Simple** - easier to use than abuse. A Wiki that reinvents HTML markup (*[b]bold[/b]*, for example) has lost the path!
- **Open** - Should a page be found to be incomplete or poorly organized, any reader can edit it as they see fit.
- **Incremental** - Pages can cite other pages, including pages that have not been written yet.
- **Organic** - The structure and text content of the site are open to editing and evolution.
- **Mundane** - A small number of (irregular) text conventions provide access to the most useful page markup.
- **Universal** - The mechanisms of editing and organizing are the same as those of writing, so that any writer is automatically an editor and organizer.
- **Overt** - The formatted (and printed) output will suggest the input required to reproduce it.
- **Unified** - Page names will be drawn from a flat space so that no additional context is required to interpret them.
- **Precise** - Pages will be titled with sufficient precision to avoid most name clashes, typically by forming noun phrases.
- **Tolerant** - Interpretable (even if undesirable) behaviour is preferred to error messages.
- **Observable** - Activity within the site can be watched and reviewed by any other visitor to the site.
- **Convergent** - Duplication can be discouraged or removed by finding and citing similar or related content.

These principles lead to the main non-functional property of a Wiki, timeless and therefore never finished [52]. Unfortunately, this temporal property has a negative influence on citations. Citing a Wiki is considered unscientific because retrieving a Wiki page at a later point in time does not necessarily have to match the original quoted content. Even worse, its correctness cannot be guaranteed. Fortunately, if the Wiki community consists of enough users, other users quickly correct the malicious spreading of false information. The same holds for deleted content, which can be rolled back immediately based on the page history [49][47].

A Wiki can be described as a hypertext document, i.e. a directed graph where the nodes are the pages and the set of edges are represented by links [53]. Therefore, a Wiki link connects a source page with a destination page. More precisely one could say that the logical structure of a Wiki is flat and there is no directory hierarchy. Flat in this context means that there are no links between pages. A flat structured Wiki consists only of a set of pages. The user has the absolute freedom to create relationships and impose a structure. Two advantages of this approach are: First, it empowers a uniform and simple paradigm for structuring pages. Second, it overcomes the limitation of a hierarchical directory system that requires an item to be placed in precisely one directory [54].

Most Wikis provide an indexing system for information retrieval purposes or to give structure to the underlining flat Wiki. The three most popular indexing systems are unstructured tagging, classification and thesaurus. Unstructured tagging uses uncontrolled keywords which can be assigned to terms without any rules. Classification uses tags to sorts and structures terms. Finally, a thesaurus uses a controlled vocabulary, which can be used as keywords. Two popular vocabularies for modelling a thesaurus are ISO 25964 [10] or SKOS [11]. Within the thesaurus, there exist three types of relations: equivalence, hierarchical and associative [55].

An example of an index that could be used to describe multiple terms or pages are synonyms. We could think of the index/tag/descriptor 'synonym' to connect terms such as 'cancer', 'carcinoma' and 'neoplasm' [55].

3.6.2 Semantic Wikis

Semantic Wikis try to be more expressive about the relationship between the content represented in a Wiki. There are two branches of semantic Wikis. The first branch uses RDF like triplets that are either specified in the markup or separately. The second branch uses consistent ontologies which are specified outside of the semantic Wiki and imported. Thereby ontologies play a similar role within the Semantic Wiki as a schema does for a relational database [54] .

[10] https://www.niso.org/schemas/iso25964
[11] https://www.w3.org/TR/2008/WD-skos-reference-20080829/skos.html

3.6.3 Distributed Wikis

Distributed Wikis, such as highly available Wikis, use a P2P infrastructure to increase scalability and fault tolerance. These features are accomplished by sharing storage and workload and by replicating content at different locations. These systems replicate and distribute the content behind the scenes. An end-user can, therefore, not control these mechanisms. Further distinctions are made between the underlying P2P overlay network architecture of structured and unstructured highly available Wikis [53].

3.6.4 Decentralized Wikis

Decentralized wikis aim to support a social collaboration network and adapt many ideas from Decentralized Version Control Systems (DVCS) used for software development. They promote the multi-synchronous collaboration model [56], in which multiple streams of activity proceed in parallel. The main structure of a decentralized wiki is similar to that of a distributed wiki. However, the unstructured overlay network is a social collaboration network; its edges represent relationships between users who have explicitly chosen to collaborate. Node synchronization is a manual process. Users can choose to replicate pages and manually publish their changes. Published changes are propagated along the edges of the social network. It is up to the user to decide whether to merge or discard the propagated changes [53].

3.6.5 Federated Wikis

Federated Wikis are a special case of decentralized Wikis. Ward Cunningham first coined Federated Wiki as part of his project to build the smallest federated Wiki [57] project. The main principle of federated wikis is to allow divergence. Unlike a Wiki, there are no restrictions to the number of pages for the same concept. The page title identifies each concept. Furthermore, the user has no obligation to synchronize their pages among each other. This design allows the representation of several points of views. The critical difference with decentralized Wikis is that users can search and browse the global network [53].

3.7 Tree-Merge Algorithms

Tree difference and merge algorithm try to find the differences in tree data structures and merge the differences of revisions. In 1986 Zhang and Shasha proposed an efficient algorithm [58] to compute the tree edit distance, which is the minimum number of node deletions, insertions, and replacements that are necessary to transform one tree into another [59]. Since then, a variety of Tree-Diff algorithms have been developed. Peters survey [60] and the paper [61] from Fluri et al. give a broad overview. Two inspiring approaches that purpose diff and merge algorithms are:

First, Asenov et al. proposed a novel algorithm to versioning trees. Their approach builds on a standard line-based VCS, but provides diff and merge algorithms that utilize the tree structure to provide accurate diffs, conflict detection, and merging [62].

Second, McClurg et al. proposed diffTree a new three-way merge algorithm for tree-structured data, and prove that it satisfies several properties which make it intuitive for users. Additionally, they also show how to implement real-time collaboration with seamless online and offline support by applying this merge function continuously and automatically [63].

Chapter Four

Solid Wiki

This Chapter describes the concepts for *Solid Wiki* - our decentralized one. The Chapter is structured as follows: First, the functional requirements are introduced, which represent the core features. Then, we elaborate on the non-functional requirements, which mostly depend on the deploy domain.

4.1 Functional Requirements

Solid Wiki follows the path of decentralized Wiki's as described in Section 3.6.4 and Section 3.6.5. The Wiki incorporates the multi-synchronous collaboration model [56]. Such a model allows multiple concurrent user activities. Each user works within a disconnected personal repository. Synchronization only takes place by individual user efforts. The architecture enables two types of synchronization. Users either merge changes from another repository to their repository or as an alternative, they can merge changes applied to the personal repository to a public or private repository managed by a group. Hence, no real-time synchronization takes place.

Solid Wiki allows the creation of federated Wikis. A Wiki can either be standalone or be composed of multiple source Wikis. A federated Wiki includes a subset of Wikis available on the global network. From a theoretical perspective, a Global Federated Wiki (GFW) can be defined. In practice browsing and searching pages of a GFW comes with the same obstacles for websites on the Web. Therefore we consider a federation mainly for users to organize their Wikis. Further Solid Wiki decouples pages from the Wiki itself. Pages can be part of zero or multiple Wikis.

Additionally, Solid Wiki does not rely on a Wiki specific markup language and neither on a Wiki engine. It uses HTML as a compile target, which allows the browser engine to function as a Wiki Engine. Editors must implement a bijective conversion between the desired markup to edit and HTML to persist the page. The use of HTML as compile target

has the advantages that user-specific editors can emerge. Secondly, it links the page to the current Web 2.0 if retrievable over HTTP.

Figure 4.1 exemplary show how a user could interact and organize personal and work Wikis. The following Section highlight five key requirements.

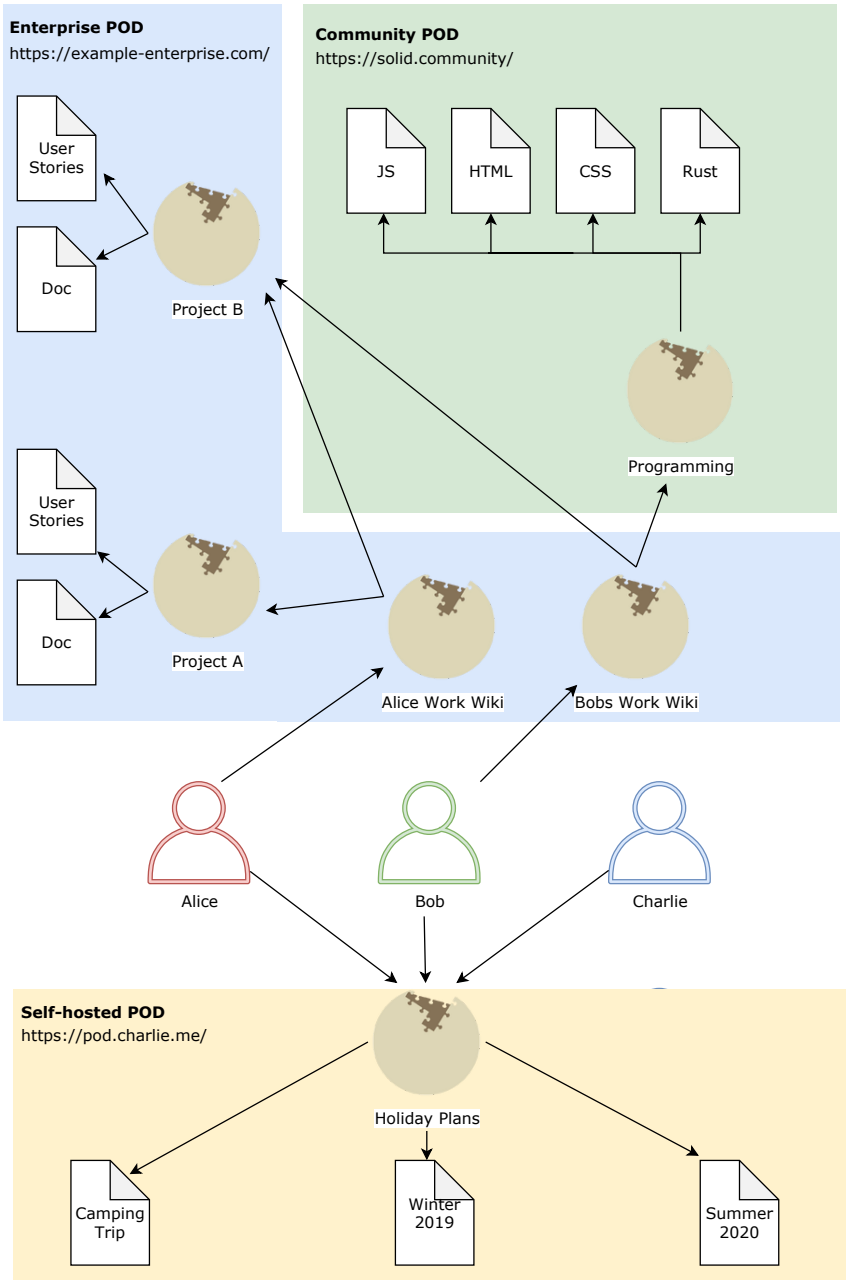

Figure 4.1. The diagram highlights the key concepts of Solid Wiki from a user perspective how users can arbitrarily compose a Wiki despite the physical location of the content. Additionally, it illustrates how the Wiki operates over three different pod providers.

4.1.1 Page Identifiers

Solid Wiki identifies pages through URI's. The URI's are derived from the page title and can be part of the URL. A page, therefore, has exactly one URI but may be located under several URL's. The page space defined by the page identifiers is flat and does not impose any hierarchy. As an example, a page with the title 'Switzerland' identifies by the URI `https://solid.wiki/Switzerland`. and occurs on multiple locations such as `https://alice.pods.org/Switzerland` and `https://bob.pods.org/Switzerland`.

4.1.2 Linking

Solid Wiki distinguishes between explicit and implicit links. The links semantic align with the page identifier (URI) and the page locator (URL) introduced in the previous Section. A page URI represents an implicit link, and a URL represents an explicit link, where an explicit link either links to a specific or the latest version of a page. A page does not need to be part of the federation. Any URL which points to *Solid Wiki Page* is valid. On the other hand, an implicit link only identifies the page but does not specify which one to choose, if the federation contains multiple pages with the same identifier. Additionally, implicit links can link to non-existing pages. Interestingly, a link to a non-existing page transforms into an applicable link as soon as a federation creates a page or a federation which already contains the page is defined as a source.

4.1.3 Forking

The forking of a page is the result of the users' desire for independent editing. The forking operation splits the page history at a particular point in time. The operation copies the entire history and all dependent resources. As a result, the forked page is an exact copy of the original page at the time of forking. Especially in a decentralized environment, with availability guarantees for resources, it is crucial to replicate complete pages and their data. Additionally, the original page's fork information must be updated to list the new fork. An important side-effect of the forking is better availability because of the page replication.

4.1.4 Merging

The fork operation introduced in the previous Section 6.5.8 promotes pages to diverge. The inverse operation, merging, combines two diverging pages to a unified merged revision.

The process does not automatically lead to convergence since both diverging parties must agree. However, it is a tool that lets users unify their changes. As discussed in Section 3.6, a Wiki page is never finished. Moreover, decentralized Wikis further stress this property since anybody can fork a page. Consequentially, the possibility for new subjective page edits increases, making merging a critical feature.

4.1.5 Versioning

A robust version system is a crucial component for collaborative software. Furthermore, in a decentralized context, where no central authority keeps track of changes, it plays a substantial role to answer what changed an by whom, as well as to explore the evolution and to checkout previous revisions.

4.2 Non-Functional Requirements

The previous Section 4.1 defined the functional requirements which should be part of a decentralized Wiki. This Section, on the other hand, describes the non-functional requirements, which may vary in relevance depending on the deployed domain or environment.

4.2.1 Ownership

Within a Solid Wiki, ownership is defined as the user's ability to control a digital resource. Anybody who can access a resource can gain its ownership by forking or copying the resource. The extreme case, a public resource, can be forked or copied by anybody. As a result, users gain the right to hold and modify resources but not to distribute them. Nevertheless, users must comply with licenses which might come with a resource, especially regarding its distribution. Ultimately, only social and legal barriers govern the participant's will to comply with the licence.

4.2.2 Access Control

Controlling who has access to which resources is particularly vital in decentralized environments and especially if deployed on a global network. Access Control is not a problem specific to Wikis, but rather a general problem. Therefore we reduce the problem to a minimum and straightforward policy which allows the Wiki to function without imposing

unnecessary workloads on managing access control. For such a simple schema, we only distinguish between private and public resources. A holder of a resource, despite public or private, always has full access. On the other hand, other users can only read the public resource but in general, not write nor update them. In some cases, if required, they are additionally granted append access. Also, the exchange of notification between users should take place with the described access rights. For this, the protocol must follow a pull semantic. The goal should be to reduce the number of resources which are actively managed by the user.

4.2.3 Notifications

Notifications inform the user about changes in resources of interest. They are the driving motor of how information propagated through the network. Notifications play an essential role in the social interaction between the participants. We need to answer two questions. What is the interest and who is interested in being notified about it?

For a Wiki, users might be interested in updates or forks of a page. The interest might be specific or implicit. A specific interest is defined by actively following a page. On the other hand, an implicit interest derives by forking a page. To what extend pages contained in the fork tree propagate their changes, should be handed over to the user. From a theoretical standpoint, every node in the tree might file a notification. However, this eventually leads to an overflow of information. Therefore the application layer must reduce the stream of notifications according to the user's preference.

4.2.4 Information Flow

User interaction causes *divergence*, *aggregation* or *convergence* of the pages content. The *Solid Wikis* nature encourages the user to fork a page. Eventually, as time passes, users modify pages. The Wiki reaches an extreme state of maximal divergence if every user holds a distinct copy of a page. At this point, if not earlier, participants may combine their views such that the variety of modification are combined or aggregate and lead to consensus. Although participant might collectively agree on some views, it is unlikely to reach network-wide consensus. However, clusters of convergence groups are more probable. In Section 5, we further highlight this process and show how real-world constrain naturally lead to convergence.

Chapter Five

Scenarios

Wiki communities vary in their degree of consensus. In this chapter, we describe two scenarios: (a) a community with non-controversial content and (b) with controversial content. The following sections introduce two scenarios, which illustrate the previously discussed characteristics. This Chapter aims to give the reader a better understanding of the requirements introduced in the previous Chapter 4. Additionally, the two concrete examples demonstrate how the domain governs the information flow. To illustrate this, we first construct a straightforward Wiki for recipes. Then, a Wiki as a working tool for unions to edit a paper for a popular initiative. Finally, we conclude with an outlook to sketch the big picture and the vision.

5.1 Recipes

This scenario describes collaboration in a non-conversational environment. Recipes are well-defined procedures, where the cooked meal meets the expectation if the chef correctly follows the instructions. Although the instruction precisely defines the procedure, chefs often derive from the recipe to add a touch. The deviation might originate due to a lack of ingredients or to be creative and explore a new variation. That natural behavior leads to divergence. Subjectively a recipe might look and feel like the best possible procedure for a dish. Moreover, it could be the only correct way to prepare the dish. But is it? Somebody grown up in Naples might always favour the real pizza - Pizza Napoletana - over any deviating recipe. But there is usually not an objective way to do something, even in such non-controversial subjects is affected by regional differences and personal ideologies. A system hosting such content must, therefore, allow divergence upon the participant's views.

In an environment that by its nature introduces divergence, is unlikely to enforce convergence that satisfies all the participants. Therefore we speculate that the path to consensus must follow a natural consequence which the participants share. Participants might agree

due to a diet or due to allergies and intolerances. Among these constraints, we think it is likely that some cluster of consensus might build up.

5.2 Swiss Federal Popular Initiatives

This scenario describes collaboration in a conversational environment. In Switzerland, citizens have the right to request an amendment to the Federal Constitution by a popular federal initiative. The initiators of an initiative must elaborate a specific draft article or a general proposal. Additionally, it is helpful if the initiators condense some convincing arguments before collecting 100,000 signatures. Once they collected the required signatures, parliament examines whether the initiative respects the principles of consistency of form, unity of subject matter and the mandatory rules of international law. If the parliament validates the initiative, it is put to a popular vote. Additionally, the Federal Council and Parliament may propose a direct or indirect counter-proposal to the initiative [64].

In this scenario, we can detect two actors, the initiators and the council, which both are composed of members from different political parties. Additionally, since any citizen can participate in the process, the initiators are even more heterogeneous and also, including other legal forms such as association and companies. Besides the initial motivation, which may differ at the beginning of the processes, they must converge and find consensus to succeed. On the other hand, voters and journalists could also profit if the actors show full transparency over their development process. Moreover, they could investigate, summarize and share their findings. Voters usually rely on the final proposal and the news article to decide. Finally, a transparent process allows an assessment of the interests and influence that each participant had during the process.

5.3 Vision

This chapter presented two concrete scenarios where we believe that our Wiki would be a great fit. However, as Figure 4.1 reveals, our vision aims to be a tool that allows the user to curate independent of the content or the physical localization. Additionally, such a flexible Wiki allows the user to curate everything at one place where the origin of the content might be public, personal or professional. In summary, the Wiki would exactly match its definition ‘What I know is’, reflecting what I know and care about at a given point in time.

Chapter Six

Implementation

This Chapter is structured as follows: First, we introduce the frontend architecture and development design. Then, we present the concrete implementation of Solid Wiki, which is written in TypeScript [12] and supposed to run in a browser environment. Much attention is spent on maintaining compatibility with current technologies. As a result, Solid Wiki Pages are indexable by current search engines but can also be cloned and pulled by Git clients. Next, we present the protocols, which synchronize notifications by a mutated LDN protocol. Furthermore, we describe two possibilities to integrate Git into Solid. Finally, the Chapter ends with the developed Tree-Merge algorithm, which allows merging HTML without conflict regarding the file format.

6.1 Solid Wiki Architecture

The *Solid Wiki* application is a Web App written in TypeScript, which operates with data persisted on Solid pods. Users control their data with apps that either make use of a subset or the entire Wiki data from one or multiple pods. Our application, which is one of many possible implementations, allows users to interact with their data. Figure 6.1 shows how the implementation is divided into five main modules, the *Domain Services Module*, the *Git Module*, the *Tree Merge Module*, and the *Background Tasks Module* which the current Chapter entails in further detail.

As already mentioned, considerable attention is brought upon maintaining compatibility with current technologies such as browsers, search engines and Git clients. This attention leads to the use of HTML to represent the body of the Wiki page. In general, Wikis use a custom markup language that the Wiki engine compiles to HTML. However, inverting that principle yields already interoperability with browsers and search engines. The architecture

[12] https://www.typescriptlang.org/

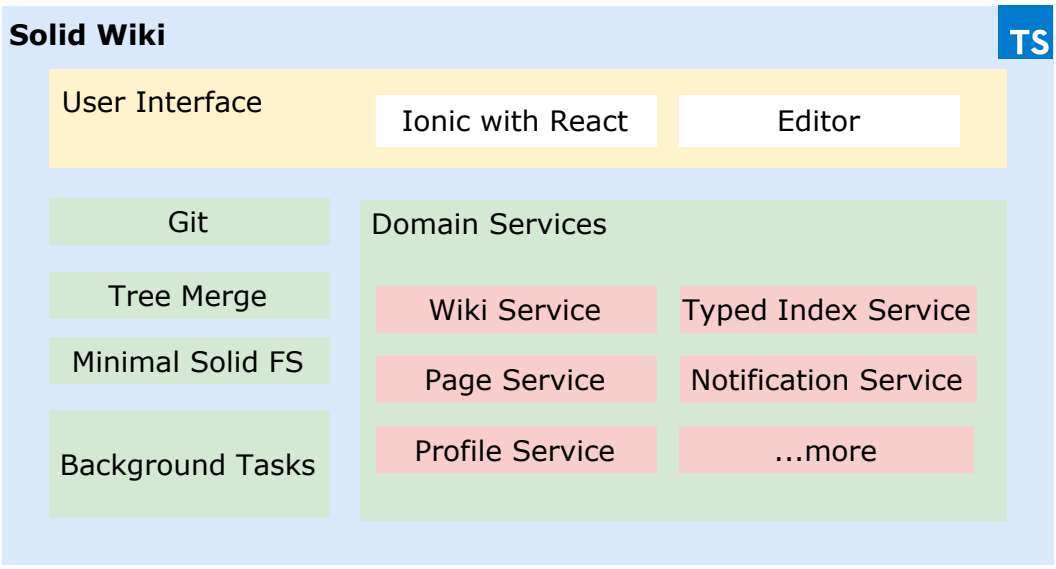

Figure 6.1. The diagram shows the Solid Wiki archtitecture in blue and the independet software modules in green and the User Interface (UI) in yellow, which we implemented and could be separeated into a dedicated library.

described until now mainly enables applications, which build on the standards of the current Web 2.0, to process the data.

Besides permeability to current technologies, we aspire to be applicable for the new Web 3.0, a Web where applications provide data in a machine-readable way. Solid apps discover and process such data by following the links encoded in the data itself. Apps, therefore, either interpreted and process the entire or only a subset of the data. As an example, a full-fledged Wiki application could be able to interpret the entire data model defined in Section 6.4. However, a notification app running as a background service on a smartphone or a smartwatch would only process the notifications.

6.2 Domain-Driven Design

In our implementation process we followed DDD [65] principles. DDD is a software design principle mostly used in enterprise applications with the goal towards clean and maintainable software. The principle defines the design rules of how to express a domain in software. We make use of three building blocks: entities, repository and services. An entity is something that is part of the domain, which we would like to model and also refer to as a domain model. A Profile, a Wiki or a Wiki Page are three example entities.

Within the context of LD a Linked Data Resource (LDR) represents an entity object identified by the RDF resources URI. A repository provides basic access over its entity. Every repository provides essential accessibility to its corresponding domain object.

A service is a stateless object that implements simple or complex application logic by using as many repositories as required. Figure 6.2 exemplarily show how we implemented the domain logic. The concrete example describes how we defined the `Profile` and the `TypeIndexRepository` domain objects. The `WikiService` uses the `ProfileRepository` to get all type indexes from a Solid profile. Once loaded, the `WikiService` uses the indexes to load all the Wikis with the `TypeIndexRepository`. As a result, the `WikiService` now abstracts the Solid App Discovery procedure. An application developer can now use the service by merely instantiating the `WikiService` and call the method with the user profile to get all Wikis (`await new WikiService().getAllWikisFormProfile(profile)`. We applied this design for all domain objects listed in Appendix B.2.

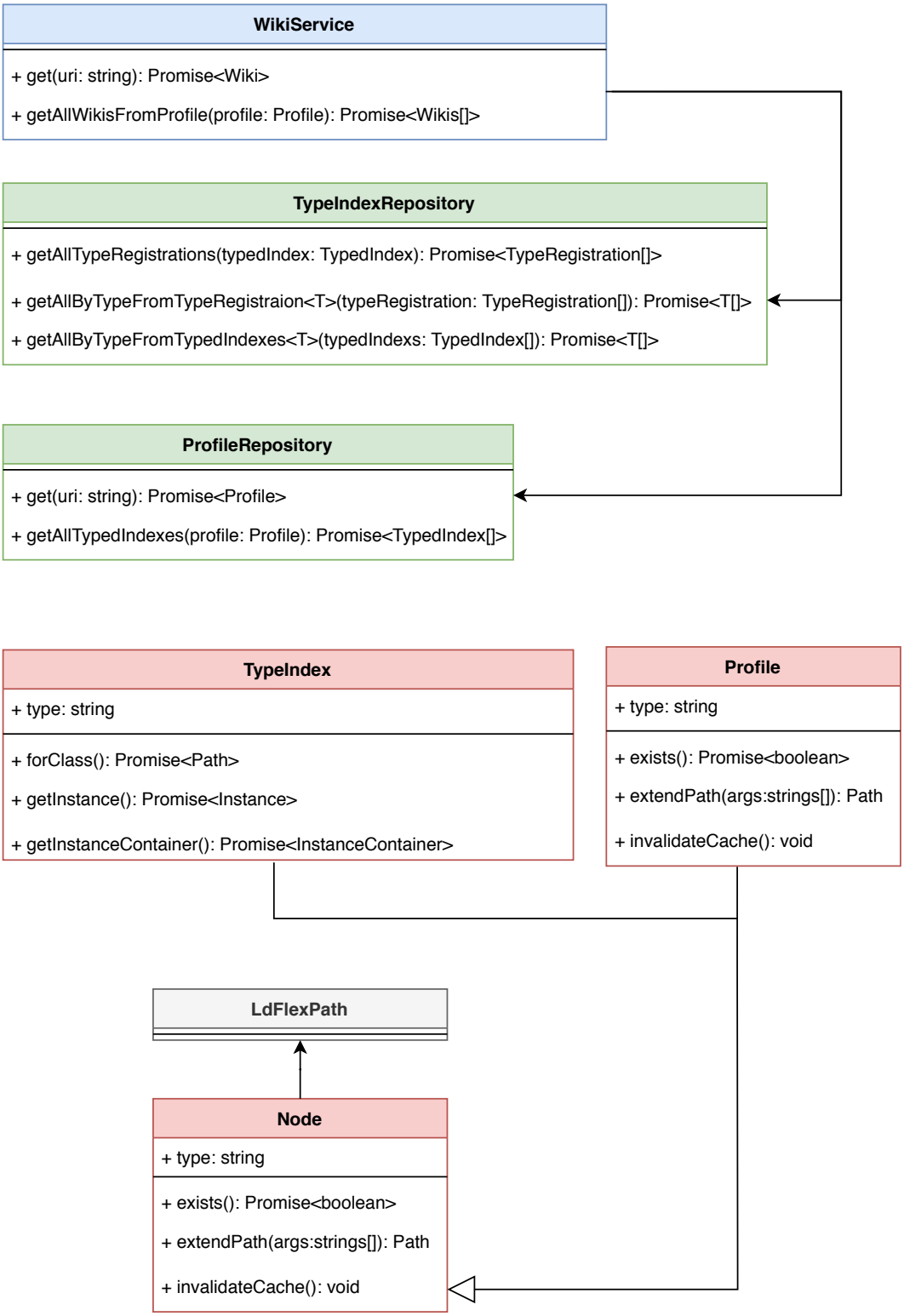

Figure 6.2. Solid App Discovery as Domain-Driven Design Example

The example shows how the Wiki service uses the Solid domain objects *type index* and *user profile* to load all Wikis from a Solid user.

6.3 Dependencies

This Section briefly introduces the libraries used to implement *Solid Wiki*. The implementation depends on six JavaScript libraries used to implement the five modules shown in Figure 6.1.

Solid Auth Client [13] a library that allows to securely log in to Solid Pods and read or write. The library is mostly used within the domain repository classes to create new entities, but also within the *Tree Merge* and *Git* modules to interact with the Solid Pod.

Solid File Client [14] is a library with a high-level API to create, read and manage files and folders. The *Minimal Solid FS* and *Git* modules use this library. Notably, the fork functionality within the *Git* module relies on the functionality to copy folders.

LdFlex [15] is a domain-specific language for query LD. *LdFlex* is used to implement the entities of the domain models where each entity class wraps a *LdFlex* object. To improve the development experience, we created TypeScript type definitions for *LdFlex* [16] and *LdFlex Comunica* [17] .

RDF Namespace [18] a library that provides short aliases to common RDF namespaces. The library is mostly used within the entity and repository classes of the domain models.

X-Tree Diff Plus [19] is a implementation of X-Tree Diff [66] an efficient change detection algorithm for tree-structured data. *X-Tree Diff Plus* is used within the *Tree Merge* module. The tree merge algorithm depends on the output computed by the diff algorithm, where *X-Tree Diff Plus* computes the difference between two Wiki Pages.

The UI uses the *Ionic Framework* [20] an open-source mobile UI toolkit for building high quality, cross-platform native and web app experiences. *Ionic* is used together with React [21] , which dictated what HTML editor is choosen. Therefore, the *Quill Editor* [22] is used since it smoothly integrates within a TypeScript React project environment.

[13] https://github.com/solid/solid-auth-client
[14] https://github.com/jeff-zucker/solid-file-client
[15] https://github.com/LDflex/LDflex
[16] https://github.com/FUUbi/DefinitelyTyped/tree/master/types/ldflex
[17] https://github.com/FUUbi/DefinitelyTyped/tree/master/types/ldflex-comunica
[18] https://gitlab.com/vincenttunru/rdf-namespaces
[19] https://github.com/yidafu/x-tree-diff
[20] https://ionicframework.com/
[21] https://reactjs.org/
[22] https://quilljs.com/

6.4 Data Model

This Section walks through the data model defined by the Wiki Ontology in Appendix B.1. Figure 6.3 and 6.4 graphically visualize the defined Wiki Ontology.

The created ontology aims to be specific, such that from a developers perspective, the data objects are expressive. Additionally, the model tries to conform to the standards recommended by the Solid specification to embrace interoperability. LDP encourages LDPC to be used over the RDF Collection. Therefore we model a collection as LDPC holding multiple resources. Nevertheless, defining every collection as LDPC leads to confusing and unclear ontologies. Therefore, we use `rdfs:subClassOf` to model a custom container to be a subclass of `ldp:container` and `rdf:subPropertyOf` to model a custom contains property to be a subproperty of `ldp:contains`.

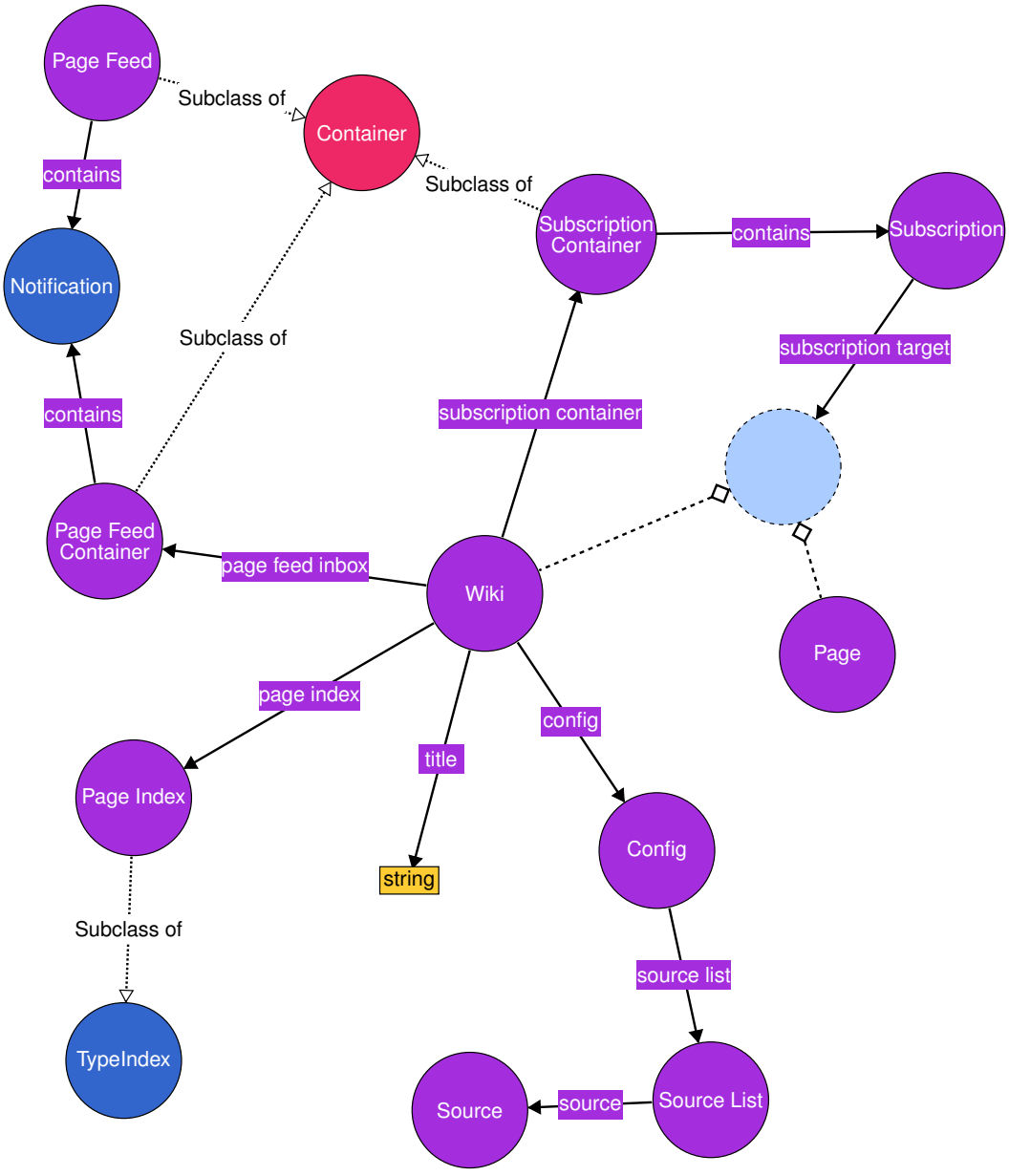

Figure 6.3. Wiki Ontology: Wiki Data Model

A Wiki consists of a Title; a Page Index which is a subclass of a Typed Index from the Solid Terms Vocabulary; a Config with a Source List that lists all source Wikis; a Page Feed Container that contains Feed Notifications from the Solid Terms Vocabulary; and a Subscription Container that contains Subscription which either holds a Page or a Wiki as subscription target. The circle colors group the classes by their vocabulary and arrows with a label represent a property. Arrows with a 'Subclass of'- label indicate that all the instances of one class are instances of another.

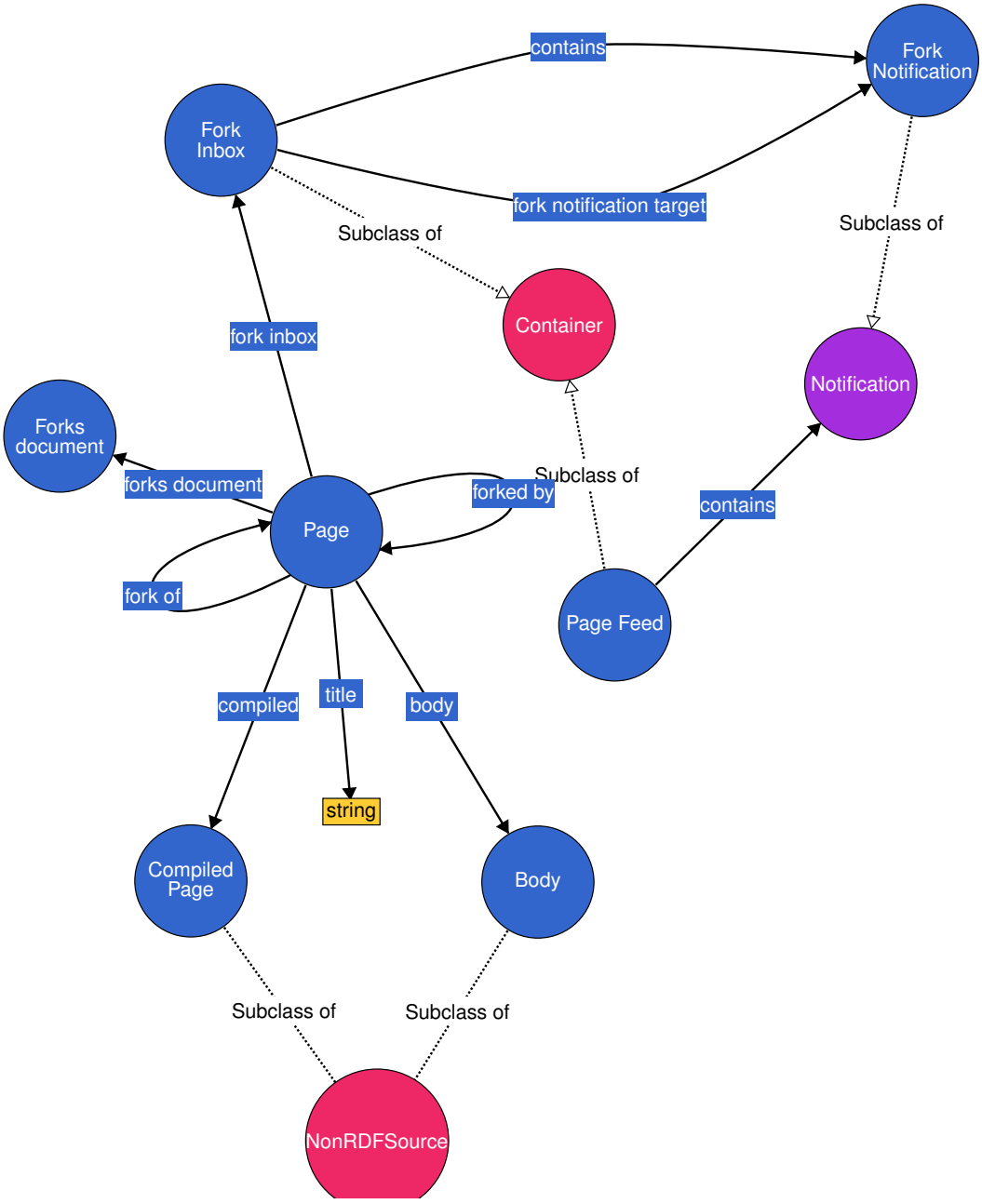

Figure 6.4. Wiki Ontology: Page Data Model

A Page consists of a Title; a Body represented as HTML; a compiled version of the Wiki Page; a Forks Document that embeds fork statements; and Fork Inbox which contains Fork Notification.

6.5 Procedures

This Section addresses the procedures defined as functional requiremens in Section 4.1.

6.5.1 Create Wiki

A Solid Wiki application creates a Wiki resource, as shown in Listing A.1. The resource is composed of a title, a config, a page index and subscriptions. The default configuration, as shown in Listing A.2, only holds the source list, which lists the transitive dependent Wikis as shown in Listing A.3. The page index, as shown in Listing A.4, as well as an LDPCfor subscriptions, are empty.

6.5.2 Create a Page

A Solid Wiki application creates a page resource, as shown in Listing A.5. A new page contains a title, an empty body file, an empty forks container, a fork inbox and a commit. The title is defined in a dedicated resource, as shown in Listing A.6. The forks LDPC, as well as the fork inbox, are empty. Section 6.5.8 explains the fork properties role in more detail. Finally, the initial commit, as shown in Listing A.7, thereby tracks the title and the body.

6.5.3 Edit a Page

Editing the Wiki page mutates the files in the working directory. Changes are either applied to the title or the body. The working directory, therefore, keeps track of the work in progress. A visitor would still read the unchanged index.html or the body defined by commit head. Changes could be immediately visible if the Wiki implementation presents the title and body in the working directory as default.

6.5.4 Commit a Page Change

The commit procedure stages and commits the mutated and tracked files (tile and body). Once persisted on the pod, the procedure creates a page change notification into the page feed inbox.

6.5.5 Add a Page

Adding a page from another Wiki only embeds the page reference in the Wikis page index. A Wiki application dereferences the page on runtime.

6.5.6 Delete a Page

Within a Solid Wiki, we consider soft and hard deletion. A soft deletion deletes the Page from the Wikis Page Index. A hard deletion additionally removes the page container and all page resources. After a soft deletion user can still find the Page on their pod, but Wikis should not list the Page anymore. Moreover, it is the only option if the user only has read access to a page but listed the Page in the index. On the other hand, if the user owns the Page and performs a soft deletion, a Solid Browser can be used to delete it completely.

6.5.7 Subscribe and Unsubscribe to a Wiki or Page

Subscribing to a wiki or a page creates a new subscription in the subscription container. Unsubscribing, on the other hand, results in the deletion of the subscription from the container. As the Wiki Ontology in Figure 6.3 shows, a subscription target might either be a Wiki page or a Wiki. As a result, the Wiki application periodically checks the subscribed pages and all the pages of the subscribed Wikis for new notification. The synchronization protocol thereby follows a pull semantic, which Section 6.6.1 addresses.

6.5.8 Forking

Forking includes copying all page resources. The resources location change. Consequentially, their URLs change according to the new location. Therefore, all resources must use relative paths to reference other resources. In contrast, a permanent reference, such as a profile reference, must be defined by an absolute URL. Additionally, the procedure appends a notification to the fork inbox. Section 6.6.2 describes the exact notification algorithm in depth.

6.5.9 Merging

The merge procedure merges two wiki pages. The pages must originate by a joint base revision. The procedure first copies all deviating commits by branching the version history. Subsequently, a merge algorithm merges the two branches head revisions. We

entirely decoupled the merge algorithm from the procedure, such that new algorithms may emerge. In Section 6.8 we will introduce our merge algorithm.

6.6 Protocols

The protocol introduced in this Section synchronizes notification among Wikis. In both cases, we could have used LDN, but we wanted to reduce the need for sender verification to a minimum. Therefore, we changed LDN to follow a pull semantic. Additionally, the design is inspired by the publish-subscribe pattern and Really Simple Syndication (RSS). A notification is consequently not created on the receivers inbox but the sender's side. A receiver must first subscribe to that inbox. In our case, subscribing to a Wiki or a Page introduces the user's interest. As a result, the application periodically synchronizes the receivers with the sender's inbox. The approach works great if the user introduces the interest as it is the case for page changes. But it does not work for notifications where the receiver does not know or does not have an interest in the sender beforehand.

6.6.1 Notification of Page Changes

The *page change synchronization protocol* synchronizes page change notifications. The synchronization process starts if the following two preconditions are true: Firstly, the user must either subscribe to a page or a Wiki. Secondly, the user must open the client application, which triggers the cyclic notification pull check. If both conditions are true, the application runs the protocol, illustrated in Figure 6.5, as follows: (1) The subscribed Wiki page inbox is checked for new notification. (2) Did the sender create a new page change notification? (2a)YES? Then copy the notification from the sender to the receivers inbox. (2b) NO? Wait for the next trigger.

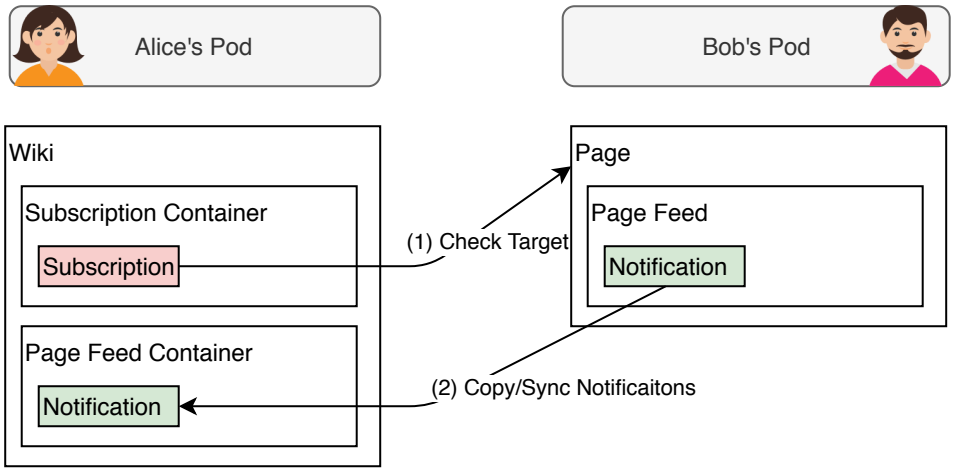

Figure 6.5. Notification of Page Changes

6.6.2 Notification of Forking

The *fork notification synchronization protocol* synchronizes fork notifications to update the fork document. The sender of the fork notification files a new notification in the receivers fork inbox. This procedure follows the LDN protocol and therefore the receiver must grant the sender with the append permission for the fork inbox. As an additional step, the receiver checks the fork and consequently updates the fork document. The entire protocol is illustrated in Figure 6.6 and runs as follows: (1) Sender: Fork a Wiki page. (2) Sender: Append a notification to the receivers inbox. (3) Receiver: Read the notification in the background task and check if the notification target matches the Page. If true, add the fork to the forks document. Otherwise, it ignores the notification.

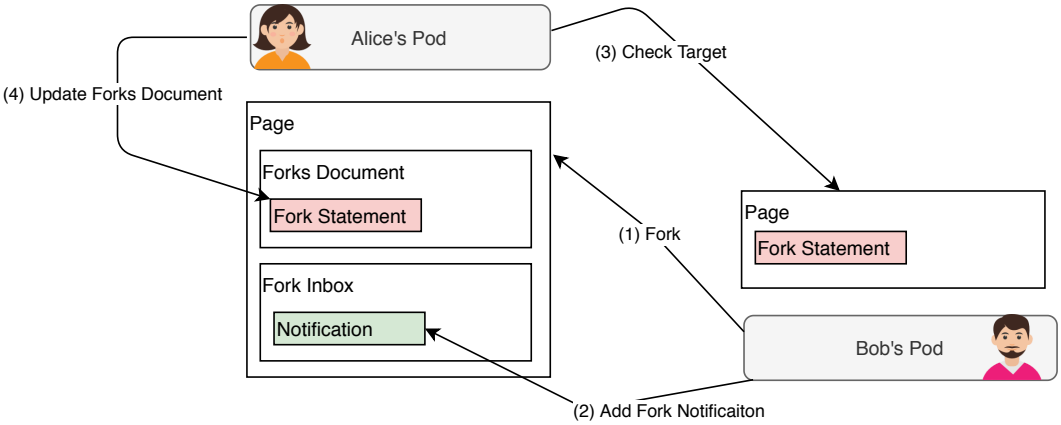

Figure 6.6. Notification of Forking

6.7 Solid Git

Solid Git enables a Solid pod to offer a Git backend interface. Git is not part of the Solid specification. Although some consideration to support Git can be found in the solid-git Github repository. This thesis explores two approaches, distinguished by their transfer protocol both protocols, the smart and dumb protocol run over HTTP.

6.7.1 Dumb Git Transfer Protocol

The Dumb Git HTTP expects a bare Git repository served by a Web Server. In general, the Git command-line interface is used to create a repository and enable the post-update hook. However, regular users will not get access to the server. Further, the main objective of this approach is to work with the current Node Solid Server (NSS). Therefore, our approach leverages full responsibility to Isomorphic Git. Unfortunately, we could not use LightningFS. Therefore, we created *Minimal Solid FS* a rudimentary file system that implements the same subset of the NodeJS file system as the LightningFS does. The *Minimal Solid FS* mimics a Network File System. Each file system call results in an HTTP request. Additionally, the Solid authentication client sets the authentification header attributes before invoking the HTTP request.

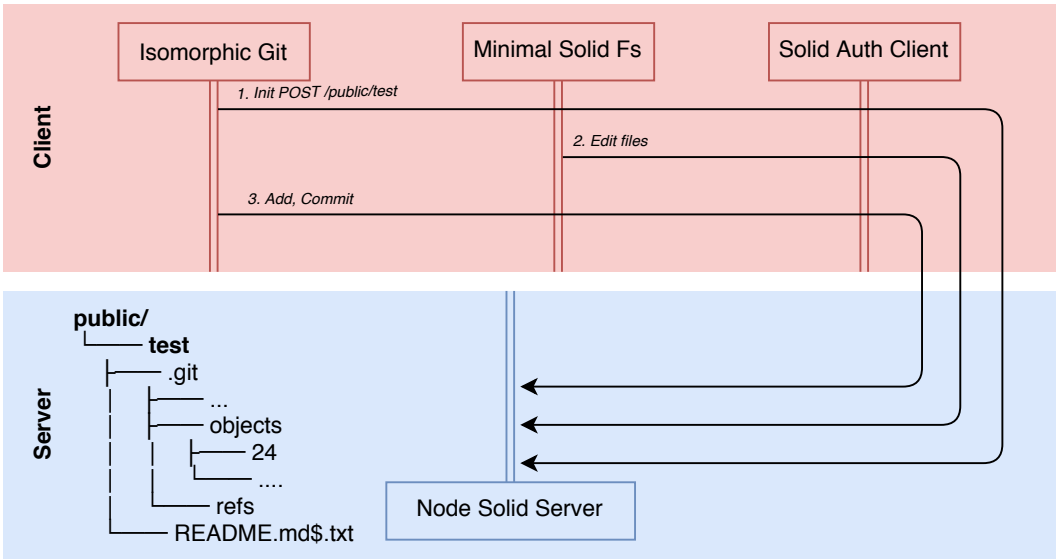

Figure 6.7. Smart Git Transfer Protocol with Minimal Solid Fs

6.7.2 Smart Git Transfer Protocol

The Smart Git HTTP implementation requires NSS to be patched. The patch includes our NGHB implementation. NGHB enables git-http-backend functionalities for Node.js HTTP servers. The servers must thereby implement a request handler that matches the request URL against git upload or receive packs. The NGHB handler then pipes the matching request through the Common Gateway Interface (CGI) to the git-http-backend. Our NGHB implementation only sets the variables required by the git-http-backend instead of setting all variables defined by RFC 3875 [23] . This extension enables the server to serve repositories with the smart transfer protocol. Besides, the server can, if desired, allow the initialization of a bare-git repository. A significant advantage of this approach is that clients can use a standard setup with Isomorphic Git and LightningFS. Only the HTTP module needs to be customized to use the Solid Authentication Client.

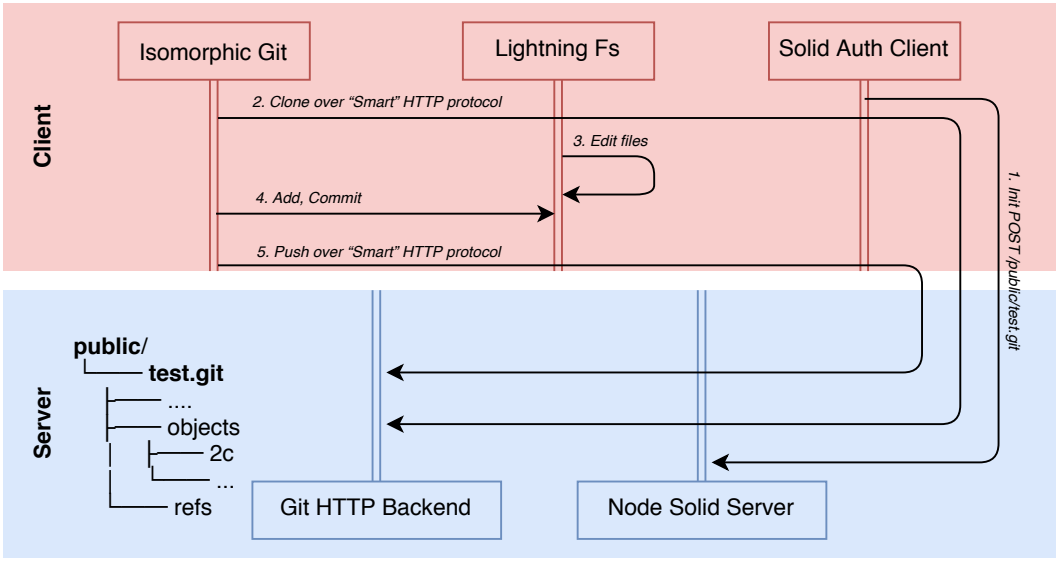

Figure 6.8. Smart Git Transfer Protocol with Git HTTP Backend

[23] https://tools.ietf.org/html/rfc3875

6.8 Tree-Merge Algorithm

A merge algorithm merges two revision. The presented merge algorithm in this Chapter is a three-way merge algorithm. A three-way merge algorithm requires a base revision which is a common origin of the two revisions. A diff algorithm thereby computes the differences between the revisions and the base. Git conventionally uses the 3diff which operates on lines. Text line granularity as an atomic component for the diff algorithm has proven to be extremely useful. Nevertheless, diffing on lines does not use information encoded by the file format. Furthermore, there are no guarantees that the merged revision conforms with the file format.

The elaborated Tree-Merge Algorithm merges two HTML pages with a common base revision. As a result, the algorithm distinguishes textual and structural changes.

As already discussed in Section 3.7 structural merge algorithm's use the information encoded through the structure. Our Wiki implements a three-way tree merge algorithm. The algorithm takes three HTML Trees (T_{base}, T_A and T_B) as an input and outputs a merged HTML Tree (T_M). The developed tree merge algorithm can be split up in seven steps:

I. Find the common base revision. II. Convert the HTML to be represented as XTree object. III. Compute the tree difference among the trees. IV. Map the tree differences to their changesets. V. Sanitize the changesets according to a policy. VI. Apply the changes within the conflict-free changesets. VII. Convert the merged XTree back to HTML.

The seven steps in detail look as following:

I. First, the algorithm must define a joint base revision T_{Base} by traversing the history. T_{Base} represents the common origin of T_A and T_B. Otherwise, if the algorithm can not find T_{Base}, it exits with an error.

II. The mapping function `mapHTMLToXTree` converts the HTML to an XTree representation. Each `HTML Element` maps to an `XTree Element Node`. `HTML Text Nodes` do not directly map to the corresponding `XTree Text Nodes`. Instead, text nodes are split by a space character, such that an XTree node represents a word of the text node.

III. The `diff` function computes the difference between T_A, T_B and the base T_{Base}. We use X-TreeDiff+ [66] and its implementation by @dovyih/x-tree-diff-plus, with some additional patches, as a diff-tree algorithm.

IV. The `asChangeSet` function maps the computed changes from the former Step III to a changeset. The changeset groups all changes according to their operation: delete, insert, update, move or no operation.

V. The `sanitizeChangeSet` function sanitizes the changeset computed in former steps by enforcing a provided policy. The sanitize policy might either be an operational or a source policy. An operational policy enforces a specific hierarchy among conflicts. Operational conflicts arise either for delete/update and delete/move operations. Therefore, the policy defines which changes should be applied. On the other hand, if a delete/move conflict arises, the operational policy can not be used. In that case, the source policy defines which of the two must be applied. The source policy can operate without the definition of an operational policy; the opposite is not applicable.

VI. The `applyChanges` function applies the changes within the conflict-free changesets as follows: Ignore unmodified nodes since they are already contained in the merge tree. Apply insertion from both changesets in depth-first order. Otherwise, it could happen that a child node would be inserted before its parent. Next, flag all deleted nodes as deleted. Then apply the modification introduced by the update operations. Then the move operations are applied, also in depth-first order. Finally, we check if the previous steps introduced any cycles; if so, they are removed.

VII. Last but not least, the `mapXTreeToHTML` converts the resulting merge tree back to HTML.

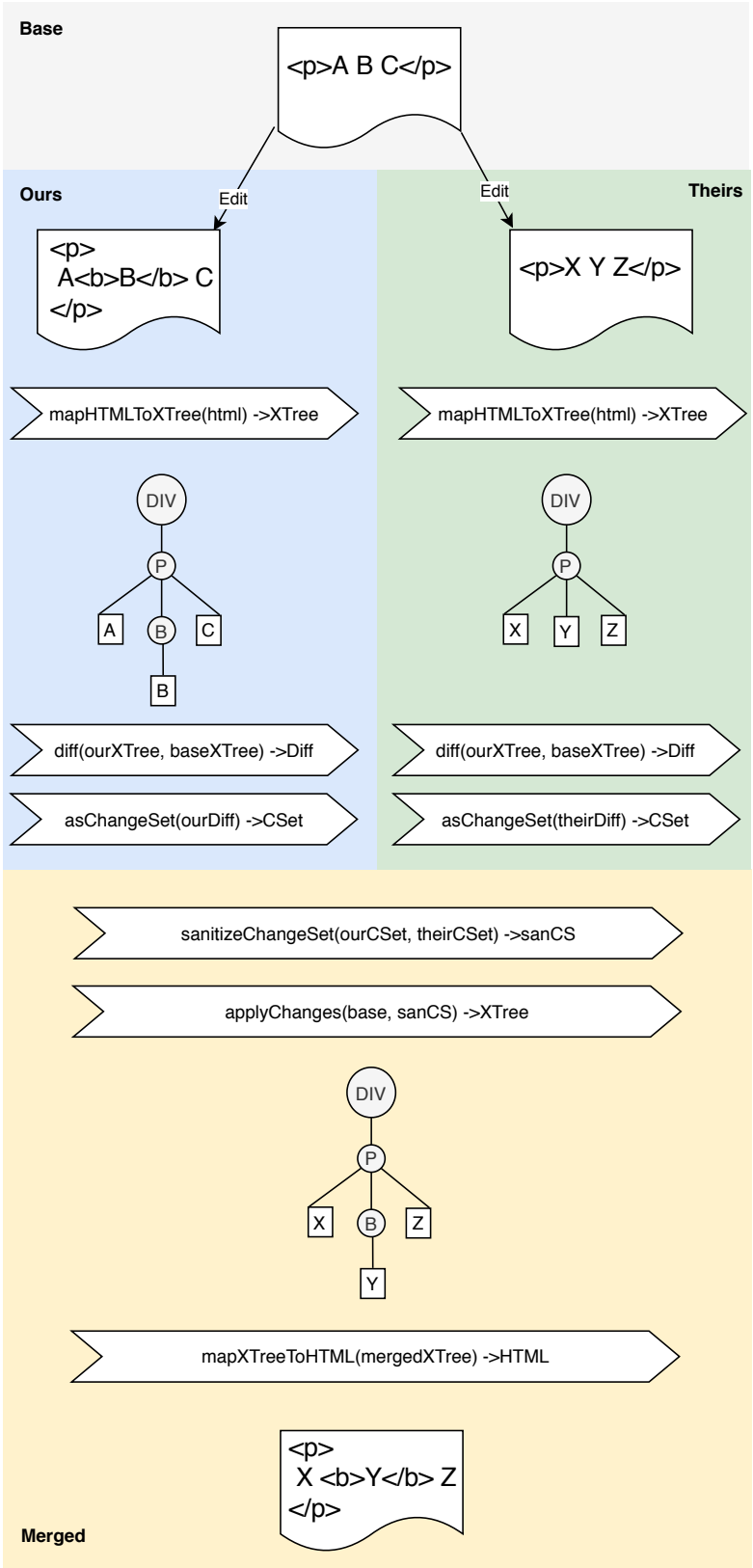

Figure 6.9. Tree-Merge Algorithm

The diagram illustrates the seven steps of the merge algorithm. The first step in finding a common base revision is omitted. The example merges to HTML documents with one paragraph. The first users induce a structural change by making the letter B bold. The second user induces a textual change by rewriting the paragraph content to 'X Y Z ' . The merge algorithm merges both changes to the resulting paragraph: X **Y** Z

Chapter Seven

Evaluation and Results

This Chapter describes the selected evaluation methods and the results. The used examination method evaluates Solid Wiki and Solid Gits performance. Further, we present the tested merge cases of the merge algorithm. To conclude, we present the use of the implemented Wiki on the Recipes Scenario described in Section 5.1.

For the evaluation our implementation we used a HP ZBook 15 G4 with an Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz and a virtual server instance with an Intel Core Processor (Broadwell, IBRS) 2095 MHz. Both server and client ran on Ubuntu 18.04.3 LTS. The server runs a Node Solid Server 5.2.4 with an Apache 2.4.29 proxy. In summary, the test revealed that the application is primarily I/O and CPU bounded. We draw this conclusion by the presented results in this Chapter and the Figures C.1 and C.2.

7.1 Solid Wiki

The performed benchmark-test, shown in Appendix C.1, examines the Wikis core procedures: creating and loading Wikis and Pages. The test starts by creating a new Wiki and a page. Then it creates a batch of pages and adds them to the Wiki. The batch size increases up to 100 Pages in steps often. Last but not least, the test reads all Wikis from the user's profile and all pages from the Wiki. The test repeats this described procedure ten times.

In Figure 7.1, we can see that creating and getting all Wikis and Pages are on average under one second. Both Figures 7.2 and 7.3 show a linear growth in elapsed time to add or read Wikis.

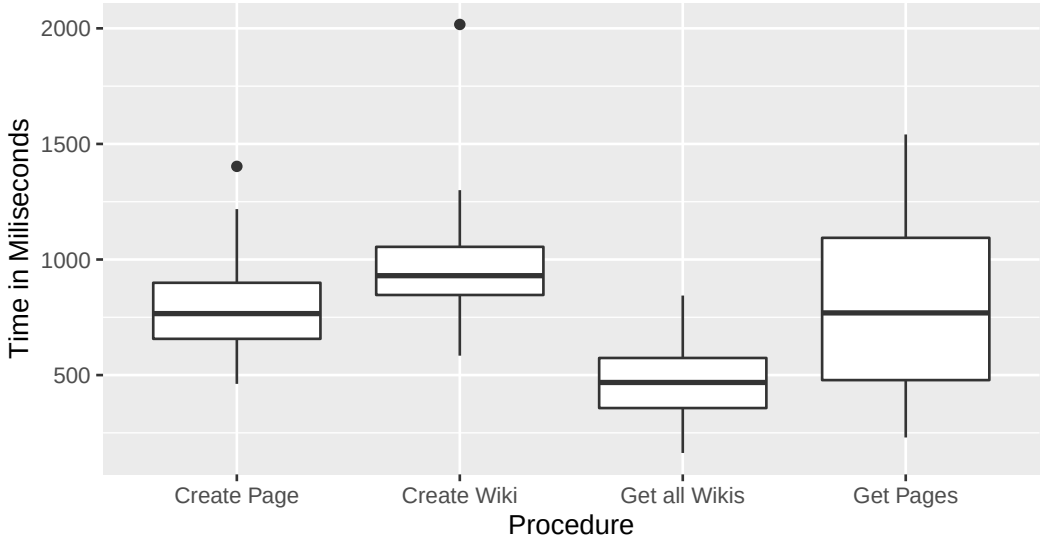

Figure 7.1. Benchmark Test: Create and Gets all Wikis and Pages.

The boxplot shows the times in milliseconds for creating and loading all Wikis and Pages where the number of Wikis and Pages increases from 1 to 100 as described. A boxplot shows the inter-quartile range (box), which represents the measure points lying between the lower and the upper quartile. The vertical line indicates the full range of the measured samples and the dots represent outliers. The horizontal line within the box represents the mean.

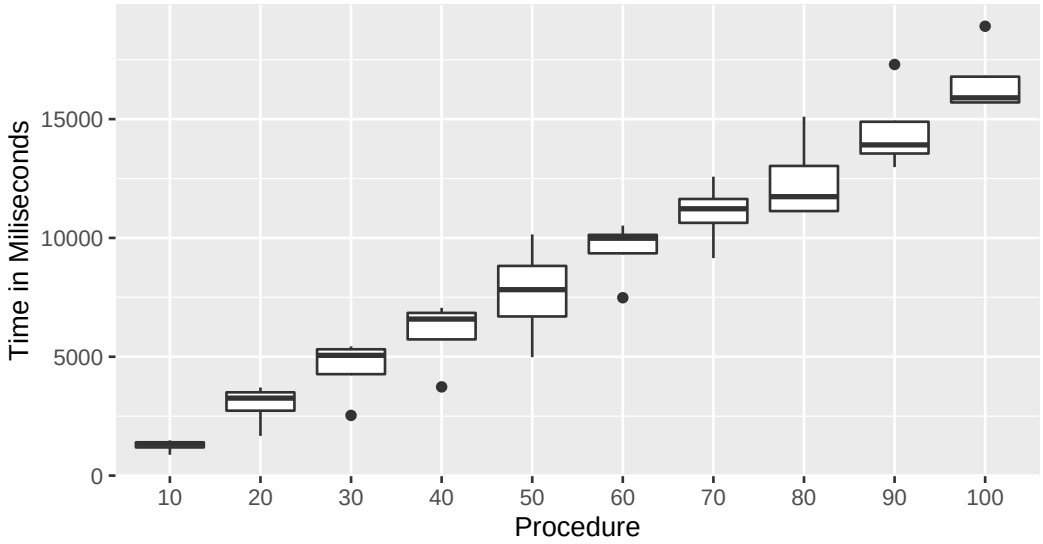

Figure 7.2. Benchmark Test: Add all Pages to a Wiki.

The boxplot shows the times in milliseconds for adding Pages to a Wiki. The time T for adding pages to the Wiki increases linearly with the number of Pages N_p and roughly follows the formula $T = N_p * 150ms$

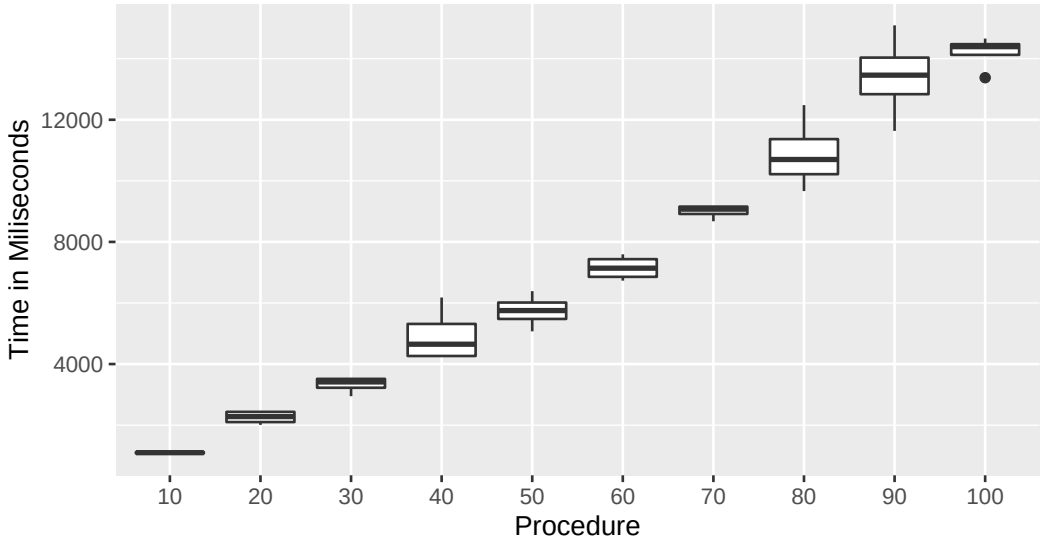

Figure 7.3. Benchmark Test: Read all Pages from a Wiki.

The boxplot shows the times in milliseconds to read the title for all Pages of a Wiki. The time T for reading pages from the Wiki increases linearly with the number of Pages N_p and can be defined as $T = N_p * 140ms$

7.2 Solid Git

The performed benchmark test, shown in Appendix C.2 examines Solid Git's init and commit procedures. The test ran against the dumb and the smart git backend. For both cases the Apache proxy was configured to serve HTTP/1 and HTTP/2. Figure 7.4 and 7.5 show that the smart backend is 4 to 6 times faster than the dumb backend.

7.3 Tree-Merge Algorithm

The Tree-Merge algorithm was developed test-driven and iteratively. The unit test listed in Appendix C.3 shows the tested subset of possible merge-operations. The test cases aim to generalize to arbitrary edits by covering a broad collection of edits. The unit tests cover simple edits, inserts, updates and deletion of text but also more complex structural modification. All unit tests run successfully, which ensures basic usability but does not prove general correctness.

7.4 Recipes Wiki

The Recipes Wiki, an exemplary Wiki was created for demo purposes. We made use of a recipes colleciton from the *Open Recipes* [24] GitHub repository. The databases dumb contain 173278 recipes from 33 websites. We created a Solid user *recipes* and imported all 173278 recipes to its Solid pod. The import procedure creates a Wiki for every website and adds all recipes to the corresponding Wiki. The resulting Wikis are explorable by the demo application under this *link* [25] . Additionall, we provide screenshots in Figure 7.6, 7.7, 7.8 and 7.9, which present four screen in case the demo is not maintained anymore. The screens show four aspects: Firstly, a list of Wikis. Secondly, an overview of a specific Wiki shows the source, notification and subscription information cards. Thirdly, a list of all pages in a federated Wiki where the colors indicate the different source Wiki. Finally, a Wiki page where the buttons from left to right allow the user to edit, view the history, fork or merge the page.

[24] https://github.com/fictivekin/openrecipes
[25] https://wiki.solid.ma.parrillo.eu/wikis/?user=https://recipes.solid.ma.parrillo.eu/profile/card#me

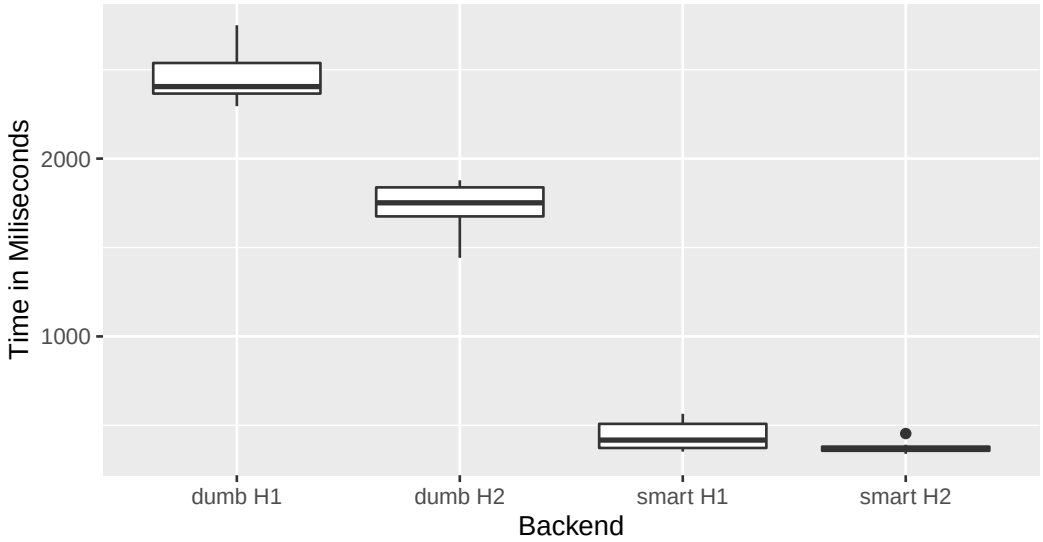

Figure 7.4. Benchmark Test: Initiate a Git Repository

The boxplot shows the times in milliseconds for initiating a Git repository with the dumb and smart Git Backend. Both backends were tested against serving over HTTP/1 (H1) and an HTTP/2 (H2).

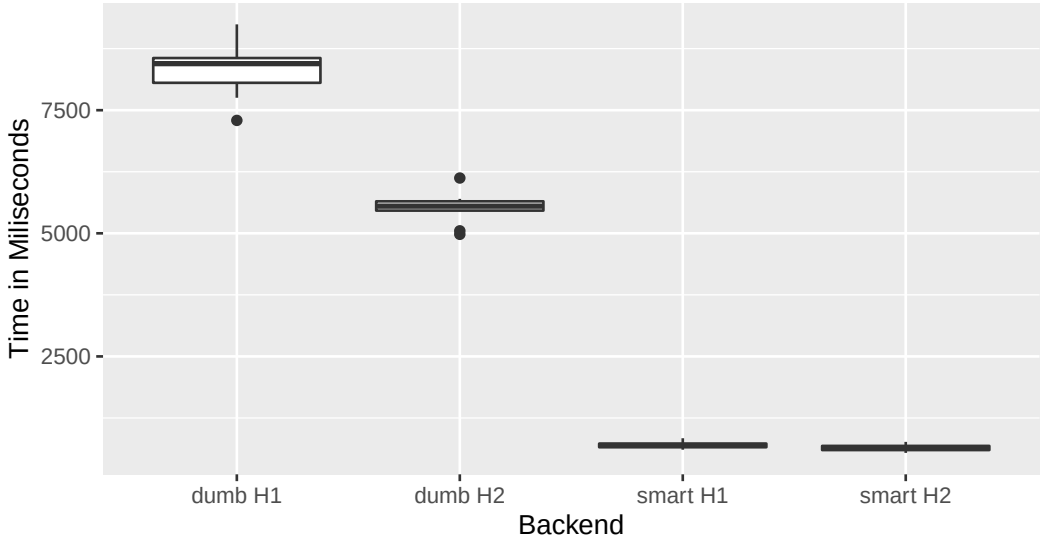

Figure 7.5. Benchmark Test: Git Commit

The boxplot shows the times in milliseconds for committing a Git repository with the dumb and smart Git Backend. Both backends were tested against serving over HTTP/1 (H1) and an HTTP/2 (H2).

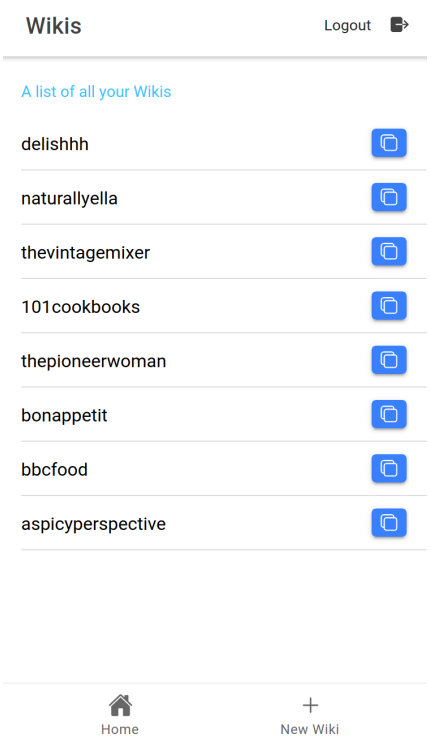

Figure 7.6. Screenshot: List of all Wikis

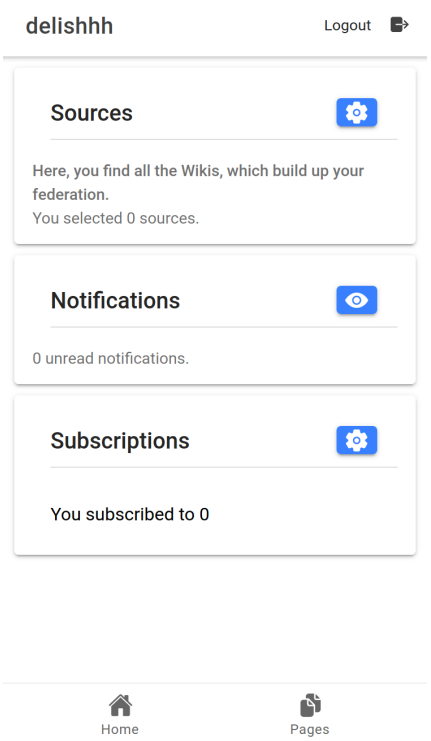

Figure 7.7. Screenshot: Wiki Overview

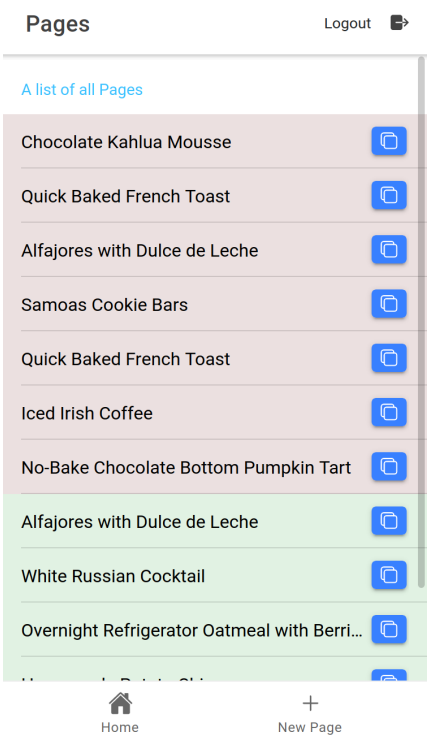

Figure 7.8. Screenshot: List Pages

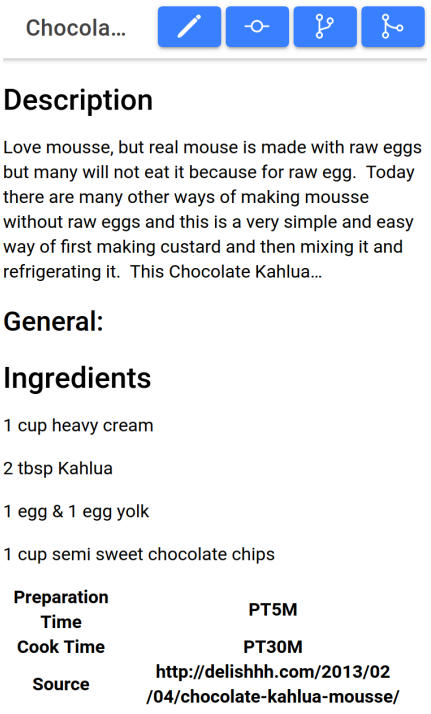

Figure 7.9. Screenshot: Page View

Chapter Eight

Discussion and Future Work

This Chapter discusses our work, provides insights regarding design decisions and draws attention to the limitations of our implementation. In the second part of this Chapter, we discuss and share our work experience with the Solid ecosystem. Also, we name topics for future work.

8.1 Solid Wiki

Our implementation successfully fulfils the mentioned requirements defined within the scope of this thesis and implements a decentralized Wiki based on the Solid ecosystem. Solid Wiki consists of the Wiki services, the Tree-Merge algorithm, the Notification protocols and the integration of the Git versioning system.

The implementation allows the creation of federated Wikis. Users can define arbitrary source Wikis which span a federated Wiki as described in Section 3.6.5. This compelling but straightforward design of the Wiki Ontology, as shown in Figure 6.3, allows for arbitrary Wiki composition. The composition of Wikis is thereby independent of the physical location. The application takes only the semantic connection induced by the user into account, as highlighted in Section 4.1. However, an application developer must be cautious since recursive relationships are valid. Solid mainly provides the basis to realize this federation characteristic. More precisely, it derives from LD which makes this particular type of resource linking extremely simple. Although LD did not force us to define a specific ontology, we felt the need to explore the capabilities of RDF, RDFS and OWL. The result is exceptionally appealing since a well-defined ontology can be used with many generic tools. An example of one such tool is WebVOWL, which we used to generate the visualizations. Future work should explore additional aspects of how to use these ontologies. We believe it is powerful because it allows us to generate the domain repositories or documentation automatically.

Furthermore, the developed page change synchronization protocol is effective, secure and low-maintenance, since there is no need for sender verification. Hence, users only require read access for foreign pods. However, this protocol is only useful for RSS-like notifications.

Our benchmark tests showed that if we load multiple resources from a container it quickly leads to performance issues. The performance mainly depends on two factors. The first factor is the hardware specifications, which are the clients, the servers as well as the intermediate network connection. The second factor is the software specifications, which depends on how the application developer decides to organize collections. The LDP specification recommends to organize collections as LDPC. As a result, the loading time for the entire collection increases linearly as shown in Figure 7.1 and 7.2.

With the current state of the Solid specifications and the NSS, we saw two options to overcome this performance issue. We could have defined all the collection in a single resource. Nevertheless, this would violate the LDP recommendation and defeat the point of LDPC. The other option would make use of the globing functionality provided by NSS. Unfortunately, the current server implementation response only includes the content of all resources but with no association to the resource location. Therefore, we dismiss this option at the moment since it is only useful for presenting but not updating data. Also, globing might not be supported in the future.

Future work should assess this shortcoming and most probably evolve the Solid server specification. A possible approach could be that NSS accepts SPARQL queries against a container. The query would thereby evaluate based on the resources in the container.

Users can collaborate multi-synchronously and curate their collection of pages and organize them in Wikis, where the content and definition of Wikis or pages is up to the users' preferences. Additionally, the forking functionality allows for seamless integration of pages to a user's pod such that a page becomes editable and aligned with the idea of a read-write Web.

The current implementation lacks availability and completeness for the fork tree, since the current design of fork propagation only keeps track of adding direct forks of a page. As a result, the fork tree can only be traced back to the source if all forks along the fork tree exist.

Further work should investigate the extension of the protocol to synchronize all pages along the fork tree. Although this may sound trivial, it is difficult to insure the validity of a fork to prevent bad actors from spreading misinformation.

We explored two possible scenarios and sketched the big picture. However, we believe there is a wide range of possible domains where Solid Wiki could be used. This space of possible use-cases originates by the Wikis generic framework, which does not impose any restrictions on its content. This generic feature allows our Wiki to be deployed on many domains. As an additional example, we explore the application space within universities or teaching in general. For instance, a teacher can use the Solid Wiki to collaborate on a group presentation. The students can now use the same presentation to summarize it collaboratively. Consequently, the next generation of students could build on the summaries of the previous semester. As a result, former students could explore decades later what, how and when things changed.

In summary, we implemented the desired basic functionality, which could be used for the above example. However, there are still quite a few unanswered and exciting questions to research. How can we efficiently implement search functionality besides the proposed method of relating to search engines which operate on the Web 3.0? How and to what extent can we add a partial, subjective view of the federated Wiki spanned over all transitive Wikis?

8.2 Solid Git

We implemented the Dumb Git Transfer Protocol, which relies on a custom implementation of a Solid client-side file system. We then continued to implement the Smart Git Transfer Protocol, which required us to integrate the Git server functionalities to NSS.

The dumb protocol perfectly demonstrated the obstacles which an intense read-write application would suffer from. The evaluation in Section 7.2 underlines the claim that it takes four to six times longer to process compared to the smart implementation. On the other side, the smart protocol yields impressive and promising results with loading times under 500ms.

However, the smart protocol additionally implements an applications protocol that reduces the number of requests to a minimum, which leads to the speedup. Even if the specification of Solid server would be changed to serve a Git Backend, this approach is not

generally applicable.

Therefore, future work should investigate how to revise the Solid server specification to allow application-specific protocols with a minimum communication overhead. This question could again be tackled by allowing clients to execute queries over a defined set of resources on the server.

8.3 Tree-Merge Algorithm

The developed Tree-Merge algorithm is able to arbitrary merge HTML in a three-way merge fashion. Additionally, the algorithm uses the structural information encoded in the HTML format to reduce merge conflicts. The use of this additional information benefits the merge result in two ways. The merge result is valid HTML, with structural and textual edits merged conflict-free in regards to this file format.

Ultimately, we implemented a data structure that acts like a Conflict-free Replicated Data Type (CRDT). However, a formal proof of the algorithm remains subject to future work. Nevertheless, we prepared an experimental foundation inspired by similar research [62], which we verified with Unit-Tests as shown in Appendix C.3.

Future work should further formalize and validate this approach. Additional research should also investigate how CRDT could be used in a multi-synchronous collaboration environment where the data structure is only used during the merge process. Additionally, further research should extend the algorithm to merge HTML attributes, such as `href` or `style`. Notably, the `href` attribute must be supported to prevent the merge algorithm form breaking Wiki links.

8.4 Solid Ecosystem

Basically, at the beginning of our work in early 2020, the Solid Ecosystem [26] specification was still in a provisional state in. The specification consisted mostly of titles used as placeholders. Therefore this thesis generally bases on the Unofficial Solid Specification Draft [27] . Fortunately, the specification evolves weekly. The contribution from this year already exceeds the contribution from 2019 (71 Commits) at the beginning of July (108 Commits). This positive development indicates that the Solid ecosystem is growing and becoming more mature.

[26] https://solid.github.io/specification/
[27] https://github.com/solid/solid-spec

Chapter Nine

Conclusion

We implemented a decentralized Wiki based on Web technologies defined by the Solid specifications. The multi-synchronous collaboration model at its core combined with a merge algorithm integratable into rich text editors promotes it to a broader non-technical audience. On top of these functions, we allow users to explore our implementation via a simple UI using a browser.

Within our work, we sketched two possible scenarios. The recipes and popular initiatives are just a fraction of the possible use-cases for Solid Wiki. We envision a wide range of possible applications.

As our evaluation has shown, our idea to synchronize notifications with a pull-semantic has proven to be effective at reducing sender verification. We have shown that such a secure and low maintenance approach will be straightforward for data publishers, such as newspapers, to integrate into their current publishing stack. When publishers adapt our methodology, their content could be saved and edited with ease. Not only would Solid Wiki allow a user to organize content from a variety of sources, but also it allows to curate content over a long period. Ultimately, the publisher could form a symbiotic relationship from greater user engagement with their content.

The Solid ecosystem provides a practical methodology to separate data form service providers. The segregation allows other application to reuse the data, which is similar to the UNIX style maxim [67] to *expect the output of every program to become the input to another, as yet unknown program*, which we could now practice on the World Wide Web. In the spirit of the Web, to make information available to the world, we explored and successfully developed a Wiki that lets us subjectively curate, share and collaborate on information encountered on the Web.

Bibliography

[1] Jim Isaak and Mina J Hanna. User data privacy: Facebook, cambridge analytica, and privacy protection. *Computer*, 51(8):56–59, 2018.

[2] Nathaniel Persily. The 2016 us election: Can democracy survive the internet? *Journal of democracy*, 28(2):63–76, 2017.

[3] Edward L Bernays. *Propaganda*. Ig publishing, 2005.

[4] Florian Zollmann. Bringing propaganda back into news media studies. *Critical Sociology*, 45(3):329–345, 2019.

[5] Alex Carey. *Taking the risk out of democracy: Corporate propaganda versus freedom and liberty*. University of Illinois Press, 1997.

[6] David Miller and William Dinan. *A century of spin: How public relations became the cutting edge of corporate power*. Pluto Press London, 2008.

[7] E. Pariser. *The Filter Bubble: What The Internet Is Hiding From You*. Penguin Books Limited, 2011.

[8] Seth Flaxman, Sharad Goel, and Justin M Rao. Ideological segregation and the effects of social media on news consumption. *Available at SSRN*, 2363701, 2013.

[9] Ken Ward. Social networks, the 2016 us presidential election, and kantian ethics: applying the categorical imperative to cambridge analytica's behavioral microtargeting. *Journal of Media Ethics*, 33(3):133–148, 2018.

[10] Tim Berners-Lee and Kieron O'Hara. The read-write linked data web. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 371:20120513, 03 2013.

[11] Banesh Hiremath and Anand Kenchakkanavar. An alteration of the web 1.0, web 2.0 and web 3.0: A comparative study. *imperial journal of interdisciplinary research*, 2:705–710, 03 2016.

[12] Chelsea Barabas, Neha Narula, and Ethan Zuckerman. Defending internet freedom through decentralization: Back to the future. *The Center for Civic Media & The Digital Currency Initiative MIT Media Lab*, 2017.

[13] Genevieve Gebhart and Tadayoshi Kohno. Internet censorship in thailand: User practices and potential threats. In *2017 IEEE European symposium on security and privacy (EuroS&P)*, pages 417–432. IEEE, 2017.

[14] Dan Goodin. Tunisia plants country-wide keystroke logger on facebook. *Online at http://www. theregister. co. uk/2011/01/25/tunisia facebook password slurping/, retrieved*, pages 11–19, 2012.

[15] I Thomson. Facebook "glitch'that deleted the philando castile shooting vid: It wcis the police-souirces, 2016.

[16] Engin Bozdag. Bias in algorithmic filtering and personalization. *Ethics and information technology*, 15(3):209–227, 2013.

[17] Reuben Binns, Michael Veale, Max Van Kleek, and Nigel Shadbolt. Like trainer, like bot? inheritance of bias in algorithmic content moderation. In *International conference on social informatics*, pages 405–415. Springer, 2017.

[18] VV AA. Decentralization: A sampling of definitions, 1999.

[19] Paul Baran. The beginnings of packet switching: some underlying concepts. *IEEE Communications Magazine*, 40(7):42–48, July 2002.

[20] Paul Baran. *On Distributed Communications: I. Introduction to Distributed Communications Networks*. RAND Corporation, 1964.

[21] Nathan Schneider. Decentralization: an incomplete ambition. *Journal of Cultural Economy*, 12(4):265–285, April 2019.

[22] Bryn Loban. Between rhizomes and trees: P2p information systems. *First Monday*, 9(10), October 2004.

[23] Eytan Adar and Bernardo Huberman. Free riding on gnutella. *First Monday*, 5, 04 2001.

[24] Yochai Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, 2006.

[25] Evgeny Ponomarev. Decentralized web developer report 2020. Report, Fluence Labs. https://medium.com/fluence-network/decentralized-web-developer-report-2020-5b41a8d86789.

[26] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI Global, 2011.

[27] Markus Lathaler Richard Cyganiak, David Wood. Rdf 1.1 concepts and abstract syntax. W3c recommendation, W3C, 2012. https://www.w3.org/TR/owl2-overview/.

[28] Ramanathan V. Guha Dan Brickley. Rdf schema 1.1. W3c recommendation, W3C, 2014. http://www.w3.org/TR/rdf-schema/.

[29] W3C OWL Working Group. Owl 2 web ontology language document overview (second edition). W3c recommendation, W3C, 2012. https://www.w3.org/TR/owl2-overview/.

[30] Google. Enable rich results with structured data. Google guides, Google. https://developers.google.com/search/docs/guides/intro-structured-data.

[31] TIM BERNERS-LEE, JAMES HENDLER, and ORA LASSILA. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[32] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE intelligent systems*, 21(3):96–101, 2006.

[33] Matthias Samwald, Jose Antonio Miñarro Giménez, Richard D Boyce, Robert R Freimuth, Klaus-Peter Adlassnig, and Michel Dumontier. Pharmacogenomic knowledge representation, reasoning and genome-based clinical decision support based on owl 2 dl ontologies. *BMC medical informatics and decision making*, 15(1):1–10, 2015.

[34] Janet Piñero, Núria Queralt-Rosinach, Àlex Bravo, Jordi Deu-Pons, Anna Bauer-Mehren, Martin Baron, Ferran Sanz, and Laura I. Furlong. DisGeNET: a discovery platform for the dynamical exploration of human diseases and their genes. *Database*, 2015, 04 2015. bav028.

[35] Javier D Fernández, Nelia Lasierra, Didier Clement, Huw Mason, and Ivan Robinson. Enabling fair clinical data standards with linked data.

[36] Niel Chah. OK google, what is your ontology? or: Exploring freebase classification to understand google's knowledge graph. *CoRR*, abs/1805.03885, 2018.

[37] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. Introduction: What is a knowledge graph? In *Knowledge Graphs*, pages 1–10. Springer, 2020.

[38] Amy Guy Sarven Capadisli. Linked data notifications. W3c recommendation, W3C. https://www.w3.org/TR/ldn/.

[39] Sarven Capadisli, Amy Guy, Christoph Lange, Sören Auer, Andrei Sambra, and Tim Berners-Lee. Linked data notifications: A resource-centric communication protocol. pages 537–553, 05 2017.

[40] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. Solid: A platform for decentralized social applications based on linked data, 2016.

[41] W3C Solid Community Group. The solid ecosystem. W3c editors's draft, W3C. https://solid.github.io/specification.

[42] Tim Berners-Lee Andrei Sambra, Henry Story. Web identity and discovery. W3C editor's draft, W3C, 2014. https://www.w3.org/2005/Incubator/webid/spec/identity/.

[43] John Arwe-IBM Corporation Ashok Malhotra Oracle Corporation Steve Speicher, IBM Corporation. Linked data platform 1.0. W3c recommendation, W3C, 2014. https://www.w3.org/TR/ldp/.

[44] Bruno Harbulot Toby Inkster, Henry Story. Webid authentication over tls. W3C editor's draft, W3C, 2014. https://www.w3.org/2005/Incubator/webid/spec/tls/.

[45] Sacip Toker, James L. Moseley, and Ann T. Chow. Is there a wiki in your future? applications for education, instructional design, and general use. *Educational Technology*, 48(5):22–27, 2008.

[46] Ward Cunningham. *What Is Wiki*, 1995 (accessed November 23, 2019).

[47] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. pages 1–11, 09 2019.

[48] Haya Ajjan and Richard Hartshorne. Investigating faculty decisions to adopt web 2.0 technologies: Theory and empirical tests. *The Internet and Higher Education*, 11(2):71 – 80, 2008.

[49] Lara C. Ducate, Lara Lomicka Anderson, and Nina Moreno. Wading through the world of wikis: An analysis of three wiki projects. *Foreign Language Annals*, 44(3):495–524, August 2011.

[50] François Bry, Sebastian Schaffert, Denny Vrandečić, and Klara Weiand. *Semantic Wikis: Approaches, Applications, and Perspectives*, pages 329–369. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[51] Ward Cunningham. *Wiki Design Principles*, 1995 (accessed November 23, 2019).

[52] Brian Lamb. Wide open spaces: Wikis, ready or not. *EDUCAUSE review*, 39(5):36–48, 2004.

[53] Alan Davoust, Hala Skaf-Molli, Pascal Molli, Babak Esfandiari, and Khaled Aslan. Distributed wikis: A survey. *Concurrency and Computation: Practice and Experience*, 27, 11 2014.

[54] François Bry, Sebastian Schaffert, Denny Vrandečić, and Klara Weiand. *Semantic Wikis: Approaches, Applications, and Perspectives*, pages 329–369. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[55] Jakob Voss. Collaborative thesaurus tagging the wikipedia way, 2006.

[56] Paul Dourish. A divergence-based model of synchrony and distribution in collaborative systems, 1994.

[57] Ward Cunningham and Michael W Mehaffy. Wiki as pattern language. In *Proceedings of the 20th Conference on Pattern Languages of Programs*, page 32. The Hillside Group, 2013.

[58] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

[59] Benjamin Paaßen. Revisiting the tree edit distance and its backtracing: A tutorial. *arXiv preprint arXiv:1805.06869*, 2018.

[60] Luuk Peters. Change detection in xml trees: a survey. In *3rd Twente Student Conference on IT*, 2005.

[61] Beat Fluri, Michael Wursch, Martin PInzger, and Harald Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on software engineering*, 33(11):725–743, 2007.

[62] Dimitar Asenov, Balz Guenat, Peter Müller, and Martin Otth. Precise version control of trees with line-based version control systems. In *International Conference on Fundamental Approaches to Software Engineering*, pages 152–169. Springer, 2017.

[63] Jedidiah McClurg, Sebastian Burckhardt, Michał Moskal, and Jonathan Protzenko. difftree: Robust collaborative coding using tree-merge. 2015.

[64] What is a federal popular initiative? https://www.ch.ch/en/demokratie/political-rights/popular-initiative/what-is-a-federal-popular-initiative/.

[65] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[66] Dong Ah Kim and Suk-Kyoon Lee. Efficient change detection in tree-structured data. In Chin-Wan Chung, Chong-Kwon Kim, Won Kim, Tok-Wang Ling, and Kwan-Ho Song, editors, *Web and Communication Technologies and Internet-Related Social Issues — HSI 2003*, pages 675–681, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[67] E.N.; Tague B.A. McIlroy, M.D.; Pinson. Unix time-sharing system: Forward. *THE BELL SYSTEM TECHNICAL JOURNAL*, 57:6, 8 1978. https://archive.org/details/bstj57-6-1899/mode/2up.

List of Figures

2.1 Centralized, Decentralized and Distributed Networks [20] 6

3.1 Overview of Linked Data Notification 10
3.2 Solid Architecture [40] 12

4.1 Solid Wiki Operational Overview 19

6.1 Solid Wiki Architecture 26
6.2 Solid App Discovery as Domain-Driven Design Example 28
6.3 Wiki Ontology: Wiki Data Model 31
6.4 Wiki Ontology: Page Data Model 32
6.5 Notification of Page Changes 36
6.6 Notification of Forking 37
6.7 Smart Git Transfer Protocol with Minimal Solid Fs 38
6.8 Smart Git Transfer Protocol with Git HTTP Backend 39
6.9 Tree-Merge Algorithm 42

7.1 Benchmark Test: Create and Gets all Wikis and Pages. 44
7.2 Benchmark Test: Add all Pages to a Wiki. 44
7.3 Benchmark Test: Read all Pages from a Wiki. 45
7.4 Benchmark Test: Initiate a Git Repository 47
7.5 Benchmark Test: Git Commit 47
7.6 Screenshot: List of all Wikis 48
7.7 Screenshot: Wiki Overview 48
7.8 Screenshot: List Pages 48
7.9 Screenshot: Page View 48

C.1 Server CPU Usage Monitoring 79
C.2 Client CPU Usage Monitoring 79

Glossary

API Application Programming Interface. 4, 29

CGI Common Gateway Interface. 39

CRDT Conflict-free Replicated Data Type. 52

DDD Domain-Driven Design. 2, 26

DNS Domain Name System. 7

DVCS Decentralized Version Control Systems. 16

DWeb Decentralized Web. 7

FTP File Transfer Protocol. 4

GFW Global Federated Wiki. 18

HTML HyperText Markup Language. 4, 13, 14, 18, 25, 29, 32, 40–42, 52

HTTP Hypertext Transfer Protocol. 4, 7, 10, 19, 38, 39, 46, 47, 59

LD Linked Data. 2, 8, 9, 27, 29, 49

LDN Linked Data Notification. 3, 10, 25, 37

LDP Linked Data Platform. 9, 10, 30, 50

LDPC Linked Data Platform Container. 9, 30, 33, 50

LDPR Linked Data Platform Resource. 9

LDR Linked Data Resource. 27

NGHB Node Git HTTP Backend. 2, 39

NSS Node Solid Server. 38, 39, 50, 51

OWL Web Ontology Language. 8, 49

P2P Peer-to-Peer. 7, 16

RDF Resource Description Framework. 8–11, 15, 27, 29, 30, 49

RDFS Resource Description Framework Schema. 8, 49

RSS Really Simple Syndication. 36, 50

SPARQL SPARQL Protocol and RDF Query Language. 11, 50

UI User Interface. 26, 29, 54

URI Uniform Resource Identifier. 10, 20, 27

URL Uniform Resource Locator. 10, 20, 34, 39

WYSIWYG What You See Is What You Get. 13

Appendix A

Turtle Files

A.1 Wiki

```
@prefix wiki:  <http://example.org/wiki/> .

wiki:
    a                 wiki:Wiki,
                      ldp:Resource ;
    wiki:title        "A Wiki Title" ;
    wiki:config       <config> ;
    wiki:pageIndex    <page-index> ;
    wiki:subscription  <subscriptions/> ;
    wiki:pageFeedInbox <page-feed-inbox/>
.
```

Listing A.1: http://example.org/wiki/wiki

A.2 Config

```
@prefix config:     <https://example.org/wiki/config.ttl> .
@prefix sourceList: <https://example.org/wiki/config/source-list.ttl> .

config:
    a               wiki:Config,
                    ldp:Resource ;
    wiki:sourceList sourceList:
.
```

Listing A.2: http://example.org/wiki/config

A.3 Empty Source List

```
@prefix sourceList:      <https://example.org/wiki/config/source-list> .
@prefix privateWorkWiki: <https://example.org/private/work/wiki.ttl> .
@prefix bobsCoolWiki:    <https://bob.com/public/cool-wiki/wiki.ttl> .

sourceList:
    a            wiki:SourceList,
                 ldp:Resource ;
    ldp:contains bobsCoolWiki:,
                 privateWorkWiki:
.
```

Listing A.3: http://example.org/wiki/config/source-list

A.4 Page Index

```
@prefix pageIndex:    <http://example.org/wiki/page-index.ttl> .
@prefix pageInstance: <http://example.org/wiki/pages/CoolPage/page.ttl> .

pageIndex:
    a solidTerms:TypeIndex,
      wiki:PageIndex,
      ldp:Resource ;
.

:703948b4df1d3ab77244c7d17f67b43fbb9d74fdbcb372e8af818d7b64db7bf
    a                   solidTerms:TypeRegistration ;
    solidTerms:forClass wiki:Page ;
    solidTerms:instance pageInstance: ;
.
```

Listing A.4: http://example.org/wiki/page-index

A.5 Page

```
@prefix wiki: <http://parrillo.eu/ns/solid/wiki#> .
@prefix ldp:  <http://www.w3.org/ns/ldp#> .


page:
    a              wiki:Page,
                   ldp:Resource ;
    wiki:body      <pageDir/body.html> ;
    wiki:fork      <pageDir/forks/> ;
    wiki:forkInbox <pageDir/fork-inbox/> ;
.
```

Listing A.5: http://example.org/wiki/pages/first-page/page

A.6 Page Title

```
@prefix wiki: <http://parrillo.eu/ns/solid/wiki#> .
@prefix page: <page> .

page:
    wiki:title "First Page"
.
```

Listing A.6: http://example.org/wiki/pages/first-page/page-title

A.7 Git Commit Object

```
@prefix :          <#> .
@prefix wiki:      <http://parrillo.eu/ns/solid/wiki#> .
@prefix ldp:       <http://www.w3.org/ns/ldp#> .
@prefix schema:    <http://www.schema.org/> .
@prefix bob:       <https://bob.solid.ma.parrillo.eu/profile/card#me> .
@prefix git:       <https://git> .
@prefix gitObject: <../.git/objects/> .
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:      <http://www.w3.org/2000/01/rdf-schema#> .


<./.git/object/23/4afjfsaf0923809asdf>
    a          git:Commit,
               ldp:Resource ;
    git:tree   <../../.git/objects/4c/910246a447995f3999fe52e7c2eedd1cad6fe7> ;
    git:parent <../../.git/objects/23/4590238450983425734856> ;
    git:author [
                   git:name  "Bob" ;
                   git:email "bob@hello"
               ] ;
.

<../../.git/4c/910246a447995f3999fe52e7c2eedd1cad6fe7>
    a            git:Tree,
                 ldp:Container ;
    ldp:contains gitObject:4c910246a,
                 gitObject:3999fe52e7,
                 gitObject:d1cad6fe7
.

<../../.git/4c/910246a447995f3999fe52e7c2eedd1cad6fe7>
    a            git:Tree,
                 ldp:Container ;
    ldp:contains gitObject:3999fe52e7,
                 gitObject:d1cad6fe7
.

<../../.git/objects/39/99fe52e7>
    a        git:Blob,
             ldp:NonRDFSource ;
    git:path <page/body.html>
.

<../../.git/objects/39/99fe52e7>
    a        git:Blob,
             ldp:NonRDFSource ;
    git:path <page/title>
.

<../../.git/objects/23459023845098342573485 6>
    a            ldp:NonRDFSource ;
    rdfs:seeAlso <../../.git/objects/23/4590238450983425734856>
.
```

Listing A.7: http://example.org/wiki/pages/first-page/commits/234afjfsaf0923809asdf

Appendix B

Implementation

B.1 Solid Wiki Ontology

```
@prefix :           <http://parrillo.eu/ns/solid/wiki> .
@base               <http://parrillo.eu/ns/solid/wiki> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:        <http://www.w3.org/2002/07/owl#> .
@prefix skos:       <http://www.w3.org/2004/02/skos/core#> .
@prefix dcterms:    <http://purl.org/dc/terms/> .
@prefix rdfs:       <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ldp:        <http://www.w3.org/ns/ldp#> .
@prefix wiki:       <http://parrillo.eu/ns/solid/wiki> .
@prefix solidTerms: <http://www.w3.org/ns/solid/terms#> .
@prefix xsd:        <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

:
    a          owl:Ontology ;
    rdfs:label "Solid Wiki Ontology"
.

#########################################################
## Wiki Classes and Properties
#########################################################
:Wiki
    a          rdfs:Class ;
    rdfs:label "Wiki"
.

:wikiTitle
    rdf:type           rdf:Property ;
    rdfs:label         "title" ;
    rdfs:subPropertyOf dcterms:title ;
    rdfs:domain        :Wiki ;
    rdfs:range         xsd:string ;
.

:pageIndex
    a           rdf:Property ;
    rdfs:label  "page index" ;
    rdfs:domain :Wiki ;
    rdfs:range  :PageIndex ;
.

:config
    a           rdf:Property ;
    rdfs:label  "config" ;
    rdfs:domain :Wiki ;
    rdfs:range  :Config ;
.

:pageFeedInbox
    a                  rdf:Property ;
    rdfs:label         "page feed inbox" ;
    rdfs:subPropertyOf ldp:inbox ;
    rdfs:domain        :Wiki ;
    rdfs:range         :PageFeedContainer
.
```

```
56
57  wiki:subscriptionContainer
58      rdfs:label          "subscription container" ;
59      a                   rdf:Property ;
60      rdfs:subPropertyOf  ldp:Container ;
61      rdfs:domain         wiki:Wiki ;
62      rdfs:range          wiki:SubscriptionContainer
63  .
64
65  #############################################################
66  ## Wiki Page Index Classes and Properties
67  #############################################################
68
69  :PageIndex
70      rdfs:subClassOf solidTerms:TypeIndex ;
71      rdfs:label      "Page Index" ;
72  .
73
74
75  #############################################################
76  ## Wiki Config Classes and Properties
77  #############################################################
78
79  :Config
80      a          rdfs:Class ;
81      rdfs:label "Config" ;
82  .
83
84  :sourceList
85      a           rdf:Property ;
86      rdfs:label  "source list" ;
87      rdfs:domain wiki:Config ;
88      rdfs:range  wiki:SourceList ;
89  .
90
91  #############################################################
92  ## Config Source List Classes and Properties
93  #############################################################
94
95  :SourceList
96      rdfs:label  "Source List" ;
97      a           rdfs:Class ;
98      rdfs:domain ldp:Resource ;
99  .
100
101
102 :source
103     a           rdf:Property ;
104     rdfs:label  "source" ;
105     rdfs:domain :SourceList ;
106     rdfs:range  wiki:Source ;
107 .
108
109 #############################################################
110 ## Source Classes and Properties
111 #############################################################
112
113 :Source
114     rdfs:label  "Source" ;
115     a           rdfs:Class ;
116     rdfs:domain ldp:Resource ;
117 .
118
119 #############################################################
120 ## Page Feed Container Class and Properties
121 #############################################################
122
```

```
wiki:PageFeedContainer
    a               rdfs:Class ;
    rdfs:subClassOf ldp:Container ;
    rdfs:label      "Page Feed Container" ;
.

wiki:containsFeedEntries
    a                   rdf:Property ;
    rdfs:label          "contains" ;
    rdfs:subPropertyOf  ldp:contains ;
    rdfs:domain         wiki:PageFeedContainer ;
    rdfs:range          solidTerms:Notification
.

#########################################################
## Page Feed Container Class and Properties
#########################################################

:PageFeed
    a               rdfs:Class ;
    rdfs:subClassOf ldp:Container ;
    rdfs:label      "Page Feed" ;
    rdfs:domain     wiki:Page ;
    rdfs:range      wiki:PageFeed ;
.


#########################################################
## Subscription Container Classes and Properties
#########################################################

wiki:SubscriptionContainer
    a               rdfs:Class ;
    rdfs:subClassOf ldp:Container ;
    rdfs:label      "Subscription Container" ;
.


:containsSubscriptios
    a                   rdf:Property ;
    rdfs:subPropertyOf  ldp:contains ;
    rdfs:label          "contains" ;
    rdfs:domain         wiki:SubscriptionContainer ;
    rdfs:range          wiki:Subscription ;
.

#########################################################
## Subscription  Classes and Properties
#########################################################

:Subscription
    a          rdfs:Class ;
    rdfs:label "Subscription" ;
.

wiki:subscriptionTarget
    a           rdf:Property ;
    rdfs:label  "subscription target" ;
    rdfs:range  wiki:Wiki,
                wiki:Page ;
    rdfs:domain wiki:Subscription ;
.

#########################################################
## Page Classes and Properties
#########################################################
:Page
```

```
    rdfs:label "Page" ;
    a          rdfs:Class
.

:pageTitle
    rdf:type           rdf:Property ;
    rdfs:label         "title" ;
    rdfs:subPropertyOf dcterms:title ;
    rdfs:domain        :Page ;
    rdfs:range         xsd:string ;
.

wiki:body
    a           rdf:Property ;
    rdfs:label  "body" ;
    rdfs:domain wiki:Page ;
    rdfs:range  wiki:Body
.

wiki:compiled
    a           rdf:Property ;
    rdfs:label  "compiled" ;
    rdfs:domain wiki:Page ;
    rdfs:range  wiki:CompiledPage
.

:forkOf
    a           rdfs:Class ;
    rdfs:label  "fork of" ;
    rdfs:domain wiki:Page ;
    rdfs:range  wiki:Page ;
.

:forkedBy
    a                  rdf:Property ;
    rdfs:subPropertyOf rdf:predicate ;
    rdfs:label         "forked by" ;
    rdfs:domain        :Page ;
    rdfs:range         :Page .

:forksDocument
    a           rdf:Property ;
    rdfs:label  "forks document" ;
    rdfs:domain wiki:Page ;
    rdfs:range  wiki:ForksDocument ;
.

wiki:forkInbox
    a                  rdf:Property ;
    rdfs:subPropertyOf ldp:inbox ;
    rdfs:label         "fork inbox" ;
    rdfs:domain        wiki:Page ;
    rdfs:range         wiki:ForkInbox
.

#########################################################
## Page Body Classes and Properties
#########################################################
:Body
    rdfs:subClassOf ldp:NonRDFSource ;
    rdfs:label      "Body" ;
    rdfs:comment    "The current version of the page, complied as HTML. (ex. page/
    index.html)"
.

#########################################################
## Compiled Page Classes and Properties
```

```
256 ########################################################
257
258 :CompiledPage
259     rdfs:subClassOf ldp:NonRDFSource ;
260     rdfs:label      "Compiled Page" ;
261     rdfs:comment    "The current version of the page, complied as HTML. (ex. page/
        index.html)"
262 .
263
264 ########################################################
265 ## Fork Statement Classes and Properties
266 ########################################################
267
268 :ForksDocument
269     a          rdfs:Class ;
270     rdfs:label "Forks document" ;
271 .
272
273 ########################################################
274 ## Fork Inbox Classes and Properties
275 ########################################################
276
277 wiki:ForkInbox
278     a               rdfs:Class ;
279     rdfs:subClassOf ldp:Container ;
280     rdfs:label      "Fork Inbox" ;
281 .
282
283 wiki:forkNotificationTarget
284     a           rdf:Property ;
285     rdfs:label  "fork notification target" ;
286     rdfs:domain wiki:ForkInbox ;
287     rdfs:range  wiki:ForkNotification
288 .
289
290 :containsForkNotifications
291     a                  rdf:Property ;
292     rdfs:subPropertyOf ldp:contains ;
293     rdfs:label         "contains" ;
294     rdfs:domain        wiki:ForkInbox ;
295     rdfs:range         wiki:ForkNotification .
296
297
298 ########################################################
299 ## Fork Notification Classes and Properties
300 ########################################################
301 wiki:ForkNotification
302     a               rdfs:Class ;
303     rdfs:subClassOf solidTerms:Notification ;
304     rdfs:label      "Fork Notification" ;
305 .
306
307 :contains
308     a                  rdf:Property ;
309     rdfs:subPropertyOf ldp:contains ;
310     rdfs:label         "contains" ;
311     rdfs:domain        wiki:PageFeed ;
312     rdfs:range         solidTerms:Notification
313 .
```

Listing B.1: Solid Wiki Ontology

B.2 Full Listing of the Implemented Domain Objects

```
domain
         file
         BinaryResource
         iana
             media-types
                text
                   turtle
                       TurtleResource
         linked-data-platform
             BinaryResource
             ContainerRepository
             Container
             ResourceRepository
             Resource
         profile
             ProfileRepository
             ProfileService
             Profile
         rdf
             Literal
             Node
         solid
             terms
                inbox
                    Inbox
                notification
                    NotificationRepository
                    Notification
                typed-index
                   DocumentRepository
                   DocumentService
                   Document
                   TypedIndexRepository
                   TypedIndexService
                   TypedIndex
                   type-registration
                       instance
                          InstanceContainer
                          Instance
                       private
                          PrivateTypedIndexRepository
                          PrivateTypedIndexService
                          PrivateTypedIndex
                       public
                          PublicTypedIndexRepository
                          PublicTypedIndexService
                          PublicTypedIndex
                       TypeRegistration
         wiki
             config
                 ConfigRepository
                 ConfigService
                 Config
                 source-list
                    SourceListRepository
                    SourceListService
                    SourceList
             inbox
                 PageFeedInboxRepository
                 PageFeedInboxService
                 PageFeedInbox
             Index
             page
                 body
                     Body
                 feed
```

```
                    PageFeedRepository
                    PageFeedService
                    PageFeed
                fork-document
                    ForksDocumentRepository
                    ForksDocumentService
                    ForksDocument
                fork-inbox
                    ForkInboxRepository
                    ForkInboxService
                    ForkInbox
                PageRepository
                PageService
                Page
                title
                   Title
            pageIndex
                PageIndexRepository
                PageIndexService
                PageIndex
            subscriptions-container
                subscription
                    SubscriptionRepository
                    Subscription
                SubscriptionContainerRepository
                SubscriptionContainer
            title
                Title
            WikiRepository
            WikiService
            Wiki
```

Listing B.2: List of all Domain Objects: The Listing shows a complete list of all implemented domain objects, where every domain object holds its domain model, repository and service.

Appendix C

Evaluation

C.1 Solid Wiki Benchmark

```
import { ProfileRepository } from '../../services/domain/profile/ProfileRepository'
import { Page }              from '../../services/domain/wiki/page/Page'
import { PageService }       from '../../services/domain/wiki/page/PageService'
import { PageIndexService }  from '../../services/domain/wiki/pageIndex/PageIndexService'
import { WikiService }       from '../../services/domain/wiki/WikiService'
import { session }           from '../../services/session/Session'
import { getHexHash }        from '../../services/utils/Util'

function getRandomBody ( bytes : number ) {
    const buffer = new ArrayBuffer ( bytes )
    const array  = new Uint8Array ( buffer )
    array.fill ( 42 )
    const decoder = new TextDecoder ()
    const content = decoder.decode ( buffer )
    return `<div>${ content }</div>`
}

export async function benchmark1 () {
    const log = ( message : string ) => console.log ( `Run Test: ${ message }` )

    const profileRepository = new ProfileRepository ()
    const wikiService       = new WikiService ()
    const pageService       = new PageService ()
    const pageIndexService  = new PageIndexService ()

    const webId   = await session.getWebId ()
    const origin  = new URL ( webId! ).origin
    const profile = await profileRepository.get ( webId! )

    let result = 'i,test,time\n'

    let pages : Page[] = []
    for ( let a = 0 ; a < 10 ; a++ ) {

        for ( let i = 0 ; i <= 10 ; i++ ) {
            let wikiURL = ''
            let pageURL = ''
            let t1 : Date
            let t2 : Date

            try {
                log ( `Next ${ i }` )
                const wikiName = new Date ().toISOString ()
                const wikiHash = await getHexHash ( wikiName )
                wikiURL        = `${ origin }/public/wikis/${ wikiHash }`

            t1       = new Date ()
            let wiki = await wikiService.getWiki ( wikiURL )
            wiki     = await wikiService.createBlank (
                profile ,
                wikiURL ,
                wikiName ,
                'public'
```

```
54              )
55              t2       = new Date ()
56              // @ts-ignore
57                  result += `${ i },"Create Wiki",${ t2 - t1 }\n`
58
59              t1 = new Date ()
60              await wikiService.getAllWikisByProfile ( profile )
61              t2 = new Date ()
62              // @ts-ignore
63                  result += `${ i },"Get all Wikis",${ t2 - t1 }\n`
64
65              const pageTitle = new Date ().toISOString ()
66              const pageHash  = await getHexHash ( pageTitle )
67              pageURL         = `${ origin }/public/wikis/pages/${ pageHash }`
68
69                  let page       = await pageService.get ( pageURL )
70                  const pageBody = getRandomBody ( 1024 )
71              t1                 = new Date ()
72              page               = await pageService.create (
73                  {
74                      title :     pageTitle ,
75                      body :      pageBody ,
76                      container : pageURL
77                  }
78              )
79                  t2             = new Date ()
80                  // @ts-ignore
81                  result += `${ i },"Create Page",${ t2 - t1 }\n`
82
83                 // log ( `Created new page: ${ wikiURL }` )
84
85
86                  pages = []
87                  for ( let j = 0 ; j < 10 * i ; ++j ) {
88                  //    log ( `Add page: ${ i }   ${ j }` )
89
90                      const pageTitle = new Date ().toISOString ()
91                      const pageHash  = await getHexHash ( pageTitle )
92                      pageURL         = `${ origin }/public/wikis/pages/${ pageHash
    }`
93                      let page        = await pageService.get ( pageURL )
94                      const pageBody  = getRandomBody ( 1024 )
95                      t1              = new Date ()
96                      page            = await pageService.create (
97                          {
98                              title :     pageTitle ,
99                              body :      pageBody ,
100                             container : pageURL
101                         }
102                     )
103                     pages.push ( page )
104                 }
105
106
107                 t1 = new Date ()
108                 for ( let p of pages ) {
109                     log ( `Add page ${ p.toString () }` )
110                     await wikiService.addPageToWiki (
111                         profile ,
112                         wiki ,
113                         p )
114                 }
115                 t2 = new Date ()
116                 // @ts-ignore
117                 result += `${ i },"Add ${ pages.length } Pages",${ t2 - t1 }\n`
118
119
```

```
                t1    = new Date ()
                pages = await pageIndexService.getPages ( wiki )
                log ( `Added page ${ pageURL } to wiki ${ wiki }` )
                t2 = new Date ()
                // @ts-ignore
                result += `${ i },"Get Pages",${ t2 - t1 }\n`

                t1 = new Date ()
                for ( let p of pages ) {
                    await p.getTitle ()
                }
                t2 = new Date ()
                // @ts-ignore
                result += `${ i },"Read ${ pages.length } Pages",${ t2 - t1 }\n`


            }
        catch
            ( e ) {
            log ( `Error page ${ pageURL } to wiki ${ wikiURL }` )
        }
        log ( result )
    }
    }

    console.log ( result )
    log ( `Complete.` )
}
```

Listing C.1: Solid Wiki Test Procedure

C.2 Solid Git Benchmark

```
import { ProfileRepository }   from '../../services/domain/profile/
    ProfileRepository'
import { PageService }         from '../../services/domain/wiki/page/PageService'
import { DumbBackendService }  from '../../services/git/DumbBackendService'
import { SmartBackendService } from '../../services/git/SmartBackendService'
import { session }             from '../../services/session/Session'
import { getHexHash }          from '../../services/utils/Util'

function getRandomBody ( bytes : number ) {
    const buffer = new ArrayBuffer ( bytes )
    const array  = new Uint8Array ( buffer )
    array.fill ( 42 )
    const decoder = new TextDecoder ()
    const content = decoder.decode ( buffer )
    return `<div>${ content }</div>`
}

export async function benchmarkGit () {
    const log = ( message : string ) => console.log ( `Run Test: ${ message }` )

    const profileRepository = new ProfileRepository ()
    const pageService       = new PageService ()

    const webId   = await session.getWebId ()
    const origin  = new URL ( webId! ).origin
    const profile = await profileRepository.get ( webId! )


    async function runInitAndCommit (
        GitBackend ,
        backendType : 'smart' | 'dumb' ,
        iterations : number
    ) {

        let times = 'i,backend,git_init,git_commit\n'
```

```
35          let t1 : Date
36          let t2 : Date
37
38          for ( let i = 0 ; i < iterations ; i++ ) {
39              log ( `Next ${ i }` )
40              const newRepo = await getHexHash ( new Date ().toISOString () )
41              const repoUrl = `${ origin }/public/git/benchmark/${ newRepo }`
42
43              const page = await pageService.create ( {
44                                                          container : repoUrl ,
45                                                          body :      'hey, lets
    test git.' ,
46                                                          title :     'test'
47                                                      } )
48              const git  = new GitBackend ( repoUrl )
49              t1         = new Date ()
50              await git.init ()
51              t2 = new Date ()
52              // @ts-ignore
53              times += `${ i },${ backendType },${ t2 - t1 }`
54
55              t1 = new Date ()
56              await git.commitPage (
57                  page ,
58                  'hello' ,
59                  profile
60              )
61              t2 = new Date ()
62              // @ts-ignore
63              times += `,${ t2 - t1 }\n`
64          }
65
66          log ( times )
67      }
68
69
70      await runInitAndCommit (
71          SmartBackendService ,
72          'smart' ,
73          10
74      )
75      await runInitAndCommit (
76          DumbBackendService ,
77          'dumb' ,
78          10
79      )
80
81
82      log ( `Complete.` )
83  }
```

Listing C.2: Solid Git Test Procedure

C.3 Tree-Merge Unit Tests

Base	`This`
Ours	`This is cool.`
Theirs	`This is`
Result	`This is cool.`

Table C.1

Should merge inserts

Base	`<p>hey</p>`
Ours	`<p>yoo</p><p>hey</p>`
Theirs	`<p>hey</p><p>yoo</p>`
Result	`<p>yoo </p><p>hey </p><p>yoo </p>`

Table C.2

Should merge multiple inserts

Base	`This is not really nice.`
Ours	`This is not nice.`
Theirs	`This is really nice.`
Result	`This is nice.`

Table C.3

Should merge deletions.

Base	`<h1>H1</h1> <div> <p>P1</p><p>P2</p></div><div><p>P1</p><p>P2</p></div>`
Ours	`<div><p>P1</p><p>P2</p></div><div><p>P1</p></div>`
Theirs	`<h1>H1</h1><div><p>P1</p></div><div><p>P2</p></div>`
Result	`<div><p>P1 </p></div><div></div>`

Table C.4

Should merge nested elements and deletions.

Base	`This is not really nice.`
Ours	`This isn't really nice.`
Theirs	`This is not genuinely nice.`
Result	`This isn't genuinely nice.`

Table C.5

Should merge updates.

Base	H1 A0 B0 C0 D0
Ours	H1 A1 B0 C1 D0
Theirs	H2 A0 B2 C0 D2
Result	H2 A1 B2 C1 D2

Table C.6

Should merge nested updates 1.

Base	This is not really nice.
Ours	I think we got it This are not really nice.
Theirs	These is notgenuinenice.
Result	I think we got it These are not genuine nice.

Table C.7

Should merge nested updates 2.

Base	A B C
Ours	ABC
Theirs	A B C
Result	A B C

Table C.8

Should merge multiple move to a deeper level.

Base	A B Ct 1 2 3 X Y Z END
Ours	A B Ct 123 X2 END
Theirs	A B 1 2 3 Insert X Y Z END
Result	A B 123 Insert X2 END

Table C.9

Should multiple structural node updates.

Base	
Ours	A
Theirs	A B
Result	A B

Table C.10

Should merge edits from an empty base.

Base	
Ours	A B C
Theirs	ABC
Result	A B B C C

Table C.11

Should merge edits form an empty base 2.

Base	
Ours	`<p>A</p><ul><li>B</li><li>C</li><li>D</li></ul>`
Theirs	`<p>A</p><ul><li>B</li></ul>`
Result	`<p>A </p><ul><li>B </li><li>C </li><li>D </li></ul>`

Table C.12

Should edits from an empty base 3.

C.4 Wiki Benchmark Test CPU-Usage

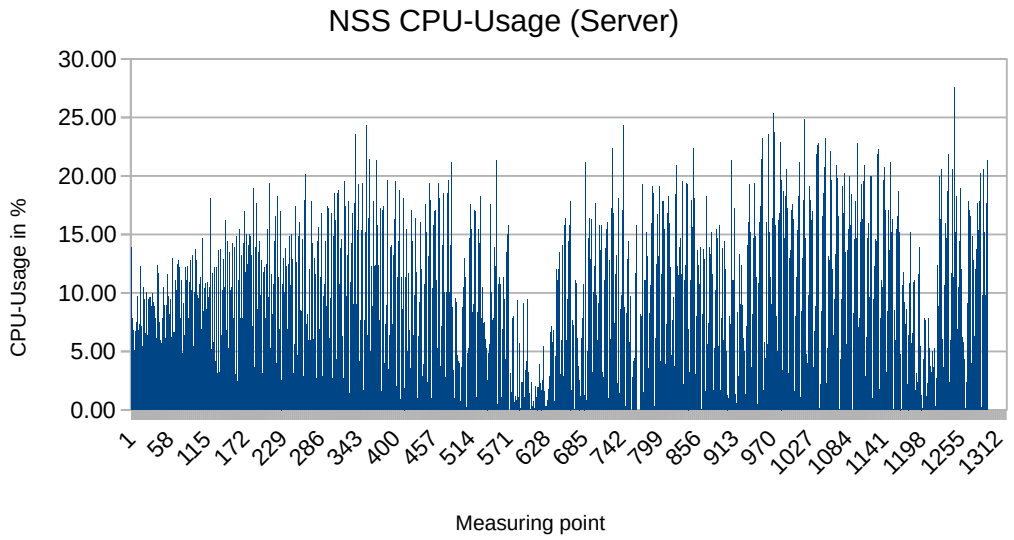

Figure C.1. Server CPU Usage Monitoring

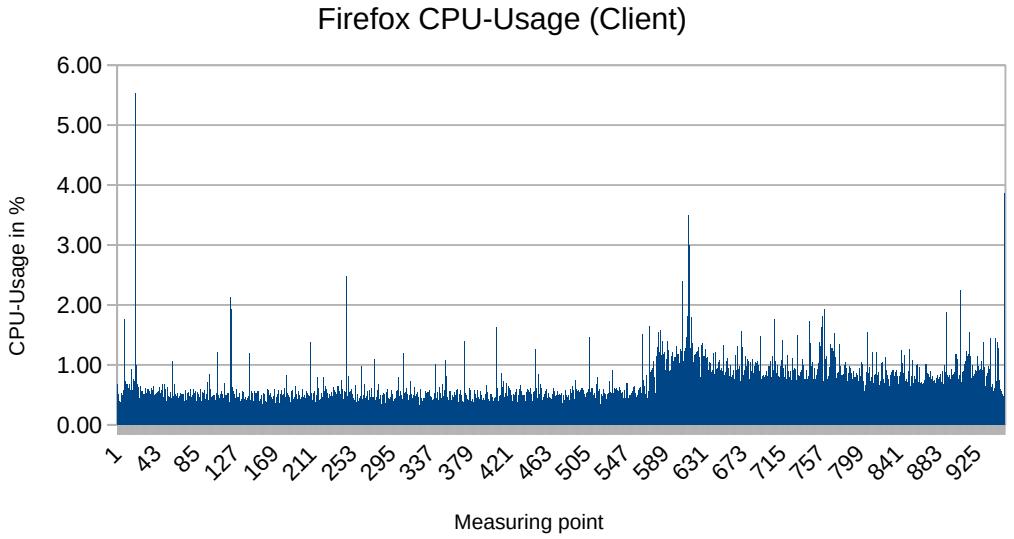

Figure C.2. Client CPU Usage Monitoring

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Fabrizio Parrillo

Matriculation number — Matrikelnummer
13-276-985

Title of work — Titel der Arbeit
A Decentralized Wiki based on the
Solid Ecosystem

Type of work — Typ der Arbeit
Master Thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 22 July 2020