A Demonstration of the Solid Platform for Social Web Applications

Essam Mansour[1] Andrei Vlad Sambra[2] Sandro Hawke[2] Maged Zereba[1]
Sarven Capadisli[2] Abdurrahman Ghanem[1] Ashraf Aboulnaga[1] Tim Berners-Lee[2]
[1]Qatar Computing Research Institute, HBKU
[2]Decentralized Information Group, MIT CSAIL

ABSTRACT

Solid is a decentralized platform for social Web applications. In the Solid platform, users' data is managed independently of the applications that create and consume this data. Each user stores their data in a Web-accessible personal online datastore (or pod). Each user can have one or more pods from different pod providers, and can easily switch between providers. Applications access data in users' pods using well defined protocols, and a decentralized authentication and access control mechanism guarantees the privacy of the data. In this decentralized architecture, applications can operate on users' data wherever it is stored. Users control access to their data, and have the option to switch between applications at any time. We will demonstrate the utility of Solid and how it is experienced from the point of view of end users and application developers. For this, we will use a set of Solid servers and multiple Web applications that use these servers. We believe that experience with a concrete platform such as Solid is highly valuable in truly appreciating the power of a decentralized social Web.

1. INTRODUCTION

Social Web applications, such as Facebook, Twitter, Doodle, Wikipedia, Craigslist, and many more store data in centralized repositories that can be thought of as "data silos". Each application (or set of applications based on one social network platform) controls its own data and often has its own authentication and access control mechanisms. As a result, users cannot easily switch between similar applications that could allow reuse of their data, or switch from one data storage service to a different one. Developers are restricted to the data access APIs provided by a specific platform, and cannot easily develop applications that can run on multiple platforms. These and other problems of centralization have been recognized for a long time, and there have been many proposals for "re-decentralizing" the social Web such as Diaspora[1], Musubi [3], and WebBox [10], among others. None of these proposals has been widely adopted yet, and the technical details of the decentralization platform are still a topic of investigation among researchers and practitioners. For example, W3C has Social Web Working Group actively investigating decentralization standards[2].

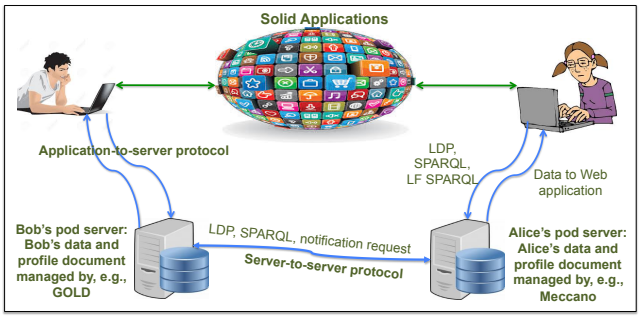

Figure 1: The Solid platform. A user stores their data in a personal online datastore (pod) that resides on a pod server. The user controls their identity using an RDF profile document stored in their pod. To use a Solid application, the user loads the application from an application provider. The application obtains the user's pod from the identity profile. It then follows links from the profile to discover data on the user's pod, as well as on other pods, performing authentication when needed.

Our current activity in this space centers around a platform that we call *Solid*[3] (for *Social Linked Data*) [7]. The Solid platform supports decentralized social Web applications, relying as much as possible on W3C standards and Semantic Web technologies to realize the architecture shown in Figure 1. The platform specifies all the protocols required in the figure, such as authentication, application-to-server and server-to-server communications. This demonstration showcases the Solid platform, focusing on the experience of the end user and the application developer. In particular, we demonstrate through a set of applications and servers supporting these applications that Solid enables a high degree of interoperability between applications, easy sharing of data and the social graph between applications, and portability of data between servers. These features are demonstrated through standard applications such as maintaining the list of contacts of a user, and more novel applications enabled by Solid such as collaborative authoring of scholarly articles.

We are developing the Solid platform as part of our Crosscloud project[4], which aims to address the research challenges

[1]https://diasporafoundation.org

[2]https://www.w3.org/Social/WG
[3]https://github.com/solid/solid
[4]http://crosscloud.org

WWW'16 Companion, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
http://dx.doi.org/10.1145/2872518.2890529.

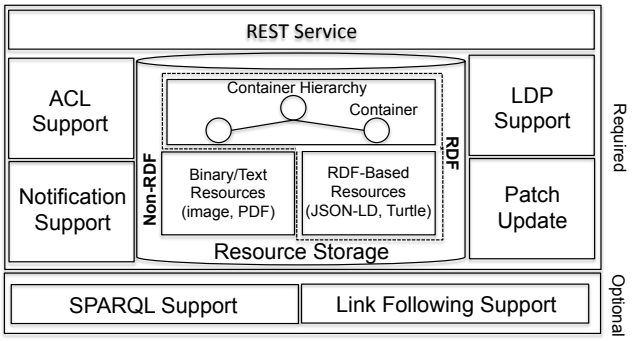

Figure 2: Overview of a pod server. A pod stores RDF and non-RDF resources. The server support LDP, patching resources, access control, live updates, and optionally SPARQL.

related to building a decentralized social Web. A good choice of protocols on the wire is essential to Crosscloud, and Solid provides such protocols. In addition to designing protocols, the Crosscloud project must address several research questions. For example, what are the data models and design patterns that applications should use to store data? How can we ensure that applications agree on a vocabulary for the concepts that they use, and how to integrate data from different applications when needed? What is the best way to support notifications from application to application? An interesting topic of investigation is the extent to which Web traversal and complex data retrieval can be offloaded from client to server. Supporting application developers is also important to build momentum around the Crosscloud ecosystem. In addition, suitable models for security and privacy are essential for the social Web, and decentralization makes the questions around these models more complicated [1, 9]. This demonstration of Solid provides a framework for appreciating these questions.

The rest of this document is organized as follows. We present a brief overview of the Solid platform in Section 2. Section 3 then discusses application development in Solid and presents some of the Solid applications that will be used in the demonstration. In Section 4, we describe some possible demonstration scenarios. Section 5 concludes.

2. OVERVIEW OF SOLID

In the Solid platform, each user stores their data in a Web-accessible *personal on-line datastore* (or *pod*). Applications run as client-side Web applications in a browser or as mobile applications. These applications use an authentication protocol to discover the user's identity and profile data, as well as relevant links that point to the user's pod, which contains application data. Solid supports decentralized authentication and access control, and it also supports standardized data access mechanisms. We describe these two aspects next.

Decentralized authentication, a global ID space, and global single sign-on are a critical part of the Solid ecosystem. Solid uses WebID [8] to provide these features, although other solutions exist and can potentially interoperate with Solid. In Solid, a user has to register with an identity provider, and this identity provider stores the user's WebID profile document associated with a cryptographic key. In most cases, a pod provider would also operate as an identity provider, offering WebID "accounts" to its users.

Table 1: Pod servers. databox.me, meccano.io, and rww.io act as public pod servers as well as identity providers, allowing users to create WebIDs.

Name	Platform	Running Service
gold	golang	*https://databox.me/*
meccano	Java+Jena	*https://meccano.io/*
ldphp	PHP	*https://rww.io/*
ldnode	node.js	not public

Application data in Solid is stored in users' pods and pods are stored on *pod servers*. Data is managed in a RESTful way, as defined by the Linked Data Platform (LDP) recommendation [6]. LDP enables applications to manage data items within hierarchical containers (which can also be called collections or directories). Each data item and container has a URI, and LDP defines the protocol for manipulating the data items and containers through HTTP requests on their URIs; for example, POST/PUT to create, PUT/PATCH to update, and GET to retrieve. Items can be found through their URIs, or by following links from other items. Solid distinguishes between structured data, which is represented in Solid using RDF [5], and unstructured data that can be of any type, e.g., videos, images, Web pages. This allows structured data to be parsed and serialized in various formats such as Turtle or JSON-LD.

Additional to LDP support, pod servers may offer optional SPARQL support. Servers that support SPARQL allow applications to express complex data retrieval operations, including operations that require server-to-server communication via link-following SPARQL. This simplifies Solid application development, since it enables a developer to delegate complex, multi-pod data retrieval operations to the server.

Pod servers in Solid are application-agnostic, so that new applications can be developed without having to modify the servers. For example, even though LDP 1.0 contains nothing specific to "social", many of the W3C Social Web Working Group user stories[5] can be implemented in Solid, using only LDP and application logic, with no changes to the server.

The requirements of a pod server are illustrated in Figure 2. A pod server needs to store RDF and non-RDF resources, and it needs to support basic LDP access to these resources, patching resources, access control lists (ACLs), live updates, and optionally SPARQL. There are several ways in which the underlying storage for RDF data can be implemented in a pod server, e.g., using the file system, a key-value store, a relational database system, or a graph database system (i.e., a triple/quad store).

We have implemented several prototype servers, listed in Table 1. Our **ldphp**[6], **gold**[7], and **ldnode**[8] servers store all their data in the file system. In this case, both RDF and non-RDF resources are stored as files, including the RDF resources representing ACLs and the metadata documents corresponding to non-RDF resources (all of which are defined by LDP). Our **meccano** server stores RDF data in a graph database system (currently we use Jena[9]), and it stores non-RDF data in the file system. **Meccano** implements all Solid operations via SPARQL queries, and it also implements complex data retrieval using link-following

[5]http://www.w3.org/wiki/Socialwg/Social_API/User_stories
[6]https://github.com/linkeddata/ldphp
[7]https://github.com/linkeddata/gold
[8]https://github.com/linkeddata/ldnode
[9]http://jena.apache.org

SPARQL. Using an RDF database simplifies querying large data sets, efficient data retrieval (i.e., subsets of graphs), as well as patch operations.

3. APPLICATION DEVELOPMENT

In this section, we discuss application development in Solid and give examples of the Solid applications that we have implemented. The intention is to demonstrate the flexibility of the underlying architecture of Solid and the benefits of decentralization.

Solid application development is supported by a set of libraries and components. For example, all the applications that we have developed use the rdflib.js library (the core library from Tabulator [2]) to handle RDF resources. Another library is solid.js[10], which simplifies the development of Solid applications by abstracting some of the more complex operations. We have also provided modules for authentication[11] and signup[12] that are designed for reuse as Web Components [4]. We are continuously growing the set of libraries and components in the Solid ecosystem, and we expect this to significantly accelerate the adoption of Solid.

We have developed several Solid applications for common day-to-day tasks, listed in Table 2. Some of these applications use the AngularJS and jQuery frameworks, which provides a proven set of features in terms of application interactivity. For this demonstration, all applications were developed as responsive Web applications and tested in Firefox and Chrome. Screenshots of the **contacts** and **dokieli** applications are shown in Figures 3 and 4. We describe these two applications next.

The **contacts** application manages a list of contacts stored on a user's pod. In Solid, a user's social graph is made up of the contacts stored on their pod, the contacts of these contacts, and so on, where each user is identified by a WebID. Thus, **contacts** can be viewed as an interface for managing a user's distributed social graph. The **contacts** application maintains a set of vCards for a user's contacts using the vCard ontology[13]. Each vCard is a resource with a unique URI, and can contain the WebID of the user that it represents in addition to other fields such as name and e-mail. A user may mark a vCard as public or allow a vCard to be accessed by an individual or a group of people (identified by WebIDs).

One of the interesting social features in our **contacts** application, enabled by Solid, is the ability to search in the "contacts of your contacts" using link-following SPARQL. A user can search in their pod for a vCard matching search criteria such as name, email, or address. In addition, the **contacts** application can use a link-following SPARQL query to search for a contact in the public contacts on pods that can be reached from WebIDs in the user's vCards (via link following). The user gets a list of vCards matching the search criteria, and the URI of each answer vCard indicates the source of this card. This search capability provides an example of the innovative social features supported by a decentralized social platform such as Solid.

dokieli is a decentralized article authoring, annotation, and social notifications application[14]. While it is a general purpose tool to write and manage articles, it is compliant

[10]https://github.com/solid/solid.js
[11]https://github.com/linkeddata/webid-login
[12]https://github.com/linkeddata/webid-signup
[13]http://www.w3.org/2006/vCard
[14]https://github.com/linkeddata/dokieli

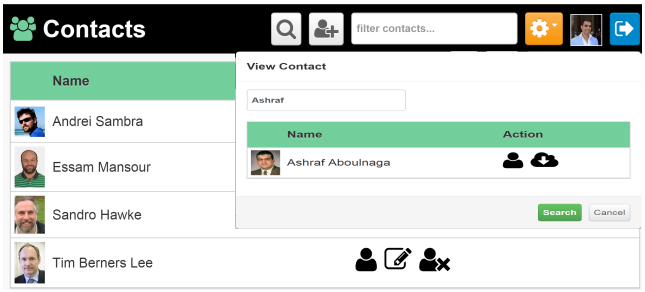

Figure 3: The contacts application maintains a set of vCards using LDP and uses link-following SPARQL to search for vCards in the user's pod and pods reached from WebIDs in these vCards.

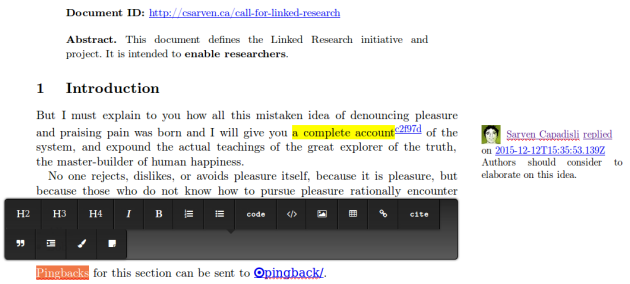

Figure 4: dokieli is a general purpose authoring and interaction tool among different pods.

with the Linked Research initiative[15], and provides social features and interactions for scholarly communication. Articles can also be semantically annotated, anywhere from a fragment in a sentence to descriptions, e.g., hypothesis, workflows, evaluations, and have their own URIs to foster discovery and reuse. All URIs can be dereferenced and have their content representations in HTML and RDF.

dokieli employs the WebIDs and pods of authors, reviewers, and commenters for instance to store the information created by these participants and can assign them different access controls. For example, annotations and social notifications like replies, peer-reviews, liking, resharing, can reside in the contributor's pod.

Our experience with Solid application development confirms that Solid provides a feature-rich platform that supports portability and interoperability. Applications can work with multiple pod server implementations, and it is easy to change the applications that use data without changing the data (e.g., by forking and adding features).

Implementing social features in Solid applications is simple and requires a thin layer of reusable code. Applications and pods interact with each other by taking advantage of the HTTP/1.1 methods, which do the heavy-lifting.

Using SPARQL in Solid applications is also relatively simple. Developers can write SPARQL queries to express data access features such as search, filtering, or fetching top results. Developers can also write link-following SPARQL queries to follow links between pods.

4. DEMONSTRATION PLAN

Demonstration participants will be able to interact with all the applications shown in Table 2, and will be able to

[15]https://github.com/csarven/linked-research

Table 2: Solid applications. An asterisk (*) indicates a third-party application, not developed by us.

Name	Function	Usable At
contacts	Manage a list of contacts	*http://mzereba.github.io/contacts*
contacts	Manage a list of contacts	*http://linkeddata.github.io/contacts*
calendar	Event manager	*http://mzereba.github.io/calendar*
dokieli	Decentralized authoring, annotation, and social notifications	*https://dokie.li*
pad	Shared collaborative editing	*https://github.com/timbl/pad*
profile-editor	View and update a user's profile	*http://linkeddata.github.io/profile-editor*
warp	Solid file browser	*http://linkeddata.github.io/warp*
cimba	Microblogging (cf. Twitter)	*http://cimba.co*
zagel	Instant messaging/group chat	*https://solid.github.io/solid-zagel*
***webid.im**	Instant messaging/chat	*http://webid.im*
***shamblokus**	Strategy game (cf. Blokus)	*http://deiu.github.io/Shamblokus*

store data on two different pod servers: databox.me and meccano.io. This section provides a specific demonstration scenario using these servers.

The demonstration scenario involve two users, Alice and Bob, using different pod servers. Alice will use the **gold** server at databox.me, and Bob will use the **meccano** server at meccano.io. We will show that although these are two totally different servers, both users can use the same applications to access and maintain their data. This can be shown using any of the applications in Table 2. An application will be able to create, modify, delete, and retrieve resources in the user's pod. Demonstration participants can view these resources using the **warp** file browser, and can also see the client-server interaction involved.

Besides the basic Solid functionality, the demonstration will turn to interoperability and access control. Interoperability will be demonstrated through the **dokieli** application enabling social interactions among users, and through applications using link-following queries. For example, we will demonstrate how Alice can use link-following queries in the **contacts** application to search in the public contacts of Bob. In addition to demonstrating interoperability, these examples will also demonstrate access control. They will also demonstrate other featured of Solid such as delegation[16] in order to allow a pod to speak on behalf of its owner. As before, demonstration participants can view the resources being created, observe client-server interactions, and also server-to-server interactions.

Another form of interoperability is having multiple applications use the same data. We will show that a user can use two different **contacts** applications to manage the same set of contacts. We will also demonstrate the portability provided by Solid by showing how Alice can easily migrate her pod from databox.me to meccano.io. After this migration, Alice needs to change her WebID profile to point to the new storage, and her applications will be redirected to the new pod.

5. CONCLUSION

Re-decentralizing the social Web is an important topic and an active area of research. The Solid platform is a concrete instance of a decentralized platform for social Web applications, providing decentralized authentication, decentralized data management, developer support in the form of libraries and web components, and a suite of running servers and example applications. This demonstration will show how the Solid platform can enable social applications while allowing each user to retain control of their pod. Demonstration participants will experience Solid from a user and application developer perspective. They will gain insights into the interoperability and portability features provided by Solid, the rich social features that it can enable, and the client and server machinery behind these features. A concrete appreciation of such a platform is very valuable in the ongoing discussion on re-decentralization.

[16]https://github.com/solid/solid-spec#webid-delegated-requests

6. REFERENCES

[1] L. M. Aiello and G. Ruffo. LotusNet: Tunable privacy for distributed online social network services. *Computer Communications*, 35(1), 2012.

[2] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic Web. In *Proc. Int. Semantic Web User Interaction*, 2006.

[3] B. Dodson, I. Vo, T. J. Purtell, A. Cannon, and M. S. Lam. Musubi: Disintermediated interctive social feeds for mobile devices. In *Proc. World Wide Web Conf. (WWW)*, pages 211–220, 2012.

[4] D. Glazkov and H. Ito. Introduction to Web components. *W3C Working Group Note*, 14, 2014.

[5] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. 2006.

[6] A. Malhotra, J. Arwe, and S. Speicher. Linked Data Platform Specification. W3C Recommendation, 2015. http://www.w3.org/TR/ldp/.

[7] A. Sambra, A. Guy, S. Capadisli, and N. Greco. Building decentralized applications for the social Web. Tutorial at the World Wide Web Conf. (WWW), 2016.

[8] A. V. Sambra, H. Story, and T. Berners-Lee. WebID Specification. 2014. http://www.w3.org/2005/Incubator/webid/spec/identity/.

[9] S. Schulz and T. Strufe. d2 deleting Diaspora: Practical attacks for profile discovery and deletion. In *Proc. IEEE Int. Conf. on Communications (ICC)*, 2013.

[10] M. Van Kleek, D. A. Smith, N. R. Shadbolt, and mc schraefel. A decentralized architecture for consolidating personal information ecosystems: The WebBox. In *Proc. Workshop on Personal Information Management (PIM)*, 2012.