WebBox: Supporting Decentralised and Privacy-respecting Micro-sharing with Existing Web Standards

Conference Paper · January 2012

CITATIONS
3

READS
151

8 authors, including:

Daniel A. Smith
University of Southampton
75 PUBLICATIONS **530** CITATIONS
SEE PROFILE

Max Van Kleek
University of Oxford
125 PUBLICATIONS **1,103** CITATIONS
SEE PROFILE

Oshani Seneviratne
Rensselaer Polytechnic Institute
27 PUBLICATIONS **247** CITATIONS
SEE PROFILE

m.c. Schraefel
University of Southampton
286 PUBLICATIONS **3,405** CITATIONS
SEE PROFILE

WebBox: Supporting Decentralised and Privacy-respecting Micro-sharing with Existing Web Standards

Daniel A. Smith[1], Max Van Kleek[1], Oshani Seneviratne[2], mc schraefel[1], Alexandre Bertails[2], Tim Berners-Lee[2], Wendy Hall[1], Nigel Shadbolt[1]

[1]Electronics and Computer Science
University of Southampton
Southampton, UK
{ds, emax, mc, wh, nrs}@ecs.soton.ac.uk

[2]CSAIL / W3C
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
{oshani, bertails, timbl}@csail.mit.edu

ABSTRACT

The popularity and plurality of social sharing and collaboration platforms today demonstrates the demand for sharing information at a wide variety of granularities and scale — from public blogging of micro-data (e.g., microblogging posts, page "Likes", and place-"check-ins") to specific, full-on collaborative sharing of documents and projects. The centralised and fragmented nature of today's sharing platforms, however, creates several problems, among which include issues of seamless access, ownership, and control. In this paper, we present an alternative approach that puts the user back in control over their data, and supports secure, private sharing and collaboration in a distributed, user-centric manner "out of the box". This approach represents a novel configuration of Web-standard, non-novel components, which we demonstrate are sufficiently generic to support types of sharing offered by the most popular sharing platforms today. By offering a standards-compliant method of globally sharing triples, with known semantics about how they are received, our approach enables social applications to utilise the Web as the distributed system it has always promised to be. Beyond such sharing, we discuss what other capabilities such a platform enables, and its implication for future Web architectures and tools.

1. INTRODUCTION

From the outset, the Web has been designed as a decentralised information-sharing medium in which anyone, anywhere, could share any information simply by creating a document on a Web server[12]. In the two decades since its creation, however, the ways that people use the Web to share information has experienced at least two significant changes. The first has been the rise of "micro-sharing" — the propagation and dissemination of information smaller than documents — including blog posts, Tweets (microblog entries), status updates, comments/responses, and notifications, such as Foursquare "check-ins"[1], or Facebook "Likes" [2]. The second has been in how this information is shared; instead of being disseminated in a purely decentralised way as the Web was originally designed, much of this sharing is done at a handful of massive, centralised, domain-and-app-specific sharing sites, such as Twitter (for microblogging), Facebook (for social network updates), and Delicious (for social bookmarking). These two shifts are not unrelated; the desire of share bits of information sparked apps like Twitter and Delicious into existence; the ultimate desire to share this kind of information drove the rise of many of these services.

Why should such centralisation be of concern? First, competition and lack of integration among these services has meant that we have to fragment our data into each of these "silos" (or "walled gardens"[27]) to use them. In so doing, we have to comply with the requirements of each service, including re-shaping the data to fit each particular platform's representations, and agreeing to its terms of service. Once our data has been handed over, we have little control over how it is subsequently handled: e.g., how it is accessed and visualised, how or where it is stored, and with whom it is shared or disclosed.

The implications of centralisation extend beyond issues of flexbility, ownership and control. Availability can potentially suffer, as connectivity problems and "cloud outages" can affect our ability to access our own, potentially vital data. Privacy issues also abound: centralised services have unprecedented views of our activities based on pattern(s) of data access and content shared. For free cloud services, this data is essentially capital that, through data mining and advertisement, generates the revenue necessary to sustain the platform. While a viable business model, the lack of alternatives has meant that we essentially don't have a choice: we either use these services (and be "mined") or give up the social sharing of data altogether.

Despite these trade-offs, concerns around these limitations have largely been secondary to the immediate benefit received from social information sharing, a capability so important and transformative that, despite occasional reactions to Facebook privacy violations [28] we tend to overlook the long term issues arising from the use of such services.

There are, without question, advantages to the user experience afforded by centralised social sharing platforms: since

[1]Foursquare — `http://www.foursquare.com`
[2]Facebook "Like" — `https://developers.facebook.com/docs/reference/plugins/like/`

WWW2012 Lyon, France

all content is displayed together new content is easily discovered; moreover, it is simple to cross-reference multiple streams. We propose, however, that the perception that one *must* have a centralised service to deliver these user experiences is an artificial one.

To demonstrate that a decentralised social data sharing approach that is practical, straightforward and readily assembled today from off-the-shelf protocols/components, we present WebBox. WebBox is a platform that allows social, collaborative applications to be built entirely using both standard Web and Semantic Web protocols and components: a standard RDF triple store, a SPARQL endpoint, HTTP server, WebACL and WebID. With these components, WebBox supports secure sharing and collaboration over structured data objects in a schema-agnostic manner, enabling applications spanning blogging, to lifelogging, datablogging, and collaborative note-taking.

In the following sections, we describe how we derived the requirements for WebBox by analysing the functionality offered by the most popular sharing services. We then describe the architecture of WebBox itself, including its components, and ways that WebBox-compatible applications run, access data, and communicate with other WebBoxes. We address scalability considerations of the system, and techniques to mitigate update traffic. Next, to demonstrate the simplicity with which WebBox-based sharing applications can be built, we walk through SocialBox, a WebBox application that provides sharing capabilities spanning blogging, microblogging, Web page annotation, social news aggregation, note-taking, and real-time collaborative authoring and revision. We conclude the paper with a discussion of current limitations, and future work opportunities for this extensible platform.

2. REQUIREMENTS ANALYSIS

In order to determine the set of capabilities needed of our WebBox architecture, we performed a pre-study of the most popular existing social sharing platforms to characterise the basic data handling and sharing functionality these platforms offer. We then derived functionality categories by an open coding approach borrowed from social science [31]. We first studied each site, and derived descriptions of each site, either from direct experience in using the site, or based upon a description obtained from the site or Wikipedia. Then we listed each site's basic capabilities. These capabilities were then arranged by similarity, and those that were most similar were combined. The result of this process yielded the following eight sharing capabilities:

2.1 Pre-study: Identifying Common Capabilities of Popular Sharing Sites

1. **Hosting of Content/Resources** - Support for sharing by directly hosting content (text, blog entries, news articles, images, audio/video, etc)

2. **Comments/Annotation** - Support for the creation and sharing of comments or annotations (textual or structured), either stand-alone or linked to a particular resource

3. **Link Sharing** - Support for the sharing of links to Web resources.

4. **Re-Sharing/Re-Linking** - Support for the re-presentation of a resource hosted elsewhere with a new URI to serve as a new local anchor point for annotations

5. **Collaborative Editing** - Allowing resources to be updated after being created, specifically by more than one user (potentially simultaneously).

6. **Access Control** - Support for restricting access of shared resources to a 1) group 2) to individuals

7. **Survey/Solicitation** - Support for polls, forms and other methods of soliciting information from others.

8. **Data blogging/sharing** - Support for the sharing of data sets, such as those from automatically captured activities (e.g., location patterns, physical exercise logs).

Table 1 illustrates each of the sharing platforms we examined and the capabilities they provide.

While this initial analysis allowed us to identify the specific functionality that people rely upon today, it also made apparent each platform's limitations. First, all services were extremely specific about the kinds of data that could be hosted and shared. For example, YouTube and Vimeo only host for video content, while other sites host specific structured data, such as Foursquare's "check-ins", Delicious bookmarks, and textual "tweets" on Twitter. Even the "big" social networks restrict what kinds of content can be posted — a fixed set of status update types comprising textual status updates, photos, videos, generic Web links, and links to specific external photos/video/audio content hosts. To support new applications, we relaxed this constraint so that, in WebBox, hosted and shared resources could represent any structured or unstructured data objects.

The second capability that distinguished these sites is the ways in which they allowed users to control scope of sharing — from the individual controls of Google Docs, to Diaspora's groups or *aspects*, to not having any controls at all, in the case of various blogging platforms. Since different access control schemes are appropriate in different situations, we sought to support all such schemes.

The third significant limitation was manifest in the fact that, with the exception of Google Docs and Etherpad, most of the data hosted by such sites were 'read-only' once-posted. Since collaborative authoring could be a useful feature regardless of the artifact, we made this capability the third basic requirement of WebBox.

Generalizing and relaxing the above constraints in the ways mentioned resulted in the following three functional capabilities we sought for WebBox:

- **Content Hosting** - The storing and hosting of arbitrary structured and unstructured data
- **Access Control** - Flexible access control, supporting multiple granularities of scope (e.g., private, individuals, groups, public)
- **Sharing and Collaborative Editing** - (in conjunction with **Content Hosting**, enables multi-user editing, comments/annotation, link sharing, concurrent editing)

The following sections describe how these capabilities compare with those provided by other frameworks, and detail how they are realised in WebBox.

app type	example sites	capabilities							
		hosting	comments / annotation	link sharing	re-sharing	group / individual access control	surveys / polling	data - blogging	multi-user editing
Video and photo sharing sites	Flickr / Youtube	yes	yes			no / yes			
Social networking sites	Facebook/Google+	yes	yes	yes	yes	yes / yes			
Blogging sites	blogger.com, Wordpress.org, Livejournal	yes	yes		yes	some / some			
Social news site	Reddit, Digg, Hacker News, Slashdot		yes	yes		no / no	yes		
Social bookmarking	del.icio.us			yes		no / no			
Microblogging	Twitter / identi.ca	yes	yes	yes		no / no			
Tumblelogging (Microcontent sharing)	Tumblr / Posterous	yes	yes	yes	yes	no / yes			
In-page recommendation	"Like" button (fb) / tweet this		yes			no / no			
Polling	doodle/formspring					no / yes	yes		
Location sharing	Google Latitude / Firebird / Plazes					no / no		yes	
Personal data logging	Nike+, GetFit		yes			no / yes		yes	
Media consumption tracking	Last.fm (audio tracking) , scribd		yes	yes		no / no		yes	
Note taking	SubehaEdit, Etherpad-likes (Piratepad), Webnotes					no / yes			yes
Document authoring	Google docs, ZOHO notebook					no / yes			yes
Web site annotation	Google Sidewiki, Diigo, Stickis, Webnotes		yes			no / yes			yes

Table 1: Popular sharing platforms on the Web, and basic data handling and sharing capabilities they support.

3. RELATED WORK

Growing concerns about individual privacy and ownership of personal information in the face of the ever-increasing amount of information we share online has yielded number of efforts to come up with solutions. The first concerns systems to facilitate the decentralised storage and keeping of personal data under the user's control. The ProjectVRM group at Harvard [3] has studied the feasibility of such approaches from socio-technical and economic perspectives. Their analysis and continued discussion [2] have shown that the decentralisation of personal information, driven by cost-incentives for corporations and need to comply with data protection policy (such as with the EU Data Protection Directive 94/95/96[18]) have indicated a significant shift towards greater personal-ownership of data in the next decade.

Implementations of the VRM approach, however, are still in early stages. Mydex [9], a Community Interest Company of the UK government, has produced a proof-of-concept personal data store based on the Java platform called Higgins [5]. Similar open source personal data storage containers include The Locker Project [22], data.fm [24], Owncloud [7], and OpenStack [1], which each all provide various degrees of easy-to-set-up "personal cloud" software that can be used to store and host content on the user's own server on the Web.

Although built with the same goal, these platforms exhibit several differences. Higgins and Openstack are generic schema-agnostic data containers that provide simple storage and retrieval APIs for this data, typically via a REST-style API. The Locker Project, Owncloud, and Diaspora, meanwhile are social-network inspired and centered around a fixed set of simple data types, such as hosting files, status messages, photos, files, and calendar events. The data.fm project provides a platform to create linked data in a generic read/write Web style. Opera Unite [6] released with Opera 10 in 2009 goes a step further by giving users access to a 'Web server' within their Web browser, which can be used to share media with other users seamlessly.

While we considered basing WebBox on one of these platforms, these platforms were either insufficiently schema-agnostic to be an application platform (as in Diaspora or the other social-network oriented platforms), or seemed not to address needs concerning sharing i.e. pertaining to keeping subscribers notified of changes. Access control was also done in ad-hoc, non-standards compliant ways and there seem to be some security lapses (as in Opera Unite where it exposes the file system to outside apps).

Another desideratum for a Web sharing architecture is that it provides seamless integration of data that is available in a decentralised setting. The Nepomuk Semantic Desktop project [15] has created a group collaboration architecture using Semantic Web technologies and peer-to-peer networks. Although Nepomuk is ontology-driven and supports group collaboration, it is geared towards the desktop experience rather than a cloud-based Web experience. As desktop applications are becoming increasingly obsolete, we believe that the concept of WebBox is a step in the right direction in providing a universal platform for collaboration. PrPl [26] describes a decentralised social networking infrastructure that allows users to share personal data in

[3]ProjectVRM — `http://cyber.law.harvard.edu/projectvrm/Main_Page`

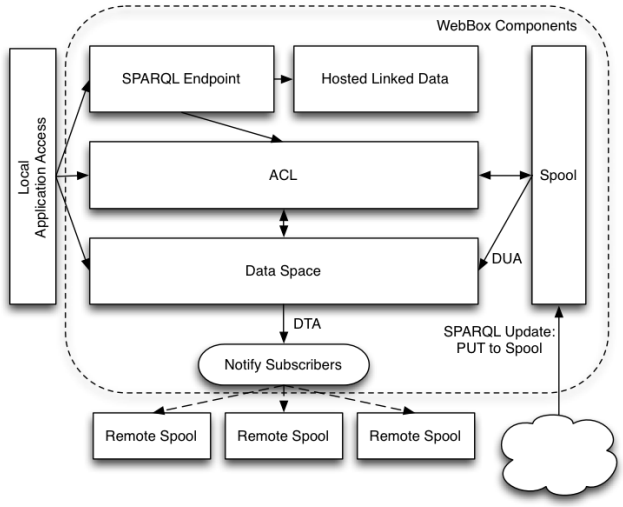

Figure 1: WebBox Components and Data Flow.

a peer-to-peer network through intermediaries called 'butlers'. While similar to our concept, PrPl requires applications to be written using a specialised language, whereas our solution is language agnostic and only depend on open Web standards.

This inspired us to create a standards-based, appli-generic, schema-agnostic secure platform for structured data sharing with a focus on scalability and usability. The above is a review of related work pertaining to our overall goal and approach; in the following sections we reference further related work in context as it informs the specific related components.

4. ARCHITECTURE AND DESIGN

In this section, we provide a detailed specification of the WebBox architecture, including its components, messaging, authentication and access controls, and describe how applications can use standard protocols to interact with WebBox. We subsequently describe how we implemented a prototype WebBox using several popular open-source servers and tools.

4.1 Architectural Overview

Unlike the current infrastructure of the Web, where we use browsers to access content served up by Web servers over which we have little control, WebBox makes the assumption that users will have our own WebBoxes, a data store and HTTP server which serves to host and store their data, and to receive data from collaborators. Their WebBox can live on the user's own device(s), or on a virtual host on some hosting provider.

WebBox is made up of a number of software components, as shown in Figure 1:

1. **Data Space** Knowledge base of all received messages and the external linked data they reference. Maintains lists of who is subscribed to the published entities.
2. **ACL** Maintains permissions for endpoints and published entities.
3. **SPARQL Endpoint** Endpoint to publish new entities, and to modify existing ones. ACL determines access, which will typically be to local users only. The SPARQL endpoint also provides a knowledge base of all published entities.
4. **Hosted Linked Data** RDF descriptions of entities that have been created by users of the system. They are retrievable using HTTP GET, read-only access, and this access is determined by the ACL.
5. **Spool** Endpoint to allow messages to be sent to the system. When they are received, they are put into the local messaging knowledge base, as well as any external linked data that has to be resolved.
6. **Applications** Local applications access the data via the SPARQL endpoint, where they can be queried for existing data, send data to remote endpoints, and publish new data.
7. **Data User Agent (DUA)** Takes messages from the spool, analyses who put them there against their permissions in the ACL, and takes appropriate actions. For example, revolving external data entities and storing them in the data space.
8. **Data Transfer Agent (DTA)** Responsible for notifying subscribers to data when that data has changed, or when new data has been published. Uses HTTP PUT to add data to remote WebBox SPARQL Update endpoints.

In the following section we detail the necessary components that are important for the operation of the system outlined above.

4.2 Messaging

When a user updates or creates a new entity in their WebBox, WebBox takes care of propagating the notification of this update to the WebBoxes of people that is has been shared with. WebBox achieves this by using a publicly writable URL that conforms to the SPARQL Update specification. Therefore, anyone can PUT a new file of RDF there, which is then processed by the user that owns the "inbox", or by software on their behalf.

The WebBox is used to send RDF metadata to a specific user. This mechanism can be used by any application with any ontologies, and therefore provides a rich mechanism to share structured semantic data with anybody. Users can identify their WebBox using the predicate <`webbox:address`> [4] to point to the URL of their WebBox. Users can run their own WebBoxes, or use a service that will do it for them (in a similar way to e-mail, some people run their own, or more typically rely on organisations to do this for them). The following section outlines the messages used in our system:

1. **Create Message**: A local application sends an HTTP PUT (corresponding to SPARQL 1.1 Graph Store HTTP Protocol [23]), to upload an RDF file to the SPARQL endpoint. The location and filename used in the PUT request determine the location of the file when hosted as linked data.

[4]In this section we refer to URIs using the `webbox` prefix, which has the full namespace of `http://webbox.ecs.soton.ac.uk/ns#`. Thus, `webbox:address`, for example, has the URI: `http://webbox.ecs.soton.ac.uk/ns#address`

2. **Modify Message**: A local applications uses the SPARQL Update 1.1 to "INSERT DATA" and "DELETE DATA" in order to modify the data in the graph. These commands are directed at the URI of the linked data itself.

3. **Read Message**: An HTTP GET with a "Content-accept" header including "application/rdf+xml", according to RDF publishing best practices [13]. Additionally WebBoxes offer a SPARQL query endpoint over a graph of all publicly hosted linked data, in order to allow for knowledge discovery against query patterns (e.g., to discover all resources published of a particular `rdf:type`).

4. **Share Message**: In order to share a resource with someone, a single triple is HTTP PUT to their WebBox (corresponding to SPARQL 1.1 Graph Store HTTP Protocol [23]). The purpose of the triple is to identify the resource that is being shared, and the user it is being shared with, using the predicate `sioc:addressed_to`. For example, to share a blog post with a colleague you might put the following triple to their WebBox:

   ```
   <http://hip.cat/posts/34> sioc:addressed_to
   <http://danielsmith.eu/#dan>
   ```

   The triple is serialised into a single RDF file, which has a GUID as the filename, and PUT to the WebBox. We suggest the use of a GUID to avoid clashes with existing files, which would result in a failed upload. Alternatively SPARQL 1.1 Update can also be used, using the "INSERT DATA" action to insert a new graph of triples.

   It is the the responsibility of the receiver to dereference and resolve the data of that URI, from its source, if they wish. We suggest this method of minimal posting in order to give the choice of whether to receive the entity to the receiving user, and also to minimise traffic sent to remote hosts, particularly when multiple WebBoxes are being uploaded to. It is important to also update the ACL to give them read access, and optionally, write access to that item, if it is hosted locally. This can be done by adding new metadata, using the WebACL ontology, into the local sparql endpoint. For an example, see Section 4.4.

5. **Subscribe Message**: When an entity is updated, remote parties with whom the entity is shared may need to be kept informed – for example, so that other users will see changes to notes, annotations, etc. The aforementioned SPARQL-update based "messaging mechanism" described above for notifying remote parties about newly shared entities is also used to announce when updates have occurred to entities, so that remote clients can re-request the entity as necessary.

   To subscribe to the updates of a URI, a user adds a triple to the public WebBox endpoint, and authenticates with their WebID in order to confirm that they are subscribing themselves and not a third-party. Once authenticated, they can PUT a single triple (using the mechanism described above), using the `webbox:subscribe_to` predicate, for example, to subscribe to updates to the above blog post:

   ```
   <http://danielsmith.eu/#dan> webbox:subscribe_to
   <http://hip.cat/posts/34>
   ```

   This triple tells the data space to add the user with that URI to subscriber list for the item being subscribed to, and to re-send the "`sioc:addressed_to`" triple to their WebBox whenever there is an update to the resource (e.g., if the blog post is modified).

6. **Unsubscribe Message**: To unsubscribe, the process is the same as the subscription process, except that the `webbox:unsubscribe_from` predicate is used, for example:

   ```
   <http://danielsmith.eu/#dan> webbox:unsubscribe_
   from <http://hip.cat/posts/34>
   ```

   Upon receipt the WebBox removes the user from the subscriber list of that resource.

7. **Query Message**: When a resource has been shared with the WebBox, it is resolved, and stored in a named graph in the local triplestore. By default we suggest using the `webbox:ReceivedGraph` URI for the named graph, although users could reconfigure this if required.

In addition to peer-to-peer messaging, and to support large-scale distribution and sharing of content, WebBoxes can also act as publishing endpoints for organisations, groups and other collectives of common interest; for example, link aggregation "social news" sites like Reddit, Digg, Slashdot and Hacker News can be created by anyone and hosted on any suitable Web-accessible machine. We discuss this, in the context of a Message Relay Application that uses WebBox, in Section 4.5.

4.3 Authentication

Though the data is stored on many machines in a decentralised manner, requiring users to have separate accounts for each server that they edit data would become cumbersome, and will not scale. Therefore, each WebBox that corresponds to a data space will use the WebID protocol [29] to authenticate the users when the triples are pushed to the subscription WebBoxes. The Web browser extension or any client that supports the WebBox architecture will prove the possession of or access to a private key, whose corresponding public key is tightly bound to the WebID (i.e., a FOAF document) that is being authenticated. The private key is usually associated with an X.509 certificate on the user's computer, while the public key can be typically found on the FOAF profile. A Web server that receives the user request can verify that the user has write access to that URL, thus properly identifying the originator of the triples in to the data space. The WebID contains a triple, using a `webbox:address` predicate, which identifies the SPARQL Update endpoint that the can receive triples via the method described by an 'update' message in Section 4.2. In all triples the user is identified by the URI of their WebID, which is dereferenced in order to find the address of their WebBox. Thus, is is possible for a user to change the URI of their WebBox while retaining the same WebID.

4.4 Access Control

When a user publishes a new entity (e.g., a blog post or an annotation), WebBox supports full access control over who can access that entity; if the resource is to be shared, WebBox sets appropriate access controls. These access control

can be set for individual users, or set of users that are identified as a foaf:Group (e.g. "My Jogging Friends Group"). The WebBoxes allow authorised users; typically the owner of the WebBox to set these permissions. This is accomplished using the SPARQL Update protocols as described in [21]. The access control file for a particular resource on a Web server is discovered by either querying the SPARQL endpoint, or by issuing an HTTP GET request against the URI given by the HTTP link header, sent by the WebBox after the user is authenticated. Access control metadata in this file will be stored in a per file/resource basis by default. It is very easy to change this to set the ACLs to be on per document or even per resource basis. We use the Web Access Control Ontology [4] to specify the permissions. The following is an example access control rule:

```
@prefix acl:  <http://www.w3.org/ns/auth/acl#>.

<named rule> acl:accessTo <Resource that access is
granted to, i.e.  a webbox>;
acl:agent <WebID of authenticated user or agent>;
acl:mode acl:Read, acl:Write, acl:Control .
```

Read and Write modes control access to normal files, whereas the Control mode enables a user to change the ACL metadata for that file. Optionally, the user may set `acl:defaultForNew` that defines the default access control rules for new documents in a directory.

When triples are sent to the WebBox, the access control metadata file is checked for the necessary permissions so that the triples can be updated in the data space.

4.5 Applications

In the WebBox model, what used to be "server side" applications under the control of a single person or company providing the service (e.g., Facebook, Twitter and Flickr), can instead run on the user's own devices (i.e., their computer), just as current desktop or mobile applications do today. (This does not, however, preclude applications from being hosted on remote servers, as explained below.)

WebBox does not dictate how these apps must be written or distributed — existing App Markets and software distribution networks, such as the Mac App Store, and Ubuntu's APT repositories, for example, could be used to distribute WebBox compatible apps. Other approaches, such as maintaining indexes of apps, such as `VersionTracker.com` are also possible. Indeed, we imagine that one might be able both to search for applications or to request applications be developed against a functional/data-oriented specification. For instance, one might query for applications that either use the same ontology or has been mapped to the same ontology as an application the person is currently using for, say, health and fitness. Likewise they may suggest they would like a service that supports blending data sets for different views, if one does not exist.

A benefit of WebBox's approach is that it does not prescribe a new language or access protocol — applications communicate to the WebBox server via WebBox's SPARQL endpoint, inserting and updating structured messages — either generic/application specific messages (i.e., using ontologies specific to their application), or WebBox specific messages, such as those concerning Sharing and ACL behaviours (i.e., using the SPARQL Endpoint to read and update triples that use the WebACL and WebBox ontologies), as described in Section 4.2.

In our examples so far we have referred to applications that users interact with directly, however, it is not necessary for a WebBox application to provide a user interface. While some applications do provide user interfaces, (e.g. an application to send and receive messages among friends), some do not. For example, a basic application for WebBox is a message relay which receives data and automatically forwards it to subscribers, thus functioning in a similar manner to e-mail mailing lists.

A mailing list application would work by creating a new resource, and subscribing to changes to it. It would then watch for subscription requests to this data URI and maintain its own list of subscribers. New subscribers then be granted write access to the URI, and when new resources are posted to this URI (using SPARQL Update 1.1 using a 'modify' message as described in Section 4.2), it would be relayed to all subscribers. The application is free to impose its own moderation on the list, for example to require moderators to approve new subscribers, or to grant read-only access to some subscribers.

Another useful and basic app for WebBox is one that allows user to view everything that has been shared with them, and filter it using meaningful criteria. For example, it may be useful to find all resources (i.e., meeting requests, annotations and bookmarks) that related to a particular project, or that were sent by a particular person. Thus, if resources are properly marked up with primary topics, dates, and authors, it is straightforward to use existing SPARQL exploration interfaces to filter down on the contents of the WebBox's knowledge base.

In fact, ensuring that new content is well marked up is a key concept, because it falls outside the requirements of WebBox, but the usefulness of a WebBox is greatly increased when resources include useful information, such as related projects, authors and creation dates. Thus, another important application is one which keeps track of concepts that can be useful as primary topics of applications, in order to use them as a form of "Semantic tagging" [10]. One method is to have an application which keeps track of all `foaf:primaryTopic` and `foaf:primaryTopicOf` assertions, and provides a user with a searchable list of these topics. Similarly, users may find it useful to establish a new primary topic, and share it with their collaborators (for example if they are attending a conference, and the canonical URI to annotate with is being established). This can be achieved by sharing information about that entity to the WebBoxes of collaborators, and sharing some other resource, which has the `foaf:primaryTopic` of the topic's URI. Other users of the primary topic tracking application would then have this topic URI added to their index.

Another type of application is one that could analyse private data (i.e., non-shared data), and automatically respond to requests based on that private information. For example, a common issue when working across-organisations is determining shared availability for scheduling meeting times. Services such as meet-o-matic[5] and doodle[6] provide pages where each person can list their availability for different time and date periods. However, it would be simpler if users could simply share their calendars. For reasons of privacy,

[5]Meet-o-matic: `http://www.meetomatic.com/`
[6]Doodle: `http://doodle.com/`

users shouldn't have to do this. Thus, a way around this is to have an agent watch for meeting requests (e.g., if a new resource using the RDF Calendar ontology [14] is sent to their WebBox with this user as an attendee), and to automatically post back a "busy" assertion to the originator's WebBox if they are busy at that time. This method means that the user does not need to reveal any other information about their calendar, merely that the requested time will not work. The originator could then try a different time (either automatically, or manually) or alternatively content the person directly.

In summary, the WebBox approach enables application developers to turn their applications decentralised using existing SPARQL querying libraries and servers, and thus our approach stands on the shoulders of the existing mature Semantic Web application, tool and library stack.

4.6 Application Permissions

Up to this point, our examples have been locally-running trusted applications that are relatively trustworthy. However, the Web has showed us that applications that run remotely are the modern norm, for a number of reasons, both operational (i.e., to do big data integration) and commercial (i.e., not to risk any leaking of your application's code logic through distribution). Therefore it is important that any distributed data system like WebBox can support that style of application. A key difference, however, when using remote applications, is that they have the potential to "leak" private data out to companies and the greater Web. Therefore, we must ensure that users can select the specific information and granularity of information that applications have access to, rather than giving *carte blanche* global access to all information. For this purpose, we are planning to integrate a distributed authorisation method (such as DAuth [25]) into the platform to enable rich permissions to be set on a per-application basis, as future work.

4.7 Implementation

Given that WebBox uses existing protocols, a WebBox can be implemented using already readily available components. Specifically, we implemented our prototype using the open source triplestore "4store"[20], which provides SPARQL querying and SPARQL Update 1.1 functionality; ReadWriteWeb [7], which provides SPARQL Updates 1.1 support for hosting linked data; and mod_authn_webid for FOAF+SSL[30] authentication.

To tie these components together and implement the WebBox sharing semantics, we wrote a controller layer in Python, in order to:

1. Proxy SPARQL requests from the WebBox to the relevant store. Specifically, when new data is received, it is stored in both ReadWriteWeb, and in 4store. When linked data is read, it is read from ReadWriteWeb, and when SPARQL queries are received, they are routed to 4store.

2. Maintain and enforce the ACL permissions and policies.

3. Ensure that FOAF+SSL authentication credentials from mod_authn_webid are passed to the ACL handling module.

4. Fire update messages (i.e., as the DTA) when entities are updated.

5. Act as the DUA - i.e., de-spool messages arriving and dereferencing remotely updated resources, storing updated resources into the WebBox `webbox:ReceivedGraph` named graph.

[7]ReadWriteWeb: `https://dvcs.w3.org/hg/read-write-web/`

5. PERFORMANCE AND SCALABILITY

As described previously, WebBox provides sharing capabilities much like those offered by the large-scale centralised sharing platforms, Facebook and Google+, except in a decentralised manner. One of the key challenges faced by large-scale sharing services is that of scale and performance – specifically, ensuring that when millions of users use the site simultaneously, they will experience responsive, undiminished performance.

In the case of WebBox, since applications run locally to a user's computer and is primarily responsible for only handling the resources shared by the user, there is simply less data for WebBox to manage. As a result, the challenge of dealing with billions of messages per second simply does not exist, because only those messages that the user receives are stored, not those that related only to other users.

Instead, the main concern pertains to the number of HTTP / HTTPS connections made, since a connection to each receiver's server must be made with each update. A consequence of this is that each receiver will then respond by requesting the updated resource from the WebBox. Thus, an entity with N subscribers requires N outgoing change-notification messages (of message type No. 4 from Section 4.2 — 'share' message), which result in N incoming requests ('read' message). As N increases, the server load in dealing with the incoming requests increases, and thus a mitigation strategy is desirable in order to support very large values of N.

A simple strategy to reduce the overhead of sending notification messages is to use the HTTP upgrade mechanism to establish a WebSocket[19] if connections to a particular host are being made frequently. This technique can provide substantial computational savings particularly since establishing HTTPS connections repeatedly can be expensive. This approach is taken by pubsubhubbub hubs in order to mitigate the overhead of polling RSS feeds [3].

A secondary strategy, if a WebBox becomes overwhelmed, is to stagger the updates to subscribers, so that their incoming requests are likewise staggered. The delay between updates could be increased based on the load of the server on which the WebBox is running. This method works because we assume that the WebBox that received the message will instantly update its data. If this is not the case, this method may not be as effective.

An inherent benefit of our approach is that continuous polling is avoided, because updates are pushed to peers. This is similar to the benefit of using pubsubhubbub [3] instead of RSS, where a single hub polled an RSS feed and pushed updates to subscribers, thereby taking the load off server hosting the originating RSS feed, as well as preventing the need for the subscribers to continuously poll.

A more complex approach is to alter the way that updates propagate. For example, instead of sending an up-

date to each individual subscriber, a "gossip" protocol can be used where the updates are propagated between peers, rather than all directly from the originator [16]. The key benefit to this approach is that the load is spread between peers, rather than from a central point. WebBox's ability for applications to relay messages (as described in Section 4.5) means that relay hubs could be set up, in order to form gossip networks. Methodologies already exist to set up overlays over distributed semantic data, in order to achieve greater scalability in data distribution, which use gossip protocols to distribute data [8].

In practice, however, typical users do not have that many friends (Facebook users have 130 friends on average [8], and thus the number of incoming and outgoing messages is small scale.

6. CASE STUDY: BUILDING SOCIALBOX

In this section, we walk through the implementation of SocialBox, a WebBox application that encompasses microblogging, social news aggregation and bookmarking, "tumblelogging" (a form of blogging that affords one-click re-blogging of media content), life/datablogging with social feedback, as found in current social data applications described in Table 1. Unlike most current sharing platforms that constrain the kinds of data that can be shared the data added to a SocialBox can be any structured data objects. Examples of such structured data objects a user might share include vCards of contacts gleaned from an RDFa-encoded Web page, an HTML-encoded table of data (such as a recipe on epicurious[9]), or an annotated jogging log posted from a person's personal fitness logger. Moreover, because it is built on the WebBox platform, such shared items can be public, private, or shared selectively – in a secure and verified manner.

We first describe the steps required to implement microblogging/tumblelogging for SocialBox, comprised of 3 steps: (1) creating and sharing a microblog entry, (2) aggregating others' entries into a unified visualisation, and (3) supporting commenting/re-sharing. All these steps involve messages described in Section 4.2.

6.1 Creating a Simple Shared Resource: A Microblog Entry

To create a resource SocialBox makes an HTTP PUT call to create a resource ('create' message). To determine how this entry should be shared, SocialBox needs to prompt the user for the friend(s) and group(s) he or she wants to share the new resource with. To do this, SocialBox needs profiles for all known groups, friends, and followers. A user's friend list is represented in the WebBox as a FOAF profile, and is queried using an HTTP GET to retrieve the FOAF file as Linked Data ('read' message).

Adding friends or groups corresponds to performing an appropriate SPARQL Update on the FOAF profile to add the additional `foaf:knows` assertions (using SPARQL Update 1.1 with a 'modify' message).

After friends and groups have been selected, SocialBox invokes a 'share' message, which effectively sets up change notification policies in the DTM and updates access control policies using WebACL as described in Section 4.4.

[8]Facebook statistics: `https://www.facebook.com/press/info.php?statistics`

[9]Epicurious: `http://www.epicurious.com/`

6.2 Building a View of a Shared Resources

To present the user with a list of most-recent resources people have shared with each other, such as a Twitter feed or Facebook news feed, SocialBox simply makes a SPARQL query ('query' message) to identify the most recent items that have been added.

6.3 Adding Annotations

As described earlier, annotations are any resource (structured or unstructured) that is linked to its topic, (usually with the `foaf:primaryTopic` property, although the WebBox is agnostic to such choices). As such, adding support for creating comments and other kinds of annotations involves simply creating a new resource representing the annotation (as described in Feature 6.1) with this appropriate property. Retrieving all annotations involves performing a SPARQL query (as described in Feature 6.2) for all objects that are linked to a particular resource.

6.4 Adding a Social News Feed View

To cope with the fact that looking at a raw microblog stream (such as a raw Twitter stream) can be overwhelming, people often prefer to filter incoming items based upon popularity/number of votes, such as those provided by social news aggregation sites like Digg or Reddit. To provide such a view in SocialBox, we change only one aspect of the query used to retrieve resources from WebBox: the order of returned results is ordered by number of votes instead of recency. To ensure freshness of resources a clause can be added to filter for only recently created resources.

6.5 Re-sharing

Sites like Tumblr allow resources that one user has shared to be easily re-shared on the person's own tumbleblog, with links back to the original post. Unlike plain link re-sharing, users can typically comment on this particular re-share without these comments being propagated to any other re-shares. To support this behaviour, SocialBox simply creates a resource representing the re-share, linking back to the original post. Annotations can then be made relative to the re-share (*à la* Tumblr), or, alternatively, relative to the original resource.

6.6 Shared Editing

Few social network platforms support shared resource editing; on Facebook and Google+, for example, all resources added to the timeline (e.g., status updates, comments, links and so on) cannot be edited — only deleted — once posted. In many situations, however, being able to have multiple parties editing a resource, such as shared notes (e.g., Etherpad) or documents (e.g., Google Docs) is essential for effective collaboration. Since WebBox handles propagation of updates to resources automatically with subscriptions (using a 'subscribe' Message), making a shared resource editable by multiple parties simply amounts to setting the ACL policy on it to permit others to make updates. Updates are made using a 'modify' message.

6.7 Polls and Surveys

Doodle polls and Google Forms are often used to resolve scheduling issues and other matters of logistics requiring input from everyone in a group. To implement similar functionality using WebBox, SocialBox uses a structured re-

source type called a Survey which it renders as an HTML form. SocialBox provides a form-builder interface for specifying the structured contents of the form, which asserts an RDF class declaration to represent the structure of form responses. When this class resource is shared with others, SocialBox turns it into its HTML form to be displayed. The form definition specifies the assertions that will be made based on the form answers. Each form section is associated with a subject URI, and each input element is associated with a predicate. For example, a list of foods, with a "rating" from 1–5 for each food. The output of the form is a graph of assertions about the food. The resulting RDF graph is then shared to the WebBox of the survey owner (using a 'share' message), where is it processed by SocialBox.

SocialBox demonstrates that it is possible to replicate and to integrate social data sharing services in a decentralised, secure way. SocialBox also enables users to share resources without the constraints imposed by centralised services such that arbitrary data types can be added to the SocialBox, and also editing/revision of these sources is enabled: full data control is part of the WebBox platform and is utilised in the SocialBox application.

7. DISCUSSION AND FUTURE WORK

In this section we discuss topics where implementations of WebBoxes can offer additional capabilities and features that are not core to the basic requirements of a WebBox. We also describe future work in the area of distributed data sharing on the Web.

One of the most used analogs to a WebBox is e-mail, which enables users to share information directly, albeit in a way that is not machine-readable. E-mail is a good analog because users can choose to use e-mail entirely online (i.e., with a Web browser), or use a local client which downloads copies of all of their mail. In our examples we have assumed that a user has their data locally, although in reality their local computer is unlikely to be connected to the same open internet connection 24 hours a day. Instead, like e-mail a WebBox is more likely to exist on an internet server (so it can receive data without interruption), with a synchronisation process to a user's local computer (or mobile devices, as with e-mail on smart phones), so that applications can be run locally, as discussed in our examples. However, unlike in email where anybody can send any message to an inbox, resulting in spam, WebBoxes can prevent unauthorised users from authentication against a WebBox.

One assumption throughout is that connections to WebBoxes and between WebBoxes use TLS [17], so that they are fully encrypted at a protocol level, to prevent man-in-the-middle attacks. However, if a untrusted party (such as a commercial service provider) is used to host the WebBox, then users may wish for their data to be encrypted. As in FOAF+SSL [30], linked data can share public keys of users, so that an entirely machine-readable public key infrastructure can be used to encrypt data between users. That data can then remain encrypted while at the remote WebBox, and decrypted when downloaded.

One limitation of the WebBox approach is that for applications where users are collaborating and modifying shared resources, version control of the entities is desirable. We have not included version control within our definitions of a WebBox because there is not a standard protocol that could be used "out of the box" for keeping track of versions. However, we acknowledge that a distributed version control service such as Git or Mercurial could be used to keep track of revisions to files as they are updated between users. Thus, the addition of a version control system is future work.

We propose two approaches for deploying apps using WebBoxes. (1) Have organisations host WebBoxes for its members for external apps to be deployed on, or (2) Group of people share an app that will create the WebBoxes for the users of that app. The first approach is similar to the existing email architecture on the Internet, where ISPs and companies host mail servers for users to store their messages on. The second approach is geared towards novice users who may only want to use a specific app (for e.g. an app such as 'getfit.mit.edu' where people log their workout progress and share with their friends). Of course, these users should be able to use their WebBox for another app. Advanced users can host their own WebBox and deploy apps of their preference on their WebBoxes.

For WebBoxes to offer effective privacy, there is a need for intuitive user interfaces that will enable sharing of some parts of the data, and specifying subsets of users to share those pieces of data with. There is also a need for user interfaces for authoring of share and access control policies.

8. CONCLUSIONS

In this paper we have presented WebBox, an architecture that facilitates user-determined, decentralised social data services on the Web that exhibits the following key features:

1. **Fully-Decentralised** Every person runs their own WebBox, obviating the need for central servers.
2. **Flexible Data Representation** Shared data can represent any structured data, including yet-to-be-defined data schemata of future applications.
3. **Granular Sharing** Data can be arbitrarily small or large — from a single statement to an entire graph. Furthermore, resources can be shared with specific individuals, or arbitrary-sized groups.
4. **Security** Secure authentication via WebIDs ensures that the person with whom we think we are sharing our data is actually that person; TLS/SSL (HTTPS) transport ensures that only the person(s)/entities with whom you are sharing can decrypt the data.
5. **Maintains Personal Privacy** As individuals, we are able to control where data is stored and with whom it is shared.
6. **Web-Based** Critically, WebBox uses standard Web protocols to make it easy to integrate with existing environments and software.

WebBox extends the role of the web server from a document publishing platform to one that fundamentally supports distributed collaboration on data artefacts, specifically RDF resources. We demonstrated the feasibility of this approach, the ease with which applications can be written for this platform, and how existing standards and protocols enable this approach without the need for major reinvention. Critically, WebBox represents an implementation of Socially-Aware Cloud Storage [11] in which social-sharing Web applications access a user-controlled data space which

is used for private storage and shared resources, and remains entirely under the user's control. Thus unlike existing centralised sharing platforms where data and applications are inextricably tied, data can be used by multiple applications and services and shared directly between peers. From a user's perspective, their data can be managed in a single location, rather than multiple disparate locations, which leads to easier management, and reduces fragmentation and redundancy across sites and services. Moreover, this ensures that their personal data can be maintained independently of applications, ensuring its longevity and sustainability.

9. REFERENCES

[1] Open source software for building public and private clouds. *www.openstack.org*.

[2] VRM Blog. *blogs.law.harvard.edu/vrm*.

[3] A web-hook based pub-sub protocol. *http://code.google.com/p/pubsubhubbub*.

[4] Basic Access Control Ontology. *W3C* *http://www.w3.org/ns/auth/acl#*, 2004.

[5] Higgins Personal Data Services. *Eclipse wiki* *eclipse.org/higgins*, 2009.

[6] Opera Unite reinvents the Web. *http://pl.opera.com/press/releases/2009/06/16/*, 2009.

[7] Web services under your control. *OwnCloud* *owncloud.org*, 2010.

[8] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. Van Pelt. Gridvine: Building internet-scale semantic overlay networks. *The Semantic Web–ISWC 2004*, pages 107–121, 2004.

[9] D. Alexander, A. Mitchell, and W. Heath. Mydex Community Prototype Launch Press Release. 2010.

[10] D. Beckett. Semantics through the tag. In *Proceedings of the XTech Conference, Amsterdam, The Netherlands*, 2006.

[11] T. Berners-Lee. Socially Aware Cloud Storage. *W3C Design Note* *http://www.w3.org/DesignIssues/CloudStorage.html*, 2009.

[12] T. J. Berners-Lee. Information management: a proposal. oai:cds.cern.ch:369245. Technical Report CERN-DD-89-001-OC, CERN, Geneva, Mar 1989.

[13] D. Berrueta and J. Phipps. Best practice recipes for publishing rdf vocabularies. *W3C Working Group Note, 28 August 2008* *http://www.w3.org/TR/swbp-vocab-pub/*, 2008.

[14] D. Connolly and L. Miller. Rdf calendar - an application of the resource description framework to icalendar data. *W3C Interest Group Note, 29 September 2005* *http://www.w3.org/TR/rdfcal/*, 2005.

[15] S. Decker and M. Frank. The Social Semantic Web. *DERI Technical Report* *http://www.deri.ie/fileadmin/documents/DERI-TR-2004-05-02.pdf*, 2004.

[16] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[17] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. *IETF RFC* *http://wiki.tools.ietf.org/html/rfc5246*, 2008.

[18] C. European Parliament. Directive 95/46/EC of the European Parliament. *http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:NOT*, 1996.

[19] I. Fette and A. Melnikov. The WebSocket Protocol, IETF HyBi Working Group. 2011.

[20] S. Harris, N. Lamb, and N. Shadbolt. 4store: The design and implementation of a clustered rdf store. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pages 94–109, 2009.

[21] J. Hollenbach, J. Presbrey, and T. Berners-Lee. Using RDF Metadata To Enable Access Control on the Social Semantic Web. *CEUR Workshop Proceedings*, 2009.

[22] J. Miller, S. Murtha-Smith, and Team. The Locker Project. *lockerproject.org*, 2010.

[23] C. Ogbuji. Sparql 1.1 graph store http protocol. *W3C Working Draft, 12 May 2011* *http://www.w3.org/TR/sparql11-http-rdf-update/*, 2011.

[24] J. Presbrey. Read Write Linked Data Space. *data.fm* *data.fm*, 2010.

[25] J. Schiffman, X. Zhang, and S. Gibbs. Dauth: Fine-grained authorization delegation for distributed web application consumers. In *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, pages 95 –102, july 2010.

[26] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam. PrPl: A Decentralized Social Networking Infrastructure. *ACM Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond (MCS)*, 2010.

[27] D. Simonds. Online Social Networks: Everywhere and Nowhere. *The Economist print edition*, 2008.

[28] R. Singel. Facebook Debuts Simplified Privacy Settings. *Wired Magazine*, 2010.

[29] M. Sporny, T. Inkster, H. Story, B. Harbulot, and R. Bachmann-Gmur. Web Identification and Discovery. *W3C Editor's Draft* *http://www.w3.org/2005/Incubator/webid/spec/*, 2011.

[30] H. Story, B. Harbulot, I. Jacobi, and M. Jones. Foaf+ssl: Restful authentication for the social web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*. Citeseer, 2009.

[31] D. R. Thomas. A General Inductive Approach for Analyzing Qualitative Evaluation Data. *American Journal of Evaluation*, 27(2):237–246, 2006.