INTERNET DRAFT
Document: draft-dejong-decentralized-sharing-00

Intended Status: Informational
Expires: 30 November 2015

Michiel B. de Jong
(independent)
Paul Tran-Van
(Cozy Cloud)

29 May 2015

Decentralized Sharing

Abstract

This draft describes the steps and challenges of sharing documents between persons, using internet-connected servers. We investigate the situation where a document exists on a server to which the sender has access through some software application, but the recipient(s) don't. All recipient(s) do, however, have access to software applications that run on least one other server.

We discuss existing and proposed methods for the sender to initiate the communication, for each recipient to become aware of the sender's intent to share a document, for each recipient to access the document directly, for the document contents to be transmitted to server(s) to which the recipient(s) have access, for the sender to make and announce changes in the document after it was initially sent, and for the recipient(s) to communicate comments and proposed changes to the document back to the sender.

We also discuss how semantics of the document may be communicated, and how versioning conflicts may then in some cases be resolved programmatically. Given that users of personal servers don't all run the same compatible software on their server, the sending application needs to discover which application-level protocols are supported by the receiving application(s).

This Internet-Draft will expire on 30 November 2015.

Copyright Notice

Table of Contents

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [WORDS].

"SHOULD" and "SHOULD NOT" are appropriate when valid exceptions to a general requirement are known to exist or appear to exist, and it is infeasible or impractical to enumerate all of them. However, they should not be interpreted as permitting implementors to fail to implement the general requirement when such failure would result in interoperability failure.

2. Introduction

When we say a sender 'shares' a document with some recipient(s), we primarily mean the sender makes that document available for the recipient(s) to consume, but not to others. Additionally, the sender may send updated versions of the document after it was sent, and recipient(s) may reply with comments and propose changes to the document. The sender may decide to make such updates accessible to all original recipient(s) or not. When we say 'document', we mean a string of data which encodes some human-readable content in some data format.

There are several protocols for sending a document from one server to another. For instance, the sender may use an email client to connect to her server via the [IMAP] protocol, or use a web browser to connect to her server via the [HTTPS] protocol ("webmail"), to instruct her server to forward an existing mail message from her server to one or more recipients, identified by their email addresses, after which the mailserver application running on her server can deliver the document to the recipient's server using the [SMTP] protocol. The recipient(s) may then access the document from their server over the [POP], [IMAP], or [HTTPS] protocol.

However, many software applications which people use nowadays to access their documents, do not implement the SMTP, IMAP, or POP protocol (often due to the complexity of dealing with email spam). Also, there is no commonly supported programmatic way for the sender to update documents after they have been sent over SMTP, nor for the recipient(s) to reply with proposed changes. Rather than proposing such a versioning

protocol on top of SMTP, we investigate how currently existing software applications implement decentralized document sharing in practice, and where these common practices could be improved and extended.

Some software applications allow sharing documents between users who have access to the same server (we could call this centralized sharing), and some also implement application-specific protocols that allow their users to share documents with other users of the same software. This leads to a dispersion of efforts between personal server software projects. The authors hope this document will contribute to more interoperability between such software applications.

Many software applications do allow sharing documents publically, for instance by making them accessible over [HTTPS]. There are many ways to restrict access to documents that are published that way, but this does not solve the question of how recipient(s) get notified in the first place that (and how) they have access to such a document.

There is no consensus yet on how to share documents privately over the internet, in such a way that recipient(s) are notified, but other people cannot retrieve the document, nor discover its existence.

This draft does not propose one universal solution, but rather lists what puzzle pieces we are aware of, and what options may exist to put them together. In general, we are interested in the case where there may be more than one recipient, and where each recipient may have access to more than one server (but not to the sender's server). But for clarity of discussion we will at this point look at a simpler case.

Many of the suggestions given in this Internet-Draft have not yet been standardized, and many of them have not yet proven their merit in large-scale deployments. In most cases, we try to list all initiatives we are aware of to implement a certain requirement or step of the process, and references to these proposed approaches are often only informative references to specifications which have been published independently by their authors, on their personal websites, on forums, or on wikis. We try to refer to a specific revision of such publications wherever possible.

We introduce Alice, who wants to share one document with Bob. Alice and Bob both have a personal server, which they access to over a web interface, that's to say, using a web browser that connects to their server over HTTPS. This interface displays information in a

human-readable way, and allows humans to easily send commands using for instance their web browser's point-and-click gestures.

We assume there are multiple documents on Alice's server, and the web interface visualizes these for her. We make no assumptions at this point about the content of these documents; they can be text, hypertext, sound, image, or other media files. They can contain some machine-readable data such as the contact information of a person, the details of a calendar event, a move in a chess game, etcetera.

Alice decides to share one file with only Bob, and keep it invisible for the others. What are the steps through which she can achieve this, depending on the capabilities of her (the sender's) server and Bob's (the recipient's) server?

3. Initiate sharing process

To initiate the process of sharing a document from her server with Bob, Alice connects to her server with her web browser, providing some credentials (how she provides these is out of scope). She navigates the web interface of her server to select the document she wants to send to Bob.

Alice clicks or gestures her 'Share' intent for the document in some way. The software application may have the capacity to act as a [MICROPUB] client, in which case at this point Alice would also specify the MicroPub endpoint to use.

4. Recipient selection

Alice may have an addressbook on her server where she stores the identities and contact methods of people she knows. She may be able to select Bob from this addressbook using the same or another web interface. She may also have such an addressbook on her own client device, or for instance on paper. Or maybe she knows an identifier for Bob by heart. If Bob or someone else has published some of his contact information, she may use a search application to find out identifiers by which Bob can be contacted, which she can then use directly, and possibly also store in her addressbook for future reference.

It is also possible that she remembers or looks up a web page describing

how to contact Bob, and discovers a usable identifier this way. An addressbook application may also be able to infer identifiers for various contact methods from such a web page programmatically.

Other methods for discovering a contact method and identifier for Bob may rely on Alice knowing that Bob is a friend of Cindy, and she may search a list of friends of Cindy named Bob, to confirm whether the intended Bob is listed there. Similarly, she may remember that Bob is for instance a member of her rowing club, and search a directory of members of that rowing club to locate contact information about him. Again, such an inference may be helped by a software application which is capable of infering such friend-of-a-friend relationships.

Finally, Alice may have a contact identifier for Bob which she knows will allow her to communicate with him, but not to share the document with him. She may for instance contact Bob by telephone or irc to ask him for his email address.

5. Recipient notification

Depending on the contact methods for which Alice has discovered Bob's identifier in the recipient selection step, she may complete the document sharing process by sending either the document data to Bob, or a notification which contains machine-readable and/or human-readable instructions for retrieving the document. For instance, she may send the document as an email attachment, or send an email message that contains a hyperlink or URL through which Bob may access the document.

In the case of sending the entire document directly, it will be helpful to include some metadata about the document encoding. In the absence of this, the software application that Bob uses to visualize the received document may try to guess the encoding and media type of the document, for instance by analyzing its first few bytes of data.

In the case of sending only the information needed to access the document, but not the document itself, the necessary identifiers and credentials for accessing the document may be given in natural language, accompanied by human-readable instructions like 'You can download the document at the following URL: ..., using the following password: ...'.

The notification may also contain information about how Bob can comment on the document, or propose changes. Sometimes this information may be

implied by the notification method, or become obvious when Bob accesses the document. For instance, Alice may send the URL of a web page which contains a 'Comment' or 'Edit' button, inviting Bob to comment or propose changes to the document using that web page. These comments and proposed changes may then either be applied and displayed immediately, or the web application may allow Alice to review them before they are applied and displayed.

One protocol for notifying recipients is [WEBMENTION]. This is a successor of the older [PINGBACK] and [TRACKBACK] protocols. WebMention is primarily intended for sending comments on existing documents, and not for initially sharing a document in the first place, nor for proposing edits. But by "commenting" on a recipient's main identifying URL (home page), it could also be used to send a document that is not a comment on any existing document. Also, as WebMention comments are often formatted as machine-readable documents using [MICROFORMATS], a 'proposed-edit' comment type could be defined, by which Bob's server may communicate Bob's proposed edits to Alice's server in a machine-readable way. The authors intend to research both these possible extensions to the WebMention protocol.

The software application may allow Alice to select the method by which to notify Bob, or select a reasonable default. Bob may also publish machine-readable preferences indicating how Bob prefers to receive documents that are shared with him.

The notification may also include human-readable information about which types of feedback are invited from Bob. For instance, Alice may share a document which contains a collection of photos, and Bob may be invited to add photos to this collection, but not delete any of the existing ones. The authors intend to research ways of adding such feedback invitations in a machine-readable way, for instance to a WebMention notification.

Note that inviting certain types of feedback does not necessarily imply that Alice's server will automatically accept, apply, and display such feedback. Alice may want to review all proposed changes before applying them, and review all comments before displaying them.

The notification may include all necessary credentials for accessing the resource, in which case it is important for the software on Bob's server not to display such a notification publically, as this would lead to accidentally publishing a privately shared document publically.

Alternatively, the authentication step may be delayed until Bob, or software running on Bob's server, actually accesses the document.

The software that Alice uses to send the notification to Bob may use a concept of roles, which summarize which types of feedback Bob will be invited to reply with. For instance, she may select a 'contributor' role to invite Bob to send back only comments and additions, or an 'admin' role inviting Bob to propose any kind of change to the document, including deletions. We consider such roles mainly a matter of interaction between Alice and the software she uses.

6. Human access

Whichever the way of sending the notification, it SHOULD contain a URL for Bob to open with a browser to view and/or download the document from what we call an access page.

The notification MAY include a password or other type of secret which Bob needs to paste or type into the web page. The web page located at the URL from the notification MAY also allow Bob to authenticate with other methods if Alice can reasonably assume Bob will have the ability to use those. Examples of such methods are [OPENID], [INDIEAUTH], [INDIECERT], [PERSONA], [WEBID-TLS], and [WEBID-RSA]. It MAY allow Bob to authenticate by proving he can receive a secret sent to him via mobile telephone short messaging service (SMS) or email. It MAY also allow Bob to authenticate using third-party identity providers like social network sites.

Some of these methods of authentication rely on Alice to provide the correct identifier(s) for Bob in the recipient selection step. These identifier(s) are not necessarily equal to the identifier used for sending the notification. For instance, if Alice knows both Bob's WebMention end-point and his mobile phone number, she may send the notification to Bob's WebMention endpoint, but allow Bob to authenticate via SMS while accessing the document.

The relationship between Bob's identifiers in different identity systems may in some cases be discovered programmatically, for instance with [WELL-KNOWN] resources, or [REL-ME] links on the main (index) web page, for Bob's personal DNS domain name.

The list of authentication methods Bob is offered will be limited by

the capabilities of the software that displays the access page for the document. If this software is integrated with Alice's addressbook software, then the addressbook entry for Bob may also be updated with the authentication methods for which Alice provided the identifier, those which were discovered programmatically during the process, and the one that Bob ultimately used to authenticate.

The access page MAY also give human-readable instructions to Bob, like "please don't share this document publically", "please don't share outside our Petanque team", or "please discard after reading, or after **30 days". Whether or not Bob follows such instructions is out of scope.**

7. Machine access

Some software applications are able to retrieve a document with [UNATTENDED-INDIEAUTH] in reaction to a WebMention notification. Although the [SOLID] platform does not describe how to send or receive a document sharing notification, it does describe how an application can use WebID-TLS or WebID-RSA to authenticate programmatically to retrieve a document of whose existence it is aware.

If the URL of the access page contains a secret string of characters [CAPABILITIES], then an application can also access the document without human intervention.

If an application or human attempts to access the document without sufficient credentials, a 410 HTTP response code can be sent, together with a WWW-Authenticate header hinting to a proposed authentication method.

8. Publishing edits

After Alice has shared a document with Bob, if she did not send it directly as for instance an email attachment, she may still edit its contents between the time of sending the notification, and the first time Bob accesses the document through the human-readable access page, or a software application accesses it on Bob's behalf.

But even once Bob, or software on his behalf, has accessed the document for the first time, Alice may want to change its contents. For instance, Alice may share her calendar with Bob, and keep adding events to it

afterwards.

If she does this, she may send Bob a new notification, inviting him to access the document again, to see the changed version. Another way of publishing new versions of a same documents might be for Alice to publish an [RSS] feed, an [ATOM] feed, or a [COUCHDB] changes feed instead of the document itself.

Alternatively, the document could contain a reference to a feed of its change history, for instance using a hyperlink in case of a HTML document. A software application accessing the document on Bob's behalf could discover this reference programmatically and keep polling the change feed periodically or each time Bob views the document, apply Alice's changes, and possibly provide a way for Bob to view its change history.

The approach where Alice's application sends a new notification to Bob's application is more efficient if changes are rare, because it does not require Bob's application to retrieve the change feed many times (polling). The [PUBSUBHUBBUB] protocol could also be used as a way to avoid unnecessary polling of change feeds.

The change feeds could concern one document, or aggregate changes in all changes Alice ever shared with Bob into one feed. The MicroPub protocol also supports updating documents that have already been published [MICROPUB-UPDATES], so the application Alice uses to update the document could send a MicroPub Update to the application which handles the sharing of the document.

There are also situations where Bob can be expected not to make any copies of the original document version, retrieving it always directly from Alice's server when needed, and Bob may be more interested in always retrieving the latest version, instead of being notified about changes when they happen. An example could be when the document contains an [OEMBED] data snippet, which Bob embeds in another document by reference. This also means Bob will no longer have access to the document if Alice revokes his access to it, or stops publishing the document at its URL.

9. Comments and write access

Regardless of whether or not Alice has invited him to do so in the

notification or in the access page, Bob may send reactions, like comments, answers, additions, or proposed changes to Alice. He may send Alice a notification for each reaction, a reference to a change feed, for Alice's server application to poll.

It is up to Alice's application to either display all of Bob's comments and changes immediately, or for instance to only display additions and not deletions. This MAY be signalled in the notification and/or in the access page (see above), so that Bob knows whether he effectively has "write access" or "read-only access" to the document or not.

Alice's server may immediately apply Bob's changes (depending possibly on the type of change) without Alice's intervention. If there were multiple recipients, then maybe some recipients have permission to edit (i.e., changes they propose will always be applied), and others only to view (i.e., when they send a proposed change, it is not applied, or is only applied after Alice reviewed and approved it.

This means that Alice can keep modifying the resource after sending it, and Bob can also send modifications back to Alice. If Bob's application retrieved a copy of the document and becomes aware of changes to it made or approved by Alice, it SHOULD display such changes automatically, possibly with a way to browse the change history. Alice's application MAY display the document to Alice in its latest version with all its approved changes (regardless of who proposed those changes), or display both the version she shared and the modifications proposed by Bob side-by-side.

We will now consider the case where Alice sent the message to possibly more than one recipient, and the document may at any point contain changes proposed by any of these recipients (either accepted immediately by Alice's application, or reviewed and approved by Alice). When Alice's change feed contains changes initiated by one of the recipients, it SHOULD refer back to the URL where this change was originally proposed by the recipient in question. This will allow the application of a recipient to programmatically compare the "master" change feed published by Alice with the list of changes proposed by that recipient itself. This is a common pattern in [GIT] versioning, where branches can be merged into a master branch, retaining the information about the individual changes (commits).

If Bob reacts with a comment or a change proposal, he should somehow notify Alice of this. WebMention was intended to do exactly this,

for comments on a publically published document. It is common for applications to display (links to) all comments on a document on its public access page. This practice may lead to [LINK-SPAM] when applied to public documents, and when applied to privately shared documents, it may result in accidentally publishing a private comment to the other recipients of that shared document.

Bob MAY also send proposed changes to Alice over http, using a verb like PUT or PATCH. This practice is used in [READ-WRITE-WEB] and [COUCHDB], among others. Such HTTP requests SHOULD be responded to with a HTTP response status that indicates whether the change request was successful, accepted to be reviewed, or rejected.

A third practice for sending change proposals are git pull requests, which would traditionally be sent to Alice by pgp-signed email, with the proposed patch in an attachment.

The authors intend to research whether we could define a way to mark up change proposals in microformats, for Bob to post an updated version and notify Alice of this. For instance, if Alice published a document containing an h-entry of type photo-album, then a reaction to this could be of type addition, and refer to a photo which Bob wants to be added to that photo album. The same pattern could serve for adding events to a calendar, or adding files in a document which represents a file system directory.

Alice's application MAY be able to interpret changes to a document and resolve the possible conflicts between them in certain cases (see also the Data domains section below), or notify Alice and/or Bob that more information is needed to resolve the conflict. A conflict occurs when Alice and Bob both make changes in the document and arrive at different new versions. In database theory this is called the optimistic lock pattern: conflicts are tolerated, but the versions should eventually converge and reach consistency. A pessimistic lock pattern would require notifying all other actors with access to the document before starting to make any changes to the current document version.

10. Real-time collaboration

If Alice and Bob are connected to the internet simultaneously, a real-time conversation may be initated, for instance using [WEBRTC]. For example, Alice might share a document with Bob which represents a

"phone call", after which an application on Bob's server may send a [PUSH-NOTIFICATION] to Bob's (mobile) device, and when Bob gestures to "pick up the phone", his server could react to Alice's "phone call" with a document representing a WebRTC SessionDescriptionProtocol object. This would allow rapid exchange of information back-and-forth between Alice and Bob, for the duration of the real-time collaboration session.

Also, if Bob uses the access page to propose changes to the document, and Alice is also connected to the internet at that time, the interchange between Alice and Bob may be communicated between Alice's server and her client device via [WEBSOCKETS]. This scenario could be also be generalized to the situation where more than two people are viewing, and possibly editing, the same document. To reduce the information traffic that would need to pass through Alice's server, these WebSockets could be used to establish WebRTC data channels between each device, provided these devices support WebRTC, and that changes are still submitted and ultimately committed to the dominant current version of the document, on Alice's server.

11. Listing and revoking access

If Alice wants to share several documents with several people, a software application on her server SHOULD allow her to view a list of who has access to which document.

This list view SHOULD also display whether she configured the application to programmatically apply any comments and/or proposed changes from these people, and/or whether these people have been sent an invitation to comment or propose changes, whether as part of the notification they were sent, or through the human-readable access page.

It MAY also display whether the recipient has already accessed or retrieved the document or not. It MAY also give Alice the option to revoke access entirely, taking into account that the recipient may already have made a private copy of the document. It MAY also give Alice the option to change, for each recipient, whether their comments and/or proposed changes will be displayed and applied programmatically, acknowledged and queued for Alice to review, or rejected altogether.

If Bob uses an application on his own server to display the document shared by Alice, and this application is capable of programmatically applying all changes Alice sends as updates after sharing the initial

version, then Bob SHOULD EITHER be given a way to instruct that application to stop or start programmatically applying all changes sent by Alice, OR provide a way for Bob to view all versions Alice sent. It MAY also provide a way to visualize the differences between the versions published by Alice, and changes proposed by Bob.

If Alice publishes updates which correspond to changes proposed by Bob or other recipients, then these updates MAY refer to the URL where these people originally proposed such changes. That way, Bob's application can display which proposed changes have been accepted by Alice into her master version of the document. Alternatively, Bob's application could still compare a change made by Alice with each of Bob's proposed changes, to find out whether it's identical to one of them, and display this change to Bob as 'his' change, accepted by Alice, rather than as a change initiated by Alice.

12. Data domains

12.1 Discovering the semantics of a document

If Bob retrieves the access page with his browser, the presentation can be tailored to how Bob as a human wants to consume the data. For instance, multiple download/export formats may be offered using a web view in a natural language Bob understands.

When Bob's application retrieves the document, it may not be able to discover its semantics, and therefore may not be able to add it to existing document collections on Bob's server in a way that allows Bob to use the document in the way it was intended by Alice. For instance, if Bob's application stores hypertext documents as [HTML] documents, but Alice's server uses the [MARKDOWN] format for all hypertext documents, a conversion process may be necessary to translate documents shared by Alice to the format for which Bob's application can offer domain-specific features. Otherwise, his application may display the markdown document as just a string of data, or as if it were a plain text document, and not allow Bob to follow the hyperlinks from there.

Data format hints can be provided using Content-Type headers, or some self-describing data notation [JSON-LD, RDF, TURTLE, RDFA]. This can then point to a unique entry in some vocabulary of data formats.

Even if Bob's application can programmatically discover the

data format in which Alice intended the documented to be interpreted semantically, there are many incompatible vocabularies [VOCABULARIES]. To complicate things further, many competing data formats exist for the most common data domains.

For instance, consider the case where Alice wants to share a description of a person's contact details with Bob. Let's assume Alice's application sends this document in the Turtle format, and Bob's application recognizes the Content-Type header, and is able to machine-read Turtle documents.

Further assume that Alice's application uses the [DBPEDIA] vocabulary to link to the "person" concept. If Bob's application uses the [MICROFORMATS] vocabulary to interpret documents, it may be able to discover the equivalence between the "person" concept as defined by DBPedia and the "person" concept as defined by MicroFormats, through [SAME-AS-LINKS].

But even then, in order for Bob's application to give Bob the option to merge the information from the Turtle+DBPedia document into an existing Turtle+MicroFormats document representing the same person, the two vocabularies are likely to differ in which attributes they are able to express, and same-as links may not exist for all individual attributes.

Also, inference rules will sometimes be needed to for instance compose a full telephone number if parts of it are represented as an international access code or an area code, or to compose a full name from a family name and a given name or list of first name initials.

This means that communicating the intended semantic interpretation of a document in a machine-readable way involves three hurdles: first, the hinting mechanism, whether part of the document (self-describing) or added as metadata (for instance in a Content-Type header) which links to a URI for the data format, then resolving the equivalence between the many identifiers which exist for the same concept, but in different vocabularies, and finally converting Alice's data format itself to a data format that allows Bob's application to offer functionality beyond simply displaying the raw data from the document.

Sending for instance a photo, a text document, a contact (description of a person), a blog post or Activity [ACTIVITY-STREAMS], a comment, a git pull request, etc., will only work in rare cases where Bob's application is able to handle the hinting mechanism, as well as the data format, as

well as the semantic vocabulary used.

Our recommendation would be for the receiving application to support many data format hint mechanisms, as well as many data formats, as well as many vocabularies, giving preference to those often used in practice, and to those which have been accepted as W3C recommendations.

12.2 Programmatic conflict resolution

When Alice's application receives multiple proposed changes to one single version of the document, and Alice accepts both changes, then this constitutes a versioning conflict. After applying the first change, it is no longer possible to apply more changes directly, since they propose to change a version of the document which is no longer the current version. In this situation, Alice's application MAY visualize this conflict to Alice, so that she can decide how the conflicting change can be transformed into a change to be applied to the current version.

But in some cases, Alice's application can also apply inference rules to determine this transformation programmatically. To achieve this, Alice's application needs to have different inference rules that are specific to the data format, as well as to the concept about which the document expresses information.

For instance, for a document in Turtle format, representing contact information for a person, transforming the addition of a mobile phone number past a change in work place could be safe to do programmatically, without consulting Alice explicitly. The transformation rule would be specific to the data domain (contact information of a person), and also to the data format (Turtle). Attempting to apply transformation rules that work for text files to for instance raw image data will lead to undesired results.

There will also remain cases where such programmatic conflict resolution cannot be achieved at all without leading to incorrect results. For instance, when multiple proposed changes propose to change the same data field within a Turtle document, they contradict each other even at the semantic level (not just at the syntactical level), and Alice and Bob will likely want to communicate more with each other to agree which proposed change to apply, and which one to discard.

13. Security Considerations

When Alice's application sends a notification about a document that was shared with a limited audience, it SHOULD NOT send this notification in such a way that Bob's application, or Bob himself, could interpret this notification as describing a publically published document.

The access page through which Bob may view the document which Alice shared with him, SHOULD clearly display instructions for how Bob should treat the document's content, including if, how, and for how long, he may store a copy of the document on another medium, and if, how, and with whom, he may discuss or share the contents of the document.

Furthermore, Alice's application SHOULD make it clear to Alice that she depends on Bob's cooperation and decision to follow these instructions or not.

14. IANA Considerations

This document does not propose any new entries in any IANA naming registry.

15. Acknowledgements

The authors would like to thank everybody who contributed to the development of the ideas discussed in this document, including the IndieWeb and Social Web communities, the Decentralized Sharing Working Group, the participants in OuiShareLabsCamp 2015, as well as Alice and Bob themselves.

16. References

16.1. Normative References

[WORDS]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[IMAP]

M. Crispin, "Internet Message Access Protocol - Version 4rev1", RFC 3501, March 2003.

[HTTPS]
E. Rescorla, "HTTP Over TLS", RFC2818, May 2000.

[SMTP]
J. Klensin, "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[POP]
J. Myers, M. Rose, "Post Office Protocol - Version 3", RFC 1939, May 1996.

[OPENID]
OpenID Foundation, "OpenID Authentication 2.0 - Final", http://openid.net/specs/openid-authentication-2_0.html, December 2007.

[WELL-KNOWN]
M. Knottingham, E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[CAPABILITIES]
J. Tennison (ed.), "Good Practices for Capability URLs", http://www.w3.org/TR/capability-urls/, February 2014.

[RSS]
D. Winer, "RSS 2.0 Specification", http://cyber.law.harvard.edu/rss/rss.html, July 2003.

[ATOM]
M. Nottingham (ed.), R. Sayre (ed.), "The Atom Syndication Format", RFC 4287, December 2005.

[WEBRTC]
A. Bergkvist, D. C. Burnett, C. Jennings, A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", http://www.w3.org/TR/webrtc/, February 2015.

[PUSH-NOTIFICATION]
B. Sullivan, E. Fullea, M. van Ouwekerk, "Push API", http://www.w3.org/TR/push-api/, April 2015.

[WEBSOCKETS]
I. Hickson, "The WebSocket API", http://www.w3.org/TR/websockets/, September 2012.

[HTML]
I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. Doyle Navara, E. O'Connor, S. Pfeiffer, "A vocabulary and associated APIs for HTML and XHTML", http://www.w3.org/TR/html5/, October 2014.

[JSON-LD]
M. Sporny, G. Kellogg, M. Lanthaler, "JSON-LD 1.0", W3C Proposed Recommendation, http://www.w3.org/TR/2014/REC-json-ld-20140116/, January 2014.

[RDF]
G. Schreiber (ed.), Y. Raimond (ed.), "RDF 1.1 Primer", http://www.w3.org/TR/rdf11-primer/, June 2014.

[TURTLE]
E. Prud'hommeaux (ed.), Gavin Carothers (ed.), D. Beckett, **T. Berners-Lee, "RDF 1.1 Turtle: Terse RDF Triple Language",** http://www.w3.org/TR/turtle/, February 2014.

[RDFA]
B. Adida, M. Birbeck, S. McCarron, I. Hermans, "RDFa Core 1.1 - Third Edition: Syntax and processing rules for embedding RDF through attributes", http://www.w3.org/TR/rdfa-core/, March 2015.

16.2. Informative References

[MICROPUB]
A.Parecki (ed.), "MicroPub", http://indiewebcamp.com/wiki/inde \
x.php?title=Micropub&oldid=19338, May 2015.

[WEBMENTION]
IndieWebCamp Wiki, "WebMention", http://indiewebcamp.com/wiki/ \
index.php?title=Webmention&oldid=19485, May 2015.

[PINGBACK]
S. Langridge, I. Hickson, "Pingback 1.0", http://www.hixie.ch/specs/pingback/pingback, 2002.

[TRACKBACK]
See: https://movabletype.org/documentation/trackback/specifica \
tion.html

[MICROFORMATS]
See: http://microformats.org/

[INDIEAUTH]
See: https://indieauth.com/

[INDIECERT]
See: https://indiecert.net/

[PERSONA]
See: https://developer.mozilla.org/en-US/Persona.

[WEBID-TLS]
See: https://github.com/linkeddata/SoLiD#webid-tls

[WEBID-RSA]
See: https://github.com/linkeddata/SoLiD#webid-rsa

[REL-ME]
See:
http://microformats.org/wiki/index.php?title=rel-me&oldid=64351

[UNATTENDED-INDIEAUTH]
See: http://indiewebcamp.com/wiki/index.php?title=indieweb-mes \
saging&oldid=19179#Protocol_Flow

[SOLID]
See: https://github.com/linkeddata/SoLiD

[COUCHDB]
See: http://docs.couchdb.org/en/1.6.1/intro/api.html

[PUBSUBHUBBUB]
B. Fitzpatrick, B. Slatkin, M. Atkins, J. Genestoux,
"PubSubHubbub Core 0.4 -- Working Draft", http://pubsubhubbub. \
github.io/PubSubHubbub/pubsubhubbub-core-0.4.html, February
2014.

[MICROPUB-UPDATES]

A.Parecki (ed.), "MicroPub: Update", http://indiewebcamp.com/w \
iki/index.php?title=Micropub&oldid=19338#Update, May 2015.

[OEMBED]
See: http://www.oembed.com/#section7

[GIT]
See: http://www.git-scm.com/

[LINK-SPAM]
See: https://en.wikipedia.org/wiki/Spamdexing#Link_spam

[READ-WRITE-WEB]
See: https://www.w3.org/community/rww/

[MARKDOWN]
See: http://daringfireball.net/projects/markdown/

[VOCABULARIES]
See: https://www.w3.org/Social/track/issues/15

[DBPEDIA]
See: http://wiki.dbpedia.org/

[SAME-AS-LINKS]
See: http://sameas.org/

[ACTIVITY-STREAMS]
See: http://activitystrea.ms/specs/

17. Authors' addresses

Michiel B. de Jong
(independent)

Email: michiel@michielbdejong.com

Paul Tran-Van
(Cozy Cloud)

Email: paul@cozycloud.cc