

Data Service Infrastructure for the Social Sciences and Humanities

EC FP7

Grant Agreement Number: 283646

**Deliverable Report**

Deliverable: D5.6

Deliverable Name: DASISH Web Annotation (DWAN) framework

Deadline: M36

Nature:

Responsible: Daan Broeder, MPG-TLA

Work Package Leader: Daan Broeder, MPG-TLA

Contributing Partners and Editors: Valentina Ascuitti (KCL), Daan Broeder (MPG-TLA), Stuart Dunn (KCL), Twan Goosen (MPG-TLA), Indrek Jentson (University of Tartu), Kees-Jan van de Looi (MPG-TLA), Olof Olsson (UGOT), Stephanie Roth (UGOT), Olha Shkaravska (MPG-TLA), Menzo Windhower (MPG-TLA).

Table of Contents

2. Executive Summary 1

3. Introduction to DWAN framework 2

3.1 Motivation 2

3.2 User scenario’s 2

3.3 Requirements 4

4. Annotation Tools 4

4.1 Sate of the art on September 2012 4

4.2 Developments after September 2012. 9

5. DASISH Web Annotator (DWAN) 9

5.1 Framework architecture 9

5.2 DWAN’s Model and its connection to Open Annotation Model 10

5.3 DWAN Backend 13

Architecture 13

Database and Database Access Objects 14

REST Application Programming Interface 15

5.4 DWAN frontend(s) 20

Wired-Marker based frontend 20

Frontend for ELAN 23

Frontend for ANNEX 23

5.5 Testing Procedure 24

6. Results and Outlook 25

6.1 Tool’s features 25

6.2 Potential front-ends for DWAN. 25

7. APPENDIX 33

7.1 Schema 33

7.2 Wired-Marker-based DWAN frontend. Manual 39

# Executive Summary

The availability of digital archives and other research data via the internet creates new chances for collaboration. Indeed, equipped with a special software, researchers from different institution, countries and fields can work together via the network. Such collaboration can take the form of annotating the data and sharing these annotations using an annotation infrastructure. As stated in the task 5.6 description, *researchers need to be able to store the results of collaborative intellectual work either as an annotation of a single fragment or in the form of typed relations between a number of fragments.*

The aim of this document is to give a specification of a framework for annotating web-documents developed according to task 5.6 plan. By an annotation we mean a remark over a fragment(s) of a document(s).

From the technical point of view the proposed framework consists of one *backend*, constituted from the server software and the database, and possibly multiple *frontends* (clients). Developed within DASISH project the DWAN tool is an instance of the DWAN framework. It consists of the backend part and the client, which is a significantly adjusted version of the *Wired Marker* Firefox extension. The core of the backend is a database where annotations and information about corresponding annotated target documents are stored together with targets’ cached representations. Archiving cached representations in the database is relevant when annotated documents are dynamically changed pages like news sites or wiki-pages under construction.

A client in DWAN framework exchanges data with the server by sending REST requests and getting responses. Client-request bodies and server's responses have a form of XML files. The client is able to accept and send XML structures that obey a pre-defined XML schema. The schema mirrors a data model that has been designed to represent the main data structures, which are involved in constructing annotations.

# Introduction to the DWAN framework

## Motivation

In the last decades, we have witnessed large amounts of data moving to digital archives. These archives have been connected to the Internet, spreading the content through the research community. The availability of such data creates new chances for collaboration. To bring this collaborative environment to a next, higher level, the requirement is to develop a set of tools that allows groups of researchers from different institutions, countries, or backgrounds to work together. Such collaboration can take the form of annotating the data, and sharing these annotations using an annotation infrastructure.

By an annotation we mean a remark over a fragment(s) of a document(s). For instance it can be a text note containing the short English translation of a certain sentence in a target document, which is in Catalan. Annotatable documents include, for instance, web-pages or web-documents generated by linguistic software, e.g. EAF-files, created by [ELAN](http://tla.mpi.nl/tools/tla-tools/elan/).

## User scenarios

Consider examples of user scenarios that can be successfully run on the DWAN tool in its current state. It is assumed that before getting acquainted with these scenarios the reader either has certain minimal experience with Wired Marker or at least has read DWAN *Manual,* which can be found in the Appendix at the end of this document.

The term *principal*, which we use below, in general denotes either a user or a group of users. At present user and principal are synonyms for DWAN tool.

1*) Getting an annotation whose web-page page has been updated*.

The principal logs in, sees the list of annotations, chooses the one (s)he wants to see on the corresponding web-page. (S)he clicks on the annotation in the list, tries a few times to reload the page, but the annotation does not appear. The front-end cannot resolve the annotated fragment, possibly because the page has been updated and the fragment has changed its position or has disappeared completely. The principal requests the front-end to retrieve the remote cache and gets the cached representation of the page together with the other annotations done on this page by the moment of making the annotation under consideration.  Indeed, it can be seen that the page has been updated.

This use case was a part of the DWAN demo during LREC 2014. The wiki-page of "Right Sector" was used. “Right Sector” is a block of right and extreme-right groups in Ukraine. Due to highly unstable situation in the country this page is updated very often.  The reader can get the annotations on this wiki-page and their cached representations if (s)he has the DWAN frontend installed.

2) *Changing users’ access mode for a particular annotation*.

DWAN client is implemented in the way that when a principal posts an annotation then all DWAN's registered principals automatically get the right to *read* the annotation. This excludes possibility to alter it. On principal’s request, the backend can issue one of two HTML forms for changing access modes, which can be *read*, *write* or *none*. Filling up and submitting the first form allows to change the access to a particular annotation of a particular principal. The second form is used to change public access mode at once, that is all registered principals get *write* access. Only owner of the annotation can change its access modes. Of course, all the changes are invertible.

Updating access modes is implemented through backend’s HTML forms because changing access rights is not implemented in Wired Marker, and adding this feature to DWAN frontend would be quite time consuming.[[1]](#footnote-1)

3) *Creating specific principal’s folder*.

Initially the DWAN client generates the set of pre-defined folders (colored “marker”) for the logged-in principal. Annotations are sorted by the corresponding colors. If the initial collection of markers does not suffice the principal can create her own subfolder by choosing a new marker from a richer set of colors. For instance, she creates a purple marker-subfolder, where she collects all the annotations about the family of Picasso, from various web-pages.

The color information is sent to server when an annotation is posted. Unfortunately, sorting annotations by colors is visible only by the creator of the annotations and only on that instance of the client where these annotations have been created.  For the time being a client cannot interpret color information that it gets from the server and all “the others” annotations are collected in folder called *incoming* and have light-yellow color. [[2]](#footnote-2)

4) *An annotation on an annotation*.

A principal makes an annotation and then the same or other principal annotates this annotation. This feature can be used e.g. in remote discussions. It is technically possible since in DWAN annotation framework one can annotate any document accessible via the internet. DWAN annotations are such documents. [[3]](#footnote-3)

## Requirements

In the DWAN framework it is assumed that possibly multiple clients communicate with a single backend consisting of the database and the server software which implements access to the database. A client is developed specifically for a particular sort(s) of web-documents, whereas the backend is not specific and treats requests of all clients in the same way. The core of the backend is the database where annotations and information about annotated targets are stored together with cached representations of the targets. A cached representation is a copy, e.g. a screenshot, of a target document. Storing cached representations allows to retrieve the copy of an annotated document when the actual web-document under the target’s URI has been updated so that locating the annotation in it becomes difficult or even impossible. It may happen when the corresponding fragment has been significantly changed or disappeared.

A client and a server exchange data via sending REST requests and responses. A client accepts and sends XML files that obey DWAN XML schema which is a part of the server-side software. It mirrors a data model that has been designed to represent the main data classes, which are involved in constructing annotations, and relations between these classes.

# Annotation Tools

## State of the art on September 2012

Before development of DWAN began, more than 40 available annotation tools had been investigated to see if they could be (and to which extend they could be) used as a starting point for DWAN client. Selection had been based on four criteria: tool’s functionality compliance with task 5.6 requirements, if it is open software, if it can be adjusted to communicate with the backend, and platform-independency. The table below represent the results of the investigation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tool | state | Open software | Backend access | Platform (browsers) | Functionality |
| A.nnotate | active | commercial |  |  | Annotating PDF, Word and other document formats on-line |
| AnnotationEdit | active | Commercial |  |  | Annotating video, audio |
| Annotator | active | Open source   Java Script | yes |  | Library and plug-in adding annotation functionality to any web-page, but one needs to alter its html by running script there |
| Annotea, Amaya | Last release jan 2012 | Open source | yes | Different distributions Linux, Windows, MacOS | Written in C, annotating html-web documents |
| Awesome Highlighter. | Not active ??  Web-page broken |  |  | Firefox bookmarklet, or add-on,  or by using the tool ‘s website | Highlighting and clipping chunks of text on web-documents |
| Blerp | Not active ??  Web-page broken |  |  | IE PlugIn, Firefox addOn | Support conversation on top  of the web-page |
| BounceApp | active | Free app | yes | Via their web-page http://www.bounceapp.com/ | Collaborative via sending “screenshots” in e.g. Facebook, Twitter and Notable |
| The Commentor | active | Commercial with the base plan free (3 collaborators) |  | Web-site, you need an account | Collaboration on visual media projects |
| CritLink | Last executable from 2000 |  | yes | Unix | Annotating web-documents in local networks and on the internet. Different colour means different sorts of comment: support (green, +), issue (red, -), bcomment (bule, #), query (orange, ?) |
| Crocodoc | active | Commercial with free Standard edition | yes |  | Annotated PDF, word, Pwerpoint documents are saved on Crocodoc servers |
| DIIGO | active | Commercial  with free base account | yes | Firefox, Safari, IE | Annotating web-pages, saved to Diigo library, Diigo account is needed |
| DrawHere | active | ??, one needs  an account |  | Firefox, IE bookmarklet | Drawing on web-pages, shareable |
| ThirdVoice | Discounted in 2001 |  |  | Browser Plug-in | Commenting web-sites, anyone could write anything; a lot of criticism from the web-page owners |
| Wired Marker | active | Creative commons | yes | FireFox extension | Highlighting and putting text notes on the fragments of web-documents |
| Fleck | Tool of 2006, Inactive? their site does not exist any more |  |  |  |  |
| http://delicious.com/ | active | Need an account | no | Bookmarklet |  |
| http://evernote.com/ | active | Need an account,   premium is commercial | no | Server, storage of the documents | With “skitch” : annotating pdf and images, not web-pages |
| http://webmarginalia.net/ | active | Open source Java Script | yes | Firefox, Safari, Chrome, IE,  For Moodle and Open Journal system | Highlighting html |
| http://www.yandell-lab.org/software/mwas.html | active | Need an account | no |  | Annotating genomes |
| TrailFire | Last mentioned in 2007 |  |  | Firefox, IE | Annotating (notes) webpages, categrozing annotated web-pages, sharing |
| REddIT | active | Need an account | no | server | Social networking  and  news website |
| ReframeIT | obviously not available any more, only light-weight demo on website, add-ons outdated, integration info missing on official website |  |  |  |  |
| Scrible | under development: free public beta version available | premium, paid edition under development, not yet available, license:  no modifications allowed |  |  |  |
| SharedCopy | State uncertain. According to http://en.wikipedia.org/wiki/Web\_annotation: Development has stopped. Observe: copyright date of official website: 2012 |  |  |  |  |
| ShiftSpace | Development has stopped. |  |  |  |  |
| Skim | active | BDS license |  | OS X | PDF reader and node taker |
| WebNotes | active | platinum /pro/lite version/, account is needed,  modification under permission |  |  | Adding notes to PDF and web-pages |
| JKN | new? |  |  |  | <http://info.jkn.com/firefox.htm>, Light version with available features: web page annotation, organize and search notes, share notes via email, twitter, and permalink or any other similar url found from annotation evaluation lists - they didn't work at all! |
| Keeppy |  |  |  | Server | <http://www.keeppy.com/>, a social network, relevant for our purposes |
| Loomp | new? | No license information, short technical information and easy access for downloading is missing. |  |  | One Click Annotator, a WYSIWYG Web editor for enriching content with RDFa annotations, <http://loomp.org/index.php/home.html>, |
| MarkITUp | new? | MIT/GPL licence, based on former jTagEditor, |  | needs jQuery 1.4.2 Javascript library | Toolbox. Will never be WYSIWYG editor. <http://markitup.jaysalvat.com/home/>, |
| NotateIT | new? | not open sources |  | only for Windows, seems not to be compliant with other platforms | <http://www.notateit.com/> |
| WebKlipper | new? | commercial |  |  |  |

As one can see there were not that many open-software tools with suitable functionality available. Moreover, not many of them were well-documented. At the end the decision was made to select *Wired Marker* as a starting point for DASISH web-annotator client.

*Wired-Marker* is a Creative-Common licensed Firefox plugin, with the possibility to change the code under the agreement with its creators. It is platform independent since Firefox is one of the most popular browsers installable at Linux, OS X and Windows. The access to the backend database can be adjusted.

*Wired-Marker’s* functionality, though limited, still goes in line with DASISH purposes: a text fragment of an arbitrary web-document can be marked by a selected colour and a text remark (an annotation body) can be put on it. It was possible to extend the code so that the extension could communicate wit the server to retrieve an annotation from the database or send a created annotation to the backend.

Another tool, called *PundIt* can do more than Wired-Marker, but unfortunately by the time when DASISH task 5.6 team had to make a decision, it was not yet available and it got Open Source license after development of DWAN had already started.

## Developments after September 2012.

*PundIt* allows to annotate images and their fragments, to collect annotations into notebooks. In fact, Notebooks can be viewed as a generalised version of colours (markers) of *Wired Marker*. A notebook can be read exactly in the form intended by its creator.

*PundIt* has a feature, which in some cases may be considered as an inconvenience. While creating an annotation, a user must think in terms of a triple *Object-Predicate-Subject*, for instance “Karl Marx” (subject) “talks about” (predicate) “Kapital” (object). “Karl Marx” denote not only a piece of text but it is rather a wider notion, an *item*. Under this *item* one can collect texts, images or their fragments representing Karl Marks on the web-pages.

*ReframeIt* has appeared as a Firefox add-on for commenting web-pages and sharing it via Facebook, Twitter, Blogger, FriendFeed, Wordpress, RSS, HTML, e-mails.

# DASISH Web Annotator (DWAN)

## Framework architecture

The DWAN design assumes multiple clients working together with a single backend consisting of a database and a Representational State Transfer (REST) web service implemented in Java. It allows annotating any web-accessible content, linking data, creating relations, or providing feedback. Its novelty is that the created content and target annotated documents can be stored in a digital archive (in the database), which guarantees their sustainability and persistence. The digital storage for annotations and related resources is provided by TLA-MPI[[4]](#footnote-4).

DWAN is also especially meant to cater for specific linguistic tools that through their use of linguistic data formats can annotate specific linguistic items such as lexical items, annotation tags etc.



Figure 1: The DWAN Framework

## DWAN’s Model and its connection to Open Annotation Model

The class *Annotation* is the core of the model (see Figure 2). The relations *Annotation - Target*, *Target - Source*, *Target - Cached Representation* closely follow the *Open Annotation* (OA) standard. An annotation, i.e. an inhabitant of the class *Annotation*, is a structure that contains necessary information about user's annotation. In particular it contains the annotation's identifier, the reference to the owner and the time of creation. An owner is either the principal who has created the annotation or a principal to whom the ownership has been assigned. Recall, that a principal is ether a user or a group of users, and for the current version of DWAN user and principal are synonyms. Creating user's groups is the matter of the future work.

Besides the owner, an annotation has *readers* and *writers*. As one can expect, a reader is a user that can read the annotation, and a writer can also add changes to it. Thus, a registered principal can be related to an annotation by means of one of three access modes: *reader*, *writer, none*.

An annotation can have one or more *targets*. A target (i.e. an inhabitant of the "Target" class) contains the reference to the web-document (a *source*) and the precise description of the document's fragment, which is actually annotated. Moreover, a target may refer to one or more cached representations of (the relevant parts of) the target document with the precise descriptions of the annotated fragments for each representation.

The semantics of an annotation is given in its body. In the implementation a body is an arbitrary text or an XML text. In both cases a precise MIME-type must be given by a client. For instance, a body can be a plain text, which describes a relation (like contradiction) between two fragments of some web-document. In this case the body should contain references to the targets that represent these two fragments and the document.

Annotations can be gathered in notebooks.



Figure 2: a simplified class diagram representing the DWAN model

The targets of DWAN model correspond to the instances of the open annotation class oa:SpecificResource.



Figure 3: The DWAN model expressed as relations using the Open Annotation vocabulary

Multiple target sources are represented as instances of oa:Composite. Each of oa:item of the composite is either an instance of oa:SpecificResource or oa:Composite.

A cached representation of an annotated target source is referred via the target’s state, see the figure above. The properties oa:hasState and oa:cachedSource are used. The metadata of the cached representation are presented via dc-properties and dctypes: mimeType is presented as ​dc:format, tool is presented as ​dc:publisher, type is presented as ​dc:description; note that ​dc:type cannot be used here because its value must (recommended) be from the DCMI Type Vocabulary[[5]](#footnote-5); therefore, e.g. "screenshot" would not be a good value here. Moreover, a cached representation must have one of the dctypes as rdf:type, and it must be compliant with dc:format value. For instance, if dc:format is "image/png" then the corresponding rdf:type must be dctypes:Image.

A principal is an agent, and for agents *Open Annotation* recommends to use foaf namespace, see ​<http://xmlns.com/foaf/spec/>. This data model is designed for social networks, and in principle suits DASISH schema for a user and permission lists. There is one little technical inconvenience: foaf agents do not have a property which can be used to define permission types (reader, writer) directly. For now, permissions are represented via property foaf:topic\_interest. For an example, see Figure 4.



Figure 4: example of an OA representation

An annotation body in DASISH can be any correct xml or a text. A generic way to present such bodies in "Open Annotation" is to consider a body, which (typically) has attributes and elements, as instances of oa:Composite. Any element and any attribute of the body becomes an oa:item of the body. If an element has sub-elements, it is an instance of oa:Composite as well, etc. An attribute or an element with now sub-elements has one of the dc:types and one of the dc:formats, and possibly additional relevant properties, such as cnt:chars for text values.

## DWAN Backend

### Architecture



### Database and Database Access Objects

A PostgreSQL relational database provides storage for all the core resources: annotations, targets, cached representations, principals and notebooks. The database contains five main tables; each of them stores a corresponding type of resource. A column in a table represents an attribute in the corresponding resource class. For instance, any resource class has an attribute *id* (an identifier of type *xml:id*). This identifier is a part of the URI through which a client accesses an instance of the resource. URI has the form <service-uri>/<resource/<id>, e.g. *https:/dasish.mpi.nl/api/annotations/e3c834f0-34c4-11e3-aa6e-0800200c9a66*. Each of five resource tables has its column *external id*, that keeps public identifiers. From the programming point of view an external identifier is a UUID string generated by the server when a resource, e.g. an annotation, is added to the database.

Annotation bodies are stored in the table *annotation* in the column *body*.

Furthermore, there is a number of join tables representing the relations between the resources, which are described as relations between the resource classes. These relations induce a hierarchy between the resources. Indeed, any of the relations can be abstracted to "refers" so that we have that a *principal* refers to an *annotation* or a *notebook*, an *annotation* refers to a *target* and a *target* refers to a *cached representation*. As one can see, cached representations have the lowest position in this hierarchy. This hierarchy induces a "cascading" mechanism of adding and deleting resources in the database. For instance, removal of an annotation from the database triggers the removal of its targets, except for the ones to which other annotations still refer. In turn, removal of the targets triggers removal of all the corresponding cached representations unless some other targets refer to a cached representation under consideration.

*Database Access Objects* (DAO's) are used to programmatically access the data in the database. The DAO mechanism allows to form and call SQL database commands like SELECT, UPDATE, INSERT, and DELETE from Java methods. Methods for basic manipulations over resources (retrieving, updating, adding and deleting) are defined in the corresponding DAO java interface. For instance, the AnnotationDao.java interface lists the signatures of all necessary basic operations over the table *annotation* and the join tables *annotation-targets* and *annotations-principals-permissions*. By a basic operation we mean an operation which demands a single SQL statement. The interfaces are implemented using *SpringDAO[[6]](#footnote-6)*, which utilises a JDBC[[7]](#footnote-7) connection to access the data store. For instance, the *add annotation* method is implemented in JdbcAnnotationDao.java class as a single java method. As one expects, this method forms and calls an INSERT command for the table *annotation*.

Due to the presence of join tables there must be a mechanism that takes care of correctly sequencing basic operations. For instance, consider a complete procedure of deleting an annotation. The annotation's internal database identifier occurs in three join tables *annotations-targets*, *annotations-principals-permissions*, *notebooks-annotations*. If the annotation record is deleted from the table *annotations* before the corresponding rows in the join tables are removed, then the join tables have references to the non-existing annotation (via its internal identifier), and the database will signal an integrity error. To prevent such errors we have introduced a java class *DBDispatcher.java* which calls the methods from the DAO implementations in the correct order. Moreover it triggers cascading of the operations when necessary. For instance, complete deletion of an annotation amounts to purging the join tables first, then deleting the corresponding record in the *annotation* table, and then triggering removal of the annotation's unused targets.

Auxiliary resource-info classes generated by JAXB for the corresponding xml types *TargetInfo*, *AnnotationInfo*, *NotebookInfo* contain references to the corresponding resource plus the most important information about the resource.

### REST Application Programming Interface

The server and a client communicate with each other by means of a REST Application Programming Interface (API). A REST API is a collection of requests which the server must recognise and respond to in an appropriate way. Requests are made by means of a URL starting with the server's location specified by the type of requested resource and its identifier when applicable.

Requests of method type GET are used to retrieve information about resources stored in the database. For these GET requests the URL generally contains the identifier of a requested resource (as a *path request parameter*). For instance, it can be the identifier of an annotation or the identifier of a cached representation. Passing a principal identifier as a parameter is not required, because the active principal is known from the session via an identification procedure (e.g. *Shibboleth*). A PUT (resp. DELETE) request is used to update (resp. delete) the resource of which the identifier is given as a request parameter. Only the *owner* has DELETE rights. POST is performed when a client wants to create a new annotation. Most information necessary to fulfil a PUT or POST request is not given as a request parameter, but given serialized in the request body. For instance, to submit an annotation a client needs to fill in the request body with the XML-element corresponding to class *Annotation*. All the information necessary to create an annotation should be placed in the corresponding nodes of the XML-element.

If a POST (PUT) request is sent then in the case of success the server returns the serialized information about the added (resp. updated) resource together with a standard HTTP response code. If an annotation is posted or updated the server returns an xml document of type *envelope*, which contains a serialization of the resource together with the list of actions which client should perform to complete the request in a sound way. For instance, if an annotation is posted so that for one of its targets there is no cached representation in the database the list of action contains reminder to post a cached representation for the corresponding target id.

In the case of failure of the request the corresponding error status (with the detailed message when necessary) is returned, e. g. 401 *Unauthorized access* if the principal is not logged in (except for the log-in service).

Before describing the requests in more detail we give the list of used notations.

| **notation** | **meaning** |
| --- | --- |
| *aid* | annotation identifier |
| *cid* | cached-representation identifier |
| *datetime* | date and time, including time zone,  as defined in <http://www.w3.org/TR/xmlschema-2/#dateTime> |
| *nid* | notebook identifier |
| *prefix* | the prefix of a namespace |
| *tid* | target identifier |
| *text* | some text |
| *prid* | principal's id |
| *URI* | URI, as defined in <http://tools.ietf.org/html/rfc3986> |
| *Principal* | a user (person) or a group of users |

In the tables below all the requests are listed and the corresponding server responses are described.

**Principal realm**

| **Resource** | **Description** | **Return (xml) type** |
| --- | --- | --- |
| GET api/authentication/login | Redirects to the login page, if the principal is not logged-in, or messages otherwise. | String message |
| GET api/authentication/principal | Returns logged-in principal. | Principal |
| GET api/principals/*prid* | Returns principal with the given prid. | Principal |
| GET api/principals/*prid*/current | Returns *true* if the *prid* is logged-in; *false* otherwise. | CurrenPrincipalInfo |
| GET api/principals/info?email=*user@mail.com* | Returns the principal with the given e-mail address. | Principal |
| GET api/principals/admin | Returns the string with the name and the e-mail of DWAN admin. | String |

**Annotations**

**api/annotations**

| **Resource** | **Description** | **Return (xml) type** |
| --- | --- | --- |
| GET api/annotations?  link=*URI*& text=*text*&  access=[[*read, write*]]&  ns=*prefix:ns*&owner=prid&  after=*datetime1*&  before=*datetime2* | Returns the annotations filtered by the request parameters list of info-s of the annotations to which the logged-in principal has *read* (resp. *write*) access. Their links contain *uri*, their bodies contain *text*. Moreover, these annotations are created between *datetime1* and *datetime2*. If the parameter *link* is omitted, then considers all annotated objects to which the principal has *read*/*write* access. The default *datetime1*is 01 Jan 1970, 00:00. The default *datetime2* is today. | AnnotationInfoList |
| POST api/annotations | Adds a new annotation by picking up its XML-serialization from the request body. | Envelope AnnotationResponseBody |

**api/annotations/aid**

The table below describes requests in which the logged-in principal has authorized access to *aid*. *Authorized access* means that the principal has *read* access for GET-methods and *write* access for PUT body methods. Any logged-in principal can POST an annotation. To change permissions of the annotation the principal must be the *owner* of the annotation. If the principal tries to perform a request for which (s)he does not have privileges, the status 403 *Forbidden* is returned.

| **Resource** | **Description** | **Return (xml) type** |
| --- | --- | --- |
| GET api/annotations/*aid* | Returns the annotation that has this *aid*. | Annotation |
| GET api/annotations/*aid*/targets | Returns the list of the *tid*-s of all the targets of *aid*. | ReferenceList |
| DELETE api/annotations/*aid* | Removes *aid* from the database, together with all its targets to which no other annotation refers | String messaging how many rows have been deleted (should ne 0 or 1) |
| PUT api/annotations/*aid* | Updates the annotation with *aid*. For instance, it is used when *prid* wants to correct typos in the annotation body and change annotated fragments. (See PUT api/annotations/*aid*/body for correcting body only.) The serialized representation of the updated annotation is given in the request body. | Envelope AnnotationResponseBody |
| PUT api/annotations/*aid*/body | Updates the body of the annotation *aid*. Used e.g. for correcting typos in the text part. | Envelope AnnotationResponseBody |
| GET api/annotations/*aid*/permissions | List of permissions for the *aid*. In user is not included in the list his access is defined by *public* attribute. | PermissionList |
| PUT api/annotations/*aid*/permissions | Updates the permission list. New permission list is given serialized in the request body. | Envelope PermissionResponseBody |
| PUT api/annotations/*aid*/permissions/*prid* | Updates the access mode for the annotation *aid* and principal *prid*. New access mode is given in the body of the request. | String messaging how many rows have been updated/added  (should be 0 or 1). |

**Targets**

A target represents a specific fragment of a specific version of an annotatable source. An instance of the *Target* and *TargetInfo* type has a string attribute *version*, which is to be filled by a client when an annotation is posted (or updated) and sent to the server. An *Annotation* type contains target-info elements that keep information about the annotation targets.

**api/targets**

| **Resource** | **Description** | **Return (xml) type** |
| --- | --- | --- |
| GET api/targets/*tid* | Returns the target with a given id. | Target |
| GET api/targets/*tid*/versions | Returns the lists of the URIs of all the sibling-versions of the *tid*, that is targets related to the same source (the same link). | ReferenceList |
| POST api/targets/*tid*/fragment/ *fragmentdescriptorstring*/cached | It is a 2-part POST, with the request body consisting of serialised *CachedRepresentationInfo* instance, and a single file representing the chacher representation itself: HTML document, image, etc. multiple files must be archived. | CachedRepresentationInfo |
| DELETE api/targets/*tid*/cached/*cid* | Removes connection *tid*-*cid*s. The cached representation is removed from the database as well, unless there are more references to this representation. | String messaging how many rows in the junction table have been removed,  should be 0 or 1 |

**api/cached**

It is possible to store the cached representation not only of the fragment precisely corresponding to annotation's target but of a larger fragment and even of the entire annotatable document. For instance, the DWAN client sends to the server the entire DOM of the annotated page, when an annotation is created. The relation between the target and its cached representation should be completed by a fragment descriptor pointing to the position of the annotated fragment in the cached representation. For instance, for a screenshot it may be an (x,y) -position of a left-upper corner of the annotated fragment and the size of a rectangle.

| **Resource** | **Description** | **Return (xml )type** |
| --- | --- | --- |
| GET api/cached/*cid*/metadata | Returns the meta-information of *cid* if it exists. | CachedRepresentationInfo |
| GET api/cached/*cid*/stream | Returns the file (stream), which is the cached representation with *cid* if it exists. | Stream, it is up to the client to interpret it correctly |
| GET api/cached/*cid*/content | Returns the image file, which is the cached representation with *cid* if it exists. | Image |

**Notebooks[[8]](#footnote-8)**

**api/notebooks**

| **Resource** | **Description** | **Return (xml) type** |
| --- | --- | --- |
| GET api/notebooks | Returns notebook-infos for the notebooks accessible to the logged-in principal. | NotebookInfoList |
| GET api/notebooks/owned | Returns the list of all notebook references owned by the logged-in principal. | ReferenceList |
| GET api/notebooks/*nid*/readers | Returns the list of *prid*-s who allowed to read the annotations from the notebook. | ReferenceList |
| GET api/notebooks/*nid*/writers | Returns the list of *prid* that can add annotations to the notebook. | ReferenceList |
| GET api/notebooks/*nid*/metadata | Returns all metadata about a specified notebook *nid*. | Notebook |
| GET api/notebooks/*nid*? maximumAnnotations=  *limit*& startAnnotation=*offset*&  orderby=*orderby*&  orderingMode=[[1,0]] | Returns the list of all annotations aid-s contained within a Notebook with related metadata. Parameters: *nid*, optional *maximumAnnotations* specifies the maximum number of annotations to retrieve (default -1, all annotations), optional *startAnnotation* specifies the starting point from which the annotations will be retrieved (default: -1, start from the first annotation), optional *orderby*, specifies the RDF property used to order the annotations (default: dc:created ), optional *orderingMode* specifies if the results should be sorted using a descending order *desc*=1 or an ascending order *desc*=0 (default: 0 ) | ReferenceList |
| PUT /notebooks/*nid* | Modifies metadata of *nid*. The new notebook’s name must be sent in request’s body. | Envelope NotebookResponseBody |
| PUT /notebooks/*nid*/*aid* | Adds an annotation aid to the list of annotations of *nid.* | Envelope NotebookResponseBody |
| POST api/notebooks/ | Creates a new notebook. Returns the *nid* of the created Notebook in response’s payload. | Envelope NotebookResponseBody |
| DELETE api/notebooks/*nid* | Deletes *nid*. Annotations stay, they just lose connection to *nid .* | https status, no xml |
| POST api/notebooks/*nid* | Creates a new annotation in *nid*. The content of an annotation is given in the request body. In fact this is a short cut of two actions: POST api/annotations and PUT /notebooks/nid?annotation=*aid*. | Envelope NotebookResponseBody |

## DWAN frontend(s)

### Wired-Marker based frontend

Originally Wired-Marker is freeware developed in Japan as part of the Integrated Database Project sponsored by the Ministry of Education, Culture, Sports, Science and Technology (development code name *ScrapParty*) for supporting the construction of databases. Tool’s concept and design belong to BITS Co., Ltd.[[9]](#footnote-9) and Prof. Okubo[[10]](#footnote-10).

Wired-Marker is licensed under a Creative Common license. This assumes “No-Derivative works” condition, which means that the modified code cannot be distributed. According to the special agreement between BITS Co., Ltd and MPI this condition has been waived.

Wired-Marker as well as its derivative DWAN client, is a Firefox extension that can be used with Firefox versions greater than 2.0.0. One can download the DWAN client as an xpi-file from the DASISH GitHub repository at <https://github.com/DASISH/dwan-client-wiredmarker/releases>. More detailed description on how to install the extension can be found in the Manual at the end of this document. After installation is completed, *DASISH web-annotator* is added to the Firefox menu.

The source code is written in JavaScript and contains XUL files as well. XUL stands for **X**ML **U**ser Interface **L**anguage, which is a [user interface markup language](http://en.wikipedia.org/wiki/User_interface_markup_language) developed by [Mozilla](http://en.wikipedia.org/wiki/Mozilla). XUL is implemented as an [XML](http://en.wikipedia.org/wiki/XML) dialect; it allows for [graphical user interfaces](http://en.wikipedia.org/wiki/Graphical_user_interface) to be written in a similar manner to [Web pages](http://en.wikipedia.org/wiki/Web_page). One of the possibilities to develop Firefox add-ons such as Wired-Marker is to use FoxBeans plug-in in Netbeans IDE. Then one will work with the NetBeans project of type Mozilla Addon.

From the user’s point of view original Wired-Marker is a highlighter that allows marking fragments of a web-document by different colors. An annotated fragment can be a text fragment or the whole image. This feature is present in DWAN client as well. On top of that, an annotated fragment is preserved not only in the local client’s database within the extension but also sent by DWAN client as an XML file to the backend database where it is stored. DASISH developers have implemented synchronization of the local and the backed-database.

The fragment is represented by the XPointer link that consists of the link to the page and the fragment descriptor defining the location of the fragment in original document. The information about the color is represented in the fragment as a CSS-property in the fragment as well.

Another user can view that annotation in his/her DWAN client simply by reloading the annotated page. An annotation made by other instances of DWAN client is listed in the directory of “incoming” annotations on the left-hand side of the browser window.

The corresponding annotated fragment appears as a light-yellow colored fragment.

The original colors, set up by other instances of the client, are not displayed, but they are saved on the database and transferred by the server on GET request of the client. Upgrading the DWAN client so that it will interpret the color correctly, is left to the future work.[[11]](#footnote-11)

Annotations in the “incoming” folder always come from the backend database and are generated by another instance of the DWAN client. Annotations generated by “this” client are distributed across the subfolders (“markers”) of the local folder according to their colors.

In order to access the database and therefore use DWAN and its sets of functionality (e.g. view and post annotations), one needs to log-in. DWAN offers two ways of authentication, via *Shibbolleth* or via Spring basic-authentication. If user’s institution is listed as a Shibboleth Identity Provider, (s)he can use her/his institution credentials by choosing the institution from the list of Identity Providers. Otherwise, the user can create an account by filling up and submitting the registration form referred to at the page <https://lux17.mpi.nl/ds/webannotator-basic/>. After that the user must set up server for working with basic-authentication. For this one goes to DASISH Web Annotator>Settings>Server and sets the link <https://lux17.mpi.nl/ds/webannotator-basic> in the “User Specified” box .

When a user creates an annotation, the client sends it to the server together with a cached representation of the annotated page (on the moment of annotation). A cached representation can be requested by the user later, for instance if the client cannot deliver the annotation because the page has been changed and the fragment cannot be resolved. Consult the manual for more detail. The cached representation is sent as a serialized DOM for the html document. For images only links are sent. The next step would be to zip the HTML, images, CSS and JavaScript for the cached representation. This is done in Wired-Marker, but not posted to the backend for now.

It is possible to annotate an image, but not its fragment. The mouse pointer must be on the image, and the rest of steps are the same as for annotating text. The title and the annotation body are assigned automatically, with the annotation body getting the name of the image file. The title and the body can be edited later.

To editing an annotation one selects it in the list on the left-hand side of the browser window. Next, selecting “Properties” triggers a pop-up form. Altering its corresponding fields and tabs allows editing the annotation body and its title.

In the original Wired-Maker it is not possible to assign and reassign *read*, *write* and *none* access rights for a particular user given a particular annotation. However, the DWAN framework assumes changing access rights. When an annotation is created by DWAN client all registered users except the creator (“owner”) get *read* access. The owner has *write* access and can change the rights of other users. Additional HTML forms are produced by backend server upon the request from the client, filling which allows the owner to reassign the rights for a particular user and an annotation, or change the public access mode for a given annotation.[[12]](#footnote-12)

Working on the turning of Wired-Marker into a DWAN client, the DASISH team has figured out that four of Wired-Marker’s drawbacks cannot be fixed in reasonable amount of time. First, it does not allow to annotate multiple-target annotations that means that a text note can be put exactly on one fragment. For instance one cannot annotate simultaneously two text fragments, by making a remark that they contradict each other. The second drawback is already mentioned: an annotation read from the database loses its original colour while interpreted by the client[[13]](#footnote-13). Third, fragments of images cannot be annotated by Wired-Marker, but only the whole image. Fourth, adding notebooks would demand significant refactoring of the original code. In a sense, coloured directories of local folder can be seen as notebooks.

At the end of this section we sum up the features that have been added to (or changed in) Wired-Marker to adjust it to DASISH requirements:

* design: customization of existing visual features (e.g. sidebar, top menu, right-click menus, add-ons manager view), customization of visual features for extended functionality (login/logout button, extended *Settings* menu for backend configuration);
* functionality: GET, PUT (update), POST, DELETE annotations; POST and GET for cached representations; authentication (login/logout);
* miscellaneous: rewrite of chrome.manifest for development in extension proxy file environment; extension code updates to ensure support by current Firefox versions (Wired-Marker only supports Firefox versions 2.0 – 10.\*, <http://www.wired-marker.org/en/index.html>); hyperanchor (<http://www.hyper-anchor.org/en/technical_format.html>) mapping to xpointer (used on POST/GET) (<http://www.w3.org/TR/xptr-framework>/, <http://www.w3.org/standards/techs/xpointer#w3c_all>)
* getting updated annotation bodies[[14]](#footnote-14).

### Frontend for ELAN

The ELAN frontend for the DWAN backend is being prepared in the context of the COLTIME project. ELAN is an annotation program for media files. Comments in ELAN-speak are called Annotations in DWAN-speak.  Since ELAN already had the notion of *annotations*, in this section we use the word *comment* for DASISH annotations to avoid confusion.

Users can make ELAN annotations on "tracks" parallel to their media, called "Tiers". These annotations refer to time-wise fragments of the media. However, there was no specific support to comment on the ELAN annotations themselves. For instance, several researchers might want to coordinate their work, or review each other's work. Sometimes the tier system of ELAN would be used creatively for this purpose. This however has several drawbacks. For example, annotations on a single tier can't overlap each other time-wise. Multiple comments referring to the same period become cumbersome.

On the other hand, the DWAN backend is an ideal vehicle to store these comments: it is based on comments which refer to some URL, or even more specifically to some fragment of the URL by means of a fragment specifier.

To use this principle, ELAN adds a unique resource identifier to the files it processes, an URN such as urn:nl-mpi-tools-elan-eaf:59d08e6a-5cd9-4aed-8aa4-7074c270e635. This is necessary because ELAN operates on files local to a user's computer, and therefore they have no universally accessible URL.

On the other hand, once an ELAN file is imported into the TLA archive, it will have a stable URL assigned and the online viewer (ANNEX) can use that.

To refer to subsections of the media, fragment identifiers have been introduced. They can refer to a specific time period (#t=1.000/2.000), named tier (#tier=Gebruik) or even to a specific annotation (#anno=a1, using its internal ID).

Currently, there are no cached representations, since it is unclear what form they should take and how to present them to the user. Without a presentation, there is no point in creating them.

### Frontend for ANNEX

ANNEX is an open source online visualizer for time-aligned annotation files, primarily targeted at the EAF (ELAN Annotation Format) format. It powers an ELAN like web-interface, where users can visualize and browse trough the annotations of a time-aligned annotation file in the same fashion as in ELAN, while using a standard (Flash enabled) web-browser.

As with the ELAN frontend for the DWAN backend, ANNEX interaction with the DWAN is being developed under the scope of the COLTIME project.

Given that ANNEX deals with the same type of concepts as ELAN, it follows that also in ANNEX’s context, DWAN annotations are also referred as *comments*.

Primarily being a visualization tool for archived materials, ANNEX currently does not offer any creation or modification functionalities over time-aligned annotation files. This leverages the need for extra functionality allowing users to create comments on the existing annotations.

For this purpose the DWAN backend stands out as an ideal server-side engine to store, search and retrieve such comments. Furthermore, given its web application nature, in the case of ANNEX’s the gap between the desired functionality and the one already offered by the DWAN backend, is rather smaller than for ELAN’s case, since ANNEX’s already relies on URLs and part identifiers to fetch its own data. This consequently eliminates the need for the EAF URN described in the ELAN section of this document and at the same time, enables the use of DWAN cache representations pointing to ANNEX’s URLs, which in turn already support time period (‘time=’ and ‘duration=’) and tier specification (‘tier=’) parameters.

## Testing Procedure

The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The detailed plan, which can be found at <https://github.com/DASISH/dwan-testing>, identifies the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.

Testing is performed at several points in the life cycle as the product is constructed. Testing is a very “dependent'” activity. As a result, test planning is a continuing activity performed throughout the system development life cycle.

The scope of DWAN testing activity includes: server API for DWAN release 1.0 server side software, DWAN release 1.0 client side software for Firefox browser, DWAN User Manual. The scope of this testing activity does not include: DWAN release 1.0 server side software, DWAN development documentation Requirements.

Testing consists of several phases, each phase may or may not include testing of anyone or more of the following aspects of the DWAN software (listed alphabetically): availability, content, functionality, performance, reliability, scalability, security, usability.

The API for the server side software is tested separately with several Python scripts. The client side software is tested manually by following some basic test scenarios.

Testing is performed on the client side with operating system Windows 7, Windows 8, Mac OS X or Linux. For testing of the browser plugin the latest Mozilla Firefox version (29 or later) is used. For the testing of the server API the Python programming environment with the unit testing framework and the package Requests 2.3.0 https://pypi.python.org/pypi/requests/) is used.

All discovered software anomalies during the testing are registered in the project issue management pages under the GitHub <https://github.com/DASISH/dwanclientwiredmarker> and https://github.com/DASISH/dwanbackend.

In general, testing will only stop if the DWAN server becomes unavailable. If testing is

suspended due to the DWAN server becoming unavailable, testing will be resumed once access to the DWAN server is reestablished. Certain individual test cases may be suspended, skipped or reduced if prerequisite tests have previously failed e.g. usability testing may be skipped if a significant number of navigational tests fail.

# Results and Outlook

## Tool’s features

DWAN is the solution for collaborative annotation. It allows annotation of any web-accessible content, both web pages in HTML format and XML documents. With DWAN one can annotate, link data, create relations, and provide feedback. An important feature of DWAN is that created content and sources can be stored in a digital archive, which guarantees their sustainability and persistence. Moreover, the DWAN Framework allows to store a cached copy for each version of the resource, any time a new annotation is made; it is therefore possible to either view the cached copy of the resource or remap the annotation to the updated resource. Archiving all versions of the various annotations created is a crucial feature of DWAN.

DWAN consists of a single backend with a database, hosted by the Max Planck Institute that ensures digital storage for all annotations and related sources, and a RESTful web service. The service is implemented in Java, ready for use by potentially many clients. Different clients, with different interests, and covering different use cases, all have access to the database via a uniform service interface. A client accesses the annotation by means of methods on a REST interface available over HTTP. To call one of the server’s REST methods, the client submits a request on a URL. To access an already existing annotation, the client needs to pass the annotation’s external identifier, which has been generated by the server when the annotation has been added to the database. The REST interface also provides methods to requests all annotations on a resource accessible by a specific user. REST requests define communication between the server and a client. In order to communicate with the server, clients must satisfy certain requirements: first of all, they should be able to send and receive requests in XML format according to the DWAN Schema; then, such requests should also satisfy DWAN’s API patterns.

## Potential front-ends for DWAN.

Tools that can be used with DWAN backend are listed in the following table and their usage is explained in more detail in potential user scenarios below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bookends | Configurational; editorial | bibliographic annotation | Contextualization | text | informal | N |
| LitBlitz Literature Notes Manager | Editorial | bibliographic annotation | Commenting, critical responses and stating preferenes | text | informal | N |
| NoodleTools | Configurational | bibliographic annotation | Commenting, critical responses and stating preferenes | text | informal | Y |
| Projects | Configurational | bibliographic annotation | Contextualization | text | informal | N |
| Qigga | Configurational | bibliographic annotation | Contextualization | text | informal | N |
| Sente | Configurational | bibliographic annotation | Cataloguing | text | informal | N |
| Greenshot | Editorial | image annotation | Commenting, critical responses and stating preferenes | images | informal | N |
| HyperImage | Editorial | image annotation | Linking | images | informal | N |
| NewRadial (INKE) | Configurational | image annotation | Linking | text; image | informal | N |
| Skitch | Configurational | image annotation | Commenting, critical responses and stating preferenes | text; image | informal | N |
| UVic Image Markup Tool | Editorial | image annotation | Commenting, critical responses and stating preferences | images | informal | N |
| Juxta | Configurational; editorial | image annotation; syntax/semantic annotation | Linking | text | formal | N |
| MapHub | Editorial; configutrational | map annotation | Contextualization | geospatial | informal | Y |
| NB | Editorial | PDF annotation | Commenting, critical responses and stating preferenes | text; image | informal | Y |
| Skim | Editorial | PDF annotation | Contextualization | text; image | informal | N |
| iAnnotate | Editorial | PDF annotation | Commenting, critical responses and stating preferences | text; image | informal | N |
| Advene | Editorial | schema definition | Linking | video | informal | Y |
| Anvil |  | schema definition | Commenting, critical responses and stating preferenes | video | informal | N |
| Annotator\'s Workbench | Editorial | segmenting video | Commenting, critical responses and stating preferenes | video | informal | N |
| CLAWS Tagger | Editorial | syntax/semantic annotation | Cataloguing | text | formal | N |
| GATE | Editorial | syntax/semantic annotation | Collaborative tagging | text | formal | Y |
| MMax2 | Editorial | syntax/semantic annotation | Commenting, critical responses and stating preferenes | text | informal | N |
| Melita | Editorial; configutrational | syntax/semantic annotation | Contextualization | text | formal | N |
| Pundit | Configurational | syntax/semantic annotation | Linking | text; image | formal | Y |
| Thinkport Annotator | Editorial | syntax/semantic annotation | Commentin, critical responses and stating preferences | text | informal | Y |
| UAM CorpusTool | Configurational | syntax/semantic annotation | Commenting, critical responses and stating preferences | text | formal | Y |
| Versioning Machine | Editorial | syntax/semantic annotation | Commenting, critical responses and stating preferences | text | informal | N |
| Word Hoard | Editorial | syntax/semantic annotation | Commenting, critical responses and stating preferences | text | formal | Y |
| WordFreak | Editorial | syntax/semantic annotation | Contextualization | text | formal | N |
| brat rapid annotation tool | Editorial; configutrational | syntax/semantic annotation | Contextualization | text | formal | N |
| QDA Miner - Qualitative Data Analysis Software for Qualitative Research | Editorial; configutrational | syntax/semantic annotation; image annotation | Linking; cataloguing | text; image | informal | N |
| Name | task type | task sub-type | process type | asset type | Formal/ informal |  |
| Annotation Graph Toolkit (AGTK) | Configurational | time-series annotation | Cataloguing | text | formal | N |
| VideoANT | Configurational | time-series annotation | Linking | video | informal | N |
| Mediathread | Editorial; configutrational | web media annotation | Linking; cataloguing | text; image; video | informal | N |
| Rehersal Assistant | Editorial | web media annotation | Contextualization | video; audio | informal | N |
| Vertov | Editorial | web media annotation | Commenting, critical responses and stating preferences | text; image | informal | N |
| [A.nnotate.com](http://www.google.com/url?q=http%3A%2F%2FA.nnotate.com&sa=D&sntz=1&usg=AFQjCNHCW2XM4YRxjHiGg5htgLagcKXMdw) | Editorial | web-page annotation | Commenting, critical responses and stating preferenes | text; image | informal | N |
| Annozilla (Annotea on Mozilla) | Editorial | web-page annotation | Commenting, critical responses and stating preferenes | text; image | informal | Y |
| Fleck | Editorial | web-page annotation | Commenting, critical responses and stating preferenes | text; image | informal | N |
| NoteBook | Editorial | web-page annotation | Commenting, critical responses and stating preferenes | text; image | informal | N |
| Project Pad | Editorial; configutrational | web-page annotation | Commenting, critical responses and stating preferenes | text; image; video; sound | informal | N |
| SharedCopy | Editorial | web-page annotation | Commenting, critical responses and stating preferenes | text; image | informal | N |
| Springpad | Configurational | web-page annotation | Commenting, critical responses and stating preferenes; Collaborative tagging | text; image | informal | Y |
| Trailfire | Configurational | web-page annotation | Linking | text; image | informal | Y |
| Pliny | Editorial | web-page annotation; PDF annotation | Commenting, critical responses and stating preferenes | text; image | informal | N |
| Bibliopedia | Configurational; editorial | wiki annotation | Contextualization | text | informal | N |
| FromThePage | Editorial | wiki annotation | Transcription | text | informal | Y |
| ANNIS | Editorial |  | Contextualization | text | formal | N |
| Annotator | Editorial |  | Commenting, critical responses and stating preferenes | text; image | informal | Y (can be stored in Annotea) |
| Annotorious | Editorial |  | Commenting, critical responses and stating preferenes | video | informal | Y (via OKF) |
| [Atlas.ti](http://www.google.com/url?q=http%3A%2F%2FAtlas.ti&sa=D&sntz=1&usg=AFQjCNEzCxeB-0hVKhmoTMXq23ju8syo1Q) | Synthetic |  | Contextualization | text; image | informal | N |

User scenario 1: *Bibliographic annotation.*

Review of tools available: LitBlitz Literature Notes Manager, NoodleTools, Projects, Oigga, Sente. All but one of these are configurational, i.e. that they tend to support the organization and ordering of database records, rather than the annotation of those records with further information.

Scenario: a user has a bibliography they have formed over five years of research, on a specific geographic area. In this case the bibliography is the archaeology of Cyprus in the Byzantine period. Each bibliographic reference is the authority for a particular spelling of a particular place-name, e.g. “Paphos” as opposed to “Pafos”. The user wishes to use their bibliographic resource to annotate place-name references in the third-party document with their bibliography. This may be viewed as ‘enhanced citation’.

Formal/informal: The annotations of the text is a formal annotation requirement, as the third party text is being annotated with pre-existing information. The annotations of the bibliography are informal, as they provide free text information on each individual item.

Asset: The asset is purely textual. Previously the researcher had kept it in a Word document on their local hard-drive but recently, as one of the outputs of a research project, they have published it online as part of an inventory, marked up in XML, of Byzantine monuments in Cyprus. It is available on a webpage as a list of publications with author, title, periodical title (if appropriate), date of publication and page reference.

Annotations take the form of links to the bibliographic records in the researcher’s database, and also the annotations they have made on the bibliographic records. The latter might include ‘is this reference up to date’ or ‘is it being cited in agreement or disagreement’.

The annotations in the bibliography should be able to link simultaneously to multiple bibliographic references.

Necessary functions:

* Highlight text, placing markers on particular publications as aides-memoire for publication they are working on. This would be whole records/paragraphs rather than individual words.
* They may also wish to Add comments in the form of scribbled notes.
* They may wish to Share selected parts of the original resource via email, Twitter, and Facebook, although email is likely to be far the most useful of these, as they will wish to share references to their bibliography with individual colleagues.
* Enhance text with links. Using records in the bibliography to annotate sections of text in a second document. This would be done by embedding hyperlinks in the second document, pointing back to the bibliography records.
* In the application therefore, the third party text is annotated twice, first with the bibliography and second with the annotations of the bibliography. Both types are displayable in hover-over boxes on the third party document.

User scenario 2: *Image annotation*.

Review of tools available: Greenshot, HyperImage, NewRadial (INKE), Skitch, UVic Image Markup Tool. These tools are both configurational and editorial. This reflects the need to both organize image collections with annotations, and to link comments/notes with them.

Scenario: User has downloaded a large (1000+) image collection from www.flickr.com/commons. It is themed around European cultural heritage in the nineteenth and twentieth centuries, containing primarily images of objects from museums, but also contains images documenting specific events. These could include major political events such as those connected to WW1, or scenes from everyday life and objects (see example from the University of Reading’s Museum of English Rural Life).

This scenario is applicable to scholars, but also, potentially, to museum and collections curators.

Formal/informal: Mostly, the functionalities required are informal. The main need is to support the user in providing commentaries on individual images, and to select particular parts of particular images for specific commentary on those specific parts. However, the user may also wish to construct formal lists/taxonomies of the various aspects depicted. These could include objects (e.g. teapots, statues, vases, weapons, vehicles), time periods, and locations. Asset: the assets are images, stored either locally in the user’s computer, or in a private cloud space.

Necessary functions:

* The primary function needed is to Add comments in the form of scribbled notes (text to image). Either the user will wish to tag entire images or selected parts. In the example below, they will wish to define a particular part of the image, and associate tags and/or full text comments with these. In the example given, this might include ‘steam tractor’, ‘hat’, ‘person’, and ‘building.
* The user is likely to wish to share selected parts of the original resource via email, Twitter, and Facebook. In the case of a scholar, they wish to share only by email. In the case of a curator, or public engagement professional, they may wish to share via social media, e.g. using the #AskACurator or #MuseumsWeek hashtags. To do this, they will have to Save their own annotations locally.
* It will be necessary to Track versions of annotations.
* The user will wish to Tag a whole images with keywords. This functionality is already supported by www.flickr/com/commons, so the use of the Flickr API would be more appropriate than the construction of new system.
* They should have the ability to embed bibliographic references in the annotations. They could then, for example, connect related entries from the V&A catalogue in London (http://collections.vam.ac.uk), treating each collection entry as a bibliographic entity.

User scenario 3. *Web page annotation*.

Review of tools available: Mediathread, Rehersal Assistant, Vertov, A.nnotate.com, Annozilla (Annotea on Mozilla), Fleck, NoteBook, Project Pad, SharedCopy, Springpad, Trailfire. All but three of these tools are editorial. This reflects the fact that browser-based bookmarking and generic services such as https://delicious.com are adequate to meet most researchers’ needs for organizing collections of web pages, the need for editorial, comment-based annotation is far more acute.

Scenario: User is researching methods used in 3D reconstruction of archaeological sites and objects. They have a need to both define and add annotations to a variety of different web pages, especially results of searches using Google Images and Google Scholar. Specifically they are interested in linking data created in the Unity 3D modelling package with Geographic Information Systems (GIS) data. They therefore need to compile a profile of web resources which refer to this issue. They are leading on this task in a collaborative team, and thus need to share their annotations with colleagues remotely, and with research students. These colleagues will need to be able to add annotations as well, and formulate replies to existing annotations.

Formal/informal: this is an informal referencing requirement, as the researcher will only be adding new information in the form of annotations.

Assets: the assets are primarily text and images, but may also include video. They are not stored locally.

Examples include: Official advice from Unity (http://unity3d.com/learn/resources/talks/gis-terrain-unity),

Q&A threads (http://answers.unity3d.com/questions/17829/how-can-i-import-

gis-data-into-a-unity-project.html) and bibliography (http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5567608&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs\_all.jsp%3Farnumber%3D5567608).

Necessary functionalities:

* Firstly, the use will need to Save their own annotations in the form of Add comments in the form of scribbled notes (text to text and text to image).
* These are stored in a shared collaborative space. The annotations will need to contain metadata detailing the page URL and the part of the page being referred to. It will be necessary to specify start and end points, allowing the user to Highlight text and Highlight images. For this scenario, it will not be necessary to highlight parts of images.
* Each annotation will have to be able to point to multiple parts or the same web page, or to multiple web pages.
* In a shared collaborative environment, it will be necessary to Track versions of annotations, including responsibility for different versions.
* This scenario reflects the probability that collaborative annotation is likely to be of (scholarly) use only within relatively well defined groups of researchers working on a common task. The tools overview suggests that there is less demand for community-wide annotation applications.

User scenario 4: *Syntactic and Semantic annotation*.

Review of tools available: CLAWS Tagger, GATE, MMax2, Melita, Pundit, Thinkport Annotator, UAM CorpusTool, Versioning Machine, Word Hoard, WordFreak, brat rapid annotation tool, QDA Miner – Qualitative (Data Analysis Software for Qualitative Research).

Text annotation, both structured (syntactic) and unstructured (semantic) is a fundamental part of the research process in most disciplines. It is by far the most common form of annotation currently carried out by humanities scholars, and supported by the current tools offering. The tools above therefore support a range of configurational and editorial tasks.

Scenario: User (a Latinist and historian) is creating a digital critical edition of Marcus Tullius Cicero’s judicial speeches. They have downloaded the fifty-two surviving examples from the Perseus Digital Library (http://www.perseus.tufts.edu/hopper) and stored them locally.

Formal/informal: Informal annotations are critical here, to add context, historical allusions, biographical notes on persons mentioned and places referred to. However formal annotation methods may also be required, especially in support of automated parsing and natural language processing (NLP). However, much of this information will be already be available as TEI XML markup in the Perseus documents.

Necessary functionalities:

* A primary function is to be able to Highlight text that is relevant to a) particular arguments made by Cicero, important passages and references to important exchanges. It will also be necessary to highlight quotations which have significance in other contexts. They will also wish to highlight important general entities (see below).
* One the text is highlighted, the user will wish to Add comments in the form of scribbled notes (text to text). As well as free text, they will wish to construct annotations using their own vocabulary lists of important general entities. These will include, but not exhaustively: important personages, such as Caesar, Sextus, Pompey, contemporary events such as the formation of the First Triumvirate and the Civil, places such as Rome, Brundisium, roles such as aedile and senator, laws. Any word, phrase or passage that the user wishes to associate with these events would need to be defined and an associative term or terms selected.
* Assuming the critical edition will involve translation or all or part of the corpus, the user will need to annotate any passages where the translation is, for any reason, indirect.
* It will be essential for the user to be able to Track versions of annotations, and to be able to delete obsolete versions.
* They will need to be able to Save their own annotations.
* The user will need to be able to Modify text: Add information to text (within the text) as well as delete information (within the text) if, in their judgement, there is repetition or trantextual inaccuracy, or if abridgement is needed for any other reason. The deletion, and the text deleted, should be preserved as an annotation.
* The user will need to be able to embed links to other texts, bibliography, video and image media.

User scenario 5. *Wiki based annotations*.

Review of tools available: Bibliopedia, FromThePage. The requirements for wiki based annotation are similar to those required for web page annotation. However, there is an additional requirement to capture and annotate changes made to the wiki pages over time. Both available tools have primarily editorial functions.

Scenario: User is conducting a project to capture the reception of public monuments, including the Parthenon in Athens. They will therefore need to annotate not only the main page of the wiki, but also the ‘Talk’ history of the page, and are likely, later on, to have edits/additions to make to the Wikipedia page itself. The project is therefore about using annotation to capture discussion about a contentious page, and Formal/informal: only informal annotations are relevant here.

Assets: The assets involved are text and images.

Necessary functionalities:

* UC4: Modify text: Add information to text (within the text).
* UC 5: Modify text: delete information (within the text).
* The use will need to Save their own annotations in the form of Add comments in the form of scribbled notes (text to text and text to image).
* These are stored in a shared collaborative space. The annotations will need to contain metadata detailing the wiki URL and the part of the page being referred to. It will be necessary to specify start and end points, allowing the user to Highlight text and Highlight images. For this scenario, it will not be necessary to highlight parts of images.
* Each annotation will have to be able to point to multiple parts or the same wiki page, or to multiple web pages.
* In a shared collaborative environment, it will be necessary to Track versions of annotations, including responsibility for different versions.
* To gauge discussion on the topic, there is an important requirement to be able to share selected parts of the original resource via email, Twitter, and Facebook.

# APPENDIX

## Schema

There are 5 sorts of resources in DASISH: *CachedRepresentation*, *Target*, *Principal*, *Annotation*, *Notebook*. Each of them has the corresponding xsd-type in the schema. There is no type with the name *CachedRepresentation* because a cached representation is a "pure" resource like an image or a text file that does not contain any meta-information about itself. The metadata of a cached presentation are defined via an instance of *CachedRepresentationInfo* type.

Each of resource types has an obligatory attribute "id" which contains DASISH identifier pointing to the location of the resource on the DASISH server. Resource-info types *TargetInfo*, *AnnotationInfo*, *NotebookInfo* contain reference to the corresponding resource plus the most important information about the resource. There are corresponding list-of-resource-info types: *TargetInfos*, *AnnotationInfos*, *NotebookInfos*.

<?xml version="1.1" encoding="UTF-8"?>

<xs:schema targetNamespace="http://www.dasish.eu/ns/addit"

xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"

xmlns:dasish="http://www.dasish.eu/ns/addit">

<xs:import namespace="http://www.w3.org/XML/1998/namespace"

schemaLocation="http://www.w3.org/2005/08/xml.xsd"/>

<xs:complexType name="List">

<xs:sequence/>

</xs:complexType>

<xs:complexType name="ReferenceList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="href" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<xs:complexType name="CachedRepresentationInfo">

<xs:sequence>

<xs:element name="mimeType" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="tool" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="type" type="xs:string" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

<xs:attribute ref="xml:id" use="required"/>

</xs:complexType>

<!-- used in the target -->

<xs:complexType name="CachedRepresentationFragment">

<xs:sequence>

<xs:element name="fragmentString" type="xs:string" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

</xs:complexType>

<xs:complexType name="CachedRepresentationFragmentList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="cached" type="dasish:CachedRepresentationFragment" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<xs:complexType name="Target">

<xs:sequence>

<xs:element name="lastModified" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>

<xs:element name="link" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>

<xs:element name="version" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="siblingTargets" type="dasish:ReferenceList" minOccurs="1"/>

<xs:element name="cachedRepresentatinons" type="dasish:CachedRepresentationFragmentList"

minOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

<xs:attribute ref="xml:id" use="required"/>

</xs:complexType>

<xs:complexType name="TargetInfo">

<xs:sequence>

<xs:element name="link" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>

<xs:element name="version" type="xs:string" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

</xs:complexType>

<xs:complexType name="TargetInfoList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="targetInfo" type="dasish:TargetInfo" minOccurs="0"

maxOccurs="unbounded"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<xs:complexType name="Principal">

<xs:sequence>

<xs:element name="displayName" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="eMail" type="xs:string" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

<xs:attribute ref="xml:id" use="required"/>

</xs:complexType>

<xs:complexType name="CurrentPrincipalInfo">

<xs:sequence>

<xs:element name="currentPrincipal" type="xs:boolean" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

</xs:complexType>

<xs:complexType name="CurrentPrincipalInfoList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="currentPrincipalInfo" type="dasish:CurrentPrincipalInfo" minOccurs="0"

maxOccurs="unbounded"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<xs:simpleType name="Access">

<xs:restriction base="xs:string">

<xs:enumeration value="read"/>

<xs:enumeration value="write"/>

<xs:enumeration value="none"/>

</xs:restriction>

</xs:simpleType>

<xs:complexType name="Permission">

<xs:attribute name="principalHref" type="xs:anyURI" use="required"/>

<xs:attribute name="level" type="dasish:Access" use="required"/>

</xs:complexType>

<xs:complexType name="PermissionList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="permission" type="dasish:Permission"

minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

<xs:attribute name="public" type="dasish:Access" use="required"/>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<xs:complexType name="Annotation">

<xs:sequence>

<xs:element name="ownerHref" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>

<xs:element name="headline" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="lastModified" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>

<xs:element name="body" type="dasish:AnnotationBody" minOccurs="1" maxOccurs="1"/>

<xs:element name="targets" type="dasish:TargetInfoList" minOccurs="1" maxOccurs="1"/>

<xs:element name="permissions" type="dasish:PermissionList" minOccurs="1"

maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

<xs:attribute ref="xml:id" use="required"/>

</xs:complexType>

<xs:complexType name="AnnotationInfo">

<xs:sequence>

<xs:element name="ownerHref" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>

<xs:element name="headline" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="lastModified" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>

<xs:element name="targets" type="dasish:ReferenceList" minOccurs="1" maxOccurs="1"

/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

</xs:complexType>

<xs:complexType name="AnnotationInfoList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="annotationInfo" type="dasish:AnnotationInfo" minOccurs="0"

maxOccurs="unbounded"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<xs:complexType name="AnnotationBody">

<xs:choice>

<xs:element name="textBody">

<xs:complexType>

<xs:sequence>

<xs:element name="mimeType" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="body" type="xs:string" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="xmlBody">

<xs:complexType>

<xs:sequence>

<xs:element name="mimeType" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:any minOccurs="1" maxOccurs="1" processContents="skip"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:choice>

</xs:complexType>

<xs:complexType name="Notebook">

<xs:sequence>

<xs:element name="ownerRef" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>

<xs:element name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="lastModified" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>

<xs:element name="annotations" type="dasish:ReferenceList" minOccurs="1" maxOccurs="1"/>

<xs:element name="permissions" type="dasish:PermissionList" minOccurs="1"

maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

<xs:attribute ref="xml:id" use="required"/>

</xs:complexType>

<xs:complexType name="NotebookInfo">

<xs:sequence>

<xs:element name="ownerHref" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>

<xs:element name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="href" type="xs:anyURI" use="required"/>

</xs:complexType>

<xs:complexType name="NotebookInfoList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="notebookInfo" type="dasish:NotebookInfo" minOccurs="0"

maxOccurs="unbounded"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<!--- Envelopes -->

<xs:simpleType name="AnnotationActionName">

<xs:restriction base="xs:string">

<xs:enumeration value="CREATE\_CACHED\_REPRESENTATION"/>

</xs:restriction>

</xs:simpleType>

<xs:simpleType name="PermissionActionName">

<xs:restriction base="xs:string">

<xs:enumeration value="PROVIDE\_PRINCIPAL\_INFO"/>

</xs:restriction>

</xs:simpleType>

<xs:complexType name="Action">

<xs:sequence>

<xs:element name="object" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>

<xs:element name="message" type="xs:string" minOccurs="0" maxOccurs="1"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="ActionList">

<xs:complexContent>

<xs:extension base="dasish:List">

<xs:sequence>

<xs:element name="action" type="dasish:Action" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<!-- response envelope (not a resource, used for all response on POST/PUT requests) -->

<!-- "envelope"-->

<xs:complexType name="ResponseBody">

<xs:sequence>

<xs:choice>

<xs:element name="annotation" type="dasish:Annotation"/>

<xs:element name="permissions" type="dasish:PermissionList"/>

<xs:element name="notebook" type="dasish:Notebook"/>

</xs:choice>

<xs:element name="actionList" type="dasish:ActionList" minOccurs="1"

maxOccurs="1"/>

</xs:sequence>

</xs:complexType>

<!-- ############ ELEMENTS (used by JAXB) ################# !-->

<xs:element name="action" type="dasish:Action"/>

<xs:element name="actionList" type="dasish:ActionList"/>

<xs:element name="annotation" type="dasish:Annotation"/>

<xs:element name="annotationBody" type="dasish:AnnotationBody"/>

<xs:element name="annotationInfo" type="dasish:AnnotationInfo"/>

<xs:element name="annotationInfoList" type="dasish:AnnotationInfoList"/>

<xs:element name="annotationList" type="dasish:ReferenceList"/>

<xs:element name="cashedRepresentationInfo" type="dasish:CachedRepresentationInfo"/>

<xs:element name="cashedRepresentationList" type="dasish:ReferenceList"/>

<xs:element name="cachedRepresentationFragment" type="dasish:CachedRepresentationFragment"/>

<xs:element name="cachedRepresentationFragmentList" type="dasish:CachedRepresentationFragmentList"/>

<xs:element name="list" type="dasish:List"/>

<xs:element name="notebook" type="dasish:Notebook"/>

<xs:element name="notebookInfo" type="dasish:NotebookInfo"/>

<xs:element name="notebookInfoList" type="dasish:NotebookInfoList"/>

<xs:element name="notebookList" type="dasish:ReferenceList"/>

<xs:element name="permissionList" type="dasish:PermissionList"/>

<xs:element name="responseBody" type="dasish:ResponseBody"/>

<xs:element name="target" type="dasish:Target"/>

<xs:element name="targetInfo" type="dasish:TargetInfo"/>

<xs:element name="targetInfoList" type="dasish:TargetInfoList"/>

<xs:element name="targetList" type="dasish:ReferenceList"/>

<xs:element name="principal" type="dasish:Principal"/>

<xs:element name="currentPrincipalInfo" type="dasish:CurrentPrincipalInfo"/>

<xs:element name="currentPrincipalInfoList" type="dasish:CurrentPrincipalInfoList"/>

<xs:element name="principalList" type="dasish:ReferenceList"/>

<xs:element name="referenceList" type="dasish:ReferenceList"/>

<xs:element name="access" type="dasish:Access"/>

<xs:element name="annotationActionName" type="dasish:AnnotationActionName"/>

<xs:element name="permission" type="dasish:Permission"/>

<xs:element name="permissionActionName" type="dasish:PermissionActionName"/>

</xs:schema>

## Wired-Marker-based DWAN frontend. Manual

This DWAN client is based on the Wired-Marker extension for Firefox with added functionality to communicate with the DWAN backend. Such extension enables the user to create free-text annotations on fragments of webpage content. Wired-Marker, and therefore DWAN client, runs only on the Firefox web browser. It can be downloaded from the Mozilla website, here: <http://www.mozilla.org/en-US/firefox/new/> .

It is highly recommended to make a separate Firefox profile where one installs the DWAN Wired-Marker based client. For instance, to create a Firefox profile on MAC, one can follow one of two methods. In either case it is recommended to watch the Terminal window output in order to trace any heavy exceptions or errors that might occur. One can create a profile via Terminal by using the command

*mkdir -p ~/Library/Application\ Support/Firefox/Profiles/nameofprofile*

The instance of the Firefox with the given profile can by launched by

*/Applications/Firefox.app/Contents/MacOS/firefox -profile ~/Library/Application\ Support/Firefox/Profiles/nameofprofile -no-remote*.

Alternatively, one can create a profile with the help of Profile Manager and start Firefox with that profile by:

*macuser$ /Applications/Firefox.app/Contents/MacOS/firefox-bin -p*

Having started up a Firefox instance with the profile, drag and drop the xpi file onto the Firefox window in order to install it. Another option is to load and install the xpi file by using the Firefox menu “File - Open File” dialog. [[15]](#footnote-15)

In order to access the database and therefore use DWAN you need to log-in. DWAN offers two ways of authentication, via *Shibbolleth* (A) and with a newly and specifically created user account (B).

A) If your institution is listed as a Shibboleth Identity Provider (IP), then you can just use your institution credentials. Choose from the list of Identity Providers, select, and log in.

B) If your institution is not listed as a Shibboleth IP, then you can create a user account following the following steps:

1. go to <https://lux17.mpi.nl/ds/webannotator-basic/>

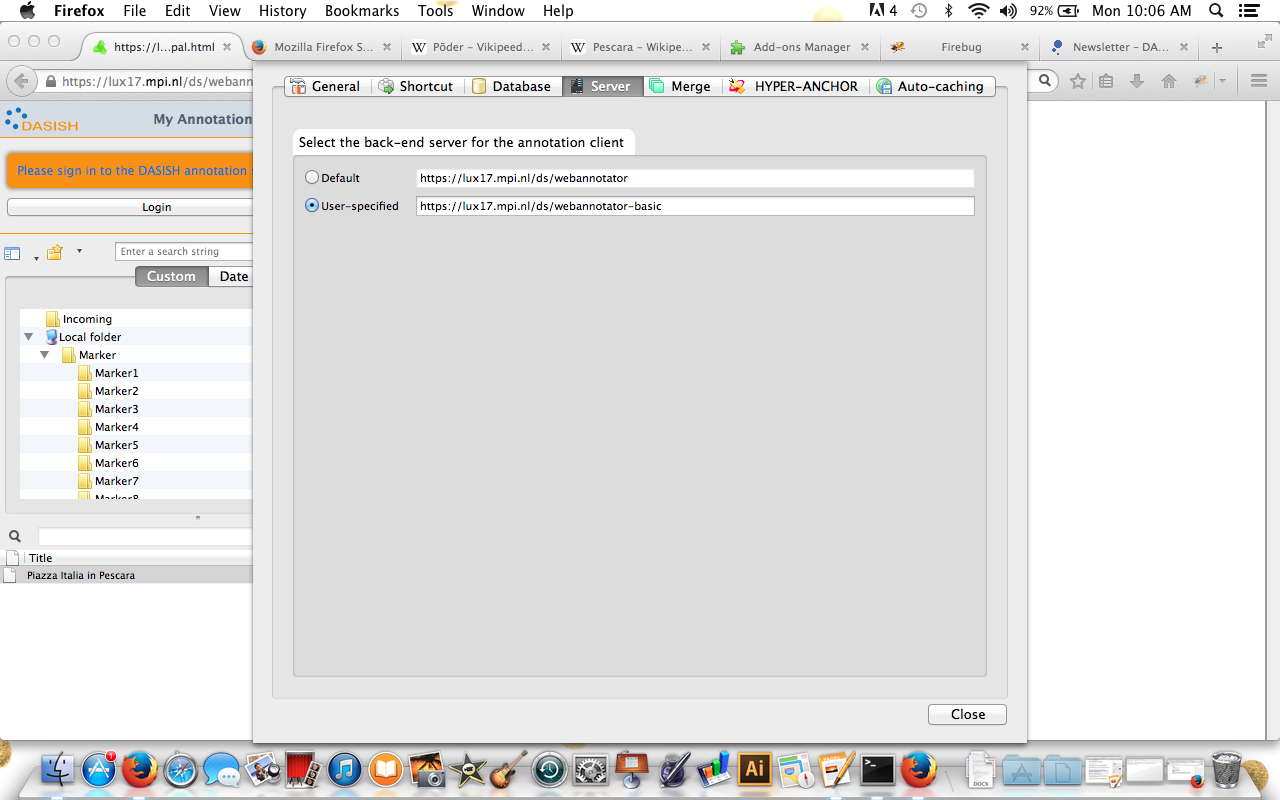
2. click on *Register as a non-Shibboleth user*

3. fill in the user registration form and submit it

4. press cancel

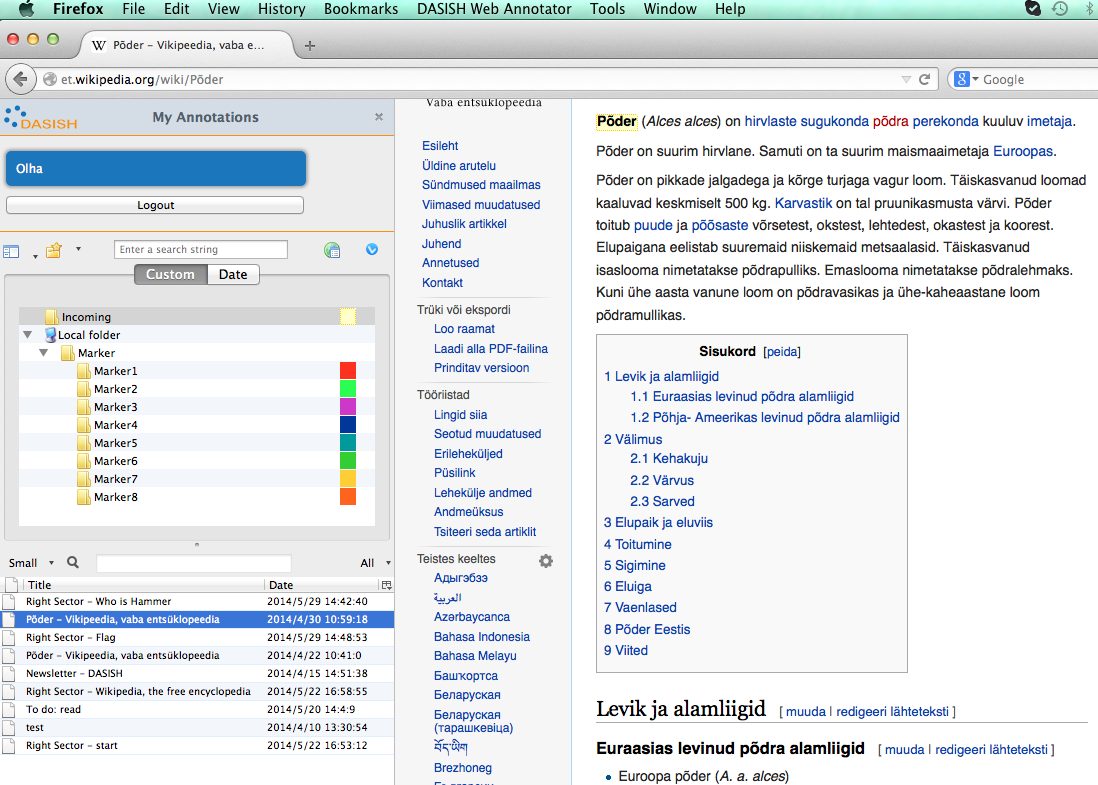
5. go to DASISH Web Annotator > Settings > Server > write this link <https://lux17.mpi.nl/ds/webannotator-basic> in the User Specified box > and close

6. Log in using the left side Log-in box

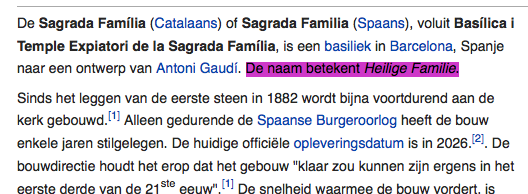


Annotations created on other clients or by other users are all listed in the **Incoming** folder, in the left side box. DASISH website is the default webpage. Browse the page you are interested in and where are the annotations you want to see. The full list of annotations will appear and they can be ordered by title or date. Please note, it is not possible to see the author of the annotations.

To see annotations from the other users, click on the annotation you want to see from the full list. It will appear on the webpage in light yellow.



To see the user’s own annotations, (s)he can also go to the **Marker** folder and click on the colour used to make the annotation.



If one or more annotations do not appear after clicking on them and also after refreshing the page, it means that the client cannot resolve the annotated fragment. The most probable reason for this is that the webpage has been changed since it was annotated.

However, one of the main features of DWAN is that it is still possible to see the annotations even if the webpage has changed by viewing the cached representations. To do this, point the mouse to the annotation in question and right-click. In the pop-up menu select "*Cached representations*" and click "*open remote cache*" in the sub-menu. If you compare this example cached representation with the current webpage, you will be able to see the annotations.

To make an annotation, the user goes to the webpage (s)he want to annotate, selects a text fragment with the mouse, right clicks and chooses the marker colour (s)he would like to use from the menu.



Nest, one needs to select the colour by left-clicking the mouse, and fill in the fields in the pop-up text-box to make the annotation. One assigns a distinctive title to the annotation and write a clear short description in the annotation box. To save the annotation, one clicks “ok” after filling in the form in the pop-up text box and it will show on the webpage.

To update the annotation, one picks it up in the list, right clicks and selects “Properties” in the menu. The form for editing will appear where by selecting tabs “Brief Overview” or “Annotation” one can edit the title and the text body respectively.



While it is not possible to delete other users’ annotations, the user can delete his/hers.

One finds the annotation to delete in the Local Folder, then right clicks on it and select Delete.

When an annotation is created by DWAN client all registered users except the creator (“owner”) get *read* access. The owner has *write* access and can change the rights of other users. Additional html forms are produced by the backend server upon the request from the client, filling which allows the owner to reassign the rights for a particular user and an annotation, or change the public access mode for a given annotation.

Users with advanced technical skills can also examine the relationship between the Backend and the Frontend directly by installing Firebug, another Firefox add-on. This is particularly useful to analyse DWAN in situations where it does not seem to behave properly. To install Firebug, go to Tools > Add-Ons > Search for Firebug and allow installation.

In cases where a clean reinstall of a Firefox add-on is needed, it is advisable to first remove the respective Firefox add-on via the Add-ons Manager that Firefox provides (about:addons) and then delete any add-on related directories contained within the Firefox profile directory of the Firefox profile that is used (e.g. "DWAN" or "WiredMarker" directories). Please note that there might be different profiles, and thus different profile folders on your machine.

Mac OS X: ~/Library/Application Support/Firefox/Profiles/[profile folder

name]

For Windows please cf. <http://kb.mozillazine.org/Profile_folder_-_Firefox>.

1. The redirection to these forms must be implemented in frontend, though. See <https://github.com/DASISH/dwan-backend/issues/9> and  <https://github.com/DASISH/dwan-client-wiredmarker/issues/22> [↑](#footnote-ref-1)
2. Hopefully this can be fixed in the frontend. See: <https://github.com/DASISH/dwan-client-wiredmarker/issues/16> [↑](#footnote-ref-2)
3. The current front-end is not able to present the annotations on xml's properly (as Olof has suggested), but they do exist in a good shape on the backend.  See <https://github.com/DASISH/dwan-client-wiredmarker/issues/18>.   If fixing it is problematic a jsp page for "advanced users", where an annotation on an annotation can be made or obtained, could be added. <https://github.com/DASISH/dwan-backend/issues/10> [↑](#footnote-ref-3)
4. The Language Archive, Max Planck Institute for Psycholinguistics, <http://tla.mpi.nl/> [↑](#footnote-ref-4)
5. http://dublincore.org/documents/2010/10/11/dcmi-type-vocabulary [↑](#footnote-ref-5)
6. http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/dao.html [↑](#footnote-ref-6)
7. Java Database Connectivity [↑](#footnote-ref-7)
8. Implemented, however testing is not completed and it is not used in the current DWAN frontend. [↑](#footnote-ref-8)
9. BITS Co., Ltd. <http://www.bits.cc> [↑](#footnote-ref-9)
10. Prof. Okubo [↑](#footnote-ref-10)
11. Update this part of the text if the bug is fixed. [↑](#footnote-ref-11)
12. TODO implemented but not deployed neither on lux16, nor on lux17 [↑](#footnote-ref-12)
13. Correct it if the problem is fixed. [↑](#footnote-ref-13)
14. Implementation to be added [↑](#footnote-ref-14)
15. Siiri (Tartu) can write on creating profile in windows. [↑](#footnote-ref-15)