Data Service Infrastructure for the Social Sciences and Humanities

EC FP7
Grant Agreement Number: 283646

Deliverable Report

Deliverable: D5.4
Deliverable Name: DASISH Web Annotation (DWAN) framework

Task Leader: Olha Shkaravska, 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), Przemek Lenkiewicz (MPG-TLA), Kees-Jan van de Looi (MPG-TLA), Olof Olsson (UGOT), Stephanie Roth (UGOT), Olha Shkaravska (MPG-TLA), Menzo Windhouwer (MPG-TLA).

SEVENTH FRAMEWORK PROGRAMME

1. Table of Contents

2. Executive Summary

The availability of digital archives and other research data via the Internet creates new chances for collaboration. Indeed, equipped with special software, researchers from different institutions, countries and fields can work together via the Internet. Such collaboration can take the form of annotating the on-line 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 provide a specification of the framework for annotating web-documents developed according to task 5.6 plan. In this context an annotation is a remark over a fragment(s) of an on-line document(s).

From the technical point of view the proposed framework consists of one *back-end*, constituted of the server software and the database, and possibly multiple *front-ends* (clients). Developed within DASISH project the DWAN tools are an instance of the DWAN framework. It consists of the back-

end part and the client, which is a significantly adjusted version of the *Wired Marker*[1] Firefox extension. We chose Wired Marker after a selection process, looking for a suitable tool as a general DWAN client for annotating objects on the Web. The selection process is separately described later.

The core of the back-end is a database where annotations and information about corresponding annotated target documents are stored together with the targets' cached representations. Archiving cached representations in the database is relevant when annotated documents are dynamic pages like news sites or wiki-pages under construction.

A client in the DWAN framework exchanges data with the server by sending REST[2] requests and getting responses. Client-request bodies and server's responses have a form of XML. 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.

The work in task 5.6 succeeded in delivering, next to the back-end, one front-end (client) tool for the DWAN framework, and in collaboration with other projects, integrated two extra client tools. For future work, this task also found a number of other tools from the Humanities domain that looked promising to integrate in DWAN. In separate chapters, we present, an analysis of the annotation task of DWAN clients in the context of a general overview of Humanities tools and Humanities research workflow. We present also some user-scenarios that could be fulfilled by DWAN, either by the current version or with some future development.

3. Introduction to the DWAN framework

3.1 Motivation and goals

In the last decades, next to the ever-growing amounts of data on the web, we have also 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 parts 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 or resources in domain specific formats such as transcriptions originally created by linguistic software, e.g. EAF-files created by the ELAN multi-media annotation tool.[3] To bring in the collaborative element such annotations should be shareable between different (groups of) users and if editable by different tools with (domain) specific capabilities.

[1] https://addons.mozilla.org/nl/firefox/addon/wired-marker/

[2] http://en.wikipedia.org/wiki/Representational_state_transfer

[3] https://tla.mpi.nl/tools/tla-tools/elan/elan-description/

Based on these ideas of shareable annotations that can be worked on by different domain specific tools we have set ourselves two goals. Addressing the first goal, we have come up with the one-server-many-clients architecture, see section 5.1 for more detail and Figure 1 General DWAN Architecture. Indeed, the server with the database is used to store annotations, which all have the same structure independent on annotatable documents. This structure mirrors an annotation itself (e.g. a text comment), a reference to the sources with the annotated fragment specifications, and, possibly, references to cached copies of the annotated documents, see section 5.2. This uniformity opens a possibility to design one database that stores annotations for different type of documents. On the client side the situation is different in general.

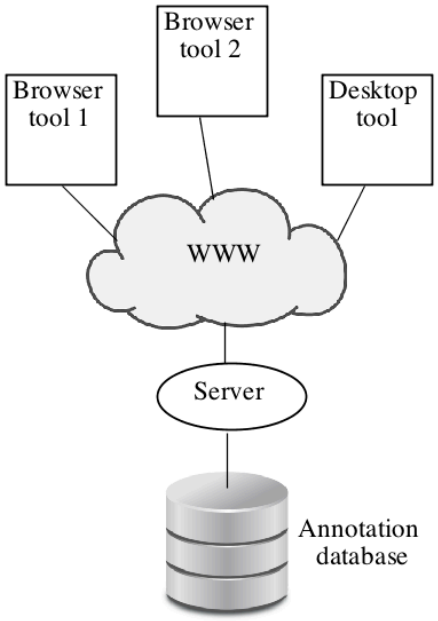

Figure 1 General DWAN Architecture

There one must take into account the variety of annotatable objects because they have different internal structure and client software must technically overwork the internal structure of a document when creating an annotation for it. Therefore a specific client is to be designed for a specific type of documents. For instance, annotating web pages and ELAN files need technically different approaches due to the completely different internal structure of the corresponding annotated documents.

Addressing the second goal, we have developed the DWAN annotating tool, which by now consists of the server software with the database and the Wired-marker-based DWAN client. Moreover, specific clients have been designed for ELAN files. The DWAN back-end and the developed clients are discussed in detail in sections 5.3 and 5.4 respectively.

3.2 Requirements and user scenarios.

As stated above, in the DWAN framework it is assumed that possibly multiple clients communicate with a single back-end consisting of the server software which implements access to the database with annotations. Annotations and information about annotated on-line resources (*targets*) are stored in the database, 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 retrieving 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. This may happen when the corresponding fragment has been significantly changed or disappeared.

The client and server must “understand” each other and therefore follow some uniform rules. In a nutshell, there are two such rules. The first one: exchange data by sending http(s)-requests from a given finite collection of requests that the server understands. The second rule: the content of requests and responds must obey a DWAN XML schema, which is a part of the server-side software. The DWAN XML Schema mirrors a data model (see section 5.2) that has been designed to represent the main data classes (*annotation*, *target*, *principal*, *cached representation* and *notebook*) and relations between these classes.

As a proof-of-the-concept for the architecture design and its technical approach we needed to develop not only back-end software but also one or more client tools that work with it. Moreover, such clients must be usable by wide communities of researchers. Before developing a client, we, first, needed to determine which user scenarios it should cover, and second, to investigate a suitable tool already exists that can be used as DWAN client prototype. If it does not fully fit into the infrastructure, the tool must be adjustable to fit the architecture and to cover the user scenarios.

The simplest and the most obvious user scenario can be called *Login and Annotate*. A principal[4] logs in and sees the lists of annotations that were made by him and his colleagues earlier. These annotations are sorted by their dates or by their headers. The principal finds a web page that he wants to annotate, selects a fragment of the text to annotate, say, by marking it with some color, and attaches a text note on this fragment. The text note should not clutter the main document. By clicking the mouse, the annotation can be saved in the local (client’s) database as well as in the server database. It should be visible on the web page as visualized by the client tool.

Second scenario. *Editing and deleting*. The principal must be able to edit the text note, to change the header of an annotation, and to give different access rights (read, write, none) for another specific principal.

Third scenario. *Retrieving cached representations*. The principal logs in, sees the list of annotations and selects the one he wants to inspect in the context of the corresponding web page. He clicks on the annotation in the list, a try a few times to reload the page, but the annotation does not appear. The client 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 made on this page earlier. Indeed, it can be seen that the page has been updated. It is worth to note that this scenario was a part of the DWAN demo during LREC 2014[5]. 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 he has the Wired-Marker-based DWAN front-end installed.

It turns out that by the time the DWAN developers team started to work on the client, the Wired-Marker Firefox extension was the open-source the tool that could cover these scenarios, except that there was no connection with the central database and the annotations made via other clients were not retrievable. However code inspection gave the impression that this feature could be added. In the next section we give the comparative analysis of the tools, which could have been used a DWAN client prototypes, in more detail.

[4] The term *principal* in general denotes either a user or a group of users. At present, user and principal are synonyms for DWAN tool.

[5] http://lrec2014.lrec-conf.org/en/

4. Annotation Tools

4.1 Potential DWAN-client prototypes: 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) used as a starting point for the 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 back-end, and platform-independency. The Table 1 represents the results of the investigation.

Table 1. Annotation tools available by Autumn 2012

Tool	State	Open software	Back-end 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 color 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	Discontinued 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 and moreover, not many of them were well documented. At the end the decision was made to select *Wired Marker* as a starting point for the 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 back-end database can be adjusted.

Wired-Marker's annotation functionality, though limited, still is in line with our goals: select a text fragment of an arbitrary web-document, mark it with a specific color and add a text comment (an annotation body). Aggregating Wired-*Marker* annotations in bundles is implemented by a collection of pre-defined folders. Each folder contains annotations made by a certain marker (color). For instance, in the red folder all annotations made by the red marker are collected.

It was possible to extend the Wired-Marker code so that the extension could communicate with the server to retrieve an annotation from the database or send a created annotation to the back-end.

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

4.2 Developments after September 2012

PundIt allows annotating images and their fragments. Moreover the tool allows aggregating annotations into notebooks that can be viewed as a generalized version of the "cultured folders" aggregation facility of *Wired Marker*. *PundIt* has a feature, which in some cases may be considered as an inconvenience, and thus it gives more points to *Wired Marker* because *Wired Marker* does not have it. 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" denotes 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.

5. DASISH Web Annotator (DWAN)

5.1 Framework architecture

The DWAN design assumes multiple clients working together with a single back-end consisting of a database and a Representational State Transfer (REST) web service that is 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 are stored in a database that can be shared with other tools in the framework (see Figure 2). At the moment the storage for annotations and related resources is provided by the DASISH partner TLA-MPI[6] that currently runs the back-end.

DWAN is also especially meant to cater for domain specific tools such as within linguistics, that through their use of linguistic data formats can annotate specific linguistic items such as lexical items, annotation tags etc. Tools for other domains can be integrated without problems, the data-model underlying the DWAN framework is discipline agnostic.

[6] The Language Archive, Max Planck Institute for Psycholinguistics, http://tla.mpi.nl/

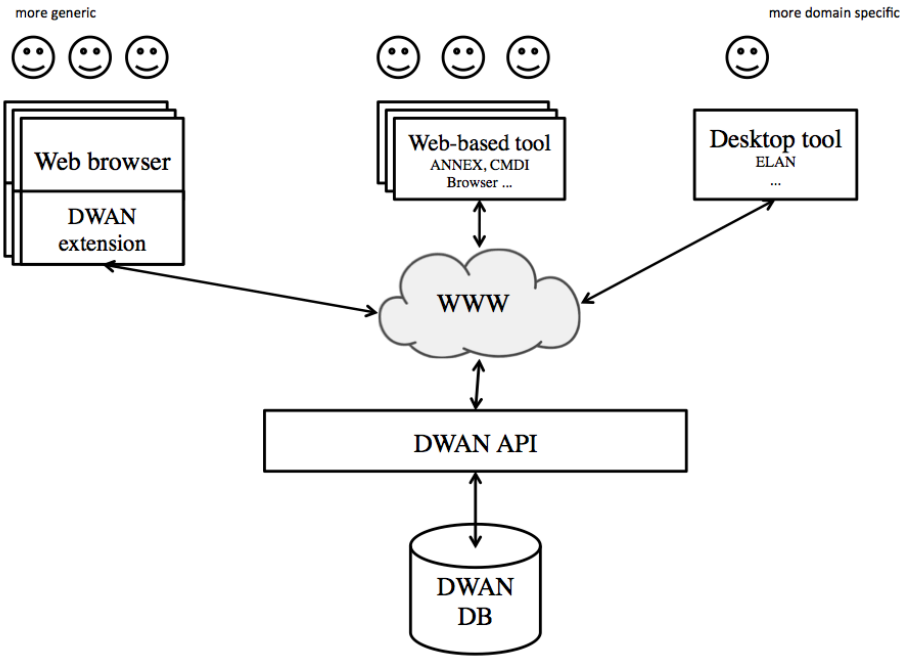

Figure 2 The DWAN Framework in more detail

5.2 DWAN's Data Model and its connection to Open Annotation Model

The *Annotation* class is the core of the model (see Figure 3). The relations *Annotation - Target*, *Target - Source*, *Target - Cached Representation* closely follow the *Open Annotation* (OA) standard. The Open Annotation Core Data Model specifies an interoperable framework for creating associations between related resources, annotations, using a methodology that conforms to the Architecture of the World Wide Web[7] (W3C). In OA an Annotation is considered to be a set of connected resources, typically including a body and target, where the body is somehow about the target. The full model supports additional functionality, enabling semantic annotations, embedding content, selecting segments of resources, choosing the appropriate representation of a resource and providing styling hints for consuming clients.

An annotation in DWAN, i.e. an object of the class *Annotation*, is a structure that contains necessary information about the 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.[8]

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. instantiation 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. A target can also be related to one or more cached representations. A cached representation is a stored record that contains representations of the relevant parts of the annotated document together with the descriptions of their respective annotated fragments.

[7] http://www.openannotation.org/spec/core/

[8] 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.

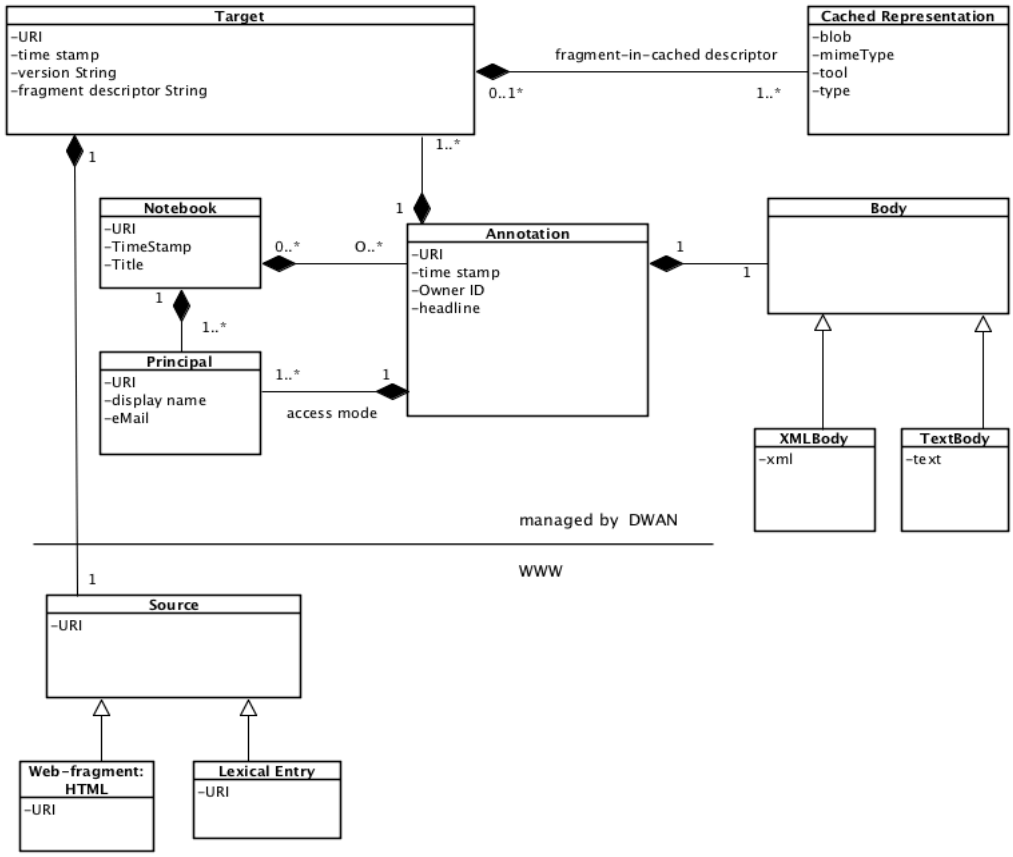

Figure 3 DWAN Data Model

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 client must give a precise MIME-type. For instance, a body can be a plain text, which describes a specific 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.

DWAN model has been designed with *Open Annotation* in mind, and therefore the mapping between DWAN-model components and open-annotation concepts is built in a natural way. The targets of DWAN model correspond to the instances of the open annotation class `oa:SpecificResource`, see Figure 4. 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[9]; 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`.

[9] http://dublincore.org/documents/2010/10/11/dcmi-type-vocabulary

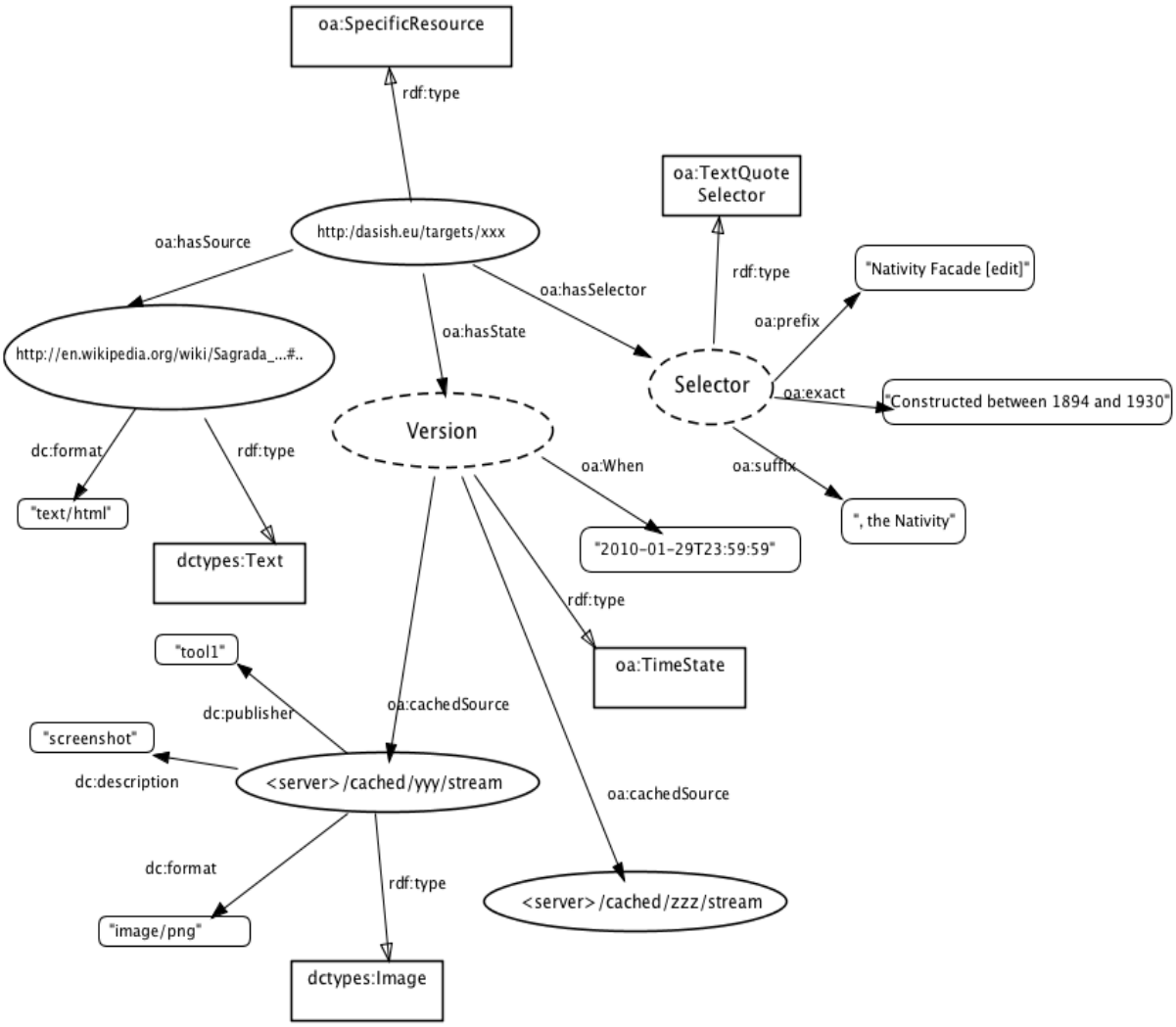

Figure 4 Example of an OA representation. Target

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 that 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 5.

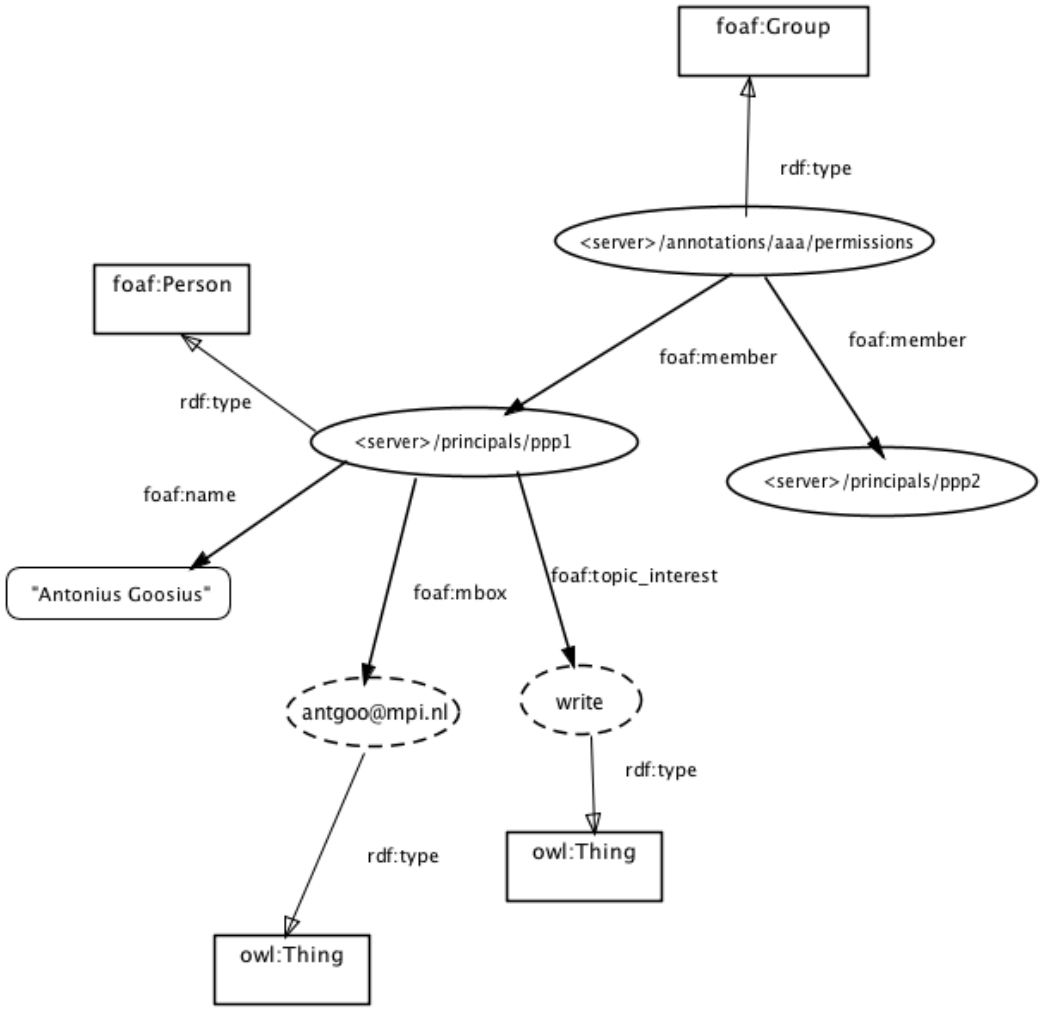

Figure 5 Example of an OA representation. Principal

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.

5.3 DWAN Back-end

Architecture in a nutshell

The core of the back-end is the Postgres[10] database where all annotations and related structures are stored, together with information about principals. The task of the back-end software is to connect a client with the database. The back-end software accepts a request from the client tool, validates it, and

[10] http://www.postgresql.org

translates it into database queries. The back-end software translates the database content and sends it to the client.

The back-end-software is a multi-layered project. Its outermost layer is generated by the *Jersey Framework*[11] which is responsible for connecting database-managing software with the web-server (e.g. *Tomcat*[12]) which hosts the database. The Jersey shell is not written by DWAN developers, but used as a library of program modules.

The remaining layers are designed and implemented by DWAN developers in Java. Next to the Jersey shell there is a package containing REST methods which accept client requests in the form of http-strings, possibly together with XML-bodies for more complex requests. For instance when a client has to send it as an XML file for posting an annotation. This XML file is *deserialised* within the POST-annotation REST method into a Java instance of the class Annotation, using JAXB technology[13]. The other way around, REST methods also translate database responds into interpretable by the clients presentations. For instance, when getting an annotation, the respond from the database, which constructed as a java object, is *serialized* by the GET-annotation REST method into an XML file, which is sent to the client. The client is responsible for converting it into a user-friendly form.

REST methods do not perform calls to the database directly. A REST method uses *Data Access Objects* (DAO's) that take a REST request together with its parameters and translate it into a Postgres database command. For instance, a GET request is typically translated into a SELET command of PostgreSQL. POST and PUT requests are translated into INSERT and UPDATE commands respectively. To be precise, a REST method does not call DAO objects directly but uses an intermediate layer, a *dispatcher* class. This is because a REST request cannot be interpreted as a single PostgreSQL command but is a chain of such commands. For instance, when getting an annotation, first users access rights must be checked via a separate DAO. If the logged-in user has "read" rights for the requested annotation, then GET-annotation request is to be fulfilled. Otherwise, the GET-annotation method returns 403 status: access forbidden. Thus, the intermediate dispatcher object is responsible for turning a REST request into a sound chain of the calls of the necessary DAO objects. The DAO layer is the innermost layer of the back-end software.

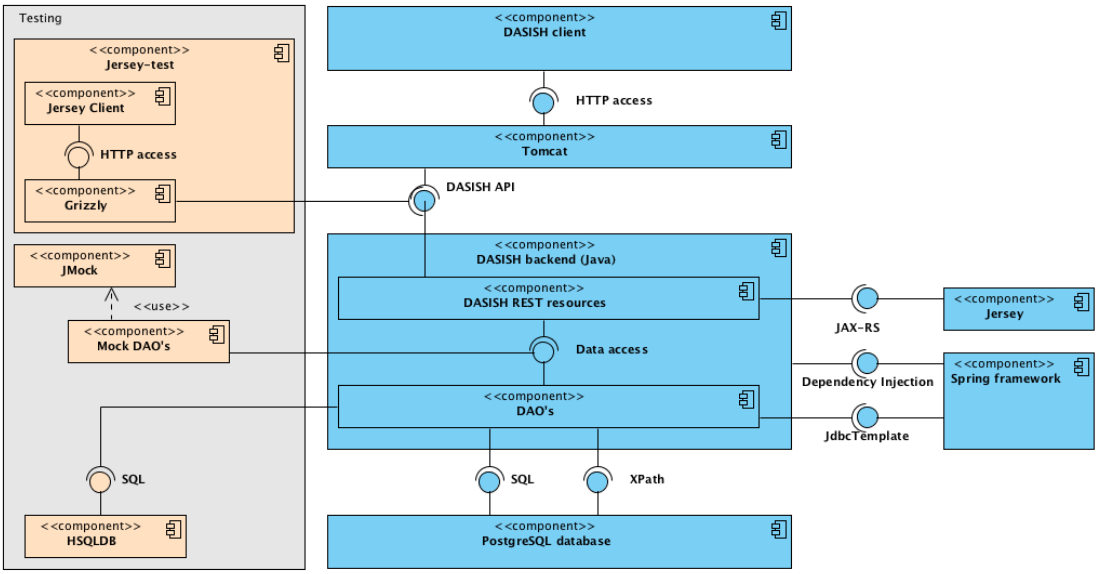

Figure 6 DWAN backend Architecture

[11] https://jersey.java.net

[12] http://tomcat.apache.org

[13] https://jaxb.java.net

Database and Database Access Objects

A Postgres relational database provides storage for all the core information 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 DWAN 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 stores 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 create 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*[14], which utilizes a JDBC[15] 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*, *and 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.

---

[14] http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/dao.html

[15] Java Database Connectivity

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 that the server must recognize 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 authentication procedure e.g. provided by *Shibboleth*, which is a "an open-source project that provides Single Sign-On capabilities and allows sites to make informed authorization decisions for individual access of protected online resources in a privacy-preserving manner" [16]. 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 fulfill 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 in Table 2.

Table 2. 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

[16] http://shibboleth.net/

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

Principal realm

Table 3. API for resource *Principal*

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

Table 4. API for resource *Annotation*. Part A.

Resource	Description	Return (xml) type
GET api/annotations? link=*URI*& text=*text*& access=[[*read, write*]]& 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

In the GET request for the future we may add a namespace parameter, *ns* . It may be used to make queries on XPath for xml annotation bodies. For instance, the following query

api/annotations?ns=rdf:http%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-ns%23&ns=owl:http%3A%2F%2Fwww.w3.org%2F2002%2F07%2Fowl%23&xpath=//owl:sameAs[rdf:resource="example:2"]

is used to find an annotation with the body:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
<owl:sameAs rdf:about="example:1"
rdf:resource="example:2"/>
</rdf:RDF>
```

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.

Table 5. API for resource *Annotation*. Part B.

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

Table 6. API for resource *Target*

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 also 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.

Table 7. API for resource *Cached Representation*

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[17]

api/notebooks

Table 8. API for resource *Notebook*

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	Envelope NotebookResponseBody

[17] The feature is implemented, however testing is not completed and it is not used in the current DWAN front-end.

Resource	Description	Return (xml) type
	response's payload.	
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

5.4 DWAN front-end(s)

Wired-Marker based front-end

The original *Wired-Marker* software 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. The tool's concept and design are credited to BITS Co., Ltd.[18] and Prof. Okubo.

Wired-Marker is licensed under a Creative Commons License. This includes a "No-Derivative works" condition, which means that the modified code cannot be distributed. According to the special agreement between BITS Co., Ltd. and the MPI for Psycholinguistics, this condition has been waived.

Wired-Marker as well as Wired-marker-based DWAN client, is a Firefox extension that can be used with Firefox versions greater than 2.0. The DWAN client can be downloaded as an XPI[19] file from the DASISH GitHub repository at https://github.com/DASISH/dwan-client-wiredmarker/releases. A more detailed description on how to install the extension can be found in the Manual, see the Appendix of the presented deliverable. After completed installation of the add-on, a new menu item called *DASISH Web Annotator* is added to the Firefox menu bar.

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 developed by Mozilla. XUL is implemented as an XML dialect; it allows for graphical user interfaces to be written in a similar manner to web pages. One of the possibilities to develop Firefox add-ons such as Wired-Marker is to use the FoxBeans plug-in for NetBeans 7.0 IDE. The plug-in adds a new project type Mozilla/Firefox add-on that can be used for extension development. Another common option is to work with a development setup that uses an extension proxy file locally. In the case of the Wired-Marker extension code, the jar structured chrome.manifest file also needed to be rewritten and adjusted to the local chrome paths. We recommend a developer to read https://developer.mozilla.org/en-US/Add-ons/Setting_up_extension_development_environment on how to set an extension development environment.

From the user's point of view, the original Wired-Marker extension is a highlighting tool that allows marking fragments of a web document with different colors. The tool (as well as the based on it DWAN client) provides a default finite collection of colors ("markers") with which a user can mark fragments of web-documents. An annotated fragment can be a text fragment or an image inserted in the document. The descriptions of the marked fragments (annotations) are collected in folders

[18] BITS Co., Ltd., http://www.bits.cc

[19] XPI stands for Cross Platform Installer (file extension).

according to their colors. These folders are accessible via standard folder menu of the user interface. If the default collection of markers does not suffice the user can create his own marker by picking up a fresh color from a rich palette provided by the DWAN client on user's demand. For instance, one can create a purple marker (purple is not a default DWAN marker), and annotate with this color fragments about the family of Picasso, from various web pages. Then all these annotations are contained in a purple folder.

An annotated fragment is preserved not only in the local client database that is connected to the extension but also sent by the DWAN client as an XML file to the back-end database where it is stored. The DASISH developers have implemented synchronization of the local and the back-end database. One of the aspects of this synchronization is that one of the provided default markers, the light-yellow one, plays a special role. By now it corresponds to the annotations created by other client instances. The client retrieves these annotations from the server's database and places them in the folder called *incoming*. In a sense, it is inconvenient because the user cannot see the colors of annotations made by the others. It is due to the fact that the original Wired-Marker was not designed as a collaborative annotating tool.

DWAN-developers team is working on possibility to retrieve the original colors of annotations created by other client instances. In fact, a DWAN client does send to the server the color information when an annotation is created, and this information is saved in the database. At the moment the DWAN client cannot interpret the color information when it retrieves the annotation from the database. The server transfers the color information on GET request of the client.

Technically speaking, the annotated 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 an original document. The information about the color is represented as a CSS property as part of the fragment.

Other users can view a particular user's annotation in their DWAN clients simply by reloading the annotated page. As stated above, an annotation made by remote instances of the DWAN client is listed in the directory of *incoming* annotations in the sidebar on the left-hand side of the browser window. The corresponding annotated fragment appears as a light-yellow cultured fragment.

In order to access the database and thus use DWAN and its functionality (e.g. view and post annotations), one needs to log in. DWAN offers two ways of authentication, firstly via *Shibboleth* and secondly via Spring basic authentication. If the user's institution is part of the DWAN connected Identity Federation[20] (s)he can use her/his institution credentials by choosing the institution name from the list of Identity Providers. Otherwise, the user can create an local account for the DWAN back-end by filling in and submitting the registration on the page accessed by the menu "DASISH Web Annotator -> Settings -> Server" and specify the desired server. Choose the default option https://myserver/ds/webannotator/ for Shibboleth authentication and choose https://myserver/ds/webannotator-basic/ for the basic authentication service.
The user needs to set the back-end server URL via the DWAN-client menu, in case the intended server differs from the default one. This can be done in the Settings dialogue window (DASISH Web Annotator>Settings…>Server), where a user-specified back-end address can be inserted[21].

When a user creates an annotation, the client sends it to the server together with a cached representation of the annotated page (in the moment of annotation). The user can request a cached representation later, for instance if the client cannot deliver the annotation because the page has been changed and the fragment cannot be resolved. Please, consult the manual for more details. The cached representation is sent as a serialized DOM for the HTML document. For images only links are sent. The next step in future development 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 back-end for now.

---

[20] Currently DWAN is connected to the CLARIN Service Provider Federation, http://www.clarin.eu/content/service-provider-federation

[21] At the moment the default server is https://lux17.mpi.nl/ds/webannotator. The user may set https://lux17.mpi.nl/ds/webannotator-basic as a user-specific server if he wants to follow basic-authentication procedure.

It is possible to annotate an image, but not an image fragment. The mouse pointer must be placed on the image, and the remaining 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 edit an annotation, select it in the list on the left-hand side of the browser window. Click the right mouse button and select *Properties* in the popped-up menu. Selecting *Properties* triggers a pop-up form for editing the annotation. The title can be edited in the *Brief Overview* tab, and annotation body can be edited in *Annotation* tab.

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 dynamic access rights. When a DWAN client creates an annotation, all registered principals except the creator (*owner*) get *read* access. The owner has *write* access and can change the rights of other users. Additional web pages from the back-end server allows the owner to reassign the rights for a particular user and an annotation, or to change the public access mode for a given annotation. On principal's request, the back-end can present one of two web pages allowing the owner of an annotation to change the annotation's access modes, which can be *read*, *write* or *none*. The first page allows changing the access to a specific annotation for a specific principal. The second page is used to change the public access mode at once, that is, all registered principals get *write* access.

Updating access modes is implemented through web pages issued by the back-end because changing access rights is not implemented in Wired Marker itself, and adding this feature to DWAN Wired-marker-based front-end would be quite time consuming.[22]

While working on the transformation of Wired-Marker into a DWAN client, the DWAN development team has established that four of Wired-Marker's drawbacks cannot be fixed within a reasonable amount of time:

- first, the original Wired-Marker does not provide multiple-target annotating; in other words, a user can put a text note exactly on one fragment of the source page; for instance, it is not possible to annotate two text fragments simultaneously, and to interrelate and link them with the remark that they contradict each other;
- the second drawback has already been mentioned: an annotation read from the database loses its original color while being interpreted by the client[23];
- thirdly, fragments of images cannot be annotated by Wired-Marker, but only the whole image;
- In the fourth place, adding notebooks would demand significant refactoring of the original code; to a certain extent, cultured directories of the local folder can be seen as notebooks.

Summing 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 back-end 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);

[22] The redirection to these pages was under implementation in the front-end by the time the deliverable was written.

[23] The client developers are working on a fix for this problem at the time of writing of this document.

	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); setting updated annotation bodies.

Front-end for ELAN

An ELAN front-end for the DWAN framework is being worked on in the context of the CLARIN NL[24] ColTime project[25] (in progress). ELAN is a multi-media annotation software tool and the goal of the project is to allow ELAN users to exchange messages or create comments consisting from a reference to a particular fragment of a media file and a message text. Such comments or messages are mapped on the DWAN annotation concept. Since ELAN already had the notion of linguistic *annotations*, in this section we will use the word *comment* for DWAN annotations to avoid confusion.

With ELAN, users can make annotations associated with specific time spans of the media file. This is organized in so-called tiers. Within each single tier, the annotations cannot overlap, but between different tiers they can. Users can organize tiers so as to use different tiers for different types of annotations. For instance, one tier could contain annotations pertaining to pitch level, while another contains information about hand gestures. However, there was until now no specific support to comment on the ELAN annotations themselves. For instance, 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 use however has several drawbacks. For example, annotations on a single tier can't overlap each other time-wise and multiple comments referring to the same period become cumbersome.

On the other hand, the DWAN back-end 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 target by means of a fragment identifier.

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

On the other hand, once an ELAN file is imported into an archive, it will be assigned a stable URL and can then be viewed using the ANNEX web tool.

Front-end for ANNEX

ANNEX is an open source online visualizer for time-aligned annotation files, primarily targeted at the EAF (ELAN Annotation Format) format just as ELAN. It provides 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. ANNEX will work in a standard (Flash enabled) web-browser.

As is the case with the ELAN front-end, ANNEX interaction with the DWAN is being developed in the context of the COLTIME project.

Given that ANNEX handles the same type resources as ELAN and also uses the same linguistic annotation model, in the ANNEX's context, DWAN annotations are also referred to as *comments*.

[24] http://www.clarin.nl

[25] http://www.ru.nl/sign-lang/projects/coltime/

However, ANNEX is only a visualization tool for archived materials and currently does not offer any functionality to create annotations. Still users would like to make and exchange comments with respect to archived media and annotation files independent of the possibility to actually add linguistic annotations.

Using the DWAN back-end to store, search and retrieve such comments is easier than in the ELAN case, ANNEX already relies on URLs and part identifiers to fetch its data, and specifically ANNEX's URLs accept time period ('time=' and 'duration=') and tier specification ('tier=') parameters are already available. This also eliminates the need for the EAF URN described in the ELAN section of this document.

5.5 Testing Procedure

The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The detailed testing plan, which can be found at https://github.com/DASISH/dwan-testing, identifies the following:

- 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 was performed at several points in the life cycle, as the product is developed. 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, and 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/dwanback-end .

For back-end the testers have implemented a python script what tries to perform several API operations (https://github.com/DASISH/dwan-testing/tree/master/scripts).

6. Social Sciences and Humanities: Results and Outlook

Annotation is an activity which runs throughout all scholarly work in all disciplines. The purpose of this section is to give context to the DASISH Web ANnotation framework (DWAN) annotation tool, to explore how annotation works in the broader context of scholarly communication in the humanities and Social Sciences (HSS), and to set out a series of scenarios which users in these domains are likely to encounter when faced with tasks requiring annotation and related activities.

This review will comprise of three main elements:

- a list of software annotation tools drawn from the Tools e-Registry for E-Social science, Arts and Humanities (TERESAH) registry, which is the primary output of DASISH Work Package 2;
- a mapping of these tools' functionality and, where it can be determined, their usages to the typology proposed by Dunn and Hedges (2012)[26] in their report on crowd-sourcing in cultural heritage and the humanities;
- and a set of user scenarios based on this analysis.

6.1 List of annotation tools used by the HSS community

The list of more than 50 tools has been generated from a simple search using the keyword annot* in the Tools e-Registry for E-Social science, Arts and Humanities (TERESAH) registry, currently under development for WP2. Here we briefly describe ten of them which form our point of view look the most promising as potential DWAN front-ends:[27]

- *ANNIS* is an open source, versatile web browser-based search and visualization architecture for complex multilevel linguistic corpora with diverse types of annotation. ANNIS, which stands for *ANNotation of Information Structure*, has been designed to provide access to the data of the SFB 632 ("Information Structure: The Linguistic Means for Structuring Utterances, Sentences and Texts"). Since information structure interacts with linguistic phenomena on many levels, ANNIS2 addresses the SFB's need to concurrently annotate, query and visualize data from such varied areas as syntax, semantics, morphology, prosody, referentiality, lexis and more. For projects working with spoken language, support for audio / video annotations is also required. In the SFB, a number of different projects collect and annotate data according to the common SFB Annotation Standard. This data, which is annotated using both automatic taggers/parsers and a small set of manual annotation tools (EXMARaLDA, ELAN, annotate/Synpathy, MMAX, RSTTool), is mapped onto the encoding standard of the SFB, PAULA (Potsdamer Austauschformat für Linguistische Annotation / Potsdam Interchange Format for Linguistic Annotation), a stand-off multilevel XML format, which serves as the basis for further processing. ANNIS2 provides the means for visualizing and retrieving this data.
- *Bibliopedia* is an open source, semantic wiki research platform designed to crawl scholarly resources including JSTOR, the Library of Congress, the Arts and Humanities Citation Index, and similar data sources, extract metadata about works cited, convert that data into a semantic web format, aggregate the different repositories, then display the results on a wiki-style website for the scholarly community to verify, add to, annotate, elaborate, and discuss.
    - We envisage Bibliopedia as an open, research-enabling platform designed to unify the many disparate, closed silos of scholarly information available today that remain

[26] http://www.ahrc.ac.uk/Funding-Opportunities/Research-funding/Connected-Communities/Scoping-studies-and-reviews/Documents/Crowd%20Sourcing%20in%20the%20Humanities.pdf

[27] The information has been taken from http://dirt.projectbamboo.org/resources

difficult and time-consuming to use. Our first goal was to extract and transform bibliographic data into a linked data format consistent with semantic web requirements, and to create large volumes of cross-references among texts, making digitized scholarly texts exponentially more useful to researchers and to machine analysis. The primary innovations Bibliopedia achieves are: 1) the aggregation and cross-referencing of separate silos of scholarly data; 2) the transformation of that information into a format consistent with the semantic web; and 3) crowd-sourcing the verification and elaboration of that data. Mapping and cross-referencing large-scale, high-volume scholarship also means that unexpected connections can be found and brought to light, along with less-known original works that might otherwise remain unread. Moreover, formatting scholarly references for the semantic web will make this data available to a far broader community and enable unexpected innovations. Bibliopedia will generate custom bibliographies and visualizations based on search results, facilitating a wide variety of scholarly inquiry and discovery. Most importantly, Bibliopedia is designed for ease of use, so as to substantially broaden participation to attract the largest possible range of humanities scholars as its user base, in particular scholars who do not normally use digital tools. Bibliopedia provides a RESTful API, SPARQL queries, linked data, Zotero-compatibility, and many other features. Built with Drupal 7, available on github, and served from the cloud for scalability, portability, and reliability, Bibliopedia is open to interested academics and libraries who would like to see what their metadata looks like on the semantic web.

- *LitBlitz Literature Notes Manager* is free web-based beta software that aims to improve how students and researchers manage their notes for literature reviews, assignment research and more. With LitBlitz, you can: avoid hours of printing, highlighting, organizing and typing; save money involved in printing 100s to 1000s of pages, highlight and write notes without shuffling a stack of papers, organize your notes into digital notebooks in real-time, easily transfer notes to your draft review/assignment. LitBlitz was designed from the ground-up to solve problems other annotation and note taking services haven't looked at or have solved poorly. It's is different from popular note taking and archiving software like Evernote in that it allows users to: take text and image snippets from their document/webpage sources rather than forcing them to archive entire documents, write "Own Notes" (personal insights) related to snippets to enable rapid draft writing and context building, manage these snippets in themed digital notebooks for fast, easy reference. The founder is open to improving through suggestions from librarians, academics and Ed Tech.
- *MapHub* is an online application for exploring and annotating digitized, high-resolution historic maps. All user-contributed annotations are shared via the Maphub Open Annotation API.
- *Pliny* is a note-taking and annotation tool. It may be used with both digital (web pages, images, PDF files) and non-digital (books, printed articles) materials. Pliny is a desktop application that runs on your computer, and manages annotations and notes that you gather as you are reading.
- *PundIt* is a semantic annotation and augmentation tool. It enables users to create structured data while annotating web pages. Annotations span from simple comments to semantic links to web of data entities (as Freebase.com and Dbpedia.org), to fine granular cross-references and citations. Pundit can be configured to include custom-controlled vocabularies. In other words, annotations can refer to precise entities and concepts as well as express precise relations among entities and contents. Read more on semantically structured annotations. Pundit is designed to enable groups of users to share their annotations and collaboratively create structured knowledge.
- *UVic Image Markup Tool* allows to describe and annotate images, and store the resulting data in TEI XML files, all within a simple enough interface that can be used by people with little or no experience in editing XML code. Designed to be Windows-only, but can be successfully run on Linux using Wine. It supports a wide variety of image formats and saves markup

information in conformant TEI P5 XML files. It has a simple, graphical interface that lets you see the image and the fields for entering your markup notes and annotations that are visually represented on the image. The tool allows knowledgeable TEI users to add additional TEI markup tags to their annotations. The tool can handle multiple images in one file. Amongst the tool's disadvantages is that editing done to Image Markup's XML files in an external editor may not be preserved.

- *Virtual Lightbox for Museums and Archives* is an educational tool for collecting and reusing in a structured fashion the online contents of museums and archives with visual components. With VLMA, you can browse and search collections, construct personal collections, export these collections to xml or Impress presentation format, annotate them, and share your collections with other VLMA users.
- *WebLicht* is a service-oriented architecture (SOA) for creating annotated text corpora. Development started in October 2008 as part of CLARIN-D's predecessor project D-SPIN, and further development and enhancement of WebLicht is an important goal of CLARIN-D, aiming to make WebLicht a fully functional virtual research environment. WebLicht employs chains of RESTful web services. Each web service encapsulates a certain linguistic tool. For example, users can access, as a web service, the query component of a corpus, a format converter, a tokenizer, a tagger, or a parser. Translation between the input format specific to some tool and the WebLicht information interchange format TCF (see below) is performed by a web service wrapper. Each web service adds at least one layer of annotation encompassing the work of the tool encapsulated by that service. The output of a chain of WebLicht services is an automatically analyzed corpus in the form of an XML document. Each WebLicht service must be able to use a common interchange format that all the other services can also process. CLARIN-D's Text Corpus Format (TCF), serves this purpose. It is broadly compatible with existing related interchange formats like Negra, Paula, or TüBa-D/Z. Moreover, format-specific converters allow interchange between them. WebLicht can be accessed only with a valid DFN-AAI/Shibboleth-based account or a local Tübingen account.
- *Zotero* allows users to bookmark and save content (PDFs, images, audio and video files, snapshots of web pages, etc.) by automatically pulling in metadata stored on websites. Users can then search, tag and annotate any entry in their library. Zotero is primarily available as a Firefox plug-in, but is now also available as a stand-alone version with connectors to other browsers. Zotero also allows students to automatically create Works Cited pages by drawing on the sources used in a document.

6.2 Functionality mapping

The DWAN framework is designed for use with different client tools that can share annotations. These tools are listed Table 9 and their usage is explained in more detail in potential use cases described below in section 6.3. The functionality of each tool has been mapped to some categorization proposed in the AHRC report on crowd-sourcing in cultural heritage and the humanities written by Dunn and Hedges (2012)28. In this report task types were identified as the following: mechanical, configurational, editorial, synthetic, investigative, and creative. Most annotation tools fall into the categories of the Configurational or Editorial task types; a task is an activity that a user undertakes in order to create, process or modify a digital asset (i.e. geospatial, text, numerical or statistical information, sound, image, video, ephemera and intangible cultural heritage). The Configurational type covers tasks that involve identifying structural patterns or 'configurations' in information, rather

[28] http://www.ahrc.ac.uk/Funding-Opportunities/Research-funding/Connected-Communities/Scoping-studies-and-reviews/Documents/Crowd%20Sourcing%20in%20the%20Humanities.pdf

than processing individual pieces of information. Some such tasks will require a predisposition for working with quantitative data. The *Editorial* type involves modifying or improving an existing asset.

A process is a sequence of tasks through which an output is produced by operating on an asset. Moreover, a tool is considered informal if it has pre-defined entities which can be added as annotations and formal if it does not.

Table 9. Tools that can be used as DWAN front-ends

Name	**task type**	**task sub-type**	**process type**	**asset type**	**formal/ informal**	**collaborative platform**
Bookends	Configurational; editorial	bibliographic annotation	Contextualization	text	informal	N
LitBlitz Literature Notes Manager	Editorial	bibliographic annotation	Commenting, critical responses and stating preferences	text	informal	N
NoodleTools	Configurational	bibliographic annotation	Commenting, critical responses and stating preferences	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 preferences	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 preferences	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; configurational	map annotation	Contextualization	geospatial	informal	Y
NB	Editorial	PDF annotation	Commenting, critical responses and stating preferences	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 preferences	video	informal	N
Annotator\'s Workbench	Editorial	segmenting video	Commenting, critical responses and stating preferences	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 preferences	text	informal	N
Melita	Editorial; configurational	syntax/semantic annotation	Contextualization	text	formal	N
Pundit	Configurational	syntax/semantic annotation	Linking	text; image	formal	Y
Thinkport Annotator	Editorial	syntax/semantic annotation	Commenting, 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; configurational	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
Annotation Graph Toolkit (AGTK)	Configurational	time-series annotation	Cataloguing	text	formal	N
VideoANT	Configurational	time-series annotation	Linking	video	informal	N

Mediathread	Editorial; configurational	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	Editorial	web page annotation	Commenting, critical responses and stating preferences	text; image	informal	N
Annozilla (Annotea on Mozilla)	Editorial	web page annotation	Commenting, critical responses and stating preferences	text; image	informal	Y
Fleck	Editorial	web page annotation	Commenting, critical responses and stating preferences	text; image	informal	N
NoteBook	Editorial	web page annotation	Commenting, critical responses and stating preferences	text; image	informal	N
Project Pad	Editorial; configurational	web page annotation	Commenting, critical responses and stating preferences	text; image; video; sound	informal	N
SharedCopy	Editorial	web page annotation	Commenting, critical responses and stating preferences	text; image	informal	N
Springpad	Configurational	web page annotation	Commenting, critical responses and stating preferences; 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 preferences	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 preferences	text; image	informal	Y (can be stored in Annotea)
Annotorious	Editorial		Commenting, critical responses and stating preferences	video	informal	Y (via OKF)
Atlas.ti	Synthetic		Contextualization	text; image	informal	N

6.3 Potential front-ends for DWAN in Social Sciences and Humanities.

Following the previous categorization in section above (i.e. task type, process type, asset type) nineteen specific cases of uses HSS researchers make of annotation, not necessarily covering just current DWAN functionality, can be identified and grouped under six topics: bibliography, image, web page, syntax/semantics, wiki, video. To describe sets of functionalities demanded to satisfy expectations for an annotating tool for each of these topics, we first introduce the following notations, where UC abbreviates "Use Case":

UC 1: Highlight text
UC 2: Add comments in the form of scribbled notes (text to text)
UC 3: Add comments in the form of scribbled notes (text to image)
UC 4: Modify text: Add information to text (within the text)
UC 5: Modify text: delete information (within the text)
UC 6: Tag an image with keywords
UC 7: Save own annotations
UC 8: Share own annotations via email, Twitter, and Facebook
UC 9: Share selected parts of the original resource via email, Twitter, and Facebook
UC 10: Collaborative annotations (different users)
UC 11: Track versions of annotations

UC 12: Textual interpretation: translation
UC 13: Enhance text with links
UC 14: Enhance text with images
UC 15: Enhance image with text
UC 16: Enhance text with video
UC 17: Enhance text with audio
UC 18: Insert definitions
UC 19: Insert references

Such Use Cases can then be grouped under six headings: bibliography, image, web page, syntax/semantics, wiki, and video. One Use Case can follow under several headings.

- **Bibliography :** UC 1, UC 2, UC9, UC11, UC13.
- **Image:** UC9, UC 11, UC 3, UC 6, UC 7, UC 8, UC14, UC 15, UC 18, UC 19.
- **Web page:** UC 1, UC 2, UC 9, UC 11, UC 3, UC 6, UC 4, UC 7, UC 8, UC 12, UC 13, UC18, UC19, UC16.
- **Syntax/semantic:** UC 1, UC 2, UC 9, UC 11, UC 4, UC 5, UC 7, UC 8, UC 12, UC 13, UC 18, UC 19, UC 16, UC 17.
- **Wiki:** UC 1, UC 2, UC 9, UC 11, UC 4, UC 5, UC 7, UC 8, UC 10, UC 12, UC 13, UC 18, UC 19, UC 16, UC 17.
- **Video:** UC 9, UC 11, UC 3, UC 6, UC 7, UC 8, UC 19.

The mapping from a topic to its list of features can be illustrated by six corresponding user scenario's.

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 researchers have 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.

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 a 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.

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 modeling 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.

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.

5) *Wiki based annotation*. Review of tools available: Bibliopedia. 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:

- The user 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.

6) *Video annotation*. Review of tools available: Advene, Annotator\'s Workbench, Annotorius, Anvil, Atlas.ti, HyperImage, Mediathread, Project Pad, Rehersal Assistant, VideoANT.

Video annotation is probably not the most common form of annotation currently carried out by humanities scholars, however, the literature review shows that several tools that support such activity are in fact used within the HSS communities. This reflects the need to both organize video collections with annotations, and to link comments/notes with them.

Scenario: User has downloaded a few videos from www.youtube.com and made a collection themed around the current use of digital tools among Social Sciences and Humanities scholars. He opens such collection to other users, or collaborators. Both the original user and their collaborator can annotate the videos and share such annotations in a research environment. The collection could include keynote speeches, university lectures , conference and seminar papers as well as software tutorials. The user then wants to share selected parts of the original resource via social media, add personal comments and then share such comments via social media as well.

This scenario is applicable to scholars and universities but also, potentially to software engineers and programmers. Formal/informal: both formal and informal annotations can be relevant here.

Assets: the assets are primarily videos but may involve text and images as well.

Necessary functionalities:

- 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.
- The user may wish to add comments in the form of Add comments
- The user will want to save their own comments in a collaborative environment
- Collaborators will have the right to view the user's annotations as well as to add their own
- In a shared collaborative environment, it will be necessary to Track versions of annotations, including responsibility for different versions.
- The user will need to be able to embed links to relevant texts, bibliography, video and image media.

Appendix A: XML 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*.

```
<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 ################# !-->
    <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="cachedRepresentationInfo" type="dasish:CachedRepresentationInfo"/>
    <xs:element name="cachedRepresentationList" 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>
```

Appendix B: DWAN Wired-Marker manual.

DISCLAIMER: As DWAN tools are under continuous development at the time of writing this report; the manual may be not completely up-to-date. The final manual with a clear description of the framework together with instructions on how to use it will be published at the end of the project in the DASISH DWAN GitHub location[29].

DWAN Wired-Marker client as a part of DWAN framework

The DWAN client is a Firefox extension that enables a user to create free-text annotations on fragments of webpage content. Moreover, the user can share his annotation with other users by granting reader or writer permissions. The DWAN client has been developed on the basis of the existing *Wired Marker* web-annotation (Firefox Plugin) software, by adjusting it for collaborative annotating needs. The DWAN Wired-Marker version is implemented by adding program modules allowing sending and receiving requests to the common server database where the annotations of all users are stored.

The database and the server software that implements access to the database, constitutes the back-end with which the DWAN Wired-Marker client communicates. The DWAN Wired-Marker instances, and also other DWAN compatible clients, have access to the database via a uniform service interface available over HTTP. In order to communicate with the back-end, 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.

The DWAN back-end and the DWAN compatible clients constitute the DWAN framework that is a solution for collaborative annotation. An important feature of the DWAN framework is that created content and sources can be stored in a shared database. In its turn, the DWAN Wired-Marker client allows a user to send and retrieve cached copies of the annotated resources.

Individuals as well as groups of researchers from different institutions, countries or backgrounds can all benefit from using DWAN framework. Research Institutes or groups of researchers can develop their own clients for their particular use and purposes and as such they will have access to the shared DWAN Database.

Download and installation

The DWAN client can be installed or downloaded from the github repository, https://github.com/DASISH/dwan-client-wiredmarker/releases. One can install it by navigating to this web page using the Firefox web browser, clicking on green button "dasishwebannotator.xpi" and following the simple standard instructions issued by the browser, like "allow" to install software from the site.

A second option is to start up a Firefox, drag and drop the xpi-file onto the Firefox window and yet another option is to: 1) download the xpi file in some directory of the user's computer; 2) run the Firefox add-on manager; 3) follow "Install add-on from File" procedure by clicking the corresponding menu (see Figure 7).

[29] https://github.com/DASISH/dwan-documentation

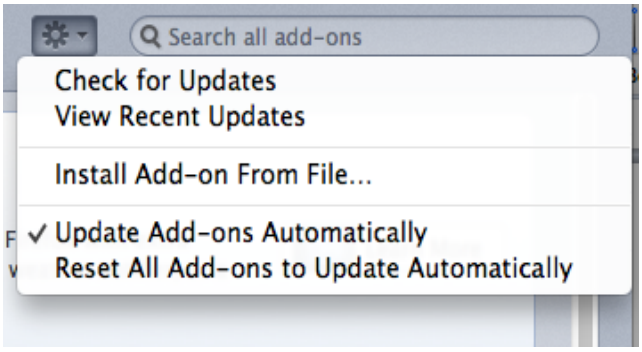

Figure 7. Firefox menu to start installation of the DWAN client from file

After installation is completed, "DWAN/Dasish Web Annotator" is added to the Firefox menu and once activated, the DWAN menu will appear on the left sidebar.

Account management and logging-in

In order to use DWAN one needs to login into the back-end. DWAN offers two ways of authentication: (A) using a federated login e.g. *Shibboleth* and (B) with a local DWAN account that you can create yourself by filling in a form on the DWAN server-generated web page where you provide your login, e-mail address and password. Below both authentication procedures are described in more detail.

(A) If your institution is part of the DWAN supported trust federation[30] and listed within the Discovery Service list of home organizations (see Figure 8) you login with your institution credentials. Choose from the list of home organizations, select, and log in.

(B) If your institution is not listed on the home organization list, you can create a user account following the following steps:

a. Go to *https://myserver/ds/webannotator-basic*. [31]
b. click on *Register as a non-Shibboleth user*
c. fill in the user registration form and submit it
d. go to DASISH Web Annotator > Settings > Server > write this link: *https://myserver/ds/webannotator-basic* in the *User Specified* box and close. See Figure 9 for an example

[30] Currently the DWAN back-end is connected to the CLARIN trust federation and allows access to all home organizations using CLARIN services as are also all eduGAIN connected home organizations.

[31] For example, the current DWAN annotation service is located at https://lux17.mpi.nl/ds/webannotator-basic/

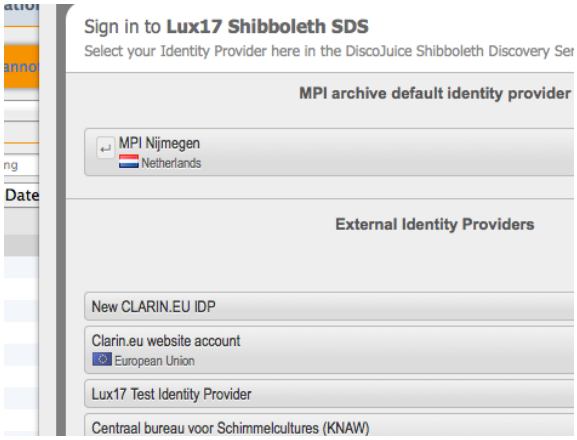

Figure 8 Authentication using Federated Identity

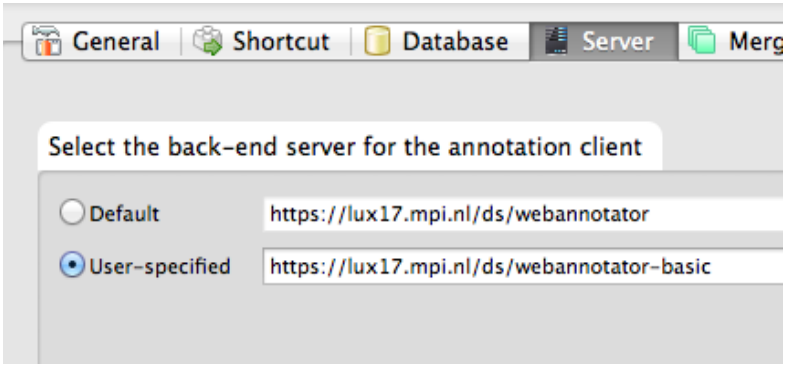

Figure 9 Configuring the Service Location

Viewing annotations

Annotations created on other client instances or by other users are all listed in the **Incoming** folder, in the left side box. The DASISH website is the default webpage. Navigate to the page you are interested in. Click the "reload" icon in the browser bar. If the page has been already annotated, a full list of annotations will appear on the bottom-left side of the browser window. The list can be ordered by annotations' 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 se from the full list. It will appear on the webpage marked by light yellow color, see Figure 10.

To view own annotations, navigate to the **Marker** folder and click on the color used to make the annotation you are interested in. You will see the list of all annotations marked by this color. Select the one you need.

If an annotation does not appear after clicking on it and also after refreshing the page, it means that the DWAN client cannot resolve the annotated fragment. The most probable reason for this is that the webpage has been changed since it was annotated.

However, in DWAN a user can see the annotations even if the webpage has changed. This is done by requesting cached representations of the corresponding annotated pages. 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. You will be able to get the cached representation of the page, which almost always looks like the original page. You will find the annotation you are interested in.

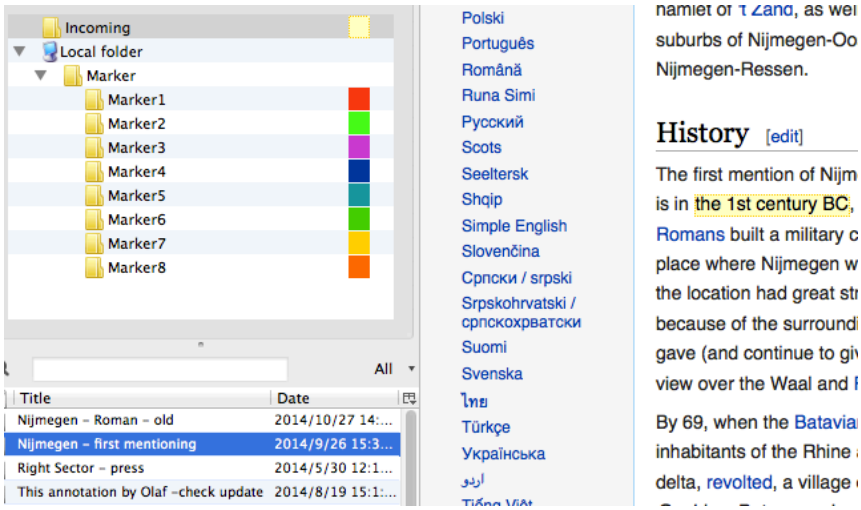

Figure 10. Viewing annotations of other users

Annotating documents and editing annotations

To annotate a web-document, navigate to the corresponding webpage and select a text fragment with the mouse. After right click select "Marker"-folder in the menu. Next, select the color you would like to use to mark the text fragment, see Figure 11. Selecting a markerFigure 11. Following this, a pop-up text-box with two fields appears. One can assign a distinctive title to the annotation in the *Title* field and write a clear short description in the *Annotation* field. To save the work, click "ok". This then (finally) creates the annotation. It is shown on the web page now.

To update the annotation, pick it up in the list, right click it and select "Properties" in the menu. The form for editing will appear, and by selecting tabs "Brief Overview" or "Annotation" one can edit the title and the text body. See Figure 12. Note, that only the creator of annotation or a user with "write" access can update the annotation.[32]

Figure 11. Selecting a marker

[32] Adding possibilities to change access rights is currently work in progress

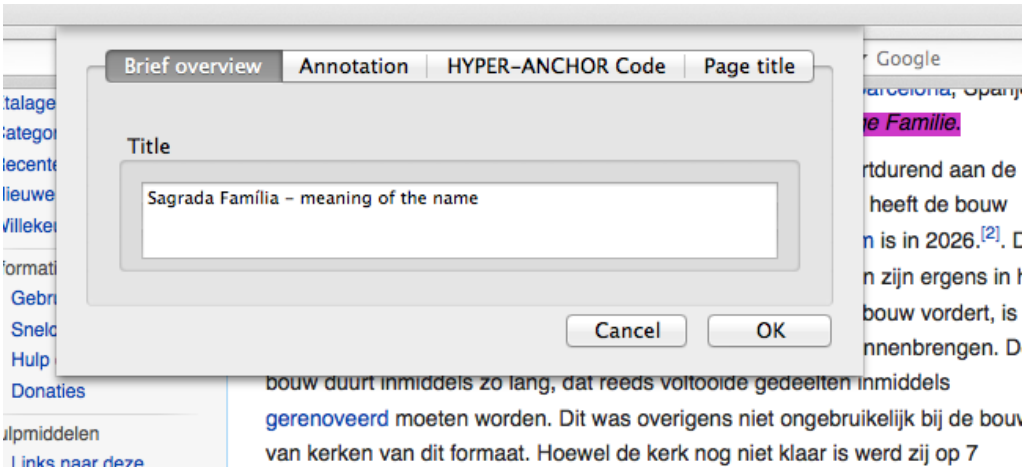

Figure 12. Editing annotation

When the user creates an annotation, all registered users except the creator ("owner") get *read* access. The owner has *write* access, and users with *write* access can edit the annotation. Only the owner of an annotation can change the rights of other users and delete the annotation.

To change the access rights of an annotation right click it and select "Permissions". Fill in the pop-up form, see Figure 13

Public access defines minimal access rights for each logged-in user. For instance, if it is set to *read* then each logged-in user is able to read the annotation. Rights for a particular user are defined as maximum of public access and his individually set rights. For instance, the user with the e-mail xxx.yyy@mpi.nl on the Figure 13 has *write* access. To delete an annotation, look for it in the list, right click it and select 'Delete'.

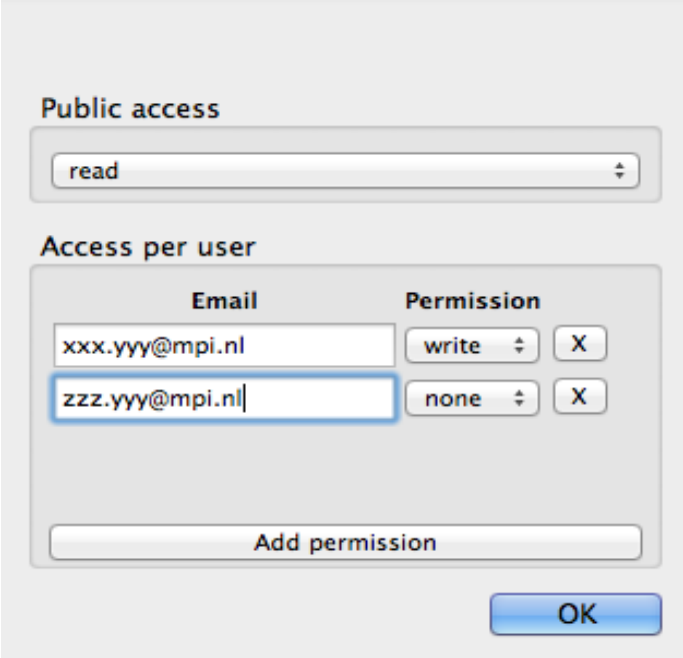

Figure 13. Changing access rights for a selected annotation

Troubleshooting

Advanced users and developers can examine the relationship between the Back-end and the Front-end directly by installing Firebug or Tamper Data, which are two other Firefox add-ons. This can be useful in situations where DWAN does not seem to work properly.

Because of the updates of the DWAN client, Firefox and operating systems, sometimes it is necessary to reinstall the client after a new release. Normally, it is necessary first to de-install the current version of the DWAN client following standard Firefox procedure of the add-on manager. Follow Tools > Add-ons in the browser menu to start the add-on manager. Within the add-on manager choose to de-install a selected extension, e.g. the DWAN client. Now, the second step: the new version of the DWAN client can be installed as it is described in the beginning of this manual.

However, sometimes the newly installed version would not work. In this case one should inform the administrator and the DWAN developers. Still to be able to work, create a new Firefox profile. Within this new profile you will download and start the new version of the DWAN client as usual. How to make a new profile and start it, is explained in detail at https://support.mozilla.org/en-US/kb/profile-manager-create-and-remove-firefox-profiles.

Alternatively, on MAC OS one can create a profile via Terminal window 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 the command
/Applications/Firefox.app/Contents/MacOS/firefox -profile ~/Library/Application_Support/Firefox/Profiles/nameofprofile -no-remote.

To create and use a new Firefox profile in Windows you can use the Firefox Profile Manager that allows you to create a new profile while retaining your original one. If Firefox is open, close it completely by choosing “File -> Exit”. Go to the Windows Start Menu and select “Run”. Enter firefox.exe –P and click OK. Click the “Create Profile” button on the “Firefox – Choose User Profile” window that comes up. Click “Next >” in the “Create Profile Wizard” window that comes up. Type in a new name in the “Enter new profile name” box and click “Finish”. Clear the “Don’t ask at startup” box so that it is unchecked and click the “Start Firefox” box. Firefox will then start with a new profile.