

# An Open Source Annotation Service for the Atlas of Living Australia

Ron Chernich  
University of Queensland  
[chernich@itee.uq.edu.au](mailto:chernich@itee.uq.edu.au)

Stephen Crawley  
University of Queensland  
[scrawley@itee.uq.edu.au](mailto:scrawley@itee.uq.edu.au)

Donald Hobern  
CSIRO  
[Donald.hobern@csiro.au](mailto:Donald.hobern@csiro.au)

Jane Hunter  
University of Queensland  
[jane@itee.uq.edu.au](mailto:jane@itee.uq.edu.au)

## Abstract

In the last quarter of 2008, the eResearch Group [<sup>1</sup>] at the University of Queensland began development on an **Annotation Service** for the Atlas of Living Australia (ALA). Twelve months on, the project has reached a state of stable production readiness, having met all of the initial objectives of the Functional Requirements for use in the ALA Portal. The development team has built two main open source components. The first, code-named *Danno*, is a Java based annotation server that implements the W3C *Annotea* public Draft Protocols [<sup>2</sup>], and the *Open Archives Initiative Protocols for Metadata Harvesting* (OAI-PMH) [<sup>3</sup>]. Users may interact with the server using the browser-independent client-side user interface components developed in parallel with *Danno* which we have named *Dannotate*, or through existing, browser-specific *Annotea* clients such as *Annozilla* [<sup>4</sup>]. Together, *Danno* and *Dannotate* form a comprehensive HTTP based annotation service for Internet or intranet content. The granularity of annotations and associated reply chains may be a full HTML page, selected areas of text on a page, and/or selected regions of images on the pages, or images served stand-alone. In this paper we describe the major technical design challenges and decisions made, highlighting the implications of some of these choices as they apply to users and administrators of the ALA, or any other systems that elect to utilize the *Danno* and/or *Dannotate* components to provide annotation services.

## Server Architecture

The *Danno* server implements two well known protocols. The first is *Annotea*, a protocol for creating and publishing sharable annotations of web documents, first issued by the W3C as a Public Draft for discussion in 2001, and revised in 2002 [<sup>5</sup>]. The second is the popular Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) which provides a convenient way for the annotations and replies to be selectively harvested by date range, or set identifier. The *Annotea* protocol was chosen because:

- 1.1 It is RDF based and extends easily without server side modifications
- 1.2 It maps well to modern REST-ful Web Service Architecture [<sup>6</sup>]
- 1.3 Other clients based on *Annotea* exist and are being actively maintained
- 1.4 Each *Annotea* identifier resolve to a URL
- 1.5 *Annotea* servers can be federated
- 1.6 RDF records enable lossless harvesting through OAI-PMH

On the down-side:

- 1.7 *Annotea* is still a W3C Public Draft for discussion, not a Standard, and is not being actively progressed
- 1.8 The specification is “by example” with no concise definition of the minimal requirements
- 1.9 The associated query capabilities are minimal
- 1.10 It has no clear underlying schema

Considered on balance, it appeared that adoption of the *Annotea* Protocol provided sufficient advantages to offset the inconveniences. Work with the actual server that has been developed over the past twelve months has supported this decision.

The requirement to make annotations available through OAI-PMH was addressed using the open source OAI CAT [7] implementation. Past experience with this component had been positive, proving reliable and adaptable. The integration to *Danno* provides:

- 2.1 Harvesting in Unqualified Dublin Core format, or as the actual internal RDF format
- 2.2 Full support of resumption tokens for harvesting identifiers, records, or sets
- 2.3 Flexible set definition
- 2.4 Provision for browsing via a HTML forms based interface with RDF records displayed using an appropriate XSLT style sheet to assist human readability.

XXXX

### Client Architecture

The project sponsors expressed a strong desire that the client interface be a thoroughly thin client, with no use of “plug-in” code, nor any other user setup or installation required. The reasons for this are sensible:

- 3.1 Many corporations and IT departments rigidly restrict the Standard Operating Environment (SOE) and prohibit the use of browser plug-in code as a potential security threat
- 3.2 The normal variation in computer awareness in users means that when faced with an installation process, fear, uncertainty, doubt and sloth will result in some potential users being lost
- 3.3 Plug-in modules are browser specific and even browser version specific. This compounds the problem of maintaining the source code with the added complication of finding developers with sufficient skills to cover all the desired platforms and associated languages and integration techniques.

These are good and valid points. The alternative is to build the complete user interface using a common browser scripting language. The only choice is ECMA Script, also known as JavaScript [8]. This presents some disadvantages:

- 3.4 There is some variation between different browser vendors in their implementation of JavaScript and the Document Object Model (DOM) API exposed through it. Microsoft’s Internet Explorer (IE) is noteworthy in this regard.
- 3.5 Pure client-side JavaScript is carefully restricted in what it can do in order to prevent its use for malicious actions. In some cases, the model for communication imposed by an *Annotea* server will be indistinguishable from a security violation
- 3.6 Creating annotations on selected regions of text and images was anticipated as being somewhere between difficult and impractical with the available technology.

The SOE profile for a typical ALA user proved difficult to forecast. From statistics obtained from a sample of server logs, it appears that IE accounts for upwards of 60% of the user-agent traffic. After loading a user request, all browsers parse the HTML page into a Document Object Model (DOM) which can be programmatically examined and modified by JavaScript code. The DOM built by IE has several underlying differences to that constructed by other browsers, all of which follow the W3C standard [9]. This further complicates the task of providing cross-browser compatibility and interoperability.

Prototyping confirmed that the expected technical challenges existed. However with the exception of IE, solutions were readily identified, including the creation fine grained annotations identified as 3.6 above. This was achieved through creating an XPath-XPointer “context” for the regions involved that were portable, giving the same selection result in different web browsers. As creating and resolving XPath-XPointer is not part of the core ECMA script API, a module to perform this was adapted from the open source code used by *Annozilla*. This code was designed as a trusted *Firefox* plug-in, so it was extensively rewritten to execute in a cross-browser, un-trusted environment. The code relies on a number of W3C DOM API calls which are not provided by IE. Introducing equivalent functionality to

IE in order to support the XPath module was successful in the case of the latest version (IE8) only. Earlier versions of IE present difficulties that remain unresolved at this time.

The browser security issue mentioned in 3.5 above is known as the “same origin” restriction [10]. This restriction is designed to prevent “man in the middle” security breaches [11] by preventing script code loaded into the page being viewed from communicating with any host except for that which supplied the page itself. This is not an issue where the pages are authored to support annotations as the web server and annotation server can be located on the same host computer, as will be the case with the ALA portal.

However, a requirement exists to enable users to view and create annotations on pages which have not been written or modified to support the required JavaScript code. So-called Plug-in code bypasses this restriction as it effectively becomes part of the browser functionality and is assumed to be “trusted”, meaning that the user understood the implications of installing the plug-in and accepted the potential risk, trusting the code vendors to behave ethically. Anecdotal evidence suggests that many users do not fully understand the risk they take in installing browser plug-in code. IT departments do, hence the frequently enforced edict against plug-in installation.

To facilitate the annotation of unadorned web pages, “*bookmarklet*” technology was adopted. Bookmarklets appear on a page as hyperlinks that can be saved as bookmarks in the browser “favourites”, or with the exception of IE, may be placed in the web browser tool-bar space. Instead of a link to a web page on a specific server, they contain executable JavaScript. When a user wishes to create or view annotations on a page not containing the required scripts, a Page Repeater bookmarklet can pass the URL of the page to remote host which loads the page, injects the required code, then returns the altered page to the browser. The original host will supply all the associated images etc, but the Repeater host is seen by the browser as the page source, allowing it to provide and receive annotations. Testing has shown instances where this scheme breaks in various ways, but for a significant majority of cases, it provides the required annotation capability. Future work will expand the coverage of the Page Repeater for types of HTML authoring now known to cause problems.

<!----- notes for selective expansion in the full paper ----->

- \* Use of Annotea “standard”
  - \* it is not a standard ... just a draft
  - \* short-comings as a standard include:
    - \* "by example" rather than clear specification
    - \* no clear specification of minimum requirements for an annotation
    - \* no treatment of schemas for annotations
    - \* query language is minimal / inadequate
  - \* REST-ful nature of Annotea is a blessing and a curse
    - \* annotea annotation ids are resolvable URLs
    - \* but this presents problems if annotation store needs to be

migrated

- \* Use of OAI-PMH
- \* Use of Triple Stores instead of relational database
  - \* implied Annotea model allows arbitrary triples to be included in annotation / reply
    - \* RDBs would have difficulty representing this
  - \* annotation == blank-node closure
  - \* allows SPARQL queries
    - \* we don't expose this in the public APIs
  - \* problems:
    - \* no standard (vendor independent) Java APIs for triple-stores
    - \* performance and reliability issues for some open source implementations.
- \* Server-side implementation issues:
  - \* import / export of annotations:
    - \* the identity problem
    - \* annotation collections
  - \* provenance of harvested / imported annotations
  - \* annotation schemas - no standards
  - \* annotation contexts - no standards for representing them -  
xpointer is incomplete
    - \* what is the user annotating?
      - \* the web page, or the information displayed in the web page?
        - \* annotation of ALA records
        - \* annotation of images that may be displayed in multiple pages
      - \* the problem of changes to the annotated page (say) - e.g. rendering contexts invalid, annotations irrelevant or ephemeral
    - \* noise annotations; e.g. annotating the page footer:
      - \* need mechanism to allow something (e.g. the page, the annotation service) to say what / how things can be annotated.
    - \* scalability - how would you support millions of users annotating millions of pages?
  - \* Client-side implementation issues:
    - \* the "same-source" security restriction limitations and importance
    - \* trusted versus non-trusted client-side code (javascript)
      - \* trusted plugin potentially places user's computer at risk
      - \* untrusted requires a trusted relay service
    - \* difficulty of cross-browser implementation:
      - \* Microsoft / IE is the big problem area, starting with its non-standard DOM implementation
        - \* cross-browser (trusted) plugins require client-side multiple codebases
          - \* might Chrome Frame be a solution for IE, or does this actually make the problem worse in the long term?
            - \* is it better to say "IE is not supported ... use FireFox/Chrome/Safari/Opera"?

---

[<sup>1</sup>] <http://www.itee.uq.edu.au/~ereseach>

[<sup>2</sup>] <http://www.w3.org/2001/Annotea/User/Protocol.html>

[<sup>3</sup>] <http://www.openarchives.org/OAI/openarchivesprotocol.html>

[<sup>4</sup>] <http://annozilla.mozdev.org/>

- 
- [<sup>5</sup>] <http://www.w3.org/2002/12/AnnoteaProtocol-20021219>
- [<sup>6</sup>] <http://java.sun.com/developer/technicalArticles/WebServices/restful/>
- [<sup>7</sup>] <http://alcme.oclc.org/oaicat/>
- [<sup>8</sup>] <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf>
- [<sup>9</sup>] <http://www.w3.org/DOM/>
- [<sup>10</sup>] [https://developer.mozilla.org/en/Same\\_origin\\_policy\\_for\\_JavaScript](https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript)
- [<sup>11</sup>] <http://www.itee.uq.edu.au/~eresearch/projects/diasb/index.php>