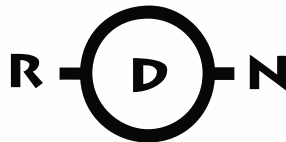

IMESH TOOLKIT
RDN ANNOTATION SERVICE
End of Project Report



by Gregory J. L. TOURTE
UKOLN
The University of Bath

August 2003

IMesh
T O O L K I T



Contents

1	Summary	2
1.1	Project's Aim	2
1.2	History	2
1.3	Results	3
2	Background	5
3	Methodology	7
3.1	UML Design Approach	7
3.2	XML based operation	8
4	Architecture	9
4.1	Database Design	9
4.2	Functionality	11
4.2.1	Rating	12
4.2.2	Annotation	13
4.2.3	Output Formats	14
4.2.4	Moderation	17
4.3	Issues	19
4.3.1	Security	19
4.3.2	Extensibility	19
4.3.3	Portability	19
4.4	Usability	19
4.5	Example of Implementation: the RDN	20
5	Conclusion: Future Developments and Possible Uses	21
	References	23

1 Summary

This document summarises the most pertinent technical results arising from the IMesh Toolkit Annotation Service implementation subproject. The Annotation Service is a JISC funded community-driven metadata layer designed to operate above existing networks, such as the RDN, as a separate non-disruptive layer.

1.1 Project's Aim

The major aims of the Annotation Service project were originally stated as follows;

“To provide users of a resource retrieval system such as the RDN with the capacity to comment upon the usefulness and other virtues or faults of resources as represented in the resource description or summary. In other words with the annotation system it would be possible to create and view a series of comments upon the resource descriptions associated with each description.” [12]

However, it became clear at a later stage in the design that the ability to overlay a comment/markings structure above a given set of resources is, depending on the amount of contextual data retained, quite powerful. The objective could more generically be stated as an application of the technique of community moderation—in the form of a metadata repository—in order to provide a means by which resources included in the RDN can be annotated by users representative of any number of distinct communities. As such, it is a direct precursor to more elaborate community-based moderation schemes and intimately related to (although distinct from) popular technologies such as collaborative filtering. This was also discussed in Richard Waller's Ariadne article [11], although he didn't go very deeply into precisely how he would be likely to use the community provided information, there are a good deal of potential applications available.

As part of the conclusion to this document, we attempt to provide suggestions as to useful directions in which this project could be further developed to take greater advantage of these possibilities.

1.2 History

The IMesh Toolkit project ¹, funded by JISC, was designed to encourage international collaboration amongst subject gateways. The Annotation Service represents one of several components designed towards this end, and was initially proposed by R. HEERY, M. DUKE and R. WALLER for Stage 2 of the IMesh project, with the initial design work completed by R. WALLER [12]. The Program architecture is discussed in Section 4, on page 9 of this document. The original specification is discussed in detail in Mr. WALLER's article

¹The IMesh Toolkit project can be found at <http://www.imesh.org/toolkit/>

available in Ariadne [11].

Due to various discussions with a domain expert, certain elements of the specification were somewhat altered between the design stage and the implementation.

The redesign required arose from a number of root causes. Mainly these were usability issues —such as an excessive number of required ‘click-throughs’ before accessing any annotation information or functions, which meant that the user had no opportunity to develop a tacit understanding of the rating system’s existence and aims by continual exposure, instead having to actively seek out the information by exploration of links (such as ‘more information’) linking.

Feasibility issues were encountered, such as the lack of a clearly defined best practice policy on the topic of authentication —indeed, the original paper made no mention of authentication in any form. Security issues included a lack of traceability— the possibility of logging activity, for example, was not discussed. This issue is closely mirrored by a usability issue in the moderation system as initially designed, which did not permit a moderator to track the existence of successive revisions within the system, leading perhaps to inconsistent behaviour on the part of moderators who suffer from a lack of contextual information.

The following design has arisen from work undertaken in the IMesh Toolkit which has been identifying specific tools which might be useful to developers of subject gateways. In this context, it was decided to undertake work upon the usefulness and functionality of annotations in a variety of contexts and for a variety of users.

In focusing upon a design for a specific service as a means of identifying the likely design implications, it was resolved to take as a starting point a service such as the RDN onto which an annotation service could be layered. The intention was to produce a design that had as little impact as possible upon the existing service. In effect this meant that the annotation system in its basic design did no more than work with the URL corresponding to a description of a resource as its starting point.

In this way it was anticipated that the system could be adopted by a number of subject gateways, since the annotation system did not depend unduly upon other system specific data.

1.3 Results

We built a system using a MySQL backend running on Slackware Linux 9.0 with MySQL server version 3.23.56 with the database architecture as described in Section 4.1 on Page 9. The system functionality is depicted in Fig. 1.

MySQL uses rather pure (though slightly incomplete) SQL as a query language. It does not support stored procedures; however, this is considered to be advantageous given that the use of stored procedures often leads to a badly fragmented codebase. The choice of SQL implies high portability, since the choice of a different backend database will not greatly influence the functionality of the Web interface scripts.

A database structure design was developed, a primary aim of which was to retain close accordance to published MySQL best practices as far as the architecture and access methods were concerned. As such, tables were designed such that their maximum size should not exceed reasonable limits (to retain portability on all systems).

The Web scripts are themselves written in Perl, with the aim of ensuring compatibility with Perl installations of version 5 and above. They are designed to communicate by means of XML-based output, which is produced by means of a Perl database handling module, DBM, twinned with XML generation code within the database access script. The XML generation code was based around an XML module distributed by the CPAN archive[6]; however, the original code was too limiting in its application scope and therefore required extensive modification.

This output may be correctly formatted by means of an appropriately designed XSLT (XML stylesheet).

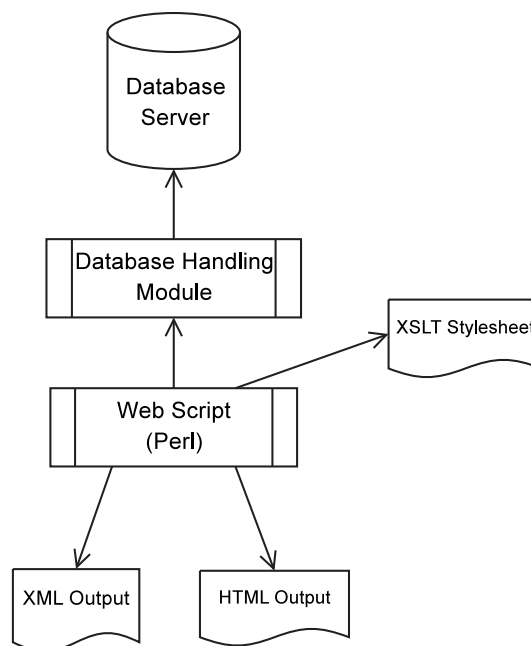


Figure 1: System Functionality Diagram

2 Background

The concept of community based metadata layers has a reasonably long history. Many notable projects or technologies involve similar concepts. Companies whose business technologies largely depend on this concept include Amazon, IMDB, cddb/freedb, Slashdot, Freshmeat or Google.

Collaborative filtering is now commonly used in everyday life; for example, Amazon.com make use of collaborative filtering based on probabilistic analysis of their user community in order to recommend books to amazon.com customers. They also make use of a simple scoring technique together with user-submitted reviews in order to provide customers with some feedback about the product, and to recommend related objects. Much of this specific technology is, however, extensively patented and/or commercialised in the private sector. Many less-known techniques also exist along similar lines.

By way of introduction, it is useful perhaps to introduce the vocabulary used within the community; in the context of the Resource Delivery Network (RDN), the objects catalogued within the RDN are known as resources. The metadata that catalogues these objects (Dublin Core metadata), that makes up the RDN catalogue, is of course also a resource. However, for the purposes of this paper the cataloguing metadata will be referred to as a metadata resource. The annotation information that is collected is also metadata (data about data), and will be referred to, following the previously established convention, as annotation metadata.

It is notable with respect to the examples given above that in effect, community contributed (or based) metadata layers holding diverse information (*e.g.* authoring information, opinions, ratings. . .) are generally collected and retained in order to provide a means of filtering content —to provide some form of context for the user to establish the likely relevance of the content in question. The annotation layer implemented in this project is unusual in this respect, as only the marking element of the annotation is really suitable for filtering purposes; the annotation layer can contain contributed content of any nature.

One point that perhaps deserves further attention in the future is the loosely specified nature of the annotation; perhaps imposition of some form of semantic overlay would increase the usefulness of any given annotation. Examples might include categorization of comments into complaints, positive feedback, or associated links.

A second point that at this stage the system does not support is the concept of appending contextual information to the annotation. As an illustrative example, consider the user who searches for ‘tree structure’, hoping to find information on a class of data structures, but through some mischance finds themselves on an RDN article that discusses horticultural

ture. The feedback left by such a user will very probably be of use to very few people indeed, but it is only through the contextual information (his or her input into the search engine that directed the user to the article) that it is understandable. In other words, without the underlying tale, users' opinions may be less useful.

Therefore, linking the annotation metadata to additional metadata providing contextual information may be of interest.

Again on the topic of context, annotation per article, whilst certainly valuable, is by no means the limit of the system's theoretical capability. With minor changes as well as the addition of a certain amount of browser-side development (*e.g.* Scripted activity on the browser) it would equally be possible for users to 'highlight' sections, and to 'scribble in the margins' in a far more literal sense. This approach conserves valuable contextual information (*e.g.* The exact line which the user feels to be inaccurate, or the exact letter that the user suspects to be erroneous due to an OCR (scanning) flaw, or the paragraph that reminded the user of a related paper to which he/she would like to refer).

Considerable work has been put into approaches to digital annotation, although the web-based multi-layer approach would appear to be somewhat unusual; certain of these approaches are referenced below. Many deal with digital annotation of printed documents, with approaches running from HP's Cooltown (RFID, infra-red beacon) style system to barcode marking. Digital annotation on digital paper has at the least an equally distinguished history. Many, such as InterNote, concentrated on annotation as a technology supporting collaboration [2]. Some, such as Brush et al's work on annotation of digital documents, apply specialised software platforms and interface elements for the purpose (in their case, a Microsoft Word-like WYSIWYG style interface aids users to visualise annotations within that context) [8]. Some systems go to the opposite extreme, however; systems such as the Third Voice utility allow users to 'graffiti' the Web with Post-it Note lookalike comments ².

Excellent reviews exist on the history, for example, of ComMentor ³, or the Xerox XLibris system [10], as well as the envisaged future of annotation; for example, [1], or [9].

²<http://www.wired.com/news/technology/0,1282,20101,00.html>

³<http://hci.stanford.edu/commentor>

3 Methodology

The approach we have taken was chosen knowing the fact that the annotation system had to be designed as a completely independent layer. For this reason we started to develop the service without any connection to any other existing services. The plan was to build a service that got information from POST or GET HTTP request and sent XML as a response.

In a second stage, once the backend of the service was built, we tried to integrate it into another service. We chose for that to use the RDN as it was hosted on the local server and therefore easily accessible.

The implementation project began by comparing and contrasting the existing specification to a number of situations (use cases) in order to test the suggested implementation with respect to the aims of the system. This step, though short in terms of time, is often invaluable —not only to test one’s understanding of the system, but also to apply the team’s experience and knowledge of best practices to the best advantage, often avoiding unpleasant surprises at some later step.

The elements in the UML Design were used in order to split the project code into ‘chunks’ of functionality at the established interfaces; for example, the web scripts were separated into their functional elements, the database, and the output format. The various elements were then considered separately.

3.1 UML Design Approach

As the UML design approach was used previously in the project, the decision was made to continue with its use in designing the revised functionality of the system.

The only element that required major alteration during the design step was the moderation system (*cf. RDN Annotation Use Case Main Model* of the UML Design document particularly as compared to Fig. 3). This decision was the result of discussions held over use cases, some of which implied that the initial design was flawed in the sense that its usability was rather low; as it was, the design did not permit the moderator, when looking at re-submitted (edited) articles, to link those resubmissions to the previous (rejected) submission. However, the alteration in question implied that the true complexity of this element of the design was rather higher than it was originally considered to be.

3.2 XML based operation

The choice of XML, formatted by application of an XML stylesheet template (XSLT) was driven by the following factors; the need to describe a standard format for the data returned, particularly ensuring that the potentially less portable database access code was restricted to as few pages as possible, and the need to provide a simple but effective templating system that would be easily accessible by future maintainers of the code.

XML seemed an obvious choice; as a data description language, it is now in reasonably widespread use on the Internet, and for good reason; it provides an excellent structure on which data may be marked for their semantic content. As a standard, it is easily accessible and there are a number of popular applications available with which XML may be manipulated. The existence of the XSLT standard [13] sealed the decision, since this standard, also well supported by various (free and commercial) applications, provides an excellent method of transforming XML documents to provide a given layout. The existence of a simple and well-supported templating language means that the code's maintainability is substantially better than might otherwise be the case.

Furthermore, the overheads required in producing and processing XML/XSLT, particularly with a low-overhead solution (compiled code) are relatively low.

However, whilst it will eventually become possible with next-generation browsers to rely on pages designed specifically in the XML/XSLT combination, this is not currently possible; the reasons for this are essentially evident in the statement that many popular browsers do not provide a clear, dependable or consistent implementation of XSLT rendering. Therefore, the result in XML/XSLT is transformed into HTML on the server side and then included into a standard web page.

The choice was taken to make use of an open source software called Xalan [5], which is a subset of the Apache project [4]; this is perhaps one of the less portable choices made in the project, but it is clear that this transformation could be achieved by any number of programs on any system which may not support Xalan, if any; for example, Microsoft .Net provides trivial access methods for producing such a transformation, although the large overhead potentially involved with the use of such a large framework makes its use for such a restricted purpose rather inadvisable. In any case, Xalan should be compilable on any platform which includes the GNU tools, or some reasonable analogue, and some form of Posix layer. Therefore in the event of any difficulties, a system such as Cygwin [7] should be sufficient to ensure its portability.

4 Architecture

The annotation service is based around one database containing at least three tables, as shown in Fig. 2, and one main script for the annotation part of the service. A similar script is required for the moderation.

4.1 Database Design

The database is split, for the purposes of efficiency, portability and convenience, into two tables; rating and annotation. Content is held in one or more table(s), for table size management reasons. The architecture chosen is as follows:

The first table contains the rating information for each URI. The only information required for the rating part of the service are the URI to which the rating is associated, the average rating, and the number of entries. There is no need to keep every single entry as the rating is, in a way, an independent feature of the service and can be used by anyone. Moreover it would be a waste of space and an unnecessary additional layer of complexity.

The second table contains most of the data used for the annotation part of the service. All the information is spread over several tables in order to avoid problems with some filesystem on which the file size is limited —with MySQL, each table is stored in one file; when the table grows, it might reach the maximum filesize limit of the filesystem (2GB on FAT16/32, 2TB on ext2/ext3⁴). In order to assure compatibility and portability on different system, the decision was made to limit the table size to 2GB[3] (we have disregarded compatibility to the FAT12 and MINIX filesystems, which have a file size limit of 32MB and 64MB respectively, for obvious reasons). This limitation means that only 32,000 comment can be fit into one table, so a new table should be created every 32,000 comments.

⁴This 2TB limitation on the ext2 and ext3 filesystems is valid if “Large File Support” is enabled, otherwise it drops to 2GB.

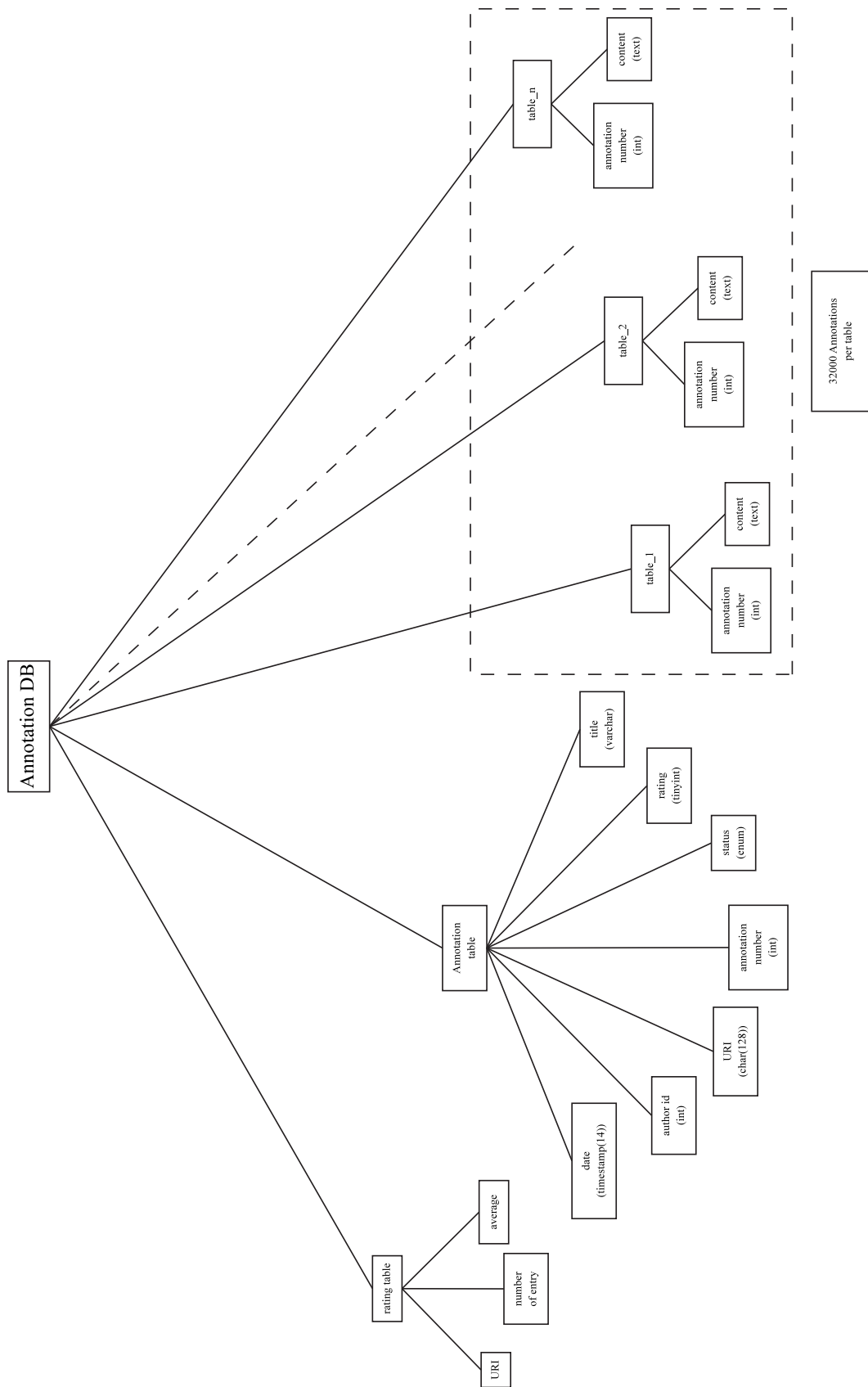


Figure 2: Database Layout

4.2 Functionality

Although the functionality of the work was described in the specs, it did not include detailed information on the likely database configurations or choices of code architecture. These elements were worked on, as mentioned previously, as modular elements. The design map used throughout can be seen in Fig. 3.

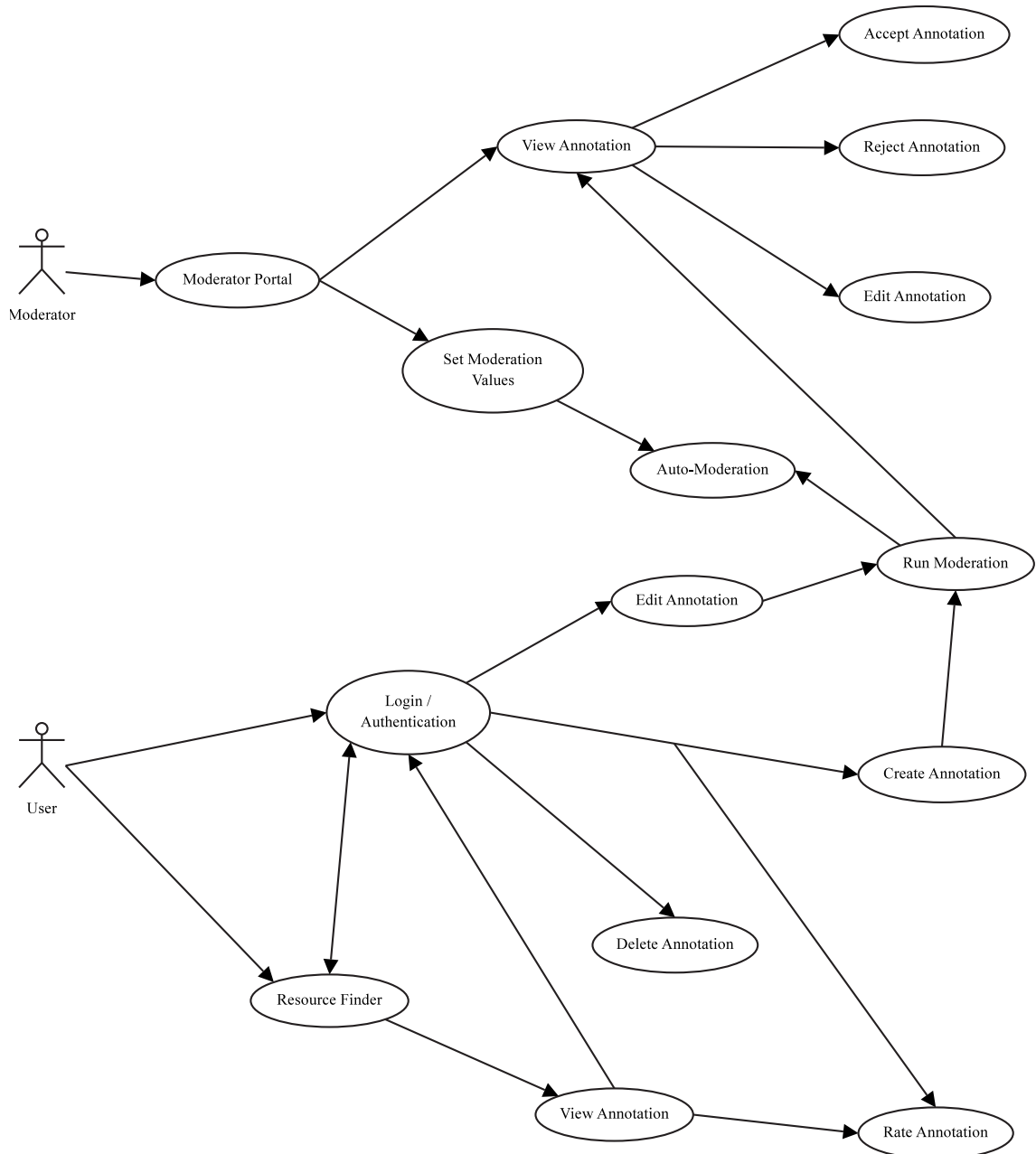


Figure 3: Functionality design diagram

Directory Layout

For purposes of both tidiness and independent operation, the following layout was chosen; Annotation and Moderation elements of the software were each kept in separate directories. This has the useful, though incidental, side-effect of massively simplifying many security and authentication questions; for example, an authentication method such as Apache's inbuilt `.htaccess` authentication method is trivial to apply in the context of restricting access to a given directory.

Running the script

The Apache server is set up to automatically serve the default 'front-end' cgi for each set of functionality. Therefore, no permanent URIs are involved; this has the advantage of increasing portability and transparency between different server-side languages (*e.g.* PHP as opposed to server-side Perl), meaning that the URLs used will not become obsolete with a simple change in backend technology.

4.2.1 Rating

- Adding A Rating
- Reading a URI Entry
- Initialising a URI

The rating functionality, from the API perspective, consists of a number of discrete functions —the actual rating of a URI being only one element of this. URIs do not initially have 'unrated' entries, for example; they are entered into the database with a 'zero average', 'zero ratings' entry during the first search that returns the URI in question. If one enters any URI without previously searching, it will also create the entry for the manually entered URI, using the same process.

Addition of new ratings and reading ratings for a given URI are, naturally, comparatively straight-forward. Reading requires only that one access the database entry for a given URI and return the values 'average rating' and 'number of ratings' corresponding to that URI. Adding a new rating to a previously-rated resource operates as follows.

$$\text{New rating} = \frac{\text{Old average} \times \text{no. of ratings} + \text{New rating}}{\text{no. of ratings} + 1}$$

The newly calculated data then simply replaces the old data previously held within the table.

4.2.2 Annotation

The annotation functionality consists of the following:

Reading an Entry

We initially access the annotation table, with a given URI as argument, returning all annotation metadata IDs whose subjects correspond to the given URI. For each ID, the information is then returned from the annotation table. The actual annotation content body is retrieved from the relevant annotation content table (see below for details). This results in a form of software ‘table join’ —the decision was taken not to use a direct SQL table join, largely in the interests of portability. Table joins are one of a number of SQL functions with relatively inconsistent behaviour over various implementations, in the author’s experience.

The information output at this point is then taken and formatted in XML. It is further processed into the required output format, in a separate process described in ‘Output Formats’, below.

Adding a New Entry

The first step in inserting a new entry is to insert the non-content information from the new annotation (*e.g.* author, rating and so on) into the annotation table. From this step is returned the annotation number, set as an auto-increment in the Annotation Table; each annotation has a unique annotation number and this annotation ‘ID’ serves as a general index.

This number is then used to define the Annotation Content table used. There are potentially several annotation content tables, for database management reasons to do with the maximum allowable table size. The annotation content table number is retrieved from the unique ID as follows;

$$\text{ID div } 32000 + 1$$

where 32000 is the maximum allowable number of entries per table, and where ‘div’ is the integral division. The annotation body is then added to the annotation content table whose number is returned by this function, along with the index annotation number.

Checking/Creating a New Content Table

One might ask what would occur when one is adding the first entry into a new annotation-content table (the 32,001st!), as the table does not yet exist.

An additional function (called as part of the process of adding a new entry) exists to solve this problem; it checks for the existence of a given content table. Should this table turn out to be as yet nonexistent, it creates it according to an inbuilt (hard coded) database

definition.

Adding Rating to Rating Table

Ratings can also be added as part of annotations; this is to permit the rating provided by an annotator to be displayed as part of their annotation. In this case, the rating is stored in the Annotation table, and then added to the rating table as described in the rating functionality section. In this sense, the rating provided is cloned—it is stored in the Annotation table as a raw value, and it is also added to the Rating table.

4.2.3 Output Formats

The output to the client is done in either XML via PERL CGI, or in HTML generated on the server from the XML and an XSLT transformation. The layout of the XML differs slightly depending on the function the page is supposed to have (*e.g.* the page function could be, for example, viewing of a given annotation, or the page governing the annotation of a given resource). In order to make the system lighter, we use one XSLT file for all the transforms, hence the different XML layouts. The layout will be explained here briefly.

Showing rating information for a given uri

```
<IMesh_Annotation_Service>
  <header>
    <uri />
  </header>
  <RESULTS>
    <Ratings>
      <average />
      <numberofratings />
    </Ratings>
  </RESULTS>
</IMesh_Annotation_Service>
```

Showing annotation information for a given uri

```
<IMesh_Annotation_Service>
  <header>
    <uri />
  </header>
  <RESULTS>
    <Ratings>
      <average />
```

```

        <numberofratings />
    </Ratings>
    <Annotations>
        <Entries />
    </Annotations>
</RESULTS>
</IMesh_Annotation_Service>

```

Showing all annotations for a given uri

```

<IMesh_Annotation_Service>
  <header>
    <uri />
  </header>
  [...Raw resource DC metadata record...]
  <RESULTS>
    <Annotation>
      <id />
      <author_id />
      <title />
      <creation_date />
      <rating />
      <content />
    </Annotation>
    <Annotation>
      <id />
      <author_id />
      <title />
      <creation_date />
      <rating />
      <content />
    </Annotation>
    [...]
  </RESULTS>
</IMesh_Annotation_Service>

```

User preview of a annotation before submission

```

<IMesh_Annotation_Service>
  <header>
    <uri />
  </header>

```

```
<RESULTS>
  <Preview>
    <author_id />
    <title />
    <creation_date />
    <rating />
    <content />
  </Preview>
</RESULTS>
</IMesh_Annotation_Service>
```

Annotation form

```
<IMesh_Annotation_Service>
  <header>
    <uri />
  </header>
  [...Raw resource DC metadata record...]
  <RESULTS>
    <form>
      <id />
    </form>
  </RESULTS>
</IMesh_Annotation_Service>
```

Confirmation output after user submission of an annotation

```
<IMesh_Annotation_Service>
  <header>
    <uri />
  </header>
  <RESULTS>
    <Confirmation>
      <id />
    </Confirmation>
  </RESULTS>
</IMesh_Annotation_Service>
```

Error message output

```
<IMesh_Annotation_Service>
  <RESULTS>
```

```

    <Error>
      <message />
    </Error>
  </RESULTS>
</IMesh_Annotation_Service>

```

4.2.4 Moderation

Earlier in this paper, a usability issue with the original design of (human-supported) moderation was identified. This issue dealt with non-tracability, the problem being that moderators could not tell a revised, resubmitted comment's origin —*e.g.* that the resubmitted comment was not linked to the original. The implication of this was that the moderator had no way of telling that the comment was revised, nor any way of accessing the moderator comments or feedback that may precede the current entry. This, the moderator lacked sufficient contextual information to provide a consistent response with relation to previous events, leading to decisions that could potentially appear arbitrary appearing decisions.

A suggested moderation system follows; it would require the use of a separate 'pre-moderation' database, from which comments would be moved on acceptance (presumably, the threads of discussion that precede would also be removed, or archived, on the basis of their age or inactivity levels). Moderator comments would, at the same time as being emailed/communicated to the user as applicable, be added to the database with a field indicating their place in the discussion. Successive revisions of the user comment would be appended similarly; thus, the system has similar characteristics to a basic discussion-board system.

The database might look as follows;

Comment context	ID	Thread depth	Correspondent
A comment	1	0	Site user
Re: A comment	1	1	Site Moderator
Re:Re: A comment	1	2	Site user

This database table also holds all of the information required for the Annotation and Content table, for transfer when accepted. Refer to the UML diagrams showing the original scenario (Fig. 4) and the revised scenario (Fig. 5) for details. This discussion information should be displayed wherever it is useful to inform actors' decisions; the fact that decisions are fully informed should lead to a consistent user and moderator experience and behaviour.

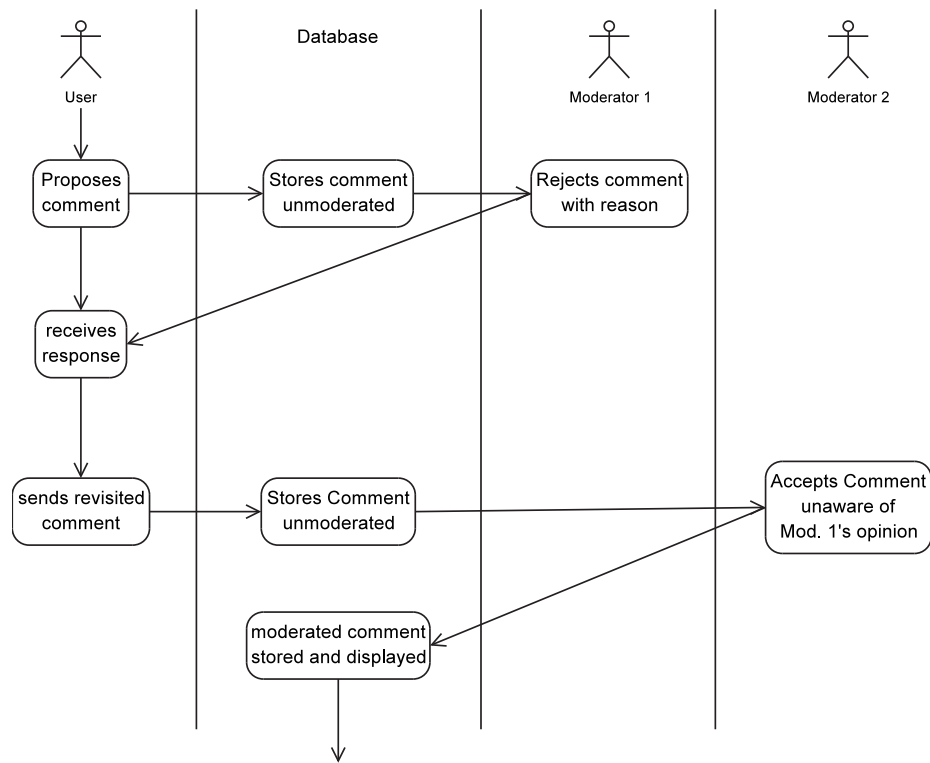


Figure 4: Previous Moderation Model

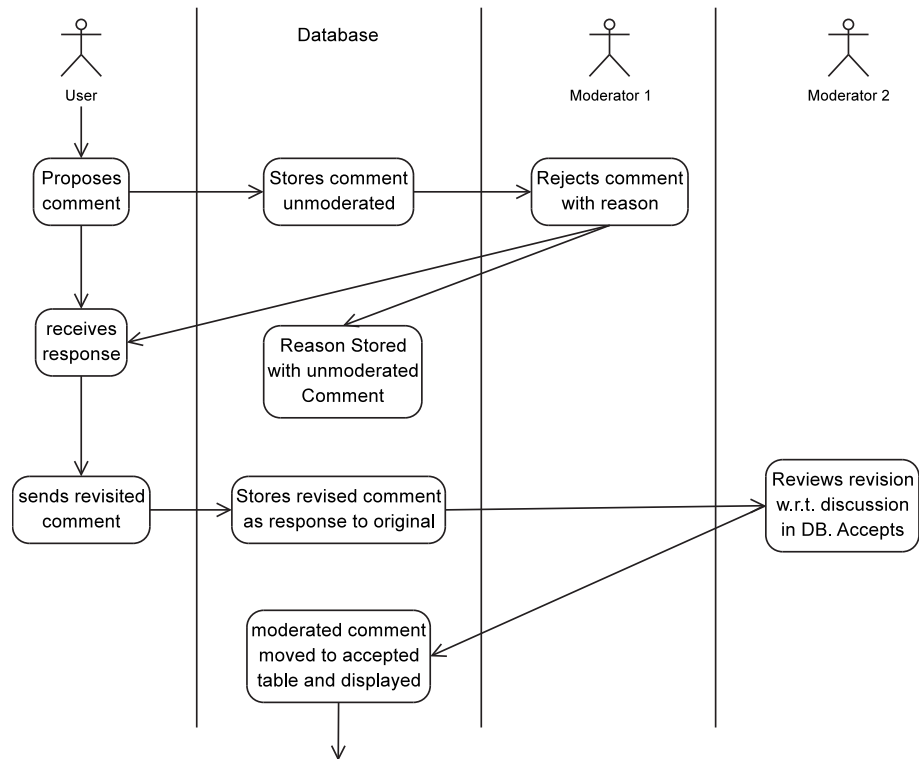


Figure 5: Moderation Replacement Proposal

4.3 Issues

4.3.1 Security

At this stage, the system requires a number of additional alterations before it could be considered as acceptable for a production system.

For example, it is absolutely necessary that a check is made on the number of accesses/ratings made by a given IP address (or UID —unique session ID, usually made up of a random hash twinned with hopefully unique elements such as exact time that the session was established). This would guard against a user intentionally (or even unintentionally) sabotaging the system by repeatedly rating the same resource, perhaps because of bias or even (although unlikely) malicious intent, to influence the rating of a resource to appear poorly or well rated.

4.3.2 Extensibility

The possibilities include going from the current setup, one central database, to include additional databases on separate computers, perhaps with an hourly database propagation. Equally it is possible to spread the information across a number of database servers, should this be necessary. The downside is of course that additions to the database, depending on the chosen replication methods and technologies, would spread relatively slowly.

4.3.3 Portability

The open-source nature of the elements used within this project implies that they are widely available, on a range of platforms (Linux, Solaris, BSD, Windows, and so on). The platform of SQL and Perl is relatively stable in terms of compatibility, meaning that no more than cosmetic changes to the code are expected or indeed likely to become necessary in order to keep up with new developments in the language specification. The system largely depends on elements that are almost ubiquitous, and therefore highly unlikely to become obsolete or leave the current code behind in terms of syntax.

4.4 Usability

As far as usability is concerned, we chose to concentrate on attempting to keep a simple, clean interface. We restricted functionality from the annotation system user's point of view to include only a pull-down list for providing the numerical 'score', with a link to the full annotation commenting system, such that the options were presented as either scoring

or annotation (with optional scoring).

The interface was tested by three users against Nielsen's heuristics, known as a 'quick and dirty' discount usability engineering method (the ten-part version was used⁵). However, before the system's public use it would be highly desirable to conduct a fuller usability study; Nielsen's heuristics reach peak effectiveness, on average, when applied by at least five users⁶. Equally, previous familiarity with the system may have influenced the results by 'hiding' certain usability flaws from the evaluators.

4.5 Example of Implementation: the RDN

A running example of the Annotation system has been implemented during the design, for the purposes of troubleshooting, real-time bug tracking, demonstration and feature testing. Although not elsewhere publicly advertised, it is available to public access at www.rdn.ac.uk/rfdev/. It uses largely the same behaviour as the main RDN site, with the annotation functionalities added. A Moderation subsystem implementation is not, however, available at this time.

⁵http://www.useit.com/papers/heuristic/heuristic_list.html

⁶http://www.useit.com/papers/heuristic/heuristic_evaluation.html

5 Conclusion: Future Developments and Possible Uses

The implementation of the Annotation project has clearly demonstrated the concepts underlying the system to be entirely feasible, with, of course, the previously mentioned alterations and reservations. The Moderation subsystem required the use of authentication methods, currently under investigation within the JISC project umbrella at this time, in order to be realistically implemented. Several authentication projects under development will soon become available; using any such suitable project, it is possible to enable an implementation of the moderation system in the real world.

Amongst other applications, annotation could also be considered as searchable metadata. This would permit a wider reach for site or content searching mechanisms. During this project, the idea was considered, but constraints due to the design made it impractical—the definition of the project stated clearly that, as a completely independent service, its existence or otherwise should not in any way influence the operation of the resource finder. As such, it would have been inappropriate to link the Resource Finder's operation to the annotations.

As an example scenario, one might imagine a researcher who reads a paper in a given context and, whilst annotating the document, drops a reference to several other resources or concepts. A researcher searching on one of these associated concepts may therefore be referred to this article as a consequence of the annotation; in this way, one might imagine that searching 'backwards' from annotation information may provide, perhaps surprising, and perhaps useful associations.

At last, to wrap up this conclusion with a question, introduced as a statement; the possibilities provided by the storage of annotation metadata are certainly wide enough. Annotation metadata alone is a potentially rich resource; when combined with other information, such as contextual background information, the possibilities are endless. Applying contextual information, for example, could help the sorting through search terms to locate more relevant resources. What other metadata exists that could contribute to the usefulness of an annotation?

One idea might be annotation in the form of links, with a given semantic content—semantic linking between learning objects. If an article deals partly with the character of Van Gogh, for example, but also deals with Monet, Manet and da Vinci, a link might be made between the Van Gogh-topic and a biography of the painter's life. This has more value as a link than as a simple textual annotation, perhaps? The interface used for the current annotation system does not yet permit us to provide such rich semantic content in any practical way—research, previously mentioned in these pages, has dealt with interfaces that enrich the user's ability to provide such contextual detail quickly and easily, for

example, by providing ‘highlighter-pen’-like abilities to highlight the text that one wishes to annotate within the paper[9]. Is this the future of annotation?

References

- [1] The future of annotation in a digital (paper) world.
- [2] ACM. *Internote: Extending a Hypermedia Framework to Support Annotation Collaboration*, ACM Hypertext '89 Proceedings, New York, 1989.
- [3] MySQL A.G. MySQL Manual. <http://www.mysql.org>.
- [4] The Apache Software Foundation. <http://www.apache.org/>.
- [5] The Apache XML Project. <http://xml.apache.org/>.
- [6] Comprehensive Perl Achive Network. <http://www.cpan.org>.
- [7] Cygwin. <http://cygwin.com/>. A UNIX-like environment for Win32.
- [8] A. J. BERNHEIM BRUSH, David BARGERON, Anoop GUPTA, and Jonathan J. CADIZ. Robust Annotation Positioning in Digital Documents. In *CHI*, pages 285–292, 2001.
- [9] GÖTZE, SCHLECHTWEG, and STROTHOTTE. The Intelligent Pen: Toward a uniform Treatment of Electronic Documents. Technical report, ACM, 2002. <http://doi.acm.org/10.1145/569005.569024>.
- [10] SCHILIT, GOLOVCHINSKY, and PRICE. Beyond Paper: Supporting Active Reading with Free-Form Digital Ink Annotations, 1998.
- [11] R. WALLER. Functionality in Digital Annotation: Imitating and supporting real-world annotation. <http://www.ariadne.ac.uk/issue35/waller/>.
- [12] R. WALLER. Annotation Service UML Design. Technical report, UKOLN, 2002. <http://www.imesh.org/toolkit/work/components/annotation/design/docs/>.
- [13] W3C XSLT Standard Specification. <http://www.w3.org/TR/xslt>.