Distributed Tagging and Annotation of Computer-Checked Proofs

Eliot Setzer[*]

Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh

Abstract

Users of theorem proving systems often complain about lemma management issues within their own proofs and have even more difficulty locating and reusing appropriate definitions and formalized theorems available from third parties. Both users' searches for and understanding of lemmas can be improved by associating relevant *metadata* with portions of formal proofs. This paper describes an architecture based on web annotation techniques intended to lower the barriers to spur-of-the-moment and collaborative creation of metadata for formal proofs. The architecture allows user annotations and both user-defined and automatically generated tags to be represented in a unified and prover-agnostic way.

1 Introduction

A wide variety of mathematical results have been formalized using theorem proving programs such as Isabelle [14], Coq [5], Matita [2], and Mizar [16]. These programs operate by processing commands contained in *proof scripts* (which in some cases may be only a part of larger *proof documents*) that describe how to build low-level representations of proofs. Proof documents are written and maintained manually, and this process presents challenges similar to those presented by ordinary source code as well as some challenges specific to formalized proofs.

Practitioners of proof formalization consulted during this research say that the most restrictive bottlenecks in the process of developing formalized proofs often come from the difficulty in locating lemmas needed at a point in the proof. The user may know that the needed lemma's definition exists somewhere, under some name, within the libraries or in the rest of the project's code base, and it may be easy to describe what the lemma states in natural language, but it can still be difficult to find among all the other lemmas. Research aiming to make the search for lemmas more systematic has mostly focused on improving search techniques and on ways for authors to include more detailed and search-friendly metadata within their proofs' text. An example of the first is dynamic indexing of saved theorems in Matita [2] and an example of the second is the literate programming system for Isabelle ([14] Chapter 4). A third, but related and complementary, possibility is to develop ways to associate metadata with proof documents without actually embedding it into them (since adding text to the documents themselves is not always possible or desirable).

To this end, this paper proposes a unified architecture for annotation and tagging of proof documents based on techniques developed for web annotation systems. *Web annotation systems* are intended to allow users to post comments about portions of any document with a URI without relying on the server hosting the document to provide comment thread management facilities. This goal is achieved by storing each comment (or a URI for the comment) together with the URI of the item it annotates in a tuple on an annotation server and providing a way to query the server for all annotations associated with a given document. While posting and retrieving comments is the intended application of web annotation, many web annotation systems are very general and can be seen as simply frameworks for defining functions from URIs to lists of URIs. Sections 3 and 3.1 describe ways of applying web annotation to proof documents.

[*]. `eliot.setzer@ed.ac.uk`; http://purl.org/NET/eliot-setzer/current-root

If support for general annotations on proof documents is made available in this way, then tags can be implemented on top. *Tags* (or more precisely *tag instances*) are simple, machine-readable annotations consisting primarily of an identifier called the *tag name* that act to group all objects tagged with the same tag name and that, in particular, allow a list of all items tagged with the same tag name to be retrieved efficiently. *Collaborative* tagging systems that permit members of a large collaborating audience to create tags, such as CiteULike and Del.icio.us, have recently become popular due to the useful structure that can emerge from many very simple tag addition operations [7]. Using the system for web annotation of proof documents it defines earlier, the paper describes an encoding of collaborative tags and the application of tags represented in this way to proof documents (Sections 3 and 3.2).

Automatically generated tags are an important case, and section 4 considers ways for these tags to coexist with manually created ones in the system. An integrated tagging system should allow user interface programs to work with both basic information like the location of identifiers' definitions and more insightful tags and annotations created by mathematicians in a unified way. Two experimental implemented systems for extracting tags automatically and exporting them to a web annotation server are described as well.

This paper appears to be the first to explicitly consider ways to make it easier for authors and other users of proof documents to add metadata directly linked to the objects defined in existing documents. To summarize, the architecture, which has been developed with this idea in mind, allows annotations and tags to be created collaboratively and represented within a unified infrastructure while not being tied to a particular theorem proving system.

2 Challenges of Proof Script Organization

This architecture for tagging and annotation is intended to address the lack of relevant metadata related to objects that appear in proof documents. More metadata would be useful both to human readers attempting to understand proof documents and to search engines indexing the objects in proof documents to improve the experience of users looking for lemmas. Examples of the kind of metadata that would be relevant include concise descriptions, tags, comments on lemmas' applicability, comments on proof technique, requests for theorems missing from theories, links to places where lemmas are used, and citations referring to alternative proofs or background on the underlying theory.

The two main reasons that there is not more of this kind of metadata associated with proof documents are that authors often prefer to move on to prove the next lemma rather than documenting the last one and that readability may suffer if proof documents are overwhelmed by metadata. These can be said to be problems of opportunity cost and verbosity, respectively.

It is hard to argue with authors' judgements about the opportunity costs of writing more proofs versus documenting just-written ones, even though authors who have just finished a proof are usually by far the best prepared to document it. Instead, it is worth considering other ways that metadata might be created. Specifically, authors can go back and add metadata later, other users of the definitions in a proof document can contribute metadata, and programs can analyze the proof documents and related data to create metadata automatically. One reason that minor metadata-related changes are not common is that, once finished, proof documents are often inconvenient to modify in place since all other scripts that depend on them likely need to be rechecked by the theorem prover each time. This is an important problem since it is often only after a lemma is referenced in several places that a user may decide it is important to document it well. If the proof document has been distributed to others, as is necessary for the second method (in which other users add metadata) to apply, then the problems with in-place documentation increase substantially. If the distribution is limited to a small group's versioning repository, then, in addition to the need for all of the document's users' provers to do rechecking, it is possible for the change to cause a merge conflict. In the case of wider distribution, updates by the author would need to be announced, or more likely deferred to the next release, and the addition of metadata by users outside of the core development team is very unlikely. Automatically generated metadata can be added at any stage that is convenient (this is described in section 4).

In any case, if metadata were added to proof documents at each possible stage just described, the documents would become much longer and probably less readable due to both the lack of a coherent structure, such as that that could be established by a single author, and the sheer verbosity of the metadata. To avoid these problems, a common approach is to restrict editing of metadata within proof documents to the author (or to a small group of authors) and to also select a common policy for metadata formatting across a project. Selecting a common policy avoids the verbosity of attempting to store all metadata possible and allows users to get a feeling for what metadata is and is not likely to be available regarding a lemma they are searching for, which prioritizes their search strategies. Example policies that limit verbosity and guide authors' insertion of metadata and users' expectations for metadata include:

- name all results with verbose descriptive names and mark the most important ones by using the `theorem` keyword rather than the `lemma` keyword
- generate most lemma names automatically, abbreviate the others, but put verbose descriptions in comments when appropriate[1]
- use literate programming facilities, where appropriate for the theorem proving system (for example, Chapter 4 of [14] for Isabelle)

Programmers face similar challenges when searching for functions, classes, etc., and deal with them in analogous ways, including naming conventions, commenting, and literate programming. However, searching for lemmas often becomes much more of a bottleneck when writing proofs than searching for functions is when writing programs because authors of formalized proofs frequently have the advantage of working directly from a well-understood but less formal version of the same proof. This means that proof authors spend more time in translation details such as remembering lemma names and less time thinking about novel solutions to the problem.

Also, metadata ranking the importance of particular proved results is much more useful than metadata ranking the importance of objects in source code would be. This is because, once the theorem prover has checked the correctness of an important proof, it is truly irrelevant which lemmas were involved. There is no chance that a bad lemma will subtly affect the behavior of the result contrary to the statement of the theorem, whereas, when investigating a tricky bug in a program, almost any piece of program source code could be at fault and all parts are of equal importance. Currently the only support that many theorem provers provide for tagging the importance of results is the choice between using a `lemma` keyword or a `theorem` keyword when declaring results (internally, the keywords are synonyms). Traditional mathematical language has many other categories such as fundamental theorem, main result, technical lemma, corollary, and trivial corollary. These gradations of importance and relationship to the overall theory are even more important in proof documents, where the importance of results always ranges from trivial details necessary to convince the automatic proof checker to the useful theorems motivating the theory, than in ordinary mathematical text.

3 Applying Web Annotation and Tagging

The proposed architecture applies the idea of web annotation and extends it to allow annotations on portions of proof documents and to allow the representation of tags in terms of annotations. Web annotation explicitly addresses the problems related to comments and search criteria having to be directly embedded into documents by completely separating annotations from the underlying documents. It also allows incompatible theorem-organization policies to coexist, each visible only to those who find it useful but all potentially useful data for search engines to index.

While many extensions and extra features are possible, the fundamental property of a web annotation server is that it accepts URIs and returns a list of URIs. This is a simplistic view in several ways, most notably because servers allow posting of new annotations, usually return metadata as well as URIs, and may return plain text annotations instead of URIs[2]. Nevertheless

1. Using meaningless lemma names such as automatically generated ones is considered a legitimate approach because it encourages users to use lemmas, instead of repeated in-line proofs, without having to worry about naming each lemma

2. Systems can get around the limitations of servers that return only plain text annotations plus metadata by encoding URIs as plain text and marking them as such in the metadata

this view is a useful abstraction and it essentially describes the only property that systems for web annotation have in common.

Various systems for web annotation exist (some are described in section 5), but the Annotea standard [9, 17] produced by the World Wide Web Consortium (W3C) is the most general and best-documented system in use and was chosen for use in this project. Though it is only a draft proposal and has not been updated since 2002, it is implemented by many more systems than competing standards: implementations of Annotea clients in the form of simple command line programs [19], browser plug-ins [18], and web services [8] are available. There is a similar variety of Annotea servers; [11] was used for testing.

While web annotation systems allow annotations on any type of file, XML and HTML are the only formats for which systems commonly allow annotations on portions of files (Annotea uses `xpointer()` fragments [21] to do this); this means that it is necessary to extend Annotea in order to store annotations on portions of proof documents. There are several possible ways to do this. The simplest system is to store a line number-column number pair. Other systems, such as storing two line number-column number pairs (to allow ranges within files to be specified as well using the start point and end point) and more syntax-specific systems, are possible. The system using two line number-column number pairs is normally sufficient when the proof documents are versioned well and not modified in-place, and it does not depend on the underlying proof syntax. Section 4 describes a way to handle the issue of versioning using automatically generated tags.

Extending the semantics of annotations to represent tags is not difficult, but there are important issues that can affect the robustness and extensibility of systems for doing so. For example, the system chosen for mapping from tag names to URIs (to store in the annotation relations) and back should be designed for extensibility of both tag name and URI syntax in anticipation of more semantically rich extensions of the tagging idea. Also, annotations used to represent tags instead of something else must be clearly distinguishable as such by programs looking to operate on tags. It is similarly important to allow users of tagging-unaware programs to filter out tags using the basic facilities of the annotation protocol so that they can view the non-tag annotations associated with a document without the large number of very simple annotations functioning as tags overwhelming the (few but important) other annotations.

3.1 Annotated Proof Documents

For a user, the first step in creating an annotation would be to select text to annotate within a user interface for viewing or editing proof documents. This would either be done directly (using a pointing device or cursor movement commands) or indirectly, for example by selecting an existing annotation and indicating that the new one should use the same range as the existing one (or, as described in section 4, that the new one should apply to the existing one itself).

The user interface or some middleware would then generate a standardized representation of the annotation, and in particular generate the URI identifying the document or portion of a document to which it applies. This is usually the most critical portion of a web annotation from an implementation perspective. Unlike the URI pointing to the annotation body, it must be expressive enough to work around existing limitations in the structure of the document it annotates (such as failing to make every portion that might usefully be referenced addressable through a meaningfully named fragment identifier) and be as robust as possible against potential future changes to the annotated document itself, or to its location, made by the publisher. In many applications of web annotation, these challenges simply translate into fragility of the annotations; annotations placed in stories that appear within the front page of a news organization's site are unlikely to work as intended for long, for example.

As an informal aid to users, proof document annotations need not be any more reliable than any other web annotations, but reliability is always desirable and a lack of at least a basic level of reliability could make the system useless. The reliability of the URI pointing to the annotated document is dependent separately on the reliability of the base URI and the reliability of the URI fragment: it is important both that the base URI continue to point to a document and that the document remains as it was when originally annotated (for some applications, the document may change in ways that preserve the meaning of the original annotation's positions, or that is likely to do so).

For concreteness, the examples below describe the steps in posting an annotation on the lemma `Zero_not_Suc`. `Zero_not_Suc` is defined in the file `src/HOL/Nat.thy` relative to the root of the Isabelle 2007 distribution, on lines 75-76. It states that for any natural number m, $0 \neq$ `Suc` m, where `Suc` is the successor function defined earlier in the file. The statement of the lemma as it appears in the file is:

```
lemma Zero_not_Suc [iff]: "0 ≠ Suc m"
```

There are various approaches to establishing canonical URIs for proof documents and objects within proof documents. In this example the canonical URI for the Isabelle 2007 version of the file defining `Zero_not_Suc` will be defined as:

```
http://purl.org/isabelle/2007/src/HOL/Nat.thy
```

Benefits of using HTTP in place of special-purpose URI schemes such as Matita and Coq's `cic:` scheme [2, 5] include compatibility with existing annotation tools and more freedom for people to host their own proof documents within the space of URIs (using the standard DNS and HTTP infrastructure), but this choice is arguable.

The server purl.org will redirect HTTP requests it receives to requests on another server; for example, for any path `$PATH`, `http://purl.org/isabelle/2007/$PATH` might be configured to redirect requests to `http://isabelle.in.tum.de/2007/$PATH`. The configuration can be changed at any time by the user that created the redirect. purl.org is widely relied upon in academia and, more importantly, is used for some of the canonical URIs in Annotea's own infrastructure [9]. This means that using almost any other domain name could be an additional point of failure, whereas purl.org already is one (but one with a lot of backing behind it).

Continuing the example, the canonical URI for the portion of code defining `Zero_not_Suc` is:

```
http://purl.org/isabelle/2007/src/HOL/Nat.thy#line-column-range(75.1,76.1)
```

This uses a URI fragment syntax based on the XPointer framework's fragment syntax [22] (a much simpler subset of the `xpointer()` fragment syntax [21]) for referencing portions of Isabelle theory files. The XPointer framework is an extensible syntax that allows arbitrary systems of referencing portions of XML files to be defined without namespace collisions, but the same type of syntax can be adopted for the URI fragments of non-XML file formats. In particular, this more specific syntax is defined within the framework in order to refer to ranges of characters given start- and end-points (each represented as line number-column number pairs as described in section 3). In this case the line numbers are 75 and 76 and both character numbers are 1.

Once a complete URI pointing into the proof document to be annotated is assembled, it is combined with the URI for the annotation itself and several other fields containing metadata to form an Annotea RDF document. An Annotea annotation record contains at least the fields [9]:

`annotates`. The document annotated (a URI without a fragment)

`body`. The document constituting the annotation

`context`. The portion of the document annotated (the same URI in `annotates` but with a fragment; this is the URI put together above)

`creator`. An identifier for the creator of the annotation

`created`. The annotation's creation time

`date`. The time at which the annotation was last modified

`type`. A link to the canonical URI representing a defined annotation type

One is more likely to find one's own annotations (and tags) meaningful and relevant than others', and filtering by `creator` can allow only one's own annotations to be shown when that is desirable. Also, users may want to filter annotations based on a combined `creator-created` or `creator-date` criterion, for example with a query such as "annotations made by so-and-so when he was refactoring this file to separate statements about natural numbers from those about positive integers." Use of the `type` field for distinguishing tags from other annotations is described in section 3.2.

Annotea servers support five operations, posting, querying, downloading, updating, and deleting, and the distinction between `annotates` and `context` is related to the distinction between querying and downloading. Most notably, `annotates` is the only field on which Annotea servers are required to allow queries, so in order to find *any* annotations that apply to a document, *all* the annotations that apply to it must be requested. The annotations returned can be filtered by the client, but querying does not return annotation bodies in order to avoid sending unneeded annotation bodies to the client at all. The downloading operation is then used to request the full details about a particular annotation.

3.2 Representing Tags

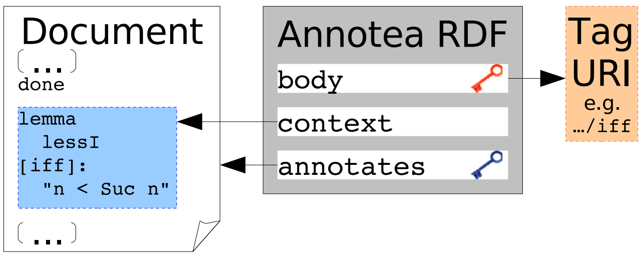

Figure 1. The representation of a tag using a single annotation. If an annotation server supports queries based on the `body` URI as well as the `annotates` URI (which is an extension), then fully featured tags can represented using a single annotation.

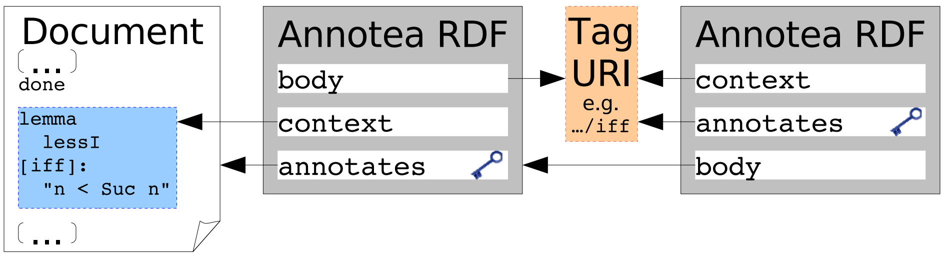

Figure 2. The representation of a tag using a pair of annotations. Annotation servers can by default only be queried based on the value of the `annotates` field (marked with a key icon), which necessitates the second annotation for moving back from the tag URI to the places where the tag is used.

The first step in implementing tags using Annotea is to define an annotation type for tags, so that they can be identified as tags and treated appropriately. The default types of annotation described in [9], `SeeAlso`, `Question`, `Explanation`, `Example`, `Comment`, `Change` and `Advice`, are specific to the manual use of annotations, but the same paper encourages the creation of other types of annotation for special purposes. The process of defining new annotation types is described in [10]. Briefly, this involves writing an RDF schema for the new types, then setting the `type` field of the annotation metadata to the canonical URI defined for the new type.

A representation of tags using a single annotation is shown in Figure 1. Assuming that web annotation services are merely functions from URIs to lists of URIs, representing each instance of a tag as a single annotation does not allow tag instances to be recovered on a per-tag basis because there is no way to query the server other than by asking for annotations based on the page that they should appear to annotate. This is a limitation even for the normal use cases of web annotation, since users are able to link to a particular annotation but are then unable to determine which document(s) it annotates, but is a fundamental problem for tagging.

The problem can be solved by adding a facility to the annotation server to allow querying based on the `body` URI (indicated by the extra, red key in Figure 1) as well as on the `annotates` URI. This solution is not especially easy to add to existing servers, however, and it may be preferable to add support for a fully general RDF query language such as the Algae query language supported by the W3C annotation server [17] or the more modern and now-standard SPARQL [20] given the additional benefits of doing so.

Alternatively, each tag instance can be well represented by a pair of annotations: one from the instance to a URI representing the tag and one from the URI representing the tag back to either the instance or the resource describing the annotation itself. If the second annotation points back to the instance, then there is no need for the underlying web annotation system to define URIs providing direct access to the annotation resources but it is necessary for clients to query the server and filter out the annotations that link back to the tag URI in order to find the tag instances. Since Annotea does define URIs for each annotation [17], which are created and returned to the client upon posting, an Annotea-based implementation of tags can set the second annotation to point back to the RDF annotation description resource and avoid the reverse lookup and filtering overhead. This solution is shown in Figure 2.

The implementation of tag extraction described in the next section adopts neither of these solutions since no client with a need to find tag instances given only the tag URI has yet been developed.

4 Automating Tag Extraction

Besides tags manually added by users using tag-enabled Annotea clients, it is possible to automate the creation of Annotea-based tags by extracting tags that exist in other formats. The experimental implementation includes two forms of tag extraction from Isabelle proof documents. The first simply creates Annotea equivalents of tags that users can manually add to the files using special comment syntax, and the second produces Annotea equivalents of Isabelle's internal representation of identifier definitions' locations.

A program for extracting tags that are syntactically encoded into proof document comments is useful both to seed the tag database and to allow authors to easily add tags to the proof document text using any text editor for later export to Annotea servers. Even though the extraction could be done by a simple lexer searching for comments and tags within them, the current implementation uses Isabelle itself to parse the proof scripts into PGIP XML [4] (a format used for communication between theorem proving systems and the Proof General [3] user interface system), then extracts the comments from the resulting XML. This was done to establish infrastructure for other potential systems to extract tags or other information from PGIP XML.

Various syntaxes for tags within comments are possible. The current version simply treats any whitespace-delimited word prefixed with a "`@`" character as a tag name. Such a simple syntax has the benefit of being relatively non-invasive: one way to use the system is to take an existing proof document, look for important keywords that already appear in the comments, and prefix them with "`@`". Figure 3 shows examples.

```
subsection {* The @classical @axiom *}
axioms
    classical: "(~P ==> P) ==> P"
```

(a)

```
{** The classical axiom
 * tags: @classical @axiom
 *}
subsection
axioms
    classical: "(~P ==> P) ==> P"
```

(b)

Figure 3. (a) Example of tags added to an existing comment by prefixing words with "`@`" (b) More conventionally, the same syntax can also be used in a way analogous to the "`@tag`" field defined by documentation generators. Of course an even more compatible syntax is also possible, in which tags are only those words (not "`@`"-prefixed) that appear on a line after "`@tag`".

The implementation of tag extraction from Isabelle's internal tags, which represent the location of definitions, works by streaming ML commands to Isabelle and processing the results. While many systems have support for this type of tags already, bringing them into the same infrastructure as annotations and user-defined tags may make both implementation and operation of tagging-aware editors (for both manual and automatic tags) simpler and more uniform.

However, adding a large number of automatically generated tags to annotation servers may overwhelm the user-generated annotations. To prevent this, manual and automatic tags can be given different values of `type` (as described in section 3.2) to distinguish them. Any Annotea client that supported filtering of the annotations displayed based on simple criteria such as the value of `type` would then have no problem hiding one type of annotation or the other as required by the user.

Probably the most important application of automatically generated tags is that they can be regenerated every time a proof document changes, and so can be used as a level of indirection to allow other tags and annotations to move seamlessly from one version of a proof document to updated versions (and even older versions). That is, rather than putting every tag and manual annotation related to an identifier on the region of text containing its definition, users can instead put all tags and annotations (at least the manually entered ones) on the URI associated with the automatically generated tag for the identifier. This way, each time a new version of the proof document is created, a new tag linking the location of the definition in the new version to the tag URI can be added automatically and all of the existing tags and annotations applying to the identifier as a concept will apply to its definition in the new version as well. Figure 4 shows an example of the arrangement of manual and automatic annotations to allow this.

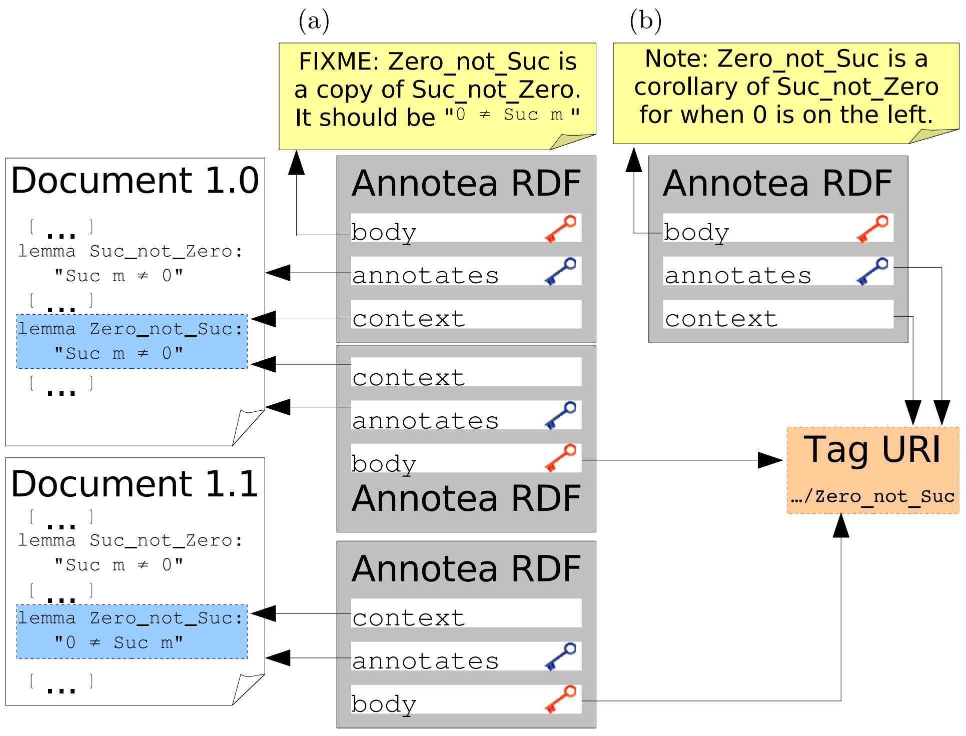
(a)
(b)
FIXME: Zero_not_Suc is a copy of Suc_not_Zero. It should be "0 ≠ Suc m"
Note: Zero_not_Suc is a corollary of Suc_not_Zero for when 0 is on the left.
Document 1.0
[...]
lemma Suc_not_Zero: "Suc m ≠ 0"
[...]
lemma Zero_not_Suc: "Suc m ≠ 0"
[...]
Document 1.1
[...]
lemma Suc_not_Zero: "Suc m ≠ 0"
[...]
lemma Zero_not_Suc: "0 ≠ Suc m"
[...]
Annotea RDF
body
annotates
context
Annotea RDF
body
annotates
context
context
annotates
body
Annotea RDF
Annotea RDF
context
annotates
body
Tag URI
…/Zero_not_Suc

Figure 4. An example of how annotating (or tagging) the URI for an automatically generated tag allows manually added tags and annotations to automatically apply to the right proof object even in new versions of the proof document. Annotation (b) is intended to apply to *Zero_not_Suc* as an abstract concept, no matter how the expression of it changes between document versions. Its `context` and `annotates` fields point to the tag URI, so it is accessible from any version of the document via the tag. Annotation (a) is very specific to version 1.0 of the document, so it annotates that version directly and it will not automatically apply to later versions. In this diagram tags are represented using the system in Figure 1 for conciseness, but the system in Figure 2 would work as well.

5 Conclusion and Related Work

Early systems for web annotation include a single-user annotation feature in the Mosaic web browser [13] and Röscheisen, Mogensen, and Winograd's [15]. The idea gained more attention with the start of the commercial service Third Voice [9] in 1999. The Annotea standard was developed starting in 2000. Presently Annotea coexists with several incompatible commercial services as well as some alternative approaches to the annotation problem, such as Wikalong (which associates a wiki page with each URI) and ShiftSpace (which provides an architecture for switching between multiple views of a web page each produced by a different plug-in).

Dmitriev et al. [6] describe applications of web annotation systems to improve the quality of domain-specific search results, and Asperti et al. [1] describe ways of using and explicitly extracting metadata related to proof documents (without using web annotation) for use in searches. [1] is not the only system for searching through proof libraries as almost every theorem proving system has at least one such facility, but it brings most of the important ideas in this area together in one system.

Currently the most common approach to dealing with useful but potentially distracting metadata verbosity is code folding. This is a simple and effective approach to dealing with verbosity in both the data and metadata present in program or proof code. However, since it does not address the factors discouraging minor metadata-related changes to proof documents, normally the only metadata that it will help to hide is that that the document's author(s) thought to include when originally writing the proofs. Nevertheless, code folding could be useful in combination with web annotation-based metadata when it is desirable to keep certain kinds of metadata embedded in the proof documents. More complex combinations of annotation, tagging, and code folding could be useful too, such as having an editor fold the bodies of all the theorems in a file other than those tagged `important`.

Future possibilities include more implementation, especially implementation of a tagging and annotation client within a proof document editing interface since this is the major missing component. Users' experiences with a basic implementation would be very valuable for prioritizing the addition of other features. Like Annotea web annotations themselves, compatibility with semantic web ideas is a goal of this architecture and it is possible to extend it to include new features and greater semantics in various ways. Marchiori has considered representing mathematical statements in full detail as RDF [12], taking this idea to its logical conclusion. This is an important long-term goal because of the potential for integrating formalized knowledge with the rest of the semantic web. However, starting with informal annotations keeps things simple, so that a system could be made usable relatively straightforwardly: representing metadata using web annotation has a well-defined set of prerequisites, many of which are already satisfied by well-tested subsets of the functionality of existing open source annotation tools.

Acknowledgements The author would like to thank Scott Owens, Georges Gonthier, Jacques Fleuriot, David Aspinall, as well as other members of the Dream Group at the University of Edinburgh for ideas about use cases for annotated and tagged proofs. The idea of tagging objects in proof documents comes from Jacques Fleuriot and David Aspinall. David Aspinall also provided many ideas to improve the paper's organization. The author was introduced to Annotea by Katsuhiko Tsujino, Masafumi Iwata, and Satoko Sakajō at Mitsubishi Electric.

Bibliography

[1] Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. A content based mathematical search engine: Whelp. In *Types for Proofs and Programs International Workshop, TYPES 2004 (LNCS 3839)*, pages 17–32. Springer, 2006. `http://www.cs.unibo.it/~sacerdot/PAPERS/types2004_whelp.ps.gz`.

[2] Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. User interaction with the Matita proof assistant. *Journal of Automated Reasoning*, 39(2):109–139, August 2007. `http://www.cs.unibo.it/~tassi/matita.pdf`.

[3] David Aspinall. Proof General: A generic tool for proof development. In *Proceedings of TACAS 2000 (LNCS 1785)*. Springer, 2000. `http://homepages.inf.ed.ac.uk/da/papers/pgoutline/`.

[4] David Aspinall, Christoph Lüth, and Daniel Winterstein. A framework for interactive proof. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants, 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007, Proceedings (LNAI 4573)*, pages 161–175. Springer, 2007. `http://homepages.inf.ed.ac.uk/da/papers/pgipimp/`.

[5] The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 8.1 edition, July 2007. `http://coq.inria.fr/V8.1/refman/index.html`.

[6] Pavel A. Dmitriev, Nadav Eiron, Marcus Fontoura, and Eugene Shekita. Using annotations in enterprise search. In *Proceedings of the 15th International Conference on World Wide Web*, pages 811–817, Edinburgh, Scotland, May 2006. International World Wide Web Conference Committee, ACM Press. `http://www2006.org/programme/files/xhtml/6006/6006-dmitriev-xhtml.html`.

[7] Scott A. Golder and Bernardo A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006. `http://www.hpl.hp.com/research/idl/papers/tags/`.

[8] Dominique Hazaël-Massieux. Annotea client in XSLT. Downloadable open source software, December 2003. `http://www.w3.org/2003/12/annotea-proxy`; runnable using the form at `http://www.w3.org/2003/12/annotea-show`.

[9] José Kahan, Marja-Riitta Koivunen, Eric Prud'hommeaux, and Ralph R. Swick. Annotea: An open RDF infrastructure for shared web annotations. In *Conference Proceedings of WWW10*, Hong Kong, May 2001. `http://www.www10.org/cdrom/papers/488/index.html`.

[10] Marja-Riitta Koivunen. *Defining New Types*. World Wide Web Consortium, January 2004. `http://www.w3.org/2001/Annotea/User/Types.html`.

[11] Andrea Marchesini. Annotea server version 0.8. Downloadable open source software, December 2007. `http://autistici.org/bakunin/annotea/`.

[12] Massimo Marchiori. The mathematical semantic web. In Andrea Asperti, Bruno Buchberger, and James H. Davenport, editors, *Proceedings of the International Conference on Mathematical Knowledge Management (LNCS 2594)*, Bertinoro, Italy, February 2003. Springer. `http://www.w3.org/People/Massimo/papers/`.

[13] National Center for Supercomputing Applications. *Annotations*, xmosaic2.4 edition, 1994. ftp://ftp.ncsa.uiuc.edu/Mosaic/Documents/Unix/XMosaic2.4ASCII/Chap4.txt.

[14] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. `http://www4.in.tum.de/~nipkow/LNCS2283/`.

[15] Martin Röscheisen, Christian Mogensen, and Terry Winograd. Beyond browsing: Shared comments, SOAPs, trails, and on-line communities. In *Proceedings of the Third International Worldwide Web Conference*, Darmstadt, Germany, April 1995. `http://www.igd.fhg.de/archive/`.

[16] Piotr Rudnicki. An overview of the MIZAR project. In *1992 Workshop on Types for Proofs and Programs*, pages 311–332, Båstad, Sweden, June 1992. `http://www.cs.ualberta.ca/~piotr/Mizar/Doc/MizarOverview.ps`.

[17] Ralph R. Swick, Eric Prud'hommeaux, Marja-Riitta Koivunen, and José Kahan. *Annotea Protocols*. World Wide Web Consortium, 2002-12-19 edition, December 2002. `http://www.w3.org/2002/12/AnnoteaProtocol-20021219`.

[18] Matthew Wilson, Doug Daniels, Jeffrey Phillips, Mike Lee, Katsuhiko Tsujino, and Eliot Setzer. Annozilla. Downloadable open source software (version 0.7.a.2), December 2006. `http://annozilla.mozdev.org/`.

[19] World Wide Web Consortium. *SWAD-Europe Annotea Tools Readme*. `http://www.w3.org/2001/sw/Europe/200209/annodemo/readme.html`.

[20] World Wide Web Consortium. *SPARQL Query Language for RDF*, 2008-01-15 edition, January 2008. Edited by Eric Prud'hommeaux and Andy Seaborne; `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.

[21] World Wide Web Consortium XML Linking Working Group. *XPointer xpointer() Scheme*, 2002-12-19 edition, December 2002. Edited by Steven DeRose and Eve Maler and Ron Daniel Jr.; `http://www.w3.org/TR/2002/WD-xptr-xpointer-20021219/`.

[22] World Wide Web Consortium XML Linking Working Group. *XPointer Framework*, 2003-03-25 edition, March 2003. Edited by Paul Grosso and Eve Maler and Jonathan Marsh and Norman Walsh; `http://www.w3.org/TR/2003/REC-xptr-framework-20030325`.