# Wiki Grows Up: Arbitrary Data Models, Access Control, and Beyond

Reid Priedhorsky
IBM T.J. Watson Research Center
Cambridge, Massachusetts, USA
reid@reidster.net

Loren Terveen
University of Minnesota
Minneapolis, Minnesota, USA
terveen@cs.umn.edu

## ABSTRACT

Ward Cunningham's vision for the wiki was that it would be "the simplest online database that could possibly work". We consider here a common manifestation of simplicity: the assumption that the objects in a wiki that can be edited (e.g., Wikipedia articles) are relatively independent. As wiki applications in new domains emerge, however, this assumption is no longer tenable. In wikis where the objects of interest are highly interdependent (e.g., geographic wikis), fundamental concepts like the revision and undoing must be refined. This is particularly so when fine-grained access control is required (as in enterprise wikis or wikis to support collaboration between citizens and government officials). We explore these issues in the context of the Cyclopath geowiki and present solutions that we have designed and have implemented or are implementing.[1]

## Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Computer-supported cooperative work, Web-based interaction

## General Terms

Algorithms, Human Factors, Reliability, Theory

## Keywords

Wikis, data models, access control, geographic wikis, geowikis

## 1. INTRODUCTION

Over the past decade, a new model of collaborative knowledge synthesis and production [2] has emerged, based on a simple yet fundamental innovation: *invert the publishing model*. Review work *after* publication, not before.[2]

This new *wiki* model is successful. Users in fact do perform the work of creating and synthesizing content as well as the meta-work of reviewing, correcting, and organizing. For example, Wikipedia, the largest and most successful wiki, has amassed over three million encyclopedia articles in English alone [23] and is generally considered to be roughly as accurate as traditional encyclopedias [8]. Wikia, a for-profit wiki company, hosts over 100,000 individual wikis [13]. And mathematical models developed by Cosley [3] suggest that the wiki model develops the same final quality as classical publishing but does so faster.

A second fundamental innovation of the model is *maximally open editing*. Whenever reasonable – and the wiki philosophy calls for a very generous interpretation of "reasonable" – any reader may make and publish changes (without pre-publication review), and anything can be changed. (Note that obstacles to open editing can be subtle, such as outmoded policy, an unfriendly user interface, technical quirks, or excessive territoriality.) Finally, an important special case is that undoing work must be easy: any change can be easily reversed.

Finally, these two innovations require a third in order to work: *transparent changes*. It must be easy for all readers to see how the wiki is being changed, and by whom. Specifically, wikis need *recent changes lists* summarizing the flow of changes in the wiki or parts of it, *watching* to enable automatic notification of users when changes of interest occur, and *diffing* to let reviewers see and analyze precisely what changed in particular revisions. This transparency is essential for enabling the meta-work tasks (such as reversing undesirable edits) which make the wiki model viable.

These three innovations – post-review, open editing, and transparency – are the fundamental properties which define the wiki model. However, like any technology, wikis are maturing. In this paper, we explore the important implications of two key dimensions of maturation.

First, wiki practitioners have discovered that in real-world wikis, limitations on editing are usually needed, and these limitations must be more subtle than allowing a given user to view or edit either the whole wiki or nothing at all. *Fine-grained access control* – the ability to specify who can or cannot view and edit particular subsets of the wiki – is required. Even traditionally very open wikis need this; for example, Wikipedia sets articles with severe vandalism or edit warring problems to a "protected" or "semi-protected" state, making them editable only by administrators or established users, respectively. And for wide adoption of wikis in enterprise settings, mature access control is essential, as contractual, business, or cultural constraints may dictate arbitrary controls on editing.

Second, wikis are moving beyond collections of relatively independent text-focused articles. For example, there is significant progress in mapping, with communities like Open Street Map and Cyclopath building large, rich geographic datasets. Other interesting possibilities include wikis for sketching, animation, or diagrams;

[1]This paper describes work done while both authors were at the University of Minnesota; IBM has no connection with Cyclopath.
[2]This introduction is a revised and extended version of [17].


WikiSym '11, October 3-5, 2011, Mountain View, CA, USA.

for maintaining an annotated bibliography with a citation graph; or for any other task dependent on a structured data model. In other words, wikis are moving into domains where multiple, highly interdependent, often non-textual objects must be visible and editable at once. In order to accommodate these developments, wikis must store not an independent state (and history) for each individual object but rather a single global state (and history) that coordinates individual objects' states. This notion is very similar to the *ACID* model (atomicity, consistency, isolation, and durability) which has been well-established in the database community for some time [11].

These two new dimensions, particularly in concert, raise significant implementation difficulties. This paper explores these difficulties and proposes solutions, which we have implemented or are implementing in the Cyclopath geowiki for bicyclists. We first consider related work (Section 2) and then outline the state of traditional wiki implementations (Section 3); we then explore the motivation, complexities, and implementation of first moving to a global wiki state (Section 4) and then global state plus fine-grained access control (Section 5).

# 2. RELATED WORK

## 2.1 Beyond public wikis

Wikipedia has received more scholarly and popular attention than any other wiki (or any open collaboration system). However, it is a *public* wiki: editing is open to essentially anyone, with only limited restrictions that have emerged over time. This model is less suitable for wikis and open collaboration systems in other contexts.

**Enterprise Wikis.** Researchers have studied wikis used in corporate settings. A common observation is that access to information must often be limited, for reasons including legal restrictions (e.g., on sharing classified information with foreign nationals) [10] and organizational power relationships or competition between stakeholders [4]. Further, enterprise wiki users often feel strong personal ownership of content they add (and are organizationally accountable for it) and a corresponding unwillingness to edit content "owned" by others [14]. In other words, enterprises often use wikis as a *technology* without accepting the wiki *philosophy* of openness.

**Volunteered Geographic Information.** The field of Geographic Information Systems (GIS) is concerned with collecting and analyzing geographic data [12]. Traditionally, work in this area has been performed by GIS professionals using specialized software. However, in the past few years, the field has become interested in having laypeople collect and contribute geographic knowledge, an approach known as Volunteered Geographic Information (VGI) [9]. However, GIS professionals prioritize precision and data quality quite highly. Therefore, they typically limit the role of volunteered information and "accept" it only after applying automated validation and/or manual expert curation [6].

**Citizen Science.** A similar approach is seen in *citizen science*; for such projects, laypeople collect and enter data that are used to aid scientific inquiry. For example, birders participating in the eBird project enter their observations of birds. Then, "automated data quality filters developed by regional bird experts review all submissions" and "local experts review unusual records that are flagged by the filters".[3] Likewise with the Encyclopedia of Life: "anyone can contribute... but materials from unvetted contributors are initially marked as unreviewed.... EOL curators then identify the best quality contributions and promote them from unreviewed to trusted status." Curators are "professional scientists" and "experienced citizen scientists" who must apply and be approved.[4]

**Summary.** Enterprise wikis, VGI systems, and citizen science are examples where an open approach is desirable but factors such as organizational culture and perceived data quality requirements constrain access policies. We present in this paper our approach to integrate access control into wikis, giving wiki owners necessary flexibility to define appropriate access for specific user groups.

## 2.2 Access control in collaborative systems

Researchers have identified a number of issues relevant to access control in collaborative systems [5, 22]. We highlight four:

- The access control model must support collaborative notions such as groups of users and joint ownership.
- The model must support fine-grained access specification.
- Users should not have total discretion to specify access rights. Both *basic access control* (who can do what to which objects) and *meta access control* (what are the permissible access policies for any given object) are needed [5].
- The model must be understandable and usable. Specifically, we identify three classes of users with different usability requirements. Programmers must understand the model fully and be able to implement any desired access specification, using a table-based language; administrators must understand the basic concepts and be able to select from a small set of meta access policies; and users must understand the different access rights (viewing, editing, etc.) and decide to whom different rights should be granted for a given object.

The basic form of access control we propose in this paper falls under the rubric of *role-based access control* (RBAC): for each access right on each object, we specify an *access control list* (ACL) containing a set of users who have that right. For simplicity of presentation, and because the simplifications are straightforward for implementers to reverse, we gloss over a few details of the model – that the lists can contain groups or roles (e.g., "administrator", "customer service representative"), that in reality most objects point to a few ACLs stored just once, etc. – and assume that each object has an independent ACL containing simply a set of users.

In forgoing more complex access control models [22], we give up some capabilities. Most notably, the model we propose has no notion of *context* – access is the same for any given user regardless of context. For example, we can't specify that a user can edit a map object only if his or her location-aware device reports that it is near the object or has been recently.

We do this for two reasons. First, we have not identified use cases that would require it, and it is consistent with the wiki philosophy to be as open as possible. Second, even relatively simple access control such as RBAC adds significant conceptual and implementation complexity to the wiki model (as we explain below), and adding yet more complexity is something to avoid.

One issue that (to our knowledge) the security community has not addressed is the difficulties that arise when undoing changes in a collaborative system with fine-grained access control. This is a core contribution of our paper.

[3] http://ebird.org/content/ebird/about

[4] http://www.eol.org/content/page/help_build_eol

## 2.3 Version control systems

As in wikis, change management is important in software development, which is also focused around a collection of collaboratively changing, highly interdependent objects (source code and other files). Much intellectual and technical effort has gone into developing *version control systems* (VCSes) which primarily support software development. Our discussion is based on Raymond [20] and Fowler [7]. We identify three generations of VCSes:

- First-generation systems such as RCS and CVS are *file oriented*: while they have a notion of a software project being a unified set of files, each file's history is independently managed. Multiple files changes can be saved (*committed*) at once, but this operation is not atomic. This model is very similar to what we call Wiki 1.0 (Section 3).
- Second-generation systems such as Subversion are *fileset oriented*. These systems have a single global history (called a *repository*) for each project, and commits are atomic. They correspond to our Wiki 2.0 (Section 4).
- Third-generation systems such as git and Mercurial are both fileset-oriented and *distributed*: instead of a single repository, there are multiple repositories which communicate changes between each other as peers, removing the need for a single, centrally managed repository. Might this architecture be Wiki 4.0? We speculate briefly in Section 6.

To our knowledge, version control systems have not grappled with the issues related to fine-grained access control that we have identified and explore as Wiki 3.0 (Section 5).

Another key feature that has steadily grown in sophistication as VCSes mature is *branching*: the notion that the history of a software project is not linear but tree-shaped or even a directed acyclic graph. For example, developers find it useful to let disruptive work continue on the main line of development while a stable version of the software is prepared for deployment, and they also find it useful to *merge* the minor bug fixes made in preparing that stable version back into the main line. Even fairly primitive systems like RCS had branching, and branching and merging in a modern system like git is so good that some observers call it "omniscient" [21]. Branching is relevant to wikis as well, a point we expand on below.

While prior work has addressed both global state management in a set of changing interdependent objects and fine-grained access control in collaborative systems, to our knowledge the intersection of these ideas is unexplored. This is the central contribution of our present work.

# 3. WIKI 1.0

This section describes how traditional wiki software, such as the MediaWiki software used by Wikipedia, works. The goal is to make explicit some of the properties and assumptions that are relevant to the modifications of the wiki model that we suggest; we do not explain in detail how any particular software operates but rather identify the general principles. We first discuss how data are stored and modified and then how access control works.

## 3.1 Core assumptions

Fundamentally, a wiki is simply a collection of objects (such as encyclopedia articles) that can be changed, with the history of changes retained for each object. Specifically, this history forms a sequence of content states; we call each such state a *revision*. A revision is created when a user does some editing work and then saves these changes to the server, making them available to other users. Revisions are atomic: they can't be partly saved, and there is no meaningful state between revisions.

| | | Article |
|---|---|---|
| *Name* | *Rev.* | *Text* |
| Cat | 1 | Cats are mammals. |
| Cat | 2 | Cats are cute mammals. |
| Mouse | 1 | Mice have pointy noses. |
| Cat | 3 | Cats are cute mammals with whiskers. |
| Mouse | 2 | Mice have pointy noses that wiggle. |

**Figure 1: Editing history of two wiki objects, the articles "Cat" and "Mouse". While the articles are stored in the same table, their states are managed independently.**

In traditional wikis, each object has its own independent history (see Figure 1). This is based on two underlying assumptions:

- Exactly one object is seen or edited at once during a single session, and thus a revision affects exactly one object, not part of an object or more than one object.

  Interface sugar is often provided to mitigate this somewhat limiting assumption. For example, one can open multiple browser tabs to view or edit different Wikipedia articles, and this creates multiple independent sessions. On the other hand, MediaWiki lets users restrict editing to a single section of a larger article, but when saved, the revision still creates a new state for the entire article.
- Relationships between objects are weak: changing one object doesn't introduce fundamental problems in other objects or cause larger-scale breakage. (In collections of articles like Wikipedia, the key relationship is the hyperlink; changing a link in Article A affects only Article A, regardless of which article is linked to.)

As a consequence, two revisions on different objects have no meaningful ordering beyond weak inferred relationships such as timestamps. At a sufficiently fine time precision (less than 1 second in Wikipedia), there is no well-defined global state; i.e., at any given instant, the global state is ambiguous. Thus, the *unit of revision* in traditional wikis is "one object".

## 3.2 Undoing work

A critical operation in wikis is undoing saved revisions which are erroneous, malicious, or otherwise undesirable. There are three different ways to accomplish this task; in the following, we assume that the bad revision is revision $N$.[5]

- Manually edit the current state until the undesirable content is removed or corrected, and then save the changes as a new revision. As this technique is effectively identical to normal editing behavior (i.e., not concerned with undoing work), it does not encounter the difficulties this paper is concerned with, and we will not consider it further.
- *Revert* the offending revisions(s) by copying revision $N-1$ and saving it as a new revision $M$. While this technique is simple and always works regardless of the presence or content of revisions between $N$ and $M$, all work after revision

[5]These techniques can be straightforwardly extended to address more than one bad revision, so we omit that discussion for simplicity of presentation.

| Article | | |
|---|---|---|
| *Name* | *Rev.* | *Text* |
| Cat | 1 | Cats are mammals. |
| Cat | 2 | Cats are cute mammals. |
| Cat | 3 | Cats are cute mammals with whiskers. |
| **Cat** | **4** | **Cats are mammals.** |

**Figure 2: History of article "Cat" after reverting revision 2. Note that *with whiskers*, added in unrelated revision 3, is lost.**

| Article | | |
|---|---|---|
| *Name* | *Rev.* | *Text* |
| Cat | 1 | Cats are mammals. |
| Cat | 2 | Cats are cute mammals. |
| Cat | 3 | Cats are cute mammals with whiskers. |
| **Cat** | **4** | **Cats are mammals with whiskers.** |

**Figure 3: History of article "Cat" after reverse-merging revision 2; the editing operations in that revision ("add the word *cute* between *are* and *mammals*") have been reversed ("remove the word *cute* from between *are* and *mammals*") and the reversed operations applied to then-current revision 3. Note that *with whiskers*, added in revision 3, remains.**

$N-1$ is removed (requiring additional editing to retrieve it from history if this content is desirable). Another drawback is that a subsequent revert of revisions in the range $[N+1, M-1]$ will reintroduce the undesirable content.

- *Reverse-merge* the offending revision(s) by computing the difference between revision $N-1$ and $N$, reversing those edit operations, and applying the reversed operations to the current revision $M-1$, creating new revision $M$. For example, if revision $N$ consists of the edit operation "add the word *sasquatch*", then the reverse editing operation is "remove the word *sasquatch*". This technique preserves (perhaps desirable) work in revisions $[N+1, M-1]$, but it is not always possible, because changes in these later revisions can make the computed reversed operations nonsensical. For example, if someone had later changed *sasquatch* to *bigfoot*, the operation "remove the word *sasquatch*" is meaningless. (This technique is called "undo" in Wikipedia; we use the term *undo* more generally to refer to any kind of editing that undoes other editing in full or in part.)

A key property of the two undo techniques is this: *an undo that is undone forms a no-op*. For example, if revision $N$ is undone (either by reverting or reverse-merging), creating revision $M$, and then revision $M$ is itself undone, the combined effect of the two undos is equivalent to doing nothing at all (though editing between revision $M$ and its undo might also be lost). This is important because it preserves the property that any change in a wiki is easily reversible.

Figures 2 and 3 illustrate the distinction between reverting and reverse-merging, respectively. Note that the two are equivalent if the newest revision is the one being undone (i.e., $N = M-1$).

## 3.3 Access control

As noted above, real-world wikis need a greater or lesser degree of access control, which we model using access control lists (ACLs): for each object and each operation (view, edit, etc.), a set of users who can do that operation on that object is specified (and this set can include the anonymous user). For example, in Wikipedia, the ACL for editing most articles is "everyone", while *protected* articles have the ACL "all administrators" and *semi-protected* articles have the ACL "all logged-in users whose accounts meet certain criteria" [24].[6] Similarly, the Lotus Communities product from IBM lets administrators restrict edit and/or view access to wiki pages to members of particular groups, called "communities".

The introduction of access control leads to a key decision: what is the right granularity for access control? Simply put, each object must have its own (logical) ACLs. ACLs which necessarily apply to more than one object are overly aggressive (for example, it would be silly for Wikipedia to protect the entire database just because the article "George Bush" was attracting frequent vandalism), and ACLs which apply to some unit smaller than one object are not needed because objects can be split (and this is particularly so when global state management is introduced below).[7]

Thus, the *unit of access control* is "one object"; specifically, the unit of revision and the unit of access control are the same. In other words, anyone who can edit a page can create an arbitrary revision, including a revision which is a revert or a reverse-merge.

To summarize, while traditional wikis have mature undo facilities and fine-grained access control, they lack global state management. We explore this deficiency in the next section.

# 4. WIKI 2.0: GLOBAL STATE

## 4.1 Motivation and general principles

The lack of a single global state and the inability to edit more than one object at once causes problems. Even in a wiki containing relatively independent objects, some operations necessarily affect more than one object. For example, renaming a Wikipedia article involves changing multiple objects – the article itself as well as every other article that links to it. With no global state, Wikipedia uses the following awkward workaround: (a) change the article's name, (b) create a new article at the old name which redirects to the new name, so links still work, (c) update these links to point to the new name, and finally (d) delete the redirect.

These steps are effort-consuming, non-atomic (to the extent that it can be difficult to ascertain progress), and frequently never completed. This problem affects routine operations as well; e.g., updating an illustration and changing its caption cannot be done atomically, because the illustration is a separate object from the article that contains its caption (and includes the illustration by reference).

These problems are even more acute in wiki systems containing data models with strong relationships between objects. For example, our own interest in these matters was sparked by the process of creating Cyclopath, a geographic wiki designed to meet the navigation needs of bicyclists in the Minneapolis-St. Paul, Minnesota metro area [18, 19]. In the current paper, we discuss the technical underpinnings of Cyclopath's wiki technology in much greater detail than previous publications, with a focus towards its application and extension in rich-data wikis beyond mapping.

[6]The German-language Wikipedia has a notion of *flagged revisions*, affecting which version of an article is seen by default. Because this scheme controls only defaults and does not actually control access, it is not relevant to our discussion.

[7]In certain cases, special powers available to administrators are also useful. For example, Wikipedia has the notion of *hidden revisions* – article states which are invisible to anyone except a small set of wiki administrators. This mechanism is used to hide edits which must be permanently hidden due to privacy or legal issues. Because such special powers do not have a significant effect on our argument, we omit their discussion for clarity.

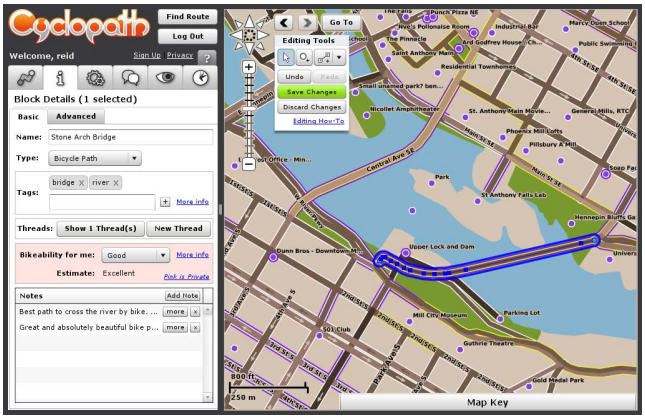


**Figure 4: The Cyclopath user interface. In addition to finding bicycle routes, the system also lets users edit the transportation network and other map objects. The system has over 2,500 registered users and been accessed by 32,000 unique IP addresses. Of the 13,384 revisions to the Cyclopath map, 7,980 (60%) affect the graph and 8,718 (65%) affect more than one object.**

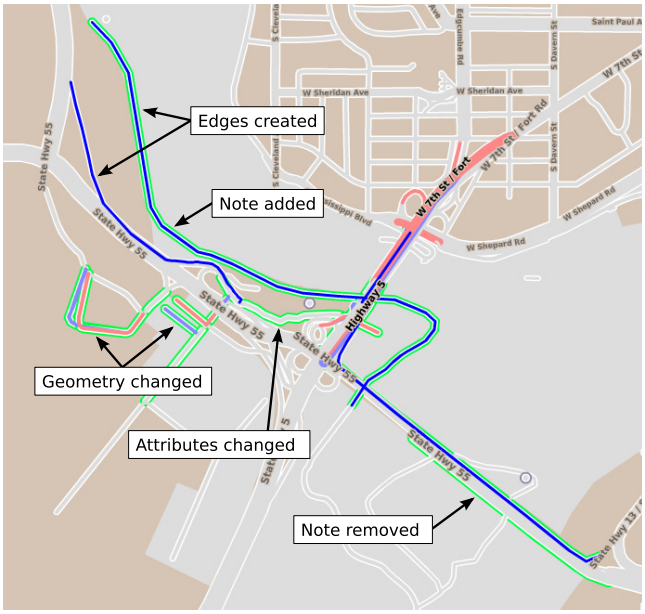


**Figure 5: A particularly rich Cyclopath revision; this one added 7 edges and changed 19, added 5 notes, and removed 4 notes. (Figure from [16].)**

Cyclopath's core data is a graph representing the transportation network used by bicyclists. Each edge in this graph is a separate object, and edits which affect the topology of the graph (e.g., adding a shortcut to the map) necessarily affect more than one edge. Other edits, such as moving the location of a node (street intersection), typically do as well. Figure 4 shows a screenshot of the Cyclopath interface, while Figure 5 illustrates a sample revision.

Another example is a wiki bibliographic database. In this application, it would be useful to store one object for each author and another for each publication, with links between the two as appropriate. Furthermore, these links must be explicit, not implicit. If there are two separate authors named John Smith, their publications should not be comingled; similarly, the same author might be listed as John Smith in some places and J. Smith in others. Also, it would be useful to store an explicit citation graph, which raises the same linking issues as noted above for Wikipedia articles.

In these types of systems, one can't even do meaningful edits unless more than one object can be viewed and edited at once, and failing to save these edits atomically – i.e., as a single revision – risks corrupting the database, either by simply producing nonsensical history (if multiple revisions are simultaneously saved and individual object updates shuffled together) or by producing partially saved revisions (if saving a revision fails partway through).

Similarly, the lack of global atomic revisions damages the reliability of undoing work. With global state, figuring out which edits across disparate objects were made together is trivial; without it, this becomes a difficult and error-prone inference task.

Other geowikis have also struggled with undoing work. For example, the Open Street Map community has implemented several techniques of varying capability and complexity but does not appear to have settled on a unified technique for undoing [15]. The techniques we outline may be useful to this ongoing effort.

The point is: we must build wikis that can handle arbitrarily complex data models and save arbitrarily complex revisions atomically. This requires global state and the ability to edit more than one object at once. In other words, we need to change the meaning of *revision*. We now have a global revision sequence: the unit of revision is now *the whole database*, not individual objects, and at every instant, there is a well-defined global state.

## 4.2 Implementation overview

We have created a wiki with these properties – Cyclopath. Display and editing of multiple objects is done in an Adobe Flex-based application which runs in the browser (this part of the software is not relevant to the topic of this paper, so we will not discuss it further). We introduce two core innovations for implementing global state: (a) a data model to support global state, including simple queries of historical states, and (b) algorithms for updating the database as new data are saved.

Vocabulary to support our discussion is as follows. Because the scope of *revision* now includes the whole database, we use the term *version* to refer to each successive state of an individual object. Similarly, a *versioned table* is a database table which is managed by the wiki state management system (as not all tables within the database are necessarily part of the wiki). There is one versioned table for each type of wiki object; for example, in Cyclopath, there is a versioned table for edges in the transportation network and another for points of interest.

The core of the data model is (a) a single Revision table plus (b) metadata in each versioned table that points back to the Revision table; this data model is illustrated in Figure 6. The important column of the Revision table is simply its ID, which we call *RID*. While we store additional data about each revision, such as a timestamp, who saved the revision, an edit comment, and other items, this is not important to the versioning system.[8]

The versioning metadata consists of five columns:

- *ID*. This is an integer which identifies the object; while not strictly versioning metadata (i.e., it or some other identifier would still be present even without versioning), it interacts closely with the versioning system.[9]

[8]We chose not to base our revision system on timestamps for the following reasons: special attention is often needed for subsecond precision, they can be quirky (e.g., one must worry about time zones, and different systems might have differing clocks), and integer revision numbers have a unique expression that is simple for both machines and humans to compare.

[9]Though it is not required by the state management system, all ver-

**Article**

| *ID* | *Version* | *VSR* | *VBR* | *Deleted?* | *Name* | *Text* |
|---|---|---|---|---|---|---|
| 5555 | 1 | 21 | 23 | no | Mouse | Mice have pointy noses. |
| 6666 | 1 | 22 | 23 | no | Cat | Cats are mammals. |
| 7777 | 1 | 22 | 24 | no | Dog | Dogs smell funny. |
| 5555 | 2 | 23 | $\infty$ | no | Mouse | Mice have pointy noses that wiggle. |
| 6666 | 2 | 23 | 24 | no | Cat | Cats are cute mammals. |
| 6666 | 3 | 24 | $\infty$ | no | Cat | Cats are cute mammals with whiskers. |
| 7777 | 2 | 24 | $\infty$ | yes | — | — |

**Revision**

| *ID* |
|---|
| 21 |
| 22 |
| 23 |
| 24 |

**Figure 6: Editing history of three wiki objects, the articles "Cat", "Dog", and "Mouse". This figure illustrates four revisions: revision 21 created the article "Mouse"; revision 22 created the articles "Cat" and "Dog"; revision 23 altered the text of both "Cat" and "Mouse"; and revision 24 altered the text of "Cat" and deleted the article "Dog". (Note that the set of objects directly affected by a revision $N$ are those objects with VSR $= N$.) In this example, there is only a single versioned table, as the example includes only one type of wiki object (encyclopedia articles). However, the data model can be arbitrarily complex (additional types of wiki objects would each have their own tables), and a given revision can affect arbitrarily many objects in arbitrarily many tables. There is only one Revision table regardless of the data model's complexity.**

- *Version*. While ID identifies the *object*, this column identifies a *particular version of* that object, and thus the primary key for the table is the pair (ID, Version). The first state of an object is stored in a row with version 1, and for each subsequent new state, the version number is incremented.[10]

- *Valid Starting Revision* (VSR). This is the first revision in which the row is the valid version of the object.

- *Valid Before Revision* (VBR). This is the first revision in which the row is *in*valid.[11] The newest (current) version of an object has VBR $= \infty$.

- *Deleted*. A boolean flag: if true, then the object has been deleted and the row contains no valid data. Undeletion can be accomplished by creating another version with Deleted set back to false.

In other words, the state of an object valid at global revision $N$ is the unique row where VSR $\leq N <$ VBR; this row can be retrieved without consulting metadata from any other rows. In particular, the set of objects directly affected by revision $N$ is those objects where VSR $= N$.

The algorithm for creating a new revision is as follows.

1. The user edits as desired. The first time an object is changed during the editing session, mark it dirty and increment its version number. (Newly created objects are assigned a version number of 1.)

2. Obtain the next revision number, $M$, and create a row in the Revision table. RIDs increase monotonically so that they can be used to easily test the order of two revisions.

3. For each dirty object:

   (a) Compare the object's new version number $v_{\text{new}}$ to the most recent version number stored in the database for that object, $v_{\text{stored}}$. If $v_{\text{stored}} \geq v_{\text{new}}$, saving the revision fails because there was an edit conflict; i.e., someone else consumed the next version number – there are no gaps in a object's version number sequence.[12]

   (b) Create a new row for the object and set data columns to the new values, whether that particular attribute was changed or not. Set Version to $v_{\text{new}}$, VSR to $M$, and VBR to $\infty$.

   (c) Set the VBR of the prior version (i.e., version $v_{\text{new}} - 1$) to $M$ (unless the object was created in this revision, in which case there is no prior version).

## 4.3 Global state and undoing

Undoing previous revisions remains important, and the three different techniques for effecting this are modified as follows. Manually editing away undesirable content is essentially unchanged; editors simply use the standard editing interface, which can now view and edit multiple objects at once, to do so. Reverse-merging is also essentially unchanged from the perspective of the state management system (see Figure 7 for an example of reverse-merging under global state), but computing and then reversing a sequence of edit operations may be more difficult, depending on the wiki's domain, and the presence of dependencies between objects may increase the likelihood that the reversed operations cannot be applied to the current state.

However, reverting raises additional complexities. First, which objects should be reverted when revision $N$ is reverted? The obvious choice is "all objects affected by revision $N$" – but what does that mean? Strong dependencies between objects may mean that objects which were not actually changed by revision $N$ might be affected. To illustrate this, we consider an example from Cyclopath. Suppose that revision $N$ splits an edge into two new edges adjacent to a new node; then, revision $N + 1$ creates a third edge adjacent to that node. If only the two edges directly altered by revision $N$ are deleted by an undo, the third edge created in revision $N + 1$ will be left dangling, attached to a now-nonexistent node. (This scenario is plausible in Cyclopath when revisions that add new street intersections are undone.)

sioned tables in Cyclopath share the same ID sequence. This simplifies future schema changes.

[10] In addition to conflict-detection benefits mentioned below, using this natural number sequence which is independent for each object makes it trivial to (a) determine whether a particular version of an object is the first and (b) access subsequent and prior versions.

[11] We use *before* rather than *ending* because VSR and VBR form a half-open interval: a row is valid from the instant revision VBR is saved until the instant *before* revision VBR (which is the next version's VSR) is saved.

[12] After such a failure, a robust implementation would perform appropriate conflict resolution and then retry the revision.

| Article | | | | | | | Revision |
|---|---|---|---|---|---|---|---|
| *ID* | *Version* | *VSR* | *VBR* | *Deleted?* | *Name* | *Text* | *ID* |
| 5555 | 1 | 21 | 23 | no | Mouse | Mice have pointy noses. | 21 |
| 6666 | 1 | 22 | 23 | no | Cat | Cats are mammals. | 22 |
| 7777 | 1 | 22 | 24 | no | Dog | Dogs smell funny. | 23 |
| 5555 | 2 | 23 | **25** | no | Mouse | Mice have pointy noses that wiggle. | 24 |
| 6666 | 2 | 23 | 24 | no | Cat | Cats are cute mammals. | **25** |
| 6666 | 3 | 24 | **25** | no | Cat | Cats are cute mammals with whiskers. | |
| 7777 | 2 | 24 | $\infty$ | yes | — | — | |
| **5555** | **3** | **25** | $\infty$ | **no** | **Mouse** | **Mice have pointy noses.** | |
| **6666** | **4** | **25** | $\infty$ | **no** | **Cat** | **Cats are mammals with whiskers.** | |

**Figure 7: Figure 6 extended to include a new revision 25 which reverse-merges revision 23; *cute* was removed from "Cat" while *that wiggle* was removed from "Mouse". New data are highlighted in bold; note that the VBR of version 3 of "Cat" and version 2 of "Mouse" were updated to 25. Contrast with Figure 8, which illustrates revision 23 being reverted instead.**

| Article | | | | | | | Revision |
|---|---|---|---|---|---|---|---|
| *ID* | *Version* | *VSR* | *VBR* | *Deleted?* | *Name* | *Text* | *ID* |
| 5555 | 1 | 21 | 23 | no | Mouse | Mice have pointy noses. | 21 |
| 6666 | 1 | 22 | 23 | no | Cat | Cats are mammals. | 22 |
| 7777 | 1 | 22 | 24 | no | Dog | Dogs smell funny. | 23 |
| 5555 | 2 | 23 | **25** | no | Mouse | Mice have pointy noses that wiggle. | 24 |
| 6666 | 2 | 23 | 24 | no | Cat | Cats are cute mammals. | **25** |
| 6666 | 3 | 24 | **25** | no | Cat | Cats are cute mammals with whiskers. | |
| 7777 | 2 | 24 | $\infty$ | yes | — | — | |
| **5555** | **3** | **25** | $\infty$ | **no** | **Mouse** | **Mice have pointy noses.** | |
| **6666** | **4** | **25** | $\infty$ | **no** | **Cat** | **Cats are mammals.** | |

**Figure 8: Figure 6 extended to include a new revision 25 which reverts revision 23; "Cat" and "Mouse" were each reverted to their respective version 1s. New data are highlighted in bold. Contrast with Figure 7, which illustrates revision 23 being reverse-merged.**

In general, computing the set of objects affected by revision $N$ requires an arbitrarily complex domain-specific dependency walking procedure. In the above example, deciding what to do about the dangling edge is hard; leaving it dangling means the graph remains inconsistent, while deleting it destroys potentially unrelated work. We argue that in such cases it is better to leverage the human knowledge that drives the wiki in the first place: notify (with his or her prior consent) the user who created revision $N + 1$ and let him or her decide what to do.

Our current approach in Cyclopath is simple. First, we have not yet implemented reverse merging, due mostly to limited personnel resources. Second, our implementation of reverting assumes that the set of objects indirectly affected by a given revision is empty. In addition to the difficulties noted above, we hypothesize that situations which might warrant dependency walking are not actually frequent enough to justify the considerable effort of discovering and building it. Thus, our revert algorithm, illustrated in Figure 8, is:

1. Find the set of objects directly affected by revision $N$ (i.e., VSR $= N$).
2. Find their immediately prior versions (i.e., VBR $= N$).
3. Create new versions containing that prior data (or synthesize a deleted version, for objects created in revision $N$) and save those as a new revision.

In summary, the techniques we outline in this section enable a relatively straightforward implementation of wiki systems that support an arbitrary data model, retaining the core wiki properties such as transparency and reversibility. We next turn to the re-introduction of fine-grained access control.

# 5. WIKI 3.0: ACCESS CONTROL REDUX

The scheme outlined in the previous section is a significant advance over prior approaches, enabling wikis with arbitrary data models. However, we must go further. Its unit of access control and unit of revision are the same, as in traditional wikis – but now that unit is the entire database. This section explores how the unit of access can be made *smaller* than the unit of revision, restoring the fine-grained access control possible in traditional wikis.

In Cyclopath, we have a number of specific use cases motivating this innovation, including:

- Sharing private routes with friends. Users wish to find or build routes and then share them with others to view and/or edit, perhaps a set of Cyclopath users or perhaps a set of people to whom they e-mail a link.
- Private watch regions. Users wish to define map regions in which they will be notified of editing activity; these are often anchored by private locations (e.g., one's home) and users often simply wish to hide their interest from others.
- Private attributes (such as ratings or IDs mapping to other databases) applied to any object the user can view, regardless of edit access.
- Private layers. We are collaborating with local transportation agencies who wish to create layers within the map, sharing them with professional colleagues (who perhaps work in different agencies) but not the public.

This innovation raises a number of interesting challenges. This

section explores these challenges, the solutions we have identified, and how we are implementing them in Cyclopath.

## 5.1 The mutability of access control

These challenges arise when access rights are revoked. That is, if a user never could be removed from an object's ACL once added, then none the complexities we elaborate below come into play. This property could be guaranteed as follows: if a user is to be removed from an object's ACL, instead of changing the ACL, create a copy of the object with the new ACL and delete the old object.

However, recall that wikis retain all historical states of all objects. Thus, this approach is untenable; while the old object would be removed from the current view, it would still be available in historical views under the old ACL. This would make it impossible to recover from certain common mistakes.

For example, if an object containing personal information is made public by mistake, it is ethically essential that this error be rectifiable quickly, without invoking special procedures or asking for sysadmin help. (Something similar already happens in Cyclopath on a fairly regular basis – people enter points of interest called "home", not realizing the entire map is public.) Thus, access to objects, even historical versions, must be revocable; ACLs must be fully mutable and changes managed separately from the wiki state management system. (As a consequence, a historical view for user $U$ of revision $N$ may not be identical to how the wiki actually looked at the time revision $N$ was saved, because $U$'s access may have changed.)

## 5.2 Undoing work

Normal editing is largely unaffected by an access control unit smaller than the revision unit. The user interface only shows objects that can be viewed, and it only enables editing on objects which can be edited. Thus, by definition, revisions created by normal manual editing affect only objects to which the user has edit access.

Rather, the complexities arise when undoing prior revisions, because undo revisions are concerned with a pre-defined set of objects – those in the revision being undone. Some of these objects might not be editable by the user performing the undo, either because the undoing user is a different person with less access or because access to some objects has been revoked. For example, suppose that revision $N$, containing objects $A$ and $B$, is saved by user $U$. User $U$ necessarily has access to both $A$ and $B$, because he or she saved a revision which changed both. Later, User $V$, who has access to object $A$ but not object $B$, wishes to undo revision $N$.

What happens to undo when one doesn't have edit access to every object in a revision? (In properly managed systems, this will be a relatively infrequent occurrence, but as with all corner cases, it must be handled in a reasonable way.) There are two options: don't allow the undo operation at all, or undo only changes to objects that can be edited. The second – *partial undo* – is really the only viable option. For example, if partial undo were not allowed, someone could make a revision that included one private object and many vandalized public objects, and no one could undo it. In any case, without partial undo, users would likely emulate it (to a lesser degree of precision) with manual editing.

In prior iterations of the wiki model, a revision can only be undone entirely or not at all. This is no longer the case. However, the property that *an undo that is undone forms a no-op* remains – provided that the second undo is a complete, not partial, undo (i.e., the user who made the second undo has edit access to every object in the first undo).

The notion of partial undos is conceptually awkward with the notion of atomic revisions. Thus, users should be warned before (a) making access control changes that could create situations which force partial undos and (b) saving a partial undo. In the latter case, the system could offer to request that an administrator or someone with greater access perform the undo.

## 5.3 Implementation overview

The core challenge of implementing access control in Cyclopath is this. First, the Cyclopath data model contains a number of entities which contain other entities (e.g., the transportation graph contains edges, and each edge contains a number of attributes) – a natural structure for making access decisions using recursion. However, SQL databases, such as the one which stores the Cyclopath data, are bad at recursion. Thus, in order to avoid a proliferation of special-case queries, we are implementing a system which can check access on any object by consulting only that object's ACLs and no others.

Our implementation has two key parts. First, each object has four different ACLs specifying (a) who can view the object if given a direct link, (b) who can browse to the object or find it in searches, (c) who can edit the object, and (d) who can change the object's ACLs. These ACLs are implemented with standard optimizations such as user groups, storing each unique ACL only once, etc. (This part is the *basic access control* of Dewan & Shen [5].)

Second, each object can constrain the ACLs of objects it contains, perhaps transitively. (This part is the *meta access control*.) In contrast with the work of Ahn and Sandhu [1], among others, our constraints are between objects rather than user roles. For example, the transportation graph can specify that the edges it contains must be publically viewable and editable, and so must the attributes of those edges. Or, the definition of attribute *rating* can specify that all rating attributes must be private (visible and editable only by the user who created them); also, this constraint can be given a higher priority than the ones specified by the transportation graph layer, enabling the private attribute on edges despite the layer's prohibition of such attributes. Because these constraints are checked only when objects are created or the constraints changed, the need for recursion when checking access is eliminated.

In summary, by accepting the possibility of partial undo and leveraging basic and meta access control, wikis can implement both the global state management needed by wikis with arbitrary data models and the fine-grained access control needed by wikis in the enterprise. We now conclude the paper and briefly explore future work.

# 6. DISCUSSION

Wikis exploded into the public consciousness with Wikipedia. The encyclopedia's embrace of a radical new philosophy, the wiki, and its three core properties – inverting the publishing model, maximally open access, and transparent changes – led to an immense and immensely useful information repository, almost as if by magic. But the promise of the wiki model goes beyond encyclopedias. The model is relevant and useful to any community which makes use of information that is distributed among members of the community, or which members can collectively find and synthesize. *This is many communities indeed.* Why restrict the technology to relatively unstructured information like text articles and to communities which require only simple access control?

There is one key obstacle. Lack of global state management prevented wikis from managing highly interdependent data, restricting their use to contexts where no such dependencies existed or where they could be kludged away. We solved this problem and detail our solution. However, in doing so, we introduced another problem: because the unit of access control was still the same as the unit of revision (as in traditional wikis), and this unit was now the entire database, the traditional implementation of fine-grained access con-

trol in wikis was no longer tenable. We have solved this problem as well; we outline an implementation of access control in wikis with a unit of access smaller than the unit of revision.

Thus, this work advances wikis from a technology supporting encyclopedias and other repositories of relatively independent objects to a technology supporting arbitrary data models in rich access control contexts.

Yet there is more to be done. What will Wiki 4.0 look like? We speculate that future wiki systems will continue to adapt innovations which emerged first in version control systems, perhaps even by building wikis on top of established VCSes.

For example, there is a need for *branching* – the ability to persistently save in-progress work, with access to history, undo, and all other wiki features, without affecting the main wiki. Use cases include private collaboration on changes before merging them into the main wiki, disruptive experimental edits inappropriate for the main wiki, and all the other reasons branching is found in VCSes. In fact, branching already happens in systems like Wikipedia. Users copy articles, do work and get feedback on the copy, and then merge the changes back to the main article. This is indeed branching, but done manually. In particular, merging continuing work on the main article into the copy is tedious and error-prone.

Another potential advance paralleling the development of VCSes is *distributed wikis*. Users might not trust a single server or might need to comply with organizational requirements on keeping certain information behind the firewall. Distributed wikis could retain a unified view of information split between different servers.

Finally, we close with the observation that this paper describes techniques for making wikis both *less simple* and *less open*. We believe this is a positive step. First, we have identified new domains where introducing the wiki model requires changes that make it more complex. Like any software system, a wiki should be as simple as possible – yet no simpler. Thus, rather than forgoing these opportunities, we argue that wikis should instead thoughtfully add the necessary complexity.

Similarly, we have identified new domains that simply require tighter access control than is consistent with the traditional wiki philosophy. Some of these domains might ultimately be flexible (for example, wiki software capable of matching a relatively closed organizational culture could later be adjusted towards greater openness, while wiki software that requires openness in order to function would never be adopted in the first place), while some closed data is uncontroversial (for example, personal data about individuals). Regardless, it is difficult to enable judicious use of access control without also enabling excessive or backward use. In both cases, greater flexibility in simplicity and openness creates possibilities for applying the wiki model that would not otherwise be available.

## 7. ACKNOWLEDGEMENTS

We thank Michael Ludwig and Landon Bouma for their design and implementation work on these issues. Andrea Forte provided thoughtful feedback on [17], which developed into this paper's introduction. Anonymous reviewers provided important feedback. This work is supported in part by NSF (IIS 05-34692 and IIS 08-08692).

## 8. REFERENCES

[1] Gail-Joon Ahn and Ravi Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3:207–226, November 2000.

[2] Yochai Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, 2006.

[3] Dan Cosley et al. Using intelligent task routing and contribution review to help communities build artifacts of lasting value. In *Proc. CHI*, 2006.

[4] Catalina Danis and David Singer. A wiki instance in the enterprise: Opportunities, concerns and reality. In *Proc. CSCW*, pages 495–504, 2008.

[5] Prasun Dewan and HongHai Shen. Flexible meta access-control for collaborative applications. In *Proc. CSCW*, pages 247–256, 1998.

[6] Sarah Elwood. Geographic information science: New geovisualization technologies emerging questions and linkages with GIScience research. *Progress in Human Geography*, 2008.

[7] Martin Fowler. VersionControlTools, February 2010. http://martinfowler.com/bliki/VersionControlTools.html.

[8] Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438(7070):900–901, December 2005.

[9] Michael Goodchild. Citizens as sensors: The world of volunteered geography. *GeoJournal*, 69:211–221, 2007.

[10] Jonathan Grudin and Erika Shehan Poole. Wikis at work: Success factors and challenges for sustainability of enterprise wikis. In *Proc. WikiSym*, 2010.

[11] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15:287–317, December 1983.

[12] Chor Pang Lo and Albert K.W. Yeung. *Concepts and Techniques of Geographic Information Systems*. Prentice Hall, 2nd edition, 2006.

[13] Sarah Manley. 100,000 wikis on Wikia, April 2010. http://community.wikia.com/wiki/User_blog:Sarah_Manley/100,000_wikis_on_Wikia.

[14] Sean A. Munson. Motivating and enabling organizational memory with a workgroup wiki. In *Proc. WikiSym*, 2008.

[15] OpenStreetMap. Change rollback, August 2010. http://wiki.openstreetmap.org/w/index.php?title=Change_rollback&oldid=514591.

[16] Reid Priedhorsky. *The Value of Geographic Wikis*. PhD thesis, University of Minnesota, August 2010. http://reidster.net/pubs/thesis.pdf.

[17] Reid Priedhorsky. Wiki, absurd yet successful. In *CHI 2011 Workshop on Crowdsourcing and Human Computation*, 2011.

[18] Reid Priedhorsky et al. How a personalized geowiki can help bicyclists share information more effectively. In *Proc. WikiSym*, 2007.

[19] Reid Priedhorsky and Loren Terveen. The computational geowiki: What, why, and how. In *Proc. CSCW*, 2008.

[20] Eric Raymond. Understanding version-control systems, January 2008. http://www.catb.org/~esr/writings/version-control/version-control.html.

[21] Patrick Thomson. Git vs. mercurial: Please relax, August 2008. http://importantshock.wordpress.com/2008/08/07/git-vs-mercurial/.

[22] William Tolone et al. Access control in collaborative systems. *ACM Computing Surveys*, 37:29–41, March 2005.

[23] Wikimedia Foundation. Wikipedia, January 2011. http://wikipedia.org/.

[24] Wikipedia. Protection policy, March 2011. http://en.wikipedia.org/w/index.php?title=Wikipedia:Protection_policy&oldid=417774228.