skWiki: A Multimedia Sketching System for Collaborative Creativity

Zhenpeng Zhao,[1] Sriram Karthik Badam,[1] Senthil Chandrasegaran,[2] Deok Gun Park,[1]
Niklas Elmqvist,[1] Lorraine Kisselburgh,[3] and Karthik Ramani[1,2]
[1]School of Electrical & Computer Engineering, [2]School of Mechanical Engineering, and [3]Brian Lamb School of Communication
Purdue University, West Lafayette, IN 47907, USA
{zhaoz, sbadam, senthil, park573, elm, lorraine, ramani}@purdue.edu

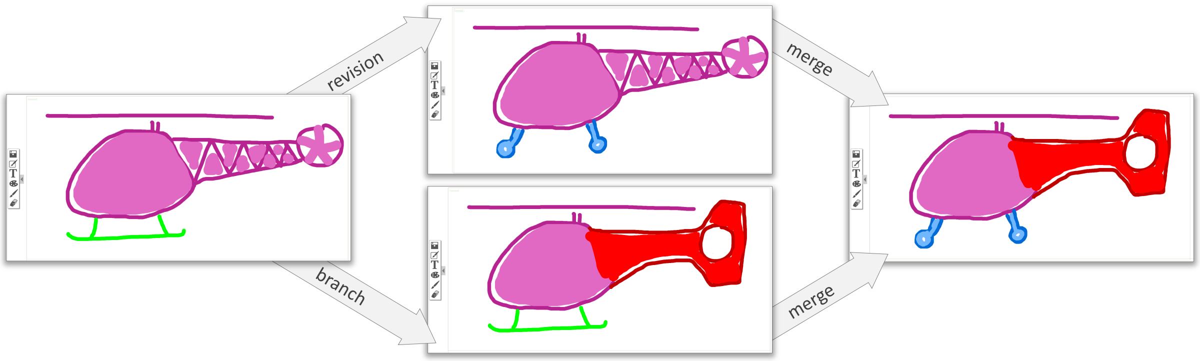

Figure 1. Two parallel paths of concept sketches of a toy helicopter in skWiki. skWiki supports multiple co-existing revision histories for the same multimedia object. This is possible using *paths*, which track state changes over time and users so that the complete operation history is preserved.

ABSTRACT

We present skWiki, a web application framework for collaborative creativity in digital multimedia projects, including text, hand-drawn sketches, and photographs. skWiki overcomes common drawbacks of existing wiki software by providing a rich viewer/editor architecture for all media types that is integrated into the web browser itself, thus avoiding dependence on client-side editors. Instead of files, skWiki uses the concept of paths as trajectories of persistent state over time. This model has intrinsic support for collaborative editing, including cloning, branching, and merging paths edited by multiple contributors. We demonstrate skWiki's utility using a qualitative, sketching-based user study.

ACM Classification Keywords

H.5.3 Information Interfaces and Presentation: Group and Organization Interfaces—*Web-based interaction*

Author Keywords

Collaborative editing; creativity; wikis; sketching.

INTRODUCTION

While more of a design philosophy than a technology in its own right, the Web 2.0 revolution has turned the traditional content creation model on its head. Instead of a few highly institutionalized creators distributing content to a mass audience, these new technologies are empowering users themselves to become creators, distributors, and marketers of digital content in vast collaborative enterprises on the Internet.

So far, computer support for such large-scale distributed collaboration has mostly centered on *convergent processes*: processes where the outcome is a single artifact, concept, or idea; where the role of the computer is to mediate communication, arbitrate conflicts, and consolidate information. However, intellectual pursuits are often a combination of *convergent* (or analytical) processes and *divergent* (or creative) processes [22]. For example, during early design for a new line of toys, the goal is to generate as many and as diverse ideas as possible rather than to prematurely fixate on a particular one. Most computer-supported collaboration tools, be they for socially constructed knowledge bases such as Wikipedia, collaborative text editors such as Google Docs, or product lifecycle management (PLM) systems such as Enovia, are designed for analytical and convergent processes, but have little provisions for divergent ones.

Inspired by source code management (SCM) systems [24] such as Git or SVN, where *branching*—creating a divergent version of a project, directory, or a single file—is a natural operation in any project, we adopt the concept of *paths* as persistent state over time owned by a particular user. A document or file represented as a path is thus stored as the ordered (and timestamped) sequence of document-specific operations that created and modified it. For example, a digital photograph is the sum of all its changes, such as importing pixels from a digital camera, adding a Gaussian blur filter, and finally changing the image data to grayscale. Furthermore, because paths also track ownership, the concept intrinsically supports divergent collaborative editing: modifying someone else's entity simply branches its path and creates a new list of changes since the original state. Convergent processes are also supported: merging two paths means accepting all or some of the changes into the merged path.

To validate the path concept, we present SKWIKI (pronounced Squeaky), a web-based collaborative content editing framework that implements the path model for divergent creative digital media processes. skWiki supports multiple multimedia path types beyond the marked-up text traditionally supported by Wiki software. Each such multimedia type, implemented as a plugin in the skWiki framework, comes with separate viewers and editors. *Viewers* are used to render the content that users see in their web browser when visiting a skWiki page. *Editors*, on the other hand, provide basic editing operations for each entity type inside the browser itself. This eliminates the dependency on offline client-side media editors—e.g., Photoshop, Visio, and CorelDRAW—and provides a unified editor interface inside the user's own web browser. Our skWiki prototype currently supports rich text, free-hand sketches, and photographs.

Beyond these viewers and editors, one of the main components of skWiki is the *path viewer*: a visual management interface for viewing and navigating in the evolving graph structure formed by the media paths in the system. The path viewer visually represents how these paths are created, modified, deleted (although deletion is not persistent in the path model), branched, and merged. Most of these path operations are transparent from the user's point of view. For example, the path viewer allows a user to view and edit a path—their own or someone else's—at any time, thereby creating a new branch from that point. Similarly, changes to a media entity are automatically saved and committed to the skWiki system. Because of this automatic support for both collaboration and revision control, we posit that skWiki can be adopted by virtually any audience involved in digital content creation, including both product, industrial or web designers as well as engineers, researchers, and creative artists.

RELATED WORK

In today's highly networked and increasingly complex world, most intellectual endeavors are collaborative in nature [2]. Many digital media projects today are built by a wide range of contributors distributed across the Internet. Wikipedia is a prime example, with more than 24 million articles in 285 different languages that is maintained by more than 100,000 active contributors working together. This form of collaborative editing of digital media is now gaining acceptance in a broad set of domains. In this section, we describe the prior art in wikis, creative design, version control, and collaborative editing.

Wikis for Collaboration

Ward Cunningham developed the first Wiki in 1995, calling it "the simplest online database that could possibly work" [17]. Wikis quickly caught on as a means for collaboratively creating, vetting, and maintaining online documents, its application made famous by Wikipedia. Wagner [34], emphasizes its versatility in providing a repository and a means for many-to-many collaboration by incorporating aspects of various communication tools such as email, chat, and multimedia applications, with a temporal database for history support. Traditional wiki systems such as MediaWiki, TWiki, and Confluence, enable such collaborative editing via a web browser, and support rollback, external links, and heterogeneous content such as code, documents, images, and rich text content. The use of wikis for content creation and not just content storage was extended to software development as well [17], especially for project management, where project documentation and discussion can be placed in context with project content. The use of traditional wikis is further extended by semantic wikis that capture and identify metadata in wikis using RDF and OWL frameworks, such as IkeWiki [28], SeMedia Wiki [3], and PlatypusWiki [32].

Recent work on wikis are starting to extend the traditional wiki history model for text. Sabel's work [26] on version history of a wiki page considers an adoption coefficient that is defined by the structural similarity between two versions of a text document, and uses it to arrange versions of a wiki page into a weighted tree. Similarly, Priedhorsky and Terveen [21] discuss implementation challenges and solutions for maintaining a single global state and history for non-textual objects using the Cyclopath geowiki as an example. They propose solutions for some of the challenges inherent with maintaining multiple object types, such as a ordering revisions, global state and undo, and access control.

Collaboration in Design

Design is often defined as an ill-defined problem whose solution is obtained through creative exploration alternated with pruning of the design space [22]. This alternating of the creative or *divergent processes* and the analytical or *convergent processes* has been studied extensively, resulting in now-commonplace creative methods such as brainstorming [20], brainwriting [25], and collaborative sketching [29]. While technology has long been used in an attempt to scaffold creativity, part of the challenge in this endeavor lies in the nature of creativity itself. Torrance [33, p. 47] describes creativity as "the process of sensing difficulties, problems, gaps in information, missing elements, something askew; making guesses and formulating hypotheses about these deficiencies; evaluating and testing these guesses and hypotheses; possibly revising and retesting them; and finally communicating the results." This continuous inspiration, guesswork, and evaluation results in a stream of ideas, only some of which are developed further.

Early work in collaborative creative support for design was in the form of multi-user drawing support [14, 19, 31] for collaboration between geographically distributed participants. The i-LAND environment [30] was pioneering in its use of (now) ubiquitous technology such as display walls and tabletops along with conventional computers in support of creativity. Recent work in this area includes IdeaVis [7], TEAM STORM [10], and GAMBIT [27].

Greene [8] lists support for exploration and experimentation, collaboration, iteration, and domain-specific action as essential for applications intended to support creativity. Hilliges et al. [13] also recommend an interface that allows actions to be carried out both individually and collaboratively.

Version Control and Collaborative Editing

Version control systems manage content change and maintain a history of its evolution, and are commonplace in the software industry. The Source Code Control System (SCCS) was one of the earliest source code management systems developed in the early 70s to store, update, and retrieve all versions of code [24]. Currently popular SCM systems include CVS, Subversion, and Git. More recent developments record and visualize developer activity at interaction level to enhance program history representations [23, 35].

While these version control systems were primarily created for software engineering, most can be used for any text file, and, to some extent, binary files as well. Furthermore, while not strictly classified as version control, systems such as Chronicle [9]—which clusters, probes, and visualizes a document's workflow history—and MeshFlow [6]—which visualizes, clusters, annotates, and filters the history of operations on polygonal meshes—are significant contributions towards managing history on binary files.

Collaborative editing is essentially a version control system integrated with the editor itself, and has a rich history in the CSCW field [1]. Social code-hosting repositories such as GitHub have been studied to gain insight into the influences of activity volume, commit histories, community interest, and personal interest of contributors [5]. A modern collaborative editor such as Google Docs allows multiple users to work on a document by propagating all edits in real time, thus alleviating the explicit need to commit/save versions of the document to a server. This means that the only way to "branch out" (diverge) at a given time is to make an explicit copy of the document. Google Docs maintains a history of edits to the document ordered by timestamps.

Recovering editing operations for binary data is more difficult than for text, posing another challenge for collaborative editing of digital media. To address this, Chen et al. [4] propose a version control system for image editing that integrates with the image manipulation program itself to capture the drawing commands that transformed the image. This work is perhaps the most related to skWiki, but differs in several important ways: (1) skWiki is a collaborative system designed for more than a single user; (2) we support revision control of multiple media types beyond photographs, including text and sketches; and (3) our system is entirely web-based and requires no dedicated client-side software.

SUPPORTING DIGITAL CREATIVITY

Our goal with the skWiki project is to support digital creativity in collaborative, potentially distributed, teams. Here we explore the design space for this research topic.

Requirements

Based on the literature and our collaborations with professional designers, we formulate the following requirements for a software framework to support digital creativity:

R1 **Mobility:** A basic goal is that the platform should be designed for mobile devices so that the designer can bring skWiki to any design session "anywhere, anytime;"

R2 **Collaboration:** Virtually all realistic design endeavors involve more than one participant working together in teams [2], often distributed in time and/or space [1];

R3 **Revision history:** Effective collaborative content creation requires support for branching, editing, and merging different versions of the media entities being manipulated;

R4 **Transparency:** Complexities of collaboration and revision histories should be hidden from non-expert users;

R5 **Rich media:** Even simple digital media projects today often incorporate a wide range of media types, such as rich text, photographs, audio, video, and illustrations; and

R6 **Divergent/creative work:** We want to support creative work where the focus is on creating multiple, separate, and diverse content, often drawing on other collaborators.

We found no existing software framework that fully supported all of these requirements. For this reason, we decided to create our own software framework for collaborative creativity called SKWIKI, which is described in the next section. In designing skWiki, we also felt that there was need for a new theoretical model that would capture these requirements on a storage level. Drawing from work on semantic-level revision control of images by Chen et al. [4], we propose the idea of *paths* for digital media. This is reviewed next.

The Paths Model

We define a *path* as a tuple $\langle I, O, R, L\rangle$ consisting of a unique entity identifier I, an owner O, a reference to a parent revision R (possibly empty), and an ordered and timestamped list L of *state transformation operations* that, when taken together, recreates the current state of the entity. One way to think of a path is thus as persistent state over time, i.e., all of the different versions of a particular file from the time it was created. However, this view is somewhat misleading since paths are *not* snapshots, or even deltas, at different times, but rather the actual operations that yielded those states.

Abstractly speaking, a *state transformation operation* is a function $f_T : \mathbb{S} \to \mathbb{S}$ for revision number T (a serial number starting from 0) that takes an entity state $s_{T-1} \in \mathbb{S}$ from the previous revision number and produces a transformed state $s_T \in \mathbb{S}$. A revision R for a particular path is thus a pair $\langle I, T\rangle$ comprising a path identifier and the revision number referred to in its operation list. Recreating the state for any revision number T thus becomes a function composition series

$$s_T = f_T \circ f_{T-1} \circ f_{T-2} \circ \ldots \circ f_0(\emptyset).$$

The benefit of representing paths in this way is that it avoids storing entire snapshots of the entity at every time step. In fact, in many cases, using a sequence of operations will also result in a more economic representation even than storing binary deltas of an entity's state. For example, whereas a file diff would have to replicate the entire image data if a photograph is inverted, a path just stores the invert operation.

Similar to files in a traditional file system, there are several fundamental operations defined for paths:

Create: A new path is created by allocating a unique entity identifier, assigning the owner reference, and initializing the operations list to $\emptyset$. Paths are tied to owners; the owner reference is an immutable part of the path tuple.

Render: Rendering an entity is similar to recreating its entire state up to a certain revision number T (it is also possible to map a specified time to a revision number using the time stamp associated with each operation). This is performed using the function composition above. Rendering a branched path (i.e., one with a non-null parent revision R, see below) will first recursively render the parent revision R before rendering the child path.

Modify: Changing a path is equivalent to adding one or more transformation operations to the operations list in the path. All operations are timestamped and have a revision number, which is a monotonically increasing serial number (starting from 0). Note that only the owner of a path can modify it; others would first have to branch it.

Delete: Paths cannot be deleted. However, a null operation ($f(s) = \emptyset$) can be added to the end of the operations list, which effectively renders the path's state empty. Prior versions of a deleted path can always be rendered up to the penultimate revision number to recover deleted state.

Branch: Paths cannot be copied, but can be branched. Branching a path creates a new path with a unique identifier, appropriate owner reference, and a parent revision R that specifies the original path and revision number. The operations list L for the branched path will initially be empty, but rendering it will recursively render its parents.

Merge: A path p_A can be merged with another path p_B by selectively appending one or several operations from p_A into p_B. Both paths are preserved by this operation, and the net effect is simply that p_B is modified. For a merge operation to make sense, however, the paths should be related, e.g., one of them being an ancestor of the other.

Note that operations can never be removed from the operation list in a path; a change can only be undoed either by (a) branching from a previous state, or by (b) appending the inverse operation to the end of the operations list. This is because branching for paths relies on parent revisions to never disappear, which would otherwise cause inconsistencies.

Implementing Paths

Actually realizing the theoretical paths model above requires overcoming several implementation challenges. Representation is perhaps central amongst them: the optimal way to implement paths using current technology is to use a database management system (DBMS) that stores individual paths in one table, and all of the operations in another using the path identifier as a primary key and including the revision serial number. Users should be represented in another table that can be used to track path ownership and access permissions. Below we review additional challenges in implementing paths and our recommendations for how to tackle them.

Transformation Operations

Representing the actual transformation operations is a major challenge. Because operations are specific to different types of data, this consideration is dependent on the document formats that the paths implementation supports. The best solution may be to store source code in a *domain-specific language* (DSL) [18] for each document format. A particular paths implementation may have to support several DSLs, one for each document format supported. Also, care must be taken to not mix incompatible DSL operations in one path.

The database representation for an operation thus becomes a table consisting of a path reference, a revision number (a serial number unique to each path), a time stamp, and an operation string. For example, an operation string for a bitmap image DSL might have the form `crop(100, 100, 1024, 768)` to crop an image to 1024×768 dimensions starting at position (100, 100), whereas one for text might be `insert("d", 10)` to insert the character "d" in position 10 in the current state. If the database inputs are cleaned appropriately, it might even be possible to execute these operation strings using direct evaluation (i.e., with the `eval()` function for a DSL built in JavaScript).

Operation Chunking

To minimize the number of operations for each path, a practical solution might be adopt the *chunking* approach used in graphical histories [12, 16] to group together related operations. For example, a sequence of character insertions that together spell the word "design" might be more economically represented as a single insertion of the whole word. Similarly, a list of movements of a graphical object could be replaced by a single translation for the resulting vector.

The disadvantage of operation chunking is that the time stamp for each chunked event is replaced by a single one, making it impossible to branch from the constituent events. However, such fine branch granularity is generally not necessary for practical path implementations.

Path Caching

The paths model takes a somewhat extreme view of state as the sum of all operations performed on an entity. For this reason, rendering a path may sometimes be a lengthy operation, particularly if the path has undergone many revisions or if individual operations are time-consuming. For example, certain image filters (edge detection, motion blur, image distortions) may take up several seconds to complete. In such situations, it is not practical to render a path from scratch. Instead, we use *path caching* to speed up the process.

Formally speaking, a *path cache* is the complete rendered state s_T for a path up to revision number T. Rendering any paths at revision numbers $T+N$ now simply reduces to performing the abbreviated function composition

$$s_{T+N} = f_{T+N} \circ f_{T+N-1} \circ f_{T+N-2} \circ \ldots \circ f_{T+1}(s_T).$$

In practice, this means that when rendering a revision number T, the render operation first fetches the cached path with the most recent revision number $T' \leq T$. These cached versions should be saved under unique file names in an internal directory and tracked using a cache table in the database. Caching itself should be transparently performed, for example whenever rendering a path, or when encountering a particularly time-consuming operation (or sequence of operations). By the same token, a practical implementation should probably incorporate a path cleaning mechanism that periodically removes cached versions (from both cache table and directory) whenever they have not been accessed recently.

Transparent Path Operations

The paths model may appear complex, but most of its complexity can be hidden from the point of view of the end user. In fact, to achieve most benefit from the concept, a practical implementation should most likely make all of the path operations transparent to the user. For example, a paths implementation needs no explicit "save" functionality, but will instead automatically commit all modifications (using the modify command). Similarly, rendering (particularly in the presence of any path caching), deleting, and branching paths should also not expose the above details to the end user.

Perhaps the only path operation that cannot be entirely transparent is the merge, which requires that the user explicitly selects the operations to merge from a source path into a destination path. At the same time, it might be possible to encapsulate this operation in a form of copy-and-paste that most users are already familiar with, or to use a smart merge.

Path Navigation

In a normal file system, a file explorer is sufficient to navigate and manage files and directories. In a paths implementation, however, it is not only necessary to be able to navigate the path structure (which may or may not incorporate the traditional hierarchical structure of classic file systems), but also to navigate the revisions of each path. In other words, a practical paths implementation needs a *path explorer*.

Several considerations factor into creating an effective path explorer. One visual representation of a path is to display the list of operations, suitably chunked into semantic units. However, this may result in a representation that is unfamiliar to users who are accustomed to traditional file systems. An alternative representation may use rendered snapshots (as thumbnail images or summaries, for example) of the path at various time intervals. Such a representation would even be amenable to semantic zooming, where zooming in would reveal a different visual representation with more detail, and zooming out would afford a broad overview of the path's evolution. Furthermore, it might be advantageous to use this visual snapshot representation to highlight changes from one snapshot to the next using a suitable visualization.

Several additional operations are needed in a practical path explorer. For example, the visualization should visualize branching to show how different paths build on other paths, as well as different users and their individual paths. The interface should also contain mechanisms for searching for paths by name, time, or owner, as well as bookmarking, filtering, and tagging paths. Such an interface would not only aid creative collaboration, but would also provide a framework for determining contributions, ownership, and influences of ideas, especially in early stages of design.

THE SKWIKI SYSTEM

Based on our exploration of the design space of digital creativity support, we developed a web-based collaborative editing framework for multimedia documents called SKWIKI. skWiki is based on a web service architecture (Figure 2) with server-side components to manage persistence using a practical paths implementation, and any number of users participating in the collaboration using only a web browser. The framework supports digital media projects consisting of multiple media types, such as text, hand-drawn sketches, vectorized illustrations, and digital photographs. skWiki clients can run on standard computers, mobile phones, and tablets with popular operating systems such as Android, iOS, and WebOS. The framework is built to allow extensions with additional media types as plugins.

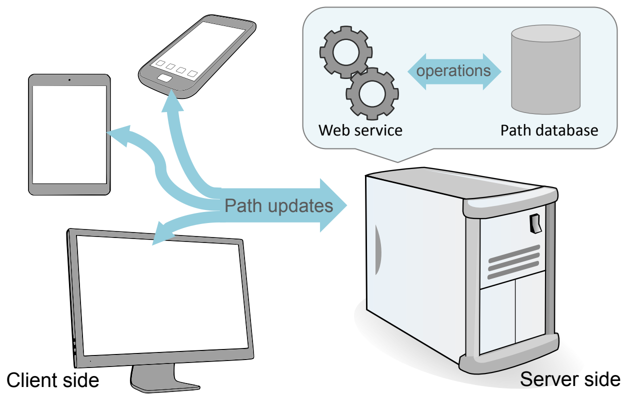

Figure 2. skWiki system architecture. The client maintains workspace and local paths storage, whereas the server is the main paths repository.

skWiki was designed primarily to support *divergent creative processes*, i.e., early design, brainstorming, conceptual art, ideation, and design alternatives. We will now explain the implementation of the paths model in skWiki, and how its practical features support collaborative creativity.

- **Transformation Operations:** Because skWiki is a controlled deployment of the paths model (as opposed to, say, implementing it for an entire operating system), we bound the operation set to the domain-specific languages used to create and modify the state. Our implementation currently supports four DSLs: bitmaps (for digital photographs), rich text (in HTML format), drawings (including freehand sketching and vector drawing), and layouts (hierarchical and spatial arrangements of entities on a page).

  Note that some operations in the above table include object identifiers as arguments to name newly created objects. Object identifiers are used by other operations, such as `removeShape()`, to refer to specific objects, and they must be explicitly named so that the operations list is a complete representation of the state of a path.

- **Operation Chunking:** Our current implementation performs only a minimum of operation chunking; for example, sequences of character insertions or deletions are chunked into strings, and long series of lines captured from a user sketching are chunked into polylines. However, similar to caching, chunking is a largely independent mechanism that can be progressively improved to be more aggressive without affecting overall skWiki functionality.

- **Path Caching:** The current implementation of skWiki uses no explicit path caching. We found that none of our DSL operations were particularly time-consuming to perform, and thus render each path completely from their creation. Furthermore, since browser-based web applications have very limited support for local storage, we wanted to avoid large network transfers of cached state. However, a practical skWiki implementation in the future should certainly provide an appropriate level of caching.
- **Transparent Operations:** All path operations in the skWiki implementation are transparent from the viewpoint of the user, including branching, undoing, and deleting paths. In our informal evaluation, we found that normal users still prefer access to standard operations such as "save" and did not fully understand the new conceptual model underlying skWiki. For this purpose, we provided an "add bookmark" button to replace the traditional "save." This is a transitional remedy until the paths concept becomes more familiar to our end users.
- **Path Navigation:** We implemented a traditional graph viewer, similar to those used in source code management systems, to explore and manage paths. We currently use bookmarked revisions to guide which important states to visualize in the path viewer. This also supports the users' mental model that their bookmarked revisions are the main units of the history. Of course, the path implementation allows the user to also drill into any revision between bookmarked ones whenever necessary.
- **Viewers and Editors:** A *Viewer* is an interpreter for a domain-specific language that is capable of decoding a sequence of transformation operations implemented in a DSL and recreating the corresponding digital content. Similarly, an *Editor* is a command generator that can generate new DSL operations in response to user interaction. Viewers and Editors in skWiki are thus plugins tied to a specific media type. Both Viewers and Editors are part of the same user interface. Viewers get allocated a content space for its associated path. Content spaces may also have an associated Editor that will partner with the Viewer to allow path modifications. The Editor will also provide a DSL-specific toolbar (Figure 3) to support editing.

Interactions

The skWiki client is a JavaScript web application that runs entirely in the user's own browser, regardless of operating system, hardware platform, and with no special software dependencies. Figure 3 shows an annotated screenshot of the main skWiki interface, which includes the editing toolbar and the path explorer. In addition to the tree-based path explorer, we also provide a gallery-based path explorer where entities are represented by thumbnails of their latest revision, but can be traversed in time using the mouse wheel.

Implementation Notes

The skWiki front-end web application is implemented in Java using the Google Web Toolkit (GWT). GWT compiles Java code into JavaScript, which runs in virtually any web browser as a rich internet application (RIA). This enables building web applications without requiring expertise in browser quirks, JavaScript, and AJAX requests.

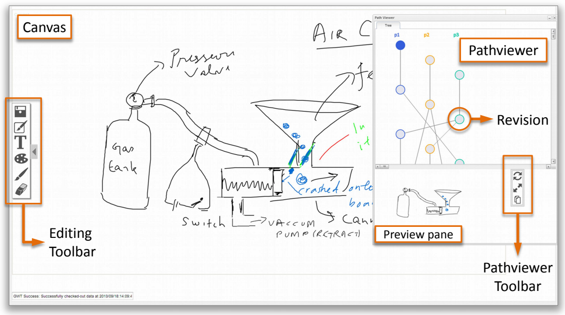

Figure 3. The skWiki browser-based user interface showing the editing toolbar and the path viewer for a sketch entity.

The skWiki server is implemented as a Java Servlet hosted in an Apache Tomcat servlet container. The server also runs a PostgreSQL database for storing all paths and DSL information. The client talks to the server using GWT-RPC (Remote Procedure Call). When the skWiki servlet receives a request, it fetches information from the database and performs the required operations. It then sends a reply to the skWiki client, which updates its own state in response. This architecture makes client-side editing into a real-time and synchronous process, whereas network-intensive or computationally expensive operations such as searching, rendering, and sorting remain asynchronously performed on the server.

USER STUDY

Our goal with skWiki is to support digital creativity in collaborative teams, both co-located as well as distributed. To evaluate our prototype, we performed a study of co-located teams on a creative task. Since free-form creativity not only takes training but is also difficult to quantify, we used a form of controlled brainstorming called C-sketch [29]. C-sketch is a sketch-oriented adaptation of the more generic method of 6-3-5 brainwriting [25], contextualized in a design environment. In the C-sketch method, designers first spend 6-10 minutes sketching out an idea, and then pass it on to the next person. Each designer then spends the next 6-10 minutes working on editing or extending the design idea given by their teammate. This process continues for 2-3 iterations, at the end of which all designs sketched initially have undergone iterative development by at least 3 people. The C-sketch workflow is shown in the left part of Figure 4.

Experimental Conditions

One of the main drawbacks of methods such as C-sketch is that some promising ideas can be lost in the series of iterative edits. Additionally, at the end of the session, there are only as many ideas as there are designers, with intermediate—and potentially promising—ideas being "lost". We hypothesized that the history support afforded by skWiki, along with the option to branch out and create multiple versions of the same sketch, will provide designers with more potential sources of inspiration and development for the design problem at hand.

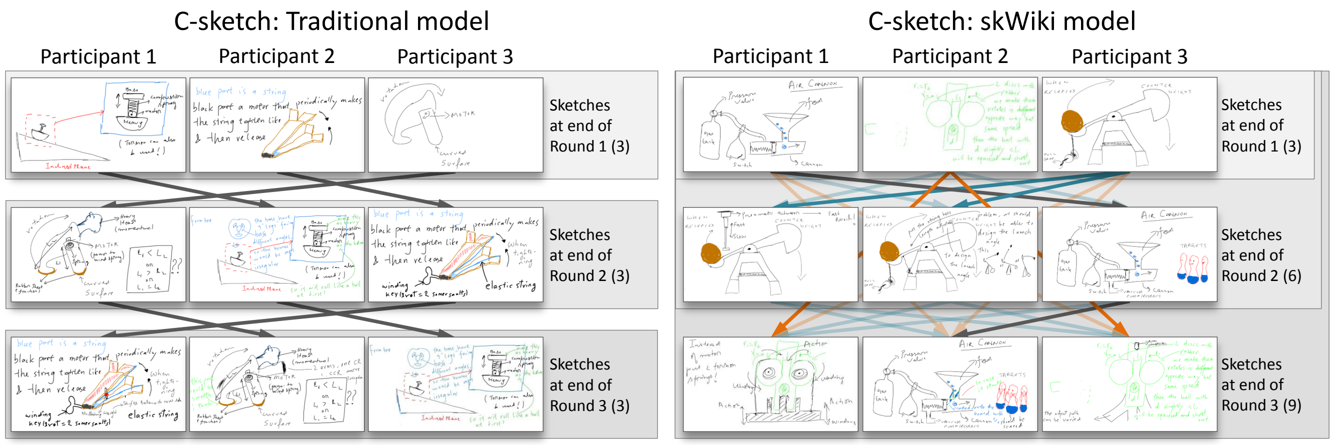
C-sketch: Traditional model
Participant 1
Participant 2
Participant 3
Sketches at end of Round 1 (3)
Sketches at end of Round 2 (3)
Sketches at end of Round 3 (3)
C-sketch: skWiki model
Participant 1
Participant 2
Participant 3
Sketches at end of Round 1 (3)
Sketches at end of Round 2 (6)
Sketches at end of Round 3 (9)

Figure 4. Schematic diagram of the two processes used in the user study: traditional (left) versus the skWiki version of C-sketch (right). Each column shows sketches made or modified by each participant (team of three). The grey arrows show the paths of the traditional C-sketch method, and colored arrows show a departure from the model enabled by skWiki. Potential paths are shown as translucent arrows, while actual paths taken are shown with solid arrows. The traditional C-sketch model emulates the pen-paper paradigm, with no duplication, and no saved states for the sketches. Thus, there are only three concepts available at the end of this session, with all intermediate concepts lost. The skWiki model allows duplication and multiple copies at each round (shown as aquamarine arrows), as well as for branching from earlier states (shown as orange arrows). The team thus has more choices at the end of each round, and nine concepts at the end of the session.

In order to study the effects of the affordances offered by skWiki, we involved each team in two different conditions:

- **Traditional:** The team used a traditional three-round C-sketch workflow as described above, with only sketch movement (no copying) between participants, and with only the latest sketch version available for each round.
- **Full skWiki:** Here the team used the same three-round method as above, with two main differences: (1) at the beginning of the second and third rounds, they could choose any sketch to work on except the sketch they had created in the previous round, and (2) when selecting a final design (end of round 3), they could choose sketches from rounds earlier than the immediately preceding one.

While the C-sketch method typically involves sketching on paper. our intent was not to compare paper vs. digital media. We thus used skWiki for both versions: a version without branching and history "rollback" for the C-sketch condition, and a version with both these features for the full skWiki condition. The aquamarine and orange arrows in Figure 4 show the branching and rollback operations respectively.

Participants

We recruited 4 teams of 3 paid participants each (11 mechanical engineering graduate students and 1 post-doctoral researcher, all male). Participants were aged between 21 and 33 years. 10 participants were comfortable with sketching, and 5 considered themselves proficient. 4 participants had prior knowledge of the C-sketch method. Participants were randomly assigned to teams based on available time slots.

Apparatus

All participants used Microsoft Surface Pro tablets, equipped with the Surface Pen for sketching and annotations. Both conditions used different versions of the skWiki interface, running on the Google Chrome browser. For the traditional C-sketch condition, the skWiki interface was provided with curbed features based on the C-sketch method, such as adopting a sketch and editing it, with no copy or history support. The skWiki C-sketch condition also required a level of feature curbing: history support was minimized to what was available at the end of each round, but not to a stage between them. These constraints helped control the experiment conditions, in addition to allowing the participants to concentrate on the method rather than spend their time on learning and remembering commands.

Tasks

The teams were assigned two tasks, one for each condition: (1) design a toy catapult with an innovative launching mechanism, and (2) design a new kind of somersaulting toy. The order of conditions and tasks were varied among the team to balance out learning effects as well as testing bias.

Each task was split into three rounds of 6 minutes each. In the first round, participants were asked to sketch one idea each for the toy, and annotate it so that their team could understand the idea without additional explanation. No verbal communication between team members was allowed during the three sketch rounds. In the second and third rounds, each participant was asked to develop or edit the sketch of another participant, without completely erasing it. In the case of the traditional C-sketch method, participants were asked to circulate their sketches clockwise to their adjacent teammate. For the full skWiki condition, participants could choose any of their teammates' sketches from any stage, but not their own. This restriction was imposed to prevent participants from continuously working on their own idea for the duration of the session. At the end of the session, participants were asked to spend 5 minutes to discuss and select the most promising concept from the set of available concepts.

Data Collection

Participants were asked to respond to questions pertaining to the usefulness and ease of the methods on a Likert scale.

A log of participant choices in rounds two and three in the skWiki condition was also recorded to identify cases of departure from the traditional method afforded due to the branching (creating copies) and history support (choosing a sketch from the first round during the third). Finally, each team's selection of the "most promising idea", and its corresponding round, was noted.

RESULTS AND DISCUSSION

The branching operations performed by participants in each team for both the traditional C-sketch and the full skWiki conditions are shown in Figure 5. Below we discuss how features that are unique to skWiki were used by the teams.

Cloning and History Rollback

The C-sketch flow of design follows a linear sharing model through a "passing the paper around" paradigm prescribed by the method that the participants were required to follow. For the skWiki method, participant activity shows instances of multiple copies of a sketch in every round, for every single team. Of these instances, three teams branched out (cloned) from earlier versions of their team members' sketches, made possible through the "history rollback" support. Three out of four teams selected their final design from the last set of iterations, shown by the "starred" nodes in Figure 5. However, it is noteworthy that one team selected a design from their second round, which would have been lost had it not been for the history rollback support.

Participant responses to survey questions support the usefulness of the branching and history rollback afforded by skWiki: of the 12 participants, 11 preferred the full skWiki model of C-sketch. Reasons cited for the preference ranged from the ability to choose a more promising idea, the availability of a larger variety of ideas to choose from, and the ability to see more popular or "growing ideas", as one participant put it. Participants also cited the ease of collaboration as another reason for preferring skWiki. The one participant who preferred the traditional C-sketch model of sharing cited his reason as the full skWiki system allowing the designer to stick to a limited set of designs if he so chooses, as opposed to C-sketch, which ensured that everyone worked on everyone else's designs. However, all 12 participants reported that they found the option to select ideas useful.

Path Viewer with Preview

Recall that the traditional C-sketch path viewer for the user study was configured to show only the latest sketches of all the users at any point of time, while for the skWiki model it showed the revisions of all users that were uploaded at the end of each 6-minute session. The post-survey responses suggest that a majority of the users (average of 75%) found it easy to decide on a version to download for the next round during full skWiki model even though more versions are shown than in the traditional C-sketch. For the full skWiki sharing model, we anticipated a decrease in the ease of browsing and choosing ideas in later rounds owing to an increase in the number of ideas to choose from. Participant responses, however, were mixed: 7 out of 12 participants mentioned that it was easy for them to choose an idea to work on in the second round, while 1 participant reported finding it difficult. 4 participants were undecided. Surprisingly, the number of participants who found it easy to choose ideas increased for the third round to 9 participants, while 3 participants found it difficult. Interestingly, this increase was accompanied by a mix of transitions: all participants who found it difficult to choose in the previous round found it easier to choose in the last round, whereas 3 participants who found choosing in the second round easy, had the opposite experience in the third round. The increase in participant ease could be explained by a greater familiarity with their team member's designs by the third round, assuming changes are clear in the thumbnail view. A more complex design change, however would entail checking out the sketch and examining it closely, a process that becomes more tedious the more choice one has. This is echoed by the participants: some suggested using larger previews, or larger thumbnails with the facility to flip through them easily.

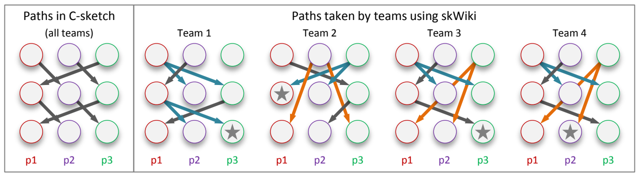

Figure 5. Comparison between paths taken in the C-sketch model (left) and skWiki (right) in the user study. Each gray node represents a sketch by a participant (labeled as p1, p2, p3) at the end of every round. Standard "passing on a sketch" operations are shown as gray arrows, branching to create multiple copies is aquamarine, and branching from history is orange. Stars indicate a sketch was selected as the best design.

From a methodology point of view, it is premature, based on this study alone, to conclude that more choice for the designer is better. In fact, allowing the designer, especially a design engineer, to freely choose a design could lead to fixation, as engineers tend to favor previously encountered designs or designs they developed themselves [15]. However, the purpose of skWiki is not to merely provide choice, but to preserve every stage of work as well as to allow for potential branching (cloning) of ideas at every such stage. In the context of the C-sketch method, Shah et al. state saturation—participants feeling "that they could no longer contribute to the idea generation process" [29, p. 191]—as one of the issues of their method. With skWiki, this saturation can be delayed since each designer can have the opportunity to work on every other designer's *initial* design, without incremental additions or modifications of features done by other designers, thus geometrically increasing the number of potential iterations. Additionally, designers can return to the problem days later and pick up where they left off, owing to the persistence of every state of their design on the server. Finally, as seen with Team 2 in Figure 5, skWiki preserves promising ideas that would otherwise be lost to further iterations.

DESIGN IMPLICATIONS

A user of the Web 2.0 generation is not only a consumer of digital content, but a creator, distributor, and marketer as well. However, most current content creation tools are geared towards convergent processes which strive to create a single article, a unified data table, or a common illustration agreed upon by all. In this work, we have tackled a diametrically opposite approach—a collaborative creativity framework for divergent processes: many design alternatives, multiple iterations, and competing yet comfortably coexisting versions. However, in traversing this path (no pun intended), we had to make several design decisions that affected the final skWiki implementation presented in this paper.

Our skWiki implementation uses the paths concept to support effortless collaborative creating, sharing, and merging for multimedia. A key component for managing these meandering paths is the path explorer, which not only tracks paths in space but also in time and across multiple users. However, a visual path explorer of this type will inevitably encounter presentation difficulties as the number of revisions and users grows. For example, our current explorer implementation uses a node-link representation that would not scale to more than a few hundred revisions and users. Because of this, applying it on a large scale in a system such as Wikipedia would simply not be feasible. More work is required here not only in visual summaries and alternate representations, but also in methods for filtering, navigating, and searching within paths and between users in the path explorer.

Scale also affects more than the mere interface layer of the system. One weakness of the paths model is that it could lead to a profliferation (if not explosion) of concurrent paths, with much of the data being redundant and replicated. Even deleted files would remain in storage forever, effectively making it impossible to ever "clean up" a hard disk or storage system. On the other hand, a paths model with appropriately designed DSL operations can also be significantly more economical than an equivalent file system. For one thing, storing an operation is often less space-consuming than storing its effect; one example is inverting a raster image. Second, the branch operation in the paths model, which corresponds to a copy in a traditional file system, is extremely lightweight: an empty branch simply contains a few bytes to store the new path identifier and track the parent path and revision number. Nevertheless, we recognize that the paths model described here is not a general replacement for a traditional file system by any measure. We simply found it well suited to our overall design rationale.

One potential weakness of the paths model is the merge operation, which is somewhat difficult to characterize to the user. It is partly a copy operation, because it replicates one or more transformation operations from a source path to a destination path. Also, it is typically performed on paths that are somehow related, for example, having a common ancestor. In our current implementation, merging two paths is operationally equivalent to branching a new child from one of the two paths followed by applying one or more latest operations of the other. However, merging is an even more powerful concept, which for example can allow identical operations to affect a large set of paths based on the operation sequence. More work is needed to explore the potential of the merge operation in future versions of skWiki. Further, the current version of skWiki does not incorporate consensus: a final idea or set of ideas that are selected by a team. In our model, all ideas exist simultaneously, and final ideas are not explicitly shortlisted or tagged for future reference. We plan to explore the use of collaborative tagging [11] to allow for tag-based selection and filtering in the paths.

One of the most closely related existing projects for skWiki is Google Docs, which supports much of the same functionality for collaborative multimedia authoring while retaining a revision history. However, compared to skWiki, Google Docs lacks many of our visualization mechanisms as well as multi-user revision tracking. It is also designed for a convergent workflow. For a group of brainstorming toy designers, one alternative be to use a single Google Docs document where each designer works on separate pages while routinely referring to each other's work. However, branching from another designer's work is not a native operation in Docs, and requires replicating that work first before editing.

Finally, it is worth comparing skWiki to wikis, which the system at least shares some common ancestry with. For one thing, wikis are notoriously difficult to use with anything other than textual content. Images are not first-class objects in a typical wiki software, and must be edited using offline desktop applications. Part of the goal for skWiki is to provide a multimedia authoring environment that is not dependent on offline desktop applications, at least not for the most common operations. However, the differences go a lot further than this: a traditional wiki and skWiki represent two radically different designs. Whereas a wiki has one copy of each document and will always show its latest version, skWiki is based on the very concept of multiple and concurrent versions of a document across time, space, and users.

CONCLUSION AND FUTURE WORK

We have presented skWiki, a web-based content authoring framework for creative processes that implements an abstract paths model in favor of a traditional file system. Paths represent entity state over time, and consists of the operations that were performed to create and modify an entity rather than snapshots or diffs. This model is particularly powerful for multi-user collaborative settings where the aim is to brainstorm and generate many design alternatives for a particular theme. The skWiki system is a practical paths implementation and allows users to collaborative work on multimedia documents consisting of images, vectors, sketches, layouts, and rich text. To validate the work, we conducted a qualitative user study involving four teams of three engineering students using the tool for designing children's toys.

This work is merely one contribution to a dialogue of how to support the new content authoring model where users themselves are involved in all stages of the process. We expect this dialogue to continue well into the future. Our own future research directions include supporting additional media types, improving the fluidity of the interface where it entirely replaces desktop applications, and further exploring the paths model presented here to its full potential.

ACKNOWLEDGMENTS

This work is partly supported by the U.S. National Science Foundation on grants IIS-1227639, IIS-1249229, and IIS-1253863, as well as the Donald W. Feddersen Chaired Professorship and the Purdue School of Mechanical Engineering. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

1. Baecker, R. M. *Readings in Groupware and Computer-Supported Cooperative Work*. Morgan Kaufmann Publishers, San Francisco, 1993.
2. Bruffee, K. A. Collaborative learning and the conversation of mankind. *College English*, 46(7):635–652, 1984.
3. Buffa, M., Erétéo, G., and Gandon, F. A wiki on the semantic web. In *Emerging Technologies for Semantic Web Environments: Techniques, Methods and Applications*, 115–137, 2008.
4. Chen, H.-T., Wei, L.-Y., and Chang, C.-F. Nonlinear revision control for images. *ACM Transactions on Graphics*, 30(4):105:1–105:10, 2011.
5. Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. ACM Computer Supported Cooperative Work*, 1277–1286, 2012.
6. Denning, J. D., Kerr, W. B., and Pellacini, F. MeshFlow: interactive visualization of mesh construction sequences. *ACM Transactions on Graphics*, 30(4):66:1–66:8, 2011.
7. Geyer, F., Budzinski, J., and Reiterer, H. IdeaVis: a hybrid workspace and interactive visualization for paper-based collaborative sketching sessions. In *Proceedings of the Nordic Conference on Human-Computer Interaction*, 331–340, 2012.
8. Greene, S. L. Characteristics of applications that support creativity. *CACM*, 45(10):100–104, 2002.
9. Grossman, T., Matejka, J., and Fitzmaurice, G. Chronicle: capture, exploration, and playback of document workflow histories. In *Proc. ACM User Interface Software & Technology*, 143–152, 2010.
10. Hailpern, J., Hinterbichler, E., Leppert, C., Cook, D., and Bailey, B. P. TEAM STORM: demonstrating an interaction model for working with multiple ideas during creative group work. In *Proceedings of the ACM Conference on Creativity & Cognition*, 193–202, 2007.
11. Halpin, H., Robu, V., and Shepherd, H. The complex dynamics of collaborative tagging. In *Proc. ACM Conference on the World Wide Web*, 211–220, 2007.
12. Heer, J., Mackinlay, J. D., Stolte, C., and Agrawala, M. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, 2008.
13. Hilliges, O., Terrenghi, L., Boring, S., Kim, D., Richter, H., and Butz, A. Designing for collaborative creative problem solving. In *Proceedings of the ACM Conference on Creativity & Cognition*, 137–146, 2007.
14. Ishii, H., and Kobayashi, M. ClearBoard: a seamless medium for shared drawing and conversation with eye contact. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 525–532, 1992.
15. Jansson, D. G., and Smith, S. M. Design fixation. *Design Studies*, 12(1):3–11, 1991.
16. Kurlander, D., and Feiner, S. Editable graphical histories. In *Proceedings IEEE Workshop on Visual Language*, 127–134, 1988.
17. Louridas, P. Using wikis in software development. *IEEE Software*, 23(2):88–91, 2006.
18. Mernik, M., Heering, J., and Sloane, A. M. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, Dec. 2005.
19. Minneman, S. L., and Bly, S. A. Managing a trois: A study of a multi-user drawing tool in distributed design work. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 217–224, 1991.
20. Osborn, A. *Applied imagination; principles and procedures of creative problem-solving*. Scribner, 1963.
21. Priedhorsky, R., and Terveen, L. Wiki grows up: arbitrary data models, access control, and beyond. In *Proceedings of the International Symposium on Wikis and Open Collaboration*, 63–71, 2011.
22. Rittel, H., and Webber, M. Dilemmas in a general theory of planning. *Policy Sciences*, 4(2):155–169, 1973.
23. Robbes, R., and Lanza, M. A change-based approach to software evolution. *Electronic Notes in Theoretical Computer Science*, 166:93–109, 2007.
24. Rochkind, M. J. The source code control system. *IEEE Trans. in Software Engineering*, 1(4):364–370, 1975.
25. Rohrbach, B. Creative nach regeln: Methode 635, eine neue technik zum lösen von problemen. *Absatzwirtschaft*, 12(19):73–75, 1969.
26. Sabel, M. Structuring wiki revision history. In *Proceedings of WikiSym*, 125–130, 2007.
27. Sangiorgi, U. B., Beuvens, F., and Vanderdonckt, J. User interface design by collaborative sketching. In *Proceedings of the ACM Conference on Designing Interactive Systems*, 378–387, 2012.
28. Schaffert, S. IkeWiki: A semantic wiki for collaborative knowledge management. In *Proceedings of the IEEE International Workshop on Enabling Technologies*, 388–396, 2006.
29. Shah, J. J., Vargas-Hernandez, N., Summers, J. D., and Kulkarni, S. Collaborative sketching (c-sketch)–an idea generation technique for engineering design. *Creative Behavior*, 35(3):168–198, 2001.
30. Streitz, N., Geissler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., and Steinmetz, R. i-LAND: An interactive landscape for creativity and innovation. In *Proc. ACM Conf. on Human Factors in Computing Systems*, 120–127, 1999.
31. Tang, J. C., and Minneman, S. L. VideoDraw: a video interface for collaborative drawing. *ACM Transactions on Information Systems*, 9(2):170–184, 1991.
32. Tazzoli, R., Castagna, P., and Campanini, S. E. Towards a semantic wiki wiki web. *Demo at ISWC*, 2004.
33. Torrance, E. P. The nature of creativity as manifest in its testing. In *The Nature of Creativity: Contemporary Psychological Perspectives*. Cambridge Univ. Pr., 1988.
34. Wagner, C. Breaking the knowledge acquisition bottleneck through conversational knowledge management. *Information Resources Management Journal*, 19(1):70–83, 2006.
35. Yoon, Y., Myers, B. A., and Koo, S. Visualization of fine-grained code change history. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, 2013.