ReadWriter: Task Automation and Feedback Support for Bloggers with Inline Syntax [[ ]]

Juho Kim[1] Chen-Hsiang Yu[1] Robert C. Miller[1] Krzysztof Z. Gajos[2]

[1] MIT CSAIL
Cambridge, MA 02139
{juhokim, chyu, rcm}@mit.edu

[2] SEAS Harvard University
Cambridge, MA 02138
kgajos@seas.harvard.edu

ABSTRACT

This paper presents ReadWriter, a novel blogging interface that enables task automation and offers on-demand writing feedback. Bloggers can express any writing-related help requests in natural language inside double brackets. ReadWriter interprets a blogger's requests and assigns them to a diverse group of contributors: software search agents, an anonymous crowd, the blogger's social connections, and readers. The entire cycle of creating a request, checking status, reviewing results, and applying to a draft occurs inside the blogging interface. Design goals are to help writers maintain flow by automatically managing housekeeping tasks, and to engage readers earlier on in the writing process. Analysis of collected requests identified distinct categories of tasks people expect from the system. Five bloggers in the lab study found the tool easy to learn and use, and exhibited various usage patterns during a writing task.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Human Factors

Keywords: Writing, reading, blogs, creativity support tools, reader-writer interaction, outsourcing.

INTRODUCTION

Blogs have gained popularity as an effective communication medium for personal broadcast. Although many blogging platforms claim to lower the barrier in blogging, a recent trend suggests a shift toward simpler forms of communication within a closed social network. A recent NY-Times article [14] quotes:

"Former bloggers said they were too busy to write lengthy posts and were uninspired by a lack of readers."

Two common challenges for bloggers are to create richer content to attract more readers, and to get more feedback from others. A popular technique for creating richer content is embedding visual aids and links to provide more engaging reading experience. In our interviews with bloggers and writers, one blogger notes:

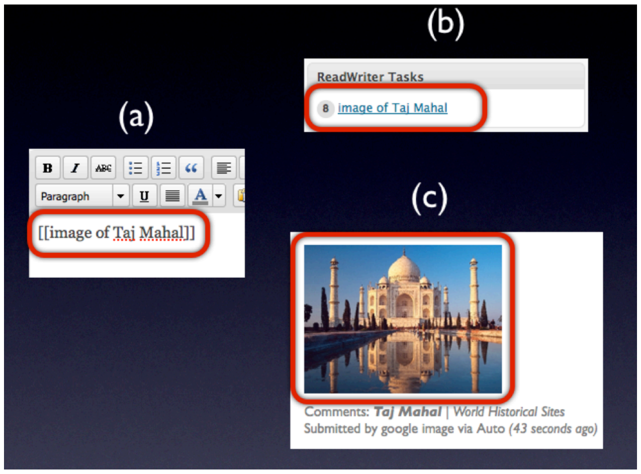

(a)
(b)
(c)
Paragraph
[[image of Taj Mahal]]
ReadWriter Tasks
8 image of Taj Mahal
Comments: Taj Mahal | World Historical Sites
Submitted by google image via Auto (43 seconds ago)

Figure 1. (a) When a blogger creates a [[ ]] task, (b) the task list captures the blogger's request, and (c) automatic search results are shown to the blogger.

"I really think every post needs to have a picture. It does get people interested. A picture, or video or something. But if I only have half an hour, I can't blog because… I have to find pictures, get the links and whatever. Sometimes I skip it. I call that housekeeping work. I don't have time to do the housekeeping."

Housekeeping tasks often involve tedious steps, such as searching, browsing, copying, and importing multimedia or links to the draft. Bloggers suffer from continuous distractions, broken flow, and wasted time, which result in resorting to plain-text posts and even opting out of blogging.

Building on other's feedback is a proven technique to improve the quality of writing [8]. Keh [11] notes *"Reader feedback on the various drafts is what pushes the writer through the writing process on to the eventual end-product."*

We argue that bloggers lack an efficient feedback platform. Commenting, the most popular feedback mechanism today, allows interactions with readers, but there are nontrivial limitations. First, comments are available only after a post is published; there is no convenient way for bloggers to get feedback while writing. Also, a vast majority of lurking readers are not motivated enough to be active contributors. The spokeswoman at LiveJournal, a commercial blogging platform, mentions [14] that *"Blogging can be a very lonely occupation; you write out into the abyss"*.

We conducted interviews with four writers and bloggers to better understand their writing process and needs for tool support. The interviews confirmed our view that bloggers suffer from the cumbersome process of searching for external sources and a lack of feedback. The findings suggest two design goals: to help writers maintain flow by automatically managing tedious tasks, and to engage readers earlier on in the writing process for enhanced feedback support.

This paper presents ReadWriter, a novel blogging interface that address the two design goals with a simple solution: double brackets `[[ ]]`. The system enables task automation for housekeeping tasks and offers writing help. Double brackets are inline helper syntax for arbitrary tasks that a blogger might need. A task can be as simple, as in `[[find an image of a peacock]]`, or more contextual as in `[[Bob, can you find another example to support this paragraph?]]`. The blogger does not have to press a button or learn markup language syntax, and simply can express intentions in natural language.

Once a task is created, the system automatically interprets the natural language query and delegates it appropriately. Based on the nature of the task, ReadWriter can route the task request to a diverse pool of contributors: search agents, a crowd, social connections, and readers. The entire cycle of creating a request, checking status, reviewing results, and applying results to a draft occurs inside the blogging interface.

In order to understand what kind of tasks people expect ReadWriter to handle, we picked 72 blog posts spanning various themes and popularity. Then we asked workers on Mechanical Turk to each submit three bracket requests that they might imagine using for the assigned post. From these responses, we identify several main classes of tasks: search delegation, writing help, fact verification, , and formatting.

The next question, then, is how do bloggers actually use ReadWriter in writing? We invited five bloggers to a lab study and assigned a writing task with ReadWriter enabled. The bloggers found the bracket syntax easy to learn and use, and created tasks for various purposes.

Main contributions of this work are: 1) the lightweight interface for expressing and managing writing help requests in natural language, 2) the support for diverse contributor groups for writing tasks, 3) the new mode of feedback support: blogger-initiated, directed, fine-grained feedback requests while writing.

With this system, we claim that writers can maintain their flow while the system manages tedious work. The simple natural language syntax helps bloggers delegate housekeeping tasks to a diverse set of contributor groups. Also, bloggers can improve the quality of their writing by issuing on-demand feedback requests. This engages readers earlier on in the writing process. The new feedback mechanism is a step toward the vision of enhanced reader-writer interaction, turning readers into active contributors.

In this paper, we first outline related work in the context of design space. We illustrate a usage scenario, followed by design decisions from user observations. The system section walks through the lifecycle of a task and presents frontend and backend systems. We report categories of task and bloggers' experience with the system. We conclude by discussing the findings and presenting the vision of better reader-writer interaction.

RELATED WORK

The idea of writing support tools that provide writers with external help is not new. Each tool tackles different segments in the design space. We outline three major design dimensions for the writing aids and discuss related work for each. The dimensions can be represented with the following questions: 'Who does the work for writers?', 'What tasks are supported by the tool?', and 'How does the tool capture a writer's intention?'.

Who does the work?

One solution presents writers with automatic recommendations selected by ambient agents. JITIR (Just-In-Time Information Retrieval) agents [18] perform proactive searches based on a user's context, and display relevant results. Concrete example systems include: the Remembrance Agent [19] that suggests relevant, pre-indexed, local documents; Writer's Aid [1] that applies AI planning techniques to perform ambient searches as a collaborative assistant; PIRA [21] that extracts salient terms from the current document and searches the corpus of digital libraries; Watson [6] that returns related web pages, applying term heuristics for query construction and similarity metrics for result clustering; Zemanta [24] that recommends related contents for bloggers, based on text analysis techniques.

Crowdsourced workers have emerged as a powerful source of writing support because of their capability of handling human intelligence tasks at a low cost. Soylent [2] introduces design patterns to improve the quality of crowdsourced work in writing and editing tasks. More research has investigated writing tasks in the crowdsourcing context [12] [15] [10].

Other work has shown how leveraging a user's social network can enhance accuracy and personalization. Bernstein et al. introduced *friendsourcing* [4], a mode of collecting personalized information from one's social connections. Aardvark [9] is a social search engine with a routing algorithm for determining the best answerer to meet a user's needs. There is little work on integrating social network resources with writing tasks.

In blogs, communication between readers and writers mainly remains one-way. Blog Muse [7] and Reflect [13], covered in more details later, attempt to bridge the gap between readers and writers with novel interactions. Some work has explored how passive readers can evolve into active contributors in social communities [17], but few work addresses the blogging context specifically.

In summary, previous research has focused on completing a writing-related task with a single worker type. ReadWriter supports a mixed pool of worker groups, namely automatic agents, crowdsourced workers, social network, and designated contacts. It overcomes the complexity of handling different groups with task interpretation and automatic routing algorithms.

What tasks are supported?

Soylent [2] offers three types of writing help for MS Word users: shortening text to a desirable length, proofreading, and arbitrary requests. While ReadWriter tackles similar writing struggles as Soylent, there are two major differences. First, ReadWriter focuses on optimizing the user's workflow of issuing a query, reviewing results, and updating content inline. The main contribution of Soylent is the patterns for crowd-powered interfaces. The two techniques can potentially be combined in a larger system. Also, ReadWriter adds social network and automatic search agents as contributors, in addition to crowd workers.

In the early stage of writing, choosing a topic is an important decision for bloggers. Blog Muse [7] allows enterprise bloggers to benefit from readers' topic suggestions. ReadWriter addresses later steps in the writing process, where bloggers can issue directed, fine-grained, and specific feedback requests while writing. At the post-publication stage, Reflect [13] facilitates active listening among discussants with enhanced commenting.

How is a task created and managed?

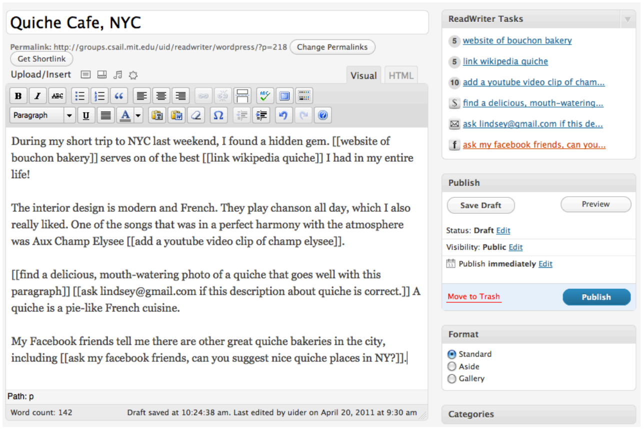
JITIR agents do not require explicit input from a user, because the system's role is predefined and constant [1]. In ReadWriter, the user initiates tasks with explicit `[[ ]]`.

Using specific syntax to indicate the user's writing and editing intentions is common in text editing languages. Wordpress Shortcodes [22] are shorthand macros that simplify complex embed operations into short code. A downside of Shortcode is that users still have to search for a URL manually and learn syntax. ReadWriter allows complete natural language queries without imposing any format constraints. Natural language-like commands can be found in the programming context as well [16].

USAGE SCENARIO

We introduce ReadWriter with a hypothetical user's blogging scenario. Carrie is a freelance writer who keeps a blog about her dining and cooking experience, as well as useful recipes and diet tips. Her blog has a few thousand readers and hundreds of RSS subscribers, and each page features the Twitter Retweet and Facebook Like buttons. The blog has drawn some clients' attention, including food magazines and local newspapers. She utilizes her blog to showcase her work and reach out to potential clients.

One day, she decides to write about a splendid quiche bakery called Bouchon Bakery she visited during her recent New York City trip. She plans to write a review of the place and dishes, as well as a paragraph with interesting background on quiche. She opens the Wordpress post page.


Quiche Cafe, NYC

Permalink: http://groups.csail.mit.edu/uid/readwriter/wordpress/?p=218

During my short trip to NYC last weekend, I found a hidden gem. [[website of bouchon bakery]] serves on of the best [[link wikipedia quiche]] I had in my entire life!

The interior design is modern and French. They play chanson all day, which I also really liked. One of the songs that was in a perfect harmony with the atmosphere was Aux Champ Elysee [[add a youtube video clip of champ elysee]].

[[find a delicious, mouth-watering photo of a quiche that goes well with this paragraph]] [[ask lindsey@gmail.com if this description about quiche is correct.]] A quiche is a pie-like French cuisine.

My Facebook friends tell me there are other great quiche bakeries in the city, including [[ask my facebook friends, can you suggest nice quiche places in NY?]].

Path: p

Word count: 142 Draft saved at 10:24:38 am. Last edited by uider on April 20, 2011 at 9:30 am

ReadWriter Tasks

5 website of bouchon bakery

5 link wikipedia quiche

10 add a youtube video clip of cham...

S find a delicious, mouth-watering...

✉ ask lindsey@gmail.com if this de...

f ask my facebook friends, can you...

Publish

Status: Draft Edit

Visibility: Public Edit

Publish immediately Edit

Format

● Standard
○ Aside
○ Gallery

Figure 2. Carrie's Wordpress authoring interface with ReadWriter installed. The top right sidebar keeps a list of tasks in the current document.

One of the sidebars is named *ReadWriter Tasks*, which is currently empty.

In her first paragraph, she wants to add the website link, map, and telephone number of the restaurant. In a place where she wants the link to be, she types in `[[website of bouchon bakery]]`. For readers not familiar with quiche, she also inserts `[[link wikipedia quiche]].`

She decides to add a video clip of the French song *Aux Champs Elysees* that was playing in the bakery. Unsure about the exact spelling, she types in `[[add a youtube video clip of champ elysee]]` and moves on.

In the second part of the article, she wants to include the background information on quiche. Then she adds `[[find a delicious, mouth-watering photo of a quiche that goes well with this paragraph]].` She notices a S icon on the list for this task, indicating that ReadWriter assigned a *hired crowd* to the task.

After drafting the paragraph on quiche, she is worried about the accuracy of content. She would like to have her professional cook friend Lindsey to check the paragraph for any errors. At the end of the paragraph she adds `[[ask lindsey@gmail.com if this description about quiche is correct]].`

Finally, she concludes the piece with useful information to readers. `[[ask my Facebook friends, can you suggest nice quiche places in NYC?]].` During the entire drafting phase she didn't open any web page.

Now she has a first draft. She goes over each item in the task list, immediately reviewing and selecting results for image, video, and link queries and picks the best one from each results page. The selection automatically replaces the `[[ ]]` content in the editor. For the bottom three, she clicks on the link and confirms creating a HIT on Mechanical Turk, sending Lindsey an email, and posting a Facebook status update. All help requests include a URL these contributors can use to submit their input.

Carrie opens the saved draft after a few hours to check the status for the three human tasks. She gets nice quiche images from Turkers that are more customized to her needs than generic Google Images search results. Her friend Lindsey was fast enough to send her feedback on the draft paragraph, and four Facebook friends got back with suggestions for other quiche places. All the submitted results are collected and displayed by the interface, so she need not visit other sites to check individual results.

DESIGN PROCESS

We adopted an iterative design process for the development of the tool. Three events that had significant influence on the tool design were a paper prototype pilot study, interviews with bloggers and writers, and functional prototype tests. The paper prototype study involved three graduate student bloggers. We asked them to handwrite a blog post with `[[ ]]` notations for outsourcing needs. We observed how and when people use `[[ ]]`. Multiple iterations of functional prototype design helped us refine the system. The rest of this section discusses findings from the interviews and design decisions that followed.

Interview

We interviewed 4 professional writing instructors, writers, and bloggers around the *(city)* area. Our goal was to better understand their writing process and discover latent needs for tool support. We asked the interviewees to walk us through a representative writing task (for writers and bloggers) and/or assessment task (for instructors). While the writers and bloggers shared their expertise from the perspective of a content creator, the writing instructors offered the perspective of a feedback provider. At the end of each session, we verbally described the ReadWriter blogging support system and asked for qualitative feedback.

P1, writer and writing instructor, mentioned that he always keeps a list of TODO items at the bottom of a document when he writes. He adds items to the list as he writes and revisits the list once he is through a first draft.

Another blogger/instructor/writer (P2) feels that 30% of her blogging time is wasted in taking care of *housekeeping work*, which she describes as *“kind of a pain to go ... copy and paste it, do a little hover over thing”*. When not confident about the writing, she emails the draft to her colleagues and family members to get feedback before publishing it. As a writer, the quality of writing is one of her top priorities in managing her blog.

P3 is a writer and English professor who has two published books and blogs about his writing practice. In his view, current blogging tools make it *“too easy to be hung up on details like formatting and beautifying”*. He believes getting feedback from different audiences is useful, helping writers to see *“outside of their self-confining world view”*.

Two themes emerged from the interviews. First is need for a tool designed to minimize distractions while writing by automatically managing housekeeping tasks. This led to the important design decision of allowing complete *natural language* tasks. One early ReadWriter design guided users to fill in a structured template from a dropdown menu with predefined options. After pilot users struggled with learning and adapting to the new interaction, the design evolved into completely format-free syntax except for double brackets.

The second theme is a lightweight process to reach out to different reader groups for feedback. The effort of initiating specific feedback requests is often a barrier for writers, in addition to managing their social capital in asking a favor. We envisioned an interface that allows writers to make inline `[[ ]]` requests and send them out with a single click. The following sections explain how ReadWriter takes a step toward the vision.

INTERFACE DESCRIPTION

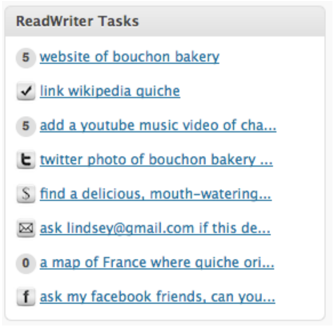
Writer Interface

We built a Wordpress plugin named ReadWriter to prototype the key ideas. The design goal of this interface is to minimize any visual clutter or input overhead on the writer’s side. All complexity of interpretation and configuration is handled by the system and management interface. This lets bloggers focus on the writing. As Figure 3 shows, the ReadWriter plugin adds a side panel to the Wordpress authoring interface.

The system accepts any natural language query inside brackets, without any grammar or format constraints. Table 1 on the next page presents example tasks collected from the Mechanical Turk experiment and user study sessions with bloggers.

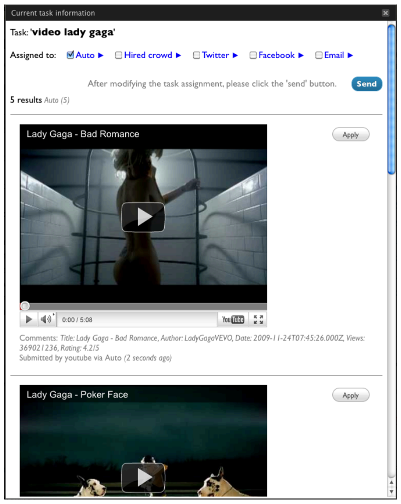
When the blogger adds a `[[ ]]` task in a draft, the interface automatically adds an item to the list on the right. The order in the list follows the position in the draft, to help writers navigate the list and the draft in synchronization.

Task information includes task status and its summary. The task status is displayed with a small icon to minimize visual clutter in the list. Figure demonstrates various states a task can have. Upon creating a task, the blogger sees either 0 or one of ✉ f t S icons. 0 means the system interpreted the task to be automated without any human input. The number inside indicates the number of results that have been collected so far for this task.


ReadWriter Tasks

- 5 website of bouchon bakery
- ✓ link wikipedia quiche
- 5 add a youtube music video of cha...
- t twitter photo of bouchon bakery ...
- S find a delicious, mouth-watering...
- ✉ ask lindsey@gmail.com if this de...
- 0 a map of France where quiche ori...
- f ask my facebook friends, can you...

Figure 3. Side panel view with task status and summary



Current task information

Task: '**video lady gaga**'

Assigned to: [✓ Auto] ▶ ☐ Hired crowd ▶ ☐ Twitter ▶ ☐ Facebook ▶ ☐ Email ▶

After modifying the task assignment, please click the 'send' button. Send

5 results *Auto (5)*

Lady Gaga - Bad Romance

0:00 / 5:08

Comments: *Title: Lady Gaga - Bad Romance, Author: LadyGagaVEVO, Date: 2009-11-24T07:45:26.000Z, Views: 369021236, Rating: 4.2/5*
Submitted by youtube via Auto *(2 seconds ago)*

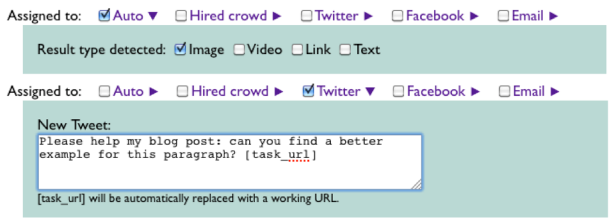
Lady Gaga - Poker Face

Figure 4. Task management interface: worker assignment and results are shown.

On the other hand, ✉ f t S indicate that the system is waiting for user confirmation on the task that involves human helpers. Each of the four icons represents a distinct group of human contributors: ✉ is specific email addresses, f is Facebook friends, t is Twitter followers, and S is a hired crowd on MTurk. The original version of the prototype automatically sent out task notification without user confirmation to minimize user actions. In our design process, however, pilot users expressed concerns on their help requests spamming other's email inbox or news feed. A similar observation was made in a link sharing system FeedMe [3]. The current version mandates user confirmation before requests are sent out to any human worker.

The blogger can manage a specific task by clicking on the task link in the list. The task management interface (Figure 4) lets users edit the current interpretation of the task and send requests to any human groups. The bottom part of this interface presents results collected for this task.

For each of the five available worker groups, the user can click open a setting menu (Figure 5) and customize the Twitter/Facebook/email message or configure other fields.



Assigned to: [✓ Auto] ▼ ☐ Hired crowd ▶ ☐ Twitter ▶ ☐ Facebook ▶ ☐ Email ▶

Result type detected: [✓ Image] ☐ Video ☐ Link ☐ Text

Assigned to: ☐ Auto ▶ ☐ Hired crowd ▶ [✓ Twitter] ▼ ☐ Facebook ▶ ☐ Email ▶

New Tweet:

```
Please help my blog post: can you find a better
example for this paragraph? [task_url]
```

[task_url] will be automatically replaced with a working URL.

Figure 5. Worker assignment control for automated (top) and Twitter tasks (bottom). The blogger can customize options for each worker.

Once the user manually sends out task requests to designated human groups, the status icon automatically switches to 0 to inform that the system has now begun gathering results from workers. This number gets updated as the system receives results.

Below the menu follow results, with a customized view for each result type. The Apply button next to each result inline-updates the selected result to the current draft. Clicking the button replaces the corresponding `[[ ]]` with the selected result. The task status now changes to ✓, meaning completed.

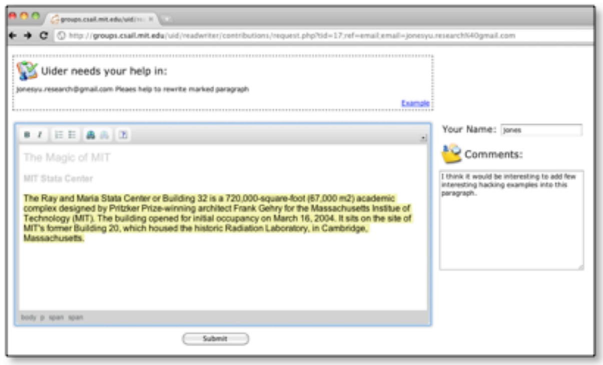
Contributor Interface

For human workers to handle writers' requests, they need an interface to access task information and submit their results. The ReadWriter system provides a simple UI for all human contributors. When a blogger specifies contributors for a specific task, the contributors receive a URL to access the contributor's interface.

As Figure 6 illustrates, the contributors interface consists of task description, rich-text editor, a link to an example, a user name field, comments, and a submit button.

Category	Information	Task Example
Search Delegation	Image	[[find an image of shale gas extraction]] *(from Turker)*
	Link (URL)	[[President Cancer Panel Report from 2010 web link]] *(from Turker)*
	Link (Document)	[[Richard Stanley, q-deformation hyperplane arrangement, pdf, arXiv]] *(from Blogger)*
	Video	[[find good videos about tina fey]] *(from Turker)*
	Fact	[[find full text of Robert burns poem tooth-ache]] *(from Blogger)*
	Image (complex)	[[Find an image to go with the article (probably of mentioned house of worship or religious imagery)]] *(from Turker)*
Writing Help	Mechanical	[[shorten Allen's quote to get to the heart of it, make easier to read]] *(from Turker)*
	High-level	[[make paragraph: one way that people share images of a trip is with a scrapbook (at least, women do, and a women suggested this to me) say what scrapbooking is]] *(from Blogger)*
Fact Verification	Confirming facts	[[research if this has been an issue with any other past president]] *(from Turker)*
Formatting	Simple fixes	[[center photo]] *(from Turker)*

Table 1. Task examples with category and a brief description. They are collected from either Turkers or bloggers from our evaluation studies.


http://groups.csail.mit.edu/uid/readwriter/contributions/request.php?tid=17;ref=email;email=jonesyu.research%40gmail.com

Uider needs your help in:

jonesyu.research@gmail.com Pleaes help to rewrite marked paragraph

Example

The Magic of MIT

MIT Stata Center

The Ray and Maria Stata Center or Building 32 is a 720,000-square-foot (67,000 m2) academic complex designed by Pritzker Prize-winning architect Frank Gehry for the Massachusetts Institue of Technology (MIT). The building opened for initial occupancy on March 16, 2004. It sits on the site of MIT's former Building 20, which housed the historic Radiation Laboratory, in Cambridge, Massachusetts.

Your Name: jones

Comments:

I think it would be interesting to add few interesting hacking examples into this paragraph.

Submit

Figure 6. Contributor's interface: all human workers submit their results using this web interface.

The task description is the original content inside double brackets. Contributors can edit the assigned task within the rich-text editor and add comments to the writer for any clarification or additional notes. The text editor displays context for the current task, the enclosing paragraph, to help contributors better understand writer's intention. The context is divided into editable and uneditable regions, highlighted with gray and yellow, respectively. Contributors can optionally specify their names that help writers to recognize and credit them for their work. Once the contributor submits a result, it is accessible to the blogger instantaneously, with an updated result count.

BACKEND SYSTEM

This section describes the backend system architecture in further detail. As a blogger adds `[[ ]]` in the editor, the task detector captures all instances of brackets and sends them to the task interpreter (Figure 7(1)). The interpretation results are then stored in the database (Figure 7(2)). Based on this information, the task dispatcher either starts automated tasks or awaits user confirmation if human workers are involved. For the automated tasks, a search agent gets results using external search APIs. The information extractor parses the result data and stores it to the database (Figure 7(5) and 7(6)). On the other hand, if the task requires human labor, the task router sends out a unique URL for the contributor's interface. Contributors from different sources access this interface to conduct the outsourced task (Figure 7(7)). When the contributors submit their results, the results are stored in the database. The Wordpress plugin periodically checks for any database updates and refreshes the task list. The following subsections discuss important technical decisions and methods.

1) Query Detection

Upon typing two consecutive opening square brackets, the system automatically adds closing brackets and encapsulates this new task with `<span>` tags. The system uses the span element to both retrieve task information and replace with user selection. The system makes Ajax calls to maintain up-to-date task information on the right-hand task list.

2) Query Interpretation

A downside of allowing complete natural language queries is that capturing accurate intent of the blogger becomes

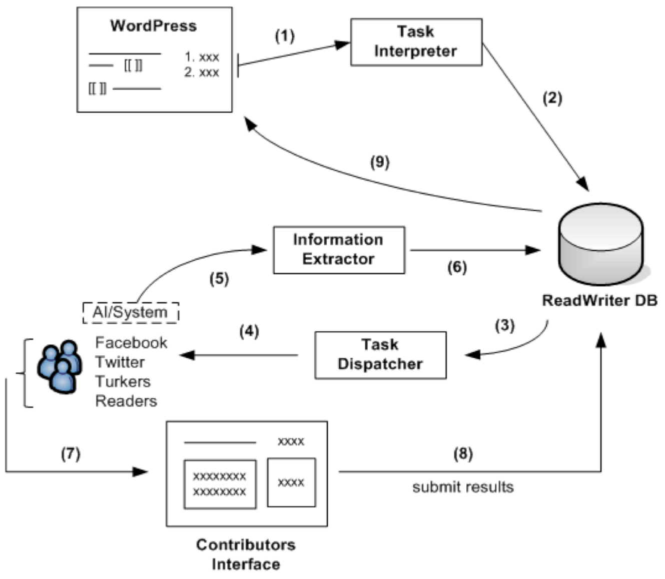


Figure 7: The system diagram with each step in the task life cycle numbered in order.

more difficult. The immediate problem is to determine appropriate work forces for the task. It involves multiple decision criteria such as monetary cost, time delay, user's social network composition and social capital, and readership management. The current prototype addresses some of these issues with heuristics.

Our rule of thumb in worker assignment is that human tasks are more complex, contextual, and subjective feedback requests, whereas automated tasks tend to be mechanical media search queries.

While solving search problems often succeed with using the content inside brackets as search terms, answering feedback requests often requires context outside of the bracket syntax. For example, if the writer issues a task that says `[[find a more compelling example for this paragraph]]`, the system needs to include the content of *this paragraph* in the task request so that contributors can see it. Our current solution is to include the enclosing paragraph of a task as context for all tasks.

For automated tasks, the system needs to extract structured information from a natural language query in order to determine a result type for the task, and transform natural language to a query for search APIs. The result type determines which engine to use for search. The query interpretation strategy can be summarized as follows:

```
1. Given a query [[ ]], determine
    worker from { automatic, hired crowd, Twitter, Facebook, email }
2.a. If worker is human,
    context = { current paragraph }
2.b. If worker is auto,
    query = { content inside brackets } and determine
    result_type from { image, video, link, text }
```

Our solution keeps a predefined pairs of *(input, result_type)*. For example, *image, figure, picture, photo, pic, and icon* all link to the *image* result type. For each word in the query, ReadWriter looks up the word in the dictionary for any match. For any match, the interpreter assigns a type to this query. Multiple types can be assigned. Because the order of words, sentence structure, and grammar of a query do not matter in ReadWriter, even simple heuristics produce reasonable output.

3) Task Assignment

Using the extracted structure from query interpretation, the system routes work to the right group of contributors. For automated tasks, we use Google Images for image search, Youtube for video search, and a combination of Zemanta and Bing for link and generic search. For human tasks, workers are either Turkers or writer's social connections. The task router creates a unique URL for a *(task, worker)* tuple, and includes it in HITs or Twitter/Facebook/email messages. This way different types of contributors can use a single interface accessible with a URL.

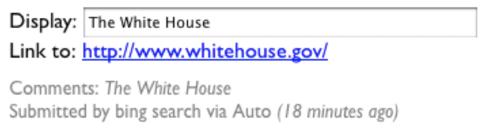

Display: The White House
Link to: http://www.whitehouse.gov/
Comments: The White House
Submitted by bing search via Auto (18 minutes ago)

Figure 8. Result display for a hyperlink.

4) Results Presentation

The interface renders an optimal view for the detected result type: an embedded player for a video clip, a thumbnail preview for an image, and editable anchor text for a hyperlink. For URLs, the interface pre-fills the anchor text with the title of a page as shown in Figure 8. The user can edit this text to customize the rendering of applied results.

5) Inline Update of a Result

The final step in the task lifecycle is to replace the `[[ ]]` content with the user's selection. The code generator minimizes user action by optimizing for each media type.

As shown in Figure 8, if the user is trying to link the official White House website, the result page has already populated the anchor text field with *The White House*. The user can simply click Apply to substitute `[[the white house website]]` with the following HTML code that renders as The White House:

```
<a href='http://www.whitehouse.gov' alt='The
White House'>The White House</a>
```

The user can either apply other results or edit the query to collect new results. In this case, the new results are presented on top of the new ones to preserve workers' effort. The user can always create new queries to get fresh results.

6) Implementation

The entire system is packaged as a single Wordpress plugin. Our plan is to register the software to the Wordpress plugin directory and deploy it to the actual Wordpress user population. With the open-source policy, we expect to attract developers to connect to more external services to support more result types (e.g. map, movie reviews, source code, Q&A). Most code is written in Javascript and PHP to achieve portability and flexibility. It will make porting the system to other web authoring platforms easier later.

EVALUATION

We conducted a two-part evaluation to verify the design concepts and assess usability of the system. First, we crawled 72 blog posts and asked crowd workers to identify opportunities for bracket queries. We clustered tasks into 4 classes. Second, we invited 5 bloggers to the lab and observed them write a blog post using our tool. We collected qualitative feedback on usability and overall experience.

What do people expect from [[ ]]?

The goal of this evaluation was to clarify the types of tasks ReadWriter can support and to understand how the system can address the distinct properties of each task type.

We handpicked 72 blogs spanning diverse themes and popularity. We selected 8 blogs from each of the following 9 categories: entertainment, business, sports, politics, technology, science, lifestyle, small business, and personal. The categorization closely matches with that of major blog directories such as Technorati [20]. The list includes blogs from top-ranked blogs in Technorati, Blog of the Day [23] from wordpress.com, Blogger's Blog of Note [5], and a random selection of small and personal blogs. We picked one most recent entry from each blog. We then asked each Turker to identify three possible `[[ ]]` tasks after reading one of the 72 posts in the corpus.

Using 216 instances of `[[ ]]` queries collected, we identified 4 task classes: Search delegation, Writing help, Fact verification, and Formatting. Two researchers made subjective decisions for each task until agreement. There were 13 that are not included in any of the 4 classes. This set consists of outliers and spam.

In Search delegation (108 tasks, 50%), Turkers expect the system to find relevant images (40 tasks), links (37 tasks), facts (18 tasks), video clips (10 tasks), or synonyms (3 tasks). The result verifies that the four result types in ReadWriter, namely image, link, video, and text, cover most search requests.

While a majority of queries are straightforward, there are a few complex examples such as `[[Find images of her other tattoos, so reader can judge if they are getting any better.]]` With specific natural language requests embedded, simply passing the query to the search engine might fail. These kinds of queries might benefit human intelligence. Fact search queries are normally more challenging for AI than others because they often require more context and retrieval techniques. Some examples include `[[find scripture verses that support the persona of the Jesus of the Bible]]` and `[[Add latest details of this product]]`.

The interpreter in ReadWriter successfully identifies all result types in this category. ReadWriter's current policy of assigning fact searches to both Turkers and automatic agents is a reasonable heuristic, although contextual text analysis techniques can potentially improve the precision.

Writing help (61 tasks, 31.5%) refers to various levels of help requests on writing. It ranges from spell check and word choice questions to proofreading sentences and paragraphs, and to even higher-level requests such as `[[organize the post in a refreshing way]]`. ReadWriter assigns most (60/61=98.3%) of the tasks to Turkers, and one request, `[[give the photos caption]]`, to an image search agent. in the absence of keywords indicating human contributors inside a request.

Fact verification (23 tasks, 10.6%) tasks request the confirmation of a stated fact. These tasks normally include terms *check* or *verify*. Some examples include `[[check the authenticity of references]]` or `[[verify dates]]`. These tasks normally require human workers, and ReadWriter assign all 23 tasks to human contributors.

Finally, Formatting (5 tasks, 2.3%) tasks ask for fixing the style inside a document, as in `[[re-format text around google ad]]`. The current ReadWriter system cannot handle these requests because it is not connected to styling functions inside an editor.

We argue that although ReadWriter can misinterpret directions, the system *fails soft* because users can easily add additional workers in the management interface. The cost of omission is not high. The over-assignment case also deploys a fail-soft technique by requiring users to always confirm before sending out task requests to any human contributors, preventing errant payments or social requests.

The task data from Turkers provides insights on how users perceive the writing assistant system, and what kind of tasks they expect the system to handle. A limitation of this approach is that Turkers' reactions do not reflect content authors' views. Especially for high-level requests such as feedback requests, one can only speculate about the original intent of the author. This data also omits any information about the user experience of the system; we discuss the user experience below.

Blogging with ReadWriter

We designed a user study to observe bloggers in an end-to-end blogging task, evaluate usefulness and usability of the interface, and collect qualitative feedback. We invited 5 bloggers to the lab (1 writing instructor and 4 graduate students in engineering and science). The session started with interviews on their current blogging practice. The discussion included blogging frequency and time commitment, topic and identity management, Twitter/Facebook integration, and communication with readers. Then we explained the goal of research and gave them a simple ReadWriter tutorial. The main task was to write any blog post she might write in their blogs using ReadWriter. In case that she needed inspiration for topics, we verbally listed some keywords (e.g. movie, music, research, class, etc.). We requested posts to be more than 2 paragraphs to assure enough usage experience. Think aloud was encouraged, although we did not remind them while they were writing to avoid disrupting their concentration. After the writing task, a discussion addressed the following questions.

User	# Tasks	# Search	# Success	# Failure	Word count
P1	7	6	4	2	171
P2	5	1	1	1	104
P3	9	6	5	4	488
P4	6	2	2	2	233
P5	6	3	4	1	93
Avg	6.4	3.6 (56%)	3.2 (50%)	2 (31%)	217.8 (34/task)

Table 2. The bloggers' post statistics.

-How difficult was it to learn to use ReadWriter?
-When did you find ReadWriter useful / frustrating?
-What kind of questions to what kind of workers?
-How would you use this tool in your blogging practice?
-Any suggestions for interface fixes or additional features?

Blogging practice

The bloggers all had one or more active blogs, except for P4 who has temporarily paused blogging for 3 months. He stopped because of time commitment and pressure of public exposure. As a writer, P1 writes about writing and sometimes posts snippets of her work. For P5, his blog is a place for personal interests and thoughts, where he would like to keep a small number of dedicated readers and friends. The others primarily post research-related content. P3 and P4 both said they avoid expressing strong opinions and posting content they are not highly confident about.

Creating tasks

As shown in Table 2, out of 32 tasks created, 18 (56%) were search delegation queries (e.g. `[[image corn-fed midwest]]`). This proportion roughly matches with the categorization result (50%). There were feedback requests, as in `[[I actually haven't seen any movie by Mamoru Hosoda any comment will be welcome]]`. The tone in the two examples is clearly different: the former looks like a search engine query while the latter looks like a personal message to others. We found no case of proofreading or fact verification, possibly because the participants were not writing long, informational posts.

We count a request as success (50%) when the user accepts one of the results or verbally indicates that the goal was met (e.g. a result gives an answer, although the user does not apply it to the draft). Failure (31%) is as when the system fails to retrieve any satisfactory result or when the blogger states so. There are sometimes unclear cases (19%). An example is when the user indicated a feedback request but decided not to actually send it out during the study. These cases are considered neither success nor failure. Because of the latency in getting results from human contributors during sessions, the results in this section only report automatic search results.

We report a few notable usage patterns. P1 created a query just for fact searching, `[[poem of address]]`, not intending to include the result in the draft. Both P2 and P3 sought help on the title, as in `[[good title, read below]]` (P2) and `[[better title]]` (P3), which was not supported by the current ReadWriter version.

P2 had a unique composition strategy: his post consisted only of bracket requests. His expectation was to only focus on architectural aspects of writing while offloading the execution to crowd workers. In one request, he inserted `[[make paragraph: talk about trip to Scotland` in the beginning and appended 4 bullet points that outlined the potential paragraph. The others did not outsource any writing tasks.

Managing tasks
Four bloggers made a clear distinction between the writing and editing phases: they finished through their first draft with `[[ ]]` marks in places, and handled each item in the list. This supports our premise that people will delegate housekeeping until initial drafts. After the first iteration, the transition between writing and editing was less obvious.

Reviewing results
The current system does not perform any result verification. This led to a few failure cases, for instance, when a Turker returned garbage data for P3's request.

ReadWriter reduces the chance of serendipity because it only samples a small number of search results. P1 pointed out that an ideal tool will preserve both efficiency and discovery. Twidale et al. [21] describe serendipity in the web search process can encourage creativity and distract users at the same time. Future design iterations of ReadWriter will seek ways to achieve both goals.

Applying results
Inline updates for images and links worked without error in all 14 tasks. A missing feature in the current prototype is adding the body content of an external website. Extracting the content should depend on human intelligence. For example, P4 requested `[[google interview questions]]` to add a few examples of Google interview questions. The automatic agent related links with 2-3 lines of description, but no results contained any actual examples. A similar incident took place for P1 when she expected to add the full text of a short poem.

Post-session interview
The participants found the double bracket syntax to be *"easy to learn"* (P1) and *"simple"* (P2). All participants were able to create their own requests after being shown 2-3 examples. Response to usefulness differed among them. P2 does not use Twitter, Facebook, or email proactively, and he expected to only use Turkers and automatic agents. A lightweight mechanism for reaching out to different contributor groups might encourage bloggers to try out new groups. P5 said, *"I didn't think of Twitter as a place I can get help from. With the system, it might be nice to ask my followers to help find a photo I missed from Google search"*. All the participants expressed willingness to use ReadWriter in their actual blogging.

How do the bloggers feel about their writing experience with ReadWriter? P4 described his experience as "similar to managing a TODO list", and P3 referred to her process as "somewhere in between brain dump and real writing". P1 said, *"You might even write more. Maybe I'll blog more. Maybe it's good for my readers. Maybe it's good for my traffic. My posts will have more variety or something."*

One recurring theme was the need for keeping a group of dedicated readers for initial reviews. P1 sends out email drafts to intimate readers and P4 wants to have his research-related post *"peer-reviewed"* before it goes out to the public. P4 notes *"ReadWriter can help junior researcher and non-expert bloggers to have higher confidence."* This model of reader-writer interaction can lead to the 'beta' publishing model. In this model, a voluntary group of readers will have exclusive access to initial drafts of the blogger and offer feedback. Using a URL as a universal access mechanism for contributors, ReadWriter can easily support additional worker groups.

DISCUSSION

Personalized contexts
One of the most desired features from the bloggers was to support personalized contexts. The bloggers created queries such as `[[search my tweets for gender breakdown computer science]]` (P3) and `[[My picture at Google on 2010]]` (P4). Personalization can be a useful add-on to the system because it can widen the range of resources that contributors can use to aid writers. The system can adaptively improve its performance by learning from the writer's preference. One important discussion that should follow the personalized engine is privacy. Tool designers should carefully consider various related issues, in order not to compromise on privacy.

Another topic related to personalized contexts is notes to self. Some bloggers used the task list in ReadWriter as their writing TODO list. They created tasks only for themselves, not intended to be outsourced to any contributors (e.g. `[[TODO: remember questions..]]` from P4). The participants who generated self-tasks finished their first draft and referred to each item in the list to finish editing. ReadWriter offers the benefit of automatic list generation in addition to task outsourcing for these users. ReadWriter can help lift writers' cognitive overhead in managing various TODO items while writing.

Diverse work forces
The design decision of including multiple contributor groups was based on an assumption that each group exhibits its different cost structure, performance, and motivation. In an ideal system, the groups will complement each other, matching the nature of the task and the writer's intention. There are nontrivial technical and social challenges in realizing this vision.

Each work has different cost structure even when the work looks seemingly free: A search agent is fast and cheap but lacks precision and comprehension; A directed reader is trustworthy and at the cost of the writer's social capital. Between the two ends lie other worker groups, each with idiosyncratic properties. ReadWriter's goal is not to automatically minimize the cost for the writer, but to provide writers with an open platform so that they can make the best decision for their current writing task.

A new reader-writer interaction model

In this paper, we present a preliminary effort in bringing readers into the writer's writing process. This work explores only a portion of all the readers, i.e. readers from writer's social network. A promising future direction would be to reach out to anonymous and unexplored readers who can help the writer.

Leveraging the unexplored readers can allow more interactions between readers and writers, including polling, submitting short answers to the writer's questions, suggesting topics for next posts [7], marking unclear content, and adding useful information to the article. One potential enhancement to the ReadWriter system is to capture such interactions with existing `[[ ]]` syntax. For example,

- [[What gadgets will survive 10 years from now?]]
- What should I write about next? ipad[[vote]] food[[vote]]
- [[4pm on Friday, RSVP plz]] (from P5)

CONCLUSION AND FUTURE WORK

This paper presents new modes of task automation and feedback support for bloggers. ReadWriter allows natural language requests in double brackets, which can then be assigned to diverse contributor groups. We present task categorization results drawn from crowdsourced data, and quantitatively report an observational study of five bloggers using ReadWriter for a blogging task.

The next step in this research is to deploy the system to the public, and collect and analyze longitudinal usage data. This data can answer many questions: how bloggers rely on their readership and social network for receiving writing help; how ReadWriter influences bloggers' writing process in the long-term; what social conventions in reading and writing emerge from the interactions embedded in the tool.

The larger vision of this work is to bridge the gap between readers and writers. Motivating lurking readers to contribute to the writing process or published article is a challenging problem. We further hope to discover benefits in closing the feedback loop, which reflects back the writer's response to reader feedback. This effort will contribute to the study of web readability enhancement. Collaboration between researchers focusing on reading and writing will potentially impact the reader-writer eco-system.

REFERENCES

1. Babaian, T., Grosz, B.J., and Shieber, S.M. A writer's collaborative assistant. *IUI '02*, ACM Press (2002), 7.
2. Bernstein, M.S., Little, G., Miller, R.C., et al. Soylent : A Word Processor with a Crowd Inside. *UIST 2010*.
3. Bernstein, M.S., Marcus, A., Karger, D.R., and Miller, R.C. Enhancing directed content sharing on the web. *CHI '10*, (2010), 971.
4. Bernstein, M.S., Tan, D., Smith, G., Czerwinski, M., and Horvitz, E. Personalization via friendsourcing. *ACM Transactions on CHI 17*, 2 (2010), 1-28.
5. Blogs of Note. http://blogsofnote.blogspot.com/.
6. Budzik, J. and Hammond, K. Watson : Anticipating and Contextualizing Information Needs. *ASIS&T 1999*.
7. Dugan, C., Geyer, W., and Millen, D.R. Lessons Learned from Blog Muse : Audience-based Inspiration for Bloggers. *CHI2010*, (2010), 1965-1974.
8. Elbow, P. (1998). Writing with power: techniques for mastering the writing process. New York, NY: Oxford University Press.
9. Horowitz, D. and Kamvar, S.D. The anatomy of a large-scale social search engine. *WWW '10*, (2010), 431.
10. Hu, C., Bederson, B.B., and Resnik, P. Translation by Iterative Collaboration between Monolingual Users. *HCOMP'10*, (2010), 2-3.
11. Keh, C.L. Feedback in the writing process : a model and methods for implementation. *ELT Journal 4414*, October (1990), 294-304.
12. Kittur, A., Smus, B., and Kraut, R.E. CrowdForge : Crowdsourcing Complex Work. *Technical Report. CMU-HCII-11-100*, (2011).
13. Kriplean, T., Toomim, M., Morgan, J.T., Borning, A. and Ko, A.J. REFLECT : Supporting Active Listening and Grounding on the Web through Restatement. *CSCW 2011 Horizons*, (2011).
14. Kopytoff, V.G. Blogs Wane as the Young Drift to Sites Like Twitter. 2011. http://www.nytimes.com/2011/02/21/technology/internet/21blog.html.
15. Little, G. Exploring iterative and parallel human computation processes. *HCOMP'10*, (2010), 4309.
16. Little, G., Miller, R.C., Chou, V.N., Bernstein, M., and Cypher, A. Sloppy Programming. In E. A. Cypher, M. Dontcheva, T. Lau, and J. Nichols, ed., *No Code Required: Giving Users Tools to Transform the Web*. Elsevier, 2010.
17. Preece, J. and Shneiderman, B. The Reader-to-Leader Framework: Motivating Technology-Mediated Social Participation. *Transactions HCI 1*, 1 (2009), 13-32.
18. Rhodes, B.J. and Maes, P. Just-in-time info retrieval agents. *IBM Systems Journal 39*, 3 (2000), 685-704.
19. Rhodes, B.J. and Starner, T. Remembrance Agent : A continuously running automated information retrieval system. *PAAM*, (1996), 122-125.
20. Technorati. http://www.technorati.com/.
21. Twidale, M.B., Gruzd, A. a, and Nichols, D.M. Writing in the library: Exploring tighter integration of digital library use with the writing process. *Information Processing & Management 44*, 2 (2008), 558-580.
22. Wordpress Shortcode. http://codex.wordpress.org/Shortcode.
23. Wordpress Blog of the Day. http://botd.wordpress.com/top-posts/.
24. Zemanta. http://www.zemanta.com/