Cloudtop: A Platform for Cloud Workspaces

ABSTRACT
Due to the web's federated nature, users depend on the desktop to transfer data between sites. However, the desktop is centered around files, making it suboptimal for use with the web. We believe that in web-centric environments, users need new workspaces that enable a consistent interface for web data. As a step towards that goal, this paper proposes Cloudtop, a platform for building novel workspaces and expanding traditional drag-and-drop semantics for use in the cloud. Cloudtop introduces Webits, a data-type that promotes a uniform and extensible representation of web resources. Cloudtop uses plugins to extract semantic data from normal web pages to create Webits. Users drag and drop Webits between Cloudtop and the web to transfer data between sites. We believe that Cloudtop helps foster experimentation with workspaces that promote a more unified experience across a diverse web.

Author Keywords
Desktop, cloud, drag and drop, workspace, semantic web

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—*Graphical user interfaces.*

General Terms
Design

INTRODUCTION
As the use of traditional desktop applications continues to diminish in favor of web applications, the browser stands a good chance of someday replacing the desktop metaphor and becoming the de facto interface to the cloud. Projects like Google's Chrome OS [1], with its absence of a desktop, suggest such a future. However, a pure web environment has several drawbacks. While the web excels at offering tightly-tuned user interfaces for specific tasks, it does so at the expense of interface consistency across sites. Web service providers each promote their own unique models and interfaces for document management and sharing, requiring users to learn the intricacies for each service that they use. Furthermore, in a browser-only environment, there is no central workspace area to hold frequently accessed information, nor a common interface for transferring data between sites. Hence, user data tends to remain in silos.

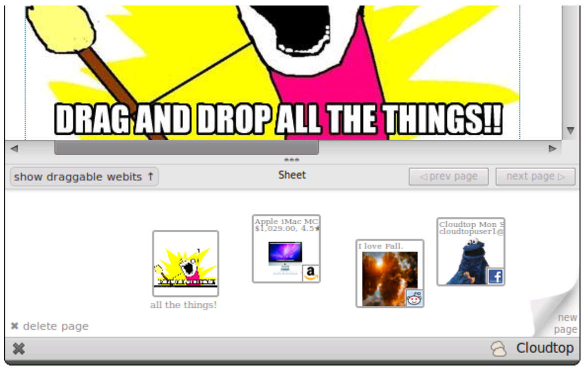


Figure 1. Cloudtop allows users to drag and drop Webits.

To address these problems, the successful desktop platform may offer some inspiration. For instance, the platform promotes interface consistency via common library code (e.g., file chooser dialogs typically look the same), while the central file system and clipboard ease data transfer between applications. The visible desktop area provides a common surface for quick access to in-use resources [14] regardless of type, as well as a place to store downloads from the web [20].

However, a naïve porting of desktop concepts for use in a browser-only world would miss opportunities for improvement. With workflows that span the traditional desktop and the cloud, users must switch between UI models (e.g., generic file systems and the web pages) and manage ancillary identifiers (e.g., temporary file names). Worse, the file system only knows about files and folders—but little about the web. Hence, storing web resources on the desktop typically incurs information loss: for instance, the desktop cannot tell the user on which pages downloaded resources were seen or any associated metadata (e.g., tag clouds, copyright, descriptions, etc) for downloaded files in general.

Our hypothesis is that in a web-centric environment, users need a new kind of workspace area, designed for the cloud, that 1) enables a simple, direct, and consistent interface for data transfer across the web, and 2) fosters new interactions not yet possible with the current web or desktop. For example, we imagine that such a workspace would allow a user shopping for a tablet device to drag products of interest directly from manufacturers' sites to their workspace as he shops, and then drag those products to an online spreadsheet to organize and compare various dimensions (e.g., screen resolution, price, storage size, etc). The user might also drag items to price comparison services which fetch and aggregate prices from vendors. The workspace may also capture representations of people, allowing a conference program committee chair to drag a paper submission into a group of peers on the workspace, and then drag that bundle into a management system to assign that paper to review to those people. The chair might also drag a group containing all program commit-

In submission to *CHI 2012*.

tee members to a web service that generates a list of names, profile photos, and affiliations for inclusion in the conference web site.

As a step towards that future, this paper proposes Cloudtop, a platform for building novel workspaces and expanding traditional drag-and-drop semantics for use in the cloud. In our prototype of Cloudtop, the platform provides a pluggable workspace area attached to the bottom of the browser window as shown in Figure 1. Users transfer resources to and from the web by directly dragging and dropping them between web pages and the workspace.

This paper's primary contributions are:

- a software platform for experimenting with workspaces that provide users with a central, consistent interface bridging data between web sites. Cloudtop is designed to support workspaces whose contents originate from and ultimately return to the web. Such workspaces 1) provide users with a local surface to collect web resources, and 2) empower users to seamlessly transfer resources across the web by direct-manipulation, without relying on web apps to build ad-hoc pathways between each other.

- an extensible, first-class data type, called a *Webit*, that provides a uniform representation of web resources. In Cloudtop, users interact directly with Webits, which are analogous to files on the traditional desktop, but unlike files, they 1) are pointers to resources on the web (but may cache data payloads), and 2) contain an extensible set of associated metadata for the resources they represent. While Webits can represent "primitive" resources on the web like photos or HTML pages, they are designed to proxy resources with rich metadata. For example, Webits can bundle metadata associated with people (e.g., name, email, address, photo), goods for sale (e.g., price, description, ratings), electronic documents (e.g., URI, authors, title), and so on. Webits, as opposed to simpler schemes like a single URL (e.g., in a browser bookmark, or in .webloc and .url files), enhance visibility of the bundled metadata and facilitate the capture of rich semantic objects rather than web pages.

Challenges and Claims

In the vision sketched above, one challenge is data interoperability between sites. Common data interpretation is the goal of the semantic web effort. By leveraging established ontologies and the Resource Description Framework (RDF) [12] for metadata representation, Cloudtop can focus on the user-facing aspects involved in exposing and transferring data between sites. Therefore, Cloudtop must provide a mechanism by which websites can produce and consume Webits (and their associated metadata) without causing undue development burden. Cloudtop uses the emerging HTML5 Drag and Drop standard [7] to make it simple to author web sites that incorporate Webits.

A second challenge arises due to the limited discoverability and visibility associated with traditional drag and drop: conventional interfaces do not indicate what elements are draggable and provide limited indication of what will happen when elements are dropped. In Cloudtop, those problems are amplified because 1) users now need to also distinguish conventional draggable elements (e.g., text snippets, images, or application-specific widgets) from Webits on pages, and 2) users also need to predict what will happen when dropping Webits into web pages, especially pages that do not natively understand Webits. For example, users might drag Webits from the Cloudtop workspace to conventional drop targets that do not interpret Webits, e.g., simple text input boxes, file upload zones, or white-space areas on web pages.

To improve discoverability of Webits on any given page, Cloudtop visually highlights the draggable Webits on command. To improve the predictability of drop behaviors, Cloudtop provides users with previews of the data that will be transferred while hovering over a drop target. This paper also proposes a set of guidelines that consistently map Webits to the associated data that is pasted (or transferred) when dropped onto traditional drop targets, allowing users to further predict how drops will work.

A third challenge is bootstrapping: without sites that produce Webits, or a way to inject Webits into the system, Cloudtop offers limited utility. Cloudtop addresses this by 1) automatically generating Webits for primitive resources (e.g., HTML snippets, links, images) when they are dragged into Cloudtop, and 2) providing a flexible plugin system, where plugin modules may create specialized Webits by modifying web pages loaded in the user's browser. For example, an amazon.com plugin, once installed in the user's Cloudtop, creates Webits containing metadata for products shown on amazon.com, ready for the user to drag into Cloudtop. In addition to creating Webits, plugin modules can also augment conventional drop targets on existing pages to recognize and consume Webits in specialized ways. Cloudtop's plugin system is versatile enough to successfully augment a variety of existing sites, as we will demonstrate.

Finally, another challenge is providing interface flexibility for Cloudtop workspaces without compromising consistency. Rather than enforce a particular workspace metaphor (e.g., hierarchical folders, tag-based collections, etc), Cloudtop instead remains agnostic and offers a pluggable workspace system to experiment with different metaphors. Cloudtop provides routines to create and manage common Webit features, and thus can enforce a measure of interface consistency, much like how a traditional GUI toolkits manage common dialog boxes across all applications. This paper presents one workspace we built, based on a scrapbook metaphor, called *Sheets*. The Sheets workspace explores an interface, at a simple extreme, that mitigates clutter via disposable notebook pages that hold transient Webits. With Sheets, users drag resources from the web into the current notebook sheet and arrange the resulting icons spatially, much like in the traditional desktop. When the current sheet becomes cluttered with icons, users simply "turn the page" to start a blank sheet (Figure 1, lower right corner). The workspace keeps all pages ordered by time, much like a physical notebook. The interface is inspired by projects in time-based desktop management (e.g., [22, 25]). We developed Sheets to evalu-

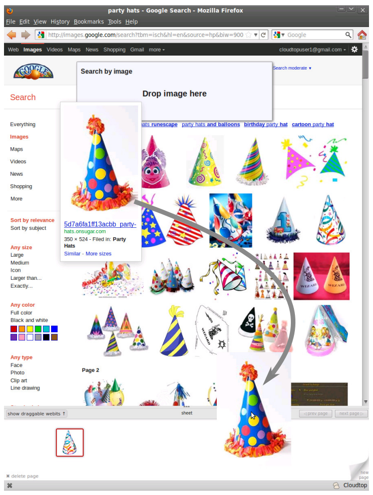


Figure 2. In Cloudtop, users drag and drop web items to create Webits.

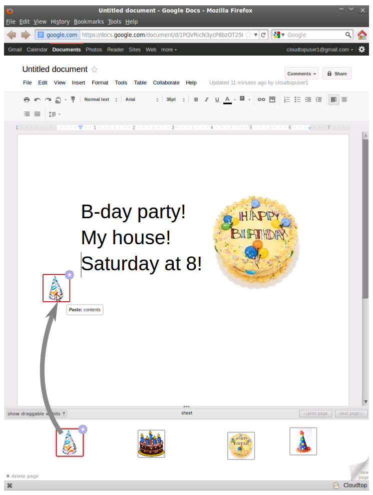


Figure 3. Users drag Webits from Cloudtop to transfer data back to the web.

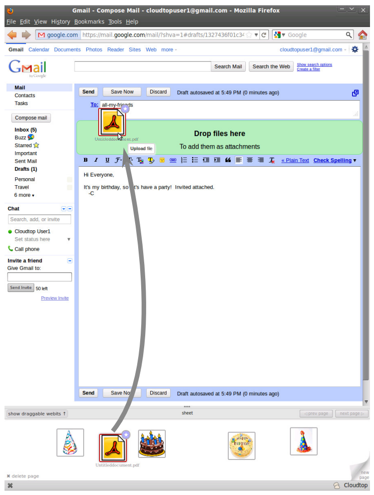


Figure 4. Cloudtop integrates with existing drop functionality in web apps.

ate the feasibility of Cloudtop's pluggable workspace system. Future work will focus on evaluating different cloud-aware workspace paradigms.

Outline

The next section overviews related work, followed by a section describing scenarios now possible with Cloudtop. We conducted an early, informal user study to obtain a sense of Cloudtop's potential usefulness; that study, described after, informs the current design of Cloudtop. Next, we highlight the platform's design and then discuss our experience developing on the platform. We then conclude with future work.

RELATED WORK

Several areas of previous work inspire Cloudtop.

Web Clipping

Many web clipping and note taking packages, such as Evernote [4], Microsoft OneNote [8], Zotero [13], list.it [27], and Clipmarks [2], aim to help users organize various snippets of text, images, and references found on the web (or elsewhere). The emphasis of these packages is to provide a digital home for scraps of information [15] as well as tools for their long-term organization and retrieval.

Cloudtop differs from these projects by focusing on the use of drag and drop as the primary mechanism to transfer resources across web sites. In other words, Cloudtop is designed to enable users to both drag resources into its workspace and also back to the web. Second, Webits are pointers with structured metadata: while Webits may represent primitive resources like text snippets and images, they can also proxy resources with richer associated semantics.

Desktop Studies

Sheets, the workspace UI implemented for Cloudtop, is also inspired by many studies on the traditional desktop. A sampling of these studies include Malone's in 1983 [23], Barreau and Nardi's in 1995 [14], Ravasio et al. in 2004 [24], and Katifori et al. in 2008 [20].

A few common themes emerge from these reports. Barreau and Nardi first observed that users place three types of information on the desktop: ephemeral, working, and archival. The studies tend to agree that the desktop is commonly used for temporary storage, such as a staging area for downloads or uploads, and as a device for reminding users of impending tasks. They also agree that archiving resources (e.g., to tidy the desktop) is difficult because doing so involves classifying each resource, a cognitively difficult task. As such, users tend to put off archiving, leading to desktop clutter. All studies agree that users naturally arrange desktop icons in meaningful, spatial arrangements, such as clustering similarly themed resources together, to aid efficient retrieval. These studies inform the design of Sheets, which borrows the spatial elements of the desktop, and alleviates clutter management with its disposable notebook page metaphor.

Web Scraping

Many systems scrape web data towards various goals. For example, Greasemonkey [6] enables users to install JavaScript scripts to alter specific web pages. Chickenfoot [16] enables end users to customize, modify, or automate existing web sites via an informal language based on keyword pattern matching, thus catering to users with less programming expertise. Cloudtop is similar to Greasemonkey in that it assumes plugin writers are generally expert programmers. However, Cloudtop differs from these systems through its additional support for plugins that 1) modify drag and drop behavior on existing pages and 2) scrape asynchronously loaded data.

Piggy Bank [19] scrapes pages open in the user's browser and generates structured data. However, Piggy Bank's aim is to enable users to subsequently browse and query that data within the browser via a Piggy Bank-generated web page. Cloudtop focuses less on allowing users to inspect structured data and more on the interactions for transferring structured data from site to site. Since Piggy Bank is also implemented as a browser extension, we imagine that it would be fruitful to leverage Piggy Bank as a plugin within Cloudtop.

Web Authorization Protocols

Cross-site authorization protocols, such as OAuth 2.0 [10], provide mechanisms to enable users to share data between sites. Such protocols rely on site operators to set up pathways between each other before users can transfer data; as such, inter-vendor pathways are likely to be limited in number and driven by business incentives. From the user's perspective, the use of these protocols suffer drawbacks in visibility. For example, when users are prompted to share their data with some external service, they must typically agree to share whole classes of the data (e.g., all their contacts, photos, emails) rather than specific items. Once they agree, there is usually little visibility to alert the user when data is transferred, as it occurs out-of-band, directly between the vendors' systems. In contrast, Cloudtop enables a vendor-neutral approach in which users have fine grained control of the data they wish to share through direct manipulation.

MOTIVATING SCENARIOS

This section portrays several scenarios in which users employ Cloudtop to gather information from the web for reuse with other web applications. The first scenario demonstrates the use of Cloudtop as a visual clipboard to transfer conventional resources (e.g., images and files) across the web. The second scenario depicts examples of plugins that create rich, draggable objects as Webits on existing web pages. Finally, the third scenario illustrates an example of a web site that natively consumes and generates Webits.

Scenario 1: Creating a Party Invitation

In this scenario, Bob creates a party invitation in a web-based word processor using various images found on the web. After crafting the flyer, he exports it to a PDF and emails it to friends. Using Cloudtop:

- Bob searches the web for art and drags images from a variety of sources, such as Google Image Search, Flickr, or Facebook, to Cloudtop (Figure 2). He also drags text snippets from relevant sites (e.g., a restaurant address) into Cloudtop as well. He optionally names the Webits after dragging them into Cloudtop to add notes.

- Once satisfied with his collection of art and text gathered within Cloudtop, Bob opens Google Docs to create the invitation and drags Webits to the document (Figure 3).

- Content with the flyer, Bob exports the invitation to PDF, which the browser downloads from Google Docs. Cloudtop intercepts the download and places a corresponding Webit within the Cloudtop pane.

- Finally, Bob opens a webmail client and drags the PDF from Cloudtop to the attachment input box to attach the flyer (Figure 4).

In workflows like the above scenario, Cloudtop facilitates collecting information without requiring the user to switch focus away from the browser or the current website. With a traditional desktop, icons of downloaded images may not be where users expect, or they may be mixed in with other existing but irrelevant icons. For non-file-based snippets (e.g., text clippings or URIs), users must interrupt their information gathering process to copy and paste data from the source to some placeholder, e.g., the target document or a scratch file.

Scenario 2: Organizing a Reading Group

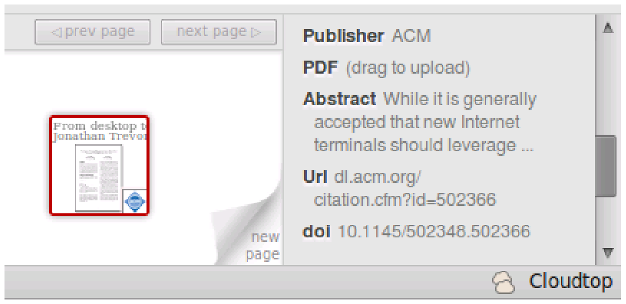
Sarah, a reading group organizer, selects a paper from the ACM Digital Library (DL) and e-mails a discussant to lead discussion on that paper. To obtain contact details, Sarah opens the discussant's Facebook profile page. Pages on Facebook may contain many draggable elements; she presses the Cloudtop "Exposé" button, as shown in Figure 7 to visually highlight the elements that have associated Webits. After finding the appropriate colleague, she drags in the picture of the discussant to Cloudtop, which creates a Webit containing various metadata about that person, e.g., his name, email address, homepage, and so on.

Next, Sarah opens the ACM DL page for the paper up for discussion to gather its bibliographic information, abstract, PDF, and so on. Since she has a Cloudtop ACM plugin installed, rather than drag various text snippets and the PDF, Sarah can drag just the image thumbnail, which creates a Webit containing all the metadata for that paper, including a cache of the PDF file. Figure 5 shows the resulting Webit and some of the metadata captured by the ACM plugin.

Sarah opens GMail to send the paper details to the discussant and other participants. GMail displays conventional text input boxes for specifying recipients, the message subject, and message body, all of which are drop targets for Webits.

Webits contain many metadata elements, only one of which is pasted by default. For example, dropping the ACM DL Webit into the rich text input box in the composition window pastes bibliographic information with appropriate hyperlinks.

Plugins may alter the default behavior associated with dropping Webits. For example, in this scenario, the GMail plugin for Cloudtop overrides the behavior when Sarah drops the Webit representing the discussant into the To, Cc, or Bcc field in GMail to paste the email address of the discussant rather



Publisher ACM
PDF (drag to upload)
Abstract While it is generally accepted that new Internet terminals should leverage ...
Url dl.acm.org/citation.cfm?id=502366
doi 10.1145/502348.502366

Figure 5. Cloudtop Webits capture metdata that the user may drag back to the web (overriding whatever may be the default for that Webit).

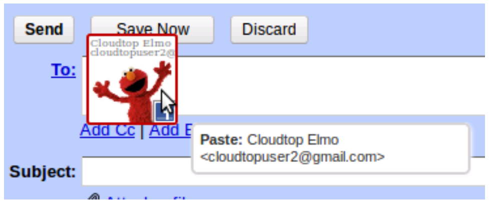


Figure 6. The GMail plugin overrides the tooltip of Webits representing people.

than the default text of his name. To help users predict such behavioral changes, the plugin overrides the default tooltip text shown to the user, as Figure 6 illustrates.

The user may also override the default drop behavior. For example, Sarah may want to paste the abstract for the paper in the message body. Cloudtop workspaces can enumerate associated metadata for a given Webit, e.g., via a panel (Figure 5); the user can browse and drag specific items from the panel to paste them. Here, Sarah drags the abstract entry in the panel to GMail's message body to paste the paper's abstract.

Scenario 3: Shopping Cart

Jim is shopping for camera equipment to start a new photography business with a business partner. As Jim shops various vendors, e.g., amazon.com and newegg.com, he drags Webits of products, produced either natively or with the help of Cloudtop plugins, from these vendors into his Cloudtop workspace. To share his selections with his partner, Jim uses a shared shopping cart service, which is independent of any vendor. The shopping cart natively understands Webits; when he drags Webits to the page, it displays and can sort relevant parameters for each product as shown in Figure 8.

When done browsing, Jim sends a link of his cart to his partner for review. Because the cart can natively inspect and manipulate its copy of the Webits, it can display the metadata associated with each, e.g. the product descriptions, directly on the page (Figure 9). The shopping cart can also serve (or regenerate) its Webits, so Jim's partner can drag the listed Webits from the cart to his own Cloudtop.

PRELIMINARY STUDY

We designed Cloudtop using an iterative process. First, we built a simple prototype called Vapor that only allows users to drag and drop primitive items. We then ran a pilot user study using Vapor to gain feedback on what kinds of interactions users expect when dragging and dropping web resources. Using what we learned in the pilot user study, we designed the Cloudtop platform to cover the various interactions users desire. This section describes the Vapor prototype and observations from the study.

Vapor Prototype and User Study

Vapor is drag-and-drop zone attached to the bottom of the browser. Users may drag in images, links, or text snippets from the web, access file downloads, and drag resources back to web sites. Vapor creates primitive Webits that only capture the item (e.g., the text, html, or file associated with an item) and two pieces of provenance metadata: the URL of item itself and the URL of the page the item came from. Vapor is not extensible and can only handle normal web items like images, links, text snippets, or downloaded files.

We conducted an informal pilot study, consisting of seven volunteers within a university computer science laboratory, to get a general sense of how users may use Vapor in typical workflows. After demonstrating features of Vapor in a brief tutorial, we asked each subject to carry out the "party flyer" scenario and an early version of the "reading group" scenario using Vapor and observed their usage. Vapor's primitive Webits do not capture metadata, so compared to the Cloudtop reading group scenario in the previous section, the Vapor scenario required users to manually drag metadata like the paper title, PDF, and abstract into Vapor.

Overall Feedback

Every subject successfully completed the tasks without material intervention or help. In their feedback, participants believed that Vapor would work well for their daily workflows, especially tasks that involved gathering resources first, followed by an aggregation or synthesis process.

We observed that users expect drag and drop operations to be "instant". Without adequate feedback (e.g., the status of the file upload process when users dragged an image into Google Docs), some users were confused when nothing appeared to happen immediately. This observation hinted that general visibility was an important requirement to address.

While few subjects employed the provenance information captured in Webits, some thought that it might be useful for tasks related to blogging or note-taking. Subjects could imagine using Vapor to collect interesting clips from the web, before pasting them and associated attribution information into other personal organization, blogging, or note-taking tools.

Copy and Paste

In the reading group scenario, many users started by using the system clipboard to copy and paste titles and abstracts. Ultimately, subjects who initially used the clipboard realized that they could use Vapor to gather all the information first and proceeded to do so, noting that Vapor alleviated the need to repeatedly context-switch between browser tabs as necessitated by a clipboard-based workflow.

The initial behavior of our subjects suggests that using the operating system's clipboard (e.g., the Ctrl-C keyboard shortcut) remains an ingrained approach for transferring *textual* information between places. However, we observed during pre-task interviews that subjects do not have a clear model of what kinds of data, beyond text, the clipboard can hold, especially when data crossed the browser-desktop boundary. Subjects noted that they especially liked the visual and visible nature of Vapor, and compared it to a more powerful cross between a desktop and clipboard.

Nearly all subjects who initially employed Ctrl-C thought it would be a good idea for Vapor to intercept copy commands and automatically create an associated Webit. One subject

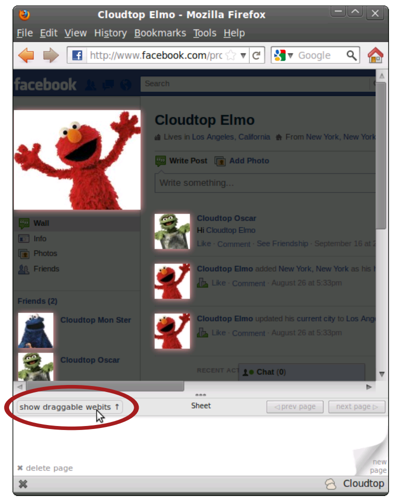


Figure 7. The Cloudtop “Exposé” button allows users to discover which page elements have extra Cloudtop information attached to them.

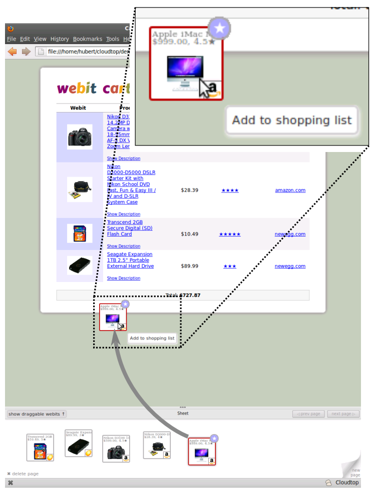


Figure 8. The “Webit Cart” understands Webits natively, allowing the page to extract information from Webits, as well as change Cloudtop tooltips (inset).

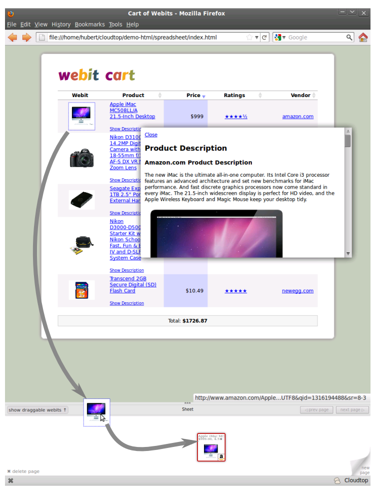


Figure 9. Since the web page understands Webits natively, it can display information embedded in the Webits and let users drag them back to Cloudtop.

disagreed with the usefulness of doing so, citing that the clipboard should remain a “light-weight” mechanism and be kept separate from Webits.

Vapor UI

All subjects approved of the fact that Vapor was part of the browser chrome, instead of being in the area behind the browser (like the traditional desktop), with some citing quicker access and constant visibility. Subjects resized the visible area of Vapor to make it larger or smaller. Some suggested that they might keep a separate browser window open, dedicated to displaying Vapor full-screen, while others preferred Vapor to be more “integrated” with and customized to the current, active tab.

Vapor Storage

With Vapor, subjects seemed pleased to be able to live completely in the web (save for certain desktop applications that they still rely upon). However, some asked where Webits are stored and seemed concerned that they might fill their local hard disks with unneeded temporary data. All cited that an ideal solution would be to have some form of universally accessible cloud storage to back the contents of Vapor.

DESIGN RATIONALE

Cloudtop is composed of three components: the core platform, content plugins, and the workspace (Figure 10). The core provides common library code, notably for data storage, and coordinates data flow between plugins and the workspace. Plugins (e.g., for ACM DL, Facebook, Amazon.com) capture metadata from resources imported into Cloudtop, and may modify and scrape web pages open in the browser to facilitate interactions with Webits. Any number of plugins may be in use at any given time. Workspaces (e.g., Sheets) implement the user-facing interface and are pluggable. Only one workspace may be active at a time. The core is the only fixed, non-customizable component of Cloudtop but is relatively small; thus, workspaces and plugins may evolve independently.

The Cloudtop core reacts to two main types of events: browser page load events and user import events. When a page loads, the browser sends Cloudtop a page load event which causes the core platform to call the onPageLoad(document) function of each relevant plugin. The plugin’s onPageLoad implementation modifies the page to facilitate drag and drop and information scraping. onPageLoad may optionally return a context object which is used to enable scraping of asynchronously loaded content.

Users import data into Cloudtop by dragging it in from the web, copying it to the clipboard, or downloading it. When users import resources into Cloudtop, the browser sends an event to Cloudtop, which forwards the event to the workspace, allowing the workspace to either handle or filter the event. If the workspace accepts the import, it asks the core to create a Webit from the event. In response, the core allocates storage for the new Webit object, assigns the Webit an identifier, and calls the appropriate import function on all relevant plugins. For example, for dropped resources, the core calls the onDrop(objectId, event, context) function on each plugin, passing in the Webit’s identifier, the drop event, and any context the plugin returned in its onPageLoad function.

Imports via the clipboard or the browser’s download manager work analogously by catching the appropriate browser events and calling onClipboardCopy or onDownload, respectively, on relevant plugins. Workspaces ultimately decide how to represent Webits imported via clipboard copy or download, if at all. Sheets, our prototype workspace, places such Webits on the visible workspace, much like how the browser places downloaded files on the traditional desktop. Alternative workspaces could provide options to enable or disable

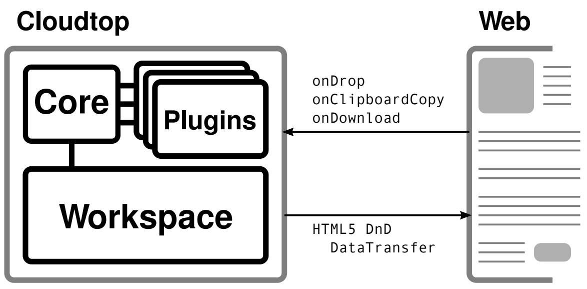


Figure 10. Cloudtop components and dataflow between the platform and the web.

that behavior or employ a separate area to display Webits imported by clipboard or download.

Each plugin import function returns a WorkspacePackage, a data structure containing information for the workspace to display and handle common interactions with the Webit. The core collects and returns the list of WorkspacePackages to the workspace, which is responsible for preparing Webits for display and attaching the HTML5 Drag and Drop API data structures necessary for them to be dropped into various targets on the web.

The remainder of this section elaborates Cloudtop's use of the HTML5 Drag and Drop API, the core platform, the plugins architecture, and the WorkspacePackage.

HTML5 Drag and Drop

The primary way in which Cloudtop interacts with web pages is through the HTML5 Drag and Drop API [7]. The API specifies a set of events that allow JavaScript to track the progress of a drag (using dragstart, dragenter, and dragleave events) as well as register callbacks for handling drops (drop events).

All drag and drop events carry a DataTransfer structure that contains a list of MIME-typed representations for that item. For example, the DataTransfer for a snippet of text holds a text/html representation containing an HTML string as well as a text/plain one containing the string without markup.

Web pages may add their own custom MIME types to the DataTransfer. During the lifetime of a single drag and drop gesture, all associated drag and drop events share the same DataTransfer object, allowing handlers fired on drag events to populate the DataTransfer with data that is consumed by drop handlers. In the context of Cloudtop, when a page loads, a plugin's onPageLoad function adds drag event handlers to draggable elements. These handlers augment the DataTransfer object with new MIME types; those MIME types are then interpreted by the plugin's onDrop handler when a Webit is dropped into Cloudtop.

The browser implements a default set of actions when dropping generic resources on standard drop targets, like text boxes (which reads the text/plain DataTransfer data item), rich text editing areas (text/html if available, otherwise text/plain), and file input dialog boxes (browser-specific MIME types). As such, to allow users to drag Webits to conventional drop targets, Webits must have (1) an encoding using the standard MIME types [9] and/or (2) custom drop handlers that can accept and parse Webits directly. In Cloudtop, plugins must provide standard MIME type representations for the Webits it creates, and may optionally augment conventional drop targets with specialized handlers.

For drop targets that do not have specialized handlers that natively understand Webits, Cloudtop enforces a consistent mapping to promote predictability between what data is pasted when dropping Webits onto conventional drop targets. While other mappings are possible, the mapping with which we have experimented is as follows:

- Dropping a Webit onto an input box that supports rich text pastes a rich text summary of the Webit. For example, dropping a Webit representing a product for sale into Google Docs pastes a summary combining the product title, cost, ratings, and description. Dropping a Webit representing a person pastes the person's name, hyperlinked email, and profile image.
- Dropping a Webit onto a simple text input box pastes a short textual summary. For instance, dropping a Webit representing a product or person pastes the name of the product or person, respectively.
- Dropping a Webit onto white space opens a primary URI associated with the Webit in the browser. For example, dropping a product Webit into the browser opens the product page, while dropping a person Webit opens his homepage.

Core Platform

The core platform offers library routines, manages the dataflow between plugins and workspaces, and handles the persistence of Webits. In Cloudtop, Webits may be atomic or collections of other Webits. Collection Webits are simply Webits that hold references to other Webits, collection or atomic, and Webits may belong to multiple collections. Support for collections in Cloudtop enables different resource organization models in workspaces, from traditional, single-parent hierarchy (e.g., file system folders) to tag-based groups. The core storage schema manages this bookkeeping and provides routines for altering Webit membership. The storage schema is deliberately confined, as each plugin and workspace is expected to manage its own storage schemas for plugin- or workspace-specific data, e.g., to persist the specific metadata of a Webit or the coordinates of its current position on the workspace.

Plugins

Plugins fulfill two roles: 1) to augment pages when those pages are loaded (e.g., to implement specialized behavior for Webits dropped onto targets) and 2) to generate metadata for Webits, typically by scraping. Plugins declare regular expressions on URIs to indicate the web pages that they are able to handle.

Augmenting elements on pages requires that plugins obtain a handle to those elements it wishes to augment, and thus must wait for those elements to load. Augmenting static web pages, in which all elements are loaded before the DOM load event fires, is straightforward: plugins operate on the page in their onPageLoad function and use the XPath or DOM API to obtain handles to the appropriate elements. However, many sites load content asynchronously or in response to user action. As an extreme example, when loading certain GMail or Facebook pages, the browser may execute JavaScript that programmatically builds the document asynchronously. In such cases, the browser fires the load event once the bare DOM loads, prompting the onPageLoad functions to execute, even though the user-facing page has not fully loaded yet.

Cloudtop provides hooks to handle changes to document title or location URI (through the onTitleChange and onLocationChange plugin functions), as web pages that lazily construct whole documents typically change the document title and add fragment identifiers to the current location URI once page construction completes. One caveat is that plugins need to gracefully handle spurious title changes. When this technique fails, e.g. due to pages that do not modify the document title or URI, plugins can handle asynchronous resource loading by simply polling.

Plugins can scrape data on web pages lazily, i.e., at any time between when pages load and when users drop resources into Cloudtop. This is challenging to do without platform support because the HTML5 drop event does not provide enough information to plugins, e.g., the document to scrape or the DOM element the user dropped. To work around this limitation, Cloudtop provides an "element locker" that allows plugins to retrieve the DOM object of the specific element that the user dragged. The locker is a map between local identifiers and cached DOM element objects. To use the locker, a plugin returns a context object from its onPageLoad function. In response, when users drag an element on a page, Cloudtop assigns that element an identifier and puts that identifier and context object in the locker. When the drop completes, Cloudtop finds the identifier associated with the dropped element, dereferences the identifier in the locker, and inserts the associated DOM element into the plugin's context object, which it then passes to the plugin's onDrop function. The plugin can access the entire DOM via the document object referenced in the element object. As an optimization, Cloudtop only activates the element locker if plugins request it since (1) tracking elements slows down the browser and (2) the element locker complicates the browser's garbage collection system by keeping copies of elements around longer than would normally be necessary.

Interfacing Plugins and Workspaces

Workspaces rely on the plugins to provide enough information, encoded in the WorkspacePackage data structure, to display and prepare Webits for dragging back to the web. Each WorkspacePackage for a given Webit contains: 1) a set of MIME type default renderings for that Webit, including rich and plain text, that Cloudtop maps to various conventional drop targets; 2) a URI or HTML snippet to display when users "open" the Webit (e.g., by double clicking on it); 3) metadata for the Webit encoded as RDF triples; and, 4) a set of UI hints (e.g., icon image, textual labels, etc). As an example, consider the WorkspacePackage for an amazon.com product Webit. Its default renderings include a detailed, rich description and a short abbreviated one, respectively mapped to rich and plain text input boxes when the Webit is dropped. The URI to display when users open the Webit is the product page on amazon.com. The RDF metadata encodes various attributes about that product (e.g., product title, price, ratings, and so on). Finally, the set of UI hints includes a URI to a thumbnail of the product, an array of text snippets describing the product (ordered from most important to least), and a "favicon", which the workspace may use as a "badge" on Webits.

Each plugin that successfully contributes to a Webit's metadata must produce a WorkspacePackage. The RDF metadata associated with each WorkspacePackage can be combined because RDF assertions accommodate namespaces, and some workspaces may be able to leverage multiple sets of UI hints. However, some elements in the WorkspacePackage are mutually exclusive, such as the MIME types for default rendering.

The core platform merges the RDF metadata across all WorkspacePackages, ranks the WorkspacePackages, and passes them with the merged metadata to the workspace. The workspace which may elect to use only the highest ranking WorkspacePackage or implement a policy appropriate to its UI needs. Our current ranking approach requires each plugin to provide a regular expression that matches URIs of pages that it purports to handle. Plugins with expressions that do not match a given page URI are not invoked to handle that page, so they produce no WorkspacePackages. WorkspacePackages are ranked by how well their associated plugins match the URI of the page containing the resource dropped into Cloudtop. For example, a plugin that matches all sites (.*) is less specific and thus ranks lower than a plugin that matches all of amazon.com (amazon.com/.*), which ranks lower than a plugin that only handles amazon.com product pages (amazon.com/gp/product/.*).

While there are no doubt more sophisticated schemes, we believe that this approach is reasonably effective when users install two different classes of plugins: those that only run on specific sites and are non-overlapping, and those that run on every site (e.g., to capture common attributes). While Cloudtop could have enforced a two-class scheme to eliminate the need for ranking, we believe this would be too restrictive.

Webits

Workspaces are entrusted with attaching the necessary DataTransfer structure to Webits so users may drag and drop those Webits onto drop targets. Workspaces attach the combined RDF metadata, encoded in RDF/JSON [11], along with elements of the WorkspacePackages, into a Cloudtop MIME type in the DataTransfer. Sites, either natively or with the help of plugins that augment them, can detect when Webits are dropped via the existence of the Cloudtop MIME type and parse the data. Sites can also natively create Webits for consumption by adding a Cloudtop MIME type, with the associated payload, to a draggable element's DataTransfer. By encoding URIs in the metadata, Webits enable a dataflow in which sites that consume Webits may contact, out-of-band, the sites that originally created those Webits.

We considered simpler schemes for representing Webits, such as a single URI. For objects with rich metadata and semantics, this might work if those objects publish machine-readable descriptions (e.g., as RDF) online. However, many resources do not currently have such descriptions, so representing them as a single URI to web pages (e.g., that describe those resources for human readers) suffers several drawbacks. First, some web pages may describe multiple semantic objects and there may not be obvious ways to disambiguate and identify a specific one. Second, using URIs shifts the scraping burden onto the target sites that receive them. Sites would have to potentially know how to scrape every other site, inviting further inconsistency in how data is collected and interpreted.

EXPERIENCES WITH CLOUDTOP

We implemented Cloudtop as a cross-platform browser extension for Mozilla Firefox 6.0. As a browser extension, Cloudtop can thus 1) appear as a central component available in all browser tabs and windows and 2) intercept user interactions with web page elements. We evaluate Cloudtop's flexibility by reporting on our experience developing plugins to create Webits on various sites and the Sheets workspace. Table 1 lists the plugins we wrote.

For storage, the current prototype uses Firefox's SQLite-based database to store Webit metadata and the user's local Firefox profile directory to store binary files. While using the local Profile directory means that Webits are only available on one computer, we hope to lift this restriction by storing Webits in cloud storage (e.g., Dropbox [3]).

Provenance Plugin

One way we evaluated Cloudtop's flexibility is by how much functionality is implementable as plugins rather than as built-in code to the core platform. One feature of Vapor, the initial prototype, is that it automatically captures and provides UI elements to access provenance of resources (e.g., the URI of the resource and the URI of the page containing that resource) dragged into the workspace. A plugin-based implementation of provenance capture, as opposed to being a fixed component in Cloudtop's core, enables the feature to evolve independently. Our plugin captures provenance by inspecting the dropped DOM element and the associated document object.

Tooltips Plugin

Another example of implementing "core" functionality in plugins is the tooltips feature that displays previews of pasted data. The tooltip plugin implements this feature by modifying web pages to add user-visible DOM nodes that function as tooltips. When the plugin is called to handle a new loaded web page, it augments various standard drop targets (e.g., HTML input boxes, text-areas, file upload zones) to display the appropriate tooltips when the user hovers over each respective target.

The tooltips plugin also demonstrates communication between plugins, as other plugins (and webpages) must be able

Plugin	LoC	Comments
facebook.js	448	Scrape. *Asynchronous*. Out of band fetch of email address.
amazon.js	414	Scrape.
acmdl.js	402	Scrape. *Asynchronous*. Out of band fetch to cache associated PDFs and EndNote.
newegg.js	366	Scrape.
provenance.js	356	Scrape.
reddit.js	344	Scrape. Save permalink of entries and out of band fetch of linked pictures.
tooltip.js	185	Augment (general tooltip implementation).
gmail.js	137	Augment (to specially handle Webits dropped in To/Cc/Bcc fields). *Asynchronous*.
gdocs.js	30	Augment (to override tooltips). *Asynchronous*.

Table 1. A sampling of plugins and associated lines of code. Entries labeled *Asynchronous* indicate sites that load resources asynchronously.

to override the default preview text. For example, when dragging Webits representing people over GMail's input boxes for specifying message recipients, the Cloudtop GMail plugin overrides the tooltip preview text to display the associated email addresses.

Out of band: Facebook, Reddit, & ACM Plugins

For additional flexibility, plugins may also obtain data out-of-band when it is not available on the current page. Cloudtop provides routines to make network requests in the background, as well as higher-level routines that load other webpages (in an invisible iframe) and return the associated DOM nodes. This enables plugins to follow links on the loaded page, invoke API calls on web services, or even load other pages for scraping.

For example, when users drop a person's thumbnail from Facebook into Cloudtop, the Facebook plugin must attempt to obtain that person's email address, even if it is not displayed on the current page. The Facebook plugin inspects the URI of any page associated with the person, extracts the person's Facebook numeric ID, and scrapes contact information from the associated profile page that it fetches out-of-band. Another example is the Reddit plugin, which targets the social news site; the plugin detects news items that are pictures and automatically fetches the high resolution images. The ACM Digital Library provides another example in which scraping bibliographic data is made trivial given a paper's BibTEX or EndNote file, but those formats are not initially visible on the loaded page. The plugin uses the paper's ID to construct a URI for the EndNote file and load it for parsing.

Asynchrony: GMail and Facebook Plugins

Several sites load resources asynchronously, e.g., GMail and Facebook. The plugins for those sites both use the onTitleChange hook to detect when the page contents change dynamically. onTitleChange prompts these plugins to begin inspecting pages to, e.g., find form elements and override their tooltips. As shown in Table 1, implementation effort is independent of asynchrony but instead is commensurate with the complexity of scraping, suggesting that the Cloudtop platform eases challenges associated with asynchronous resource loading.

Data Interoperability: GMail and Facebook Plugins

In the scenario described earlier, when the user drags a Webit representing a person from Facebook to a recipient field in GMail, the browser pastes the person's email address rather than the person's name (the default behavior). This is because the GMail plugin inspects Webits dropped into various fields and overrides what is pasted. Thus, the GMail plugin must be able to infer whether certain Webits represent people. To promote data interoperability, Facebook and GMail respectively produce and interpret metadata adhering to the Friend-of-a-Friend [5] specification, which defines a standard vocabulary for describing people. In addition, the Facebook plugin adds an additional MIME type, application/rdf+xml, to the DataTransfer of Webits it produces. Thus, sites designed to accept RDF will be able to natively interpret the associated metadata in those Webits.

Sheets: Workspace Plugin for Webits

User-facing workspaces are pluggable in Cloudtop, and we designed a simple example workspace, called Sheets. Sheets is a workspace plugin UI with a chronological notebook (or scrapbook) metaphor, sans collections or hierarchy. Sheets is straightforward to implement as a UI for Cloudtop, as workspaces have complete control over the display canvas.

CONCLUSION AND FUTURE WORK

This paper introduces the Cloudtop platform and focuses on its use to enable drag and drop as the primary interaction for transferring rich resources between sites. Cloudtop features pluggable workspaces and plugins for creating and interpreting Webits on web pages. We imagine several avenues for future work with plugins, workspaces, and the core platform.

One potential workflow involves sharing Webits, e.g., via email or instant message. A Cloudtop plugin could attach a file containing serialized Webits to a message in a web-based mail client, and on the recipient end, the plugin could detect such attachments and create the necessary Webits.

We believe that Cloudtop is an apt platform to explore other workspace models, such as ones with organizational support for housing important Webits. For example, with Cloudtop, we imagine it possible to adapt conventional interfaces, like the traditional desktop area, file system model, or even command line interface, for use with resources in the cloud. However, we suspect that the cloud demands alternative workspace models, such as ones that promote sharing (e.g., [17, 26, 28]) or capture user context to aid resource finding (e.g., [18, 21]).

While Webits can be viewed roughly as files for the web, they are fundamentally different in that files contain actual payload, whereas Webits are essentially pointers with caching. Webits do not update their caches, and thus serve as one-time snapshots for the resources they represent. This may be immaterial for certain workspaces that are designed to house resources en-route between web sites. However, for workspaces that provide a long-term place for data, we expect that Cloudtop will need the facility to update Webit pointers. We did not implement the ability for users to change the metadata values associated with Webits, but we imagine that it would be straightforward to do so. Cloudtop would route the change request to the appropriate plugins, which would update internal state. However, the platform would need to manage user-prompted metadata changes alongside automatic updates.

To conclude, we developed Cloudtop to help foster experimentation with cloud-aware workspaces and interactions that promote a consistent and visible experience across the web. We view Cloudtop as a step towards future web environments that not only offer greater interface uniformity throughout, but do so while remaining open, diverse, and specialized.

REFERENCES

1. Chromium OS. http://www.chromium.org/chromium-os.
2. Clipmarks :: Add-ons for firefox. https://addons.mozilla.org/en-US/firefox/addon/clipmarks/.
3. Dropbox. https://www.dropbox.com/.
4. Evernote. http://www.evernote.com/.
5. The friend of a friend project. http://www.foaf-project.org/.
6. Greasemonkey. https://addons.mozilla.org/addon/greasemonkey/.
7. HTML5 drag and drop. http://www.w3.org/TR/html5/dnd.html.
8. Microsoft OneNote 2010. http://office.microsoft.com/en-us/onenote/.
9. Mozilla development network: Recommended drag types. https://developer.mozilla.org/en/dragdrop/recommended_drag_types.
10. OAuth 2.0. http://oauth.net/2/.
11. RDF JSON. http://docs.api.talis.com/platform-api/output-types/rdf-json.
12. RDF/XML syntax specification (Revised). http://www.w3.org/TR/rdf-syntax-grammar/.
13. Zotero. http://www.zotero.org/.
14. Barreau, D., and Nardi, B. A. Finding and reminding: file organization from the desktop. *ACM SIGCHI Bulletin* (July 1995).
15. Bernstein, M., et al. Information scraps: How and why information eludes our personal information management tools. *ACM TOIS* (2008).
16. Bolin, M., et al. Automation and customization of rendered web pages. In *ACM UIST* (2005).
17. Fass, A., Forlizzi, J., and Pausch, R. MessyDesk and MessyBoard. In *ACM DIS* (2002).
18. Freeman, E., and Gelernter, D. Lifestreams: a storage model for personal data. *ACM SIGMOD Record* (1996).
19. Huynh, D., Mazzocchi, S., and Karger, D. Piggy bank: Experience the semantic web inside your web browser. *Web Semantics: Science, Services and Agents on the World Wide Web* (Mar. 2007).
20. Katifori, A., et al. Evaluating the significance of the desktop area in everyday computer use. *ACHI* (2008).
21. Kleek, M. V., et al. Gui — phooey!: the case for text input. In *ACM UIST* (2007).
22. Lepouras, G., et al. Time2Hide: spatial searches and clutter alleviation for the desktop. In *ACM AVI* (2008).
23. Malone, T. W. How do people organize their desks?: Implications for the design of office information systems. *ACM TOIS* (1983).
24. Ravasio, P., Schär, S. G., and Krueger, H. In pursuit of desktop evolution: User problems and practices with modern desktop systems. *ACM TOCHI* (June 2004).
25. Rekimoto, J. Time-machine computing: a time-centric approach for the information environment. In *ACM UIST* (1999).
26. Sun, Y., and Greenberg, S. Places for lightweight group meetings. In *ACM GROUP* (2010).
27. Van Kleek, M. G., et al. Note to self: examining personal information keeping in a lightweight note-taking tool. In *ACM CHI* (2009).
28. Voida, S., et al. Share and share alike: exploring the user interface affordances of file sharing. In *ACM CHI* (2006).