Clui: A Platform for Handles to Rich Objects

Hubert Pham, Justin Mazzola Paluska, Robert C. Miller, and Steve Ward
MIT CSAIL
Cambridge, MA 02139
{hubert, jmp, rcm, ward}@mit.edu

ABSTRACT

On the desktop, users are accustomed to having visible handles to objects that they want to organize, share, or manipulate. Web applications today feature many classes of such objects, like flight itineraries, products for sale, people, recipes, and businesses, but there are no interoperable handles for high-level semantic objects that users can grab. This paper proposes Clui, a platform for exploring a new data type, called a Webit, that provides uniform handles to rich objects. Clui uses plugins to 1) create Webits on existing pages by extracting semantic data from those pages, and 2) augmenting existing sites with drag and drop targets that accept and interpret Webits. Users drag and drop Webits between sites to transfer data, auto-fill search forms, map associated locations, or share Webits with others. Clui enables experimentation with handles to semantic objects and the standards that underlie them.

Author Keywords

Cloud; handles; semantic web

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces - Graphical user interfaces.

INTRODUCTION

As web applications become more popular, it is important to consider the need for and design of visual handles to rich objects on the cloud. Sophisticated web applications today feature an expanding class of objects that users want to manipulate, like flight itineraries, financial accounts, real estate, restaurants, messages, products for sale, and people. While primitive data types on the web (e.g., text, images, links, and pages) might describe those objects, there are no visual handles to high-level, semantically-rich objects for users to grab. Such handles would enable users to efficiently organize, find, and share structured information. For example, a user looking for flights on AA.com could drag and drop a prospective itinerary to other sites to search for competing flights (e.g., on kayak.com), along with related services, like hotel rooms (e.g., on hotels.com) and car rentals (e.g., on alamo.com). The user could then share that handle to ensure friends are on the same itinerary.

UIST'12, October 7–10, 2012, Cambridge, Massachusetts, USA.

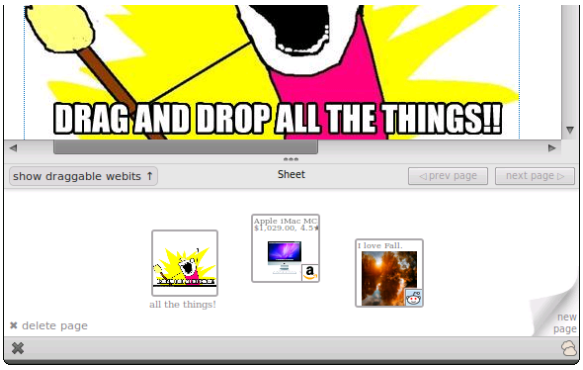


Figure 1. Clui appears as a pane in the browser and is shown here holding several Webits, visual handles to objects on the web.

Without standardized handles to rich data types, each application that requires them must implement its own handles, which invites inconsistency and poses challenges for interoperability across sites. For example, Google Docs features document handles in its document list, which differ from and are not interoperable with Microsoft Office 365's Sharepoint document handles. Similarly, representations of people on Facebook are different from those in GMail. The lack of a generic representation for rich data objects limits what data users may transfer between different web applications; hence, user data tends to remain in silos.

Our hypothesis is that in a web-centric environment, users need a new kind of handle that is generic and designed to represent an evolving set of semantically rich objects. Such handles would 1) enable a simple, direct, and consistent interface for data representation and transfer across the web, and 2) foster new interactions not yet possible with the current web or desktop. For example, handles would allow a user shopping for a tablet device to drag products of interest directly from retailers, like amazon.com or apple.com, to his 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. Handles could capture representations of people, allowing a conference program committee chair to drag a paper submission into a group of peers that come from GMail contacts, and then drag that bundle into a conference management system to assign that paper to review to those people. The chair might also drag a group containing all program committee members to a web service that generates a list of names, photos, and affiliations for inclusion in the conference web site.

To explore such interactions, this paper proposes Clui, a platform for experimenting with handles and the workspaces that

support them. Clui's primary contributions are:

- an extensible data type, called a *Webit*, that provides uniform handles to rich resources, and
- the Clui platform, a browser extension for experimenting with Webits and Webit-aware workspaces.

In a Clui-enabled system, users interact directly with Webits by dragging and dropping or copying and pasting them to transfer resources between sites. In contrast to typical web URIs, which point to resources, Webits contain an extensible set of associated metadata for the resources they represent. For example, Webits may bundle metadata associated with people (e.g., name, email, address, photo), goods for sale (e.g., price, description, ratings), electronic documents (e.g., authors, title, provenance), and so on.

Clui offers several mechanisms to inject Webit support into existing webpages: 1) by automatically generating them for primitive resources like HTML snippets, links, and images, 2) via a flexible plugin system to augment existing web pages to produce or accept Webits, and 3) through library functions that web developers call to add "native" support for Webits in their web applications.

As a platform, Clui enables experimentation with handles to semantic objects and the data transfer standards that underlie them. In order to test the feasibility and interoperability of Webits, we have built a variety of Clui plugins that augment existing webpages with the ability to generate and consume Webits. We have also built a Webit-aware workspace called Sheets, which appears as a pane below the browser tab, that provides users with a local surface to collect Webits, inspect them, and transfer them between sites (Figure 1).

This paper explores several claims about Webits and verifies those claims by illustration of workflows and examples. The summarized claims are: that Webits can represent a wide range of high-level concepts of interest to users; that they make interaction consistency across sites possible and ease common tasks; and that Webits, with the Clui platform, are easy to create and consume on existing web pages.

The next section is an overview of related work. Following that, we describe Webits in further detail and show various scenarios in which they are useful. We then describe an early, informal user study to obtain a sense of Clui's potential usefulness; that study informs the current design of Clui. 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 Clui.

Web Clipping & Scraping

Many web clipping and note taking packages, such as Evernote [3], Microsoft OneNote [9], Zotero [14], list.it [32], 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 [16] as well as tools for their long-term organization and retrieval. Clui differs from these projects by focusing on the use of handles to rich objects, rather than just primitive data types, and by focusing on information transfer instead of organization.

Clui is similar to projects that detect and create structured data in documents and clipboards, like Citrine [29], Apple Data Detectors [27], and Microsoft's Live Clipboard [7]. Clui differs by providing user-visible handles. Data sharing workflows in those projects, like Citrine's ability to paste structured data into forms and spreadsheets, inspire some of Clui's workflows.

Greasemonkey [5] enables users to install JavaScript scripts to alter specific web pages. Clui plugins are similar in that they assumes plugin writers to be expert programmers.

Projects that empower users to clip and scrape structured content from the web typically need to provide interfaces that semi-automate the process of scraping. For example, Dontcheva et al.'s work [18, 17] explores visual techniques that enable users to extract and relate structured content across different sites as a means to summarize and collect information. Fujima et al. [21] explore tools that enable users to construct new interfaces that bridge data between elements clipped from existing pages. Yahoo Pipes [13] provide a similar service that enables users to compose new data flows of content found across the web. Vegemite [25] describes techniques to 1) import web data into spreadsheets via direct manipulation and 2) generate scripts (by observing user action) to run on each row of the data table. d.mix [22] enables developers to obtain code snippets of functionality by clipping representative elements on a webpage. The visual approach of scraping-by-clipping, as proposed in these projects, would improve the development process for Clui plugin authors. The techniques for relating structured content across sites, as discussed in Dontcheva's work, would enable users (or ultimately the system) to merge related Webits found across the web. Techniques reported in Vegemite would be useful for automatically mapping Webit metadata fields to spreadsheet columns.

Piggy Bank [23] 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. Clui 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, it would be fruitful to leverage Piggy Bank as a library within Clui.

Microformat standards [8] enable site authors to embed semantic data within their pages. We imagine that Clui may someday automatically generate Webits on sites with microdata without requiring plugins for those pages.

Desktop Studies

A few common themes emerge from many studies on the traditional desktop, e.g., Malone's in 1983 [26], Barreau and Nardi's in 1995 [15], Ravasio et al. in 2004 [28], and Katifori et al. in 2008 [24]. Barreau and Nardi first observed that users

place three types of information on the desktop: ephemeral, working, and archival. The studies 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 suggested that Clui's initial workspace prototype should borrow from the spatial elements of the desktop.

While studies like these should certainly inform the design of Webit-aware workspaces, we believe that there are some notable differences between traditional desktops and workspaces for Webits. For example, in web-centric workflows (as envisioned by browser-only OSes, like Chromium OS [1]), data scraps not only come from the web but also ultimately return there (e.g., by posts to blogs, tweets, or services that curate and organize content). This suggests that local workspaces for Webits might need to prioritize temporary management rather than long-term retrieval.

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 suffers 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, Clui enables a vendor-neutral approach in which users have fine grained control of the data they wish to share through direct manipulation.

USER INTERFACE

This section overviews Webits, outlines the interaction model that they offer, and describes several scenario workflows.

A Webit is a user-manipulable handle to a collection of rich objects on the web. A Webit bundles properties about the objects that it represents. For example, Webits can provide handles to people, capturing names, contact information, addresses, and so on. Webits can also provide handles to places, like restaurants (bundling addresses, ratings, menus, hours of operation) or apartments for rent (bundling the monthly rent, address, photographs). A Webit may contain other Webits, so a flight itinerary Webit might contain a set of Webits to provide handles to passengers, the flight segments, and the origin and destination cities, while a Webit that represents a trip might embed flight itinerary Webits as well as Webits for hotel and car reservations. The set of objects that Webits may represent is extensible by leveraging existing semantic web technologies to express and interpret object metadata.

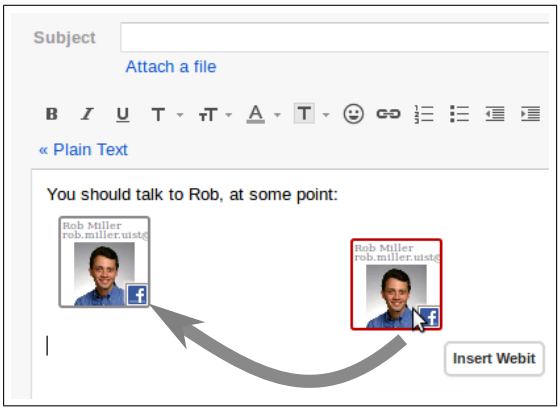
User Interaction Model

All Webit workflows in Clui involve three fundamental operations: discovering Webits on a web page, manipulating Webits in a local workspace, and transferring Webits back to the web. Webits appear as icons that, when clicked, reveal a panel containing metadata properties that describe the Webit. Users typically interact with Webits by dragging them from web pages into a temporary workspace and then dragging them to (other) web applications to share the information contained within the Webits. Workspaces are pluggable, and we designed a simple one, called Sheets. Sheets features a chronological notebook (or scrapbook) metaphor, without collections or hierarchy.

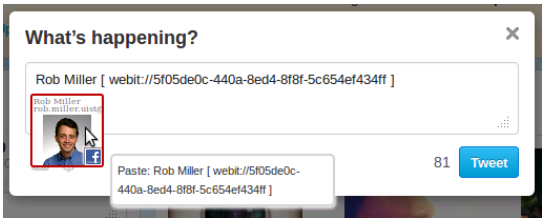
Users discover Webits in one of two ways. A web page may explicitly expose Webits to the user, e.g., as a Webit icon to be dragged. Alternatively, the web page may embed Webits inside other draggable elements, such as a picture of a product. To aid discoverability, Clui visually highlights available Webits on command. Clui also displays tooltips during drags to help the user predict the outcome of a drop.

To enhance the consistency of Webit actions across web sites, Clui provides three interface features:

Graceful Degradation Dragging a Webit to a web page results in the Webit being uploaded in its entirety, without loss of information. For example, when dragging Webits to rich-text input boxes, e.g. GMail's message composition window or a Google Docs document, Clui pastes the same iconic representation of the Webit the user dragged and embeds all of the associated metadata, so that it may be dragged by other users (e.g., when they see the Webit in a received email message).



When it is not possible to represent Webits as icons, e.g., when pasting a Webit into a plain-text input box, Clui pastes a short description of the Webit as well as a globally dereferenceable identifier for that Webit.



When the data is rendered later (e.g., in this example, as a tweet), Clui restores the iconic version of the Webit (Figure 4) by dereferencing the Webit's identifier.

Customizable Drop Behavior Clui enables web applications or plugins to modify the default drop behavior and provides hooks so that those apps may specify tooltips to describe the altered behavior. For example, when dragging a Webit representing a person onto the "To" field of an email composition form, Clui pastes the person's email address rather than the person's name.

Webit Metadata Transparency A Webit combines many attributes that describe the represented object. Users may override what is pasted by inspecting the Webit's metadata panel and dragging the desired piece of metadata. For example, the user may paste a friend's homepage URI by dragging the "Homepage" field in the metadata panel.

These three features play key roles in user workflows, shown in the example scenarios below.

Scenario 1: Finding An Apartment

Jack is looking for a new apartment and roommates. He searches sites like craigslist.org for potential leads. A plugin generates a Webit on each page that represents an apartment, capturing metadata like the cost of the monthly rent, the location of the apartment, the landlord's contact information, and the description of the property (Figure 2). To keep track of a promising apartment, he drags the apartment's Webit into his Clui workspace. After collecting a few candidate apartments, Jack drags the Webits to Google Maps, which, with the help

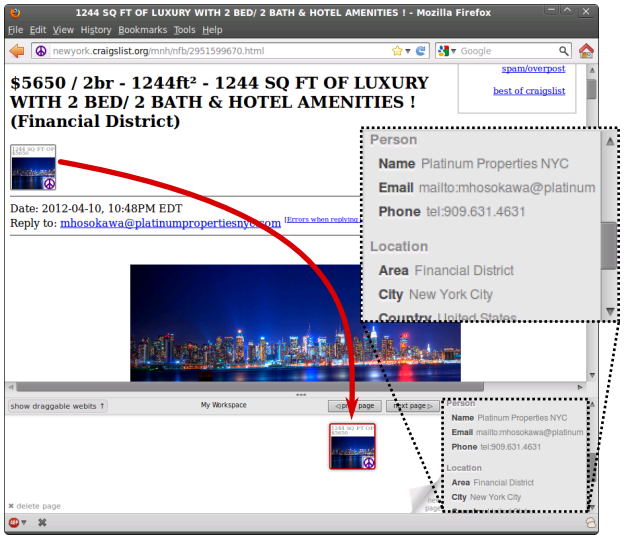


Figure 2. A Craigslist plugin generates an inline Webit to represent the apartment for rent. The Webit contains embedded Webits that represent the contact person and location of the apartment, as shown in the inset.

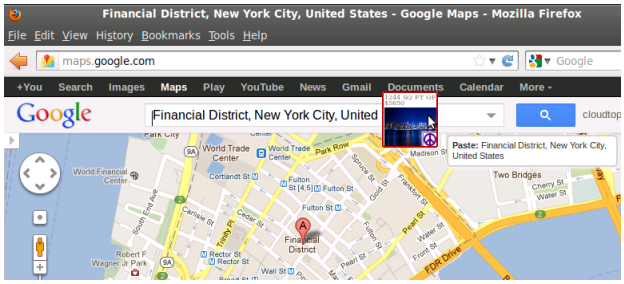


Figure 3. Dragging a Webit to Google Maps maps the location data present within that Webit.

of an installed Google Maps plugin that parses dropped Webits for locations to search, maps the locations of the apartments he likes (Figure 3). When he chooses the apartment in the best location, Jack composes an email to the landlord. Rather than typing the address, Jack drops the apartment's Webit onto the "To" field, which pastes the email address of the landlord. Finally, Jack advertises for roommates by sharing the Webit with his friends over email, Facebook, and Twitter. While the Twitter input box for tweets only accepts plain-text, Clui pastes enough context such that those who view the tweets will see an inline, draggable Webit (Figure 4), assuming they have Clui installed.

Scenario 2: Organizing a Reading Group

Sarah organizes a weekly reading group. She selects papers from the ACM Digital Library (DL) and emails a group member of her choosing to lead the discussion. To obtain contact details, Sarah opens the discussant's Facebook profile page. Pages on Facebook may contain many possible Webits embedded in existing elements; she presses the Clui "Show Webits" 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 Clui.

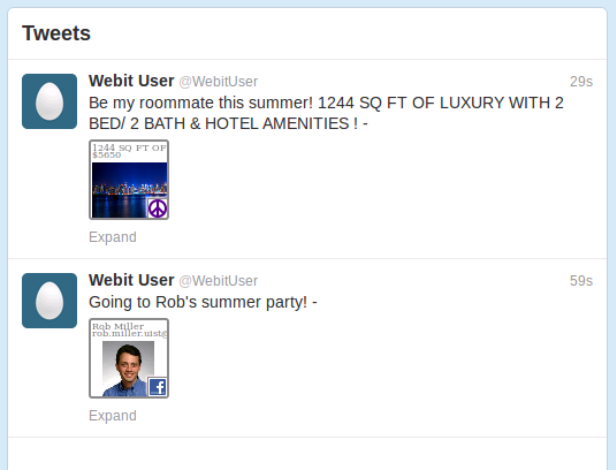


Figure 4. Even though some services, like Twitter, accept and store only plain-text input, they can still embed Webits with dereferenceable Web-it identifiers. Clui can later re-create interactive Webits in the viewer's browser.

Next, Sarah opens the ACM DL page for the paper up for discussion to gather its bibliographic information, abstract, PDF, and so on. Sarah drags just the image thumbnail for the paper, which represents a Webit containing all the metadata for that paper, including a cache of the PDF file. Figure 6 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. Sarah may drag the Webit that represents the paper directly into the message body to share the paper and all of its metadata. However, to provide context, she pastes a copy of the abstract by dragging the "Abstract" item in the metadata panel into the email.

In both of the scenarios above, Webits improve efficiency in several ways. Bundling relevant bits of information into one handle alleviates the need for users to manually parse, copy, and paste those bits from site to site. Sharing Webits keeps that information intact for the recipient. Plugins that customize the behavior of Webits when dropped in various contexts can scan for and paste the relevant data types, without requiring the user to hunt for that information.

Scenario 3: Booking a Trip

Mary is shopping for flights. She first visits AA.com and enters the origin and destination airports, along with the relevant dates. After perusing several options, she finds an itinerary that is potentially satisfactory, but she still wants to comparison shop in case there are cheaper alternatives. The AA.com plugin generates a Webit that represents her candidate itinerary, so before leaving AA.com, she drags that Webit to her Clui workspace.

Mary opens kayak.com to find alternative flights. Rather than re-enter the airports and dates, she drags the Webit from AA.com into the kayak.com form, which auto-fills the relevant fields with the help of a kayak.com Webit plugin (Figure 5). In contrast to browser auto-fill features that fill already-visited forms with values cached from previous submissions, Webits enable users to auto-fill forms with new values as well as auto-fill never-before-seen forms.

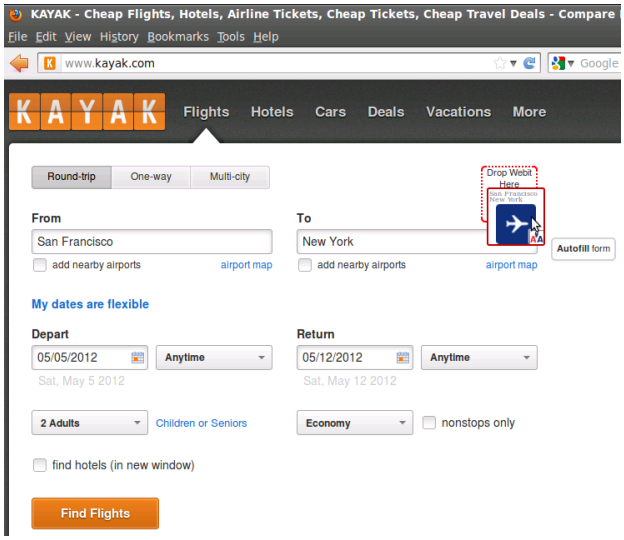


Figure 5. Dragging Webits onto a form auto-fills its inputs. Here, a Kayak.com plugin inspect the metadata in an itinerary Webit and maps it to form elements.

Scenario 4: Shopping

Previous scenarios illustrate the use of plugins to enable existing pages to create and process Webits. This scenario illustrates an example of a web application that natively consumes and generates Webits.

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 Clui plugins, from these vendors into his Clui 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, along with product descriptions, as shown in Figure 9.

When done, Jim clicks a button to generate a Webit that represents the cart, housing the individual Webits within. Jim shares that Webit with his partner, who may inspect and purchase the cart.

Figure 6. Webits capture metadata that may be dragged and dropped.

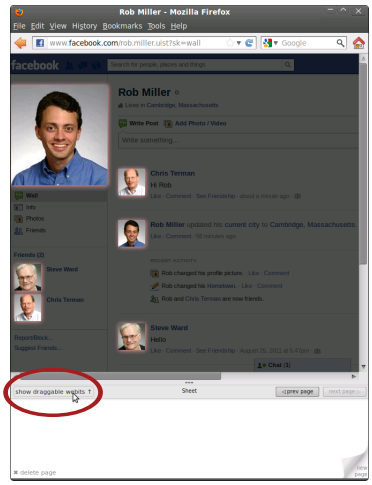


Figure 7. The "Show Webits" button helps users discover Webits that are embedded in existing page elements.

Figure 8. The "Webit Cart" understands Webits natively; thus, it can directly extract Webit data and change tooltips.

Figure 9. Since the "Webit Cart" understands Webits natively, it can also render information embedded in the Webits.

PRELIMINARY STUDY

We designed Clui using an iterative process. First, we built a simple prototype called Vapor that supports the drag and drop of only text, links, and images. We ran a pilot user study using Vapor to gain feedback on what kinds of interactions users expect when dragging and dropping web resources. We discovered that users of Vapor needed handles to rich data types rather than just primitive ones, informing our design of the Clui platform.

Vapor Prototype and User Study

Vapor displays a drag-and-drop zone like Sheets. Vapor creates primitive Webits that capture the resource (i.e., a text snippet, link, or image) and two pieces of provenance metadata: the URL of the item (for images and links) and the URL of the page containing the item.

We conducted an informal pilot study, consisting of seven volunteers within our 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 an early version of the "reading group" scenario using Vapor and observed their usage.

User Feedback

In the reading group task, participants needed to compose an email with paper abstracts and titles, gathered from non-adjacent text snippets on ACM Portal pages, along with the URI of the relevant Portal pages. Some participants immediately dragged abstract and title snippets from the page into Vapor, while others habitually relied on using the operating system clipboard, repeatedly switching back and forth between GMail and the Portal page. Ultimately, subjects who initially used the clipboard realized that they could use Vapor to gather all the information first and proceeded to do so.

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. Participants noted that Vapor alleviated the need to repeatedly context-switch between browser tabs, as a clipboard-based workflow necessitates. Subjects also noted that they especially liked the visible nature of Vapor, and likened it to a powerful cross between a desktop and clipboard.

All subjects noted that spatial element of Vapor was important, and often clustered related Webits into groups, reinforcing earlier findings [15, 28]. Many participants noted that they enjoyed the "freedom" that Vapor affords, especially for collecting and "quickly organizing" (spatially) different information scraps throughout a task. Subjects approved of the fact that Vapor was part of the browser, instead of being in the area behind the browser (like the traditional desktop), with some citing quicker access and constant visibility. However, subjects did resize 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 often needs to upload resources to the web, which introduces variable network delays into user operations. 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 a file into GMail), some users were confused when nothing appeared to happen immediately. This observation hinted that general visibility was an important requirement to address.

Design of Clui

Vapor's primitive Webits do not capture extensible metadata, so unlike the Clui reading group scenario in the previous section, in Vapor users must manually drag individual snippets of text. This observation suggested that users actually need more than just an easier way to collect web snippets; they also need a way to access and transfer objects with all of the associated

metadata bundled. The need to collect related information inspires the current design of Clui Webits as handles for rich semantic objects.

ARCHITECTURE

Clui is composed of three components: the core platform, plugins, and the workspace (Figure 10). The core provides common library code, notably for storage and rendering of Webits, and coordinates data flow between plugins and the workspace. Workspaces (e.g., Sheets) implement the user-facing interface and are pluggable. Only one workspace may be active at a time. There are two types of plugins. *Web plugins* add Webits to existing web pages by scraping those pages for metadata. They also augment drop targets on web pages with functionality to accept and parse dropped Webits. *Presentation plugins* translate classes of Webit metadata into user-friendly representations in the metadata panel. Any number of plugins may be in use at any given time. The core is the only fixed component of Clui, but it is relatively small. Workspaces and plugins may evolve independently.

The Clui core reacts to two main types of events: browser page load events and user import events. When a page loads, the browser sends Clui a page load event which causes the core platform to call the onPageLoad(document) function of each relevant Web 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 the Clui workspace by dragging it in from the web or copying it to the clipboard. Clui also creates Webits to represent downloaded files, which may be later dragged to the web to upload the associated files. When users import resources into Clui, the browser sends an event to the core platform, 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 Web 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.

The remainder of this section elaborates Clui's use of the HTML5 Drag and Drop API, the internal representation of Webits, the plugins architecture, and system limitations.

HTML5 Drag and Drop

The primary way in which Clui interacts with web pages is through the HTML5 Drag and Drop API [6]. The API specifies a set of events to track and handle drags (using dragstart, dragenter, and dragleave events) and drops (drop events). All drag and drop events carry a DataTransfer structure that contains a list of MIME-typed representations for the item being dragged. For example, the DataTransfer for a text snippet holds a text/html representation containing an HTML string as well as a text/plain one containing the string without markup.

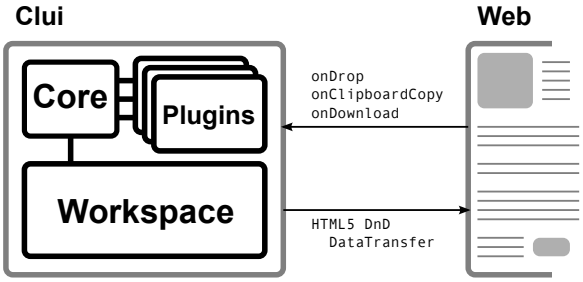


Figure 10. Components of and dataflow between Clui and the web.

Web pages and browser extensions 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 Clui, when a page loads, a Web plugin's onPageLoad function adds drag event handlers to draggable elements. These handlers augment the DataTransfer object with new Clui-specific MIME types that represent Webit metadata. On a drop into the Clui workspace, those additional MIME types are then interpreted by plugins' onDrop handlers to append the metadata to the Webit.

Webits

Clui represents a Webit as an immutable collection containing a Webit UUID, a representation of the Webit's user-facing icon, the Webit's metadata, and browser-specific metadata (like default DataTransfer MIME types to embed).

Webits leverage existing semantic web technologies, like established vocabularies, ontologies, and the Resource Description Framework (RDF) [12], to express and interpret metadata for the objects they represent. Vocabularies and ontologies are extensible, so Webits, in principle, can describe anything for which a vocabulary exists or can be defined. Webits embed metadata by encoding it as RDF/JSON [11], enabling sites receiving Webits to immediately interpret those statements without additional dereferencing.

A system like Clui only works when sites can interpret Webit metadata—that is, if sites understand a common vocabulary and associated semantics for a given resource. Establishing a standard for each resource type takes time and experimentation. However, Webits can embed multiple descriptions, using different vocabularies, each contributed by different plugins, thus increasing the chance that a website will understand one of the descriptions. In providing a user interface handle that abstracts away representations of semantic data, Webits offer an adoption path that enables vocabulary experimentation while supporting interoperability.

Clui provides JavaScript library support for 1) developers who wish to add native Webit support to their websites, and 2) plugin authors who wish to add functionality to websites that they do not control. There are two concrete representations of Webits: an internal JavaScript object representation, used within plugin and website JavaScript code, and an external JSON-serialized representation, used as the wire-format (embedded in the DataTransfer) for transferring Webits between

web sites and Clui. The internal JavaScript object representation gives developers full programmatic access to the Webit, including helper methods for extracting RDF statements and generating an iconic representation of the Webit.

Clui uses both the JSON-serialized Webit as well as the Webit identifier to reliably upload Webits to sites for sharing. In order to make the Webit identifier globally dereferencable, Clui uploads the JSON-serialized Webit to a cloud-based Webit sharing service; the sharing service identifies the Webit with a URI of the form webit://UUID. When users drop Webits onto conventional form elements as a means to share those Webits (e.g., in a new email or blog post composition), Clui first attempts to display an iconic representation of the Webit in the target element when possible, and embed the JSON-serialized representation in a manner invisible to the user. For example, when dropping Webits onto a GMail message body that the user is composing, Clui inserts HTML that renders the iconic representation of the Webit and embeds its associated serialized data inside an HTML attribute. When displaying Webits inline on page elements is impossible (e.g., inside plain-text boxes), Clui pastes a human-readable representation of the Webit, along with a dereferencable link to the stored Webit.

When Clui encounters a Webit on a website, it first attempts to read the associated JSON-serialization, if an embedded one exists. If that succeeds, Clui regenerates the Webit icon and makes it draggable. Otherwise, Clui searches for the associated webit:// identifier and dereferences it for rendering.

Web Plugins

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

Plugins generally augment or scrape data on web pages by operating on known DOM elements, e.g., by using XPaths. Typically, plugin authors would use a web debugger to select relevant DOM elements and obtain the associated XPaths.

Plugins must wait for elements to load before augmenting or scraping them. Operating on static web pages, in which all elements are loaded before the DOM load event fires, is straightforward: plugins inspect, augment, or scrape the page in their onPageLoad function. 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.

Clui 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 title or URI, plugins can handle asynchronously loaded resources by responding to DOMSubtreeModified events or by polling.

Plugins that scrape or augment DOM elements are inherently brittle and prone to breaking as sites change. While tools that feature visual techniques for clipping and scraping (e.g., [17, 18]) may help ease the development burden of updating plugins, the most resilient approach is for web authors to directly publish semantic data. Web site authors may use the Clui API to create Webits and specify the semantic data in RDF/JSON. In the future, we imagine that web site authors could instead just embed microdata within their pages, from which Clui could automatically generate Webits.

Presentation Plugins

Presentation plugins interpret RDF statements and generate user-visible handles to the metadata. Each Presentation plugin handles a specific class of metadata, e.g., people, products, provenance, or location, and produces a set of key/value pairs shown to the user, e.g., in the metadata panel of Clui's workspace. Unlike Web plugins, which are site-specific, Presentation plugins work with metadata from any site as long as it matches the plugin's RDF vocabularies. In practice, this allows multiple Presentation plugins to extract metadata from a single Webit, enabling the extraction of (e.g.) location and contact information from a single set of RDF statements.

Limitations

When sharing Webits, Clui preserves the information by embedding it inside the drop site, but when that is not possible, Clui must paste a webit://UUID pointer and upload the Webit to a sharing service. One potential limitation is scalability of the sharing service, although existing services (e.g., pastebin.com) already provide Internet-scale pastebins for text and photos.

Another concern is the security and privacy associated with uploading Webits to such services. Currently, nothing prevents unauthorized access to uploaded Webits, though that could be addressed by storing Webits on personal storage servers with adequate access control. Additionally, Webits may contain sensitive information that is opaque or hidden to users. For example, a Webit representing a purchased flight itinerary might contain Webits for credit card accounts and billing addresses. A user that shares the itinerary Webit with colleagues (or on a blog) may be unaware that he is also sharing sensitive data. Future work must address this limitation, e.g., by improving the visibility of embedded Webits.

Webits on web pages currently do not degrade gracefully for users without Clui installed. Those users either see an icon that they cannot drag or a webit://UUID link they cannot dereference. One improvement is to use a dereferencable URI that conventional browsers can load (e.g., http://clui-project.org/webit/UUID). That URI could load a web page that displays a user-friendly description of the Webit and an invitation to install Clui.

Web Plugin	LoC	Comments
facebook.js	390	Scrape. *Async*. Background fetch of email address.
amazon.js	323	Scrape.
craigslist.js	317	Scrape and Augment (to add Webit icon).
acmdl.js	306	Scrape. *Async*. Background fetch of associated PDFs and EndNote bibliographic data.
provenance.js	288	Scrape.
newegg.js	285	Scrape.
reddit.js	275	Scrape. Save permalink of entries and background fetch of linked pictures.
aa.js	218	Scrape.
tooltip.js	199	Augment (general tooltip implementation).
gmail.js	188	Augment (to specially handle Webits dropped in To/Cc/Bcc fields). *Async*.
kayak.js	183	Augment (to add Webit drop zone to autofill form).
gmaps.js	118	Augment (to extract location data from Webits).
webit_uri.js	62	Augment (to display Webits inline).
twitter.js	49	Augment (to display Webits in Tweets).
gdocs.js	30	Augment (to override tooltips). *Async*.

Table 1. Properties of Web plugins. Entries labeled *Async* indicate sites that load asynchronously. LoC means lines of code.

To inspect a Webit's metadata, users currently must drag the Webit to the workspace first. Future work will enable users to inspect metadata inline on web pages containing Webits.

EXPERIENCES WITH CLUI

We implemented Clui as a cross-platform browser extension for Mozilla Firefox. As a browser extension, Clui 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 Clui's flexibility by reporting on our experience developing plugins to create Webits on various sites. Table 1 lists our Web plugins and Table 2 our Presentation plugins.

Provenance Plugin

One way we evaluated Clui'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 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 Clui'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 modifies 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, textareas, file upload zones) to display the appropriate tooltips when the user hovers over each respective target.

Presentation Plugin	LoC	Typical Metadata
publication.js	65	Title, Authors, Journal, doi
product.js	58	Vendor, Price, Ratings
realestate.js	56	Rent, Size, Number of Bedrooms
provenance.js	54	URL, Context, Snippet Text
person.js	45	Name, Email, Homepage, Phone Number
itinerary.js	44	Origin, Destination, Dates, Travelers
social_bookmark.js	37	Comments, Permalink
location.js	35	Area, City, Country

Table 2. Properties of Presentation plugins, including the typical metadata that they parse. LoC means lines of code.

The tooltips plugin also demonstrates communication between plugins, as other plugins (and web pages) must be able to override the default preview text. For example, when dragging Webits representing people over GMail's input boxes for specifying message recipients, the Clui GMail plugin overrides the tooltip preview text to display the associated email addresses.

Background Loading: Facebook, Reddit, & ACM Plugins

For additional flexibility, plugins may also obtain data from pages other than the page with the Webit. Clui 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 Clui, 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 fetched in the background. 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. Finally, with the ACM Digital Library (DL), scraping bibliographic data is trivial given a paper's EndNote file, which is not initially visible on the DL page. The ACM DL plugin downloads and parses the EndNote file to create Webits.

Asynchrony: GMail and Facebook Plugins

Several sites load resources asynchronously, e.g., GMail and Facebook. The plugins for those sites use the onTitleChange hook to detect when the page contents change dynamically, prompting them to inspect the DOM to, e.g., find form elements and override tooltips. As Table 1 suggests, implementation effort is independent of asynchrony but instead is commensurate with the complexity of scraping.

Data Interoperability and Presentation

When users drag a Webit representing a person from Facebook to a recipient field in GMail, the GMail plugin overrides default behavior and pastes the person's email address. The GMail plugin inspects Webits dropped into various fields and must infer whether those Webits represent people. To promote data interoperability, the Facebook and GMail plugins respectively produce and interpret metadata adhering to

the RDF Friend-of-a-Friend [4] specification, which defines a standard vocabulary for describing people.

Presentation plugins (Table 2) demonstrate another example of interoperability. Though Webits may originate from different sources and embed multiple Webits, Presentation plugins can each interpret the parts of Webits that it understands. For instance, the products plugin interprets Webits that come from different vendors, like amazon.com and newegg.com. Similarly, the real estate, person, and location plugins parse metadata relevant to each in apartment Webits.

CONCLUSION AND FUTURE WORK

This paper introduces the Clui platform and Webits as interoperable handles to high-level semantic objects on the web, like real estate, flight itineraries, and products. Clui features plugins for creating and interpreting Webits on existing web pages, as well libraries for web site developers to add native Webit support.

Natural extensions for Clui include Webits that represent live information (e.g., stock prices or weather), visual tools for plugin authors, and site-agnostic plugins that interpret microdata. We also hypothesize that having handles to semantic objects enables new interactions. For example, a user might purchase the contents of a shopping cart Webit filled with product Webits by dragging it to a web application that accepts Webit carts and Webits representing locations for the shipping address. The application could then inspect the Webits and use APIs exported by individual retailers to purchase the associated products. Such interactions need not be limited to web pages, but may occur directly on local, Webit-aware workspaces. For example, in addition to holding Webits, workspaces could also provide handles to actions that operate on specific kinds of Webits. A "purchase" action, when applied to product Webits, could prompt for additional Webits to complete the transaction.

Clui is an apt platform for future work in exploring new workspace models that manage Webits. For example, with Clui, 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., [19, 30, 33]) or capture user context to aid resource finding (e.g., [20, 31]). Furthermore, Webits and their workspaces might also provide new approaches for improving consistency in common tasks, like access control specification, which sites today each handle differently. Webit-aware workspaces could bridge those differences; for example, workspaces could consistently represent access control lists on different sites as a Webit containing people Webits, which users manipulate to affect a resource's access.

The cloud's arrival as a dominant application platform demands new user interfaces that leverage its advantages. Clui enables experimentation with handles to semantic objects and the standards that underlie them. We view Clui as a step towards future web environments that offer greater interface uniformity via consistent and interoperable handles.

ACKNOWLEDGMENTS

This work is sponsored by the T-Party Project, a joint research program between MIT and Quanta Computer Inc., Taiwan.

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. Evernote. http://www.evernote.com/.
4. The friend of a friend project. http://www.foaf-project.org/.
5. Greasemonkey. https://addons.mozilla.org/addon/greasemonkey/.
6. HTML5 drag and drop. http://www.w3.org/TR/html5/dnd.html.
7. Live clipboard - wiring the web. http://liveclipboard.org/.
8. Microdata — HTML standard. http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html#microdata.
9. Microsoft OneNote 2010. http://office.microsoft.com/en-us/onenote/.
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. Yahoo pipes: Rewire the web. http://pipes.yahoo.com/pipes/.
14. Zotero. http://www.zotero.org/.
15. Barreau, D., and Nardi, B. A. Finding and reminding: file organization from the desktop. *ACM SIGCHI Bulletin* (July 1995).
16. Bernstein, M., Van Kleek, M., Karger, D., and schraefel, m. c. Information scraps: How and why information eludes our personal information management tools. *ACM TOIS* (2008).
17. Dontcheva, M., Drucker, S. M., Salesin, D., and Cohen, M. F. Relations, cards, and search templates: user-guided web data integration and layout. In *ACM UIST* (2007).
18. Dontcheva, M., Drucker, S. M., Wade, G., Salesin, D., and Cohen, M. F. Summarizing personal web browsing sessions. In *ACM UIST* (2006).
19. Fass, A., Forlizzi, J., and Pausch, R. MessyDesk and MessyBoard. In *ACM DIS* (2002).
20. Freeman, E., and Gelernter, D. Lifestreams: a storage model for personal data. *ACM SIGMOD Record* (1996).
21. Fujima, J., Lunzer, A., Hornbæk, K., and Tanaka, Y. Clip, connect, clone: combining application elements to build custom interfaces for information access. In *ACM UIST* (2004).

22. Hartmann, B., Wu, L., Collins, K., and Klemmer, S. R. Programming by a sample: rapidly creating web applications with d.mix. In *ACM UIST* (2007).

23. 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).

24. Katifori, A., Lepouras, G., Dix, A., and Kamaruddin, A. Evaluating the significance of the desktop area in everyday computer use. In *ACHI* (2008).

25. Lin, J., Wong, J., Nichols, J., Cypher, A., and Lau, T. A. End-user programming of mashups with vegemite. In *ACM IUI* (2009).

26. Malone, T. W. How do people organize their desks?: Implications for the design of office information systems. *ACM TOIS* (1983).

27. Nardi, B. A., Miller, J. R., and Wright, D. J. Collaborative, programmable intelligent agents. *Commun. ACM* (Mar. 1998).

28. 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).

29. Stylos, J., Myers, B. A., and Faulring, A. Citrine: providing intelligent copy-and-paste. In *ACM UIST* (2004).

30. Sun, Y., and Greenberg, S. Places for lightweight group meetings. In *ACM GROUP* (2010).

31. Van Kleek, M., Bernstein, M., Karger, D. R., and schraefel, m. c. Gui — phooey!: the case for text input. In *ACM UIST* (2007).

32. Van Kleek, M., Bernstein, M., Panovich, K., Vargas, G. G., Karger, D. R., and schraefel, m. c. Note to self: examining personal information keeping in a lightweight note-taking tool. In *ACM CHI* (2009).

33. Voida, S., Edwards, W. K., Newman, M. W., Grinter, R. E., and Ducheneaut, N. Share and share alike: exploring the user interface affordances of file sharing. In *ACM CHI* (2006).