Wildcard: End user modification of web applications through a data table

Geoffrey Litt[a] and Daniel Jackson[a]

a Massachusetts Institute of Technology

Abstract Many people use Web applications that do not exactly meet their unique needs. While the Web platform supports client-side modification through user scripts and browser extensions, most people do not have the programming skills to implement such modifications.

In this paper, we present a prototype of a browser extension called Wildcard, that empowers users to casually tweak web applications without programming. Wildcard shows the main data from a web page in a table, and maintains a bidirectional connection between the table and the original page. By directly manipulating the table, people can perform a wide variety of modifications: sorting/filtering content, adding private annotations, using spreadsheet formulas to fetch data from other web services, using custom UI elements to edit form data, and more.

We present examples of using Wildcard to solve real world problems, and explain the design principles behind the prototype. In the future, we envision continuing to build Wildcard into a fully deployed system that makes the web into a more malleable medium.

ACM CCS 2012

- *General and reference → Computing standards, RFCs and guidelines;*
- **Applied computing → Publishing;**

Keywords programming journal, paper formatting, submission preparation

The Art, Science, and Engineering of Programming

Perspective The Art of Programming

Area of Submission Programming environments, Visual and live programming

1 Introduction

In 2012, the travel site Airbnb removed the ability to sort listings by price. Users could still filter down to a price range, but could no longer view the cheapest listings first. Many users complained on online message boards that the change seemed hostile to users. "It's so frustrating!..What is the logic behind not having this function?" said one user on the Airbnb support forum. Alas, the feature remains missing to this day.

This is a familiar situation in a world of web applications that are frequently updated without user consent. For most people, when web software does not quite meet their needs, their only recourse is to complain to the developers and hope someone listens. If they know how to program in Javascript, perhaps they can implement a user script or a browser extension to patch the issue, but most people do not have these programming skills. While many have become accustomed to this status quo, we see it as a waste of the openness of the Web platform and the general pliability of software. In *Personal Dynamic Media*, Alan Kay envisioned personal computing as a medium that let a user "mold and channel its power to his own needs," but today's software is far from this vision.

In this paper, we introduce Wildcard,[1] a browser extension that aims to make software more malleable by enabling users to tweak web applications without programming. Wildcard adds a panel to the bottom of a web page that shows a structured table view of the main data in the page. The table maintains a bidirectional connection to the original page—when the user manipulates the table, the original page gets modified, and vice versa.

In Wildcard, a user can sort Airbnb listings with just one intuitive click on a table header, with no programming required. Beyond sorting and filtering data, Wildcard also supports accessing third party APIs, performing small computations, recording private user annotations, using alternate UI widgets, and other useful changes. While Wildcard does not support all changes someone might want to make to a website, it makes broad subset of changes easily accessible to end users.

Under the hood, the implementation is straightforward, because a programmer must manually write an adapter for each individual website, which uses web scraping techniques to map the web page to the table. *todo: mention the other bits of extension code too* While programming is required for part of the process, this is still very different from traditional browser extensions—instead of the programmer defining a narrow use case, the end user is able to make many different changes on top of a single site-specific adapter. Programmers can extend Wildcard with plugins which provide various bits of functionality including connecting it to specific websites and web APIs, but the end user never needs to do any traditional programming.

note the current prototype stage

[1] Wildcard was the internal pre-release name for Hypercard, which served as an inspiration for this project since it promoted software modification by end users and was a precursor to the modern Web.

In this paper, we present examples of using Wildcard to solve real world problems, and explain the design principles behind the prototype:

todo: bullet point the design principles here

In the future, we envision building Wildcard into a fully deployed system that makes the web into a more malleable medium. *todo: this needs something more*

2 Demos

To get a sense of how it feels to use Wildcard, let's see an example of using it to help with booking a trip using the travel sites Airbnb and Expedia.

2.1 Sorting and filtering

We start by opening up the Airbnb search listings page to look for a place to stay. As mentioned before, this page annoyingly doesn't let us sort by price, so we'll use Wildcard to fix that. First, we open up the Wildcard panel, which shows a table corresponding to the search results in the page. As we click around in the table, the corresponding row in the page is highlighted so we can see the connection between the views.

To sort the page in ascending order by price, all we need to do is click on a table header to sort the table, and the original page gets sorted in the same order. We can also filter to only listings with a rating above 4.5 using a filtering UI on the column header. (Filtering by rating is another feature not offered in the Airbnb site.)

Note how after finishing the sorting and filtering, we can close the table view and continue using the website in its original design. The Wildcard interface is better for flexibly manipulating the underlying data, but the original site will usually offer a more polished and pleasant experience for viewing the data after it's in the desired form.

2.2 Row actions

Most websites that show tables of data also offer various actions that can be taken on a row in the table, like adding an item to a shopping cart. Wildcard has the ability to make these actions available in the data table, passed through by the site-specific adapter. The main advantage this provides to users is the ability to easily perform an action in bulk across multiple rows.

For example, it's tedious on Airbnb to click on listings one by one to add them to a list of favorites, or open the listings in a new tab. Using Wildcard, we can just select multiple rows, right click, and then choose an action from the context menu to apply to all the rows.

Within the site adapter, each action is implemented as a web automation that can do anything available in Javascript running in the context of the page: clicking on buttons in the UI, launching AJAX requests, navigating to a new page, etc.

2.3 User annotations

It's often useful to annotate a web page with notes. Users can use Wildcard to annotate a page by adding data into a new column, which is then shown in context of the original page.

Here, we use this feature to jot down notes on pros and cons of various listings:

When we come back later to the site, the annotations will still be visible. To support this, the site adapter saves the user annotations to browser local storage for persistence, and associates each table row with a stable identifier from the original site.

2.4 Computation with formulas

The demos so far have shown small, straightforward tweaks that provide useful conveniences while requiring very little effort. But Wildcard also supports adding more sophisticated functionality through a formula system.

When traveling without a car, it's nice to evaluate potential places to stay based on how walkable the surroundings are. There's an online service called Walkscore that can rate the walkability of any location on a 1-100 scale. It would take way too much work to manually cross-reference the scores with Airbnb locations, but we can use Wildcard formulas to automatically integrate Walkscore values into the page.

Wildcard includes a formula that uses the Walkscore API to fetch the score for any latitude and longitude. When we call the formula, it fetches the associated score and populates it into the page. By copy pasting the formula into all the rows on the page, we can add Walkscore data to all the search listings:

Programmers can extend Wildcard with new formulas, which are just Javascript functions that take table data as input and return new data. Formulas can access external APIs to fetch data, but cannot have side effects that directly manipulate the page. Instead, they return data, which the user can choose to add into the page by showing a column of data in the page.

todo: demo of composing formulas (requires a real formula system...)

2.5 Custom UI elements

It might seem that Wildcard is only useful on websites that display lists of tabular data like search results. But in fact, the table metaphor is flexible enough to represent many types of data. For example, a flight search form on Expedia.com can be represented as a table with a single row:

This alone doesn't provide additional value on top of the original form, but it becomes useful when combined with two other features of Wildcard. First, Wildcard offers writable cells, where edits in either the table or the original site propagate in both directions. Second, Wildcard offers UI widgets that allow users to graphically edit the value of a cell.

Here's an example of using those features to help with filling in a form. When filling in dates for a flight search, it's important to correctly remember the planned dates for the trip. This often requires opening up a separate calendar application to look up the

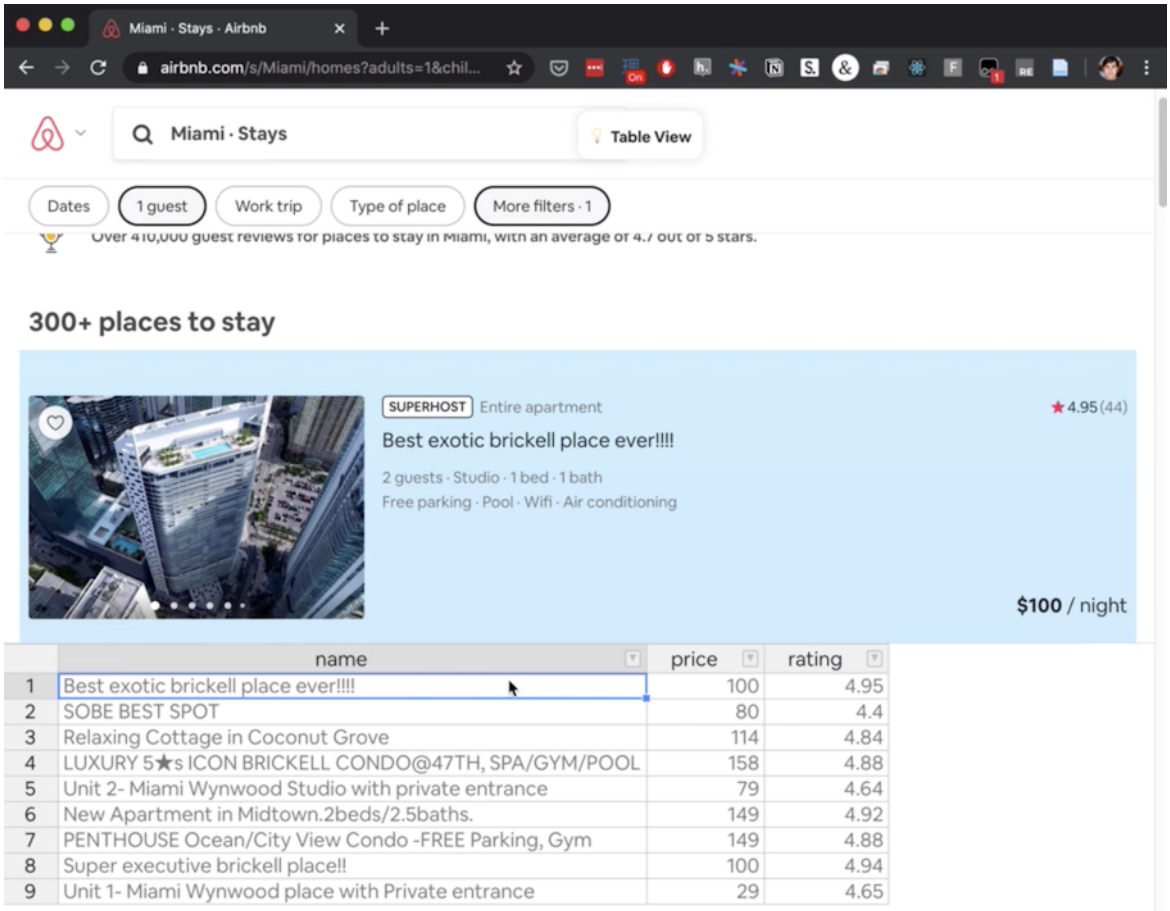


Figure 1 Opening a table corresponding to search results on Airbnb

dates, and manually entering the dates into the form. In Wildcard, we can do this without manual copying, by editing the date cell directly using a datepicker widget that has access to our calendar information. Notice how the dates in the original form update when we update the table cells.

Custom UIs enable people to use a consistent UI to enter common types of data across the web. They also allow a user to access their own private data as part of a web interface, without needing to expose that private data to the website.

The PDF version of the output will go here.

Probably make one giant full-page figure that explains the whole thing?

For example, in Fig. 1 we open up a table view that corresponds to search results on the Airbnb travel site.

Overall, we think an interactive data table is a natural computation model that presents a surprisingly large range of possibilities for end user modification of websites. While we've presented just a small sampling of use cases here to concretely illustrate some of these possibilities, we plan to continue developing site adapters, formulas, and UI elements to explore more use cases, and to eventually publicly release the tool to allow real end users to discover their own uses.

3 System Architecture

Wildcard is written in Typescript, and is injected into pages using the Tampermonkey userscript manager (although in the future we plan to deploy it as a standalone browser extension.)

In order to maximize extensibility, Wildcard is implemented as a small core program along with several types of plugins: site adapters, formulas, and cell renderers/editors. The core contains functionality for displaying the data table and handling user interactions, and the table implementation is currently built on the Handsontable Javascript library to enable rapid prototyping.

Site adapters....
Formula functions...
Cell viewers...

- Extension points
  - site adapters
    - show a snippet of adapter code
    - discuss how easy it is for programmers to make adapters
    - discuss possible future automation of adapter creation
    - Mention the technique of scraping data from AJAX requests
  - formula functions
  - cell viewer/editors
- future implementation goals
  - make it easy for programmers to add adapters + plugins, and distribute them to users. (Currently all adapters + plugins are part of the main Wildcard codebase)

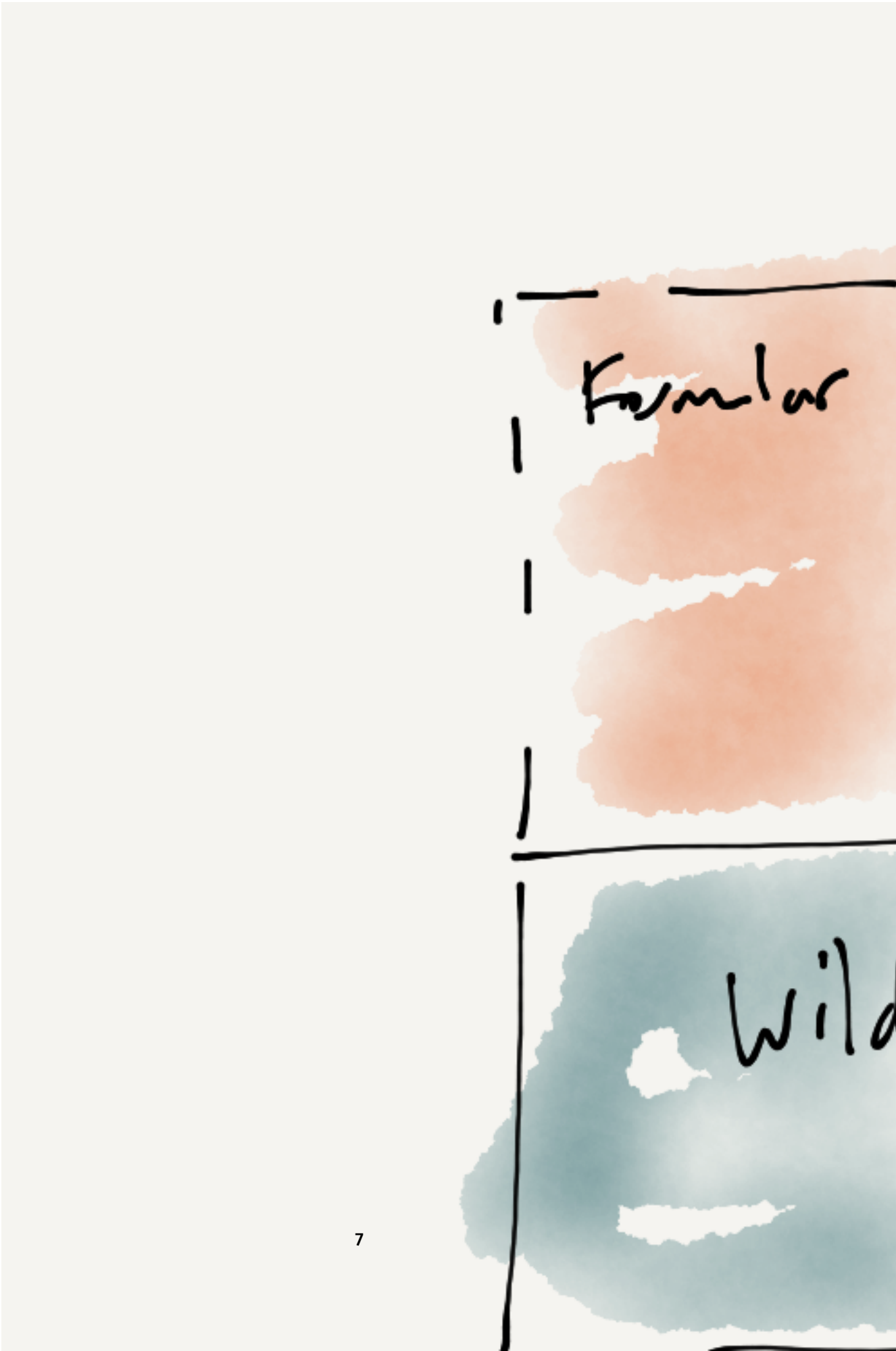
4 Design principles

The design of Wildcard is grounded in several principles, informed by prior work and our own experimentation. We hope you find these principles helpful not only for understanding our prototype, but also for designing other systems for end user programming.

Todo: add diagrams illustrating principles

4.1 Decouple UI from data

Most software does not allow users to choose their own UI elements, even for common data types. If a website provides a datepicker widget, you have no ability to provide your own replacement datepicker, with your preferred design or with privileged access to your private calendar data. This forces users to learn many different interfaces of varying quality for similar tasks. Some websites have APIs to allow users to access the data underlying the UI, but in addition to requiring programming, these are


Formular
Wild

heavyweight tools more fit for batch exports or building entire new clients than for casual UI modification.

In Wildcard, a user gets access to a view of the underlying data in the page, and can choose their own interfaces to view and modify the data. The Expedia datepicker demo showed one example of how this can be useful, but we also envision creating other widgets for visualizing and editing data. Some examples would be showing geographic data in a custom map that includes the user's own annotations, or editing a blog post in a rich text editor of the user's choice.

One benefit of decoupling data from interfaces is improved UI quality. When UI widgets can compete on their merits rather than based on network effects from the data they have access to, it creates much stronger competition at the interface level. For example, there is competition among email clients (which consume an open protocol), but not among Facebook or Twitter clients. This benefit relates to the SOLID project led by Tim Berners-Lee [2], which envisions user-controlled data as a mechanism for decoupling data from interfaces, e.g. giving users a choice of which client to use to consume a given social media feed. Wildcard has overlapping goals with SOLID, but does not require decentralized user control of data—the data can remain on a centralized server, as long as the interface can be tweaked by end users.

Another benefit of decoupling data from UI is that it becomes possible to use the same consistent interface across many applications. For example, many programmers become deeply familiar with one text editor and use it for many different kinds of tasks, even as an interactive input mechanism in the shell (e.g. for editing git commit messages). The ability to generically reuse the text editor in many contexts makes it worth investing time in mastering the tool. Beaudouin-Lafon and Mackay refer to this ability to use a UI tool in many contexts as *polymorphic* interaction [1], noting that it is a useful technique for keeping interfaces simple while increasing their power. diSessa also [4] notes that there is a connection between polymorphism and the idea of literacy in a medium: textual literacy rests on a single rich medium of writing which can be adapted to many different genres and uses.

Note: Maybe could relate this section to Concept Reuse?

4.2 Expose structure

In *Changing Minds* [4], Andrea diSessa critiques the design of modern software with a story about a hypothetical "nightmare bike." Each gear on the nightmare bike is labeled not with a number, but with an icon describing its intended use: smooth pavement uphill, smooth pavement downhill, gravel, etc. This might seem more "user-friendly" than numbered gears, but in fact, it makes it harder to operate the bike. A numerical sequence allows the user to develop intuition for the structure of the system, but isolated modes provide a superficial understanding with no grounding in structure. This understanding might be sufficient for the most common cases but breaks down in unfamiliar situations. If someone needs to go uphill on gravel, do they need to try every mode at random?

Many modern software designs fall into this trap, teaching users to use isolated modes rather than coherent structure, and the problem gets far worse when operating

across multiple applications. Unlike the UNIX philosophy of small tools interoperating through shared abstractions, in modern computing each application is in its own silo of data and functionality.

Wildcard helps people understand and modify the behavior of applications through the lens of a consistent abstraction: a data table. This abstraction strikes a balance between being both simple and generic. A data table is simpler than the DOM tree that describes the details of the UI, but is also generic enough to describe the essence of many different kinds of applications.

Creating a structured abstraction to represent a web page is a deliberate choice, and is not the only way to enable users to modify websites without directly accessing the DOM. Systems like Chickenfoot [3] and CoScripter [6] allow users to create scripts in an informal language and then perform fuzzy pattern matching to find elements in the DOM. For example, to find a button after a textbox in Chickenfoot, the user could type `click(find("button just after textbox"))`. These designs allow for expressing a wide range of operations, but they don't explicitly indicate what operations are possible—the user can only see the original page and imagine the possibilities. In contrast, Wildcard provides affordances that clearly suggest the availability of certain actions (e.g. sorting, editing a cell, adding a column with a derived value), especially to users who are familiar with spreadsheets. In addition to giving users more certainty about whether a modification is possible, these affordances might give users new ideas for things to try. Just as graphical interfaces better communicate the space of possible actions than command line interfaces, Wildcard aims to clearly communicate the space of possible modifications.

4.3 Direct manipulation of an alternate representation

In Wildcard, users manipulate an alternate representation of a web page. The interaction with the data table is direct like using a spreadsheet, but the interaction with the page is indirectly mediated through the table.

We considered other approaches where the user would interact more directly with the original UI, e.g. injecting sort controls into the page, but decided that the table view had advantages that justified the cost of adding a layer of indirection:

- *Consistency*: Even across different websites, the table view always has the same layout, making it easier to learn to use.
- *Affordances*: The table view suggests possible actions like adding a new column, which are challenging to suggest in the context of the original page.
- *Blank slate for UI*: When a custom UI element is used to manipulate a cell in the data table, there are no conflicts with the existing interface of the site.

The main challenge of making this design successful is maintaining a close mapping in the user's mind between the new representation and the original page (*note: cite Norman? Cognitive Dimensions 'closeness of mapping'?*). Wildcard provides live visual cues as the user navigates the data table, similar to the highlighting provided by browser developer tools to indicate the mapping between HTML and the original page.

In practice in our own usage, we have found that this live highlighting is sufficient to make it clear how the two representations map to each other.

4.4 Encourage casual tweaking

	Casual	*Not casual*
End user friendly	**Wildcard**	IFTTT
Requires programming	browser dev tools	editing open source desktop applications

We can evaluate a system for modifying software along two dimensions. First, the technical capability required of the user: is programming knowledge needed? Second, the level of effort required: how far out of their way the user must go to make a change? Can they casually make a small change, or do they need to make a larger project out of it? These dimensions are not orthogonal, but they are distinct. For example, setting up a workflow trigger in an end user programming system like IFTTT does not require much technical skill, but it does require leaving the user's normal software and entering a separate environment. On the other hand, running a Javascript snippet in the browser console requires programming skills, but can be done immediately and casually in the flow of using a website.

In addition to requiring no programming skills, Wildcard also aims to support frequent, small modifications. The Wildcard table appears in the course of normal web browsing, to ensure that the tools for modification are close at hand while using the original software. Ink and Switch refers to this property as having an "in-place toolchain" [5].

We also try to make simple changes possible with particularly low effort, like being able to sort a table in a single click. This property is inspired by spreadsheets, which can be useful even to someone who has learned only a small part of their functionality. In contrast, many traditional programming systems require someone to learn many complex concepts just to perform a simple task (e.g., needing to learn what `public static void main` means to write a Hello World program in Java).

4.5 First party cooperation optional

The Web is an unusually extensible platform. On many other platforms (e.g. smartphone operating systems), software is locked down unless first-party developers explicitly provide hooks for plugins and interoperability, but on the Web, all client-side code is available for browser extensions to modify. Application authors can use practices that make it easier to modify their apps (e.g. clean semantic markup), or more difficult (e.g. code obfuscation), but the default state is openness. This gives extensions freedom to modify applications in creative ways that the original developers did not plan for.

Wildcard takes advantage of this openness, and does not depend on cooperation from first-party website developers. Any programmer can add support for any website to Wildcard by building a third party adapter. This design decision acknowledges the

pragmatic need to interoperate with current websites, but we hope that eventually first party website developers will build in Wildcard support to their applications, since this would reduce the burden of maintaining adapters and make Wildcard plugins more stable.

Implementing the Wildcard adapter API could help developers by allowing users to fix some of their own issues, particularly idiosyncratic use cases that the first party developer would never plan to prioritize. Supporting Wildcard could be straightforward in a typical client-side application that already has access to a structured version of the data in the page. And while some developers might hesitate to promote extensibility in their clients to avoid unwanted changes, the most common problem of users blocking ads is already ground well trod by existing browser extensions. There is also precedent for first parties implementing an official client extension API in response to user demand: for several years, Google maintained an official extension API in Gmail for Greasemonkey scripts to use. (Incidentally, since then, third parties have continued to maintain Gmail extension APIs used by many Gmail browser extensions [7, 8], illustrating the value of collaboratively maintaining third party adapters.)

4.6 No magic

- an ecosystem of programmers collaboratively building a platform for end users
- but still empowering end users to do the creative parts, the final glue
- less brittle, less confusing and surprising (maybe Chickenfoot or Coscripter had this problem? Find a nugget in those papers to support. I remember Helena had something about auto-scraping breaking… contrast with Gmail.Js stability?)
- still leaves the door open for future automation
- maybe “first party optional” can be included here?

5 Related work

Note: a lot of this was already covered above; how to deal with that?

- Malleable software: Kay, Webstrates
- Instrumental interaction, polymorphic UI
- Web automation: Chickenfoot, CoScripter
- Wrapper induction: Thresher, Helena
- Personal data ownership: SOLID
- Extension helper libraries, e.g. Gmail.js.
- Doctorow: adversarial interoperability
- Marmite

6 Future work

- limitations / future ideas

  - the spreadsheet language is very primitive, can we make it more expressive? what are the exact semantics of fetching data from APIs? What about making API requests that do mutation, not just fetching data?
  - only works when the user is browsing. Should we explore triggers, scheduled scraping?
  - only has spreadsheet-style functional transformations. Should we explore imperative workflows? Injecting buttons into pages that do things? (Eg, imagine a "save to google maps" button that you can inject into a page)
  - limited options for how to style injected content, could explore styling (eg, maybe you can restyle a table cell and the styling is reflected when it's injected into the page?)
  - how much do adapters generalize to many sites? We've built X adapters but need to make more
  - page boundaries, eg scraping multi page results
  - how do people save scripts and share them with each other? TBD
  - nested data? lean on SIEUFERD?
  - reduce the number of primitives?
- still in early development; note the beta release plan (tentative: target public beta availability at the workshop in March?)
- Could explore automated wrapper induction, building on prior work
- Want to get more real usage of the tool and run usability studies
- Include a link to sign up for future updates

Acknowledgements

References

[1] Michel Beaudouin-Lafon and Wendy E. Mackay. "Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '00. New York, NY, USA: ACM, 2000, pages 102–109. ISBN: 978-1-58113-252-6. DOI: 10.1145/345513.345267.

[2] Tim Berners-Lee. *One Small Step for the Web…* en. https://medium.com/@timberners_lee/one-small-step-for-the-web-87f92217d085. Sept. 2018.

[3] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. "Automation and Customization of Rendered Web Pages". en. In: *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology - UIST '05*. Seattle, WA, USA: ACM Press, 2005, page 163. ISBN: 978-1-59593-271-6. DOI: 10.1145/1095034.1095062.

[4] Andrea A. diSessa. *Changing Minds: Computers, Learning, and Literacy*. Cambridge, MA, USA: MIT Press, 2000.

[5] Ink and Switch. *End-User Programming*. en-US. Mar. 2019.

[6] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. “CoScripter: Automating & Sharing How-to Knowledge in the Enterprise”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’08. New York, NY, USA: ACM, 2008, pages 1719–1728. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357323.

[7] Streak. *InboxSDK*. https://www.inboxsdk.com/.

[8] Kartik Talwar. *Gmail.Js*. en. https://github.com/KartikTalwar/gmail.js. 2019.

About the authors

Geoffrey Litt is bla bla bla

Daniel Jackson is bla bla bla