Wildcard: Modifying web applications by directly manipulating a data table

Geoffrey Litt[a] and Daniel Jackson[a]

a Massachusetts Institute of Technology

Abstract Browser extensions and user scripts can modify web applications in useful ways, but many people have unique needs that aren't met by existing extensions. Today, most of them are stuck—they can't build their own browser extensions without learning how to program in Javascript.

Wildcard is a platform that empowers anyone to build browser extensions without doing traditional programming. Wildcard shows the main data from a web page in a table with a bidirectional connection to the original page. People can directly manipulate the table to sort/filter content, add private annotations, and use spreadsheet formulas to fetch data from other websites.

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 Social Coding, General-purpose programming

1 Introduction

Todos for the introduction:

- hook with Airbnb sort
- add more bigger picture reasons, less focus on the affordances
- make this shorter

People have complaints about web apps they use, but they rarely modify those apps to meet their needs. Why not? Some guesses:

- Technical skill: Most people don't know how to use Javascript and manipulate the DOM.
- Low ROI: It takes a long time to reverse engineer a site, and it's usually not worth it

These seem reasonable but they might not tell the whole story.

There have been many research projects that try to address these hurdles. For example, Chickenfoot [1] allows for people to more quickly modify sites without using Javascript or dealing with the DOM, directly addressing both of these barriers. But, a decade later, not many people have ended up using systems like this.

I know how to do web programming, and yet I rarely modify my apps. Sometimes it actually turns out to be pretty easy to hack on a site once I start doing it so the ROI is actually pretty high. These reasons don't seem to fully explain my behavior.

A disheartening explanation might be that most people just don't care enough to make changes. This claim might be true in the current context, but it's also important to remember that motivation is connected to culture and to the space of possibilities provided by our tools. Most people probably didn't want to write letters before mass literacy.

One interesting explanation is that **people can't estimate the difficulty of a change**. It's not motivating to think about what changes I might want when I don't know whether they would take a few minutes or are completely impossible. For me, an engineer, this gets at the heart of the issue. I know I could probably implement the change, but it's so hard to estimate how long it would take (minutes? months?) that I don't even bother trying. In general, when something seems expensive to do (or even possibly expensive), it can discourage casual lightweight experimentation.

So, perhaps to encourage people to casually modify software, **apps need to provide more consistent affordances indicating what changes are possible and easy.** Maybe if it were more obvious that certain types of changes could be achieved in mere minutes, programmers and non-programmers would end up modifying our software more.

1.0.1 The Wildcard platform

Wildcard adds a panel to the bottom of a web page that shows a structured table view of some of the core data in the page. When the user manipulates the table, the original page also gets modified. We aim to make the mapping between the table and the page as direct and intuitive as possible.

Todos here:

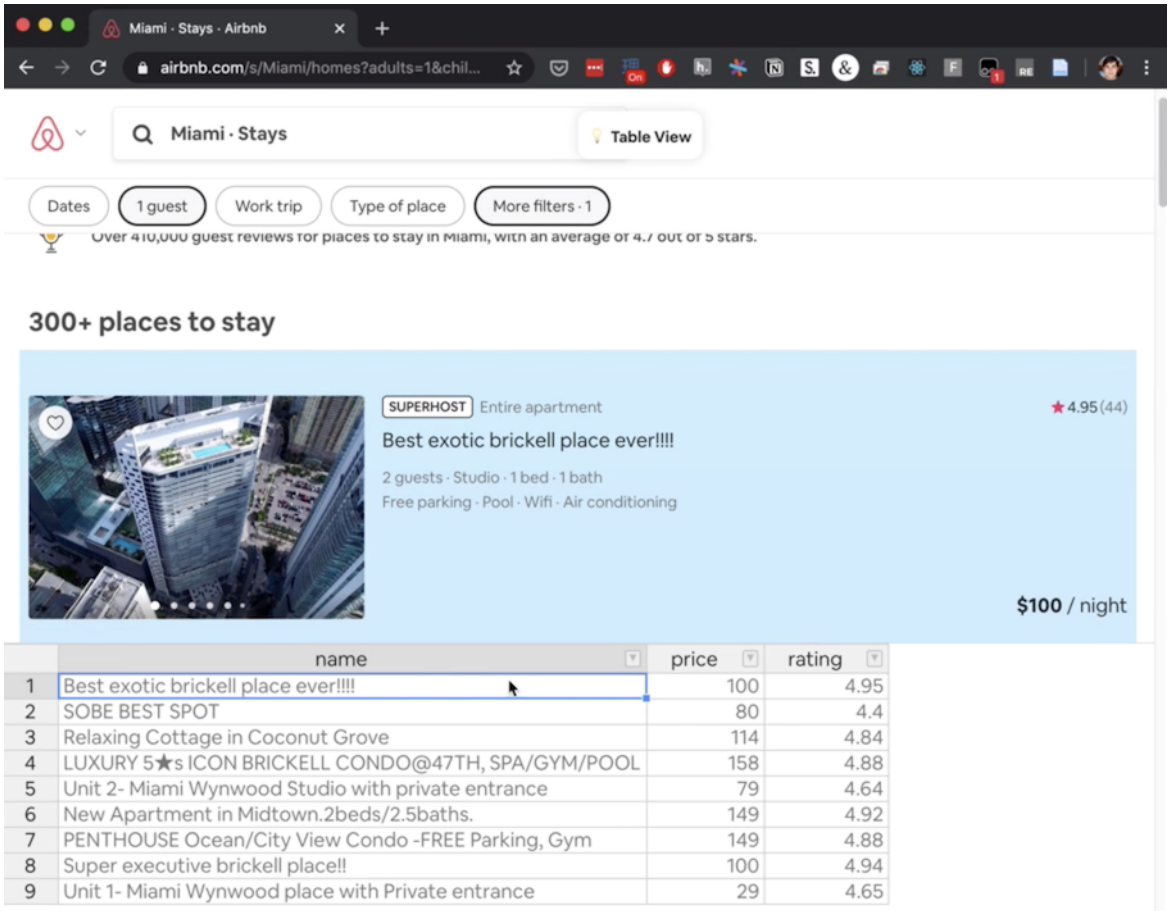

Figure 1 Opening a table corresponding to search results on Airbnb

- a better demo is actually sorting too at this point too, to show the bidirectional nature
- hint at future possibilities
- introduce an image fallback for the PDF version

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

Wildcard is fairly general and can support many useful changes to websites, which will be demoed later:

- sorting and filtering data: eg sorting shopping results
- using 3rd-party APIs and performing small computations to add new data, in the style of "web mashups": eg adding walkability scores to hotel listings
- adding private user annotations to the page: eg taking notes on different options
- using alternate UI widgets to enter data into a page: eg using a personal datepicker widget with private calendar data, to enter the right dates for taking a flight

The overall goal is to provide generic tools that fit well with the table paradigm and enable many specific useful changes. But it's important to note that Wildcard doesn't aim to provide maximum coverage all the possible ways someone might want to

modify a web page. Rather, it aims to provide a useful, simple subset of modifications, and to provide consistent affordances so that users confidently understand which modifications they can make.

2 Demo: Booking a trip with Wildcard

- Sorting search results:
  - Airbnb took away search
  - once you see the tabular view, there's an obvious interaction available to anyone familiar with tables
- Injecting new data into the page
  - Take own custom notes, saved in the browser. Maybe shared in the future
- Formulas
  - Compute values to inject into the page
  - TBD: styling
- Using a custom date picker

3 Implementation

Eventually web apps might provide the structured data table themselves. In the meantime, we need some sort of adapter to make this system work with existing sites.

Wildcard provides a system for creating a wrapper on top of existing websites. This wrapper defines how structured data can be extracted out of the page, and also how manipulating the table should modify the page.

The most basic way of building these wrappers is for skilled programmers to manually build and maintain them for popular sites. This approach beats the status quo because many people, including end users, can benefit from the generic wrapper and use it in many ways. This is different from the current world where programmers build use-case-specific browser extensions, and each extension has to implement its own interactions with the low-level DOM of a page. There's also a greater incentive for many people to collectively maintain a wrapper if it's shared.

A more advanced way would be to make these wrappers partially or totally automated, and enable end users to create them. This future work could leverage existing research on wrapper induction but isn't the focus of the current work.

4 Design principles

Here are some of the ideas behind the design of Wildcard. We hope these can be useful principles not only for Wildcard, but also for other tools that enable end users to modify software.

4.1 Expose unified structure across applications

In *Changing Minds* [2], 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. By some logic, this is more "user-friendly" than numbered gears, but in fact, hiding orderly structure from the user makes it more difficult to operate the bike. 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.

Exposing a unified higher-level abstraction is a deliberate choice, and is not the only way to enable users to program without directly using the DOM. Chickenfoot [1] and CoScripter [3] allow users to create scripts that resemble natural language and "sloppily" match queries to elements in the DOM. These designs allow for a wide range of operations, but don't explicitly indicate what operations are possible—the user must look at the original page and imagine the possibilities. In contrast, Wildcard provides affordances that clearly suggest the availability of certain types of modifications. In addition to giving users more certainty about whether a modification is possible, these affordances might give users new ideas for things to try. Because people are not used to modifying web applications, providing inspiration is an important goal. (Todo: Perhaps something to cite here, re: discoverability in GUIs vs CLIs?)

4.2 Direct manipulation by proxy

In Wildcard, users don't interact with the original page to modify it; instead they manipulate an alternate representation of the data in the page. The tool uses visual cues to indicate the mapping between the table representation and the original page. This is similar to the way that browser developer tools use in-page highlighting to indicate the mapping between HTML and the original page.

We considered alternate approaches where the user would interact more directly with the original page, but decided that the table view provided a consistent

- True DM might be right on the page itself
  - There are good UIs for this, eg Chrome DevTools element picker
- Considered for Wildcard, but went with an indirect proxy. Still maintain a close mapping.
- Pros: consistency across sites. Avoid site-specific styling issues.

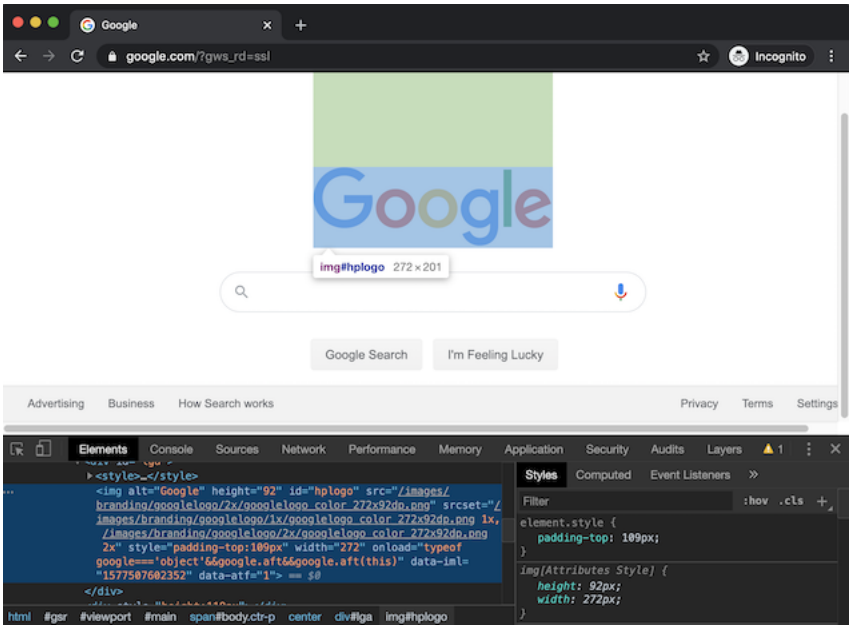

Figure 2 The Chrome Dev Tools use highlighting to show the mapping between HTML code and the page

- Cons: maybe less intuitive? More work to do the mapping, might be a source of confusion.

4.3 First party optional

- 3p-only is enough. 1p help is optional.
- Can even do things the 1p didn't want to expose.
- make it easy for 1p: just ask for a data table, no fancy schema.

4.4 Built for the web

- web has the right foundations:
  - the right platform
  - open platform
  - dev tools
  - originally intended to be
- in-place toolchain: meet the user where they are, in the page

4.5 Decouple UI from data

- bring your own views of the data
- bring your own widgets for data entry
- instrumental interaction

5 Related work

- Instrumental interaction
- Web automation
- Wrapper induction
- Personal data
- Extension helpers

6 Future work

- How far does functionality go?
  - workflows? triggers?
  - what can and can't be done
- automated wrappers?
  - lean on existing tech
- usability studies

7 meta: todos

- remove bold text for the proceedings version
- go through the Programming proceedings template checklist

Acknowledgements

References

[1] 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.

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

[3] 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.

About the authors

Geoffrey Litt is bla bla bla

Daniel Jackson is bla bla bla