Wildcard: End user modification of web applications using a live data table

Geoffrey Litt[a] and Daniel Jackson[a]

a Massachusetts Institute of Technology

Abstract Many Web applications do not meet the particular needs of their users. Browser extensions and user scripts offer a way to customize web applications, but most people do not have the programming skills to implement their own customizations.

In this paper, we present a prototype of Wildcard, an extension framework embedded in the browser that empowers users to casually tweak web applications without needing to program in Javascript or directly interact with the DOM.

Wildcard shows the data from a web page in a table, and maintains a live bidirectional connection between the table and the original page. By directly manipulating the table, users can perform a wide variety of modifications: sorting/filtering content, adding private annotations, using spreadsheet formulas to fetch data from other web services, adding 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 growing Wildcard into a fully deployed system that helps make the web into a more malleable medium for all its users.

ACM CCS 2012

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

Keywords end-user programming, software customization, web browser extensions

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 by 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. Sometimes there is a browser extension or user script that fixes an issue, and if the user is both motivated and skilled they might even be able to implement their own fix. But for most people in most situations, the only recourse is to complain to the developers and pray that someone listens—or more likely, to simply give up. While many have become used to this status quo, we see it as a tremendous waste of the openness of the Web platform. Back in 1977, in *Personal Dynamic Media* [17], Alan Kay had originally envisioned personal computing as a medium that let a user "mold and channel its power to his own needs," but today's software is more like concrete than clay.

In this paper, we introduce Wildcard, an extension framework embedded in the browser that lets users tweak web applications without programming in Javascript or interacting with the DOM. Wildcard shows a structured table view of the main data in a webpage, which maintains a live bidirectional connection to the original UI: when the user manipulates the table, the UI is instantly modified, and vice versa. This simple framework allows a wide range of possible modifications, ranging from sorting lists to fetching related data from other websites.

The system builds on two insights from prior work. The first insight is that a spreadsheet can serve as the backend for a GUI application, which enables an end user to build applications in an extremely accessible environment without doing traditional programming. The second insight is that web scraping—extracting structured data from an unstructured web page—unlocks value by making it possible to compute over that data. Our contribution in this work is to combine those two ideas in a novel way. By extracting structured data from an application and showing that data in a live spreadsheet table, Wildcard enables a wide class of end user modifications to existing applications. Wildcard makes it seem like an application is backed by a spreadsheet, even though the actual implementation is not. This allows the user to easily manipulate and compute over the application's underlying data just as they would in a spreadsheet, and then to immediately see the results of their changes in the original application's UI.

Wildcard embodies several principles we propose for designing systems that enable end user modification of existing software:

Expose consistent structure: Most applications only expose a user interface, without showing the structure of the data underneath. Furthermore, every application has a different interface, requiring users to learn many ways of using apps. Wildcard shows data from diverse applications in a single consistent structure, so that users are able to invest in learning one mechanism for manipulating the data in their applications, and for programming with that data.

Low floor, high ceiling: We aim to provide a low barrier to entry so that non-programmers can casually use the system, but a high ceiling on the complexity it can achieve. Small but genuinely valuable tweaks like sorting a page are made possible with little effort in the flow of normal use to provide a low floor, and a rich formula system creates a high ceiling on possible modifications.

Build for multiple tiers of users: We aim to make Wildcard as easy as possible for end users, by designing roles for tiers of users with differing levels of technical sophistication. Programmers or tech-savvy end users are responsible for the data extraction step of mapping existing UIs to structured data. Most end users only interact with the structured data, providing a predictable and user-friendly experience.

Wildcard is currently an early research prototype, with incomplete features and limited coverage of sites. We plan to continue privately testing the system with our own use cases, and then to release the tool publicly, enabling other programmers to contribute site adapters and other plugins. Ultimately, we hope that Wildcard makes modifying websites into a natural part of using the web, as ubiquitous as using spreadsheets in everyday computing.

2 Demo: booking a trip with Wildcard

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

The user starts by opening up the Airbnb search listings page to look for a place to stay. The page looks nice and mostly works well, but is missing some key features. As mentioned before, this page doesn't allow the user to sort by price. It also doesn't let them filter by user rating. Using Wildcard, the user can add these small features, while leaving the page's design and the rest of its functionality unchanged.

First, the user opens up the Wildcard panel, which shows a table corresponding to the search results in the page. As they click around in the table, the corresponding row in the page is highlighted so they can see the connection between the views.

Then, the user can use standard spreadsheet column header features to sort the page by price and filter by rating:

Notice how after manipulating the data, the user was able to close the table view and continue using the website with its original visual design. The table view offers a way to change the data backing a page, but does not need to replace the original interface entirely.

Most websites that show lists of data also offer 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 if the site adapter implements them. The main advantage this provides 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. Using Wildcard, we can just select multiple rows and favorite all of them with one click. Similarly, we can also open the detailed pages for many listings in new tabs.

Now the user would like to jot down some notes on the pros and cons of each listing. To do this, they can simply type notes into an additional column next to each listing, and the notes appear inside the listings in the original UI. These annotations are also persisted in the browser for future retrieval.

Wildcard also includes a formula language which enables more sophisticated tweaks that fetch external data and perform computations.

When traveling without a car, it's nice to evaluate potential places to stay based on how walkable the surroundings. Using Wildcard formulas, we can integrate Airbnb with Walkscore, an API that rates the walkability of any location on a 1-100 scale. When we call the `WALKSCORE` formula with the latitude and longitude of the listing, it returns the walk score as the cell value. Because the cell's contents are injected into the page, the score also shows up in the page body.

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 form can be represented as a single row, with a column for each input.

In previous examples the data extracted from the site was marked as read-only; users cannot change the name or price of an Airbnb listing. In this next case, the cells are marked as writable, so that changes in the table are reflected in the form inputs. This becomes useful when combined with GUI widgets for editing the value of a table cell.

Filling in dates for a flight search typically requires opening up a separate calendar app to find the right dates, and then manually copying them into the form. In Wildcard, we can make this easier by providing a datepicker widget that has privileged access to the user's calendar information.

Here we've presented just a few possibilities for how to use Wildcard. We think the interactive data table offers a flexible computational model that can support a wide range of other useful modifications, all while remaining familiar and easy to use.

3 System Architecture

Wildcard is written in Typescript. It is currently injected into pages using the Tampermonkey userscript manager, but in the future we plan to deploy it as a standalone browser extension to make it easier to install.

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 built using the Handsontable Javascript library.

Site adapters specify the bidirectional connection between the web page and its structured data representation.

For extracting data from the page and getting it into structured form, Wildcard provides ways to concisely and declaratively express scraping logic. For example, here is a code snippet for extracting the name of an Airbnb listing. It specifies the field

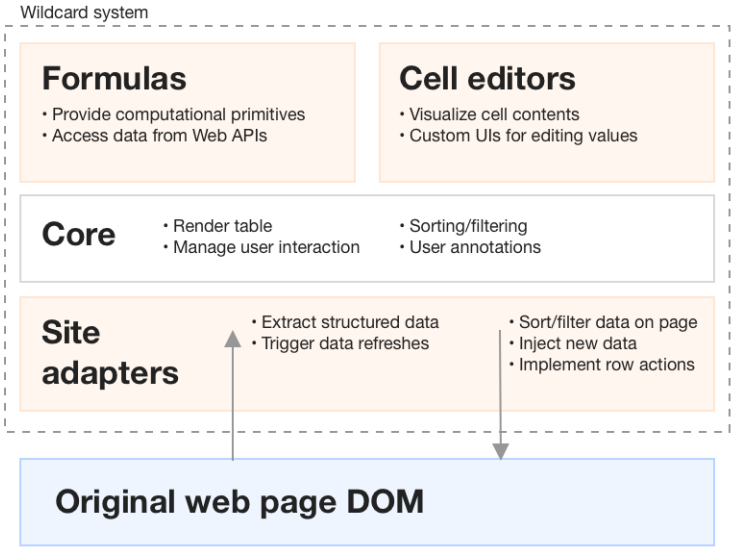

Figure 1 The architecture of the Wildcard system

name and some metadata properties, and then a function that, given the entire listing, returns the DOM element representing the name of the listing.

```
{
  fieldName: "name",
  readOnly: true,
  type: "text",
  el: (row) => row.querySelector(.${titleClass}),
}
```

The site adapter also needs to support the reverse direction: sending updates from the table to the original page. Most DOM manipulation is not performed directly by the site adapter: the adapter specifies how to find the divs representing data rows, and the core platform mutates the DOM to reflect the table state. The only exception is site-specific actions (like favoriting in Airbnb): actions are implemented as regular Javascript functions running in the context of the page, which can mutate the DOM, simulate clicks on buttons in the UI, launch AJAX requests, navigate to new pages, etc.

4 Design principles

The design of Wildcard is grounded in several principles, informed by prior work and our own experimentation. We think these principles extend beyond Wildcard and might inform the design of other end user programming tools, particularly systems enabling users to modify GUI apps.

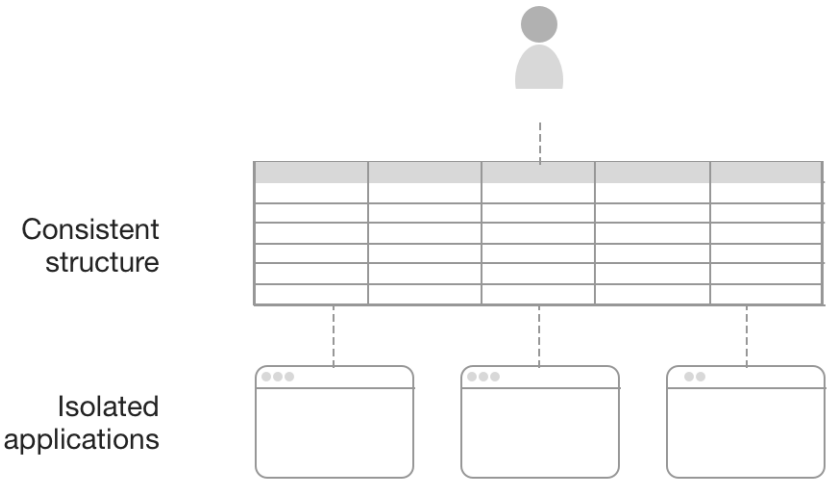

Figure 2 Wildcard layers the consistent structure of a data table over different isolated applications.

4.1 Expose consistent structure

Today, most personal computing consists of using distinct apps, each one an isolated silo of data and functionality. While there are some shared idioms and limited points of interoperability, apps fundamentally operate independently from one another. Computing does not need to operate this way—for example, UNIX offers a compelling alternate vision: many small tools, all built on the consistent abstraction of text files. Users can invest in deeply mastering a text editor, and then use it for a vast array of tasks, even as an interactive input mechanism in shell programs (e.g. for editing git commit messages). Beaudouin-Lafon and Mackay call this style of interaction *polymorphic* [3], since a single interface widget can be used in many contexts. diSessa [11] has also noted the deep connection between polymorphism and literacy: textual literacy rests on a single rich medium of writing, which once mastered, can be adapted to many different genres and uses. If people needed to relearn from scratch when switching from writing essays to writing emails, the medium would lose most of its potency.

With Wildcard, we hope to make isolated Web applications work more like UNIX, by imposing a consistent and simple data structure onto many applications. Just as UNIX uses text files to represent all data, Wildcard uses relational tables. This abstraction strikes a balance between being simple and generic: a data table is much simpler than the DOM tree that describes all the details of the UI, but is also generic enough to describe the essence of many different interfaces.

Exposing structure gives users more choice When UI widgets can compete on their merits rather than having a monopoly on a given data silo or application, it creates 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. Exposing structure also leads to improved UI consistency across apps: UNIX

users can reuse their text editor everywhere, but Web users must use a different rich text editor for each site. Tim Berners-Lee has written in more detail about these benefits [5] in the context of the SOLID project, which envisions giving users control of their own 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.

todo: move the sloppy contrast, and SOLID, down into related work

Layering a structured view on top of an existing application is not the only option available. "Sloppy programming" systems like Chickenfoot [6] and CoScripter [19] forego the intermediate layer of structure, instead allowing 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 improve discoverability and give users new ideas for things to try. "Sloppy programming" does give end-users greater access to modify the apps that they use, but does not seem to provide them with a deeper understanding of those apps or how they relate to each other, so the apps are more likely to remain siloed.

One key challenge of making a structured view successful is ensuring a close mapping in the user's mind between the new representation and the original page. Wildcard provides live visual cues as the user navigates the data table (similar to the highlighting provided by DOM inspectors in browser developer tools). In our own usage, we have found that this live highlighting makes it very clear how the two representations map to each other.

4.2 Low floor, high ceiling

Seymour Papert advocated for programming systems with a "low floor" and a "high ceiling"—making it easy for novices to get started, but also providing possible a large range of possibilities for more sophisticated users [25]. Unfortunately, many programming systems only provide either one or the other [22].

- tweaking vs creation from scratch
- in-place toolchain
- easy things easy
- easy things valuable
- this seems clearly justified by the demos

rewrite...

	In-place	*Not in-place*
End user friendly	**Wildcard**	IFTTT
Requires programming	browser dev tools	editing open source software

We can evaluate a system for tweaking software along two dimensions. First, the technical capability required of the user: is programming knowledge needed? Second, whether the change is "in-place" [16]: can it be made within the same environment that the software is typically used in? 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 supports 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.Wildcard also provides live feedback, immediately editing the page in response to user interaction, which gives users confidence in the changes they are making.

Spreadsheets can be used in genuinely useful ways even by someone who has learned only a tiny part of their functionality, e.g. for storing tables of numbers or computing simple sums [24]. This creates inherent motivation to continue using the tool and eventually learn its more powerful features. In contrast, many traditional programming systems require an enormous time investment before a beginner is able to create any valuable software. Like spreadsheets, Wildcard aims to support genuinely useful actions with little effort: for example, sorting a table with a single click, or adding an annotation by typing into a cell. We would expect many Wildcard users to start by using these bits of functionality, before possibly moving on to more complex features like formulas.

High ceiling?

4.3 Build for multiple tiers of users

- make things easy for the last tier of users (low floor)
- expose predictable structure, don't make them participate in data extraction

Some end user programming systems are designed with a single user in mind—a lone non-programmer, needing to achieve a task through some combination of individual skills and a powerful tool. But in fact, real-world usage of end user programming systems is far more collaborative. Spreadsheets are built together by users with different levels of technical expertise: some users just write simple formulas, while their officemates help write more complex formulas or even program macros in traditional languages [23]. Our goal with Wildcard is to consider the full ecosystem of users and to design a tool that best combines all of their abilities.

This design principle is most salient in our approach to building site adapters. Many web augmentation projects aim to make the full process of tweaking available to

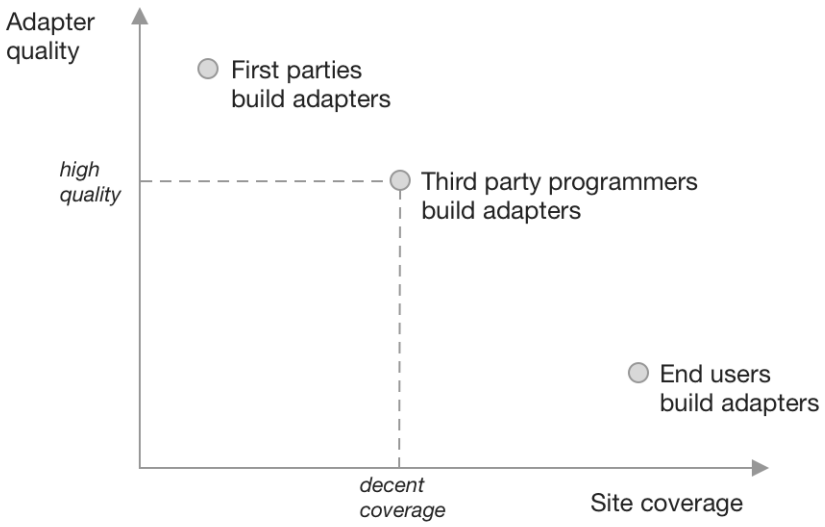

Figure 3 Having third party programmers create site adapters balances site coverage and adapter quality.

end users with no programmer help needed, through a combination of automated heuristics and interactive feedback. This is a laudable goal because it makes these tools work with any site, and these approaches are often quite successful, but they also inevitably break down sometimes [6] [20]. In contrast, human programmers working together have been able to maintain stable scraping libraries, even for complex sites like Gmail [26, 27]. Our primary approach to building site adapters is having programmers manually create them, because we are willing to trade off lower site coverage in return for more reliable results on the sites that are covered, building confidence for users. Like the collaboration between an expert and novice spreadsheet user, this architecture leverages more skilled programmers while still providing end users with a flexible programming environment to meet their own needs. On the other hand, we remain very open to incorporating other tools in the future that enable end users to also create their own adapters.

In any system for tweaking software, another important consideration is how to involve the first party developers of the original software. Our approach is to avoid depending on their cooperation, but also to aim for eventual cooperation.

This approach takes advantage of the unusual openness of the Web 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. At the most extreme, this can even include

modifications that are hostile to the desires of the original developers. Although ad blocking extensions make this style of extension very prominent, most ideas for extending software merely fall outside the range of possibilities originally imagined by the developer, rather than being actively hostile.

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. But the platform is still open to first parties developing their own adapters, which would often be in their interests. Providing Wildcard support would allow users to build extensions to fulfill their own feature requests. It also would not necessarily require much effort: adding Wildcard support would be more straightforward for a first-party than a third-party because they have direct access to the structured data in the page. 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 stable Gmail extension APIs used by many browser extensions [26, 27], illustrating the potential of collaboratively maintaining third party adapters.)

5 Related work

In the broadest sense, Wildcard is inspired by systems aiming to make software into a dynamic medium where end users frequently create and modify software to meet their own needs, rather than only consuming applications built by programmers. These systems include Smalltalk [17], Hypercard [15], Boxer [10], Webstrates [18], and Dynamicland [28]. (The project's name Wildcard comes from the internal pre-release name for Hypercard, which doubly inspired our work by promoting both software modification by end users and the ideas behind the Web.)

While similar in high-level goals, Wildcard employs a different solution strategy from these projects: whereas they generally require building new software from scratch for that environment, Wildcard instead aims to maximize the malleability of already existing software. This approach has the pragmatic benefit of immediately being useful in a much broader context, although it also requires working within rigid constraints. We also see Wildcard growing beyond its current bounds and becoming more similar in the future to these other systems. With substantial future work, Wildcard could grow from merely being a platform for tweaking existing software into a platform for building new software from scratch, designed from the ground up for end user tweakability.

More narrowly, Wildcard builds on three areas of existing work: web augmentation, spreadsheet-based app development, and web scraping. Our contribution in this work is to combine these areas in a new way: extracting structured data from a website and directly exposing it to the user as a live data table, with the end goal of modifying an existing application. We think this combination builds on valuable ideas and learnings from each of the areas, while also overcoming some of the downsides of each.

5.1 Web augmentation

Wildcard's goals are closely shared with other systems that provide interfaces in the browser for end users to augment and modify websites while using them.

5.1.1 Structured augmentation

Wildcard's approach is most similar to a class of tools that identify structured data in a web page, and use that structure to support end user modification of the page.

Sifter [13] enables users to sort and filter lists of data on web pages, providing a result similar to Wildcard's sort and filter functionality. The underlying mechanism is also similar: Sifter extracts structured data from the page to enable its user-facing functionality. Wildcard aims to extend this approach to support much broader functionality than just sorting and filtering. In support of this goal, Wildcard also shows the structured data table directly to the user, whereas Sifter only shows sort and filter controls, without revealing the underlying data table. The extraction mechanism is also different: Sifter uses a combination of automated heuristics and interactive user feedback to extract data, whereas Wildcard currently relies on programmers creating wrappers for extracting structured data, likely leading to higher quality extraction but on fewer sites.

Thresher [12] enables users to create wrappers which map unstructured website content to Semantic Web content. Like Wildcard and Sifter, Thresher augments the experience of original page based on identifying structure: once semantic content has been identified, it creates context menus in the original website which allow users to take actions based on that content. Wildcard and Thresher share an overall approach but focus on complementary parts of the problem: Thresher aims to enable end users to create content wrappers, but the actions available on the structured data are created by programmers; conversely, Wildcard delegates wrapper creation to programmers but gives end users more flexibility to use the structured data in an open-ended way.

5.1.2 Sloppy augmentation

"Sloppy programming" [20] tools like Chickenfoot [6] and Coscripter [19] enable users to create scripts that perform actions like filling in text boxes and clicking buttons, without directly interacting with the DOM. Users express the desired page elements in natural, informal terms (e.g. writing "the username box" to represent the textbook closest to the label "username"), and then using heuristics to determine which elements most likely match the user's intent. This approach allows for expressing a wide variety of commands with minimal training, but it also has downsides. It is difficult to know whether a command will consistently work over time (in addition to changes to the website, changes to the heuristics can also cause problems), and it is not easy for users to discover the space of possible commands.

Wildcard offers a sharp contrast to sloppy programming, instead choosing to expose a high degree of structure through the familiar spreadsheet table. Wildcard offers more consistency: for example, clicking a sort header will always work correctly as long as the site adapter is maintained. Wildcard also offers clearer affordances for what types of actions are possible, or, crucially, what actions are *not* possible, which is

useful to know. On the other hand, Wildcard cannot offer coverage of all websites, and has a narrower range of possible actions than sloppy tools. We expect that with enough site adapters and formulas, these downsides can be mitigated.

5.2 Spreadsheet-based app builders

Prior work has made the powerful realization that a spreadsheet can serve as an end-user-friendly backing data store and computation layer for an interactive web application. Research projects like Object Spreadsheets [21], Quilt [4], Gneiss [7], and Marmite [29], as well as commercial tools like Airtable Blocks [1] and Glide [9] allow users to view data in a spreadsheet table, compute over the data using formulas, and connect the table to a GUI. Because many users are already familiar with using spreadsheets, this way of creating applications tends to be far easier than traditional software methods; for example, in a user study of Quilt, many people were able to create applications in under 10 minutes, even if they expected it would take them many hours.

Wildcard builds on this idea, but applies it to modifying existing applications, rather than building new applications from scratch. For many people, we suspect that tweaking existing applications provides more motivation as a starting point for programming than creating a new application from scratch.

An important design decision for tools in this space is how to deviate from traditional spreadsheets like Microsoft Excel or Google Sheets. Quilt and Glide use existing spreadsheet software as a backend, providing maximum familiarity for users, and even compatibility with existing spreadsheets. Gneiss has its own spreadsheet implementation with additional features useful for building GUIs. Marmite provides a live data view that resembles a spreadsheet, but programming is actually done using a separate data flow pane rather than spreadsheet formulas. (Marmite's approach led to some confusion in a user study, because users expected behavior more similar to spreadsheets [29].) Airtable deviates the furthest: although the user interface resembles a spreadsheet, the underlying structure is a relational database with typed columns. Wildcard's table is most similar to Airtable; the structure of a relational table is most appropriate for most data in websites, and we have not yet found a need for arbitrary untyped cells.

5.3 Web scraping / data extraction

Web scraping tools focus on extracting structured data out of unstructured web pages. Web scraping is closely related to the implementation of Wildcard, but has different end goals: web scraping generally extracts static data for processing in another environment, whereas Wildcard modifies the original page by maintaining a bidirectional connection between the extracted data and the page.

Web scraping tools differ in how much structure they attempt to map onto the data. Some tools like Rousillon [8] extract data in a minimally structured relational format; other tools like Piggy Bank [14] more ambitiously map the data to a rich

semantic schema. In Wildcard, we chose to avoid schemas, in order to minimize the work associated with creating a site adapter.

In the future, we might be able to integrate web scraping tools to help create more reliable site adapters for Wildcard with less work, and to open up adapter creation to end users. Sifter was built on top of the Piggy Bank scraping library, suggesting precisely this type of architecture where web scraping tools are used to support interactive page modification.

6 Future work

Our prototype of Wildcard is still in early development; there are still many limitations to resolve and open questions to explore.

The most important question is whether the combination of features provided by Wildcard can be combined in enough useful ways to make the system worth using in practice. While initial demos are promising, we need to develop more site adapters and use cases to more fully assess this question. We plan to continue privately testing the system with our own needs, and to eventually deploy the tool publicly, once the API is stable enough and can support a critical mass of sites and use cases. We also plan to run usability studies to evaluate and improve the design of the tool.

Here are a few of the largest limitations in the current system:

- Wildcard only extracts data visible on the page, which means that subsequent pages of results in paginated lists are not included in the table. (Sifter [13] uses techniques that might help get around this.)
- There is no mechanism for end users to express imperative workflows (e.g. "click this button, and then..."); they can only write formulas that return data and then inject the resulting data into the page. While this makes the system simpler, it also may exclude many valuable use cases. We may add a system for expressing workflows like this, although it's not obvious how it would fit together with the existing table view. Gneiss [7] contains some mechanisms for writing spreadsheet formulas which can handle behaviors like reloading data in response to a user pressing a button, which might prove helpful.
- Wildcard's data model only shows a single table at a time, without any notion of relationships between tables. A richer data model with foreign keys might be necessary to support certain use cases. For designing a tabular interface on top of a richer data model, we could learn from the interface of Airtable which shows related objects in a table cell, or add nested rows as used in other systems [2, 21].
- Only allowing site adapters to be manually coded means that the number of supported sites is limited. We plan to explore abstractions that make it as easy as possible for programmers to efficiently create new robust adapters, as well as integrating automated heuristics into the adapter creation process.

7 Conclusion

We live in a time of pessimism about the effects of the internet and the degree of centralized control wielded by large corporations. We think that one promising approach to this problem is to enable people to have deeper control over the Web software they use every day. The Web offers the potential of an open foundation, but even the success of browser extensions has only empowered users within the limited confines of the ideas that programmers have decided to build.

In this paper, we have presented Wildcard, a browser-embedded programming system that maps websites to a structured data table, enabling end users to modify their behavior. We hope that it contributes to making the web into a more dynamic medium that users can mold to their own needs.

We plan to continue developing the system and to eventually deploy it as an open-source tool. To receive future updates on Wildcard and notifications about a public release, sign up for the email newsletter.

We are also looking for private beta testers. If you have an idea for how you might want to use Wildcard, please contact us. We would love to collaborate on building site adapters and formulas to support your use case.

Acknowledgements

References

[1] *Airtable: Organize Anything You Can Imagine*. https://airtable.com.

[2] Eirik Bakke and David R. Karger. "Expressive Query Construction through Direct Manipulation of Nested Relational Results". en. In: *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. San Francisco, California, USA: ACM Press, 2016, pages 1377–1392. ISBN: 978-1-4503-3531-7. DOI: 10.1145/2882903.2915210.

[3] 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. Palermo, Italy: ACM, 2000, pages 102–109. ISBN: 978-1-58113-252-6. DOI: 10.1145/345513.345267.

[4] Edward Benson, Amy X. Zhang, and David R. Karger. "Spreadsheet Driven Web Applications". en. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST '14*. Honolulu, Hawaii, USA: ACM Press, 2014, pages 97–106. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647387.

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

[6] 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.

[7] Kerry Shih-Ping Chang and Brad A. Myers. "Creating Interactive Web Data Applications with Spreadsheets". en. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST '14*. Honolulu, Hawaii, USA: ACM Press, 2014, pages 87–96. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647371.

[8] Sarah E. Chasins, Maria Mueller, and Rastislav Bodik. "Rousillon: Scraping Distributed Hierarchical Web Data". en. In: *The 31st Annual ACM Symposium on User Interface Software and Technology - UIST '18*. Berlin, Germany: ACM Press, 2018, pages 963–975. ISBN: 978-1-4503-5948-1. DOI: 10.1145/3242587.3242661.

[9] *Create an App from a Google Sheet in Minutes · Glide*. en. https://www.glideapps.com/.

[10] A. A diSessa and H. Abelson. "Boxer: A Reconstructible Computational Medium". en. In: *Communications of the ACM* 29.9 (Sept. 1986), pages 859–868. ISSN: 00010782. DOI: 10.1145/6592.6595.

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

[12] Andrew Hogue and David Karger. "Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web". en. In: *Proceedings of the 14th International Conference on World Wide Web - WWW '05*. Chiba, Japan: ACM Press, 2005, page 86. ISBN: 978-1-59593-046-0. DOI: 10.1145/1060745.1060762.

[13] David F. Huynh, Robert C. Miller, and David R. Karger. "Enabling Web Browsers to Augment Web Sites' Filtering and Sorting Functionalities". en. In: *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology - UIST '06*. Montreux, Switzerland: ACM Press, 2006, page 125. ISBN: 978-1-59593-313-3. DOI: 10.1145/1166253.1166274.

[14] David Huynh, Stefano Mazzocchi, and David Karger. "Piggy Bank: Experience the Semantic Web Inside Your Web Browser". en. In: *The Semantic Web – ISWC 2005*. Edited by Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pages 413–430. ISBN: 978-3-540-32082-1. DOI: 10.1007/11574620_31.

[15] Hypercard. "HyperCard". en. In: *Wikipedia* (Dec. 2019). Page Version ID: 931376685.

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

[17] A. Kay and A. Goldberg. "Personal Dynamic Media". In: *Computer* 10.3 (Mar. 1977), pages 31–41. ISSN: 1558-0814. DOI: 10.1109/C-M.1977.217672.

[18] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. "Webstrates: Shareable Dynamic Media". en. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*. Daegu, Kyungpook, Republic of Korea: ACM Press, 2015, pages 280–290. ISBN: 978-1-4503-3779-3. DOI: 10.1145/2807442.2807446.

[19] 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. Florence, Italy: ACM, 2008, pages 1719–1728. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357323.

[20] Greg Little, Robert C. Miller, Victoria H. Chou, Michael Bernstein, Tessa Lau, and Allen Cypher. "Sloppy Programming". en. In: *No Code Required*. Elsevier, 2010, pages 289–307. ISBN: 978-0-12-381541-5. DOI: 10.1016/B978-0-12-381541-5.00015-8.

[21] Matt McCutchen, Shachar Itzhaky, and Daniel Jackson. "Object Spreadsheets: A New Computational Model for End-User Development of Data-Centric Web Applications". en. In: *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - Onward! 2016*. Amsterdam, Netherlands: ACM Press, 2016, pages 112–127. ISBN: 978-1-4503-4076-2. DOI: 10.1145/2986012.2986018.

[22] Brad Myers, Scott Hudson, and Randy Pausch. "Past, Present, and Future of User Interface Software Tools". In: *ACM Trans. Comput.-Hum. Interact.* 7 (Mar. 2000), pages 3–28. DOI: 10.1145/344949.344959.

[23] Bonnie A. Nardi and James R. Miller. "An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development". In: ACM Press, 1990, pages 197–208.

[24] Bonnie A. Nardi and James R. Miller. "Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development". en. In: *International Journal of Man-Machine Studies* 34.2 (Feb. 1991), pages 161–184. ISSN: 00207373. DOI: 10.1016/0020-7373(91)90040-E.

[25] Mitchel Resnick. *Designing for Wide Walls*. en. Aug. 2016.

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

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

[28] Bret Victor. *Dynamicland*. https://dynamicland.org/.

[29] Jeffrey Wong and Jason I. Hong. "Making Mashups with Marmite: Towards End-User Programming for the Web". en. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '07*. San Jose, California, USA: ACM Press, 2007, pages 1435–1444. ISBN: 978-1-59593-593-9. DOI: 10.1145/1240624.1240842.

About the authors

Geoffrey Litt is a PhD student at MIT, researching programming tools for end users and developers.

Daniel Jackson is a professor in the Department of Electrical Engineering and Computer Science at MIT, associate director of CSAIL, and a MacVicar Fellow. He leads the Software Design Group.