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 Web applications do not meet the precise 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, a live programming system embedded in the browser that empowers users to casually tweak web applications without needing to program in Javascript or interact with the DOM.

Wildcard shows the 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, users 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 helps make the web into a more malleable medium for all 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

© Geoffrey Litt and Daniel Jackson
This work is licensed under a “CC BY 4.0” license.
Submitted to *The Art, Science, and Engineering of 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 a motivated, programmer 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. In *Personal Dynamic Media* [17], Alan Kay 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, a live programming system embedded in the browser that enables users to tweak web applications without programming in Javascript or interacting with the DOM. 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, and using alternate UI widgets to control a page. While Wildcard does not support any change someone might want to make to a website, it makes a variety of common and useful changes more accessible to end users, as well as to programmers who want to avoid tedious low-level programming.

Under the hood, the implementation of Wildcard is straightforward, because a programmer must manually write an adapter for each website which uses common web scraping techniques to map the data from the web page into the Wildcard table format. Although there is still manual programming involved, the result is very different from traditional single-use browser extensions, because 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.

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

- *Expose consistent structure*: turning isolated applications into a structured medium
- *Encourage casual tweaking*: using frequent software modification as a starting point for end user programming
- *Build for an ecosystem*: designing for first party developers, third party programmers, and non-programmer end users

We also explore related work on web augmentation tools, spreadsheet-based app builders, and web scraping tools. Our novel contribution in this paper is to combine ideas from each of these areas into a new kind of tool: a programming environment which maps data from web pages onto a live table view, enabling end users to modify the data.

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 mundane and ubiquitous as using spreadsheets.

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 doesn't let us sort by price, but we can 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.

When we click the Price column header in the table, both the table and the original page become sorted by price. We can also filter listings by rating, another feature not offered in the Airbnb site.

After finishing the sorting and filtering, we can close the table view and continue using the website in its original polished design.

2.2 Row actions

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.

Within the site adapter, each action is implemented as a regular Javascript function running in the context of the page, which can click on buttons in the UI, launch AJAX requests, navigate to a new page, etc.

2.3 User annotations

It's a common practice in spreadsheets to add a column next to a table for recording notes about the item in each row. In Wildcard, this pattern can be used to create private annotations in a website. Here, we use this feature to jot down pros and cons of various listings:

In order to match the annotations with the existing design, the site adapter specifies a reasonable location in each row for injecting new data.

Annotations are persisted in browser local storage so the user can come back and view them later. The site adapter associates each table row with a stable identifier from the original site so that annotations can be displayed with the appropriate item.

Currently annotations are private to the browser of the user that created them, but it could be useful in the future to allow users to share annotations.

2.4 Computation with formulas

The demos so far have shown straightforward tweaks that provide useful conveniences with little effort. But Wildcard also supports more sophisticated tweaks that fetch external data and perform computations, using a formula language.

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 can rate 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 score as the cell value. Because the cell is shown in the page, the score also shows up in the page body.

Wildcard formulas are just Javascript functions that take table data as input and return new data. Formulas can have limited side effects like accessing external APIs to fetch data, but do not directly manipulate the page. Instead, they return data into the table, which is then injected into the page with the same mechanism used for injecting user annotations.

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 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 (it's not allowed to change the name of an Airbnb listing), but in this case, the cells are marked as writable, so that changes in the table are reflected in the original form. This becomes particularly useful when combined with custom cell editing UIs, which 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. Filling in dates for a flight search typically requires opening up a separate calendar app to look up the correct dates, and then manually copying them into the form. In Wildcard,

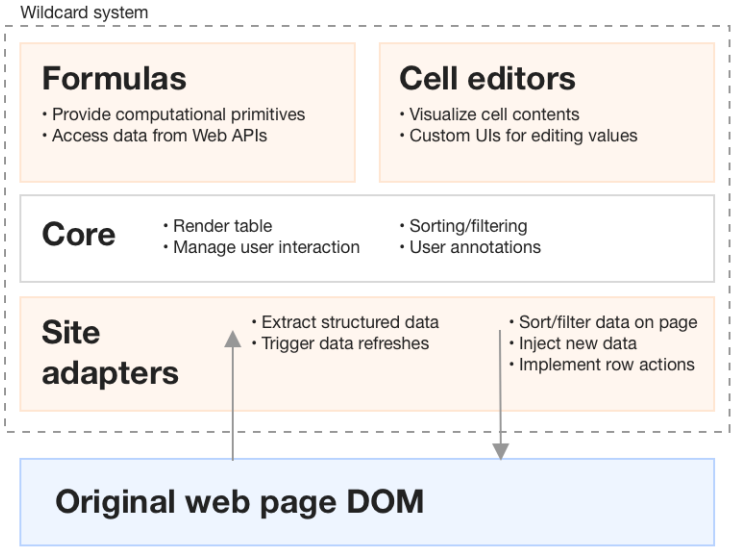

Figure 1 The architecture of the Wildcard system

we can make this easier by editing the date cell using a datepicker widget that has privileged access to our calendar information.

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 it to the website server.

Overall, an interactive data table is a computation model that presents a surprisingly large range of useful possibilities for end user modification of websites, while also remaining familiar and easy to use. While we've presented concretely illustrated here just a few 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. 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 name and some metadata properties, and then a function that extracts the name div from a row div.

```
{
  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) which runs Javascript code that can mutate the DOM or perform other arbitrary behaviors.

So far we have only implemented three site adapters in Wildcard, but we plan to implement many more in the near future to continue to test the limits of the API. Eventually we will also encourage third party programmers to build and maintain adapters for popular sites.

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 this specific system and can more generally inform the design of end user programming tools, particularly systems enabling users to modify GUI apps.

4.1 Expose consistent structure

In *Changing Minds* [11], 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. An ordered numerical sequence allows the user to intuitively grasp the structure of the system—"if I shift up, it gets harder to pedal"—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?

Today, most personal computing consists of using distinct apps, each one an isolated silo of concepts, interfaces, data, and functionality. While there are some shared

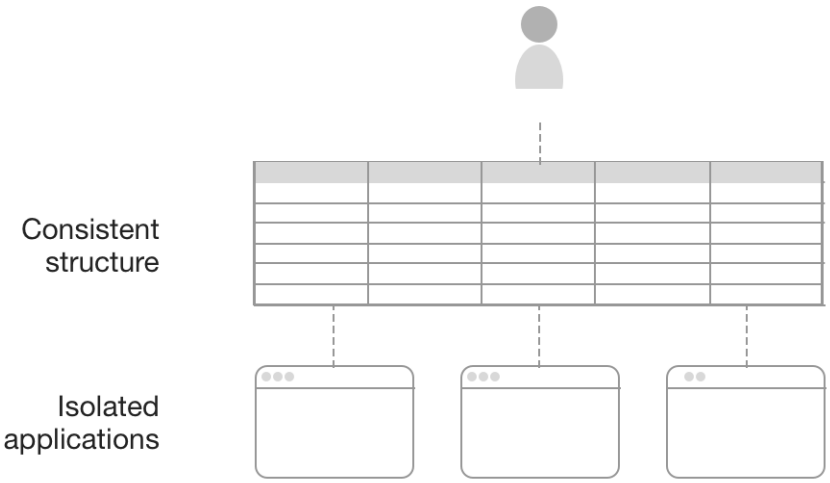

Figure 2 Wildcard layers the consistent structure of a data table over different isolated applications.

idioms and limited points of interoperability, the fundamental expectation is that each app operates independently. If a new need emerges, a user must hope "there's an app for that"; creating a new tool to meet the need is often out of the question.

Computing does not need to operate this way. As one example, UNIX offers a compelling alternate vision of small tools connected through a common abstraction of text files. Just like riding a bike with numbered gears, UNIX requires users to build up a structured understanding of the system, which then allows them to flexibly react to new needs not served by existing tools. Users can also deeply master a text editor and 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 can be used in many contexts. diSessa [11] notes 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; this is the situation in software today.

With Wildcard, we hope to make isolated Web applications work a little bit more like UNIX, by imposing a consistent and simple structure onto many isolated 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 more structured data to users provides improved quality and consistency of user interfaces. 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 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.

Layering a structured view on top of an existing application is not the only option available. 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. We also think that sloppy programming does little to solve the problem of thinking of applications as isolated worlds, whereas Wildcard contributes to giving users a structured understanding of the different apps that they use.

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 Encourage casual tweaking

In order to serve the widest possible audience, Wildcard is designed for tweaking existing software for one's own needs, not for designing a new application from scratch. Creating a new application requires motivation and other skills besides programming that not everyone has, but having frustrations with some aspect of an app which could be solved by a small tweak is a nearly universal experience.

	Casual	*Not casual*
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, 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" [16]. Wildcard also provides live feedback, immediately editing the page in response to user interaction, which gives users confidence in the changes they are making.

Wildcard also aims to make some valuable changes possible with particularly low effort: for example, sorting a table in a single click, or adding an annotation by filling in 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. This property is inspired by spreadsheets, which can be used productively even by someone who has learned only a tiny part of their functionality, e.g. for storing tables of numbers or computing simple sums [22]. In contrast, many traditional programming systems require someone to spend days learning complex concepts before being able to create a useful piece of software—printing "hello world" is not a genuinely valuable activity.

4.3 Build for an ecosystem

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 [21]. 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 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 [23, 24]. 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

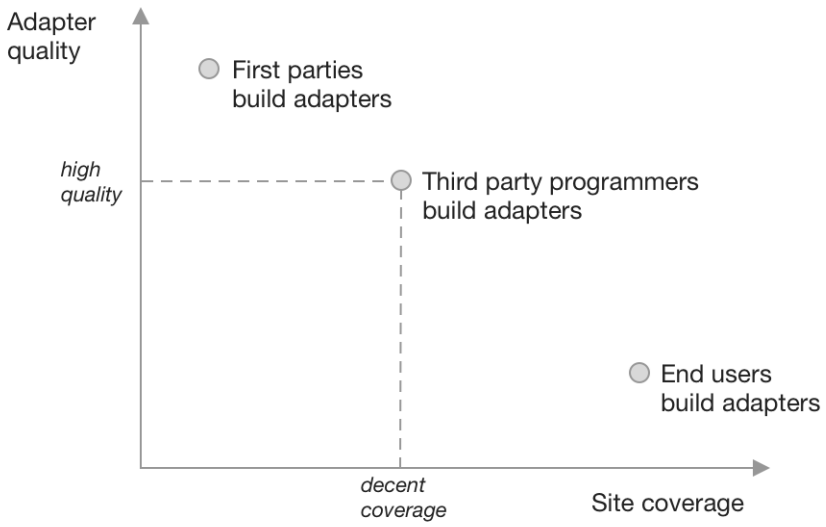

Figure 3 Having third party programmers create site adapters balances site coverage and adapter quality.

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, a notion Cory Doctorow calls adversarial interoperability. 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. However, we also 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, and would not necessarily require too much effort—we envision making it straightforward to add Wildcard support in a client-side application which already has 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 [23, 24], 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 [25]. (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 in the future and becoming more similar 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 which 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

It is a powerful realization to notice that a spreadsheet can serve as an end-user-friendly backing data store and computation layer for an interactive web application. Research projects like Quilt [4], Gneiss [7], and Marmite [26], 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 much 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 powerful 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. Quilt and Glide use existing web spreadsheet software as a backend, providing maximum familiarity and 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 [26].) Airtable deviates the furthest from spreadsheets: although the user interface has formulas, shows live feedback and is easy to use, 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 Limitations and future work

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 SIEUFERD [2] which uses nested rows.
- Only allowing site adapters to be manually coded means that the number of supported sites is limited. As mentioned previously, we may explore also building wrappers using automated heuristics or end user scraping tools.

7 Conclusion

We live in a time of pessimism about the effects of the internet, especially given the degree of centralized control that large corporations wield. We think that one promising solution is to enable more people to have deeper control over the Web software they use every day. The Web offers a promising opportunity 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] Bonnie A. Nardi and James R. Miller. “An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development”. In: ACM Press, 1990, pages 197–208.

[22] 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.

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

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

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

[26] 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.