Wildcard: Spreadsheet-Driven Customization of Web Applications

Geoffrey Litt
Massachusetts Institute of Technology
Cambridge, MA, USA
glitt@mit.edu

Daniel Jackson
Massachusetts Institute of Technology
Cambridge, MA, USA
dnj@mit.edu

ABSTRACT

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

In this paper, we present *spreadsheet-driven customization*, a technique that enables end users to customize software without doing any traditional programming. The idea is to augment an application's UI with a spreadsheet that is synchronized with the application's data. When the user manipulates the spreadsheet, the underlying data is modified and the changes are propagated to the UI, and vice versa.

We have implemented this technique in a prototype browser extension called Wildcard. Through concrete examples, we demonstrate that Wildcard can support useful customizations—ranging from sorting lists of search results to showing related data from web APIs—on top of existing websites. We also present the design principles underlying our prototype.

Customization can lead to dramatically better experiences with software. We think that spreadsheet-driven customization offers a promising new approach to unlocking this benefit for all users, not just programmers.

CCS CONCEPTS

• **Software and its engineering → Integrated and visual development environments**.

KEYWORDS

end-user programming, software customization, web browser extensions

ACM Reference Format:
Geoffrey Litt and Daniel Jackson. 2020. Wildcard: Spreadsheet-Driven Customization of Web Applications. In *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming (<Programming'20> Companion), March 23–26, 2020, Porto, Portugal*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397537.3397541

<Programming'20> Companion, March 23–26, 2020, Porto, Portugal

ACM ISBN 978-1-4503-7507-8/20/03.
https://doi.org/10.1145/3397537.3397541

1 INTRODUCTION

Web applications often don't meet the precise needs of their users. Sometimes there is a browser extension available to patch an issue, and if the user is a programmer they might be able to fix it themselves. But for most people, the only recourse is to complain to the developers, or, more likely, to simply give up. Back in 1977, in *Personal Dynamic Media* [16], Alan Kay envisioned personal computing as a medium that let a user "mold and channel its power to his own needs," but today, software behaves more like concrete than clay.

In this paper, we present *spreadsheet-driven customization*, a technique that enables end users to customize software without doing any traditional programming. The idea is to augment an application's UI with a spreadsheet: a data table view, synchronized live with the application's data. When the user manipulates the spreadsheet, the underlying data is modified and the changes are propagated to the UI, and vice versa.

We have implemented this technique in a prototype browser extension called Wildcard and used it to build demos which suggest that this paradigm can support useful customizations, ranging from sorting lists of data to adding whole new features to applications (shown in Section 2).

Our approach requires extracting structured data from the user interfaces of existing applications, but we hide the complexity of data extraction from end users. Programmers write *site adapters* which use web scraping techniques to extract structured data from existing applications and map it to the spreadsheet table. Our prototype suggests that it is possible to implement site adapters for real websites; Section 3 describes some of the techniques and challenges involved.

Spreadsheet-driven customization is based on three design principles, described in Section 4:

- **Expose a universal structure**: By mapping the data in all applications to the same universal table abstraction, Wildcard enables users to learn a generic customization toolkit that they can apply to any site.
- **Low floor, high ceiling**: Wildcard provides an easy entry point for end users, since small tweaks like sorting data can be performed with a single click. At the same time, it also supports a variety of richer customizations, like adding private annotations to a webpage or joining in related data from a web API.
- **Design for an ecosystem of users**: Spreadsheet-driven customization combines the efforts of programmers and end users, rather than putting the full burden on end users. We also envision a role for first party developers to help make their applications more customizable.

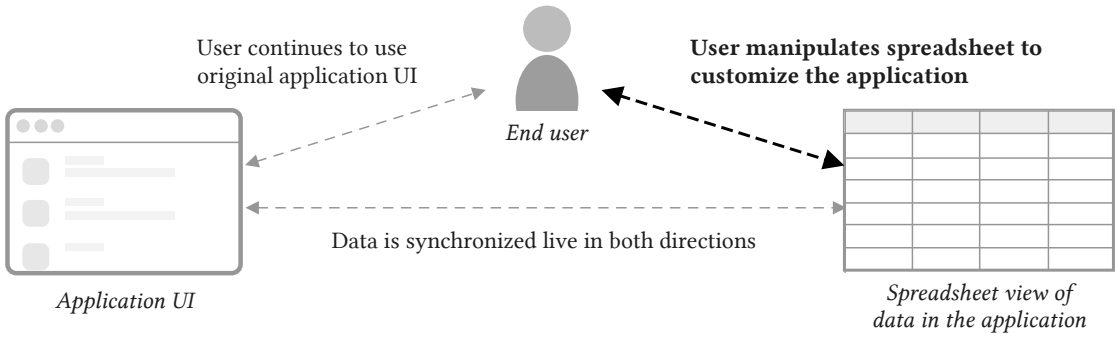

Figure 1: An overview of spreadsheet-driven customization

Prior work [6, 8, 21] has enabled end users to create "spreadsheet-driven applications" which use spreadsheets as a backing data layer. Spreadsheet-driven *customization* applies this idea in a different context: customizing existing software, rather than building new software from scratch. Our technique does not require that the application actually be backed by a spreadsheet; it merely uses the spreadsheet as an interface for viewing and modifying the internal state of the application. In Section 5, we describe further how Wildcard relates to existing work on spreadsheet-driven apps, as well as work in other areas like web customization and web scraping.

The Wildcard extension is currently an early research prototype. We plan to continue testing the system with our own use cases to explore how the spreadsheet abstraction maps to real websites and customization needs. Eventually we plan to release the tool publicly in order to learn from real use cases, discover usability challenges, and to test the feasibility of building and maintaining site adapters. Section 6 describes remaining open questions and our future plans.

2 DEMOS

Here are some examples of using Wildcard to customize websites in useful ways.

2.1 Augmenting Search Results

In 2012, the travel site Airbnb removed the ability to sort accommodation searches by price. Users could still filter by price range, but could no longer view the cheapest listings first. Many users complained 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.

Using Wildcard, the user can fix this omission, while leaving the page's design and the rest of its functionality unchanged. Figure 2 shows an overview of augmenting the Airbnb site. 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 to indicate the mapping between the views.

Then, the user clicks on the price column header to sort the spreadsheet and the Airbnb UI by price (Figure 2, Note A). They also filter to listings with a user rating above 4.5 (another feature missing in the original Airbnb UI).

After manipulating the data, the user closes the table view and continues using the website. Because the application's UI usually has a nicer visual design than a spreadsheet, Wildcard does not aim to replace it—at any time, the user can use either the UI, the spreadsheet, or both together.

Many websites that show lists of data also offer actions on rows in the table, like adding an item to a shopping cart. Wildcard has the ability to make these "row actions" available in the data table through the site adapter. In the Airbnb UI, saving multiple listings to a Favorites list requires tediously clicking through them one by one. Using Wildcard row actions, the user can select multiple rows and favorite all of them with a single click (Figure 2, Note B).

Next, the user wants to jot down some notes about each listing. To do this, they type some notes into an additional column in each row, and the notes appear inside the listings in the original UI (Figure 2, Note C). The annotations are saved in the browser's local storage and associated with the backend ID of the listing, so they will appear in future browser sessions that display the same listing.

Wildcard also includes a formula language that enables more sophisticated customizations. When traveling without a car, it's useful to evaluate potential places to stay based on how walkable the surroundings are. Using a formula, the user can integrate Airbnb with Walkscore, an API that rates the walkability of any location on a 1-100 scale. When the user calls the `walkscore` formula with the listing's latitude and longitude as arguments, it returns the walk score for that location and shows it as the cell value. Because the cell's contents are injected into the page, the score also appears in the UI (Figure 2, Note D). Unlike traditional spreadsheets, formulas automatically apply to every row in the table.

2.2 Snoozing Todos

In addition to fetching data from other sources, Wildcard formulas can also perform computations. In this example, the user would like to augment the TodoMVC todo list app with a "snooze" feature, which will temporarily hide a todo from the list until a certain date. Figure 3 shows an overview of this customization.

The user opens the table view, which shows the text and completed status of each todo. They start the customization by adding

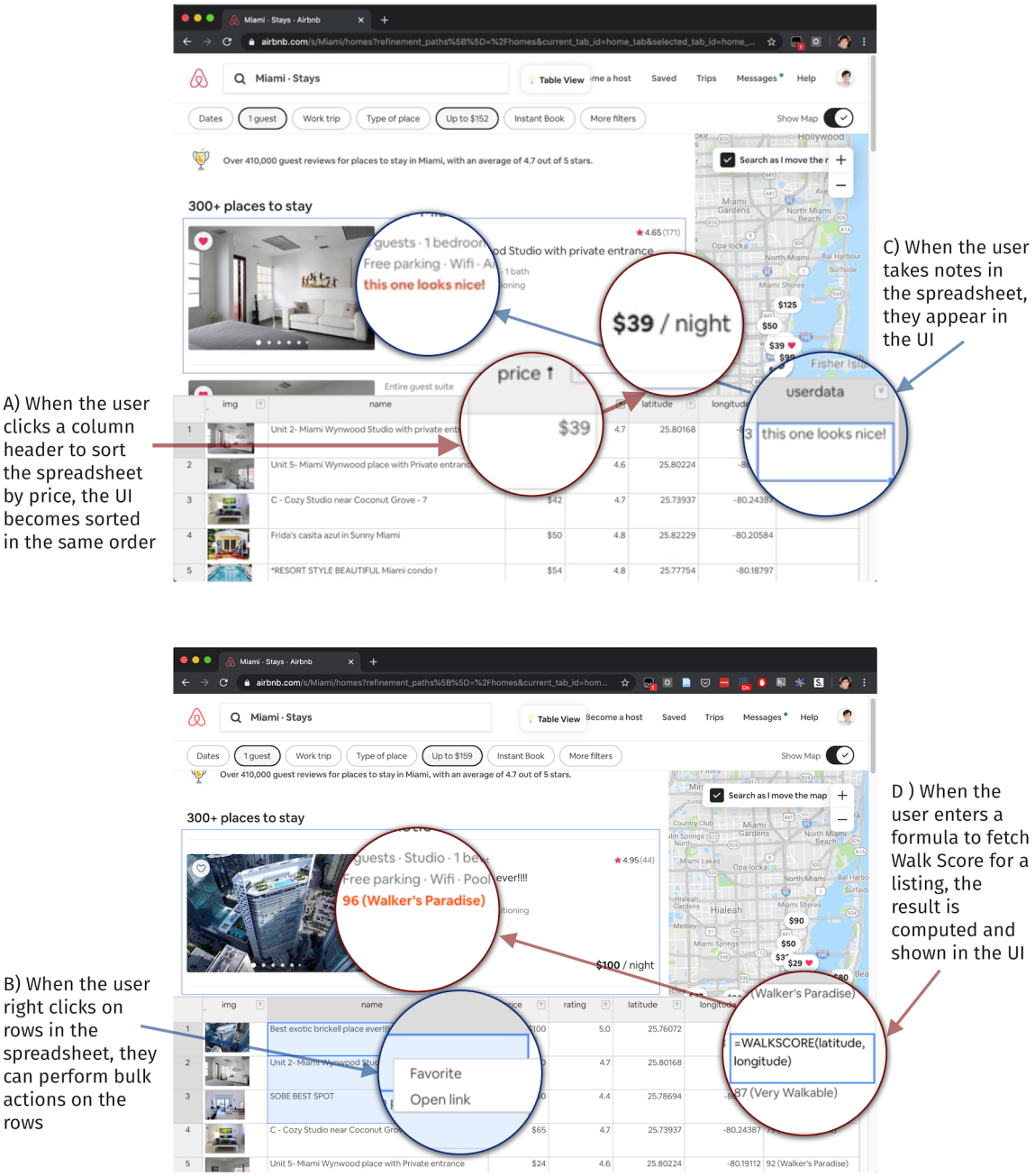

Figure 2: Using Wildcard to augment the Airbnb search page for booking accommodations

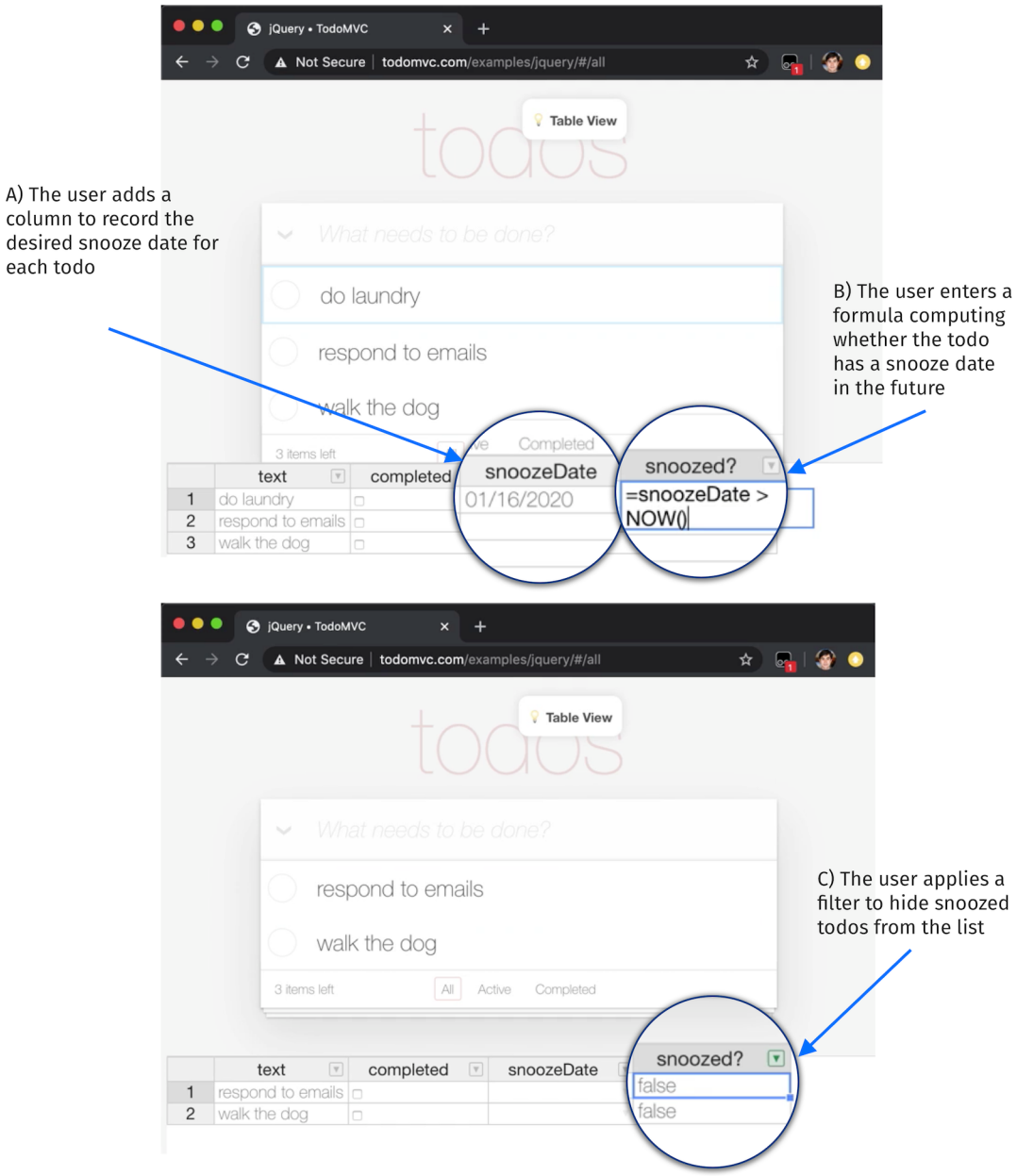

Figure 3: Using Wildcard to add a "snooze" feature to the TodoMVC todo list app

a new column to store the snooze date for each todo (Figure 3, Note A).

The next step is to hide snoozed todos. The user creates a snoozed? column, which uses a formula to compute whether a todo is snoozed—i.e., whether it has a snooze date in the future (Figure 3, Note B). Then, they simply filter the table to hide the snoozed todos (Figure 3, Note C).

Because the built-in `NOW()` function always returns the current datetime, snoozed todos will automatically appear once their snooze date arrives.

Because this implementation of snoozing was built on the spreadsheet abstraction, it is completely decoupled from this particular todo list app. We envision that users could share these types of customizations as generic browser extensions, which could be applied to any site supported by Wildcard with no additional effort.

2.3 Adding a Custom Datepicker

It might seem that Wildcard is only useful on websites that display lists of tabular data, but the table metaphor is flexible enough to represent many types of data. For example, a flight search form on Expedia can be represented as a single row, with a column corresponding to each input (Figure 4, Note A).

In some of the previous examples, the table cells were read-only (because users can't, for example, change the name or price of an Airbnb listing). In this case, the cells are writable, which means that changes in the table are reflected in the form inputs. This becomes especially useful when combined with GUI widgets that can edit the value of a table cell.

Filling in dates for a flight search often requires a cumbersome workflow: open up a separate calendar app, find the dates for the trip, and then carefully copy them into the form. In Wildcard, the user can avoid this by using a datepicker widget that shows the user's personal calendar events (Figure 4, Note B). The user can directly click on the correct date, and it gets inserted into both the spreadsheet and the original form.

In this section we've presented just a few use cases for spreadsheet-driven customization, to suggest some possibilities of the paradigm. We think the spreadsheet model is flexible enough to support a wide range of other useful modifications while remaining familiar and easy to use.

3 SYSTEM IMPLEMENTATION

Wildcard is written in Typescript, and built as a cross-browser extension for Chrome, Firefox, and Edge.

Unlike the freeform 2D grid of traditional spreadsheets, Wildcard represents data in a relational table with named and typed columns. We still refer to our view as a spreadsheet because it includes a data table UI and reactive formulas, two essential ingredients of the spreadsheet interaction model.

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

Site adapters are a key part of Wildcard, since they specify the bidirectional connection between the web page and its structured data representation. Wildcard provides an API for programmers to concisely express how the DOM should be mapped to this structure.

Sometimes, more advanced scraping techniques are necessary to extract data from the application. For example, we have prototyped a mechanism for site adapters to observe AJAX requests made by the browser and extract data directly from JSON responses. This mechanism was used to implement the Airbnb Walkscore example, since latitude and longitude aren't shown in the Airbnb UI, but they are available in AJAX responses. This technique seems promising because AJAX responses already contain data in a structured form, and web applications are increasingly loading data using AJAX. Another technique we might consider is enabling site adapters to scrape data across multiple pages for paginated lists of results (as explored in [14]).

Because Wildcard requires a live bidirectional connection between the UI and the structured data, there are some additional challenges beyond those typically associated with web scraping.

First, the site adapter needs to support sending updates from the table to the original page. We have made this fairly straightforward, because most DOM manipulation is not performed directly by the site adapter. Instead, Wildcard automatically mutates the DOM to reflect the spreadsheet state, using the same specification that was used to extract the data. The only exception to this is row actions (like favoriting an Airbnb listing), which are implemented as Javascript functions that can perform arbitrary behaviors like simulating clicks on buttons or mutating the DOM.

Another challenge is triggering updates to the spreadsheet data in response to UI changes that happen after initial page load. Site adapters are responsible for recognizing these changes by observing the DOM. So far, we have been able to use event listeners and the MutationObserver API to successfully observe changes, but it may prove challenging to observe changes on some sites only through the DOM.

4 DESIGN PRINCIPLES

The idea of spreadsheet-driven customization is guided by three design principles, inspired by prior work and our own experimentation. We think these principles might also broadly inform the design of tools for end user software customization.

4.1 Expose a Universal Data Structure

Today, most personal computing consists of using applications, which bundle together behavior and data to provide some set of functionality. While there are some limited points of interoperability, applications generally are designed to operate independently of one another.

Computing does not need to be organized this way. For example, UNIX offers a compelling alternative: many small single-purpose programs, all of which manipulate a universal format of text streams. Users get high leverage from mastering a text editor and other text manipulation tools, since they can be applied to a huge variety of tasks. A user's preferred text editor can even serve as an interactive input mechanism in shell programs, e.g. for editing git commit messages.

Spreadsheet-driven customization aims to bring some of this UNIX ethos to the world of isolated Web applications by creating a consistent data structure to represent the data inside many applications. In UNIX, the universal format is a text stream; in Wildcard, it is a relational table. Because Wildcard maps the data from all applications to the table format, users can master the tools provided by Wildcard and apply them to customize many different applications.

This idea relates to Beaudouin-Lafon and Mackay's notion of *polymorphic interface instruments* [5], which are UI elements that can be used in different contexts (for example, a color picker that can be used in many drawing applications). It also relates to ideas of literacy in a medium. DiSessa notes that textual literacy depends on the fact that writing can be adapted to many different genres [10]: if people needed to relearn reading and writing from scratch when switching from essays to emails, the medium would lose its potency. Generic tools are especially important for software

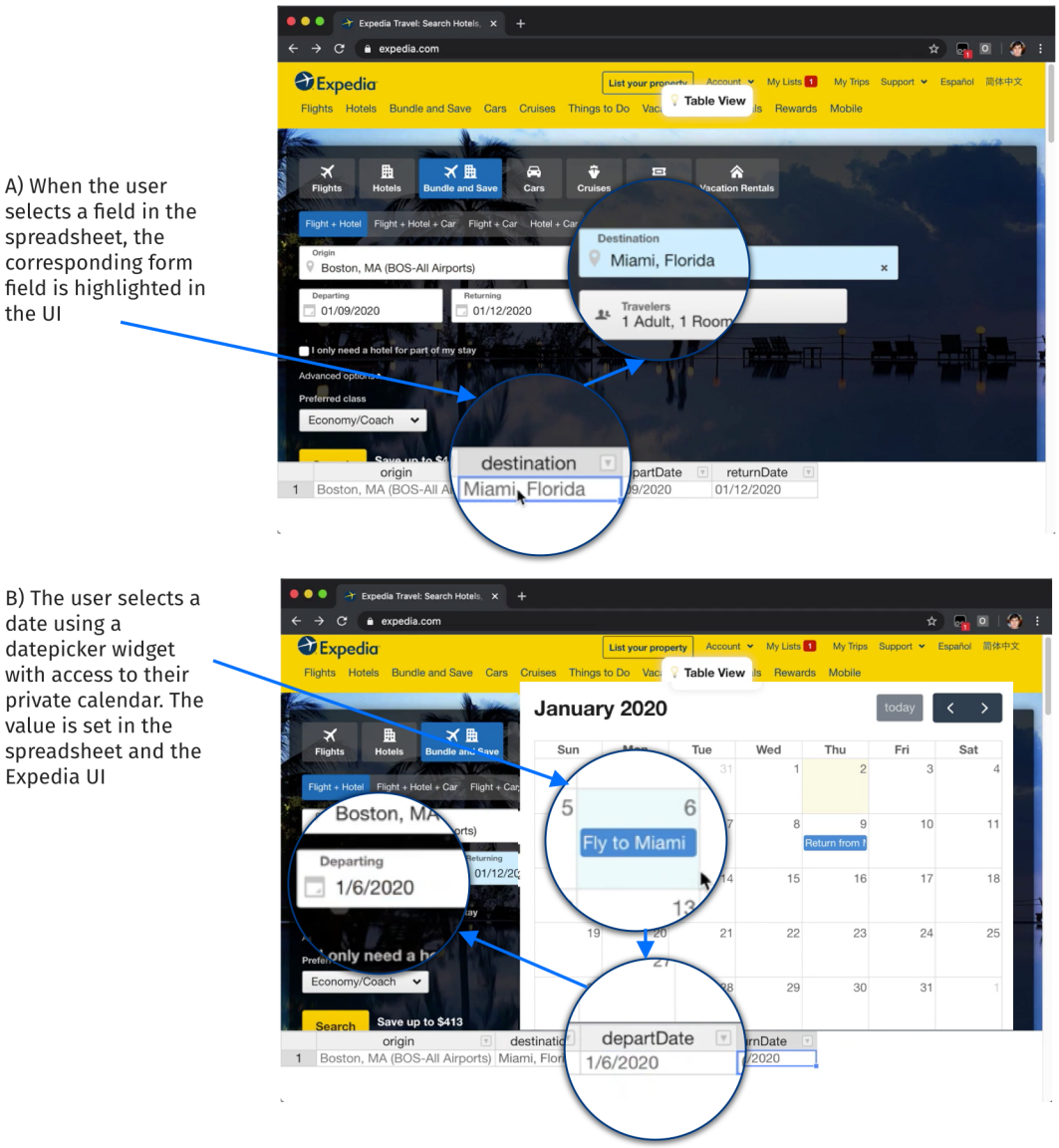

Figure 4: Using Wildcard to augment the Expedia page for booking a flight

customization, because a common barrier to customizing software is not having enough time [20]—it's more likely that people will customize software frequently if they can reuse the same tools across many applications.

This design principle leads to several challenges. First, any universal abstraction has its limits, and may not naturally express the data in every application. We plan to explore the limits of the table abstraction further by trying to build adapters for more sites with varied data formats. We expect that many types of data will fit easily into tables: lists of search results, news articles, and messages can all naturally be seen as relations. Documents (e.g. Google Docs) or graphs (e.g. social network friend graphs) may prove more challenging to map to the table abstraction.

4.2 Low Floor, High Ceiling

Seymour Papert advocated for programming systems to have a "low floor," making it easy for novices to get started, and a "high ceiling," providing a large range of possibilities for more sophisticated users [24]. Our goal is for spreadsheet-driven customization to meet both of these criteria.

One of the most interesting properties of spreadsheets is that users familiar with only a tiny sliver of their functionality (e.g., storing tables of numbers and computing simple sums) can still use them in valuable ways. This supports the user's natural motivation to continue using the tool, and to eventually learn its more powerful features if needed [23]. In contrast, many traditional programming

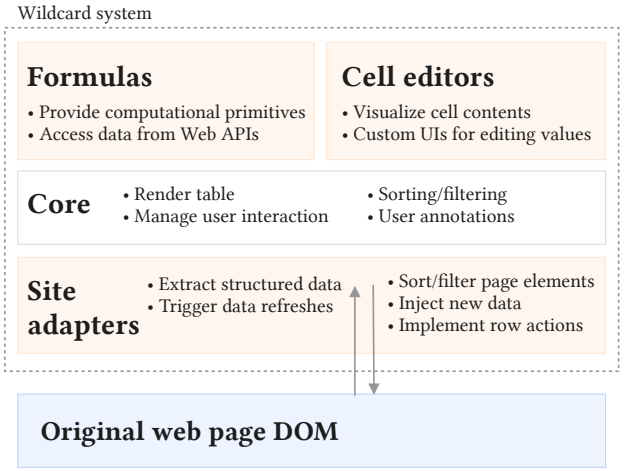

Figure 5: The architecture of the Wildcard prototype

systems require an enormous upfront time investment before someone is able to write a program that helps them achieve a useful task.

As part of ensuring a low floor, we have focused on including features that have immediate value for novices. For example, a user can sort a table with a single click, or simply type in some annotations. We would expect many Wildcard users to start by using these simpler features before moving on to more sophisticated features like formulas.

Another aspect of providing a low floor is providing an "in-place toolchain" [1]: minimizing the effort of moving from using to customizing by making customization tools available in the same environment where the user is already using the software. This quality is distinct from the level of technical skill needed to use the tool. For example, setting up a workflow in an end user programming system like IFTTT does not require much programming skill, but it does require leaving the user's normal software and entering a separate environment; conversely, running a Javascript snippet in the browser console requires programming skills, but can be done immediately in the flow of using a website.

Wildcard provides an in-place toolchain because the spreadsheet can be instantly opened in the browser window while using any supported website. Once the user starts editing, Wildcard also provides live feedback, so that even if a user isn't yet totally familiar with Wildcard, they can learn to use the system through experimentation.

Since we have only built a few site adapters and demos so far, it is still too early to tell how high the ceiling is for the customizations that can be achieved with Wildcard. But we think that with enough built-in formulas (both standard spreadsheet formulas for computation, and formulas for fetching data from other sources), the language could support a wide variety of customizations. We plan to explore this aspect further by trying to solve real problems and observing where limitations emerge in practice (discussed more in Section 6).

4.3 Design For an Ecosystem of Users

Real-world spreadsheet usage is highly collaborative. Many users only perform simple changes, while their coworkers help with writing complex formulas or programming macros [22]. Inspired by this collaborative dynamic, we aim for spreadsheet-driven customization to bring together the abilities of different users.

The main way we do this is by separating website customization into two separate stages: structured data extraction, performed by programmers who code site adapters in Javascript, and then customization using the resulting spreadsheet, which is available to all end users. This architecture frees end users from needing to think about data extraction, and enables a community of end users to reuse the efforts of programmers building site adapters.

The group of users building adapters does not necessarily need to be limited only to programmers. In the future, we plan to explore enabling end users to also create site adapters, drawing on related work in this area [9, 14]. But even then, we still envision a separation between highly motivated, tech-savvy end users building adapters, and more casual end users just using the spreadsheet view.

The first party developers of the original software can also play a role in enabling customization. Although spreadsheet-driven customization does not depend on cooperation from first party website developers, their participation in exposing structured data would eliminate the need for third party site adapters. The first parties could themselves benefit from such a system; providing Wildcard support—a fairly low-effort undertaking for a developer with direct access to the structured data—would allow users to build extensions to fulfill some of their own feature requests. There is also precedent for first parties implementing an official client extension API: for several years, Google maintained an official extension API in Gmail for Greasemonkey scripts to use.

Google no longer maintains its own API, but third parties have continued to maintain successful replacements—Gmail.js [25] is an open source wrapper API with over 70 contributors and hundreds of commits. This library demonstrates another benefit of designing for an ecosystem: it's easier for many programmers to collectively maintain a generic site adapter that can support many extensions, rather than each programmer creating custom scraping logic for their own single-purpose extension.

5 RELATED WORK

Spreadsheet-driven customization relates to existing work in four areas: malleable software, web customization, spreadsheet-driven applications, and web scraping.

5.1 Malleable Software

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 [16], Hypercard [15], Boxer [11], Webstrates [17], and Dynamicland [26]. In fact, the name Wildcard comes from the internal pre-release name for Hypercard, a project which doubly inspired our work by promoting both software modification by end users and the ideas behind the Web.

Although Wildcard shares broad goals with these projects, it employs a different solution strategy. These other systems generally require building software from scratch in a new environment. On the other hand, Wildcard aims to maximize the malleability of software built with existing tools.

With substantial future work, we think Wildcard could become more similar to these other projects, growing from a platform for tweaking existing software into a platform for building new software from scratch. This would likely end up resembling existing tools for building spreadsheet-driven applications (discussed more below), but with an additional focus on customization by end users of the software.

5.2 Web Customization

Wildcard's goals are closely shared with other systems that provide interfaces in the browser for end users to customize websites.

5.2.1 Structured Augmentation. Wildcard's solution approach is most similar to other tools that identify structured data in a web page and then use that structure to support end user customization of the page.

Sifter [14] enables users to sort and filter lists of data on web pages, which resembles Wildcard's sorting functionality. The underlying mechanism is similar, since Sifter extracts structured data from the page to enable its user-facing functionality. However, the systems also have significant differences:

- Wildcard hides data extraction from end users by having programmers develop site adapters, whereas Sifter uses a combination of automated heuristics and interactive user feedback to involve end users in extracting data. User studies of Sifter indicated that users were surprised by the need to participate in the data augmentation process, which influenced our decision to hide this process from the user. We suspect Wildcard's workflow results in a simpler end user experience, but at the cost of only working with a smaller subset of supported websites.
- Wildcard supports a broad set of customizations. Sifter only supports sorting and filtering.
- Wildcard shows the structured data table directly to the user. Sifter only shows sorting controls, and does not reveal the underlying data table.

Thresher [12] enables users to create wrappers which map unstructured website content to Semantic Web content. Like Wildcard and Sifter, Thresher augments the page: 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 focus on complementary parts of the customization process. Thresher aims to enable end users to create content wrappers, but the actions available on the structured data are determined in advance. Conversely, Wildcard delegates wrapper creation to programmers, but gives end users more flexibility to use the structured data in an open-ended way.

5.2.2 Sloppy Augmentation. "Sloppy programming" [19] tools like Chickenfoot [7] and Coscripter [18] enable users to create web automation scripts without directly interacting with the DOM. The scripts can perform actions like filling in text boxes and clicking on buttons. Users express the desired page elements in natural, informal terms (e.g. writing "the username box" to represent the textbox closest to the label "username"), and then the system uses a set of 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 [19]:

- *Reliability*: It can be difficult to know whether a command will consistently work over time. Changes to the website or to the system's heuristics can cause unexpected changes in behavior.
- *Discoverability*: it can be difficult for users to explore the space of possible commands.

Wildcard offers a sharp contrast to sloppy programming, instead choosing to expose a high degree of structure through the 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. On the other hand, Wildcard's explicit site adapter approach means that fewer websites can be customized, and also that users can't perform customizations if the relevant data is not exposed in the spreadsheet.

Another difference between Wildcard and these tools is that there is no way for users to express imperative workflows with sequences of actions in Wildcard. Chickenfoot and Coscripter have shown these types of workflows to be useful in practice so it might be worth considering how to incorporate them, but it's not obvious how such workflows could fit into Wildcard. Spreadsheets have a fundamentally different computation model, and site adapters created by programmers cannot easily account for every possible action a user might want to perform in a page.

5.3 Spreadsheet-Driven Applications

Many researchers and companies have explored the idea of *spreadsheet-driven applications*: using spreadsheets as a backing data store and computation layer for interactive web applications, enabling end users to create such applications more easily. Research projects like Object Spreadsheets [21], Quilt [6], Gneiss [8], and Marmite [27], as well as commercial tools like Airtable [2] and Glide [3] allow users to view data in a spreadsheet table, compute over the data using formulas, and then connect the table to a GUI. Because many users are already familiar with spreadsheets, this way of creating applications tends to be much easier than traditional software methods; for example, in a user study of Quilt, many users 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 far 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 [27]. 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.

5.4 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 customizes the application's UI 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 [9] extract data in a minimally structured relational format; other tools like Piggy Bank [13] more ambitiously map the data to a rich semantic schema. In Wildcard, we chose to avoid semantic schemas, in order to minimize the work associated with creating a site adapter.

In the future, we might try to integrate existing web scraping tools, to help create more reliable site adapters for Wildcard with less work, and to open up adapter creation to end users. Although Wildcard has additional requirements beyond traditional web scraping as discussed in Section 3, there is still enough overlap that existing tools might prove helpful.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented *spreadsheet-driven customization*, a technique that enables end users to customize software by augmenting an application's UI with a spreadsheet. We still have many open questions which we hope to answer through targeted development and usage of Wildcard.[1]

The most important question is: what are the limits of this computational model? What types of useful customizations can it support or not support? While initial demos suggest a variety of use cases, we plan to develop more site adapters and demos to explore this question further. We will start by privately testing the system with our own needs and then eventually deploy the tool publicly once it has a stable API 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.

We suspect that some areas of the current model may prove overly simplistic. For example, Wildcard's data model currently shows a single table at a time, without any notion of relationships between tables. A richer data model with foreign keys might help support certain use cases. For designing a spreadsheet interface on top of a richer relational model, we could learn from other systems that address this design challenge [4, 21].

[1]Up to date information about Wildcard is available on the project website (https://geoffreylitt.com/wildcard), which includes a signup for receiving updates and notifications about a public release. We also welcome hearing from readers who might be interested in being private beta testers or who have feedback on the project.

Another question is how easily site adapters can be created and maintained for real websites, which often include complex markup and change frequently. We plan to explore this question by creating adapters for sites with different data structures and in different domains, and perhaps running user tests with programmers. Possible future improvements we've considered include developing automated heuristics to assist with the adapter creation process and developing new abstractions that make it easier for programmers to efficiently create robust adapters.

Ultimately, we hope that spreadsheet-driven customization helps the Web reach its full potential as a dynamic medium that users can mold to their own needs.

ACKNOWLEDGMENTS

This project was partially supported by the SaTC: CORE program of the CISE division of the National Science Foundation, Award Number 1801399, and by the International Design Center, a collaboration between MIT and the Singapore University of Technology and Design. Thanks to Tarfah Alrashed, Glen Chiacchieri, David Karger, Steve Krouse, Rob Miller, Santiago Perez De Rosso, Arvind Satyanarayan, Daniel Windham, Maggie Yellen, and the anonymous workshop reviewers for providing valuable feedback on this work.

REFERENCES

[1] 2019. End-User Programming. https://www.inkandswitch.com/end-user-programming.html.
[2] 2020. Airtable. https://airtable.com.
[3] 2020. Glide. https://www.glideapps.com/.
[4] Eirik Bakke and David R. Karger. 2016. Expressive Query Construction through Direct Manipulation of Nested Relational Results. In *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. ACM Press, San Francisco, California, USA, 1377–1392. https://doi.org/10.1145/2882903.2915210
[5] Michel Beaudouin-Lafon and Wendy E. Mackay. 2000. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '00)*. ACM, New York, NY, USA, 102–109. https://doi.org/10.1145/345513.345267
[6] Edward Benson, Amy X. Zhang, and David R. Karger. 2014. Spreadsheet Driven Web Applications. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST '14*. ACM Press, Honolulu, Hawaii, USA, 97–106. https://doi.org/10.1145/2642918.2647387
[7] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. 2005. Automation and Customization of Rendered Web Pages. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology - UIST '05*. ACM Press, Seattle, WA, USA, 163. https://doi.org/10.1145/1095034.1095062
[8] Kerry Shih-Ping Chang and Brad A. Myers. 2014. Creating Interactive Web Data Applications with Spreadsheets. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST '14*. ACM Press, Honolulu, Hawaii, USA, 87–96. https://doi.org/10.1145/2642918.2647371
[9] Sarah E. Chasins, Maria Mueller, and Rastislav Bodik. 2018. Rousillon: Scraping Distributed Hierarchical Web Data. In *The 31st Annual ACM Symposium on User Interface Software and Technology - UIST '18*. ACM Press, Berlin, Germany, 963–975. https://doi.org/10.1145/3242587.3242661
[10] Andrea A. diSessa. 2000. *Changing Minds: Computers, Learning, and Literacy*. MIT Press, Cambridge, MA, USA.
[11] A. A diSessa and H. Abelson. 1986. Boxer: A Reconstructible Computational Medium. *Commun. ACM* 29, 9 (Sept. 1986), 859–868. https://doi.org/10.1145/6592.6595
[12] Andrew Hogue and David Karger. 2005. Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web. In *Proceedings of the 14th International Conference on World Wide Web - WWW '05*. ACM Press, Chiba, Japan, 86. https://doi.org/10.1145/1060745.1060762
[13] David Huynh, Stefano Mazzocchi, and David Karger. 2005. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *The Semantic Web – ISWC 2005 (Lecture Notes in Computer Science)*, Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen (Eds.). Springer, Berlin, Heidelberg, 413–430. https://doi.org/10.1007/11574620_31

[14] David F. Huynh, Robert C. Miller, and David R. Karger. 2006. Enabling Web Browsers to Augment Web Sites' Filtering and Sorting Functionalities. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology - UIST '06*. ACM Press, Montreux, Switzerland, 125. https://doi.org/10.1145/1166253.1166274

[15] Hypercard. 2019. HyperCard. *Wikipedia* (Dec. 2019).

[16] A. Kay and A. Goldberg. 1977. Personal Dynamic Media. *Computer* 10, 3 (March 1977), 31–41. https://doi.org/10.1109/C-M.1977.217672

[17] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*. ACM Press, Daegu, Kyungpook, Republic of Korea, 280–290. https://doi.org/10.1145/2807442.2807446

[18] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & Sharing How-to Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1719–1728. https://doi.org/10.1145/1357054.1357323

[19] Greg Little, Robert C. Miller, Victoria H. Chou, Michael Bernstein, Tessa Lau, and Allen Cypher. 2010. Sloppy Programming. In *No Code Required*. Elsevier, 289–307. https://doi.org/10.1016/B978-0-12-381541-5.00015-8

[20] Wendy E. Mackay. 1991. Triggers and Barriers to Customizing Software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Reaching through Technology - CHI '91*. ACM Press, New Orleans, Louisiana, United States, 153–160. https://doi.org/10.1145/108844.108867

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

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

[23] Bonnie A. Nardi and James R. Miller. 1991. Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development. *International Journal of Man-Machine Studies* 34, 2 (Feb. 1991), 161–184. https://doi.org/10.1016/0020-7373(91)90040-E

[24] Mitchel Resnick. 2016. Designing for Wide Walls. https://design.blog/2016/08/25/mitchel-resnick-designing-for-wide-walls/.

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

[26] Bret Victor. 2020. Dynamicland. https://dynamicland.org/.

[27] Jeffrey Wong and Jason I. Hong. 2007. Making Mashups with Marmite: Towards End-User Programming for the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '07*. ACM Press, San Jose, California, USA, 1435–1444. https://doi.org/10.1145/1240624.1240842