Customizing Software by Direct Manipulation of Structured Data

Anonymous Author(s)

Abstract

In this paper we propose *table-driven customization,* a new paradigm for enabling end users to customize software applications without doing traditional programming. Users directly manipulate a tabular view of the structured data inside the application, rather than writing imperative scripts as in most customization tools. We extend this simple model with a spreadsheet formula language and custom data editing widgets, which provide sufficient expressivity to implement many useful customizations.

We describe Wildcard, a browser extension which implements table-driven customization in the context of web applications. Through concrete examples, we demonstrate that this paradigm can be used to create useful customizations for real applications. We share reflections from our usage of the Wildcard system, including its strengths and limitations relative to other customization approaches. We further explore how new software architectures might encourage application developers to promote this style of end-user customization.

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

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

ACM Reference Format:
Anonymous Author(s). 2020. Customizing Software by Direct Manipulation of Structured Data. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

There have been many attempts at empowering end users to customize their software by offering them a simplified programming tool. Some scripting languages (AppleScript, Chickenfoot) have a friendly syntax that resembles natural language. Visual programming tools (Mac Automator, Zapier) eliminate text syntax entirely. Macro recorders (Applescript, Helena, WebVCR) remove some of the initial programming burden by letting a user start by concrete demonstrations.

These approaches have many differences, but they all share something in common: an imperative programming model, with mutable variables, conditionals and loops. End users ultimately must use these traditional programming constructs to express their ideas, and the object of interest is a *script*, a sequence of commands.

We have known for decades about an alternate approach: *direct manipulation* [8], where "visibility of the object of interest" replaces "complex command language syntax". Direct manipulation is the de facto standard in GUIs today, but when it comes to customizing software, it is rarely to be found. In this work, we ask: what would it look like to build a software customization interface that deeply relies on direct manipulation? We take inspiration from spreadsheets and visual database query interfaces [1, 3], which have successfully enabled end users to construct queries, edit data, and compute derived values through direct manipulation of structured data.

In this paper we present a technique called *table-driven customization* which applies these ideas from visual query interfaces in the context of software customization. We augment an application's UI with a table view, where the user can see and manipulate the structured data inside an application. Changes in the table view result in immediate corresponding changes to the original user interface of the application, enabling the user to customize an application with live feedback. The table view and the original UI serve as coordinated *multiple representations* of the same underlying state.

To demonstrate the viability of this approach, we have developed a browser extension called Wildcard which implements table-driven customization in the context of the Web. Wildcard is compatible with existing unmodified applications because it uses web scraping techniques to extract relevant state into a table and to reify changes in the original application UI. In Section 2 we present a many examples of using Wildcard to implement useful customizations across real websites, ranging from sorting lists of data to adding whole new features to applications.

In Section 3 explain the *table adapter* abstraction, which allows many different types of underlying state to be bidirectionally mapped to a table of data. We explore how table adapters are general enough to support many customizations beyond the ones we've built so far in Wildcard; we also discuss how applications could be re-architected to expose tabular data directly to end users, rather than relying on web scraping techniques.

In Section **??** we discuss in more detail the implementation of the Wildcard browser extension. We reflect on the experience of using the tool to customize websites, and some of the challenges from the perspective of programmers and end users.

Table-driven customization embodies several design principles for building customizable software systems, discussed in Section 5:

- *Customization by direct manipulation*: End users should be able to customize an application by examining and modifying its data, rather than by writing imperative scripts.
- *Third-party semantic wrappers*: Typically, customization tools which don't rely on official extension APIs resort to offering low-level APIs for customization. Instead, we suggest a community-maintained library of semantic wrappers around existing applications, offering end users a more convenient semantic API for customization.
- *Gentle slope*: Customization tools should ideally offer a "gentle slope" from normal use to customization. We explore how our approach meets this goal by making some software customizations as easy as normal use of the software.

Table-driven customization relates to existing work in many areas. In particular, our goals overlap with many software customization tools, and our methods overlap with direct manipulation interfaces for working with structured data, including visual database query systems and spreadsheets. We explore these connections and more in Section 6.

2 Examples

To concretely illustrate the end user experience of table-driven customization, here are several real examples of using the Wildcard browser extension to customize websites.

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 1 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 1, 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 1, Note B). Similarly, the user can open the detailed pages for many listings at once.

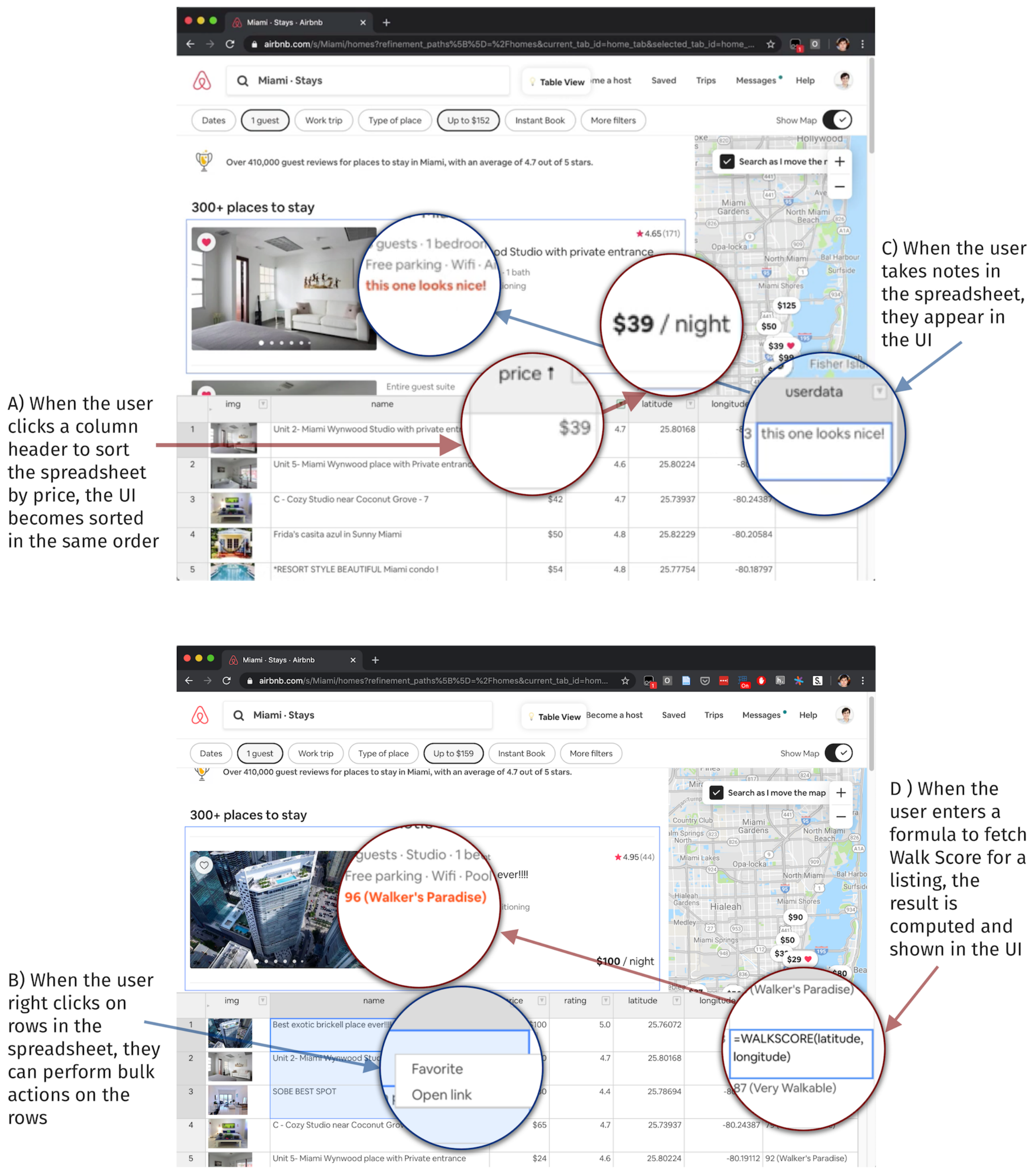
Next, the user wants to jot down some notes about each listing. To do this, they type some notes into an additional column in each row, and the notes appear inside the listings in the original UI (Figure 1, Note C). The annotations are saved in the browser and associated with the backend ID of the listing, 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 1, Note D).

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 2 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 a new column to store the snooze date for each todo (Figure 2, Note A).



A) When the user clicks a column header to sort the spreadsheet by price, the UI becomes sorted in the same order

	img	name	price ↑		latitude	longitude	userdata
1		Unit 2- Miami Wynwood Studio with private ent	$39	4.7	25.80168		this one looks nice!
2		Unit 5- Miami Wynwood place with Private entranc		4.6	25.80224		
3		C - Cozy Studio near Coconut Grove - 7	$42	4.7	25.73937	-80.24387	
4		Frida's casita azul in Sunny Miami	$50	4.8	25.82229	-80.20584	
5		*RESORT STYLE BEAUTIFUL Miami condo !	$54	4.8	25.77754	-80.18797	

C) When the user takes notes in the spreadsheet, they appear in the UI

B) When the user right clicks on rows in the spreadsheet, they can perform bulk actions on the rows

	img	name	price	rating	latitude	longitude	
1		Best exotic brickell place ever!!!!	$100	5.0	25.76072		=WALKSCORE(latitude, longitude)
2		Unit 2- Miami Wynwood Stud		4.7	25.80168		
3		SOBE BEST SPOT		4.4	25.78694		87 (Very Walkable)
4		C - Cozy Studio near Coconut Gro	$65	4.7	25.73937	-80.24387	
5		Unit 5- Miami Wynwood place with Private entrance	$24	4.6	25.80224	-80.19112	92 (Walker's Paradise)

D) When the user enters a formula to fetch Walk Score for a listing, the result is computed and shown in the UI

Figure 1. Using Wildcard to augment the Airbnb search page for booking accommodations

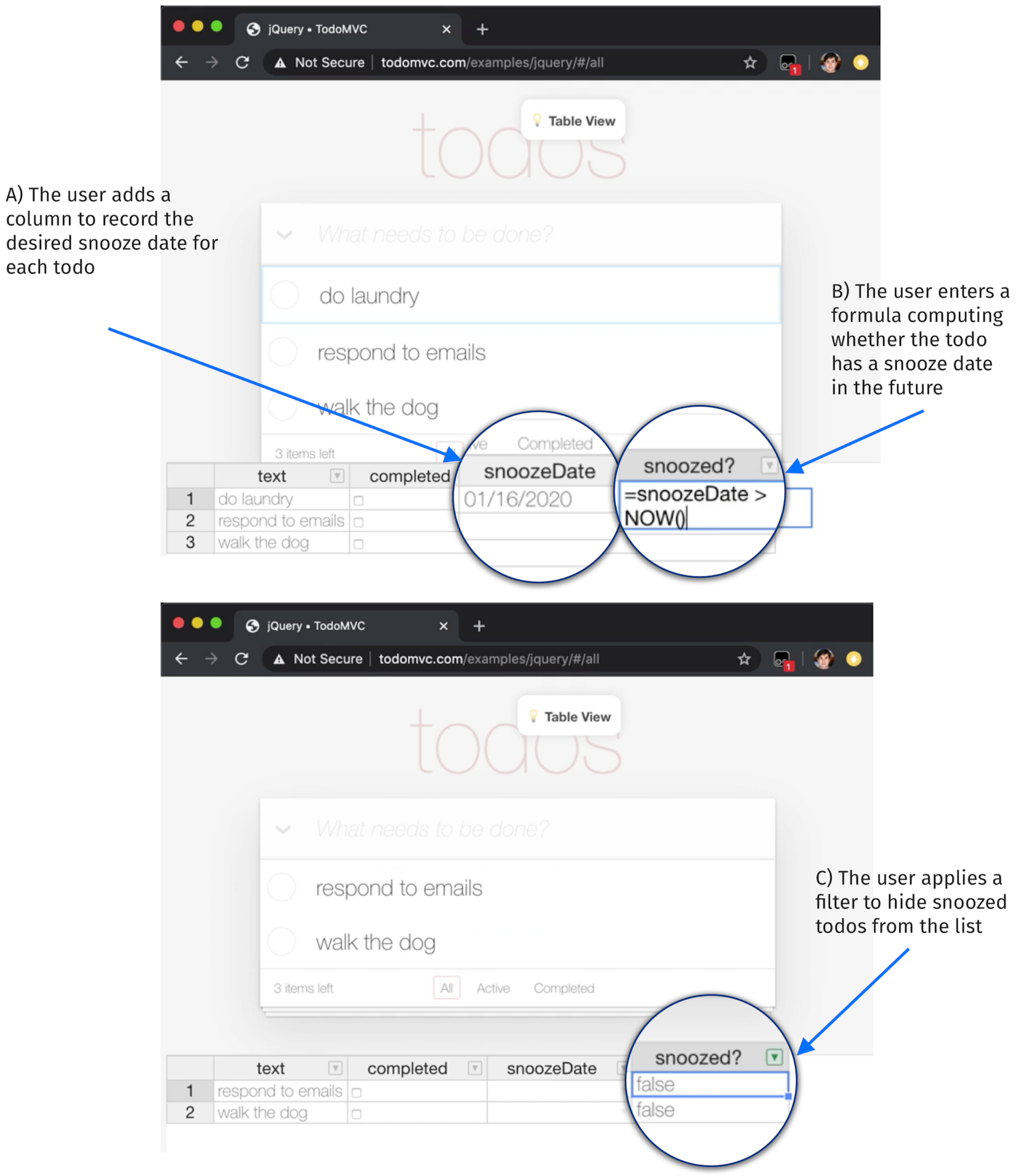

jQuery • TodoMVC
Not Secure | todomvc.com/examples/jquery/#/all
Table View
todos
What needs to be done?
do laundry
respond to emails
walk the dog
3 items left
All
Active
Completed
text
completed
snoozeDate
snoozed?
01/16/2020
=snoozeDate > NOW()
false
false
A) The user adds a column to record the desired snooze date for each todo
B) The user enters a formula computing whether the todo has a snooze date in the future
C) The user applies a filter to hide snoozed todos from the list

Figure 2. Using Wildcard to add a "snooze" feature to the TodoMVC todo list app

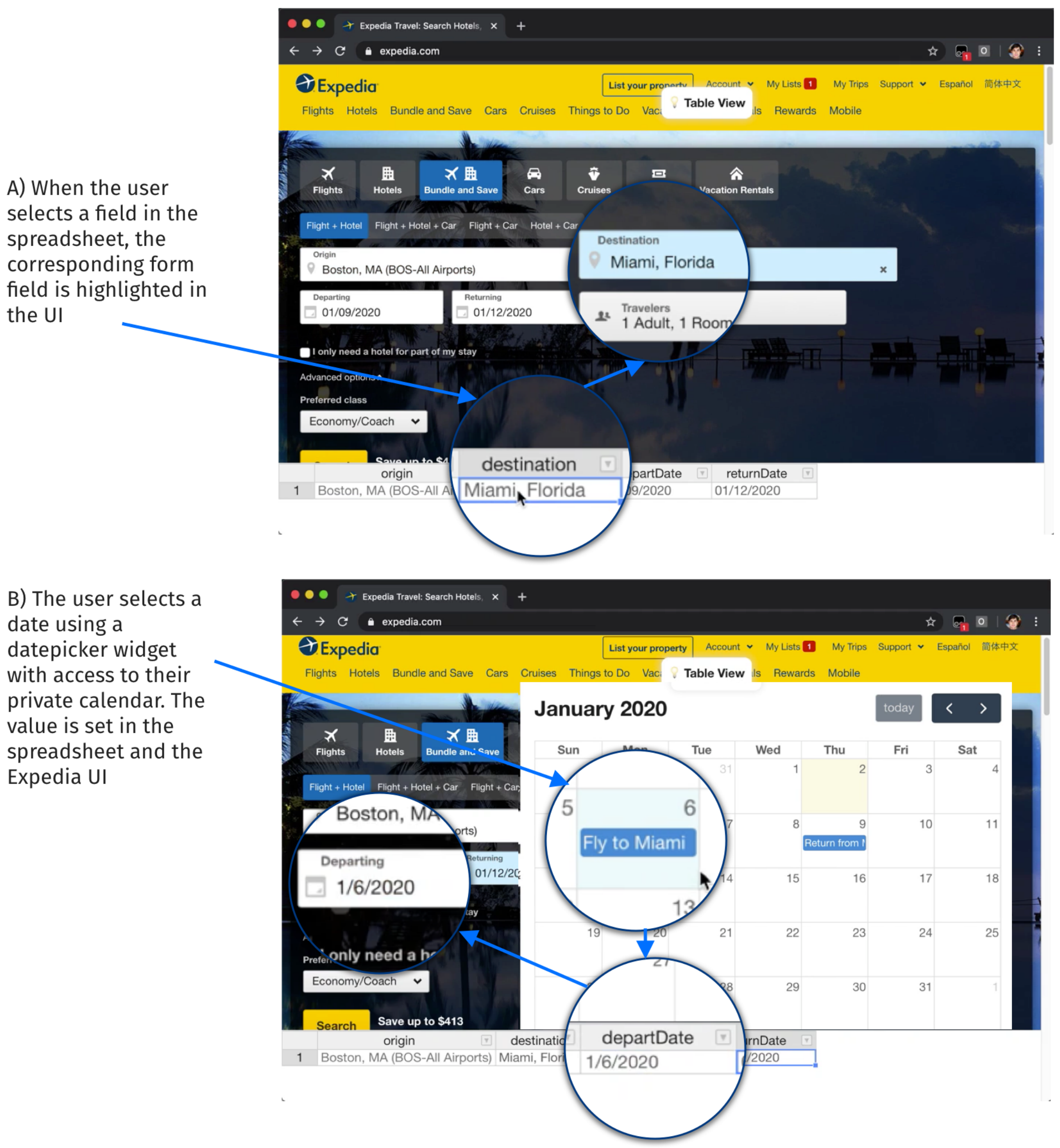
The next step is to hide snoozed todos. The user creates a snoozed? column, which uses a formula to compute whether a todo is snoozed—i.e., whether it has a snooze date in the future (Figure 2, Note B). Then, they simply filter the table to hide the snoozed todos (Figure 2, Note C).

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,



A) When the user selects a field in the spreadsheet, the corresponding form field is highlighted in the UI

B) The user selects a date using a datepicker widget with access to their private calendar. The value is set in the spreadsheet and the Expedia UI

	origin	destination	departDate	returnDate
1	Boston, MA (BOS-All Airports)	Miami, Florida	1/6/2020	01/12/2020

Figure 3. Using Wildcard to augment the Expedia page for booking a flight

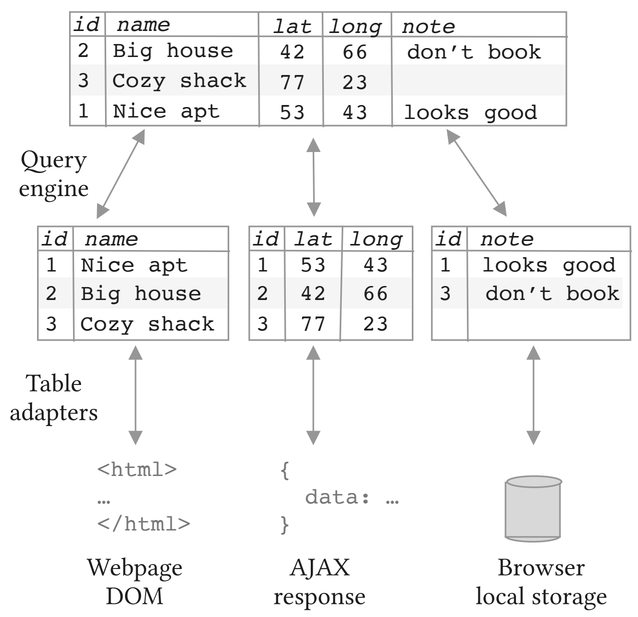
with a column corresponding to each input (Figure 3, Note A).

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 3, Note B). The user can directly click on the correct date, and it gets inserted into both the spreadsheet and the original form.

3 System architecture

- basic idea: end user editing a table. but these edits have effects.
- table adapter: component that presents an interface similar to a SQL table. but inside, it can do anything to produce that data, and can do anything with incoming edits.
  - out: table



id	name	lat	long	note
2	Big house	42	66	don't book
3	Cozy shack	77	23	
1	Nice apt	53	43	looks good

id	name
1	Nice apt
2	Big house
3	Cozy shack

id	lat	long
1	53	43
2	42	66
3	77	23

id	note
1	looks good
3	don't book

Figure 4. The table adapter architecture

- in: table
- in: other tables

The basic data unit in table-driven customization is a Table: an ordered list of tuples, each of which has an ID and some attributes. The user always sees a single table in the table editor, but this table is the result of combining data across one or more underlying tables, much like viewing the result of a SQL query across multiple database tables. Each individual table's data is managed by a "table adapter" component, and a query engine coordinates across the various table adapters.

(*Todo: drive this explanation via concrete example. Add a figure.*) For example, one table adapter manages scraping data from the DOM of a site, while another manages extracting data from AJAX requests. Each table adapter produces a table of data, and the two are joined together by a shared ID column to produce what looks like a single table of data extracted from the site. Similarly, when the user makes annotations on a site, these annotations are handled by a third table adapter, which manages a data table stored in the browser's local storage.

3.1 Table adapter API

(*Todo: formalize this more? Would math notation help?*)

In order to promote uniformity within the system, we've defined a small abstract API, which all table adapters conform to.

The main purpose of a table adapter is to return a table representing some data source. The table is live-updating: a table adapter can produce a new version of its data table at any point (e.g., updating the data table in response to a change in a website's DOM). When this happens the new data is pushed to the editor and the query results are refreshed.

Table editors also accept various incoming commands.

The editor can request to a table adapter to make an edit to a record. The meaning of making an edit can vary depending on the adapter: in the local storage adapter, an annotation can be persisted into local storage; in the DOM scraping adapter, an edit can represent filling in a form field. An adapter can also mark values as read-only if it wouldn't be meaningful to edit them; for example, the DOM scraping adapter typically marks page content as read-only, except for editable form fields.

The query engine also sends each table adapter information about the entire cross-table query view being created by the user. This functionality is currently mainly used by the DOM scraping adapter to make changes to the original web page:

- observes the sort order of the query view to re-order rows in the page
- observes the content of other joined tables to render annotations in the page
- observes the currently selected row in the UI, to highlight the corresponding row in the page

3.2 Table adapters

Here we describe in more detail the table adapters which we have implemented to power the customizations shown above.

DOM scraping adapters are a key part of the Wildcard system. These adapters extract data from the DOM of a web page, and manipulate the DOM to re-order rows, edit form entries, and inject annotations. To create a DOM scraping adapter for a specific website, a programmer does not need to implement all this functionality from scratch. They only need to write a single scraping function which uses CSS selectors and DOM APIs to extract and return relevant elements. The Wildcard framework then wraps this function to implement the rest of the needed functionality.

(*todo: get into the Sorting Problem? i.e., once we sort the page, the original sort order is lost from the page. Maybe not necessary...*)

3.3 Other table adapters

An **AJAX scraping adapter** intercepts AJAX requests made by a web page, and extracts information from those requests. A programmer writes a function which specifies how to extract data from an AJAX request, and the framework handles the details of actually intercepting requests and calling the programmer-defined function.

The **local storage adapter** simply stores a table of data in the browser.

3.3.1 Future Adapters

We have designed the table adapter API to be general enough to additional types of functionality in the future. Here are three concrete examples of hypothetical possibilities:

Integrated website adapters: A key benefit of the table adapter abstraction is that Wildcard is not coupled to web scraping as the only means for integrating with existing sites, but can also accommodate first party developers adding support directly into their own websites. A DOM Scraping table adapter could be swapped out for such an "integrated website adapter," which implements the exact same interface as a scraping adapter but by directly accessing the internal state of the user interface, without needing any scraping logic.

We think it is possible to create such adapters for existing frontend web frameworks which result in minimal effort on the part of the application developers. For example, we have created an initial prototype of a plugin for the Redux state management library. Redux implements the Model-View-Update pattern, which represents the entire state of a user interface as a single centralized object. To configure an integrated website adapter for such an existing application, the user can simply specify a function projecting the centralized application state as a table, and define handlers for how data edits to the table should affect the state.

Shared storage adapter: It would be useful to share user annotations between people and across devices—for example, collaboratively taking notes with friends on a list of options for places to stay on Airbnb. The existing Local Storage Adapter could be extended to share live synchronized data with other users. This could be achieved through a centralized web server (perhaps an existing service like Google Sheets capable of storing tabular data), or through P2P connections which might improve the privacy guarantees available for the shared annotations.

Third party API data adapter: Currently, the main mechanism for including data from web APIs in Wildcard is using spreadsheet formulas. However, a web API could also be wrapped to expose a table API that would dynamically create tables in response to queries. For example, when fetching walkability scores for many GPS locations, the query engine could request a table of walkability scores for various pairs of latitude and longitude, and the table adapter could dynamically perform API queries to populate a result table. (*todo: why is this better than spreadsheet formulas? I don't think it offers any more room for batching performance optimization because we've already framed formulas as whole-column, not row specific*)

3.4 Query engine

The query engine is responsible for coordinating across multiple table adapters. It joins data across multiple tables and creates a single result table which is shown to the user through the editor. It also handles all user interactions and routes appropriate messages to each table adapter.

First, every query involves a primary DOM scraping table adapter which associates records in the result with elements in the application's user interface. At minimum, the primary table adapter needs to return record IDs and have the ability to manipulate the application's UI. It can also optionally return data about each record.

Next, additional tables (AJAX data, local storage data) are left joined by ID. (*todo: discuss IDs here?*) Finally, the result table can be sorted and filtered by any column.

One way to think of this model is a tiny constrained subset of the SQL query model. We've found that this simple model has proven sufficient for meeting the needs of customization in practice, and minimizes the complexity of supporting more general and arbitrary queries. But because it fits into the general SQL paradigm, it could theoretically be extended to support more types of queries.

The query engine is also responsible for executing formulas. We have built a small language resembling the formula languages in visual query tools like SIEUFERD. As in those tools, and unlike in spreadsheets, formulas automatically apply to an entire column of data, and can reference columns rather than only specific cells.

3.5 Table editor

In Wildcard we provide a basic table editor as the user interface on top of the query engine. It is built with the Handsontable Javascript library, which provides UI elements for viewing and editing a table, as well as basic query operations like sorting and filtering.

todo: other table instruments

4 Evaluation

To determine the viability of writing DOM scraping adapters, we built adapters for 10 websites, including transactional sites like Amazon and Uber Eats, media consumption sites like Hacker News and Youtube, and a medical results entry platform managed by a member of our research group. The adapters ranged from 36 to 117 lines of code, averaging 68 lines. In addition, an external developer unaffiliated with the project contributed one adapter, designed to sort the Github page listing a user's repositories, and described the experience as "very straightforward."

Some of the challenges of writing a DOM scraping adapter are the same ones involved with writing normal web scraping code, but the more interactive nature of Wildcard introduces additional challenges. One 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. Another challenge is persisting updates to the DOM—some websites use virtual DOM frameworks that can occasionally overwrite changes made by Wildcard. In practice, we've managed to create an implementation that works around these issues and results in a usable customization experience for all the websites we've tried.

5 Discussion

5.1 Customization by direct manipulation

There have been many attempts at simplifying aspects of traditional programming to enable non-programmer end users to perform customization. End user scripting languages (AppleScript, VBA, Chickenfoot, CoScripter) have innovated at the syntax level, aiming to improve ease-of-use. Visual programming environments (Mac Automator, Zapier) go further by eliminating text syntax entirely, but maintain the same computational model. Programming-by-demonstration and macro recording environments (Applescript, Helena, WebVCR) remove some of the initial programming burden by starting with concrete demonstrations, but typically require programming later in the process.

However, all of these approaches share something in common: **an imperative programming model**. They all involve a list of commands executed in sequence, with control flow constructs like conditionals and loops, and mutable variables. This makes some intuitive sense: many customizations seem most naturally expressed as a series of commands, each of which has a side effect on the UI: first enter this value, then click this button, etc. The result is that end users eventually need to become familiar with all of these traditional imperative programming constructs, even if only in diluted form. (todo: bolster the case that this is a barrier to learning)

Imperative programming is not the only possible model. Spreadsheets have demonstrated another computational model: data editing via direct manipulation, combined with pure functional expressions for computation, with automatic reactive updates provided by the runtime. We call this a data-oriented model, because the user is focused on editing data values and producing the correct derived data values, rather than encoding a sequence of commands. The success of spreadsheets with end user programmers suggests that finding a way to apply data-oriented programming model to software customization might enable more people to modify their software. However, it's not immediately obvious how the spreadsheet paradigm applies to customization.

A key insight from our work is that, rather than representing customization as a series of commands issued to an existing application, we can present a projected view of the application's internal state, and let the user directly edit that projection as a means of customizing the application. In some sense, this is similar to the DOM inspector, which provides a user with an alternate representation of the UI that they can directly manipulate (e.g., by selecting an element and deleting it from the DOM tree). But rather than provide a low-level view of the user interface elements, we instead provide a higher-level view of the semantic state of the application.

Add here:

- multiple representations [2]
- address Tchernavskij's point about contrasting with instrumental interaction

5.2 Third-party semantic wrappers

Software customization tools typically fall into one of two categories: third-party, or semantic.

Third-party customization tools enable customization without using official extension APIs, enabling a broader range of customizations on top of more applications. For example, web browser extensions have demonstrated the utility of customizing websites through manipulating the DOM, without needing explicit extension APIs to be built in. However, third-party customization comes with a corresponding cost: these tools can typically only operate at a low level of abstraction, e.g. manipulating user interface elements. This makes it harder for end users to write scripts, and makes the resulting scripts more brittle. (todo: support this claim more, provide examples)

Semantic customization tools use explicit extension APIs provided by the application developer. Examples of this include accessing a backend web API, or writing a customization in Applescript that uses an application-specific API. The main benefit is that this allows the extension author to work with meaningful concepts in the application domain—"create a new calendar event" rather than "click the button that says new event."—which makes customizations easier to build and more robust. However, this style limits the range of extensions that can be built, to only those which the official plugin API supports.

(*Footnote this paragraph?*) This categorization is slightly oversimplified, and there are existing customization ecosystems that span both categories. For example, part of the success of browser extensions stems from the fact that the DOM encourages the use of standardized semantic UI elements; even if an app developer doesn't anticipate customization, merely using of semantic and accessible HTML creates a collateral benefit of easier extension. Another example of a middle ground is AppleScript, which provides system-wide low-level APIs for manipulating GUIs as a means of black-box customization, while also enabling developers to optionally add application-specific APIs to add support for semantic customization.

With Wildcard, we use a hybrid approach that takes the best of both worlds. Programmers implement an API wrapper that is internally implemented as a black-box customization, but externally provides a semantic interface to the application. End users get an ergonomic and simplified customization experience, but without the need to depend on a first-party application developer exposing extension APIs.

One way to view this approach is as introducing a new abstraction barrier into black-box extension. Typically, a black box customization script combines two responsibilities: 1) mapping the low-level details of a user interface to semantic constructs (e.g., using CSS selectors to find certain page elements), and 2) the actual logic of the specific customization. (todo: could easily show examples of this from browser extensions, Chickenfoot, etc) Even though the mapping logic is often more generic than the specific customization (e.g., finding a given input element is independent of what text to insert into that element), the intertwining of these two responsibilities in a single script makes it very difficult to share the mapping logic across scripts. With Wildcard we propose a decoupling of these two layers: a community-maintained mapping layer, shared across many specific customizations by individual users. This architecture has been successfully used by projects like Gmail.js, an open source project that creates a convenient API for browser extensions to interface with the Gmail web email client.

5.3 Gentle slope

Originally came from [7]

- here’s where “automation vs customization” comes in: automation frames it as entering a whole new scripting environment; we frame it more as just an alternate UI that you can use.
- this is the heart of the gentle slope
- backend APIs: terrible
- the best would be right inline with the GUI (cite Scotty, instrumental work) but this has its own problems
- we settle for a compromise: an alternate structured view. (explicitly contrast this clearly)
- the CLI GUI thing

6 Related Work

This paper extends a workshop paper by Litt and Jackson [5] which presented an earlier prototype of the Wildcard system. We build on that work in this paper by rearchitecting the internal implementation of the system around the table adapter abstraction, evaluating it on many more websites, and by more fully characterizing the behavior and design of the system.

6.1 Customization tools

- customization tools
  - browser extensions
  - ScrAPIr
  - Thresher
  - Sifter
  - customization research tools: Chickenfoot, Coscripter
  - Wildcard, like Chickenfoot, wants to hide HTML from users. But we show a structured data view, whereas Chickenfoot shows nothing
  - visual customization tools:
    * Mac Automator
    * Zapier
    * IFTTT
    * These are weakly direct manipulation. But usually the thing you directly manipulate in these UIs is the *script*: not the actual *data* you want to operate on. VERY DIFFERENT.
  - desktop customization: Applescript, VBA, COM

6.2 Visual database query tools

- database GUIs:
  - Sieuferd [3]
  - Airtable [1]
  - other similar tools?
  - Liu & Jagadish: rules for a spreadsheet algebra for database queries [6]
  - Shneiderman paper
  - spreadsheets

6.3 Other

- instrumental interaction, Scotty [4], Webstrates
- browser dev tools
- Self, Smalltalk
- multiple representations
- Semantic Web

References

[1] 2020. Airtable. https://airtable.com.

[2] Shaaron Ainsworth. 1999. The Functions of Multiple Representations. *Computers & Education* 33, 2-3 (Sept. 1999), 131–152. https://doi.org/10.1016/S0360-1315(99)00029-9

[3] 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

[4] James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2011. Cracking the Cocoa Nut: User Interface Programming at Runtime. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology - UIST ’11*. ACM Press, Santa Barbara, California, USA, 225. https://doi.org/10.1145/2047196.2047226

[5] Geoffrey Litt and Daniel Jackson. 2020. Wildcard: Spreadsheet-Driven Customization of Web Applications. In *Companion Proceedings of the 4th In- Ternational Conference on the Art, Science, and Engineering of Programming*. Association for Computing Machinery, Porto, Portugal., 10. https://doi.org/10.1145/3397537.3397541

[6] Bin Liu and H. V. Jagadish. 2009. A Spreadsheet Algebra for a Direct Data Manipulation Query Interface. In *2009 IEEE 25th International Conference on Data Engineering*. 417–428. https://doi.org/10.1109/ICDE.2009.34

[7] Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. 1990. User-Tailorable Systems: Pressing the Issues with Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Empowering People - CHI '90*. ACM Press, Seattle, Washington, United States, 175–182. https://doi.org/10.1145/97243.97271

[8] B. Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (Aug. 1983), 57–69. https://doi.org/10.1109/MC.1983.1654471