# A Unified Model For Web Scraping & Customization

Kapaya Katongo
MIT CSAIL
Cambridge, MA, USA
kkatongo@mit.edu

Geoffrey Litt
MIT CSAIL
Cambridge, MA, USA
glitt@mit.edu

Kathryn Jin
MIT CSAIL
Cambridge, MA, USA
kjin@mit.edu

Daniel Jackson
MIT CSAIL
Cambridge, MA, USA
dnj@csail.mit.edu

## ABSTRACT

Websites are malleable: users can run code in the browser to customize them. However, this malleability is typically only accessible to programmers with knowledge of HTML and Javascript. Testing

## CCS CONCEPTS

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

## KEYWORDS

end-user programming, software customization, web scraping, programming by example

**ACM Reference Format:**
Kapaya Katongo, Geoffrey Litt, Kathryn Jin, and Daniel Jackson. 2021. A Unified Model For Web Scraping & Customization. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Many websites on the internet do not meet the exact needs of all of their users. End-user web customization systems like Thresher, Sifter and Vegemite help users to tweak and adapt websites to fit their unique requirements, ranging from reorganizing or annotating content on the website to automating common tasks. Millions of people also use tools like Greasemonkey and Tampermonkey to install browser userscripts, snippets of Javascript code which customize the behavior of websites.

A new approach known as data-driven web customization promises to enable web customization without traditional programming through direct manipulation of a spreadsheet-like table, right within the context of a browser. In this paradigm, a table is added to a website that contains its underlying structured data and is bidirectionally synchronized with it. Changes to the table, including sorting, filtering, adding annotations and running computations in a spreadsheet-like formula language are propagated to the website thereby customizing it.

While end-user friendly, data-driven customization suffers from a divide between generating the table and using it to perform customizations: the table is generated through web scraping code written by programmers but used via direct manipulation by end-users. This divide limits the agency of end-users because they rely on programmers to write the required web scraping code before being able to customize. In prior work, we harnessed programming-by-demonstration to empower end-users to achieve the task of web scraping, without writing code, within the context of the table used for customizations. However, this reduced the expressiveness of web scraping for programmers and both hid the underlying program synthesized to perform the web scraping and prevented modifications to be made to it.

The next task was therefore to achieve the seemingly opposing goals of empowering end-users to perform web scraping within the context of the table without disempowering programmers from being able to utilize the expressiveness of programming and view and modify the web scraping program. To solve this, we present a unified model for web scraping and customization, with the aforementioned table formula language acting as a bridge between the two. Demonstrations are used to empower end-users to achieve the task of web scraping, with the synthesized web scraping program presented as a formula in the table in a manner


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

that enables programmers to mitigate the lack of expressiveness in demonstrations by being able to view and modify the synthesized program.

This model for web scraping and customization builds on several key ideas and design goals we discuss in Section X. To test its viability, we implement it as an extension of Wildcard, a browser extension that implements data-driven web customization. Our contributions are:

- A unified model for web scraping and customization that enablers end-users to perform the task of web scraping and programmers to view and modify the web scraping program synthesized from a demonstration, both within the context of the table used for customization
- A novel combination of design goals which make our unified model for web scraping and customization interactive and seamless for both end-users and programmers
- An example gallery of websites that can be customized via our unified model for web scraping and customization and a preliminary user study providing some qualitative results of using it

## 2 MOTIVATING EXAMPLE

## 3 SYSTEM IMPLEMENTATION

### 3.1 Wrapper Induction Algorithm

### 3.2 CSS Selector Synthesis

### 3.3 Live Programming

### 3.4 Limitations

#### *3.4.1 Wrapper Induction Algorithm.*

#### *3.4.2 CSS Selector Synthesis.*

#### *3.4.3 Live Programming.*

## 4 DESIGN PRINCIPLES

### 4.1 Unified Environment & User Model

In the first iteration of data-driven customization, web scraping and customization were divided: web scraping was done by programmers in an Integrated Development Environment (IDE) while customizing (via direct manipulation and formulas) was done in the browser. This type of divide between tasks can be seen in other domains:

In data science, workflows revolve between cleaning and using data which often happen in different environments (e.g. data wrangling tools and live notebooks). Wrex [1], an end-user programming-by-example system for data wrangling, reported that "although data scientists were aware of and appreciated the productivity benefits of existing data wrangling tools, having to leave their native notebook environment to perform wrangling limited the usefulness of these tools." This was a major reason Wrex was developed as an add-on to Jupyter notebooks, the environment in which data scientists use their data. In web scraping, users have to switch from the environment in which they are using the scraped data (database, spreadsheet etc) to the environment in which the data is scraped if they need more or need to fix omissions. This can be seen in tools like Rousillon, FlashExtract, import.io, dexi.io, Octoparse and ParseHub.

Based on this, the second iteration of data-driven customization enabled web scraping (via demonstrations) and customization to both be performed in a uniform environment via the customization table in the browser. This relates to the idea of "in-place toolchains" for end-user programming systems: users should be able to program using familiar tools (spreadsheet table) in the context where they already use their software (browsers).

In spite of this uniform environment, web scraping and customization were still divided: web scraping had to be performed prior to customization in a separate phase. Early user tests revealed that this discontinuity between the two phases was a source of confusion. Vegemite, a system for end-user programming of mashups, reported similar findings from its user study in which participants thought that "it was confusing to use one technique to create the initial table, and another technique to add information to a new column."

Armed with this, the iteration of data-driven customization we present has gone beyond providing a uniform environment for web scraping and customization to providing a unified user model for web scraping and customization. Both web scraping and customization are performed in the same, single phase, with users being able to seamlessly interleave the two as desired. A user can start out by demonstrating to populate a column in the table, proceed to populate the next column with the results of a formula and then either switch back to demonstrating to populate additional columns or continue populating columns with the results of formula operations on the scraped data.

### 4.2 Functional Reactive Programming

In general terms, functional reactive programming (FRP) is the combination of functional and reactive programming: it is functional in that it uses functions to define programs that manipulate data and reactive in that it defines data flows through which changes in data are propagated.

FRP has seen wide adoption in end-user programming through implementations such as spreadsheet formula languages (Microsoft Excel & Google Sheets) and formula languages for low-code programming environments (Microsoft Power Fx, Google AppSheets, Glide, Coda & Gneiss).

Because of this, data-driven web customization already provides functional reactive programming via a spreadsheet-lie formula language aimed at increasing the expressiveness of customizations. The language provides formulas to encapsulate logic, perform operations on strings, call browser APIs and even invoke web APIs. As per the FRP paradigm, users only have to think in terms of manipulating the data in the table without having to worry about traditional programming concepts such as variables and data flow.

Our unified model for web scraping and customizations extends this formula language to mitigate the limitations of programming by demonstration. Demonstrations are represented as formulas containing the corresponding, synthesized web scraping code. As with other formulas in the language, web scraping formulas can be modified (or authored from scratch) and run to achieve more expressive or robust web scraping.

Aside from the programmatic benefits of representing demonstrations as formulas, there is also a key usability benefit. Mayer et al report that "a key impedance in adoption of PBE systems is the lack of user confidence in the correctness of the program that was synthesized by the system." They describe how even though FlashFill, a programming-by-example tool for string manipulation in Excel, received many positive reviews from popular media sources, a prominent Excel user expressed caution because of the lack of insight into what the synthesized program is actually doing. Similar sentiments were reported by the authors of Wrex who interviewed data scientists that "were reluctant to use data wrangling tools that transformed their data through 'black boxes.' "

By representing the demonstrations as formulas, our system provides insight into what the synthesized code is doing. In future work, we aim to emulate Mayer et el's work by providing a means for users to choose between possible web scraping programs and pick the one that best achieves their desired goal.

### 4.3 Mixed-Initiative Interaction

Early ideas about mixed-initiative interactions can be traced to the work of Eric Horvitz in which he advocates for "designs that take advantage of the power of direct manipulation and potentially valuable automated reasoning."

Our unified model for web scraping and customization offers mixed-initiative interaction by presenting the result of web scraping by demonstration as a formula. This is advantageous as it not only allows users to delegate automation (via synthesis of web scraping programs) to the system via demonstrations but also keeps the interaction loop open by allowing users to view and modify the output of the demonstration.If desired, users can also achieve web scraping by manually authoring web scraping formulas, switching to generating them via demonstrations at anytime in the process in a seamless and fluid manner.

This type of mixed-initiative interaction can be seen in other programming-by-example systems: - Wrex takes examples of data transforms and generates readable and editable Python code. This was motivated by their formative study in which participants emphasized the need for programming-by-example systems to "produce code as an inspectable and modifiable artifact"

- Small-Step Live Programming By Example presents a paradigm in which programming-by-example is used to synthesize small parts of a user authored program instead of delegating construction of entire program
- Pileg et al outline how programming-by-example is not enough to differentiate all the possible programs and present an interaction model in which users not only provide feedback about the expected output of the program but also the program itself
- Sketch-N-Sketch integrates direct manipulation and programming for the creation of Scalable Vector Graphics (SVG). Users can start out by creating a shape via programming and then switch to modifying its size or shape via direct manipulation which updates the underlying program to reflect the changes. Their central theme is that users do not have to choose between direct manipulation and programmatic systems

### 4.4 Live Programming

## 5 EVAULATION

### 5.1 Example Gallary

### 5.2 User Study

### 5.3 Cognitive Dimensions Of Notation

## 6 RELATED WORK

## 7 CONCLUSION AND FUTURE WORK

## REFERENCES

[1] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. [n.d.]. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu HI USA, 2020-04-21). ACM, 1–12. https://doi.org/10.1145/3313831.3376442