# Crowd-Sourced Wrapper Construction with End Users

Steve Gardiner

September 25, 2015

Committee Members:

| | |
|---|---|
| Anthony Tomasic | (ISR/LTI) |
| John Zimmerman | (HCII) |
| William Cohen | (MLD/LTI) |
| Michael Franklin | (AMPLab, UC Berkeley) |

# Contents

**Abstract**

The web contains a tremendous number of data sets presented visually, which computers cannot currently read. Most people, however, understand the data sets with little difficulty, suggesting the potential for applying the techniques of crowd-sourcing to the problem of understanding web data sets. In this thesis we study several issues with respect to crowd-sourcing a collection of wrappers, or small programs mapping data sets to their logical structure, from a crowd of end users. We pay special attention to the majority of users who are not programmers.

We present a prototype system, Mixer, that allows end users to demonstrate and execute repetitive ad hoc data retrieval actions over multiple data sources. The evaluation of the prototype suggests that end users, under the strong assumption that input to the individual query systems, as well as their output, is fully understood, are able to construct and combine data from multiple data sources. Furthermore we present another prototype system, SmartWrap showing that end users, explicitly including non programmers, can demonstrate actions sufficient to construct for a data set a wrapper, i.e. the instructions needed to understand the data set. A pilot crowd is able to construct wrappers for most requested data sets, but gives no guidance that the wrapped data sets are useful or relevant to anyone. To narrow the search for relevant data sets we turn to an audience that theory predicts will make use of additional structure in web pages: blind people. We present the theory and the results of a preliminary study demonstrating the concrete benefits non visual users of the web stand to gain from increased structure in web pages and more specifically from the introduction of web tables in place of template-driven visual data sets.

# Chapter 1

# Introduction

There is a vast amount of structured information available in data sets on the internet, but most of it is not in machine-readable form. Instead, the information is encoded in a way that is easily decoded visually by web users. In this thesis, we explore several avenues for gathering and using structured data sets via a crowd, and present several prototype systems we have built to investigate those avenues.

At a high level there are three potential paths to a web where a large proportion of data sets are machine readable. First, web authors could encode machine-readable metadata in the pages. This approach underlay much of the work on the Semantic Web. While the raw amount of structured metadata available on the web appears to be growing [76], the web itself is likely growing at an even faster rate. One reason many web authors do not choose to embed metadata in their pages is that they do not perceive any benefit, monetary or social, in doing so. A second path is for computer systems to automatically markup web data sets with semantic information. Unfortunately this is currently beyond the current state of the art. In Section 2.5, we detail some of the results from the literature describing the difficulties and limitations of current fully automatic approaches. A third approach, studied

here, is to attempt to connect with a crowd of third-party internet users to crowd-source a large number of wrappers.

A central concern in crowd-sourcing is understanding and leveraging the motivations of the crowd in question. In this thesis we detail several prototype systems we have built and evaluated in an attempt to understand each crowd's potential for contributing to the global goal.

In Chapter 3 we describe the Mixer system, which allows nonprogrammers to automate repetitive ad hoc information retrieval tasks over one or more wrapped sites using Programming by Demonstration techniques. The system was designed for users who had not acquired programming skills and did not perceive constructing programs as part of their role. We evaluated the system with a group of administrative assistants. We found that while they were able to construct the desired result set with the tool, they struggled to connect the tool conceptually with tasks they encountered outside the lab.

In Chapter 4 we describe the SmartWrap system, which allows nonprogrammers to create wrappers for web data sets by demonstrating the first two rows of a spreadsheet-like interface. We evaluated the system first in the lab, with a small group of participants, about half of whom had a zero exposure to programming. We found that lab participants exhibited a high level of success in constructing wrappers for the lab pages. A larger follow-up on Mechanical Turk found that many turkers were able and willing to construct wrappers for a relatively modest cost.

Bonnie Nardi [78] quotes the paradigmatic non-programmer[1] as saying "I didn't have much time, so I did it the long way." Mixer and SmartWrap were attempts to extend programmatic capability to people lacking programming training. Even when such users are able to understand the benefits of the programs they construct, they struggle to connect those benefits to tasks

[1]The countersign from the compulsive programmer is "I would rather write programs to help me write programs than write programs"

they face in their daily lives. Although they may estimate that a tool such as Mixer or SmartWrap could save them time on a given task, they perceive these time savings as a risky proposition and instead choose to perform the task in a familiar, risk-free but potentially more time-consuming manner.

Fundamentally, using and combining web data sets is not that difficult for sighted web users. In the aggregate, users spend a lot of time on these tasks in a way that is theoretically addressable by automation, but for each individual task the time-savings are not appreciable.

In Chapter 5 we introduce the enTable system, which applies the efforts of users of SmartWrap towards the problem of improving web accessibility of data sets for blind internet users. In contrast to sighted users, blind web users[2] do perceive an appreciable difficulty in using the web as it is provided by web authors. The literature, as detailed in Section 2.7.1, indicates that using the web through a screen reader takes about 12 to 15 times as much time as using the web visually. EnTable addresses a part of this discrepancy by rewriting difficult, incomprehensible template layouts as simple tables.

In Chapter 6 we describe in greater detail the problems that the use of templates cause for blind access to the web, and how enTable is designed to address them.

Chapter 6 states the case why wrappers will, in theory, improve the situation for blind people on the web. The discrepancy between sighted and non visual use of the web, however, is the result of many barriers for non visual web use. Chapter 7 attempts to quantify the magnitude of the barrier resulting from the use of templates to present data sets without machine-readable metadata. Based on the performance of screen reader users on data sets with and without appropriate metadata, we estimate that providing machine access to the structure may cut the time required for non visual access

[2]the same issues also affect low vision users to a different degree, but we focus here on people who experience the web aurally, through a screen reader

by as much as one half. As failure to include machine-readable metadata inconveniences blind users to the extent demonstrated in Chapter 7, the inclusion of machine-readable metadata in web data sets served to American citizens could potentially (depending on legal interpretations of the American Disabilities Act, which are beyond the scope of this thesis) be stipulated in existing U.S. law.

# Chapter 2

# Related Work

## 2.1 Forms and Spreadsheets

Malone et al [73] note the potential of semi-structured forms as a means of expressing human practice and intention in a manner that is amenable to agent assistance. Their work focuses on structuring email conversations so that agents can assist in the coordination of human activities, essentially providing a mechanism for the agent to eavesdrop on the human communication. VIO [109] complements Malone by providing the reverse: a form mechanism whereby users are given insight into the actions of the agent, and hence the opportunity to identify and repair agent errors.

Nardi and Miller [79] build on the work of Lewis and Olson [70] in singling out spreadsheets, which can be viewed as frameworks for the creation of ad hoc forms, as an emblematic context where people routinely "program", in the sense that they perform nontrivial computational behavior. Nardi and Miller delineate several specific aspects of spreadsheets which render them particularly acceptable to end user interviewees. First, the computational paradigm of spreadsheets matches the way the end user conceptualizes the task; Norman [81] characterizes this alignment as bridging the "Gulf of Execution"

between the user's conceptualization of the goal and the system's formalism. In particular, the high-level functions provided by the spreadsheet shield the user from the difficult task of "synthesizing" the desired functionality from simpler primitives. Secondly, spreadsheets compactly represent the entire task in a single tabular view, often on a single screen.

Cypher [25], coordinating with a group of early adopters to develop a real-world tool for web automation, reports that the administrative end users indicated that the only input necessary was a humble CSV. This corroborates the assertion that users feel comfortable with their ability to provide high-quality curated spreadsheets – they were more concerned that the tool would propagate unwelcome actions on the web than that the input CSV contained flaws.

Google Fusion Tables [44] offers users a low-viscosity path from spreadsheets to relations featuring more powerful database operations. Users can simply upload existing spreadsheets into the cloud, and the system will guess schema information for the uploaded table. When users discover a need to execute complex queries, or joins, over their Fusion Tables, then they can "pay" for it "as they go" [72]. That is, they can perform a small amount of overhead work to enable those operations; users who do not need (or do not yet know that they need) database operations can continue to treat their tables as simply spreadsheets in the cloud. Our work complements this effort, providing a service to the subclass of users who create spreadsheets from web data, in exchange for which the users provide insight into the content and composition of the spreadsheets.

## 2.2 End User Programming

Nardi [78] characterizes end users as users who eschew tasks, such as programming, which they perceive as irrelevant or distracting to the task at

hand: "I didn't have much time, so I did it the long way." She poses this standpoint on the other extreme of a spectrum from that of inveterate programmers, who say "I would rather write programs to help me write programs than write programs." In Nardi's view, an end user is someone who perceives the task at hand as the job, and does not view automating the task as part of that job.

Rode et al [89] provide empirical support for this view in an examination of automation in the household. They find that women can successfully set an oven to cook a meal at a selected temperature at a later time, that men can successfully set a VCR to record a program on a selected channel at a later time, and that each failed at the alternate task. Abstractly, both devices allow users to instruct a device to turn on at a specified time and run for a specified duration, and to set the state of a specific feature: the channel of the VCR and the temperature of the oven.

Ko et al [66] reiterate the distinction between end users and novice programmers, defining end user software engineering as software creation performed by anyone for idiosyncratic or ad hoc purposes. This definition suggests that end user programming systems can be helpful even for professional programmers, who might wish to avoid devoting programming effort towards "one-off" tasks.

## 2.3 Programming by Demonstration

The goal of a programming-by-demonstration (PBD) system is to produce a program based on a handful of demonstrations of the output of the program. There are several challenges surrounding successful implementation of the idea, most of which find expression in web automation PBD systems.

One challenge surrounds the issue of how to communicate the induced program back to the user. Systems which produce code in a general-purpose

programming language (e.g. Chickenfoot [11]) offer the attractive property that they allow a user to express any programmatic sequence. On the other hand, general-purpose programming languages are difficult to understand and modify without extensive knowledge of the language. Nonprogrammers might find them incomprehensible and possibly intimidating.

On the other extreme, some systems (e.g. Vegemite [71], Mixer [42]) do not explicitly represent the program to the user at all. This does not intimidate users. But it does make it difficult for them to understand when the program makes errors, and very difficult for them to repair the errors.

CoScripter [68] takes a middle approach, coding the induced actions in a "sloppy" programming language designed to resemble natural language. Users can readily understand the intent of the instructions, and can make simple modifications to them. Sloppy interpreters are permissive, attempting to interpret user requests as formal programming statements. Without full natural-language understanding (which is known to be hard), however, a system is still quite likely to receive input from users which it cannot interpret.

Wrangler [64] takes a hybrid approach. In response to a user action, Wrangler generates a ranked list of English-like descriptions of possible generalizations of that action, asking the user to select the appropriate generalization. Users are not required to understand the statements of a formal programming language, and simultaneously all user actions have a straightforward interpretation in the programming system.

Lau [67] examines in depth the application of Machine Learning to PBD systems. Using a Version Space argument, she shows that the number of examples a learner needs to unambiguously delineate the proper program often outstrips by several orders of magnitude the number of examples an actual user will have the patience to provide. Often the learner will need several hundred examples and an individual user will want to provide fewer than a dozen. This order-or-magnitude gap between the sample size required

by a learner and the sample size provided by users provides part of our motivation for dividing the work of generating a sample set among a large crowd of workers. Each crowd worker can be expected to contribute only a "user-sized" set of training examples while the learning system receives access to a "learner-sized" set.

Mixed-initiative research focuses on advancing methods for collaboration between computer agents and people where each party has its own knowledge, ways of reasoning, and abilities to understand and act in order to advance toward a common goal [1, 14]. Many issues remain to be answered, including several interrelated needs with respect to interaction between agent assistants and people [33]:

**Awareness** knowledge of problem and goal must be shared by human and agent

**Task** roles and responsibilities must be shared between human and agent

**Communication** both human and agent must be able to express knowledge and needs.

PBD interfaces present a particular challenge with respect to the awareness issue: the user and the system have a fundamental mismatch with respect to the goal of the interaction. The central goal of a PBD system is to infer a program from the user's actions; for the user the construction of the program is subsidiary, at best, to the goal of completing some task. As noted above, Rode et al [31] observe that users are far less successful in performing programming tasks outside their perceived area of responsibility. Consequently, in our work we attempt to avoid presenting the user with tasks that feel like programming.

The task issue concerns the division of action between humans and agents. The principal actions of a PBD session [18] are program demonstration (or creation), program invocation, and program execution.

The communication issue arises in a couple of ways from what Cypher [9] calls the classic challenges of PBD: (1) inferring the user's intent; and (2) presenting the created program to the user. The first challenge concerns the user communicating with the system via the demonstrated actions, and the second challenge concerns the system communicating the recorded action sequence to the user.

The first challenge arises because the user's actions usually insufficiently delineate a unique program, a point illustrated by Lau et al with an explicit version space argument [19]. PLOW [2] receives richer input from the user by eliciting and utilizing natural language explanations for the user's actions. Wrangler [16] asks the user to select after each action the statement in the implementation language corresponding to the level of generalization required.

As to the second challenge, Modugno and Myers [75] further delineate the communication role played by the program in PBD systems, as a list of opportunities presented to the user:

1. the user can confirm that the program will behave as desired;
2. the user can correct or generalize the program; and
3. the user can store all or part of the program for later use or modification.

Although many PBD systems outside the web context communicate the program in forms other than as lines of code, the code approach is the most common in web PBD systems. Chickenfoot [11] records web actions as general JavaScript. CoScripter [68] chooses a slightly more user-friendly approach, representing the program in a "sloppy" or natural programming language. Query-by-Example and Office-by-Example [110] utilize a form as a shared communication structure, but require a user to understand and specify programmatic variable structure within the forms.

## 2.4 Tables

Tables are a commonly used mechanism in print material for presenting data about a series of msotly homogenous items. Wright and Fox [105] summarize some ergonomic studies with respect to how people use print tables. They report that some mechanisms for saving space in print tables tends to increase consultation time of the information in the table. They also find that many people struggle with information left implicit in the table, and benefit from more information being explicitly stated.

Wang [60] proposes a table formalism general enough to faithfully represent the information in about 95% of tables in four different print domains. This formalism was used by a number of systems. Hu et al [57] report that even the co-authors had difficulty agreeing on a single ground-truth representation for many of the tables under consideration. Jha and Nagy [63] constructed a notation tool allowing users to associate items in a table with the formal parts of the schema in Wang's notation. They report that users needed over an hour of training in order to create reliable markup for simple tables.

The specification of the `<table/>` tag in HTML seems to have followed a similar evolution. Tabular data was adopted in HTML 3.0[1], inspired heavily by the implementation from the CALS Table Group [8], which was created to be flexible enough to represent tables in a large set of manuals used by the U.S. military.

## 2.5 Wrapper Induction

Early wrapper work [37] learned extractors from text representations of the HTML of pages, but later work consistently showed the value of consider-

[1] `http://www.w3.org/MarkUp/html3/`

ing structural information (from the DOM), visual layout information [5], and styling information (from CSS). Recent work [46, 27] gets good results by considering features rendered as XPath [19] expressions, encompassing a fairly rich set of structural and textual features.

A key question related to wrapper induction is how many instances must be labeled to induce high-quality wrappers for a set of pages. Cohen, Hurst, and Jensen [21] report that about two-thirds of attributes in their corpus could be reliably extracted based on provision of a single labeled example of the attribute. Approximately 80% could be extracted given at most three examples. Gulhane et al [46] corroborate the sufficiency of relatively few examples; they report that their annotators, tasked with fully labeling relations from a training set until the generated wrapper extracts the remainder of the training set, labeled an average of 1.6 example relations.

Another key concern is how quickly websites change, rendering previously-induced wrappers ineffective. Gulhane et al [46] poll wrapped sites periodically for detectable changes, and derive an estimate that "approximately 2% of websites experience some sort of page structure change each day," implying a site remains without detected change a mean time of 50 days. They then present an algorithm for distinguishing changes needing new labeled examples from more innocuous changes. Dalvi et al [26] estimate that the script generating a given website will change "two or three times per year," indicating a mean time between change of 121 to 183 days. Differences in methodology may account for some of the discrepancy between these estimates, as well as differences in the populations of pages watched.

Most work on wrappers assumes the existence of high-quality labeled pages created by trained annotators (often the researchers themselves), and many researchers [20, 46] cite the labeling of examples as a bottleneck in the scale of the system. A small number of systems present a user interface designed to make it easier for annotators to label relations in the pages.

Lixto Visual Designer [5] allows users to point and click on the first example of an attribute in a page, and then refine the resulting selection by iteratively adding conjunctive conditions or new disjunctive alternatives. Users would seem to need a firm grasp on Boolean logic, which nonprogrammers have been observed to struggle with [45]. Irmak and Suel [61] provide a user interface for interactively specifying a particular desired selection from the larger set of tuples available on the page. They leave aside the issue of specifying the schema and the superset of tuples, suggesting that they expect users to understand schemas and types. Based on the user's selection of a tuple, the system responds with a ranked list of potential sets of tuples and asks the user to select the most appropriate set.

Wrangler [64] employs a similar interaction in the context of end user programming of data cleaning scripts. When the user performs a cleaning operation on a cell of a spreadsheet, Wrangler responds with a ranked list of generalizations of that cleaning operation and asks the user to choose the most appropriate generalization. For example, a user who deletes a prefix consisting of a single token from a cell might be asked to choose whether to remove only that fixed string from all cells in the column, or to uniformly remove the first token, or to act only on the first cell, leaving all other cells in the column unmodified. A user study verified that users familiar with data management were able to effectively use this technique to clean data more quickly than when working manually.

## 2.6 Crowdsourcing and Human Computation

Luis von Ahn kicked [101, 103, 102] off the present wave of interest in models of human computation by supplying proofs of concept system that internet users would supply solutions to difficult AI problems in exchange for access to restricted resources (CAPTCHA) or just for the fun of playing a game

(ESP game, etc). Gamification as a general strategy for eliciting cognitive effort from a crowd of human users has proved fruitful, but some problems are easier to make fun than others.

Quinn and Bederson [85] recast the design space for crowdwork systems in terms of trade-offs between cost, quality, and speed, or the rate at which crowdworkers will complete the available tasks. In this formulation, gamification is one of several means to lower cost, increase speed and increase quality of the crowdworker output.

Understanding the effects of the many parameters of possible crowdworking configurations is a large subfield of research. Mason and Watts [74] examined the effects of monetary compensation on Mechanical Turk workers. Among their findings were that increasing payment for a task did tend to increase the speed with which Turkers completed the task, but did not generally tend to increase quality of submissions.

MonoTrans [55, 56] explored the possibility of gathering machine translation data using plentiful monolingual workers instead of more rare bilingual speakers. By increasing the worker pool, MonoTrans was able to gather data more cheaply and more quickly than when relying on bilingual speakers, but careful design was required to avoid a dramatic drop in quality.

## 2.7 Web Accessibility

The problem of Web Accessibility generally is to make the web more usable by people with disabilities. Since disability takes many forms [49], the field is quite broad. Here we focus only on people with visual disabilities.

People with visual disabilities generally make use of several types of assistive technology. People with low vision often use screen magnifiers, such

as ZoomText[2] or MAGic[3], which magnify the screen by a large amount and allow the user to pan around within the much larger screen surface. People with little to no vision often use screen readers, such as JAWS[4], Windows-Eyes[5] or the free product NVDA[6], which read the contents of the web aloud using a synthesized voice. Screen reader users who read Braille also might use refreshible Braille displays, which display tactilely the same text as the screen reader reads; a common use is to disambiguate homophones or to check the spelling of unfamiliar words.

Two approaches towards web accessibility which are relevant here are as follows. First, the field's understanding of how web pages can be made usable for the visually impaired can be distilled into recommendations to be used[7] by web designers. Second, a third-party can re-write, or transcode, a web page to increase its utility for visually impaired users.

This section proceeds as follows. First, we review some literature establishing that the web is significantly less usable for people with visual impairments than for sighted users. Second, we compare some published results on reading speeds, establishing that website structure is responsible for a large portion of the performance gap. Third, we review some transcoding approaches to improving web accessibility. Finally, we briefly describe the legal status of web accessibility in the United States.

### 2.7.1 The Screen Reader Performance Gap

The added inconvenience and difficulty for blind web users over sighted users is persistent and manifest, and stems from numerous causes. Kevin Carey [15]

[2] `http://www.aisquared.com/zoomtext`
[3] `http://www.freedomscientific.com/Products/LowVision/MAGic`
[4] `http://www.freedomscientific.com/Products/Blindness/JAWS`
[5] `http://www.gwmicro.com/Window-Eyes/`
[6] `http://www.nvaccess.org/`
[7] or, more often, ignored

argues for quantifying the barriers in terms of the discrepancy between non visual users of a tool versus their sighted peers, in terms of success rate at a certain task and time taken to achieve that success. He argues that concrete measurements can move the conversation from qualitative notions of equitability and into quantitative discussions of acceptable parameter ranges. As an additional benefit, barriers with measured impact can be prioritized relative to each other.

Several studies measure the performance differential of plausible tasks by blind users relative to control sighted users. Most of them report the same metrics of time on task and success rate. Gunderson and Mendelson [47] compare the time on task between pairs of sighted and people with low vision, matched by computer expertise; all participants eventually succeeded on all tasks. They report that the visually impaired participants spent 50%-200% more time on most tasks than their sighted peers, but that HTML tables presented a clear outlier. For lookups in a table, visually impaired users took about 1000% more time: "the table lookup task was trivial for the sighted subjects, but almost impossible for the visually impaired subjects." Pernice and Nielsen [84] conducted a large study, asking 60 participants, equal parts with full, low, and no vision, to complete four tasks. They report that the average blind participant successfully completed half a task and devoted 16:46 completing it. The average low vision participant successfully completed about 0.9 of a task in 15:26. The average sighted participant successfully completed about 3 tasks in about 7:14. A simulated user working at the reported rates for blind, low vision, and sighted participants would successfully complete in one hour 1.8, 3.3 and 26 tasks, respectively. This gives a rough estimate that blind people expend about 14.5 times as much time on tasks relative to sighted users, and people with low vision expend about 8 times as much time. Takagi et al [95] report that 5 blind participants complete a task in an average of 536.6 seconds, about 12 times as long

as the 5 sighted participants, who completed a task in an average of 43.6 seconds. Craven and Brophy [24] presented 40 participants, half sighted and half visually impaired, with four tasks and report that the visually impaired participants spent between 1.4 and 4 times as long as sighted participants.

Estimation of relative task performance rates for sighted and visually impaired people is complicated by the necessity to simulate a plausible workload for comparison, over which results are aggregated. This is analogous to the problems encountered in benchmarking computation or database systems, where the results can vary greatly depending on the tasks used for evaluation. Carey [15] touches on this issue, arguing that equitability implies minimizing the performance gap for people with disabilities relative to their peer group, where the peer group will vary with the intended audience of a piece of software or intervention. Gunderson and Mendelson [47] asked their participants to complete one web search, three tasks using the search forms of various sites, and one task involving lookups in a page marked up as an HTML `<table/>`; the last task generated the largest differential between the sighted and low vision participants. Pernice and Nielsen [84] presented their participants with one fact-finding task, one web shopping task, one information retrieval task, and one product comparison task. Takagi et al [95] examined two web shopping tasks. Here, we focus on lookups within data sets presented on a single web page, which are trivial visually but laborious with a screen reader. The literature cited here will serve as a baseline. Since data sets comprise a significant portion of the work load of web users, improvements to data set lookups will have a significant impact on the aggregate performance gap.

| type of reading | sighted speed [16] | blind speed [3, 29, 93] |
|---|---|---|
| **Scanning** | 600 Wpm | |
| **Skimming** | 450 Wpm | 500-580 Wpm |
| **Reading** | 300 Wpm | 290-350 Wpm |

Table 2.1: A table summarizing research results about reading rates in sighted and blind people

### 2.7.2 Visual and Audio Reading

At first blush, it seems plausible that a significant part of the slower performance of blinded internet users in comparison to sighted internet users stems from the slower comprehension of web sites through the auditory channel than through the visual channel. Research on reading rates, however, does not support this notion.

Table 2.1 summarizes some research results on reading rates, juxtaposing results from research into sighted reading with those from research into listening speeds for blind people. The research is reported in different units, and were converted for comparison using the formulae provided by Carver [16]. The second column comes from Carver's summary[8] of several decades of research into reading [16]. Asakawa et al [3] report that blind participants were able to understand synthesized speech with 90% recall at a speed of about 300 Wpm, and to aurally skim with 50% recall at about 500 Wpm. Dietrich et al [29] similarly report that blind participants were able to understand speech at about 290 Wpm with about 90% recall, and speech at about 580 Wpm with about 50% recall. Dietrich et al additionally report from fMRI studies of the participants that blind people may process ultra-fast spoken text using a part of the visual cortex which is devoted to visual processing in sighted people. Stent et al also find that experienced users of screen readers can recall more than 90% of text at 300-350 wpm, and recall at leat 50% of

[8]The data come from Carver's Table 2.2; Carver uses the portmanteau "rauding" to describe reading with full understanding ("auding")

text at around 500 wpm.

There are many limitations with attempting to do any direct comparisons of the numbers in Table 2.1. To name only one caveat among many, the reading rates from Carver and Stent are for English text, the listening rates from Asakawa are for Japanese speech, and those from Dietrich are for German speech. Here, we will take the data as merely indicative that it is difficult to stipulate that the discrepancy between the speed of reading versus listening is large enough to cause the discrepancy described above.

### 2.7.3 Transcoding for Web Accessibility

Many research projects have pursued a transcoding approach where websites presenting accessibility challenges are re-written in a more usable form. The Accessibility Commons [65] proposes to serve all of the transcoding metadata packages to screen reader clients from a unified architecture; the paper also presents useful pointers to the various projects they propose to unify. AccessMonkey [7] extends the GreaseMonkey[9] tool, which allows arbitrary re-writes of web pages written in the JavaScript programming language, and allows users to apply a collection of scripts improving the accessibility of various pages.

The Social Accessibility project [90] used a crowd to fix accessibility problems on websites, and reported large numbers of users were willing to contribute their work to the project. The problems addressed by Social Accessibility are the specification of appropriate alternative text for images, the redesignation as semantic headers of header text with shared visual styling, and adjustments of reading order. All of these improvements bear on the abilities of screen reader users to reach the desired content more quickly.

[9] `http://wiki.greasespot.net/Greasemonkey`

### 2.7.4 Web Accessibility and U.S. Law

United States law requires providers of certain types of services to accommodate people with disabilities. The following account draws heavily from Peter Blanck's book [10] focused on web accessibility for persons with cognitive disabilities.

The Rehabiliation Act of 1973[10] was augmented in 1986 with the addition of Section 508, providing that U.S. federal and state governments should make use of information technology accessible to everyone, including those with visual and other disabilities. The original Section 508 included little mechanism for enforcement or oversight, and in 1998 was amended (as part of the Federal Electronic and Information Technology Accessibility Compliance Act). The new Section 508 included concrete requirements for accessible websites and charged the Access Board with reviewing and revising those standards for the information technology decisions of governmental entities. The standards published by the Access Board also served as inspiration and guidance for the standardization of accessibility guidelines at W3C, first published [17] in 1999 and revised [14, 88] in 2008.

The WCAG 2.0 Standard [14, 23, 22] sets forth several guidelines at three increasing levels of conformance, "A", "AA" and "AAA". At the minimum "A" level of conformance, Guideline 1.3.1 stipulates that "Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text." To conform, WCAG puts forth the technique "H51: Using table markup to present tabular information" [22], described in the accompanying draft as follows:

> The objective of this technique is to present tabular information in a way that preserves relationships within the information even when users cannot see the table or the presentation format

[10] 29 U.S.C §701

> is changed. Information is considered tabular when logical relationships among text, numbers, images, or other data exist in two dimensions (vertical and horizontal). These relationships are represented in columns and rows, and the columns and rows must be recognizable in order for the logical relationships to be perceived.

Under a narrow reading, the technique would apply only when the author presents the data using rows and columns. Under a broader reading, the technique applies whenever a logical relational structure exists within the pieces of data presented. In Chapter 6 we present the argument that the broader definition would be assistive to people who use screen readers, and in Chapter 7 we empirically demonstrate the assistance provided by conveying the relational structure to the screen reader program.

When the HTML `<table/>` tag is used, WCAG 2.0 provides several other techniques for compliance:

**H39** Using caption elements to associate data table captions with data tables

**H43** Using id and headers attributes to associate data cells with header cells in data tables

**H63** Using the scope attribute to associate header cells and data cells in data tables

**H73** Using the summary attribute of the table element to give an overview of data tables

**H79** Identifying the purpose of a link in a data table using the link text combined with its enclosing table cell and associated table header cells

But, again, these techniques do not apply to data which are not contained in a "tabular" data table.

The requirements of Section 508 that websites conform to WCAG 2.0 (conformance level "AA") apply only to governmental websites. Title III of the Americans with Disabilities Act of 1990[11] requires all public accommodations and commercial facilities to provide reasonable accommodation to people with disabilities. Many early legal cases testing the reach of the ADA focused on the standing of the plaintiffs, i.e. determining whether they qualified to a strict definition of having a disability. For example, in *Sutton v United Airlines*[12] the court found that the legally blind plaintiffs could not seek claims of wrongful termination by virtue of their visual disability because, after correction, their vision was normal. As people with normal (corrected) vision they had no standing to file claims under the ADA, even though the court did not contest that they were fired based on their visual disability. In 2008, Congress passed the ADA Amendments Act, directing courts to interpret broadly the definitions of disability.

The National Council on Disability (NCD), an independent federal agency, holds the view [40] that the ADA requires all public commercial websites to be accessible to people with disabilities. Wicker and Santoso [104] argue in the Communications of the ACM that the right to equal access to websites is part of the general human rights, including those guaranteed to Americans by the ADA.

The courts, however, disagree with the NCD's assertion that all commercial websites are subject to the accessibility requirements of the ADA. Instead, they find the act applies more narrowly to entities that operate a "place of public accommodation," as stated (and partially enumerated) in the text of the Act. Courts found[13] that the ADA did not require `southwest.com` to be accessible, because it was not a place of public accommodation enumerated in the ADA. However, courts have held that the websites of the

[11] 42 U.S.C. §§12181-12189

[12] *Sutton v United Airlines* (10th Cir 1999) 527 U.S. 471

[13] *Access Now, Inc. v Southwest Airlines, Co.* (SD Fla. 2002) 227 F. Supp. 2d 1312

corporations Target[14] and Disney[15] must be accessible to people with visual impairments, by virtue of the websites having a connection, or "nexus", to a physical place of public accommodation. Both Target and Disney settled with the plaintiffs and agreed to incorporate accessibility standards into their websites. Target additionally entered into an agreement[16] with the National Federation of the Blind to test and certify the redesigned websites.

Different courts have decided differently about the applicability of the ADA to commercial websites, and the final determination may eventually be decided by the US Supreme Court [10], or by Congressional amendment to the statute. Even in the absence of clear US law, however, some companies choose to increase the accessibility of their websites in the wake of legal decisions. For example, after the *Target* decision, Amazon entered into an agreement with the National Federation of the Blind to make its website more accessible[17]. Similarly, after the *Disney* decision, Charles Schwab[18] entered an agreement to conform to WCAG standards.

The above discussion concerns the applicability of the ADA to commercial websites. Although courts have stopped short of affirming the NCD's view that the ADA applies to all commercial websites, they have admitted that the ADA applies to some of them, e.g. those with a nexus to a physical place of public accommodation. The following paragraphs summarize some court findings with respect to what the ADA may require when it does apply. In each paragraph, we summarize the legal findings, then extrapolate their relevance to the web context.

[14] *National Federation of the Blind v Target Corporation* (CA Sup. 2006) 452 F. Supp. 2d 946

[15] *Shields v Walt Disney Parks and Resorts U.S., Inc.* (CD Cal. 2011) 279 FRD 529

[16] `http://pressroom.target.com/news/target-nfb-settlement`

[17] `http://www.dralegal.org/impact/cases/amazoncom-structured-negotiations`

[18] `http://lflegal.com/2012/05/schwab-agreement/`

**Accessibility rights can overrule aesthetic concerns** In *Colorado Cross-Disability Coalition v Holister*[19] the defendants claimed that making the stately front stairways of their stores accessible for wheelchair users would impact their efforts to create a "brand design to convey a certain ambience and appearance to make the store inviting as a whole." They argued, since wheelchair users could make use of a side door, that there was no ADA violation. The court found that the ADA requires "equal enjoyment" of the store's experience by people with disabilities, and thus that the entryway must be made accessible unless plaintiffs could show that it is "structurally impractical to do so." In the web context, this would seem to imply that reasonable accommodation means nearly anything short of changes that are demonstrably technically infeasible.

**The ADA can compel providers to undertake nontrivial efforts to support accessibility** In *State of Arizona v Harkins Amusement*[20] the defendants claimed that providing captions and narratives to movies shown at their theater chain would alter the content of its services. The Ninth Circuit court found that a public accommodation must provide auxiliary aids to provide equal experiences to people with hearing disabilities unless they could show that doing so would create unreasonable burden or cost. In the web context, this would imply that extra effort to make webpages understandble by screen reader users could be compelled, unless they could be proven to be prohibitively expensive.

**The ADA's requirement of equal enjoyment goes beyond the minimum of bare accessibility** In *Baughman v Disney*[21], the defendant, Dis-

[19] *Colorado Cross-Disability Coalition, et al v Abercrombie & Fitch, et al, and J.M. Hollister Co.* (D Colo 2011) 835 F Supp 2d 1077

[20] *State of Arizona v Harkins Amusement* (9th Cir 2010) 603 F.3d 666

[21] *Baughman v Disney* (9th Cir 2012) 685 F.3d 1131

ney, argued that providing the same services to everyone in their amusement park ought to suffice under the terms of the ADA, even if access by a patron in a wheelchair is made "uncomfortable or difficult" by its policies. The court disagreed, summarizing Disney's argument as stating "any discomfort or difficulty [the plaintiff] may suffer is too darn bad." The opinion continues "Disney is obviously mistaken. If it can make Baughman's experience less onerous ... it must take reasonable steps to do so." In the web context, this would imply that simply providing a website that works for most users, presumably sighted and able to use a mouse, would be insufficient to conform with the ADA. Rather, steps which can provably make websites less onerous for people with disabilities, so long as they do not incur unreasonable costs, must be taken. In the Target case, cited above, the court drew precisely this analogy to argue that blind web users must be accommodated under the ADA.

**Users' accessibility concerns can override rightsholders' copyright claims** The United States Copyright Act[22] generally holds that content creators hold exclusive rights to their creations for a certain period of time. In 1996, however, the Chafee Amendment[23] was passed, excepting from copyright claims works for "blind or other people with disabilities." Based on the provisions of the Chafee Amendment, the National Library Service for the Blind and Physically Handicapped[24] is permitted to record any "nondramatic" written material as audio for use by people unable to use print, even when such material would otherwise be covered by copyright. In *Authors Guild v HathiTrust*[25] the courts affirmed that HathiTrust, a partnership between several university libraries and Google, had the right to digitize other-

[22] 17 U.S.C. §§101-810
[23] 17 U.S.C. §121
[24] `http://www.loc.gov/nls/`
[25] *Authors Guild v HathiTrust* (2d Cir 2014) 755 F.3d 87

wise copyrighted material on a mass scale. In the web context, these findings indicate that it would be legal for a third-party to alter a website to make it accessible, despite the copyright claims of the website's author. Use of the third-party alteration, however, may be limited only to blind or other people with disabilities.

The United States statutes discussed above regulate websites doing business with United States citizens, regardless of where the websites are hosted. Thus when a website is determined to be subject to the ADA, it does not matter where the server is located, e.g. Target could not evade ADA provisions by hosting its website in Canada. We close this discussion by briefly noting disability law outside the United States.

Similarly to the provisions of Section 508, several countries, including India, Japan, and Hong Kong, require that governmental websites be accessible by people with disabilities [10]. The European Commission has a Mandate 376, requiring that governmental IT acquisitions be usable by people with disabilities. The Canadian Charter of Rights and Freedoms asserts that the government shall not discriminite against its citizens in provision of information. In *Jodhan v Canada*[26] the court found that the government's inaccessible websites had discriminated against the needs of the blind plaintiff.

Australia's Disability Discrimination Act seems to go further, prohibiting discrimination on grounds of disability by all commercial entities [10]. In *Maguire v SOCOG*[27], the court found that the Olympic Organising Committee's inaccessible website had discriminated against the blind defendant.

The United Nations Convention on the Rights of Persons with Disabilities, from 2008, affirms that web equality is "indispensable to other fundamental human rights such as freedom of speech." [10]. Many countries[28] have signed

[26] *Jodhan v Attorney General of Canada* (2010) 2010 FC 1197
[27] *Maguire v Sydney Organising Committee for the Olympic Games* (2000) No. H 99/115
[28] the United States is not among them

and ratified the treaty. The Marrakesh Treaty, negotiated in 2013, provides for exceptions to copyright similar to those provided by the Chafee Amendment, but for use by people outside the United States. Harpur and Suzor [50] discuss this treaty in the context of addressing the "book famine" whereby, worldwide, people who cannot use print have access to only 1% of all published books. They conclude that signatories to the Marrakesh Treaty "will be obliged to introduce a minimum baseline exception that allows visually-impaired people to receive, make, and use accessible copies of works" but that more international coordination is needed in the face of the vast scope of the book famine.

# Chapter 3

# Mixer

## 3.1 Introduction

As the size and richness of the web has steadily increased over the last decade, users have ratcheted up their expectations for the scope of information easily available from the web. Web interfaces, in contrast, usually permit access to the information they expose in a highly constrained manner. The gap between the form of the information required and the form provided by the deployed interfaces is bridged in practice by human intelligence. In particular, office workers in an administrative capacity are regularly assigned mundane, repetitive data integration tasks, entailing the gathering of information from several sources into an ad hoc report, often in response to an email [98]. Administrators do not consider these tasks difficult, but they do consider them very tedious. The repetitive and procedural structure of the tasks makes them ripe for automation; however, the actions taken in response to the retrieved information generally require human judgment. This combination of automation and human judgment invites a mixed-initiative approach that weds the administrator's understanding of the desired report with a programmatic agent actually performing the bulk of the mundane

retrieval.

As an example, suppose a university dean wishes to investigate previous collaborations between her university and a certain research lab, and further that she has assigned her assistant the task of finding all professors in the university who have published a paper with someone from the lab. The straightforward solution is to look up in turn each professor's publications in a digital library and store all of their collaborators from the lab in some intermediate location, such as a column of a spreadsheet. This solution, while effective, illustrates well the tedium involved, as it requires the administrator to perform the same series of clicks and copies and pastes, each time with different input, until the output report is complete.

The tedious, repetitive nature of the tasks evokes the concept of programming by demonstration (PBD). The application of PBD to administrative data integration tasks follows from the insight that once the user has shown how to look up a single example, the system has sufficient information about the procedure to look up the rest of the examples. In an effort to realize the promise of PBD in facilitating administrative data integration tasks, we developed Mixer, a Mixed-Initiative PBD system that allows users to train an agent to perform the tasks. Our current implementation builds on our previous Wizard-of-Oz study, which demonstrates the effectiveness of a spreadsheet-like user-created form as a medium of communication between a human user and a simulated computer agent. Users were quite successful in using the mocked-up system, but they struggled with the following issues: (1) specifying 1-to-many relationships in a manner usable by the agent, (2) specifying precision in the retrieved report, and (3) selecting meaningful segments of text on the page. Mixer as presented here addresses these shortcomings, and also incorporates insights from other explorations of web PBD [68, 71]. Mixer presents several innovations over previous approaches. First, Mixer presents a unified modeless interface for integrating data, whether that data

come from one or several data sources. Additionally, Mixer leverages the insights of Mixed-Initiative design to facilitate collaboration between the user and the agent to accomplish the user's goal.

To evaluate our design decisions, we conducted an evaluation with real administrators. The administrators were asked to retrieve multiple items from a single data source and to link information across multiple data sources. The evaluation results show that: (i) Using the Mixer table based interface, administrators can conceive of, create, and use forms that effectively communicate to the Mixer agent both the information they want and the information the agent needs to automate the task; and (ii) administrators recognized the value of automating this type of mundane task and indicated they would incorporate a mixed-initiative tool like Mixer into their work practices.

The remainder of this chapter is organized as follows. First, we describe the design of Mixer. We then describe the study performed and the results obtained. Next, we discuss the implications of the present study. Lastly, we situate this work within the related work in the literature, and conclude.

## 3.2 Design

At a high-level, interaction with Mixer requires the user to construct the first row of a table, and in doing so, the user demonstrates to the agent what information they want and where this information can be found. When this row is complete, the user releases the agent to follow the pattern until the retrieval is complete. If the user desires a subset of this information, they can export the resulting table to a spreadsheet and use the spreadsheet to make the subset they desire selectable. Below we detail an example, depicted in Figure 3.1, of how the interaction works. In the example, the user looks up the names and affiliations of the coauthors of a particular researcher using an online interface which allows accessing this information for one publication

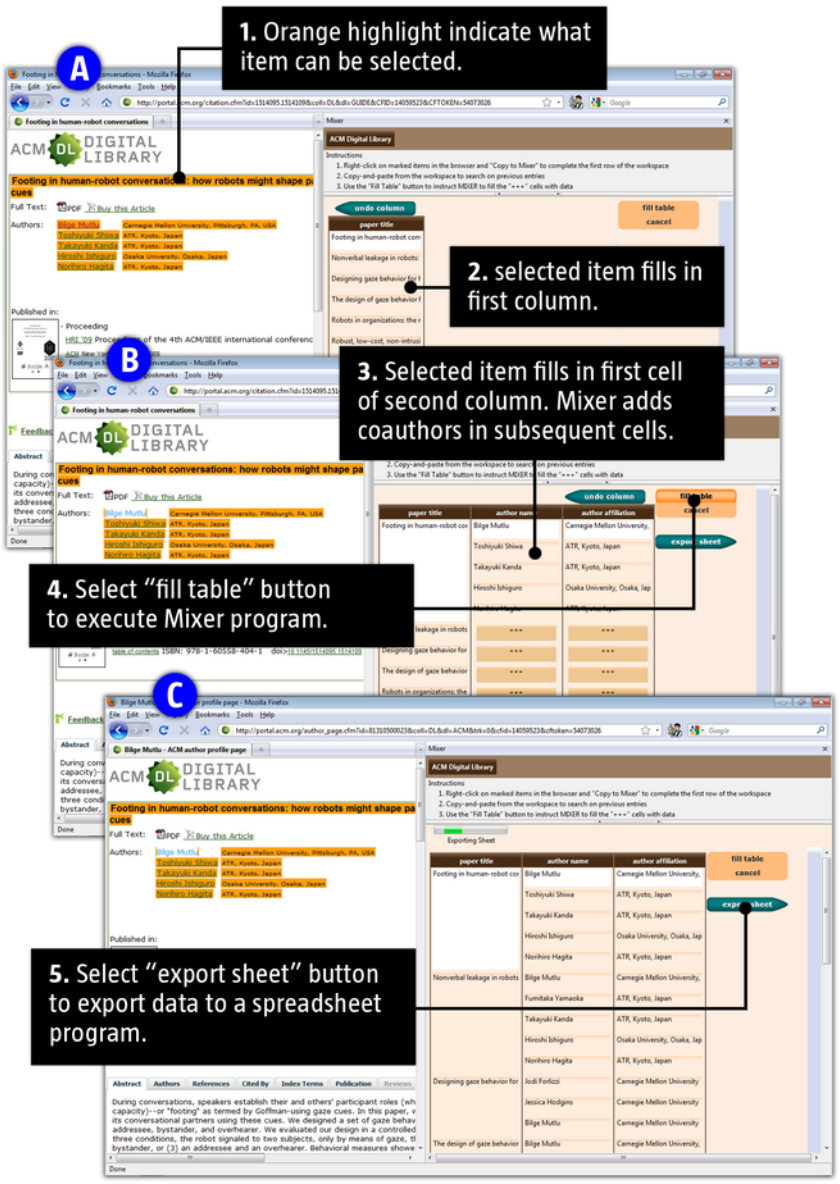


Figure 3.1: A user interacting with Mixer to extract the coauthors of a researcher based on the researcher's papers on the ACM Digital Library

at a time.

Users begin using Mixer by navigating their browser to the first page they wish to retrieve information from (referred to as the target page). In this example the page contains a listing of the publications of the researcher, with a brief description of each publication and a link to a detailed record of the publication. Once there, the user clicks on the Mixer button appearing within the browser's chrome. This click causes two actions. First, the Mixer workspace (referred to as workspace) appears in a frame to the right of the target page. Second, Mixer augments the target page, highlighting any element the agent can accept as an input in orange.

To add an element to the workspace, the user right-clicks the element and selects "Copy to Mixer" in a context menu. In response, Mixer constructs a new column in the workspace, gives it a heading, fills in the first row with the element the user selected, and fills in the remaining rows with all elements that match the user's selection (Figure 3.1A). In this case the user selects the title of the first publication on the list and the agent adds the title "paper title", adds the user-selected publication, and adds all of the remaining publications to the column. The cell at the top of the filled-in column has a white background, indicating a human selection while the cells below have an orange background, indicating that the agent selected these elements. In this example, the user only needs the publication title from the first page; however, if the user required additional elements from this page, right-clicking and selecting "Copy to Mixer" would cause the agent to create additional columns. Earlier Mixer interface designs allowed users to simply copy and paste elements from the target page to the workspace. However, we observed that people had trouble understanding what was "legally" selectable since they could never see the underlying data scheme. In addition, they had trouble copying and pasting text that appeared as a link. In the current design, the highlighted elements on the target page are intended to clarify

what can be selected and the right-click action allows users to add an element that is a link, without navigating away from the target page.

In our previously reported evaluation of the Mixer interaction design, we noted that some participants struggled to create an effective table. Using the example of the publications, many would copy and paste the first publication title into the top of the first column and then they would copy and paste the second title on the list to the top of the second column. To prevent users from making this mistake, Mixer automatically fills in the first column as soon as the first element is selected. Now that the user has a column with all of the publication titles, the user next needs to demonstrate that they also want authors to go with each publication. To advance the task, the user navigates through the publication link causing the target page to change from the publication listing to the publication detail page.

Mixer detects that the user's action depends on the contents of a cell in the workspace, i.e. the link corresponds to the first publication. Accordingly, Mixer begins an implicit loop over all publications in the column. An additional wrinkle which does not appear in the demonstrated task, is when a user must type input from the workspace into a query form on a separate page. In such cases, Mixer prevents direct typing because the agent needs to be shown an explicit connection between the contents of a cell in the workspace and the input to a query. When a cell's contents are copied to the clipboard and then pasted into a form's widget, Mixer is able to deduce the connection. When the cell's contents are simply typed into the widget, however, Mixer cannot be certain that the query input in fact comes from the workspace entry rather than from elsewhere in the page; to avoid this problem, Mixer issues a warning to the user when directly typing into a query form. In a more extreme case, the administrator might modify the contents (e.g. stripping off the first name and using only the last name), making the matching of the contents to the workspace quite difficult. This tension between re-using

a variable by value and by reference has a long history in PBD, see e.g. the discussion of distinguishing constants from variables in Myers [77].

Mixer augments the publication detail page: selectable elements are highlighted in orange. These elements include the publication's authors' names, as well as the venue where the publication appeared (Figure 3.1B). The user right-clicks the author's name and selects "Add to Mixer." In response, Mixer creates a new column, adds the title "author name", fills in the first row with the selected author name, fills in subsequent cells with additional coauthor names available from the current page, and fills in the remaining rows with a dashed line, indicating that the agent thinks the user wants this information for subsequent publications. Similarly, the user adds "author affiliation" information to the table from the same page.

To release the agent to complete the table, the administrator clicks the "Fill Table" button to the right of the last column. In response, Mixer infers and executes a program. In this example the program contains a loop over all publications in the publication listing. Mixer iterates over each publication, one by one. For each publication, Mixer navigates through the link using exactly the same action sequence which the administrator demonstrated, modified only to correspond to the present publication. As each detail page is visited and its result integrated, the browser view shows exactly what is happening, giving the administrator confidence that the result is the same as if the task were performed manually. When the table is complete, Mixer plays a chime, letting the user know the agent is finished (Figure 3.1C).

Administrators frequently want to collect information about a subset of items that meet some criterion, for example the students who were currently failing a particular course. Since administrators are familiar with spreadsheets, we postulated that they would be able to conceptualize the filtering task as composed of the subtasks of retrieving the desired information about the whole group, then sorting on the selection attribute and cutting out all

nonqualifying rows. This functionality is present and familiar in modern spreadsheets, so Mixer does not re-implement it, but rather expects the user to use a separate spreadsheet tool (our experiments used Google Docs).

Our previous simulation of the Mixer interface [108] illuminated the way for the present implementation; however, as noted by Sundström et al [94], the Wizard-of-Oz methodology only allowed a certain level of familiarity with the algorithmic material. The actual implementation explored some new design potential and constraints. One notable example is that the implementation must take into consideration the time taken for the actual retrieval, balancing the time consumed by the network latency inherent in retrieving information from the web with the user's valuable time and attention. The implemented system, unlike the Wizard-of-Oz mockup, does not begin with access to the retrieved data. Instead the user must wait while Mixer replays the demonstrated actions necessary for retrieval. The present design replays the actions in front of the user's eyes; this furthers communication between the user and the agent in that the user understands what is happening and why it is taking time. Relatedly, the present design fills in as much of the table as it can as soon as the user selects a piece of data for inclusion. This refinement allows Mixer to communicate more effectively its understanding of the table as the table is constructed. We thereby lessen some of the issues participants encountered with our previous design, as to how to construct the table.

Additionally, constructing the table as soon as data is available allows more efficient use of the user's time. Specifically, in the previous design users engaged a tool we called the "resolver" to help specify if they wanted a single element, a subset of element, or all elements within a column. In an actually executing system, however, this means the user must make resolver decisions periodically throughout the retrieval process, with lags of unknown length between decisions. In addition, in designing the resolver tool we struggled to

find a way for users to precisely specify what they specifically wanted that did not feel like programming. The current design takes a different tack. It drives users to collect a complete set of data, and then allows them to export the table they have made to a spreadsheet, where they can use the familiar tools of spreadsheets to sort and perform calculations. This design choice gains significant ease of use at the cost of making precision more laborious.

Two other changes from the previous design bear mention. First, whereas the previous design presented the user with the target page and invited her to select any page element, the current design instead pre-highlights the selectable elements on the page. This clearly communicates to the user which actions Mixer will understand, but decreases the ability of a user to apply Mixer to novel pages. Second, we constrain the user to copy and paste data from the workspace into a query page, rather than typing.

## 3.3 Study

We recruited administrators to perform a user study aimed at substantiating the following hypotheses:

- H1: Mixer's table-based workspace interface provides an effective method of communication between the human and the agent for data integration tasks:
  - Administrators can conceive of and express information demands through designing and demonstrating the form of the information in the workspace.
  - Administrators can make sense of, and work with, information retrieved in collaboration with an agent and presented in the workspace.

- H2: Administrators will recognize the benefit of automated data integration and would be interested in using this interface for their work.

In order to test these hypotheses we culled administrative tasks from the suggestions of participants in our previous Wizard-of-Oz study of the Mixer interface [108]. To accommodate privacy concerns, we shifted the tasks to different real-world domains, where we selected isomorphic tasks which pilot participants demonstrated could realistically complete within a 90 minute experiment. Because Mixer is not intended as a walk-up-and-use system, participants were provided with a grounding introductory spiel and an experimenter-directed training task. After the completion of those tasks, the participants were asked to think-aloud while completing the remaining experimental tasks. The experimenter provided no assistance to the participants during the completion of the tasks.

We recruited $N = 12$ administrator volunteers for an experimental session lasting about 90 minutes. They were paid \$15/hour for their time. Volunteers were asked if they had experience with programming, and those who did were disqualified from participation. We began by introducing the tool and acquainting the users with its goals and concepts. The experimenter then introduced participants to the concept of the thinkaloud experimental setup.

Next users completed a preliminary survey detailing their background level of computer usage and expertise. To ensure that users understood the task we asked them to take three minutes to complete as much of a task as they could manually, specifically by copying and pasting directly from the Association for Computing Machinery (ACM) website into a spreadsheet.

Then, to illustrate the use of the system, the participant performed a representative Mixer task with minute direction from the experimenter. The training task was:

- Task 0: Find all researchers from Institution X who published in the

latest conference of Conference Z

Then, one by one, we asked them to respond to messages in a pre-loaded email account. Each message contained a request from a contrived boss for the completion of an experimental task; users indicated completion of the task by replying to the email with their best attempt at the answer. During the completion of the tasks, the participants' actions and audio were recorded using Camtasia for later analysis.

- Task 1: Find all researchers from Institution Y who published in the latest conference of Conference W
- Task 2: Find all coauthors of Researcher R in the last three years
- Task 3: Find email addresses for all members of Club C
- Task 4: Find all coauthors of Researcher S in the last three years

The tasks were mostly from the ACM domain in order to minimize the amount of domain knowledge presupposed or learned in-experiment on the part of the participant. Task 1 was chosen as an isomorph of the demonstration task to cement the participant's understanding of the process of extracting a subset, then using spreadsheet functionality to select the appropriate subset. Task 2 has the same form, but the web interactions are novel, sometimes changing which pieces of information require a new server response. Task 4 is a repeat of Task 2 with different parameters, but introduces a minor complicated factor that Researcher S's first paper is published alone (i.e. the only coauthor is the first author). Task 3 is completely novel in the sense that the output of one website is used as the input of another. Participants were not instructed how to use Mixer to combine data from multiple websites, nor were they alerted that Task 3 had any characteristic different from the rest.

Additionally, Tasks 2 and 4 had two different solution paths. The ACM listing of an author's publications lists all publications with links to pages about the individual papers; alongside the link is a listing of metadata about the paper including authors and publication date. Thus the problem may be solved within a single page, since all needed information is present in the page. Alternately, the problem can be solved by extracting the required metadata from each publication's page in turn.

Following completion of the tasks, participants answered a post-study questionnaire containing the TAM3 [100] (Technology Acceptance Model 3) instrument. TAM3 measures a new technology's perceived usefulness and perceived ease of use. Previous research shows a strong relationship between these two perceptions and eventual system use. TAM3 responses were made on a 7-point Likert scale (1 = "extremely unlikely to use," 4 = "neither," 7 = "extremely likely to use").

## 3.4 Findings

After the participant had correctly communicated the desired behavior to Mixer, Mixer turned control back over with a filled table of data. In 42 (about 88%) of the tasks the participant was able to correctly filter the data and direct the completed form to the experiment's simulated boss. In one case, a participant was unable to do so for Task 1 and gave up; that same participant was able to correctly marshal the data in the subsequent tasks. In four cases, participants needed one or two more attempts to effect the correct answer. In one case, a participant needed five attempts.

Our 12 participants successfully completed all tasks. They eventually constructed all of the necessary tables with the agent's help. Because the experiment required the participants to thinkaloud as they worked, the amount of time participants took to accomplish the task is not meaningful; instead,

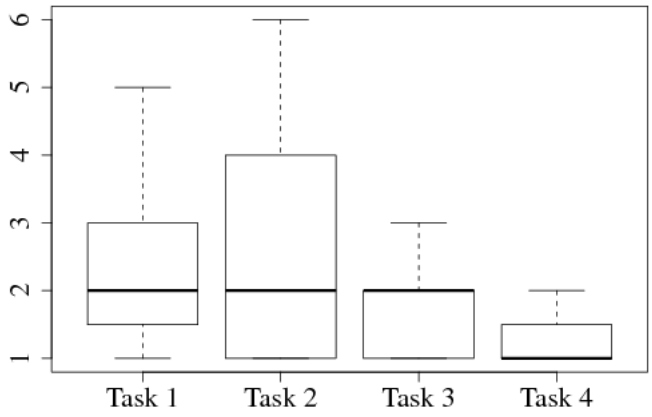


Figure 3.2: Number of attempts to construct the correct workspace

we record the number of attempts they made before they were able to productively turn control over to Mixer. We counted an attempt every time the participant started over with a fresh workspace during the completion of a task. All participants' number of attempts are presented in Figure 3.2.

Participants exhibited some difficulty constructing a workspace table containing all of the information required to complete the task. 17 task attempts failed due to missing query attributes, for example by failing to include students' names when only email addresses were strictly required. Two participants successfully extracted a spreadsheet, only to find that missing selection attributes precluded them from sorting and filtering down to the correct answer. They immediately re-demonstrated the correct workspace without error.

Two participants constructed a table without an attribute explicitly requested, then corrected their oversight. Of the overlooked attributes of all categories, only one instance occurred in the final task. Several participants appeared to struggle with Mixer's expectation that the user would only provide information in the first row, leaving Mixer in charge of filling subsequent rows. Three participants, all in Task 3, tried to continue filling subsequent

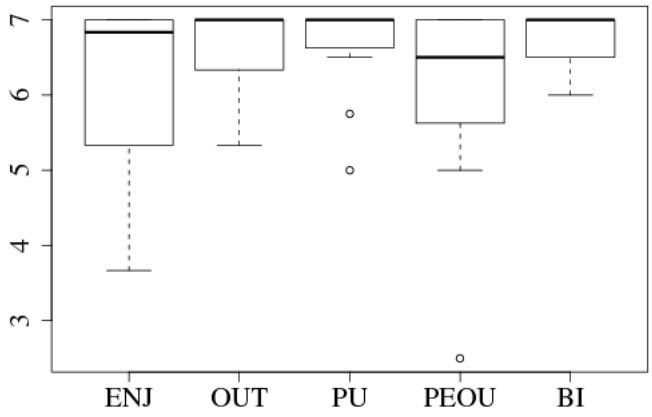


Figure 3.3: Participants' ratings of Mixer along TAM constructs

rows before clicking on "Fill Table."

Two participants, again in Task 3, attempted to search multiple email addresses at once by pasting all the names, separated by spaces, into the search box; in response, the directory application returned no results and the participants started over. One participant attempted to explicitly select a column of attributes from the target page.

Another common breakdown occurred with respect to participants' decision to invoke the program by pressing the "Fill Table" button. One participant chose to restart Task 1 after exporting a table with unfilled cells, i.e. failing to invoke the program at all. Four participants discovered a solution to Task 2 that did not require the use of the "Fill Table" button to complete the task. They then expressed confusion that the "Fill Table" button was inactive. Two of the confused participants precipitously restarted after encountering the confusion. All four participants used the same approach on Task 4 and did not hesitate to complete the task without using the "Fill Table" button. Three additional users discovered this approach while completing Task 4. Each expressed confusion that the invocation button had no effect, but all pressed onward and successfully completed the task.

Eight participants encountered the dialog box warning against typing in a query field. One successfully circumvented the dialog, typing in the information and thereby causing the task attempt to fail. One user attempted to select data not recognized by the wrapper and send it to Mixer.

Figure 3.3 shows participants' responses to the TAM instrument. The Cronbach alpha scores, all above 0.8, indicate that the scores are internally reliable, in the sense that answers to multiple questions seem to measure the same underlying construct.

Many users expressed pleasant surprise at the capabilities of Mixer, using adjectives like "cool", "awesome", and "brilliant." One user said "I want this program. Even if it can't find everybody" and another asked "When can I get this?" All of these laudatory quotes came immediately after the user was able to successfully complete Task 3. Two users praised the visibility of Mixer's practice of showing each visited page as the table is filled. Two participants expressed displeasure at the use of Google Docs for the spreadsheet export; they claimed to be more proficient with Microsoft Excel. During the closing interview, several participants inquired when Mixer would be available to them for their jobs.

## 3.5 Conclusions

Administrators could successfully use Mixer to automate tedious information retrieval tasks. They were successful at creating the first row of a table as a way of communicating to an agent the information they wanted. At first, many administrators struggled to complete a task. Sometimes this was caused by software bugs in Mixer, and sometimes it was caused by participants struggling to conceive of tables the agent could act upon. The agent often needs more context (additional columns) to complete an action than the administrator strictly needs to complete their task. Administrators strug-

gled with including this context, indicating that, initially, they had trouble seeing the problem from the agent's perspective. However, the reduction in the number of attempts needed to successfully complete a task from the first task to the fourth task (Figure 3.2) provides some evidence that administrators can quickly learn to conceive of the tables in a way that allows the agent to assist them with their task.

Mixer supports two types of tasks: repeated retrieval from within a single data source, and retrieval from across more than one data source. We expected that working with more than one data source would be more difficult, but we did not see evidence for this. A comparison of the number of attempts made on Task 3, which required participants to connect two data sources, to the other three tasks that all use a single data source (Figure 3.2) does not indicate that participants found the multiple data-source task more difficult. In addition, we see nothing in the utterances during the thinkaloud to indicate that participants made any kind of distinction between these tasks. Though administrators struggled with the fact that Mixer requires copying and pasting into forms instead of typing, they did not seem to find demonstrating a link between data sources challenging. This finding is especially interesting in the light that, as indicated above, participants were not supplied with any hints to how to effect a link between multiple data sources.

The use of wrappers to augment the target page with a highlight indicating a legally selectable element provides a design advance. The highlights were intended to help users understand the limitations of the agent's communication ability and to help negotiate the problem space between human and agent. In our previous iterations of the interface, we allowed users to copy and paste from the target page directly into the workspace table, and this often lead to breakdowns. Participants seemed to have no trouble understanding how to use these highlights and never expressed any utterances or opinions that this limited their ability to use Mixer to automate their

work. We think this technique could be used in many other mixed-initiative interfaces, where users struggle to understand the scope of what the agent can do.

Mixer's interaction design specifically avoids the challenge of precise specification: the need to communicate that the user wants only a subset from within a larger set of data. Instead, the agent retrieves the larger set and encourages users to use a spreadsheet to filter down to the precise information. The fact that participants were able to correctly do so using a spreadsheet corroborates the notion that end users can conceive of the task as a nested table, and furthermore that the unnesting of the table into a spreadsheet table is an intuitive concept for administrators.

TAM produced very high ratings for Mixer for Perceived Ease of Use, Perceived Usefulness, and Behavioral Intention (to use). Scores of 6 or above on a seven-point scale give us confidence that administrators recognize the value of automating their tedious information retrieval tasks and that they would likely use Mixer in their work. Additionally, users gave a high TAM score to the quality of Mixer's output. This may be due in part to Mixer showing exactly which pages contributed to the result as it filled in the table. Several participants singled out this aspect independently for praise.

Mixer's interaction design specifically deemphasizes and to a certain extent even hides the fact that users are engaged in a programming task when they construct the first row of the table. This is a radical departure from most work in the web PBD community. We speculate that systems that similarly deemphasize the programming aspect of the task will generally be more likely to succeed with nonprogrammers. Much more work would need to be done to rigorously evaluate this speculation, though the fact that administrators with no programming experience could successfully use Mixer and their high TAM scores reflect positively, as does the fact that nonprogrammers have struggled with other PBD systems. In the same vein, more

work would need to be done to show that the administrators' resistance to programming stems from the sense that the work is perceived to fall outside the scope of what their job should be. We can suggest that this is a rich direction for further research on PBD interfaces.

In terms of PBD, Mixer embraces the notion that administrators would be disinclined to use a tool that feels like programming. The most obvious Mixer design decision in this line is that the user does not see the program as lines of code, nor as an equivalent data-flow representation of the procedure. A more subtle example is the way Mixer enforces copying and pasting as, from the user's perspective, an arbitrary constraint, rather than asking the user to understand the programming concept of using a variable as opposed to its value. Consequently, in terms of PBD systems, Mixer is near one extreme of the spectrum, ranging from those that expect the user to construct an explicit model of the program as a sequence of low-level actions, to those that do not. The success of our participants in using Mixer, as well as their recognition of its relevance and applicability to their jobs, seems to lend credence to this notion: the less end users feel like they are engaging in "programming" while using a PBD system, the less likely they seem to be to eschew the system as unrelated to their realm of responsibility.

# Chapter 4

# SmartWrap

## 4.1 Introduction

The fact that most websites lack semantic markup sharply limits the ability of screen reader software to facilitate coherent and efficient interaction with the web. In particular, many commercial screen readers provide various navigation and orientation facilities for use with sequences of records marked up using the HTML `<table/>` tag. These facilities, however, can not be used when there is no `<table/>` tag, and, without those facilities, using a screen reader to perform even relatively simple lookups in a relation page presents serious usability challenges [47]. For large portions of the web, the `<table/>` navigation facilities provided by screen readers cannot be used effectively. Empirically, the `<table/>` tag correlates poorly with occurrences of tabular data. Only a tiny percentage of instances of the `<table/>` tag correspond to datasets [13]; conversely, as we show below, many datasets do not make semantic use of the `<table/>` tag in a way screen readers can understand.

Over the last several years, Web researchers have taken many varied approaches to "filling in" the semantic information missing from webpages, including richer semantic standards for use by web authors, fully automatic

approaches to understanding webpage semantics, and supervised extraction systems. Here, we explore the possibility of asking the vast numbers of web users, most of whom lack training in programming, to supply semantic labels for web datasets. Specifically, we leverage the fact that many internet users frequently manually scrape a dataset from a webpage into a spreadsheet, for example as part of the execution of ad hoc administrative tasks [98]. Our goal is to use the information gleaned from observing users' manual scraping tasks to inject semantics into the scraped pages, thereby improving the pages for screen reader navigation.

Towards that goal, we have developed SmartWrap, an experimental interface that reconstructs dataset semantics from the actions of manually scraping a page, and we have deployed the interface to a pilot crowd of workers on Mechanical Turk. The design of SmartWrap draws from human factors research, and is specifically designed to address the difficulty most users, especially those without programming training, face with respect to abstractions [9]. The tool communicates to the user through a spreadsheet-like interface, observing the familiar, concrete action of manually scraping a dataset from a webpage into a spreadsheet. SmartWrap applies programming-by-example techniques to the scraped locations of the page, and creates a reusable program, called a *wrapper*, for the page. The wrapper can extract the structured dataset from the contents of present and future instances of the page, as well as from other pages generated from the same web template.

Our long term goal is to construct a system where contributed wrappers are used to dynamically improve the accessibility of a growing set of datasets, based on contributions from a crowd of end users. This paper lays the groundwork for that system, focusing on a study of the ability and limitations of using crowdworkers to gather semantic information about datasets on the web. We leave to future work many important aspects of the completed

system, including the questions of how best to use the semantic information to improve websites, how to apply the semantic information to additional pages from the same template, and how to encourage contribution to the set of wrappers.

We begin by providing a description of the design of the SmartWrap tool in Section 4.3. Afterwards, our exposition follows three distinct use cases we envision with respect to crowdsourcing wrappers for screen reader use. First, a screen reader user may ask a helpful colleague for *assistance* with a particular website. In Section 4.4.2 we present evidence from a user study showing that most nonprogrammers are able to successfully construct a wrapper using the tool. Second, a screen reader user could contribute *funding* towards the construction of a desired wrapper, to be paid to a crowdworker who contributes it. In Section 4.4.4 we present evidence from a larger pilot study performed on Mechanical Turk, from which we derive an estimate of the costs of getting a wrapper from a crowd of turkers. This estimate shows that a wrapper can be so obtained for a modest cost. Third, screen reader users may simply be able to request a wrapper from a crowd of *volunteers*. In Section 4.4.5 we present an indication that some web users will use the tool voluntarily and without remuneration. We close with the Discussion (Section 4.5), a discussion of Related Work (Section 4.6), and Conclusions (Section 4.7).

## 4.2 Datasets on the Web

Many webpages present one or more datasets, where each dataset is a sequence of records from a (potentially abstract) database. For example, a webpage displaying books may contain a title, author, and cover image for each book. The webpage's author chooses a visual layout intended to support the task of the typical user, usually presumed to be sighted. Many layouts

are possible and in wide use on the web. In the book example, the page might place cover, title, and author in consecutive cells (Figure 4.1a), or it might place the cover on the left with title and author stacked to the right (Figure 4.1d). A visual inspection of Figure 4.1 readily reveals that all depict the same underlying dataset. Web designers may choose between different visual layouts for aesthetic reasons. Screen readers, however, attach special semantics to the `<table/>` tag, allowing users to use navigation commands only when that tag is used.

Historically, the `<table/>` tag was introduced into HTML to present tabular data, but was also adopted by web designers as an expedient way to arrange content visually in a grid. This practive gave rise to two problems affecting screen reader users. Firstly, the `<table/>` tag is widely [13] over-used to arrange content visually. Secondly, the `<table/>` tag is under-used in the sense that a dataset is depicted using some pleasing layout, as in Figure 4.1b-e, which does not allow a screen reader user to use the table navigation commands. An understanding of how the dataset is arranged in the HTML can allow the dataset to be re-written to a more navigable `<table/>` tag for screen reader users.

A wrapper extracts the underlying dataset based on how individual elements are laid out in the HTML, essentially reversing the rendering process. A wrapper is reusable when it can be used to annotate different data presented through the same layout. For an end-user to make a wrapper, they must abstract away the layout design and see the underlying dataset. Figure 4.1 illustrates some common visual layouts. Note that while each is visually distinct, they all use the same schema of title, author, and cover image. Also, all may be constructed through various web technologies, e.g. the HTML `<table/>` tag or CSS. We make use of a rough taxonomy of visual layouts, including tables, lists, and grids, as summarized in Figure 4.1 and its caption. These layouts make consistent use of the two-dimensional


Figure 4.1: Visual layouts possible for the same data. Clockwise from top left, (a) is a simple table with records arranged vertically and fields arranged horizontally, with fields aligned vertically; (b) is a simple list, with fields arranged vertically; (c) is a simple grid, with horizontal records containing vertically arranged fields, wrapping onto subsequent lines of records; (d) is a formatted list, with formatted records arranged vertically; and (e) a formatted grid, with formatted records arranged horizontally. Variations of (d) and (e) are possible.

layout of the page, and of text formatting, to facilitate visual navigation of the dataset. A wrapper can inform a screen reader how to navigate the dataset when a `<table/>` tag was not used, but would have been appropriate semantically. Conversely, a correct wrapper can instruct a screen reader to disregard over-used `<table/>` tags which do not correspond to wrapped datasets.

Two additional complicating features occur in many web datasets. First, many pages may contain multiple datasets. For example, a webpage might present a list of winter boots based on the user's search query, and on the same page, might present a list of other recommended winter clothing that is not shoes. Second, many layouts show nested items. For example, a book

can have two or more authors. In this case the underlying dataset differs from the previous book examples in that the author has a one-to-many structure.

Understanding the underlying dataset within the visual layout is an abstract task, but it has a concrete parallel in the common internet task of manually scraping the dataset into a spreadsheet. Many users perform this task [98], for example to sort the data or to combine it with more data from another source. The actions involved in copying all data from a dataset to the corresponding cells of a spreadsheet provides a complete mapping of locations within the webpage to parts of the underlying relation. So the fact that users do, in practice, manually scrape datasets into usable spreadsheets indicates that they are able to locate the relevant data and to formulate it as a table.

A fair amount of research supports the notion that end users strongly prefer dealing with tasks presented concretely rather than abstractly. End users are known to struggle with abstraction [9]. Moreover, users decline to understand abstractions when they do not perceive them as part of their social or professional role [89]. Even experts disagree when identifying the parts of tables [57]. Schemas in particular are difficult: Jha and Nagy [63] found that users without programming training needed almost an hour of training in order to specify a schema. Any attempt to construct wrappers with end users must address these conceptual hurdles.

## 4.3 Design Process and Design

We wanted to ground our design in the actual, but unknown, distribution of visual layouts of datasets that present accessibility problems for screen reader users. We approximated this distribution by gathering a collection of web pages from workers on MTurk, who were asked to identify web pages appropriate for scraping. Specifically, we paid turkers to submit examples of

pages that contained "many similar items which you might want to copy to a spreadsheet," with the reminder that the items might be structured "as a chart or table or list." Turkers provided 375 examples, which we analyzed to identify different layouts, or patterns designers repeatedly used to convert a dataset into a design users could understand visually. We categorized the layouts of the pages from turkers according to the taxonomy presented above (see Table 4.1). Notably, Table 4.1 shows that turkers, even when primed to think of spreadsheets, provided many examples of datasets that did not use a table layout. Turkers' perception of datasets which do not use a table layout attests to the `<table/>` tag under-use problem described above.

As an additional source of datasets appropriate for scraping, we developer market ODesk[1] over 8 months. Some of these postings contained external links to websites, which we manually examined to determine if they contained scrapable websites. From this source we attained an additional 415 datasets. These datasets have a layout distribution broadly similar to the datasets from turkers, but have fewer simple layouts and tables (which can be scraped by copy-and-paste, without paying an ODesk worker) and more "difficult" scraping pages like formatted grids and lists.

We wanted to make a tool that allows end users to express the mapping between the various types of layouts and the underlying dataset, so that mapping could be used to improve the accessibility of the dataset. The variety of layouts used led us to expect that some layouts, e.g. simple tables, would be easier to map than others, such as grids. We also expected that nesting and the use of multiple datasets would be harder for users.

Our goals led us to two requirements. First, we wanted to make a tool that could be used by people not formally trained in programming, so that the task of supplying semantics for datasets is not limited to the relatively

[1] `http://www.odesk.com`; on ODesk employers may solicit (and approve or reject) bids from freelance developers for (usually small) bespoke development tasks

Table 4.1: Distribution of visual layouts in examples.

| **Display-table pattern** | **MTurk%** | **ODesk%** |
|---|---|---|
| Simple Table | 39% | 16% |
| Formatted List | 32% | 48% |
| Simple List | 18% | 4% |
| Simple Grid | 9% | 6% |
| Formatted Grid | 2% | 26% |
| **Other Features** | **MTurk%** | **ODesk%** |
| Nesting | 3% | 5% |
| Multiple Tables | 17% | 11% |

small number of programmers. Second, the interface needed to create a shared understanding between the user and the tool on what the tool could and did understand in order for the user to successfully express the layout as a dataset. These requirements led us to modify the interactions specified previously [30, 99], where some users struggled to understand which fields shared a column and less technical users seemed to struggle to complete the tasks.

We implemented SmartWrap as a Firefox extension. The tool analyzes visited pages while it is active. The tool sees the same content as the user, without awareness of whether or how the user is authenticated. The tool uses a spreadsheet metaphor for laying out the dataset. Previous research has indicated that users show great facility, and even engage in programmatic activity, with spreadsheets [79] and constructed forms [73].

When the user encounters a page they wish to wrap, they press a button in the browser chrome. The tool then opens a sidebar to the right of the target window showing a spreadsheet-like interface ready to be filled with data (right side of Figure 4.2). The left pane maintains an unaltered view of the target, allowing the user to keep an overview of both the page being wrapped and the state of the wrapper construction. In addition, the outline of the spreadsheet shows a blank first and second row for a single column,

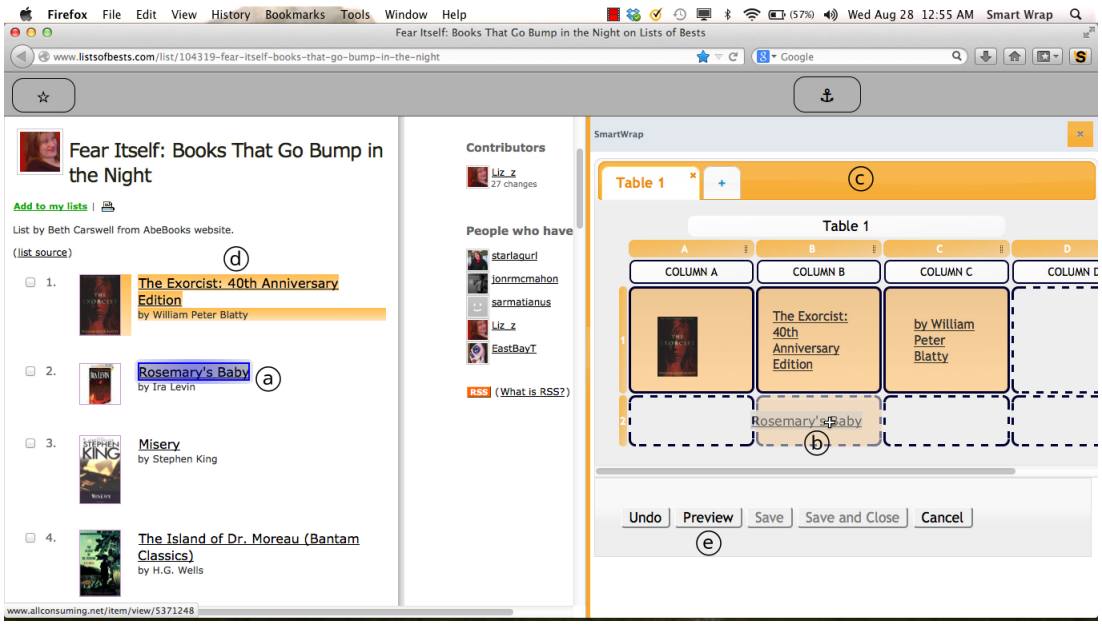


Figure 4.2: The design places a spreadsheet interface to the right of the original webpage. The user maneuvers the blue box (a) to drag the data into the appropriate box (b) of the spreadsheet, working in any order. The selected data is shown in the spreadsheet (c) and also by the highlighting introduced into the webpage (d). When the spreadsheet is complete, the user can choose to Preview (e) the collaboratively constructed wrapper.

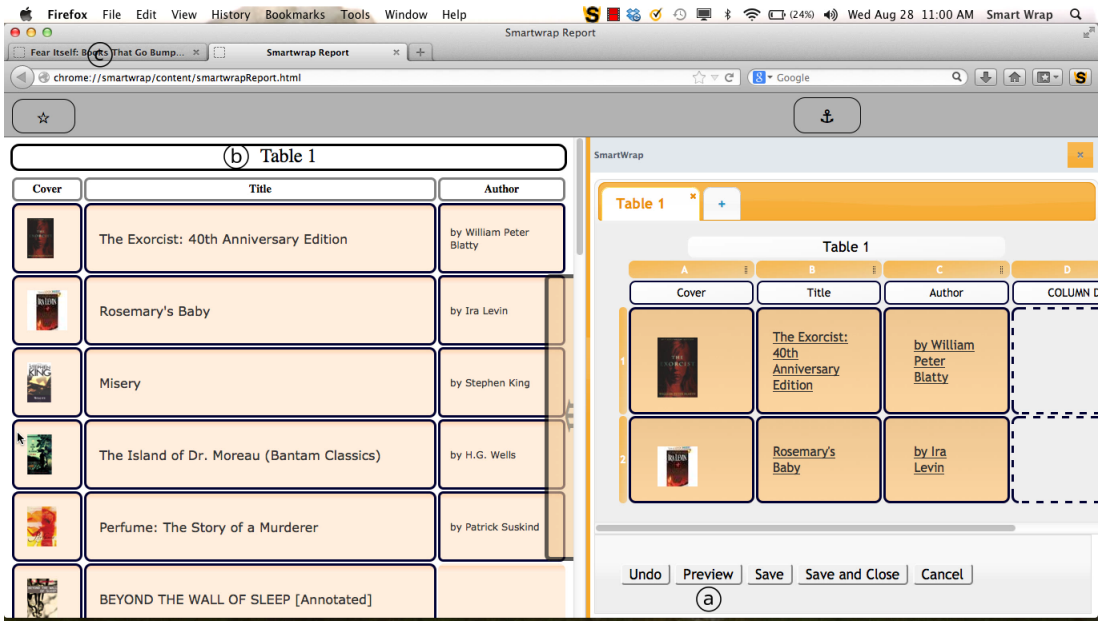


Figure 4.3: When the user presses the Preview button (a) the system responds by (i) extrapolating the wrapper for the webpage (ii) applying the wrapper to scrape the current page and (iii) presenting the output (b) of the wrapper. The user may then examine that the output matches what they expect. The original page is still open in another tab (c) and the user may review that the highlighting (see Figure 4.2(d)) covers all expected cells.

helping to communicate the user's task of constructing the first two rows to communicate the dataset to the agent.

Our system needed to communicate to the user which elements on the target page the tool could understand. Instead of allowing the user to select any text and struggling to generalize from the user's selection, SmartWrap displays a blue box that enumerates only those selections that the agent can comprehend. The tool allows the selection of any HTML element. This highlighting mechanism provides a kind of "feed-forward" mechanism precluding the user from making unproductive choices.

To select an item, the user clicks and drags the blue-boxed item over to the right-hand pane, and drops it into the appropriate cell. After the drop, SmartWrap provides two forms of feedback. First, the spreadsheet incorporates the content and provides an additional column for the user to drop new content into. Simultaneously, an orange highlight appears, letting users know which elements on the target page have been incorporated into the wrapper (Figure 4.2d).

Using the same drag-and-drop procedure, the user populates the remaining fields into the spreadsheet, including column headings and a table name, if available. Column headings and a table name are optional and can be typed or dropped from the target page. SmartWrap allows the user to work in any order they choose. The tool supports Undo operations to fix mistakes or explore alternatives.

When the user is satisfied that the two rows of the spreadsheet correspond accurately with two of the rows of the underlying dataset, they initiate the construction of the wrapper by selecting the "Preview" button (Figure 4.2e). In response, the tool sends the examples to a server, which extrapolates a wrapper for the page, as described below. The tool immediately runs the wrapper on the webpage and extracts the remaining rows of the underlying table. It then displays the extracted result (Figure 4.3b) in a new tab in

the left-hand pane of the browser window. The user can easily check that the output of the wrapper visually matches the examples in the spreadsheet. Additionally, all of the fields of the dataset are highlighted in the original webpage, so the user can switch to that tab (Figure 4.3c) and scan to make sure all the extracted fields are as expected.

If the user is satisfied that the wrapper has extracted the correct information, they can endorse the wrapper by selecting the "Save and Close" button (Figure 4.3c). In response, the tool uploads the constructed wrapper to a central server and makes it available to anyone wishing to use the wrapper. Note that the user does not see any representation of the wrapper itself, but only the output of running it on the current page. We insulate the user from the wrapper program code representation because evaluating the output of a program on a concrete example is much easier than establishing the correctness of that program.

The construction of a wrapper by the server currently uses a simple heuristic, which is not original to SmartWrap, to generalize the spreadsheet contents into reusable rules for extracting a full dataset from the page. The generalization begins with the assumption that the transformation acts in the same way on each record. The server locates the path of the HTML element containing each record and then the relative path from there to each field's contents within the record. Further records are located by reusing these paths on the rest of the document. This heuristic works for a broad collection of pages, including those used in the lab study. Based on data from the larger turk study, we estimate the proportion of dataset pages that conform to the heuristic, and present the results below.

### 4.3.1 The Heuristic

Briefly, the heuristic calculates the prefix shared by the XPaths of all extracted cells and then calculates the suffix to reach each cell from that shared prefix. What follows is a more careful exposition of the algorithm.

The heuristic uses several properties of the DOM tree.

**Definition 1.** *A (directed)* tree *with root $R$ is a (directed) graph $(N, E)$ where each $n \in N$ has a unique path from $R$ to $n$.*

The internal nodes of the DOM tree are called elements, and all text in the DOM tree is contained in leaf nodes, also called text nodes. The DOM model additionally defines leaf nodes called attributes, overloading the relational terminology; in the following, we use the term "attribute" to refer to attributes of a tuple and will use the term "attribute node" to refer to DOM attributes if the need arises.

**Definition 2.** *In a tree with root $R$ the subtree rooted at $N$* dominates *a set of nodes $A$ iff for every $a \in A$ the unique path from $a$ to the root of the tree passes through $N$.*

For each relation labeled in the page, the algorithm receives as input

**A partial schema** An ordered list of columns, each containing a unique ID and possibly a label

**The tuples** An ordered list of mappings from a subset of the column IDs to elements in the DOM; each element is labeled with an unambiguous XPath describing the path in the DOM from the root through the ordered nodes (addressed by number) of the tree to the mapped element

For each extracted relation, the algorithm returns as output

**A relation XPath** An XPath whose result set when applied to the document is a single element, used to locate the relation

**A tuple XPath** An XPath whose result set when applied to the document is a list of elements, each used to locate a tuple of the relation

**Attribute XPaths** A mapping from the column IDs to XPath expressions; each expression is used to locate an attribute within the tuple subtree

To perform the extraction, the algorithm proceeds by locating the relation node, then locating tuple nodes which are contained in the relation node, and then locating attribute nodes which are contained within a tuple node.

To compute the XPaths, the algorithm relies on repeated application of a function `LongestXPathPrefix`, which takes as input a set of XPaths and returns the prefix of the longest XPath which (1) is also a prefix of the remaining XPaths, (2) ends in the character '/' (this is the path-step separator for XPath expressions) and (3) is the longest prefix meeting these criteria.

The table prefix is computed as simply the `LongestXPathPrefix` of the set of all XPaths from all labeled attributes. The tuple XPath is the table prefix concatenated with the first element occurring after the prefix (if it weren't the table prefix could have been longer). The attribute selector for each column is just the suffix of the XPath for the attribute in that column, which occurs after the tuple XPath. Currently, the algorithm simply takes the first suffix; an exception is thrown and logged if the second row disagrees, but no attempt is made to automatically reconcile the discrepancy.

### 4.3.2 Method

Our central goal is to engage with a crowd that can and will supply semantics for a large number of datasets with usability problems. To evaluate our progress towards this goal, we evaluate a number of specific hypotheses. First, we want to check that most users will be able to supply semantics using the SmartWrap tool, not just programmers.

**H1** Programmers and non-programmers will be able to deconstruct a layout into a spreadsheet detailing the dataset.

Previous research predicts a positive answer to H1: end users routinely interact successfully with spreadsheets [79] and with datasets on the web [98], and previous systems have evaluated the performance of small groups of users with web datasets [30, 99]. We evaluate H1 for a larger group.

Second, we want to check that the process of supplying semantics is not overly taxing for the user, and thus too expensive to be practical.

**H2** Programmers and non-programmers can create wrappers with the tool in modest time and will be willing to do so for modest cost.

H2 has guided our design, and we provide cost estimates from a deployment on Mechanical Turk. Additionally, we informally compare the costs of using the crowd versus employing programmers for wrapper construction. A central part of H1 and H2 is that nonprogrammers will be able to participate in wrapper construction; this does not mean, however, that we expect the performance of programmers and nonprogrammers to be identical. In fact, we expect programmers to have greater facility creating wrappers, especially for tasks requiring more abstractions [9] such as those exhibiting nesting or multiple datasets.

Finally, we want to study whether it is feasible to expect users to contribute wrappers without being paid. We present some tentative evidence that users can and will do so.

**H3** Crowdworkers will be willing to contribute wrappers for webpages without monetary remuneration.

The Social Accessibility project [90] found that a significant number of sighted users were willing to contribute improvements for websites when those improvements aided the accessibility of the pages, suggesting a positive answer for H3.

## 4.4 Evaluation

### 4.4.1 Lab Study

In order to acquire fine-grained information towards evaluating H1, we performed a lab study with participants recruited locally. For the lab study tasks, we manually selected a subset of eleven webpages to wrap containing a broad selection of the layouts. We expected that simple tables would be easy to transfer into spreadsheets, so we selected several as introductory tasks (tasks 1-4), allowing users to gain confidence in the use of the tool. Although tasks 1-4 were all simple tables, they differed in that tasks 3 and 4 contained significantly more columns. Tasks 5-7 were lists. Task 8 was a grid. Task 9 was a formatted table, but with the added complication that the page contained multiple tables. Tasks 10 and 11 were simple tables that exhibited nesting.

We recruited 30 participants to come to our lab. Participants were recruited from a website that maintains contact with local people interested in participating in research studies. The website requires participants to be over 18 for reasons of consent, and this was our only exclusion criterion. All participants were sighted but were not asked further about their vision. Each participant took a brief survey (based on an instrument measuring technical skill developed by the European Union [32]) about their technical background, including exposure to programming and/or programming instruction. Each participant was given a brief introduction to the tool, and then asked to watch a series of short training videos describing use of the tool. We asked participants to proceed through the tasks 1 through 11 sequentially, allowing them to review the videos.

While they performed the tasks, we asked participants to ”think aloud” in order to capture both their understanding of the task and their sense of

how the tool worked. We recorded all screen actions and narratives, and coded the recordings to analyze participants' performance on the tasks. We asked participants to strive for completeness and accuracy in the extracted spreadsheets, but told them that they could skip a task if they felt frustrated after several attempts. We never asked them to complete the tasks as quickly as possible.

### 4.4.2 Lab Findings

The 30 participants exhibited a fairly diverse array of technical backgrounds. All reported basic computer skills, including opening programs and using copy and paste. Twenty-five (25) of the participants (83%) reported using a formula in a spreadsheet. Twelve (40%) reported having written a program. We asked about experience with a variety of technical phenomena, but a Principal Components Analysis found that the self-report of having written a program was highly correlated ($\rho \approx 0.89$) with the first principal component, which accounts for about 30% of the total variation. In what follows, we concentrate on comparing programmers and nonprogrammers, based on survey responses. To reiterate, we use a very low bar for distinguishing programmers from nonprogrammers, where it suffices to have written – ever – a single program. Of the 12 programmers, 7 reported having used advanced techniques, like recursion.

Over the course of the study, participants spent 1106 minutes attempting 252 tasks. Of the attempted tasks, 222 tasks (88%) were completed correctly, in the sense that participants successfully placed two rows of data into the spreadsheet interface. The other 30 were abandoned. The mean time spent on a wrapper was 4:24 and the median was 3:10; these measurements include the time spent on a task before abandonment. 72% of the tasks were completed in less than 5 minutes, and 94% in less than 10 minutes.

| Task | Succ. | time | attempts | fields dragged / extracted |
|---|---|---|---|---|
| T1: simple | 90% | 3:43 ± 3:45 | 1.2 ± 0.56 | 4 / 134 |
| T2: simple | 92% | 4:18 ± 4:44 | 1.3 ± 0.48 | 6 / 150 |
| T3: simple | 92% | 3:23 ± 1:13 | 1.0 ± 0 | 18 / 477 |
| T4: simple | 60% | 6:05 ± 3:55 | 1.4 ± 0.79 | 18 / 594 |
| T5: list | 90% | 2:37 ± 1:49 | 1.5 ± 1.3 | 8 / 100 |
| T6: list | 94% | 4:26 ± 2:25 | 2.0 ± 1.3 | 14 / 105 |
| T7: list | 94% | 3:16 ± 2:04 | 1.7 ± 1.5 | 6 / 324 |
| T8: grid | 87% | 4:12 ± 3:44 | 1.9 ± 1.7 | 4 / 38 |
| T9: multi | 80% | 5:29 ± 4:17 | 1.8 ± 1.2 | 12 / 102 |
| T10: nested | 73% | 5:39 ± 2:37 | 2.0 ± 1.0 | 14 / 1050 |
| T11: nested | 43% | 2:51 ± 1:30 | 1.3 ± 0.8 | 6 / 741 |

Table 4.2: Summary of participant completion of the individual tasks. The final column shows how many fields the participant had to drag for the most complete spreadsheet and how many total fields were extracted into that spreadsheet.

Participant successfully completed an average of 8 tasks, with a standard deviation of 3 tasks. Two participants, both nonprogrammers who reported having used a formula in a spreadsheet, spent 35 minutes on 4 and 3 tasks respectively without completing any tasks, and are removed from the subsequent quantitative analyses as outliers. Programmers were more successful overall than nonprogrammers: 97% of attempted tasks were successful versus 83% of tasks attempted by nonprogrammers. The increased success rate of programmers was particularly marked in the tasks involving multiple datasets and nesting. The baseline improvement for being a programmer was 14% (97% - 83%), and for Tasks 9, 10, and 11 it was 31% (78% - 57%). An independent samples t-test rejects the null hypothesis that the difference in success rate is zero: $t(83) = 2.08, p = 0.04$. This is unsurprising, as we expect programmers to cope better with the abstractions involved in the later tasks.

The design of the study allowed participants to abandon a task at any

| Outcome | Hazard Ratio (HR) | log HR | SE |
|---|---|---|---|
| **success** | 1.63 ($p < 0.01$) | 0.49 | 0.17 |
| **abandonment** | 0.21 ($p < 0.01$) | $-1.56$ | 0.55 |

Table 4.3: coefficients of the competing risks analysis, showing the ratio of hazard for exposure to programming

time. To control for individual and systematic persistence, we model the study as a proportional hazards competing risk model [96], where after each attempt the task is "at risk" of either ending in success, or being abandoned by the participant. The independent variables in the model are whether or not the participant is a programmer, and which task is being attempted; a random effect is introduced for each participant to control for individual variability.

Table 4.3 shows the results of the competing risks analysis. The second column shows the multiplier for the "hazard function" of terminating the task by success, or by abandonment. Programmers do indeed have both a reduced rate of abandonment (more persistent) and an increased rate of success (more skilled) at tasks. Specifically, a programmer has about a 79% reduction ($1 - 79\% = 0.21$) in the hazard of abandoning a task, and a 63% greater ($1 + 63\% = 1.63$) chance of succeeding on a given task attempt, both statistically significant at the $p < 0.01$ level. The model also estimates the base hazard rate, estimating that about 50% of tasks attempted by nonprogrammers succeed on the first attempt, versus 70% of tasks attempted by programmers.

The think aloud transcripts revealed some conceptual difficulties encountered by the participants. Many were initially confused that the target page continued to act like a normal webpage, navigating to a new page when a link was clicked, or, in Task 4, sorting a table when a table header was clicked. Most participants eventually learned this behavior. Several participants vocalized doubt whether some web elements, like images and links, belonged in the spreadsheet. Four participants expressed pleasant surprise when they

were able to place images in the spreadsheet. Many participants expressed frustration that SmartWrap's blue box only allowed them to select whole HTML elements[2], and not the cell contents they wanted.

Many participants became frustrated with the tool at some point while completing the tasks. One participant complained about the blue selector "boxes becoming big and small," finding them "confusing and irritating." Eleven (11) participants expressed frustration when they felt they had correctly communicated the dataset to SmartWrap but the tool was not able to extract the table. Two participants were especially annoyed that the selection did not allow construction of the precise dataset they wanted; one commented "it was really infuriating that I couldn't select certain pieces of text." Another participant commented "it was fun at first, but then it got frustrating." Conversely, five participants expressed relief when the tool successfully extracted a table. Ten participants volunteered praise for the tool in the think aloud. Two participants stated that they would envision using the tool in their daily lives.

### 4.4.3 Mechanical Turk Study

To evaluate whether crowdworkers would contribute wrappers for modest cost, we ran a larger study on Mechanical Turk. In addition to the 386 webpages gathered from turkers, we also asked the crowd to wrap the the 415 webpages posted on ODesk.

We invited workers on Amazon's Mechanical Turk to extract datasets from the gathered websites into the spreadsheet interface. These participants had no training, but did have access to the training videos. For each task, the worker was presented with a link to a website, chosen randomly

[2]An example can be seen in Figure 4.2, where the user has selected the text "by William Peter Blatty" but would have preferred the text "William Peter Blatty;" there is no element boundary between the word "by" and the name

from the collection of sites, and offered $0.10 to extract the dataset. Workers were permitted to label websites as non-datasets, and to report errors encountered. The task description promised workers a bonus for datasets correctly extracted. The amount of the bonus was not specified, but we paid $0.30. Workers who completed at least one task were offered payment to complete the same survey as the lab participants regarding their technical background. The spreadsheets submitted by workers were evaluated on a three-point scale with values 0 for wholly *incorrect* or empty spreadsheets, 1 for *plausible* spreadsheets with incorrect cells or merged columns, and 2 for *high-quality* spreadsheets, which well matched the underlying dataset of the linked webpage.

### 4.4.4 Mechanical Turk Findings

In the MTurk study, 241 workers completed 4133 tasks, approximately three quarters of which (2811) contained extracted datasets. About 58% of the surveyed workers reported having written a program. Based on MTurk's report of times when workers accepted and submitted a task (though of course they could be multitasking during that interval), workers spent about 15466 minutes working on the extracted datasets. The average time per dataset was about 5 minutes, and 87% of datasets were submitted less than 10 minutes after being accepted. A small number of workers contributed a large amount of the work; about 70% of the tasks were completed by the 15 workers who each completed at least 50 tasks.

Turkers were less likely to indicate a page using a table layout as a non-dataset ($t(1307)=-8.68$, $p<0.001$), and more likely to mark lists as non-datasets ($t(1434)=9.90$, $p<0.001$).

We judge the submitted spreadsheets as incorrect 17% of the time, plausible 38% of the time, and high-quality 45% of the time. Unlike in the lab

| | dataset | quality | # needed | cost |
|---|---|---|---|---|
| **table** | 89% ± 2% | 74% ± 5% | 1.5 ± 0.1 | $0.45 ± $0.15 |
| **list** | 81% ± 2% | 28% ± 5% | 4.5 ± 0.8 | $1.00 ± $0.40 |
| **grid** | 86% ± 2% | 11% ± 6% | 13 ± 7 | $3.05 ± $1.85 |

Table 4.4: Estimates of the costs to have a 95% chance of receiving at least one high quality response. The first column lists the percentage of turk responses that saw a dataset in the page, bracketed by the margin of error. The second column estimates the probability that a worker's response will be high-quality, conditioned on being treated as a dataset. The third column estimates the number of assignments needed to encounter at least one high-quality response. The fourth column estimates the cost of the expected number of responses, assuming workers are paid with the policy employed in the study.

study, there was no significant difference between the success rates of turkers who reported programming experience and those who did not.

Simple tables were more likely to be transcribed with high quality ($t(452)=14.32$, $p<0.001$) than non-tables, affirming the decreased cognitive burden of placing a table in a spreadsheet. Lists ($t(521)=-5.58$, $p<0.001$) and grids ($t(117)=-8.66$, $p<0.001$) were both significantly less likely to be transcribed with high quality.

These observations allow us to make some estimates of the costs of eliciting high quality wrappers from MTurk workers, as shown in Table 3. These cost estimates assume the same payment structure as in the study, and could be decreased in several ways, e.g. by paying less (or not at all) for merely plausible wrappers. As shown in Table 3, for pages with a table layout, people would need to make 2 wrapper requests to be 95% sure of receiving a high quality wrapper (∼$0.45). With a list layout, people would need about 6 requests (∼$1.00). A grid would require about 20 requests (∼$3.05). A wrapper for a list is an order of magnitude more expensive than a wrapper for a table, and a grid yet another order of magnitude more expensive.

Datasets exhibiting nesting were less likely to be successfully wrapped. Nesting seems to add about an order of magnitude to the number of requests required. Since the number of pages using grid layouts and nesting is small, both estimates exhibit a fairly large variance.

Table 4.4 provides an estimate of how many attempts to wrap a page the crowd would need to make to have a high probability of success, assuming the page is a dataset wrappable by the tool. By counting all pages as failures if they did not result in a success, even after being attempted in the numbers needed, we can roughly estimate the proportion of datasets which can be wrapped using the SmartWrap tool. The figures in Table 4.4 yield an estimate of $62\% \pm 16\%$ over the distribution of layouts present in the collection of layouts provided by turkers, and an estimate of $58\% \pm 25\%$ over the distribution of layouts extracted from ODesk.

For a rough point of comparison for the costs entailed by Figure 4.4, we posted job advertisements on ODesk for programmers to construct a wrapper for a simple table. The programmers who successfully completed our task charged an hourly rate of $8.89 and billed an average completion time of 70 minutes, yielding an estimate of about $9.76.

### 4.4.5 Voluntary Usage

In order to complete the Mechanical Turk tasks, turkers were given access to the tool, and some continued to use the tool for other purposes, for which they received no payment. We observed 427 such voluntary usages of the tool, i.e. for every ten paid usages we saw about one voluntary usage. This set of voluntary usages visited a total of 194 different web domains. Of this set of visited web domains, 5 web domains were visited with more than 10 distinct wrapped pages per web domain, and 143 web domains were visited exactly once. The web domains with more than 10 distinct wrapped pages

account for about 30% of the voluntary usages. The voluntary usages represented a large variety of tasks, including scraping account information from Mechanical Turk itself, scraping open access journal citations, and scraping dozens of children's birthday party providers in Canada. Given access to the tool, turkers chose to exploit the tool to solve their own scraping problems.

## 4.5 Discussion

The overall success rate in Table 4.2 lends support for hypothesis H1, suggesting that most people will be able to use a tool such as SmartWrap to make an effective wrapper. Our lab study indicates people with programming experience would experience a higher rate of success, partially due to increased persistence and ability to abstract; however, this did not seem to be the case with crowdworkers, who may be more motivated to complete simple tasks than more complex tasks. Alternately, turkers unable to complete SmartWrap tasks, regardless of technical ability, may have self-selected out of the sample.

The overall short amounts of time spent on each task lend support for H2. Extrapolating from the data suggests that about 87% of web pages containing datasets could be wrapped by an end-user in under 10 minutes. The participation rates in the MTurk study provide evidence that crowdworkers will be willing to provide requested wrappers for reasonable costs. Although the labor costs of crowdsourcing a wrapper are nontrivial, in the worst case rising to five dollars for a single page, we reiterate that wrappers are shared between users. So costs could be shared among users of the wrapper, and potentially also with a funding agency. Finally, the funding use case (see Section 4.1) provides a potentially potent method for matching crowd efforts to the pages screen reader users actually want to use, since someone offering to pay for a wrapper for a page provides a valuable demand price signal. We

leave this line of reasoning to future work.

We did not expect turkers to voluntarily perform scraping tasks in the numbers that we observed, as we had not invited or solicited anyone to use the tool outside of the paid tasks. We surmise they conceived a task from outside the assigned task, possibly to leverage the tool towards accomplishing another MTurk task, and attempted to use the tool for that task. We submit this voluntary engagement with the tool as tentative evidence for H3, specifically that a small crowd of end users will contribute wrappers to SmartWrap without payment. In the study, users did so apparently because they value its assistance with the mundane manual scraping task, but we speculate that they may also be willing to do so to improve accessibility of scraped webpages [90].

We close the discussion by reflecting on some open design issues uncovered in the lab study. A design choice that caused participants pain was the use of the blue box to delineate the selectable elements of the page. The blue box technique is clearly less expressive than allowing users to select an arbitrary range in the DOM, but is much easier to use. On the one hand, participants found it to be a usable means of selecting elements without grappling with the DOM structure of the page. On the other hand, several participants did remark on having to settle for a different dataset than the one they wanted because the text they wanted was not selectable. The blue box interface presented here seems to meet well the needs of new or casual users, being easy to pick up and precluding selections that are not interpretable by the tool. In future versions of the tool, advanced or experienced users might appreciate an additional mechanism to select richer parts of the page. Users' struggle to disambiguate an element from its ancestor elements indicate that the selection behavior should try to assist the user more.

Several lab participants did not think the tool could be used for nested structures. One possible way to inform users that they can build spread-

sheets with nesting is simply to add such structures to the training videos. This approach could also clarify that images and links can be placed in the spreadsheet. But many users will not use or attend to the documentation. Another way to encourage nesting behavior would be to have the tool assist with it. For example, when detecting that the same element has been dragged into two consecutive rows, the tool could ask the user whether the two cells should be merged, making it clear that the cell nests the other columns of the spreadsheet. We intend to return to this issue in future work, with more information about how often the issue occurs in the distribution of pages screen reader users actually wish to access.

## 4.6 Related Work

A table navigation mode was introduced by Oogane and Asakawa in 1998 [82], allowing screen readers to move their cursor about within the two-dimensional structure of an HTML `<table/>` structure. As a testament to its utility, the table navigation mode was subsequently adopted in nearly every commercial screen reader. Without its availability, tabular data is extremely difficult to navigate and understand through a screen reader. In the tasks studied by Gunderson and Mendelson [47], presented to screen reader users and to a control group of sighted users, the task involving lookups in tabular data (without table navigation mode, which had not yet been introduced) produced the most difficulty for screen reader users. Clearly, if the screen reader cannot reproduce the two-dimensional structure, then its user cannot make use of table navigation mode to move about within that structure. By restoring the relational semantics to the dataset, we hope to extend to screen reader users the manifest benefits of table navigation mode for datasets not presented with a clean `<table/>` markup.

Many research projects have pursued a transcoding approach where web-

sites presenting accessibility challenges are re-written in a more usable form. The Accessibility Commons [65] proposes to serve all of the transcoding metadata packages to screen reader clients from a unified architecture; the paper also presents useful pointers to the various projects they propose to unify. AccessMonkey [7] extends the GreaseMonkey[3] tool, which allows arbitrary re-writes of web pages written in the JavaScript programming language, and allows users to apply a collection of scripts improving the accessibility of various pages. SmartWrap can be viewed as an IDE for developing one specific, but very common, type of AccessMonkey script. AccessMonkey scripts are typically constructed by JavaScript programmers, whereas SmartWrap is designed for use by nonprogrammers. The Social Accessibility project [90] used a crowd to fix accessibility problems on websites, and reported large numbers of users were willing to contribute their work to the project.

Several projects mine large amounts of relational data from the web for programmatic use, but usually divorce the machine-readable relation from the source webpage, making it difficult to use it to improve the webpage. Freebase [39] publishes facts about a large number of entities, culled from a large number of web sources. Freebase facts are gathered by contributors or by bots written by contributors. Website authors can also choose to publish their own data using machine-readable markup such as microdata [52] or RDFa [51], and are increasingly doing so [76], although screen readers do not at present generally make use of the markup. Embedding machine-readable markup generally requires a fair amount of technical knowledge and commitment. With even more technical knowledge, Computer science researchers develop systems that understand (and can annotate) large numbers of websites, e.g. web tables [13, 1] or lists [31], leveraging the vast size of the web to discover many examples automatically. Our work focuses on gathering relational data from casual and/or nontechnical users.

[3] `http://wiki.greasespot.net/Greasemonkey`

Wrapper Induction is the problem of constructing a reusable wrapper based on example labels provided by an annotator. Many researchers [36, 38] have studied the induction of wrappers for a distribution of pages with the assumption that several representatives have been perfectly labeled, presumably by trained annotators. Given a sufficient number of high-quality examples, modern systems [46, 26] are able to generate and maintain wrappers for a large number of pages. But these systems require a large number of expensive annotation effort to function, effort which the systems will naturally devote to more strategically important pages. Our work seeks to generate data for these systems more cheaply, and more diversely, by empowering everyday end users to express wrappers for currently unwrapped datasets. There is a potential synergy between work in Wrapper Induction and work like ours in Accessibility, where volunteers to improve accessibility can supply data to induction systems whose improved models can further improve the accessibility of other pages.

A number of systems have been advanced to enable end users to program wrappers. Lixto [5] is an intelligent assistant for a database administrator constructing XML from an HTML web page. Thresher [54] assists the user with constructing RDF by example, using context menus and wizards. These early systems were largely not tested with nontechnical end users. In Dontcheva et al's [30] summaries framework, a user who wants to gather a set of records, each corresponding to a single web page, into a coherent whole, locates and labels the elements corresponding to the parts of a single record. The system responds by coloring the fields, and then is able to extract additional records from pages of the same type. Toomim et al's [99] reform system offers users a selection of potential data visualizations, e.g. a map, a timeline, etc, each with a fixed schema. The user then provides positive and negative examples of the fields of the records, and the system interactively updates the current predicted fields. The user evaluations showed that most

users understood the concepts involved, but that less technical users struggled with some aspects of the tools, e.g. the use of colors to indicate that data were in the same column. Our work refines this line of work, validates it on a larger set of users, and examines it in the context of wrapper construction for accessibility.

A few projects showcase end users' facility with placing data into useful spreadsheets. The CrowdDB [35] project extends SQL to allow some attributes or tuples to come from crowdworker responses, presented as a fillable form. CrowdFill [83] and AskSheet [86] use a spreadsheet interface to allow a requester to specify constraints over a set of fields for the crowd to fill, from which they produce data acquisition strategies that cost less than simply filling all blank cells. Our work overlaps but differs in that we allow users to specify the schema, we acquire a whole page's content as one relation, and we require that users provide the relation together with the webpage from whence it came.

A good amount of work attests to the ability of nontechnical end users and crowdworkers to accomplish data tasks. Data Wrangler [64, 48] offers users a Programming by Demonstration interface to cleaning data in the cells of a spreadsheet. Several systems [107, 34] ask crowdworkers to supply or refine typing information about columns for schema matching, providing mechanisms for limiting the number of tasks required. CrowdFind [28] asks crowdworkers to identify tuples characterized by a given predicate. MIXER [42] allows end users to demonstrate how online datasets can be joined together, then replays the user actions to construct a complete table. Potluck [58] and Vegemite [71] provide drag-and-drop visual metaphors for combining web data sources. These systems also make use of wrapped datasets, indicating another potential use for accessibility-improving wrappers.

## 4.7 Conclusions and Future Work

We have presented the design of SmartWrap, a tool enabling end users to construct a reusable wrapper simply by demonstrating some of the actions of a standard manual scraping task into a spreadsheet. The studies demonstrated that end users can successfully make use of the tool for a broad array of common layouts in use on the web, explicitly establishing that people without programming training can construct many types of wrappers. The cost estimates suggest that wrappers can be acquired from the crowd for modest cost. Finally, the voluntary usage of the tool suggests that users may contribute wrappers without remuneration, motivated either by a desire to improve the accessibility of the web or by a desire to save labor scraping the page.

The data demonstrate that a relatively small change (e.g. extracting data from a table as opposed to a grid) changes the difficulty of the task for end users. This increases the number of crowdworkers required to accomplish the task, and hence the cost, by an order of magnitude. Not surprisingly, crowdworkers produce lower quality results compared to laboratory participants. However, to our surprise, crowdworkers who reported higher technical skill did not construct significantly more high-quality wrappers than others.

The present work demonstrates the feasibility of employing crowd labor to produce reusable wrappers for web datasets, but leaves to future work the challenging issues of delivering those wrappers to screen reader users. We are currently in the process of prototyping a complementary tool that uses the contributed wrapped to improve the accessibility of web datasets. The prototype attempts to address the issue of how non visual web users can express that a particular dataset lacks the proper semantics. Encouraged by the present work, we hope that some prototype users who report problem datasets will be able to get more manageable table structures from the crowd.

We hope work on the prototype will indicate a path towards the "virtuous cycle," where reports of inaccessible datasets actually get addressed, leading to a gradual expansion of the set of wrapped and consequently more accessible datasets. Additionally, by engaging with screen reader users, the prototype will allow more direct study of the distribution of web datasets presented visually, which in the present work has been only roughly approximated by asking turkers and by observing postings on ODesk. Lastly, we intend to use the prototype to continue study of hypothesis H3, specifically whether a crowd of users will lend their eyes to provide wrappers for webpages for the purposes of improving accessibility of web datasets.

## 4.8 Acknowledgments

This research and development has been funded by grants (H133E080019 and H133A130057) from the United States Department of Education through the National Institute on Disability and Rehabilitation Research and a grant (DTRT12-G-UTC11) from the US Department of Transportation. We also thank our anonymous reviewers for helping us improve this paper.

# Chapter 5

# enTable

## 5.1 Introduction

Most data-driven web pages communicate semantic information visually. Web sites employ complex CSS and javascript in order to provide aesthetically compelling pages that are easy to parse visually. In order to communicate these pages to blind users, screen readers face the daunting task of reverse-engineering the web designer's intention and rendering it as a stream of spoken text. In general, understanding web pages is beyond the capability of computers, and so screen readers address some important special cases where the intended semantics are roughly aligned with the structure. For example, screen readers provide sophisticated mechanisms for navigating and understanding web data that makes straightforward use of the `<table/>` tag. Many web pages that present data, however, do not use the simple `<table/>` layout [4].

Figure 5.1 shows the first row of data relating to a set of products, laid out as a grid of boxes. The grid layout makes it easy for sighted users to visually compare the products on the page. When accessed with a screen reader, however, the grid layout is disorienting and difficult to navigate. For

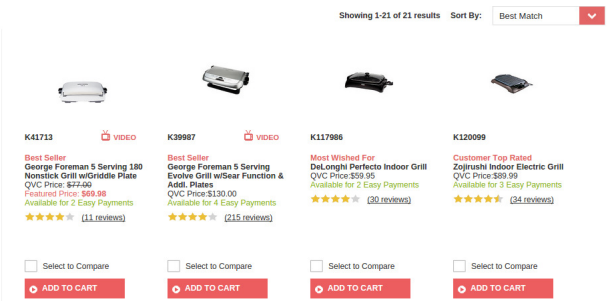


Figure 5.1: A web page presenting a data set as presented by the web author, using a grid layout

example, there is no way to scan across the prices of the different products. Figure 5.2 presents the same information using a `<table/>` layout. The layout is a straightforward use of the `<table/>` tag in the sense that each row contains information about one product and each column contains exactly one attribute about that product. The table layout is substantially easier to access and understand using a screen reader. Users can move the cursor quickly and logically around the table. For example, they can scan down the column of prices to quickly compare several products or to screen out items outside their price range. Screen reader users can move within the logical structure of the table, skipping over the intervening content without devoting attention to it. EnTable effects this transformation by rewriting complex template layouts as simple tables. This transformation cannot be accomplished automatically by current technology [4], so EnTable employs the eyes of sighted users.

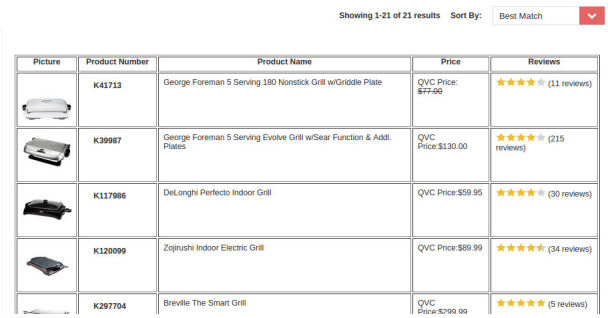


Figure 5.2: The same data set as in Figure 5.1 using a straightforward `<table/>` tag layout, which is easier to understand and navigate using a screen reader

## 5.2 Architecture

Sighted users are able to take advantage of the semantic information designers implicitly build in the visual layout of pages. Unfortunately, neither computers nor blind users can easily extract the meaning of these layouts. Since algorithmic extraction of data sets from HTML is beyond the current state of the art [4], we turn to the processing power of people. In common with crowdsourcing projects like Social Accessibility refsocialAccessibility, EnTable employs a crowd of sighted users to re-write web pages to be more useful for screen reader users. Figure 5.3 depicts the workflow of the EnTable system.

Figure 5.3 illustrates the three main components of this socio-technical system. EnTable users make requests for specific pages to be added and view rewritten web pages using a proxied environment provided by a browser extension. EnTable stores page descriptions in a central server-side repository. Sighted crowdworkers contribute table descriptions using the SmartWrap browser extension. Our previous work [41] describes the SmartWrap extension and demonstrates that a large population of crowdworkers can effectively and inexpensively provide table descriptions of pages that use com-

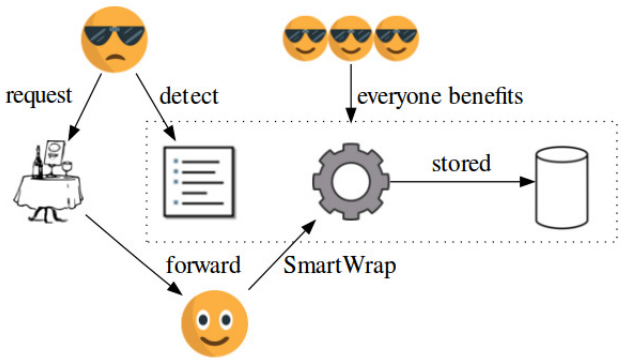


Figure 5.3: When an EnTable user detects a page with a confusing template, they submit a request to EnTable. EnTable forwards the request to a sighted crowdworker, who uses the SmartWrap interface (See Figure 5.4) to mark up the template as a table. The table description is stored in a repository, allowing all subsequent visitors to access this page as a table

plex templates for layouts. One major challenge in the SmartWrap work is that table descriptions are small programs, equivalent to nested sets of XPath or CSS selectors, and most crowdworkers are not programmers. The SmartWrap interface, shown in Figure 5.4 works with nonprogrammers, by leveraging users' familiarity with spreadsheets. SmartWrap extrapolates the required selectors based on example elements provided by the user as the contents of the first two rows of a spreadsheet.

The reported version of the EnTable extension supplies two novel functions. First, it allows screen reader users to submit requests for specific pages. Sighted crowdworkers then use SmartWrap to generate table descriptions of these pages [2]. Second, upon subsequent visits to the requested page, EnTable users can access the page as a table.

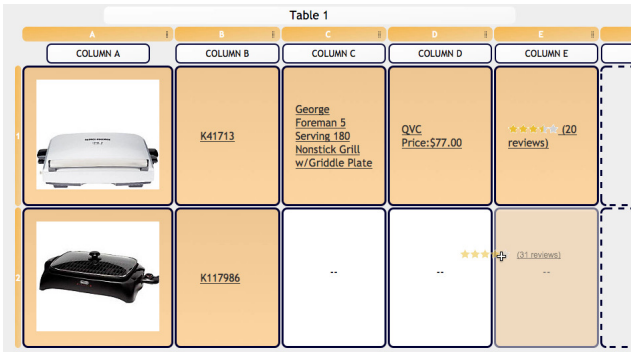


Figure 5.4: The SmartWrap interface. Sighted users drag-and-drop the information from the first two rows of the data set into a table. Based on the two example rows, SmartWrap extrapolates the pattern and presents the complete table for the user's approval. The extrapolated patterns are stored in the repository, so they can be used on different pages from the same website.

## 5.3 Conclusions and Future Work

EnTable provides an end-to-end system allowing screen reader users to report problematic data sets and view data sets that have been improved by informants. For small demonstration purposes, the demonstrator can supply semantic information for all reported pages. In a full-scale deployment, however, it is obviously infeasible for a small team to handle all requests. Previous work suggests [41, 91] that a substantial number of sighted users are willing to provide voluntary labor to benefit blind users. Whether sufficient numbers of screen reader problem reporters and sighted informants will participate to sustain the approach is an empirical question that the system allows us to study.

The implementation of the EnTable system also enables the study of several additional issues which we intend to address in future work. First is the question of how much benefit the rewriting of data sets in the more semantic form of a `<table/>` offers to screen reader users. But the difficult

part of the EnTable system is the acquisition of semantic information from sighted informants, after which the rewriting with a `<table/>` tag is just a matter of formatting the information in relational form. Screen reader users may benefit more from some other structure (or from the ability to switch among several different views of the data set), and the implementation of the present system provides a platform for comparing various alternatives.

## 5.4 Acknowledgements

This research and development has been funded by grants (H133E080019 and H133A130057) from the United States Department of Education through the National Institute on Disability and Rehabilitation Research and a grant (DTRT12-G-UTC11) from the US Department of Transportation. We also thank Jeff Bigham for his helpful comments.

# Chapter 6

# Wrappers as an Assistive Technology

## 6.1 Wrappers

In this thesis, we use the term wrapper to refer to a small piece of code that maps the contents of a web page into a single nested relation [69]. Essentially a wrapper reverse engineers the code that renders a result set from a database into a web page.

A wrapper allows a computer system such as Mixer to understand the low-level relation encoded in the web page. Mixer allows end users to combine and manipulate these low-level relations towards the accomplishment of a higher-level task, such as looking up information about a homogeneous set of items. SmartWrap complements Mixer by providing end users the capability to construct low-level wrappers for individual pages. SmartWrap also facilitates accomplishment of the higher-level task of scraping an entire relation at a time into a spreadsheet.

While end users do perceive the value of systems like Mixer and SmartWrap in the abstract, they struggle to apply them to novel concrete tasks.

One reason for the struggle is that sighted people perceive the low-level task of simply understanding the web page as trivial. Because it is so easy to understand the relation in the web page, users perceive explaining that relation to a system as a waste of time.

Blind users of the web, in contrast, do struggle to understand the content of web pages. Estimates from the literature estimate that blind people take about 12 to 15 times as long to accomplish tasks on the web as sighted people [95, 84] (see also Section 2.7.1). A large part of this discrepancy is that they are operating non visually in a visual environment: web pages are created for sighted users by sighted developers, who are often unaware of how their design choices impact non visual access. Wrappers have potential to improve the situation by assisting blind users with a large number of low-level tasks, i.e. understanding the relation encoded in individual pages. The next section describes the potential assistance provided by wrapped pages in greater detail.

A similar situation obtains in the physical environment. Blind people have no problem with planning a high-level task, like going to the bank, but have profound difficulty executing low-level parts of the task. For example, they cannot see a tripping hazard in their path or a closed sidewalk. Blind people employ a variety of assistive technologies to aid them with these low-level aspects of navigation in physical space, such as a cane or a trained guide dog. When a blind person uses, for instance, a guide dog, the person remains in control of understanding and proceeding towards the high-level task. The person knows the way to the bank; the dog does not. The dog only has responsibility to detect and communicate cues about the physical environment, such as curbs, steps, or hazards.

A screen reader is an assistive technology that serializes web pages as spoken text. Blind people who use screen readers do not expect the screen reader to understand the high-level task they are trying to accomplish. The

blind person remains in charge of the high-level task, and the screen reader's responsibility is to provide informative cues about the information contained on the web page.

But the screen reader can only provide cues that it is able to understand. In the next section, we describe concretely how the screen reader's inability to decode the relation on a page can complicate a blind person's ability to understand and interact with that page. The same software constraints that prevent programs from reading relations on web pages cause tremendous difficulty for blind people, because the software they use, i.e. the screen reader, cannot read the relation either.

The accessibility space provides an opportunity to decouple the low-level task of identifying the parts of the relation in a web page from the higher-level task of understanding and manipulating the contents of the relation. Like a computer program, a blind person views the web without recourse to the miraculous machinery of the human eye. Unlike a computer program, however, the blind person can use their human brain to decode the information presented on the page, even when the structure of the web page makes that task capriciously difficult.

Because blind people would benefit from the presence of wrappers, they may present a resource for solving additional unsolved problems in the Smart-Wrap work. For example, they can detect web pages that contain data sets that should be wrapped, by requesting a wrapper for the page. Additionally, they can detect when web pages have changed so that a contributed wrapper fails to match the relation on the page. These capabilities, however, are contingent on the actual and perceived value of wrappers for blind people. In the following section, we present the theoretical account of why wrappers would be assistive for blind people using data on the web. In Chapter 7 we describe a study to empirically establish and quantify the benefit provided by wrappers for screen reader users.

## 6.2 Wrappers and Accessibility

The fact that blind people cannot see the computer screen is a simple fact with many ramifications that are not so easily grasped by sighted people. Here we briefly describe some of the consequences, with a focus on presentations of structured data and tables on web pages.

Most blind people who use the internet make use of assistive technology called a screen reader. A screen reader speaks the contents of web pages, serializing the structured content as speech. In the most usual case, a single synthesized voice speaks the contents located at the screen reader user's virtual cursor,[1] which the user maneuvers with screen reader keystrokes. The movements of the virtual cursor are analogous to the eye movements sighted people use to move their attention across text, but have several limitations. First, screen reader keystrokes require more attention and conscious exploration than sighted exploration of the same page [43]. Second, the targets to which the cursor can be reliably moved are limited by the structure encoded in the web page. We focus on this second limitation, and work through a concrete example.

Figure 6.1 shows a result set of products on a shopping site as the site presents it in 2015. Sighted users are able to readily gather a lot of information just by glancing at the page, like that there are several products listed with pictures and some information about them. Moreover, sighted users can quickly look around the page to answer any questions they want. For example, a sighted user interested in comparing the item ratings can visually scan across the bottom of each item.

In contrast, screen reader users maneuver the screen reader cursor around the page using keyboard actions. To scan across item ratings, for example, a

[1] the cursor is often likened to "the end of a soda straw," through which the screen reader user is trying to understand the whole of the document around the cursor [33].

screen reader user has to recognize that ratings are the fifth or sixth attribute of each item, then maneuver the cursor that many times for each item. Screen reader users perform exactly this type of action, which Yesilada et al [106] call "specific reading patterns", often when consulting data on the web. To use a specific reading pattern, the screen reader user has to learn the pattern, and then mechanically apply the pattern to move within the data set.

The way the page is coded can have a large effect on how the screen reader user can move around within it. Figure 6.2 shows the same information presented using the html `<table/>` tag. While the difference between Figure 6.1 and Figure 6.2 is minor for sighted users, it can have a significant impact for screen reader users. Figure 6.3 shows the transcript of the text that a screen reader vocalizes for the information shown in Figure 6.1 and Figure 6.2. The bolded text depicts text vocalized for the tabular layout of Figure 6.2 but not for the grid layout of Figure 6.1.

The bolded cues in Figure 6.3 assist screen reader users in three important ways. First, the very first line informs the user that the page is organized as a table. This knowledge allows the user to skip a lot of the exploratory activity often devoted to understanding the page organization. Second, the explicit specification of where rows begin and end helps the user better determine how to segment the information. Without this information, users must memorize that ratings belong to the preceding, not the following, listed item. Third, the bolded cues explicitly identify which field is which.

An additional benefit of the table structure is that users can use table commands to move their cursor along the rows and columns of the table. Oogane and Asakawa [82] introduced table navigation mode in 1998. As a testament to its utility, the mode was subsequently adopted in nearly every commercial screen reader. Using table navigation mode, comparing ratings of the items becomes a simple matter of moving the cursor up and down in Column 8.

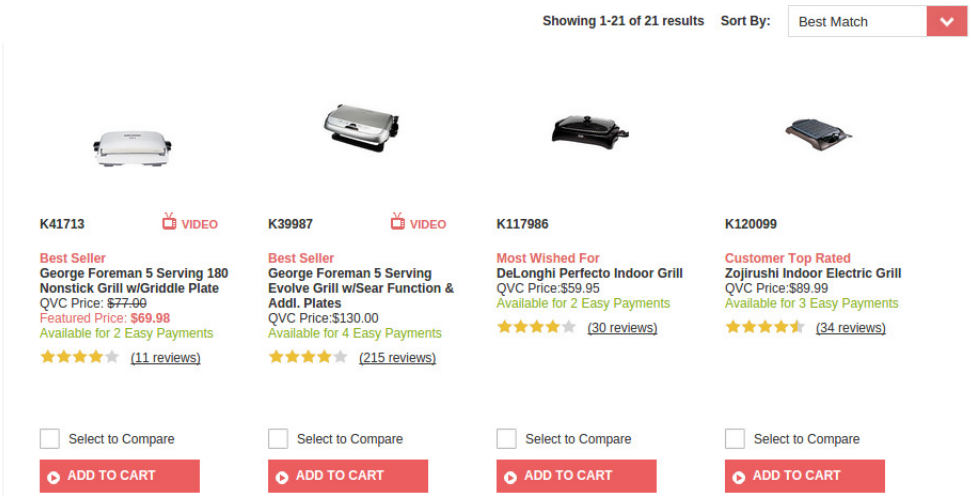


Figure 6.1: A web page presenting a data set as presented by the web author, using a grid layout


Figure 6.2: A web page presenting the same information as Figure 6.1, using a tabular layout

**table with 22 rows and 9 columns**
**row 1 column 1 Picture**
**column 2 Product Number**
**column 3 Video**
**column 4 Product Info**
**column 5 Product Name**
**column 6 Regular Price**
**column 7 Featured Price**
**column 8 Rating**
**column 9 Reviews**
**row 2 Picture column 1**
link graphic George Foreman 5 Serving 180 Nonstick Grill w/Griddle Plate - K41713
**Product Number column 2**
link K41713
**Video column 3**
link VIDEO
**Product Info column 4**
Best Seller
**Product Name column 5**
link George Foreman 5 Serving 180 Nonstick Grill w/Griddle Plate
**Regular Price column 6**
QVC Price: $77.00
**Featured Price column 7**
Featured Price: $69.98
**Rating column 8**
4.2 of 5 Stars
**Reviews column 9**
(11 reviews)
**row 3 Picture column 1**
link graphic George Foreman 5 Serving Evolve Grill w/Sear Function & Addl.
Plates - K39987
**Product Number column 2**
link K39987
**Video column 3**
link VIDEO
**Product Info column 4**
Best Seller
**Product Name column 5**
link George Foreman 5 Serving Evolve Grill w/Sear Function & Addl.
Plates
**Regular Price column 6**
QVC Price:$130.00
**Featured Price column 7**
**Rating column 8**
3.8 of 5 Stars
**Reviews column 9**
(215 reviews)

Figure 6.3: A transcript of the screen reader's vocalizations of the information in Figures 6.1,6.2; the structure information in bold is only spoken when the information is presented as a `<table/>` as in Figure 6.2.

Table navigation mode allows the screen reader user to move within the logical structure of the table. The screen reader is able to facilitate this logical movement because the `<table/>` tag expresses the logical structure in machine readable form. In the grid layout of Figure 6.1, each item is enclosed in a `<div/>` tag, and the screen reader has no way of knowing that this tag delineates an item. In the table layout of Figure 6.2, each item is delineated by a row element, allowing the screen reader to treat the item as a logical row. Only when the semantics of the document are encoded in machine readable form is it possible for the screen reader, which is after all software running on the user's computer, to let the user exploit that structure.

In the absence of tabular structure, screen reader users employ a number of strategies to navigate and understand data presented on web pages. The simplest possible, but very slow, strategy is to read through the entire page. The simple reading strategy can be sped up a bit with a little effort by skipping past each item once the user decides it is not relevant. The "reading pattern" strategy, described above, is faster again but requires more effort from the user. A much faster strategy is to use the find operation of the browser. The find operation requires a fair amount of effort, since the user needs to correctly predict a string occurring near the value sought. We expect that the ability to move logically in a table will assist screen reader users because it permits a movement strategy which is simultaneously fast and simple to use. Figure 6.4 depicts conceptually the strategies described in this paragraph.

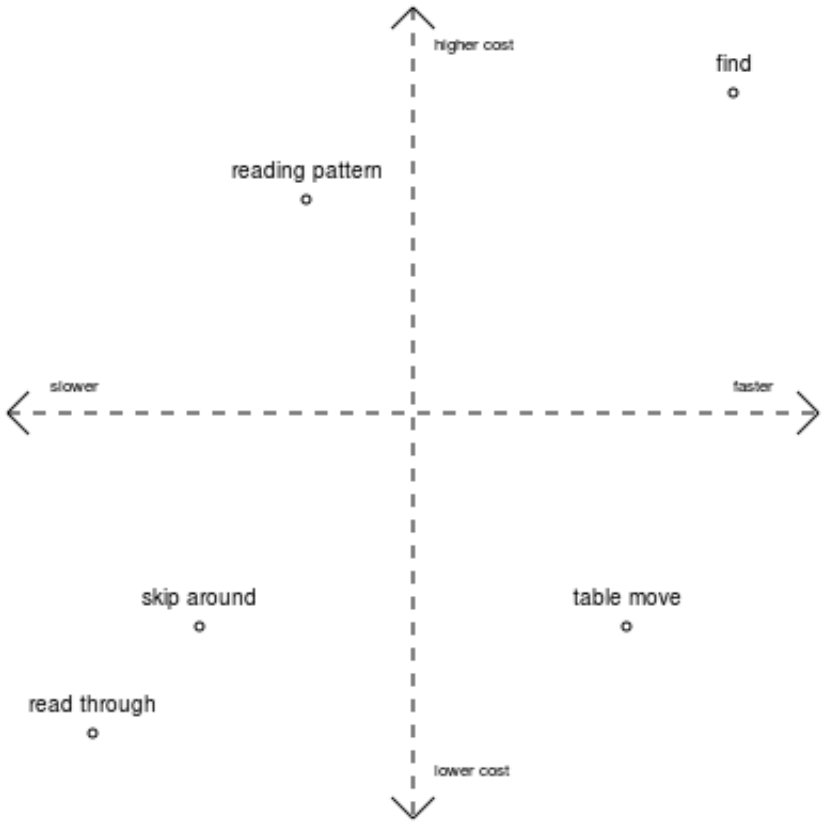


Figure 6.4: Several techniques for navigating around a web page in a screen reader

# Chapter 7

# enTable Evaluation

## 7.1 Introduction

The digitization of vast amounts of information into today's internet has been a tremendous boon to people who cannot access printed material. Whereas print documents are, in practice, only infrequently transcribed into accessible forms (e.g. Braille), digital documents are available to anyone who has access to the internet. People who are blind or have other visual impairments are able to use assistive technologies, such as a screen reader or a screen magnifier, to view information that would have been practically unavailable in a print-centric world.

Digitization, however, is not merely a binary state of digitized or not. We can digitize a print document in many ways, on a continuum we may call accessibility. As a minimally accessible example, we could provide the pages of a print document as digital images. The page images would allow a sighted person to read the document, but (barring perfect OCR) would not make it possible for a blind person to read it, nor for a sighted person to perform searches on the text of the document. A more accessible iteration of the document might render all the text of the document, in linear order,

as machine-readable text. For documents containing primarily running text, such as a novel, the linear rendering provides equal access for sighted and visually impaired people. But for documents containing a large amount of structured data, like bank statements, tax tables, or fantasy sports statistics, a linear ordering of the page contents quickly becomes a bewildering blur of words and numbers. We illustrate this point in greater detail below.

To make a more accessible iteration of the document we insist that information presented as tables in the print version be presented in equivalent form to people who cannot see the page. Here, intuition starts to fail. What would constitute an equivalent form? And how could we tell if we've found one?

Oogane and Asakawa [82] provide an invaluable and surprisingly intuitive answer: for blind people, an equivalent form to a table is a table. Just as the horizontal and vertical alignment properties of the rows and columns of a print table allow the eye to explore the logical relationships entailed by the table, a blind user of the table can understand and enjoy the relationships of the table by being empowered to move the cursor about the two-dimensional horizontal and vertical, i.e. logical, axes of the table. Oogane and Asakawa implemented this feature for instances of the html `<table/>` tag, and subsequently, as a testament to its utility, the feature was soon adopted in every major screen reader product.

In this chapter we argue that the utility of permitting tabular movement, as proposed by Oogane and Asakawa for instances of the `<table/>` tag, applies to a larger class of semantic structures encountered on the web. Tables are a typographic convention for communicating a set of logical relationships between the rows and columns of the table. It has, however, become common on the web for authors to make use of non-tabular layouts to present datasets. These non-tabular layouts effectively communicate the logical structure to sighted users, but users of screen readers are not able to reconstruct the

logical structure. As a result, consulting data sets is a significantly more difficult and time consuming task for blind users than for sighted users of the internet. This chapter quantifies this discrepancy and estimates the efficacy of efforts to augment web data sets with structural information.

This chapter is structured as follows. We begin by describing in greater detail how datasets presented without tables obscure the structural relationships in a screen reader. We then describe a preliminary study to understand the magnitude of this effect on expert screen reader users. We present the quantitative results of the study, suggesting that rewriting web data sets as tables augments the efficiency and effectiveness of screen reader users looking up information on the web. We then present some qualitative observations from the study that have implications for the insertion of assistive tables in web data sets, as well as for the design of screen readers.

## 7.2 Background

People who cannot see the screen at all typically use an assistive technology called a screen reader, which serializes the contents of web pages as a stream of audio text. Browsing using a screen reader is consistently slower than browsing visually, as attested by several studies. Researchers have compared the performance of sighted and blind users on plausible internet tasks. While the precise results will depend on the specific tasks used, here we draw out some estimates from the literature to understand the order of magnitude of the gap. Pernice and Nielsen [84] asked groups of sighted and blind users, each with 20 participants, to complete four tasks. The average blind participant completed half a task in about 17 minutes, or about 1.8 tasks per hour. The average sighted participant completed about three tasks in about 7 minutes, or about 26 tasks per hour, yielding an estimate that blind participants take about 15 times as much time as sighted participants. Takagi et

al [95] report that 5 blind participants complete a task in an average of 536.6 seconds, about 12 times as long as the 5 sighed participants, who complete a task in an average of 43.6 seconds. Craven and Brophy [24] asked 20 visually impaired and 20 sighted participants to complete four tasks, and report that the visually impaired participants spent between 1.4 and 4 times as long as sighted participants on the tasks.

It is initially plausible that some part of the speed disadvantage of screen readers is due to listening to speech simply being slower than visual reading. But in fact, screen reader users routinely [3, 93] set their screen reader speeds to around 300 words per minute (wpm), or roughly the rate at which university-educated sighted people read visually [16]. Furthermore, expert screen reader users exhibit what amounts to a superpower, in the ability to understand and transcribe the majority of synthesized speech at rates of 500 wpm and above [3, 93, 29]. There is some evidence that blind people repurpose a part of the visual cortex, active during visual processing in the brains of sighted people, to accomplish this feat [29]. The faster rate of 500 wpm is roughly equivalent to the speeds employed by sighted people in skimming [16]. Thus if web use were simply a matter of reading or skimming web pages, then we would expect screen reader use and visual browsing to take about the same amount of time.

Sighted internet users, however, typically do not read or even skim most browsed pages [80]. Instead, they scan at rates of 600 wpm [16] or higher, seemingly beyond, alas, the superpowers honed by regular screen reader users. People scanning typically have little ability to reproduce the text scanned. They do, however, acquire a great deal of orientational information about the document – for example that the document contains a banner and a sidebar that are ancillary to the main content – which they subsequently use to choose which parts of the document to attend to in order to accomplish the task at hand. In contrast, blind web users devote a large amount of time

and effort [43, 95] simply exploring and understanding the structure of the document before they even begin to address the main content.

Visual scanning of web pages is automatic and feels practically effortless. Manipulating the screen reader cursor around the page, in contrast, is an intentional and relatively laborious affair. To pull it off, screen reader users employ a variety of strategies [12, 2, 106], and devote time and effort to selecting and executing the appropriate strategy for the page under consideration.

The design of a web page can significantly assist or impede the effectiveness of screen reader users' strategies. In general, explicit structure, such as headings, assists screen reader users. Noise, such as images without informative alternative text, impedes their efforts. These design constraints have been standardized by the World Wide Web Consortium [17, 14]

The WCAG standard provides guidelines for web authors to make their sites accessible. Many web sites do not conform to the guidelines, and exhibit many usability problems for screen reader users. A number of systems have been proposed to rewrite, or transcode, web sites in a way that improves their accessibility. Such systems rest on two propositions. First, the transcoding must be possible to perform at a large scale. Second, the transcoding should be helpful for screen reader users. Dante [106] inserts headings and other reachable landmarks into web sites. The annotations are derived automatically where possible, and are solicited from ontology experts. Social Accessibility [90] inserts headings and alternative text for images, as well as reordering the elements of the page to better match the visual experience of the page. The Social Accessibility project constructed a user-friendly interface for end users to provide annotations, and found that a significant number of internet users were willing to contribute a small amount of effort towards improving sites for blind people. SmartWrap [41] demonstrated that end users are able to provide structural information about datasets, allowing

them to be re-written in more accessible forms.

To understand the benefits provided by the transcoding systems, the researchers have compared screen reader users' interactions with the transcoded and untranscoded versions of the sites. Yesilada et al [106] found that blind participants rated improved pages significantly better than original pages on a scale measuring cognitive workload involved in accessing them. Takagi et al [95] find that improved pages allowed blind participants to better focus their efforts on the main content of the page, but they did not detect an improvement in the net time on task. Their analysis indicates that many participants were not able to recognize and exploit the accessibility improvements in the experiment. The present work quantifies the potential benefits to screen reader users of adding structural information to datasets, whether from a system like [41] or by the web authors.

## 7.3 Method

The specification for tables in HTML [87] attempted to provide a simple mechanism which was sufficiently flexible to represent the rich variety of tables found in existing print documents. In doing so, the specification also formalized one of the most powerful and predictable mechanisms for layout in the early web. Web designers made heavy use of the `<table/>` tag for both purposes. The use of the `<table/>` tag in practice gave rise to two problems with regard to tabular data for visually impaired web users.

1. [`<table/>` tag *overuse*] The `<table/>` tag is widely abused to effect a grid-like layout rather than to represent tabular data.[1]

[1]The W3C explicitly recommends against using the `<table/>` tag for layout: "Tables should not be used as layout aids. Historically, many Web authors have tables in HTML as a way to control their page layout making it difficult to extract tabular data from such documents." [53]

2. [<table/> tag *underuse*] Conversely, designers frequently present tabular information (e.g. the results of a database query) using repeating templates rather than the <table/> tag.

The study reported here is designed to measure and disentangle the effects of these two phenomena with respect to the ability of screen reader users to understand and use tabular data.

We recruited $N = 19$ experienced screen reader users to complete a study using a screen reader. Participants were paid for their time. We offered participants the option of using their own computer setup at a place convenient to them, or they could choose to come to our lab and use a computer with JAWS installed. Participants were encouraged to adjust the settings of the screen reader to suit their preferences. Participants were asked to complete a brief survey of questions with respect to their expertise using a screen reader, after which they were asked to perform some factual lookup tasks. After they finished the tasks, we conducted a brief interview to follow up on any behaviors observed during the tasks. The tasks and interviews were recorded on video for subsequent analysis.

We constructed a series of 12 lookup tasks on web pages, and asked participants to complete them. Each participant began with a single practice task.[2] This practice task gave them a chance to get used to the mechanical process of reading a question, following a link to a web page, looking up the answer in that page, returning to the question page and submitting the answer in a simple form. The participant was asked to find the answer to each question within the limits of a single page, without following any outgoing links or filling any forms. Participants who strayed from the task page were reminded of this artificial constraint and asked to return to the page. After the practice task, the participant proceeded through the tasks one by one, in

[2]The practice task asked participants to look up the population of Istanbul in a Wikipedia table listing statistics about many cities.

an ordering which counterbalanced task order as well as condition order. The time from when the user first reached the question page until they submitted the answer was recorded. Time spent on pages reached by outgoing links was manually subtracted from the recorded time. Additionally we graded each answer as correct or not. Observations from the practice task were discarded.

We constructed the tasks using web pages captured from the web in 2015. The independent variable is the structure used by the web page. In the control, or "raw" condition, the web page is presented as originally captured from the web. All captured web pages present a single page of tabular data using a repeating template, not a `<table/>` tag. In the "fix underuse" condition the template presenting the information are replaced with a `<table/>` tag structures depicting the same information, with descriptive headers for all columns. In the "fix overuse" condition, the web page is additionally manipulated to remove all uses of the `<table/>` tag that do not present tabular data. The introduced data tables were generated by crowdworkers using the SmartWrap [41] tool. Each task existed in all three forms, and the appropriate one was shown to each participant.

Based on the video, we also labeled tasks as to whether participants used the various strategies described above, such as utilizing table structure or searching for text using a find operation. We expect the use of strategies to affect participant performance in two ways. First, the widespread and creative use of the browser find operation [106] may confound our measurements. We account for this by regarding the find operation as a blocking factor. Half of the tasks we constructed in such a way that the question contained a string which could be used to located the required information. In the analysis, we refer to these tasks as "findable." The other half did not have such a string. Second, our experimental manipulation introduces table structures into the page, but there is a distinct possibility that some participants will not recognize or use the introduced structure [95]. By observing

Table 7.1: Task types for the tasks. Tasks types with an asterisk were constructed to be findable.

| Task Type | Description | Example Question |
|---|---|---|
| key-value* | Given a key, lookup a value corresponding to the key | Find the phone number of a practice with name "Lavan Legal" |
| value-key* | Given a value, lookup the key | Find the name of a player born in Akron OH |
| inequality | Find a key with value greater (or less) than a provided value | Find the name of a company with over 100 applicants |
| null value* | Look for a value not listed in the page (correct answer is null) | How many years of experience does Dr. David Pier have? |
| intersection | Look for an item meeting two criteria | Find the brand of a grill rated 5 stars for under $100 |
| google | Locate a URL based on information in the snippet | Find the URL of a site promising both quotes and poems from Maya Angelou |

the participants' completion of the tasks we are able to directly observe use of tables, not just presence of tables. We expect use of tables to play a mediating role in improving participant performance. Similarly, we expect the confounding use of the find operation to play a mediating role in the effect of the availability of the find in "findable" tasks.

The tasks were designed to explore a range of issues which we expected to play a role in the utility of using data tables, using several task types as summarized in Table 7.1. The first two task types were chosen to be relatively simple. The inequality task type is not solvable by the find operation because there is no predictable string to search for. The null value task type is difficult because non-table layouts are often difficult to disambiguate between a misplaced and a missing field. Participants were instructed to enter "no answer found" when they believe the page did not contain an answer to

the question. The intersection task is made more difficult by necessitating repeated analysis of records in more than one field. The google task type was included to see how users interacted with familiar pages.

Within each task type, questions were constructed to be as natural as possible, while having a unique and unambiguous answer. When grading tasks, we counted multiple variations on the answer as correct. Additionally, we counted it as correct when the participant retrieved the wrong field of the right tuple, e.g. an address when the question asked for a phone number. Information from the wrong tuple was counted incorrect. As a baseline to validate that the questions were mostly unambiguous and also trivial for sighted users, we asked crowd workers on Mechanical Turk to complete the same tasks using the same web forms.

Based on the time spent on each task and whether or not the answer was correct, we compare the effectiveness and efficiency of participants in the different conditions. We hypothesize that introducing data tables will increase the effectiveness of participants, i.e. it will help them find more correct answers. We also hypothesize that introducing data tables will increase the efficiency of participants, allowing them to find an answer in less time. We expect that removing non-data tables will further augment participants' efficiency. Finally, and most fundamentally, we expect that screen reader users will find and make use of the data tables we put in the experimental conditions.

Hypotheses:

**H0** Screen reader users will recognize and use tables when performing data lookup tasks

**H1** Use of data tables will increase the efficiency of screen reader users on data lookup tasks

**H2** Use of data tables will increase the effectiveness of screen reader users

on data lookup tasks

**H3** Removal of non-data tables will increase the efficiency of screen reader users on data lookup tasks

## 7.4 Study Demographics

The 19 screen reader users who participated in the study were all blind. Two participants had a very small amount of residual vision. One participant used voice commands to control the screen reader, and the rest were able to use a keyboard.

Ten participants chose to work on a computer provided by the experimenter. Nine participants chose to work on their own computers; four in their home, four in their workplace, and one brought a laptop to a public library. All participants reported regularly using a screen reader on a computer and on a mobile device.

16 participants indicated that JAWS was their primary desktop screen reader; two indicated VoiceOver, and one indicated System Access. All participants except for one were able to use their primary screen readers for the study; the one exception was a Mac user who reported using JAWS regularly and used JAWS for the study. This participant was also the only one who did not use their preferred browser for the study, using Internet Explorer instead of Safari. One other participant preferred and used Safari. Four participants preferred and used Firefox, and twelve preferred and used Internet Explorer. One participant preferred and used the separate browser provided by System Access. These demographics are broadly similar to those found by WebAIM.[3]

Six participants reported slight or no customization in their screen reader

[3]http://webaim.org/projects/screenreadersurvey6/

configurations. Nine participants reported customizing their settings "somewhat," and three of them did not use their own, customized computer for the study. Four participants reported their settings "very customized" and also did not use their own computer.

10 participants reported their screen reader proficiency as Intermediate, and 9 as Advanced. 11 rated their internet proficiency as Intermediate, and 8 as Advanced. No one identified as a Beginner. All participants reported familiarity with table commands.

Participants were encouraged to configure the screen reader to suit their preferences. The median speech rate used by participants was around 300 wpm, roughly equivalent to the sighted reading rate. Four participants who used a slower speech rate of around 180 wpm may have done so because they did not adjust the rate from its default on a computer provided by the experimenter.

## 7.5 Quantitative Results

The 19 participants completed 210 tasks, responding with the correct answer in 160 tasks and an incorrect answer in 50, or about 76% correct. The median time spent on a task was 150 seconds. In the sighted baseline, 68 sighted people completed the same tasks with median time on a task of 42 seconds and got about 91% right. Table 7.2 breaks these measurements down across the conditions and findable tasks.

In 144 tasks participants were using pages with data tables. In 78 of these, or 54%, participants used at least one table operation. Participants attempted to use the find operation in 121 tasks, or about 58% of tasks. They successfully found at least one string in 98 tasks, or about 47% of tasks. Table 7.3 summarizes the use of these strategies across condition and findable tasks.

Table 7.2: Summary of participants' performance on the tasks

| Condition | Find-able | Num | Median time (s) | % correct |
|---|---|---|---|---|
| untranscoded | no | 31 | 265 | 65 |
| fix underuse | no | 32 | 188 | 75 |
| fix overuse | no | 41 | 192 | 66 |
| untranscoded | yes | 35 | 141 | 89 |
| fix underuse | yes | 37 | 146 | 81 |
| fix overuse | yes | 34 | 147 | 82 |

Table 7.3: Summary of strategies participants used on the tasks

| Condition | Find-able | % used table | % tried find | % used find |
|---|---|---|---|---|
| untranscoded | no | 0 | 61 | 45 |
| fix underuse | no | 56 | 34 | 21 |
| fix overuse | no | 58 | 41 | 29 |
| untranscoded | yes | 0 | 80 | 71 |
| fix underuse | yes | 48 | 70 | 62 |
| fix overuse | yes | 52 | 58 | 50 |

A Shapiro-Wilk test rejects the null hypothesis that overall time on task is normally distributed ($W \approx 0.86, p < 0.001$). In the following, we analyze the logarithm (base 2) of the time on task.

Figure 7.1 shows the time participants spent on tasks in the various conditions. The figure suggests that participants spend less time on findable tasks than unfindable tasks. A t-test confirms this is statistically significant ($t_{208} \approx 5.2, p < 0.001$). A Sobel test indicates that searching for part of the question mediates the effect on time of the availability of find ($z \approx -4.48, p < 0.001$), i.e. findable tasks are only faster if participants perform the find action with a good string from the question. The mediation analysis recommended by Imai et al [59, 97] agrees ($p \ll 0.001$).

Figure 7.1 also suggests that participants complete unfindable tasks in the original condition more slowly than tasks in the two fixed conditions, but faster in findable tasks. The effects for the findable tasks are not sta-

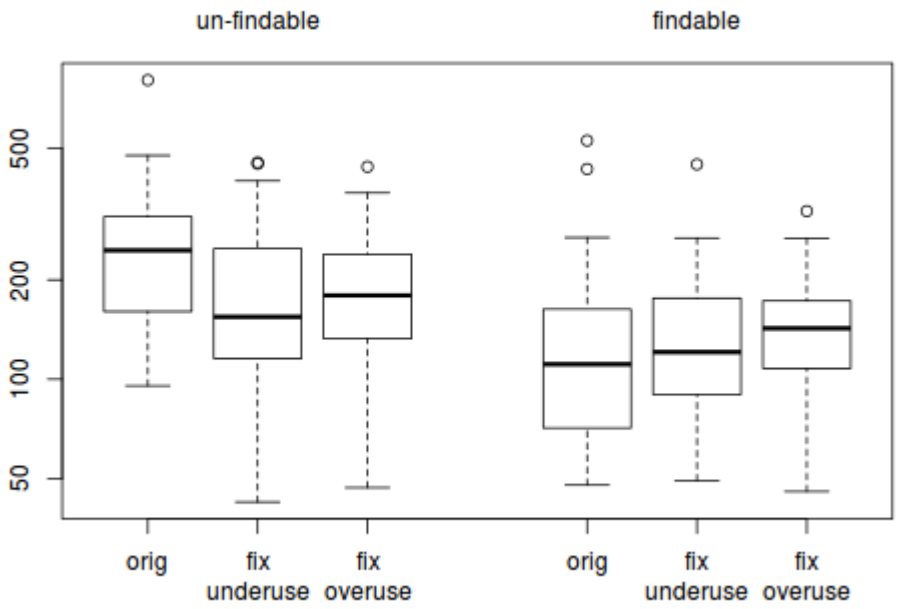


Figure 7.1: Time spent on task compared across condition and findable tasks (log scale)

tistically significant. For unfindable tasks, t-tests confirm that tasks in the original condition take less time than those in the fix underuse condition ($t_{61} \approx 2.58, p < 0.05$) and those in the fix overuse condition ($t_{62} \approx 2.4, p < 0.05$). Adjusting via the Bonferroni correction for the six pairwise comparisons suggested by Figure 7.1 makes these differences less compelling. Using the mediation analysis as before indicates that presence of tables is mediated by use of tables ($p < 0.05$), when use of find is taken into account. In other words, participants are only faster on tasks with tables when they actually make use of the tables.

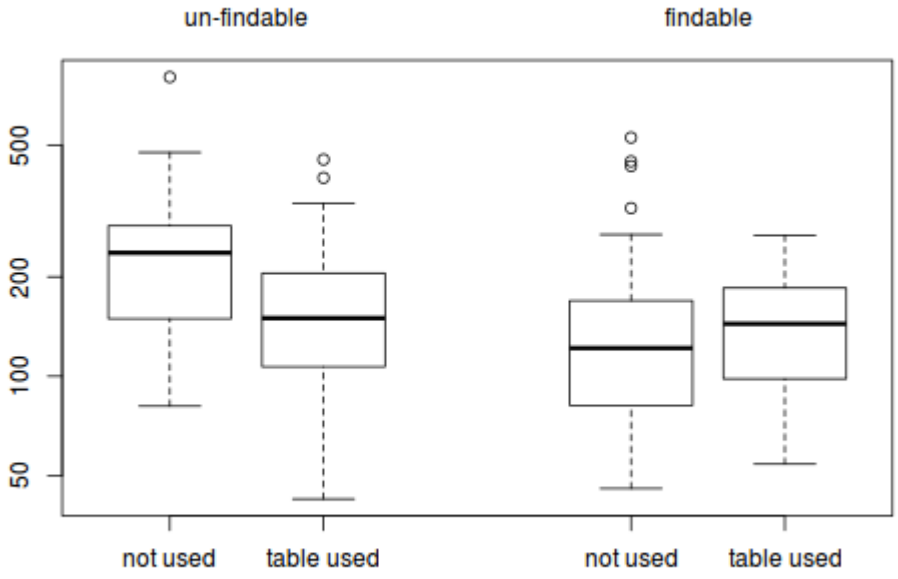


Figure 7.2: Time spent on task compared by table use and findable tasks (log scale)

Figure 7.2 shows the time spent on tasks broken down by whether tables were actually used. As before, there is no statistical difference for findable

Table 7.4: Fixed effects of a regression for log time on task versus table use and find use

| Factor | Coefficient | p value |
|---|---|---|
| intercept | $7.71 \pm 0.12$ | $\ll 0.001$ |
| table use | $-0.49 \pm 0.12$ | $\ll 0.001$ |
| find use | $-0.98 \pm 0.14$ | $\ll 0.001$ |
| non-data tables | $-0.05 \pm 0.11$ | $> 0.05$ |
| use table and find | $0.64 \pm 0.23$ | $< 0.01$ |

tasks. For unfindable tasks, participants who used tables were statistically faster than those who did not ($t_{86} \approx 4.11, p \ll 0.001$).

Table 7.4 shows the fixed effects for a regression of log task time on table use and find use. The model also includes a random effect for each task. Based on the estimates in Table 7.4 the effect of using tables, holding all other factors constant, is a time reduction of 23% to 35%. Adding a factor for the presence of non-data tables to the model does not identify a statistically significant effect.

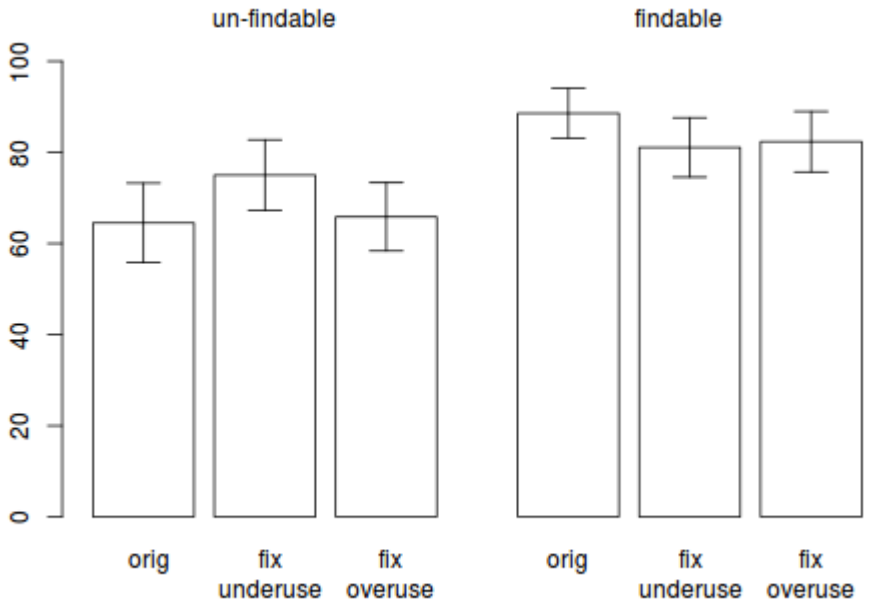


Figure 7.3: Percent correct compared across condition and findable tasks, with standard error

Figure 7.3 shows the effectiveness of participants in the various conditions. Again, findable tasks appear to be easier than unfindable tasks, and a t-test confirms this ($t_{196} \approx -2.7, p < 0.01$). A mediation analysis does not find that presence of a search-able string is mediated by actual use of that string; findable tasks appear to be easier whether or not users perform an appropriate search. No other differences suggested in Figure 7.3 are statistically significant. The effect on accuracy of table availability is again mediated by actual usage of the table ($p < 0.01$). Figure 7.4 compares the effectiveness of participants who used tables to the effectiveness of those who did not.

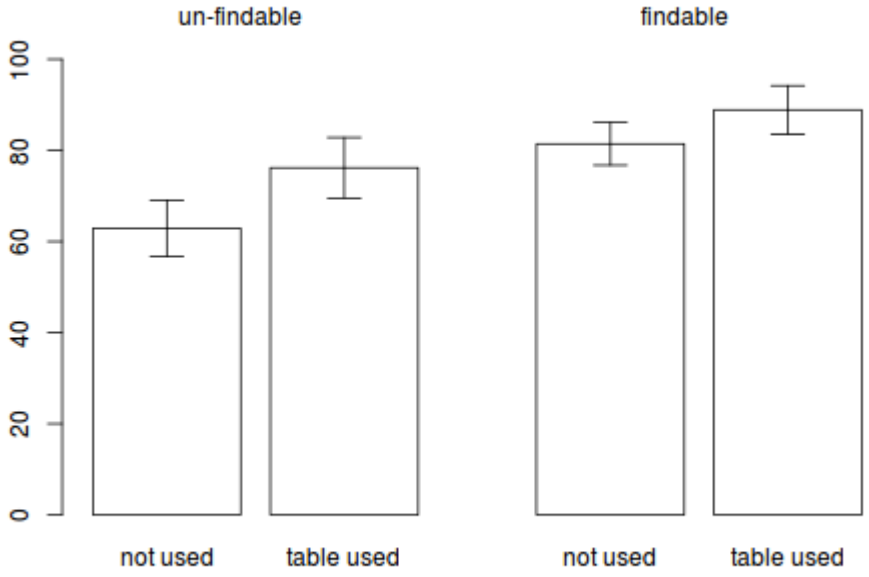


Figure 7.4: Percent correct compared by table use and findable tasks, with standard error

Figure 7.4 suggests that using tables helps participants locate a correct answer in all types of tasks. A Bayesian estimate of the difference, which pro-

Table 7.5: Fixed effects of a regression for correctness on task versus table use and find use

| Factor | Coefficient | p value |
|---|---|---|
| intercept | $0.39 \pm 0.34$ | $> 0.05$ |
| table use | $0.79 \pm 0.38$ | $< 0.05$ |
| find use | $1.14 \pm 0.43$ | $< 0.01$ |
| non-data tables | $0.35 \pm 0.36$ | $> 0.05$ |

Table 7.6: Coefficients of a proportional hazard model viewing correct task completions as events and incorrect tasks as (right) censored

| Factor | Coefficient | exp(coeff) | p value |
|---|---|---|---|
| table use | $0.75 \pm 0.2$ | 2.13 | $< 0.001$ |
| find use | $1.41 \pm 0.21$ | 4.12 | $\ll 0.001$ |
| find and table use | $-0.83 \pm 0.35$ | 0.43 | $< 0.05$ |

ceeds by drawing from the posterior distribution assuming the probability of being correct is drawn from an uninformative conjugate beta distribution, finds a 93% probability that the effect is positive. Table 7.5 shows the coefficients of a logistic regression with a random effect for task on task correctness. This model estimates the marginal effect of using tables as an improvement to effectiveness of 3–22%.

The preceding analyses model time on task and task correctness independently. A non-parametric survival model [6] allows both measurements to be modeled jointly, viewing an incorrect response as "censoring" a correct response the participant might have reached had they recognized their error. Table 7.6 shows the coefficients of a proportional hazard model, indicating that use of table roughly doubles the "hazard" of successfully completing a task, where use of find quadruples the hazard. This model estimates that use of tables reduces the median time to a successful task completion by 21–47%.

By consulting the baseline of sighted users, we can compare the time spent completing the tasks by each screen reader user to an average sighted peer

who experienced exactly the same pages as they did. Using this analysis, we find that screen reader users who did not use tables took about 5.8 times as long to complete the tasks. In contrast, screen reader users who did use tables took about 3.1 times as long, a reduction of about 2.7.

## 7.6 Qualitative Results

As noted above, data tables only assist screen reader users when users are aware of and make use of the tables. Here we discuss some observations with respect to how users found and explored tables.

Screen readers provide a command with which users can directly to the next table in the document. In response the cursor will move to the next table, and if there is no such table the screen reader will pronounce "there are no tables on this page." Participants attempted to jump to a table in 6 of 66 pages in the original condition, and in 18 of 144 pages when there was a table to find. Several participants made use of a table in the page to find where their cursor had been after re-reading the question. We observed this pattern of immediately jumping back to a table in 8 tasks, and we never observed it when there was no table in the page.

Screen readers announce the presence of a table when a user moves the cursor into the table from above. When the user somehow moves the cursor into the middle of the table, for example by using the find operation, the user may not be aware that there is a table in the page. Some users will spontaneously attempt table commands to "probe" whether there is some structure they might be able to leverage. Several participants commented that they got the feeling that a certain page "feels like a table." When such a probe fails, the screen reader announces "not in a table." We observed failed probes in 13 tasks without tables. We observed at least 9 instances of successful probes in pages with tables. One participant characterized table

probing as follows "if I think it's data that I'm looking for, and it would make sense in a table, I got nothing to lose by hitting the letter T a few times just to see." Another participant noted that probing for table structure can be useful to rule out the possibility of using table commands: "okay this list is not in a table so therefore using table commands will be useless, it's not going to help."

In the interviews, we asked participants when they do or do not use tables. One participant said "I love tables ... when I know they're there." Participants were only informed about a table's presence when they maneuvered the screen reader cursor through the beginning of the table. As a result, they frequently were unaware when a table was there. As a participant said, "I would only look for a table if I knew it was there." Several participants suggested that in the web overall they expect informative tables to be very scarce. Reflecting on the tasks, one reported "I didn't figure out soon enough that they were in a table, I just assumed it wasn't because that's been my experience with other, like Amazon, or you know, Home Depot, or wherever I'm looking for stuff." According to another participant, "a lot of websites are constructed this way [using templates], and you just have to live with doing the down arrow." One participant could not come up with a site using tables: "I'm trying to think of a site that I go to, where they do columns and stuff, and I can't think of anything at the moment."

Several participants reported an aversion to using tables. One said flatly "if I had my preference, I do like the up and down, I prefer not to have to do tables, but I will." Another participant, who assists other blind people in learning screen reader skills, said "I don't know whether their level of JAWS, do they know those table keys or not, I'm finding a lot don't." One participant said "I get annoyed with tables sometimes, because there are a lot of tables that don't always work the way they should."

The quantitative results above point to the crucial role of the find oper-

ation in data lookups with a screen reader. Here we describe a few common issues that limited participants' ability to make successful use of the find operation.

The most common failure of the find operation in our study was to search for the comparison value of an inequality. For example, in response to the question "What is the name of a book requested by over 5000 people?" participants would unsuccessfully search for the string "5000". We had intentionally introduced these types of question in the construction of unfindable tasks. We observed these types of search errors in 20 tasks. Several participants recalled these tasks in the interview and speculated that they would like to have a find feature for finding values above or below a certain threshold.

Misspellings of the search string were also common. For example, participants searched for "Istambul" instead of "Istanbul" and "Akrin" instead of "Akron". We observed 11 such misspelled find searches.

Another common failure was when the search string failed to exactly match the target string on the page. In 6 tasks, participants included a trailing space in the search string. For example, a search for the string "Akron" failed to locate the target string "Akron, OH". In 3 tasks, including an extra word made the search too restrictive. For example, a search for the string "David Pier" did not find the target string "David H. Pier". In 2 tasks, users searched for a different form of the word than appeared in the document, e.g. searching for "requested" when "requesting" was used in the document. Many participants preemptively worked around these issues. In 65 search attempts for a two word phrase, participants used only a single word in $$ of them. In 35 tasks, participants searched for only a substring of the word they were looking for, e.g. "peop" for people or "applic" for applicants or applications.

In 11 tasks, participants searched for strings that were insufficiently discriminative to aid them, e.g. "2015". In another 10 tasks participants

searched for strings they conjectured might appear in the document, which did not. For example, a user searched for “*****” to try to find a 5 star rating.

Several participants commented that tables feel faster to them, which one participant summarized by saying "you can move through the down arrow ... you just keep going boom-boom-boom." Some participants noted that tables communicate the structure of the data. One described the ability to ask the screen reader the row and column headers of the current cell: "Even though you're nowhere near the title of the column, you still know that that column, when you right arrow it's going to announce the title of the column that you're right-arrowing into, and also sometimes the title of the row that you're on."

Several participants emphasized that the screen reader's ability to provide contextual information was only as good as the structure provided by the page. One participant qualified that "headings are nice ... as long as they're intentional." Another said "Graphically [pages] look great but if the encoding behind it, isn't really there, then JAWS doesn't know, you know?" In the absence of tables, one participant complained "if it's just a big list of stuff, then often, I usually have to, after I find what I'm looking for, then I have to go back to the beginning of the list, and check the pattern." A few participants mentioned table features that can make tables confusing. One participant reported that some tables redundantly "repeat a line that really only says something once it says it twice when I arrow down in that table, you know, it's not a perfect science." Another participant reported that cells that span more than one row or column are confusing: "it doesn't feel like a maze, right, if it's a chessboard, but once there are cells of different sizes, it would be easy to keep track of it if I can touch it, but if I can't then it's very hard to keep track in my mind, that there is a row there, if there were three rows and then a rowspan, I might not realize that there's a middle row."

## 7.7 Discussion

As shown in Table 7.3, in over half of the tasks where tables were made available, participants put the table structure to use. This indicates that introduced tabular structure into data sets would likely be used by many screen reader users, in confirmation of H0. This is corroborated by the prevalence of participants attempting to jump to and probe tables.

The mediation analyses above provide formal confirmation of the pedantic point that the presence of tables does not necessarily provide any assistance to screen reader users. Instead, they have to use them. In fact, several participants ignored the tables entirely. The mediation analysis confirms that those participants took as much time, and were as accurate, as those who saw pages without tables in them. Similarly, the availability of the find operation in findable tasks made participants faster only insofar as they actually performed the search. The mediated effect did not hold for task accuracy. Participants who did not search on findable tasks still seemed to find them somewhat easier. We speculate that it is easier to recognize a tuple containing a word in the question than it is to recognize a question matching a more complex predicate, such as an inequality.

The model summarized in Table 7.4 suggests support for H1, i.e. that use of tables helps screen reader users accomplish lookup tasks more efficiently. That model estimates that screen reader users who have access to and use tables are 23% to 35% faster than those who do not.

The model summarized in Table 7.5 suggests support for H2, i.e. that use of tables helps screen reader users be more effective at lookup tasks. That model estimates that screen reader users who have access to and use tables are 3–22% more accurate than those who do not.

A survival analysis combines these two effects, and estimates that screen reader users who have access to and use tables will spend 21–47% as much

time successfully completing lookup tasks than those who do not. By matching our participants to sighted peers who saw the same task conditions, we estimate that use of tables reduced the slowdown for using a screen reader from about 5.8× to about 3.1×.

Contrary to H3, we do not find any evidence that removing non-data tables affects either the effectiveness or efficiency of screen reader users on these tasks. Our tasks were constructed from data sets from the real web, and it may be the case that non-data tables are just not a big problem in today's web for screen reader users, either because of clever screen reader heuristics, or because users can quickly discriminate between data and non-data tables. We did not focus on locating web sites that made egregious overuse of the `<table/>` tag, instead we identified sites that presented data sets without the `<table/>` tag then removed any non-data tables they happened to use. A study focusing more narrowly on identifying sites that dramatically overuse the `<table/>` tag may be able to find a significant impediment they present.

The analysis of participant failure in the use of the find operations points to some opportunities for improving the find operation for screen reader users, and potentially for sighted browser users as well. Several participants brought up the possibility of a feature providing search for a particular type (i.e. a number) meeting a particular criterion (i.e. greater than some number). The other find failures point to the very strict nature of exact string search in the browser, especially in contrast to the features available in web search engines. In a web search engine, adding words makes a search less restrictive, added whitespace is ignored, and many morphological forms of search terms are automatically simultaneously searched; the results are then presented in rank order based on the extent they match the query. Web searchers can disable these features by enclosing the query in quotes, but fewer than 6% of searches do so [62]. In exact string search (roughly equivalent to a quoted web search) adding words and whitespace makes a search more

restrictive, and only the queried morphological form is located; the results are presented in strict document order. Participants in our study encountered all of these drawbacks as they attempted to use the find operation to look up information in the task data sets. Finally, web search engines regularly suggest the correction for typos and misspelled words.

The present work measures the utility for screen reader users of having access to data sets presented using the `<table/>` tag. In broad strokes, we could say that screen reader users would be better off if all data sets were written as tables tomorrow. Such a dramatic transformation of the web, however, would have the obvious downside that it make the web harder to use for the majority of its users, the sighted. There are at least two hybrid approaches that could extend to screen reader users the benefits quantified here without disturbing the existing visual aesthetics of the web.

The first hybrid approach is that web authors could markup their data sets as tables using a machine-readable formalism such as RDFa or microdata. To provide the benefits, demonstrated here, of table movement in these marked-up data sets, screen reader developers would need to also allow table navigation mode within them. The present study was predicated on the observation that at present the `<table/>` structure is the only representation of the data set concept that is shared between the capabilities of web authoring tools, web browsers, and screen readers.

We have explored the second hybrid approach in our previous work on SmartWrap [41]. In that work, we describe the SmartWrap tool, which empowers end users, explicitly including non-programmers, to construct a wrapper for a web data sets in minutes. A wrapper is a small program which contains exactly the information needed to transform, or "wrap", the contents of a data set into an alternative form, such as a table. Given a wrapper, a page can exist in its current form for sighted people but be re-rendered in a different, tabular view for screen reader users. When the underlying page

template changes, SmartWrap can be used again to generate a new wrapper promptly.

The present work focuses narrowly on the direct benefits to screen reader users provided by having access to data sets in tabular form. To be sure, there is tremendous potential indirect benefit to screen reader users, and to everyone, of stipulating access to web data sets in machine-readable form. The Semantic Web [92] proposes encoding all web information in machine-readable form, unlocking the potential to build semantic agents that can understand and execute high-level queries. But high-level queries are built up from lower-level queries. Here we study, and quantify, the benefits provided by machine-readable markup to a human operator in directing a general-purpose semantic agent, i.e. the screen reader, in completing a low-level data query. Whatever larger potential benefit a machine-readable web promises, blind internet users are a growing constituency who can benefit from them now.

## 7.8 Conclusions

In this chapter, we have provided evidence that screen reader users make use of tables as data sets when they are provided in tabular form. We evaluated screen reader user performance on web data sets that had been transcoded as tables, and found that users could understand and use the information in this form. Furthermore, we find that consulting the resulting tables is much faster and somewhat more accurate than the original data sets as observed on the web. Contrary to our initial expectations, we do not find any evidence for or against the proposition that non-data tables impede the performance of lookup tasks on web data sets.

We see some evidence that screen reader users decide to probe for tabular structure in data sets based on an informal sense that the page "feels like a

table." This presents an initial indication that screen reader users can detect and report pages that contain data sets but do not use a `<table/>` tag. We are currently in the process of building and evaluating a companion system that mediates between screen reader users, who report pages without tabular structure, and users of the SmartWrap [41] system, who provide a wrapper for those pages on a paid or volunteer basis.

# Chapter 8

# Conclusion

In Chapter 4 we described the SmartWrap system, which allows non-programmers to create wrappers for web data sets with little time and effort. In Chapters 3 and 5 we described two application areas for wrappers.

The fundamental point of having a wrapper for a data set is so that a computer can understand the data set. A human visually looking at a data set does not need a wrapper in order to understand it; they can just read it. In this thesis, we have attempted to identify populations of people who interact with data sets in a matter mediated by a computer.

In Chapter 3, the Mixer system addresses the needs of non programmers who face repetitive data retrieval tasks. People who look up different pieces of information, e.g. names from a list of people, on the same web page or set of web pages, are potential Mixer users. Mixer allows them to automate the mechanics of the lookups using programming by demonstration. Chapter 3 shows that administrative end users can effectively demonstrate the actions necessary to successfully generate the desired result. A key issue with the Mixer system is that users have to choose to automate the task using Mixer. We have shown that they are able to do so, but not that they will recognize a Mixer-appropriate task in the wild, and use Mixer for it. We simply assume

that the value provided by the automation will be sufficiently compelling that end users will choose to complete the tasks in a way mediated by the Mixer system, rather than directly.

In Chapter 5, the enTable system addresses the needs of people who use screen readers to access the web. In contrast to potential Mixer users, screen reader users have no choice but to use the web in a mediated fashion. They either access the web mediated by the screen reader, or they do not access the web at all. In Chapter 7 we demonstrate empirically that use of a wrapper can concretely improve the experience of users using the web mediated by a screen reader.

Whereas Mixer has potential to cause a revolutionary change to the way end users accomplish repetitive lookup tasks, enTable is positioned to provide an evolutionary improvement to how screen reader users consult data sets on the web. The revolutionary change of Mixer has potential for high reward [18] but is accompanied by high risk of low user adoption. In contrast, screen readers are used regularly by millions of people today. While the potential reward is more modest, it also comes at lower risk: a wrapper contributed for a page of interest to a screen reader user will help that user achieve their goals. Moreover, wrappers produced to assist screen reader access can be applied to web automation problems. So working with screen reader users provides a way to work concretely with an existing user base to work out the issues with a crowd-based wrapper construction system, while allowing those wrappers to be used by other, more speculative systems.

More generally, the data sets considered in this thesis are an example from a wider class of issues where digital information is incompletely marked up for their semantics. Web authors do as much work as necessary to provide information in a form usable by their intended audience. The tools used for communication dictate how much semantic information must be included. For example, links on the web are nearly always marked up with the seman-

tics of a link (rather than, say as blue text) because web browsers have to know the target location in order to do the right thing when a user activates the link. Template-based data sets, in contrast, are only specified in terms of their content and constraints on the layout. The semantic information that the data set is a data set, i.e. that it has a specified structure of tuples and attributes, is generally not included because web browsers are not expected to do anything useful with that information. But, as shown in in Chapter 7, screen readers do something useful with an explicit representation that part of a page contains a data set: they allow the user to move or scan around the logical structure of the data set. Studying screen reader access to information gives a window into limitations to the semantics encoded in digital representations.

Another interesting facet of screen readers is that their users have access to human intelligence (i.e. their own) and show great facility in pragmatically switching views of the web (e.g. reading through all the text of a document versus reading only the text of the links in it) until they find the information they want. Thus the degree to which a particular view is used can provide implicit feedback on how useful it is. A widely used wrapper is more likely to be correct and useful than one which is not used or is frequently ignored.

From a software engineering perspective, the existence of an existing user base provides a concrete target to work towards. It allows us to use understood methodologies for developing for and testing with user groups. Since expert screen reader users understand their goals, they can provide more concrete and better feedback about features and flaws of the system under development. As described above, the wrappers contributed for enTable use can also be used by Mixer users who want to automate tasks over wrapped pages.

# References


[1] Marco D. Adelfio and Hanan Samet. Schema extraction for tabular data on the web. *PVLDB*, 6(6):421–432, April 2013.

[2] Chieko Asakawa. What's the web like if you can't see it? In *Proceedings of the International Cross-Disciplinary Workshop on Web Accessibility, Chiba, Japan, May 10-14, 2005*, pages 1–8, 2005.

[3] Chieko Asakawa, Hironobu Takagi, Shuichi Ino, and Tohru Ifukube. Maximum listening speeds for the blind. 2003.

[4] Sreeram Balakrishnan, Alon Y. Halevy, Boulos Harb, Hongrae Lee, Jayant Madhavan, Afshin Rostamizadeh, Warren Shen, Kenneth Wilder, Fei Wu, and Cong Yu. Applying WebTables in practice. In *CIDR*, 2015.

[5] Robert Baumgartner. Datalog-related aspects in lixto visual developer. In Oege Moor, Georg Gottlob, Tim Furche, and Andrew Sellers, editors, *Datalog Reloaded*, volume 6702 of *Lecture Notes in Computer Science*, pages 145–160. Springer Berlin Heidelberg, 2011.

[6] J. Beyersmann, A. Allignol, and M. Schumacher. *Competing Risks and Multistate Models with R.* Use R! Springer New York, 2011.

[7] J P Bigham and R E Ladner. Accessmonkey: a collaborative scripting framework for web users and developers. In *W4A*, pages 25–34, 2007.

[8] Harvey Bingham. Cals table model document type definition. Organization for the Advancement of Structured Information Standards, OASIS Technical Memorandum TM 9502:1995, October 1995.

[9] Alan F. Blackwell. Psychological issues in end-user programming. In Henry Lieberman, Fabio PaternÃš, and Volker Wulf, editors, *End User Development*, volume 9 of *Human-Computer Interaction Series*, pages 9–30. Springer Netherlands, 2006.

[10] P. Blanck. *eQuality*. Cambridge Disability Law and Policy Series. Cambridge University Press, 2014.

[11] Michael Bolin. End-User Programming for the Web. Master's thesis, MIT, June 2005.

[12] Yevgen Borodin, Jeffrey P. Bigham, Glenn Dausch, and I. V. Ramakrishnan. More than meets the eye: a survey of screen-reader browsing strategies. W4A '10, pages 13–1, New York, NY, USA, 2010. ACM.

[13] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, August 2008.

[14] Ben Caldwell, Michael Cooper, Loretta Guarino Reid, and Gregg Vanderheiden. Web content accessibility guidelines 2.0. World Wide Web Consortium, Recommendation REC-WCAG20-20081211, December 2008.

[15] Kevin Carey. The case for measurable criteria in HCI for people with impairments. ECRC, 2013.

[16] R.P. Carver. *Reading Rate: A Review of Research and Theory*. Acad. Press, 1990.

[17] Wendy Chisholm, Gregg Vanderheiden, and Ian Jacobs. Web content accessibility guidelines 1.0. World Wide Web Consortium, Recommendation WAI-WEBCONTENT-19990505, May 1999.

[18] C. Christensen. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail.* Management of innovation and change series. Harvard Business Review Press, 2013.

[19] James Clark and Steven DeRose. XML Path Language (XPath) Version 1.0. Technical report, nov 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[20] William W. Cohen. Learning and discovering structure in web pages. *IEEE Data Eng. Bull.*, 26(3):3–10, 2003.

[21] William W. Cohen, Matthew Hurst, and Lee S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 232–241, New York, NY, USA, 2002. ACM.

[22] Michael Cooper, Andrew Kirkpatrick, and Joshue O Connor. Techniques for WCAG 2.0. `http://www.w3.org/TR/2015/NOTE-WCAG20-TECHS-20150226/`, February 2015.

[23] Michael Cooper, Andrew Kirkpatrick, and Joshue O Connor. Understanding WCAG 2.0. `http://www.w3.org/TR/2015/NOTE-UNDERSTANDING-WCAG20-20150226/`, February 2015.

[24] Jenny Craven and Peter Brophy. NonâĂŘvisual access to the digital library (NoVA): The use of the digital library interfaces by blind and visually impaired people. *New Library World*, 104(7/8):321–322, 2003.

[25] A. Cypher. Automating data entry for end users. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, pages 23–30, 2012.

[26] Nilesh Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 335–348, New York, NY, USA, 2009. ACM.

[27] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. Automatic wrappers for large scale web extraction. *CoRR*, abs/1103.2406, 2011.

[28] Anish Das Sarma, Aditya G. Parameswaran, Hector Garcia-Molina, and Alon Y. Halevy. Crowd-powered find algorithms. In *ICDE*, pages 964–975, 2014.

[29] Susanne Dietrich, Ingo Hertrich, and Hermann Ackermann. Ultra-fast speech comprehension in blind subjects engages primary visual cortex, fusiform gyrus, and pulvinar–a functional magnetic resonance imaging (fmri) study. *BMC neuroscience*, 14(1):74, 2013.

[30] Mira Dontcheva, Steven M. Drucker, Geraldine Wade, David Salesin, and Michael F. Cohen. Summarizing personal web browsing sessions. In *UIST*, pages 115–124, 2006.

[31] Hazem Elmeleegy, Jayant Madhavan, and Alon Y. Halevy. Harvesting relational tables from lists on the web. *VLDB J.*, 20(2):209–226, 2011.

[32] Eurostat. Information society statistics. `http://epp.eurostat.ec.europa.eu/statistics_explained/index.php/Information_society_statistics`.

[33] Neal Ewers and Alice Anderson. Introduction to screen readers, 2001. Available at `https://www.doit.wisc.edu/accessibility/resources/video/`.

[34] Ju Fan, Meiyu Lu, Beng Chin Ooi, Wang-Chiew Tan, and Meihui Zhang. A hybrid machine-crowdsourcing system for matching web tables. ICDE 2014, pages 976–987, March 2014.

[35] Amber Feng, Michael J. Franklin, Donald Kossmann, Tim Kraska, Samuel Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. CrowdDB: Query processing with the VLDB crowd. *PVLDB*, 4(12):1387–1390, 2011.

[36] Emilio Ferrara, Pasquale de Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: a survey. *CoRR*, abs/1207.0246, 2012.

[37] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *CoRR*, abs/1207.0246, 2012.

[38] Sergio Flesca, Giuseppe Manco, Elio Masciari, Eugenio Rende, and Andrea Tagarelli. Web wrapper induction: a brief survey. *AI Commun.*, 17(2):57–61, 2004.

[39] Freebase. Main Site. http://www.freebase.com/.

[40] Lex. Frieden and National Council on Disability (U.S.). *When the Americans with Disabilities Act goes online [electronic resource] : application of the ADA to the Internet and the Worldwide Web : position paper / National Council on Disability ; Lex Frieden, chairperson.* National Council on Disability Washington, D.C, 2003.

[41] Steven Gardiner, Anthony Tomasic, and John Zimmerman. Smartwrap: Seeing datasets with the crowd's eyes. In *W4A*, W4A, pages 3:1–3:10, 2015.

[42] Steven Gardiner, Anthony Tomasic, John Zimmerman, Rafae Aziz, and Kathryn Rivard. Mixer: mixed-initiative data retrieval and integration by example. INTERACT'11, pages 426–443, 2011.

[43] Carole Goble, Simon Harper, and Robert Stevens. The travails of visually impaired web travellers. In *Proceedings of the eleventh ACM on Hypertext and hypermedia*, HYPERTEXT '00, pages 1–10, New York, NY, USA, 2000. ACM.

[44] Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 1061–1066, New York, NY, USA, 2010. ACM.

[45] S. L. Greene, S. J. Devlin, P. E. Cannata, and L. M. Gomez. No ifs, ands, or ors: A study of databases querying. *Int. J. Man-Mach. Stud.*, 32(3):303–326, March 1990.

[46] Pankaj Gulhane, Amit Madaan, Rupesh Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeep Satpal, Srinivasan H Sengamedu, Ashwin Tengli, and Charu Tiwari. Web-scale information extraction with vertex. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 0:1209–1220, 2011.

[47] J Gunderson and R Mendelson. Usability of world wide web browsers by persons with visual impairments. In *RESNA*, 1997.

[48] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. Proactive Wrangling: Mixed-Initiative end-user programming of data transformation scripts. UIST, pages 65–74, 2011.

[49] S. Harper and Y. Yesilada. *Web Accessibility: A Foundation for Research*. Human–Computer Interaction Series. Springer London, 2008.

[50] Paul D. Harpur and Nicolas P. Suzor. Copyright protections and disability rights : turning the page to a new international paradigm. *University of New South Wales Law Journal*, 36(3):745–778, 2013.

[51] Ivan Herman, Ben Adida, Manu Sporny, and Mark Birbeck. Rich structured data markup for web documents. `http://www.w3.org/TR/xhtml-rdfa-primer/`.

[52] Ian Hickson. HTML Microdata, Working Draft. `http://www.w3.org/TR/microdata/`.

[53] Ian Hickson. Html5 specification. Technical Report 1.5610, World Wide Web Consortium, 2014.

[54] Andrew W. Hogue and David R. Karger. Thresher: Automating the unwrapping of semantic content from the world wide web. In *WWW*, pages 86–95, 2005.

[55] Chang Hu, Benjamin B. Bederson, and Philip Resnik. Translation by iterative collaboration between monolingual users. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 54–55, New York, NY, USA, 2010. ACM.

[56] Chang Hu, Benjamin B. Bederson, Philip Resnik, and Yakov Kronrod. Monotrans2: A new human computation system to support monolingual translation. In *Proceedings of the SIGCHI Conference on Human*

*Factors in Computing Systems*, CHI '11, pages 1133–1136, New York, NY, USA, 2011. ACM.

[57] Jianying Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong. Why table ground-truthing is hard. ICDAR 2001, pages 129–133, 2001.

[58] David F. Huynh, Robert C. Miller, and David R. Karger. Potluck: semi-ontology alignment for casual users. In *ISWC'07/ASWC'07*, pages 903–910, Berlin, Heidelberg, 2007. Springer-Verlag.

[59] Kosuke Imai, Luke Keele, and Dustin Tingley. A general approach to causal mediation analysis. *Psychological Methods*, 15:309–334, 2010.

[60] Xinxin Wang In. Tabular abstraction, editing, and formatting. Technical report, University of Waterloo, 1996.

[61] Utku Irmak and Torsten Suel. Interactive wrapper generation with minimal user effort. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 553–563, New York, NY, USA, 2006. ACM.

[62] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information Processing & Management*, 36(2):207 – 227, 2000.

[63] Piyushee Jha and George Nagy. Wang notation tool: Layout independent representation of tables. In *ICPR*, pages 1–4, 2008.

[64] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 3363–3372, New York, NY, USA, 2011. ACM.

[65] Shinya Kawanaka, Yevgen Borodin, Jeffrey P. Bigham, Darren Lunn, Hironobu Takagi, and Chieko Asakawa. Accessibility commons: a metadata infrastructure for web accessibility. In *Assets '08*, pages 153–160, New York, NY, USA, 2008. ACM.

[66] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3):21:1–21:44, April 2011.

[67] Tessa Lau. chapter Programming by Demonstration: a Machine Learning Approach. 2001.

[68] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa A. Lau. CoScripter: automating & sharing how-to knowledge in the enterprise. In *CHI*, pages 1719–1728, 2008.

[69] Mark Levene and George Loizou. The nested universal relation data model. *Journal of Computer and System Sciences*, 49(3):683 – 717, 1994. 30th {IEEE} Conference on Foundations of Computer Science.

[70] Clayton Lewis and Gary Olson. Can principles of cognition lower the barriers to programming? In Gary M. Olson, Sylvia Sheppard, and Elliot Soloway, editors, *Empirical studies of programmers: second workshop*, pages 248–263. Ablex Publishing Corp., Norwood, NJ, USA, 1987.

[71] James Lin, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A. Lau. End-user programming of mashups with vegemite. In Cristina Conati, Mathias Bauer, Nuria Oliver, and Daniel S. Weld, editors, *IUI*, pages 97–106. ACM, 2009.

[72] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350. www.cidrdb.org, 2007.

[73] Thomas W. Malone, Kenneth R. Grant, Kum-Yew Lai, Ramana Rao, and David Rosenblitt. Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination. *ACM Trans. Inf. Syst.*, 5(2):115–131, 1987.

[74] Winter Mason and Duncan J. Watts. Financial incentives and the "performance of crowds". *SIGKDD Explor. Newsl.*, 11(2):100–108, May 2010.

[75] Francesmary Modugno and Brad Myers. Graphical Representation and Feedback in a PBD System. In Allan Cypher, editor, *Watch what I do: programming by demonstration*, pages 415–422. MIT Press, Cambridge, MA, USA, 1993.

[76] Hannes Mühleisen and Christian Bizer. Web Data Commons: Extracting structured data from two large web corpora. In *LDOW*, 2012.

[77] Brad A. Myers. Peridot: Creating User Interfaces by Demonstration. In Allan Cypher, editor, *Watch what I do: programming by demonstration*, pages 125–154. MIT Press, Cambridge, MA, USA, 1993.

[78] Bonnie A. Nardi. *A small matter of programming: perspectives on end user computing.* MIT Press, 1993.

[79] Bonnie A. Nardi and James R. Miller. The Spreadsheet Interface: A Basis for End User Programming. Technical Report HPL-90-08, 1990.

[80] J. Nielsen and K. Pernice. *Eyetracking Web Usability.* Voices That Matter. Pearson Education, 2010.

[81] D. A. Norman. *User centered system design: new perspectives on human-computer interaction*, chapter Cognitive Engineering, pages 31–61. New Perspectives on Human-Computer Interaction Series. Lawrence Erlbaum Associates, 1986.

[82] T Oogane and C Asakawa. An interactive method for accessing tables in HTML. In *ASSETS*, 1998.

[83] Hyunjung Park and Jennifer Widom. CrowdFill: Collecting structured data from the crowd. SIGMOD, pages 577–588, 2014.

[84] Kara Pernice, Jakob Nielsen, Susan Farrell, Sachi Mizobuchi, Naoko Ishida, UDIT Amy Stover, Michael Yohay, Elizabeth Franko, and Aimee Richardson. Usability guidelines for accessible web design. *Evidence–Based User Experience Research, Training, Consulting*, 2001.

[85] Alexander J. Quinn and Benjamin B. Bederson. Human-machine hybrid computation. In *In CHI'11 Workshop*, 2011.

[86] Alexander J. Quinn and Benjamin B. Bederson. AskSheet: Efficient human computation for decision making with spreadsheets. CSCW, pages 1456–1466, 2014.

[87] D. Raggett. HTML Tables. RFC 1942 (Historic), May 1996. Obsoleted by RFC 2854.

[88] Loretta Guarino Reid and Andi Snow-Weaver. Wcag 2.0: A web accessibility standard for the evolving web. In *Proceedings of the 2008 International Cross-disciplinary Conference on Web Accessibility (W4A)*, W4A '08, pages 109–115, New York, NY, USA, 2008. ACM.

[89] Jennifer A. Rode, Eleanor F. Toye, and Alan F. Blackwell. The fuzzy felt ethnography—understanding the programming patterns of domes-

tic appliances. *Personal and Ubiquitous Computing*, 8(3):161–176, 2004.

[90] Daisuke Sato, Masatomo Kobayashi, Hironobu Takagi, and Chieko Asakawa. Social Accessibility: The challenge of improving web accessibility through collaboration. W4A, pages 28:1–28:2, 2010.

[91] Daisuke Sato, Masatomo Kobayashi, Hironobu Takagi, and Chieko Asakawa. Social Accessibility: The challenge of improving web accessibility through collaboration. In *W4A*, W4A, pages 28:1–28:2, 2010.

[92] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96–101, 2006.

[93] Amanda Stent, Ann Syrdal, and Taniya Mishra. On the intelligibility of fast synthesized speech for individuals with early-onset blindness. In *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '11, pages 211–218, New York, NY, USA, 2011. ACM.

[94] Petra Sundström, Alex Taylor, Katja Grufberg, Niklas Wirström, Jordi Solsona Belenguer, and Marcus Lundén. Inspirational bits: towards a shared understanding of the digital material. CHI '11, pages 1561–1570, 2011.

[95] Hironobu Takagi, Shin Saito, Kentarou Fukuda, and Chieko Asakawa. Analysis of navigability of Web applications for improving blind usability. *ACM Trans. Comput.-Hum. Interact.*, 14(3):13, 2007.

[96] T M Therneau and P M Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, 2000.

[97] Dustin Tingley, Teppei Yamamoto, Kentaro Hirose, Luke Keele, and Kosuke Imai. mediation: R package for causal mediation analysis. *Journal of Statistical Software*, 59, 2014.

[98] Anthony Tomasic, John Zimmerman, Ian Hargraves, and Roderick McMullen. User Constructed Data Integration via Mixed-Initiative Design. In *Interaction Challenges for Intelligent Assistants*, pages 122–123, 2007.

[99] Michael Toomim, Steven M. Drucker, Mira Dontcheva, Ali Rahimi, Blake Thomson, and James A. Landay. Attaching UI enhancements to websites with end users. In *CHI '09*, pages 1859–1868, New York, NY, USA, 2009. ACM.

[100] Viswanath Venkatesh and Hillol Bala. Technology Acceptance Model 3 and a Research Agenda on Interventions. *Decision Sciences*, 39(2):273–315, May 2008.

[101] Luis von Ahn, Manuel Blum, NicholasJ. Hopper, and John Langford. Captcha: Using hard ai problems for security. In Eli Biham, editor, *Advances in Cryptology âĂŤ EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 294–311. Springer Berlin Heidelberg, 2003.

[102] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 319–326, New York, NY, USA, 2004. ACM.

[103] Luis von Ahn and Laura Dabbish. Designing games with a purpose. *Commun. ACM*, 51(8):58–67, August 2008.

[104] Stephen B. Wicker and Stephanie M. Santoso. Access to the internet is a human right. *Commun. ACM*, 56(6):43–46, June 2013.

[105] Patricia Wright and Kathryn Fox. Presenting information in tables. *Applied Ergonomics*, 1(4):234 – 242, 1970.

[106] Yeliz Yesilada, Robert Stevens, Simon Harper, and Carole Goble. Evaluating dante: Semantic transcoding for visually disabled users. *ACM Trans. Comput.-Hum. Interact*, (14):2007.

[107] Chen Jason Zhang, Ziyuan Zhao, Lei Chen, H. V. Jagadish, and Chen Caleb Cao. CrowdMatcher: Crowd-assisted schema matching. SIGMOD, pages 721–724, 2014.

[108] John Zimmerman, Kathryn Rivard, Ian Hargrave, Anthony Tomasic, and Ken Mohnkern. User-created forms as an effective method of human-agent communication. In *CHI '09*, pages 1869–1878, New York, NY, USA, 2009. ACM.

[109] John Zimmerman, Anthony Tomasic, Isaac Simmons, Ian Hargraves, Ken Mohnkern, Jason Cornwell, and Robert Martin McGuire. VIO: a mixed-initiative approach to learning and automating procedural update tasks. In *CHI*, pages 1445–1454, 2007.

[110] Moshé M. Zloof. QBE/OBE: A Language for Office and Business Automation. *IEEE Computer*, 14(5):13–22, 1981.