

Distributed Transit Rider Messaging

Final Project Report

Steve Gardiner¹, Anthony Tomasic², John Zimmerman³, James Valenti¹, Aaron Steinfeld⁴

¹ Language Technologies Institute

² Institute for Software Research

³ Human Computer Interaction Institute & School of Design

⁴ Robotics Institute, *Project PI*

Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

Point of Contact: steinfeld@cmu.edu, 412-268-6346

December 2013



A U.S. DOT UNIVERSITY TRANSPORTATION CENTER

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the U.S. Department of Transportation's University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

Abstract

Uncertainty, lack of transit system awareness, and feelings of isolation have negative impact on all riders, regardless of abilities, thereby reducing community livability and transit demand. Lower transit demand, in turn, decreases economic competitiveness and environmental sustainability.

The goal of this project is to facilitate information sharing as a means of improving the transit experience of all riders, especially for those who cannot drive. This effort is intended to leverage our existing system, Tiramisu (“pick me up” in Italian), a social-mobile computing system intended to connect riders and transit service providers using universal design. Under other funds, the team is currently advancing Tiramisu by implementing a rider-to-rider and rider-to-agency messaging system to help improve rider and agency awareness of current transit system state. The deployed message system will also allow riders to report situations observed and for the transit agency to push out critical news to riders who may be impacted by an unfolding situation.

Under this UTC project, the team seeks to research and develop methods for gathering support safety related messaging and other information from related, distributed sources for dissemination through the Tiramisu message channel. The intent is also to provide a method for such messages to also be shared with other T-SET UTC systems. While it is straightforward for a programmer to write custom code to extract relevant information from websites, this approach is expensive, brittle, and does not scale beyond a small number of sources.

Therefore, the team has focused on methods to allow non-programmers (e.g., administrative assistants, planners, etc) to train the information gathering system on websites of interest. This will allow rapid information gathering from a wide range of sources, low-cost repair when websites change, and scalability.

Carnegie Mellon University
Copyright 2013

Table of Contents

ABSTRACT	I
THE PROBLEM	1
OVERVIEW	1
BUILDING INFORMATION TABLES FROM DISTRIBUTED WEB SOURCES	1
AUTOMATING TABLE BUILDING	2
APPROACH	3
THE TIRAMISU TESTBED	3
STRUCTURED INFORMATION GATHERING	5
RELATED APPROACHES	7
SMARTWRAP DESIGN PROCESS AND DESIGN	7
PARSING DETOUR ALERTS	14
METHODOLOGY	15
SMARTWRAP USER STUDY	15
FINDINGS	15
USER STUDY RESULTS	15
DISCUSSION	17
CONCLUSIONS	18
RECOMMENDATIONS	18
ACKNOWLEDGEMENTS	19
REFERENCES	19

The Problem

Overview

Uncertainty, lack of transit system awareness, and feelings of isolation have negative impact on all riders, regardless of abilities, thereby reducing community livability and transit demand. Lower transit demand, in turn, decreases economic competitiveness and environmental sustainability. Lack of awareness can be exacerbated during detours, unexpected events, and emergency scenarios. In some cases, it can also decrease safety or expose the rider to severe weather.

The goal of this project is to facilitate information sharing as a means of improving the transit experience of all riders, especially for those who cannot drive. This effort is intended to leverage our existing system, Tiramisu (“pick me up” in Italian), a social-mobile computing system intended to connect riders and transit service providers using universal design. Under other funds, the team is currently advancing Tiramisu by implementing a rider-to-rider and rider-to-agency messaging system to help improve rider and agency awareness of current transit system state. The deployed message system will also allow riders to report situations observed and for the transit agency to push out critical news to riders who may be impacted by an unfolding situation.

Under this UTC project, the team seeks to research and develop methods for gathering situational and safety related information from related, distributed sources for dissemination through the Tiramisu message channel. The intent is also to provide a method for such messages to be shared with other T-SET UTC systems. While it is straightforward for a programmer to write custom code to extract relevant information from websites, this approach is expensive, brittle, and does not scale beyond a small number of sources.

Therefore, the team has focused on methods to allow non-programmers (e.g., administrative assistants, planners, etc) to train the information gathering system on websites of interest. This will allow rapid information gathering from a wide range of sources, low-cost repair when websites change, and scalability.

Building Information Tables from Distributed Web Sources

The web provides access to a large quantity of relational data that users consult frequently for travel decisions. Automated tools, however, generally cannot understand the relational data, making it difficult for users to track, acquire, and integrate data. For example, a transit agency may need to relay information about regular detours to riders due to home sporting events, which may change due to rescheduling or rain delays. Organizations, like transit agencies and planning commissions, typically fill this gap with human users.

Previously, we have found (Zimmerman et al. 2009) that administrative users spend a lot of time manually building tables from web relational data. Given a fixed target schema (model) and lots of data, machine learning models can successfully extract relationships, but users encounter novel or idiosyncrasies which create problems for reusability and generalizability. We believe a better approach is to provide nonprogrammers the ability to teach the system new schemas for the websites they regularly visit and support sharing to other users. This will grow the number of websites that can be processed over time.

Our approach incorporates end users' embrace of spreadsheet tables for relational data (Lin et al. 2009; Zimmerman et al. 2009) and asks the users to perform familiar actions within an instrumented web experience. We attempt to strike a bargain with the user, whereby the user executes only a subset of the selection actions required. From these examples, our system applies programming by demonstration to extrapolate the rest of the table in real-time. The user can review and revise the examples until the table conforms to expectations. Thus the user extracts the table they wanted in the first place with less effort, leaving behind data the system can use to train models. As an added user benefit, if some other user has provided data about a given page, the user can simply download the current table with no interaction.

This approach presents several challenges in terms of both human factors and machine learning. Since our point of departure is that users without technical training typically perform these mundane web scraping tasks, we must ensure that the system is usable by end users without programming training. On the machine learning side, we examine which techniques supply high quality extrapolations while training and executing in time acceptable to a user in the loop. Additionally, we investigate the problem domain to determine which techniques generalize best from partial examples on disparate sites to complete scrapers for unseen sites.

Automating Table Building

Many people often need to repeatedly retrieve information from the internet (Zimmerman et al. 2007). For example, a secretary who is given a list of names of people to invite to a meeting must retrieve their email addresses from a company directory to contact them, and then may additionally retrieve their titles and departmental affiliations from a separate online directory. In performing these tasks, people often look up multiple pieces of information repeatedly within a single database, or they combine information from more than one source (Zimmerman et al. 2009) This work is not hard, but it is very tedious and time consuming.

Researchers have worked to address this retrieval problem through the creation of programming-by-demonstration (PBD) systems that allow users to automate the process of scraping and combining information from various web pages. One example, Mixer, uses a spreadsheet interface where users copy and paste an

example of the information they want in order to train an agent to iteratively perform the remainder of the retrieval task (Gardiner et al. 2011). Testing of Mixer revealed that people with no training in programming could quickly and effectively train an agent to retrieve and join the information they desire. Mixer, however, relies on a wrapper program that describes the structure of each page (e.g., segmenting the text of a directory listing into a name, email address, and phone number). Mixer thus lowers the bar for automating repetitive data retrieval tasks, but leaves end users dependent on programmers to construct high-quality wrappers. Extending to end users the ability to construct their own wrappers fills a gap in the Mixer infrastructure, thereby “raising the ceiling” (Myers et al. 2000) of what end users can do without resorting to programmer assistance.

Many systems leverage wrappers to create datasets for a variety of applications. For example, similar wrapping can help mitigate accessibility issues (Kawanaka et al. 2008). So while the proximate motivation for enabling end user wrapper is to complete the Mixer vision, the wrappers generated have much wider applicability.

The present work extends Mixer by making a new tool we call SmartWrap, which allows end users to quickly and effectively create reusable wrappers for the pages they wish to scrape. Once one user has created a wrapper for a specific page, then any user can use Mixer to retrieve information from that page. SmartWrap, like Mixer, uses Programming By Demonstration (PBD), asking the user to provide examples of the first two rows of a dataset. This will allow rapid extraction of information on the web (e.g., train station elevator outages) for use in other applications, like transit apps.

Approach

The Tiramisu Testbed

The Tiramisu system allows riders to co-produce their city’s transit service in collaboration with the transit agency (Steinfeld et al 2012; Zimmerman et al 2011). Currently, riders leverage the smart phones they carry to generate and share GPS traces of bus locations and fullness ratings (Figure 1). This data is then aggregated in order to provide real-time arrival and fullness information to all riders. The riders collectively generate the information they desire, circumnavigating the need to install and pay maintenance for expensive proprietary systems. In addition, Tiramisu allows riders to report problems with the service and allows the transit service to identify, prioritize, and respond to service delivery needs. Tiramisu can also merge transit agency supplied automatic vehicle location (AVL) feeds with data contributed by the crowd. At this time, we are advancing Tiramisu by implementing a rider-to-rider and rider-to-agency messaging system to help improve rider and agency awareness of current transit system state. This message stream is the target deployment channel for detour, safety, and other information generated by technology developed under this work.

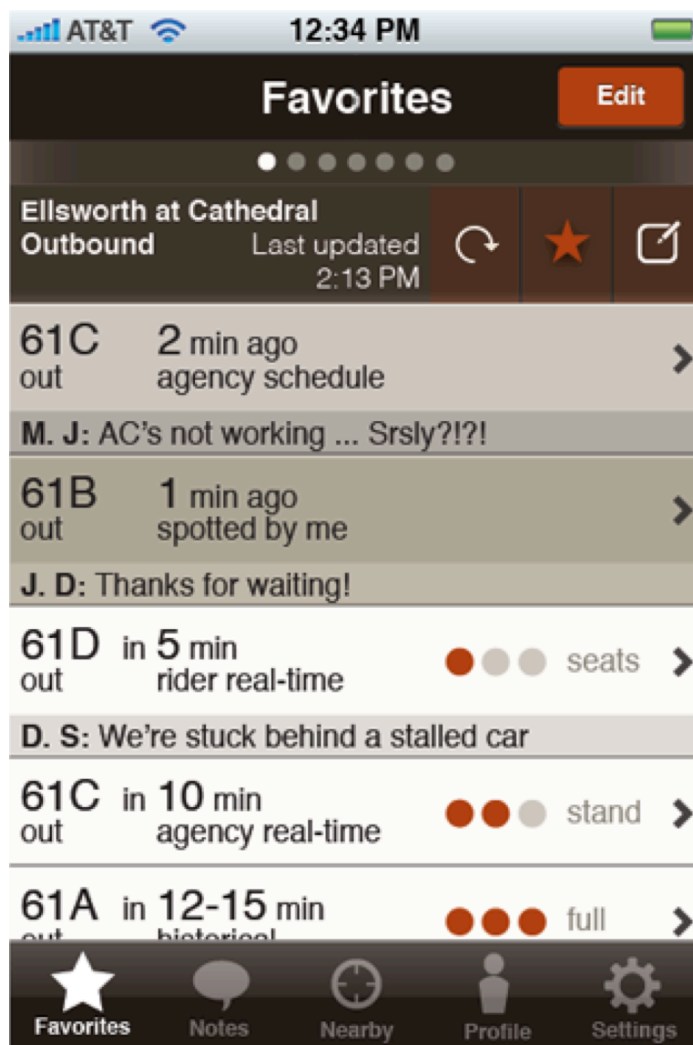


Figure 1: Message interface for the Tiramisu testbed (in development). Riders can share information about vehicle location and fullness. In the future, riders and the transit agency will be able to attach messages to individual trips, routes, and stops.

Tiramisu is designed to support the needs of riders with disabilities, a population that is dependent on transit for access to employment, health care, entertainment, and family. The app was developed using the principles of universal design, which support the incorporation of accessibility features in a manner that are valuable and useful to a broad population. For example, Tiramisu asks users to report vehicle fullness, which directly informs users of wheelchairs and those who need a seat for health or balance reasons. As a result of this focus on the needs of people with disabilities, Tiramisu received the 2012 FCC's Chairman's Award for Advancement in Accessibility in the Geo-Location Solutions category. Chairman Julius Genachowski presented this award to team representatives at a ceremony at the FCC.

Tiramisu's core development has been funded by the Rehabilitation Engineering Research Center on Accessible Public Transit (RERC-APT; www.nercapt.org) and the Traffic21 program. It has been deployed since the summer of 2011 through a spinout company Tiramisu Transit, LLC and is available to the public (www.tiramisutransit.com). The company also received a US DOT SBIR Phase I grant to evaluate revenue models for eventual self-sustainability.

Tiramisu has an installed base of thousands of users, who have contributed roughly 200k trip contributions and opened the application over a million times. The app is available for iPhones and Android, with deployments in Pittsburgh (Port Authority of Allegheny County), New York City (Metropolitan Transportation Authority), and Syracuse (Central New York Regional Transportation Authority).

As a result, the team has a significant installed base for examining technologies developed under UTC funding. The messaging version of Tiramisu is slated to be deployed to the public in early 2014. Aspects of the message stream architecture have been explicitly designed to support data generated by the tools described in this report.

Structured Information Gathering

Most database driven webpages present a sequence of records from an underlying dataset, and each record employs a *schema*, displaying mostly the same fields. For example, a webpage listing books may display in the record for each book a title, an author, and a picture of the cover. When generating a webpage from a database, the site would transform the records into HTML, resulting in the records appearing with a specific layout intended to support the users task. Sites use a wide variety of layouts. In the book example, the site might place cover, title, and author in consecutive cells (Figure 2a), or might place the cover on the left with title and author stacked to the right (Figure 2d).

The wrapper recovers the underlying dataset from the HTML – to reverse the transformation. A wrapper is *reusable* when it can be used to scrape different data presented through the same layout.

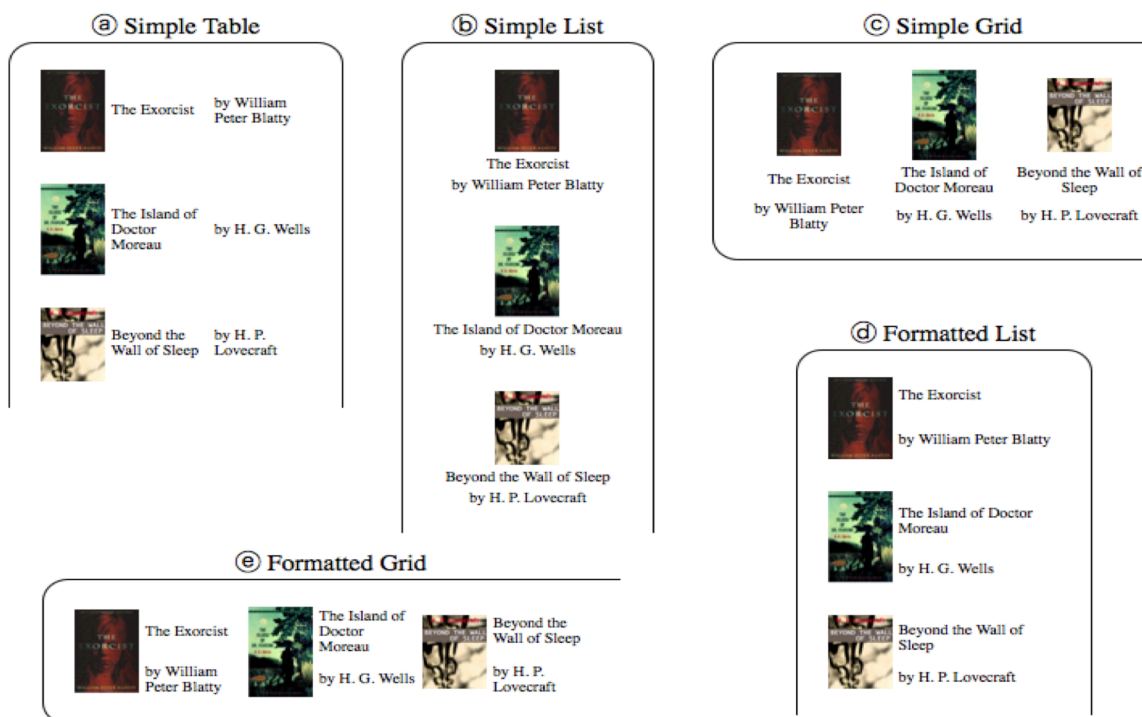


Figure 2: Different visual layouts possible for the same data. Note that while all are frequently presented via the HTML table tag, only (a) adheres to the semantics of the underlying table. Many variations of (d) and (e) are possible. While it is possible to alter layouts for some sites, legacy systems with useful transportation data are often too expensive to modify.

Several phenomena occur with sufficient frequency in real web tables to merit consideration. First, the web page may contain *multiple* datasets, and the user may want to construct a wrapper that extracts more than one of them. More subtly, a web page may contain *nesting*, wherein a record contains sub-records that a user wishes to scrape in the spreadsheet.

Given that many users do, in practice, scrape datasets from web pages into useable spreadsheets (Tomasic et al. 2007), it is clear that they are able to locate the relevant data and to formulate it as a table in the context of constructing a spreadsheet for personal consumption. The tasks, trivial for humans, of identifying and understanding the transformations of datasets presented visually, are known to be beyond the current state of the art for fully automatic systems (see e.g. Venetis et al. 2011). The central question here is whether end users can communicate these transformations to a collaborating software system. The question decomposes into two issues: can users effectively communicate (a) what pieces of data are provided by the schema and (b) where those pieces of data are positioned by the transformation? Conversely, can the software system effectively assist the user to produce a working wrapper?

Related Approaches

Nardi (1993) notes the widespread use of sophisticated forms in human-human interaction, and the surprising facility of nontechnical people in rapidly learning and making use of them. She also observes that users can more readily assimilate new formal representations when they have a preexisting interest or job-related requirement to do so. Rode et al (Rode et al. 2004) note the same phenomenon. They note the similarity between ovens and VCRs in terms of programming. 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. Surprisingly, despite the indistinguishability of the tasks at the abstract level, they found a pronounced gender difference in users' abilities to perform the tasks. Women, who generally exert more control over the kitchen, had more success programming ovens, and conversely, men, who generally exert more control over entertainment devices in the home, had more success programming VCRs. These research results lead us to speculate that one reason office workers have not readily accepted PBD systems is that they cast the task as "programming": a type of work that generally falls outside the common social role description for an administrator. Mixer and SmartWrap address this by specifically disguising the fact that the administrator is programming when interacting with the tool.

Malone et al (1987) 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 (Zimmerman et al. 2007) 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 (1990) build on the work of Lewis and Olson (1987) 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 induce 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 (Norman and Draper 1986) 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.

Our previous Mixer research demonstrated that these advantages of spreadsheets apply to administrators approaching data integration tasks, specifically pointing out the conceptual alignment between user and agent as well as the unified nature of the shared table representation. Several other systems settle on a similar tabular interface between the user and an observing PBD web data integration agent. Vegemite (Lin et al. 2009) asks the user to create a set of “VegeTables,” each of which corresponds to a script for combining two websites. Karma (Tuchinda et al. 2007), Dontcheva et al. (2007), and Mashroom (Wang et al. 2009) build separate tables for each extracted website; additionally, Mashroom explicitly uses nested tables (specifically with an eye towards comprehensibility by end users). Each of these systems asks the user to explicitly “merge” extractions from different websites into a coherent table. In contrast, Mixer and SmartWrap encourage the user to construct the single, unified table that seems to match the user’s underlying conceptualization of the task. This spares the user the confusion inherent in synthesizing, or merging, the results of the various subtasks together. As a consequence, Mixer enables users to construct integration tasks over one website, or over several websites, without necessarily observing the distinction.

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 (“Mixed-Initiative Interaction” 1999; Horvitz 2007). Many issues remain to be answered, including several interrelated needs with respect to interaction between agent assistants and people (Tecuci et al. 2007):

- *Awareness*: knowledge of problem and goal must be shared by human and software tool.
- *Task*: roles and responsibilities must be shared between human and tool.
- *Communication*: both human and tool 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. 2004) observe that users are far less successful in performing programming tasks outside their perceived area of responsibility. Consequently, Mixer explicitly attempts to avoid presenting the user with tasks that feel like programming.

The task issue concerns the division of action between humans and software. The principal actions of a PBD session (Kosbie and Myers 1993) are program demonstration (or creation), program invocation, and program execution. Mixer incrementally constructs a program by observing all actions taken within the browser, from the time that the user invokes Mixer to the time that the user presses

a button to invoke the demonstrated program. Mixer then executes the program. Thus Mixer presents a strong distinction between user actions (before invocation) and system actions (after invocation). This separation of activity is stricter in Mixer than in some PBD systems, such as Eager (Cypher 1993), which assist the user in deciding when to invoke the observed behavior.

The communication issue arises in a couple of ways from what Cypher (2010) 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 (Lau et al. 2003). PLOW (Allen et al. 2007) receives richer input from the user by eliciting and utilizing natural language explanations for the user's actions. Wrangler (Kandel et al. 2011) asks the user to select after each action the statement in the implementation language corresponding to the level of generalization required. Rather than eliciting additional input from the user, Mixer overcomes the problem by exploiting rather strong simplifying assumptions about the types of problems Mixer is expected to solve.

The user has the responsibility to demonstrate their knowledge of a single row of the table, and Mixer assumes full responsibility for inferring the best possible procedure from that demonstration. Although the user need not understand the workings (or even the existence) of the program, the user does need to be aware that the agent is observing; in other words, the user is expected to take an "intentional stance" (Dennett 1987; Maulsby and Witten 1993) with respect to showing Mixer how to perform the desired task. Mixer asks the user to intentionally demonstrate similar information to that detected automatically by TX2 (Bigham et al. 2009).

As to the second challenge, (Modugno and Myers 1993) further delineate the communication role played by the program in PBD systems, as a list of opportunities presented to the user:

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

Mixer provides limited information about the inferred program through the intermediate depiction of the workspace, giving the user implicit confirmation responsibility as well as some ability to correct unexpected columns in the workspace.

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 (Bolin 2005) records web actions as general JavaScript. CoScripter (Leshed et al. 2008) chooses a slightly more user-friendly approach, representing the program in a “sloppy” or natural programming language. Query-by-Example and Office-by-Example (Zloof 1981) utilize a form as a shared communication structure, but require a user to understand and specify programmatic variable structure within the forms. Mixer uses a single nested table form as the principal communication medium between the human and the agent, which diminishes the variety of programs Mixer can produce but dramatically simplifies the user’s interaction with the system.

Over the last few years there has been a great amount of research interest in streamlining the process of creating web mashups (Beemer and Gregg 2009). By focusing on ad hoc reports rather than mashups (i.e. the output rather than the program), Mixer differs philosophically from many mashup projects; in particular, Mixer aims to allow users to conceptualize data integration problems uniformly, whether or not some pieces of information lie across web server boundaries. Whereas mashup systems emphasize reusability and generality, Mixer focuses on how administrators can retrieve and integrate the types of data they need for their jobs.

Nevertheless, Mixer shares some overlap with mashup systems in that Mixer presents a user-friendly solution to the source modeling and data integration problems, with particular attention to the database joins. Thus, Mixer could coexist in a mashup ecosystem with user-appropriate solutions to wrapper generation (e.g. reform (Toomim et al. 2009) or the summaries of Dontcheva et al (2006)) or data cleaning (e.g. Potters Wheel (Raman and Hellerstein 2001) or Potluck (Huynh et al. 2007)). Mash Maker (Ennals et al. 2007) provides a representative mashup ecosystem, distinguishing between end user-specified wrappers and developer-provided widgets, which combine and visualize the wrapped data. In this perspective, Mixer presents a mechanism for nonprogrammers to create useful widgets without developer intervention.

SmartWrap Design Process and Design

Our goal was to develop a tool that allows end-users to show the tool the idiosyncratic pattern of a specific page and the underlying dataset, resulting in a reusable wrapper. The tool needed to make wrapper construction both faster and cheaper than hiring a programmer or having users write their own program.

Our goals led us to two requirements. First, we needed a tool that could be used by people not formally trained in programming. The tool works as part of a network for sharing and running wrappers contributed by the crowd. In this context we expect a subset of “gardeners” (Nardi 1993) to either self-select themselves from the crowd and opportunistically take on wrapper construction or to be assigned to their role as

part of their job duties. 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.

We implemented the combination of SmartWrap (with Mixer) as a Firefox extension. The tool can analyze contents of visited pages while the tool is active. When the tool is not active, it does not observe anything; the user must explicitly activate the tool to begin the process of creating a wrapper.

When the user encounters a page they wish to wrap, they press a button in the browser. The tool then opens a sidebar to the right of the target window showing a spreadsheet-like interface ready to be filled with data (Figure 3). The left pane displays and maintains an unaltered view of the target, allowing the user to maintain an overview of both the page being wrapped and the state of the wrapper construction. We selected a spreadsheet table metaphor due to previous success with this end-user programming metaphor. In addition, the outline of the spreadsheet shows a blank first and second row for a single column, helping to communicate the user's task of constructing the first two rows of a table to communicate the dataset to the agent.

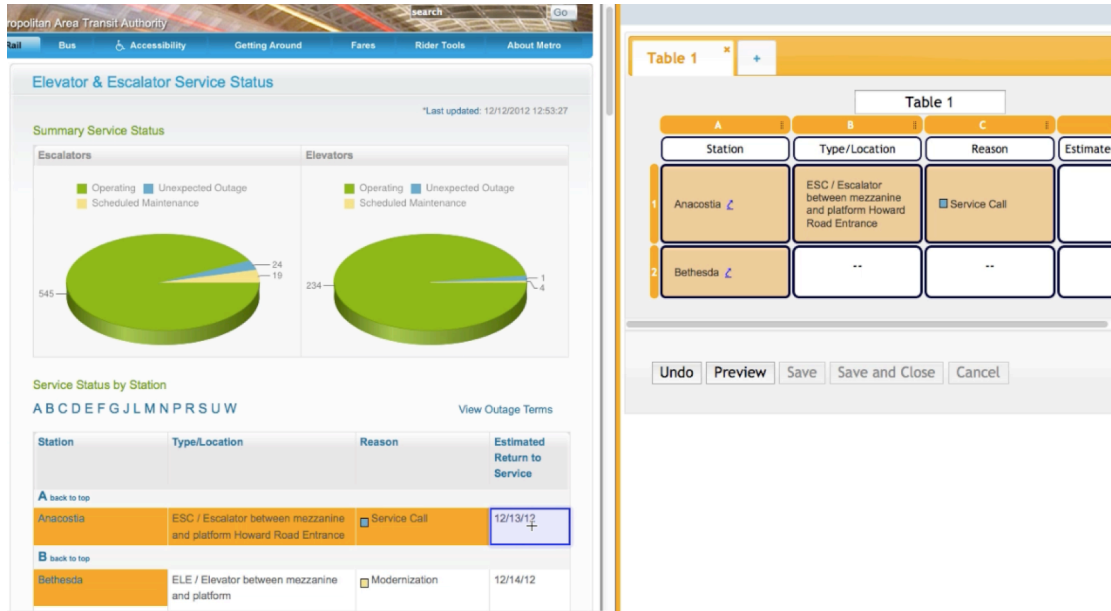


Figure 3: The design places a spreadsheet interface to the right of the original webpage. The user maneuvers the blue box (a) to locate the appropriate information and then drags it into the appropriate box (b) of the spreadsheet, working in any order. The information already selected is indicated by the contents of 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 scraper.

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 new 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.

Using the same drag-and-drop procedure, the user populates the remaining fields into the spreadsheet. When the column headers and table name are explicitly represented in the webpage, they can be dragged using the same mechanism. When they are not available, the user may manually type them in. The tool does not require the user to fill the first row before providing the second; the user is free to work in whatever order they like. This freedom allows the user to construct the table in an exploratory manner, simply trying out different alternatives of the spreadsheet until they are satisfied. The tool allows for backing out of rejected alternatives, or simply mistakes, by allowing the user to remove the contents of any individual cell in the spreadsheet; an Undo button removes the cell most recently added, allowing the user to back up through previous spreadsheet states.

Station	Type/Location	Reason	Estimated Return to Service
Anacostia	ESC / Escalator between mezzanine and platform Howard Road Entrance	<input checked="" type="checkbox"/>	12/13/12
Bethesda	ELE / Elevator between mezzanine and platform	<input type="checkbox"/>	12/14/12
Bethesda	ELE / Elevator between street and mezzanine, corner of Wisconsin Ave. and Montgomery Lane	<input type="checkbox"/>	12/14/12
Bethesda	ESC / Escalator between bus terminal and mezzanine	<input checked="" type="checkbox"/>	12/26/12
Cheverly	ESC / Escalator between mezzanine and platform to New Carrollton	<input type="checkbox"/>	12/14/12
Columbia Heights	ESC / Escalator between street and mezzanine, 14th Street Northbound Entrance	<input checked="" type="checkbox"/>	12/18/12
Columbia Heights	ESC / Escalator between street and mezzanine, 14th Street Southbound Entrance	<input checked="" type="checkbox"/>	12/18/12
Congress Heights	ESC / Escalator between mezzanine and buses/parking	<input type="checkbox"/>	12/13/12
Court House	ESC / Escalator between street (Wilson Blvd) and middle landing/tunnel	<input checked="" type="checkbox"/>	12/14/12
Crystal City	ESC / Escalator between mezzanine and platform to Huntington/Franconia-Springfield	<input checked="" type="checkbox"/>	12/14/12
Dupont Circle	ELE / Elevator between mezzanine and platform to Shady Grove Q Street Entrance	<input checked="" type="checkbox"/>	12/14/12
Dupont Circle	ESC / Escalator between street and mezzanine Q Street Entrance	<input type="checkbox"/>	4/9/13

Figure 4: When the user has filled in two rows of the desired spreadsheet and pressed the Preview button the system responds by extrapolating the scraper for the webpage, applying the scraper to scrape the current page, and presenting the output of the scraper to the user. The user may then examine that the output matches what they expect.

When the user is satisfied that the two rows of the spreadsheet correspond accurately with two of the rows of the underlying table, they initiate the construction of the wrapper for the webpage by selecting the “Preview” button. In response, the tool sends the examples to a server, which extrapolates a wrapper for the page. The tool immediately runs the wrapper on the webpage and extracts the remaining rows of the underlying table. It then displays the extracted result in a new tab in the left-hand pane of the browser window (Figure 4). The user can easily check that the output of the wrapper visually matches the examples in the demonstrated table. Additionally, all of the fields of the dataset are highlighted in the original webpage, so the user can switch to that tab 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. This uploads the constructed wrapper to a central server and makes it available to anyone wishing to use the demonstrated data from this page.

If the output of the wrapper is not correct, the user can drag more information to replace or augment the information in the example cells, and try again. Alternately, the user can discard all or part of the present table and construct the table anew. The extracted tables are free of HTML markup and other aesthetic details, thus making them easy to integrate into other software.

Parsing Detour Alerts

Another challenge for automated data gathering is handling natural language text, especially for content tied to key transit objects, like routes and stops. Figure 5 is an example detour notice for the Port Authority. None of this text is constructed or marked up for easy machine understanding. Our colleagues from the Language Technologies Institute on the NSF-funded *Let's Go!* project run an automated, phone-based transit information system for Pittsburgh. This spoken dialog system allows users to call in and request transit information using natural speech, rather than phone trees and constrained spoken commands. The Tiramisu team supplies real-time information for their system.

The *Let's Go!* team has developed natural language processing techniques for converting Port Authority detour notices into a machine usable form. Under the UTC effort and in parallel with the SmartWrap effort, they have prepared and implemented a method for passing this machine usable detour information to Tiramisu. This information will be inserted into the message system for rider use.



DETOUR

Port Authority publicizes its detours in advance of the occurrence. Numerous factors can come into play at the time of the actual event. As such, Port Authority cannot guarantee that the information provided below will be 100% accurate.

*** Note:** If a route is listed in this route summary section, but not listed below in the detour routing details, then there are no new discontinued or established service stops. In this case, the detour will be a delay rather than a change in the bus routing.

DETOUR NOTICE 13-442-0 12/16/2013

Temporary Stop Change for Routes

59 - Mon Valley	61B - Braddock-Swissvale
------------------------	---------------------------------

Temporary Mid-Term Closure Of Bus Stops On Corey Street at Braddock Avenue & Braddock Ave at Fourth

Effective: 10/27/2013 to 7/31/2014
Daily
Effective Sunday, October 27, 2013 For Approximately Nine (9) Months

Mistick Construction and Costa Construction Companies have notified Port Authority that effective Monday, October 28, 2013 the property area of a major portion of the former Braddock Hospital site will be further deconstructed, including all sidewalk areas to allow for property improvements to occur for eventual commercial and retail development on the site. The construction work for safety reasons will require the closure of the sidewalks around the property, bracketed by portions of Corey Avenue and Braddock Avenue within Braddock, Pennsylvania. The fencing of the sidewalks for approximately nine (9) months will temporarily eliminate the 61B bus stop on Braddock Avenue at Fifth Street and the outbound 59 Mon Valley stop on Corey at Braddock. Passengers waiting for 61B trips can still board buses on Braddock Avenue and Fourth Street; 59 Mon Valley passengers traveling towards North Versailles and points outbound can board the buses on the opposite of Braddock Avenue at a temporary stop on Sixth Street.

Figure 5: Example detour announcement from the Port Authority of Allegheny County.

Methodology

SmartWrap User Study

In order to validate our approach, we conducted an empirical study to see if our technology would allow users with limited computer expertise to create reusable wrappers.

We manually selected a subset of webpages to wrap containing a broad selection of the design-table patterns. We expected that simple tables would be easy – indeed trivial – to transfer into spreadsheets, so we selected several simple tables 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 than tasks 1 and 2. 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 for money or other reward. The website requires participants to be over 18 for reasons of consent, and this was our only exclusion criterion. Each participant took a brief survey [6] 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 chose to use training videos as the emergence of YouTube has made this a common way for people to learn how to perform specific software tasks. We asked participants to proceed through the tasks 1 through 11 sequentially, allowing them to access the training videos as needed.

While performing the tasks, we asked them to “think aloud” in order to capture their understanding of both the task and their sense of how the tool worked. We recorded all screen actions and narratives for subsequent analysis. 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.

Findings

User Study Results

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. Throughout the following analysis, we denote this latter group as programmers, and the remainder as nonprogrammers. Of the 12 programmers, 7 reported having used advanced programming techniques, such as 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 the data into the spreadsheet interface. The remaining 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% were completed in less than 10 minutes. Summary findings are presented in Table 1.

On average, each participant successfully completed 8 tasks, with a standard deviation of 3 tasks. Two participants, both nonprogrammers, spent 35 minutes on 4 and 3 tasks respectively without completing any tasks; these two participants are removed from the subsequent quantitative analyses as outliers. Programmers were more successful overall than nonprogrammers: 83% of tasks attempted by nonprogrammers were successful versus 97% of tasks attempted by programmers. The increased success rate of programmers was particularly marked in the tasks involving multiple tables and nesting. The baseline improvement for being a programmer was 14% (97% - 83%). 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 implies programmers had better performance.

Some participants encountered difficulty with dragging and navigation, especially during Task 4. Also, some column heads were undraggable, so participants eventually typed the undraggable column headings' content into the spreadsheet. Other problems included replacing content of an occupied cell, uncertainty about SmartWrap's ability to understand dragged images (it can), and accurately dropping dragged content into a cell.

Table 1: 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.

Task	Success	Time
T1: simple	90%	3:43 ± 3:45
T2: simple	92%	4:18 ± 4:44
T3: simple	92%	3:23 ± 1:13
T4: simple	60%	6:05 ± 3:55
T5: list	90%	2:37 ± 1:49
T6: list	94%	4:26 ± 2:25
T7: list	94%	3:16 ± 2:04
T8: grid	87%	4:12 ± 3:44
T9: multi	80%	5:29 ± 4:17
T10: nested	73%	5:39 ± 2:37
T11: nested	43%	2:51 ± 1:30

On Tasks 10 and 11, containing nested data, 60% of participants constructed on the first attempt a spreadsheet repeating the nesting field. 80% constructed this variant eventually, and were counted as correct for the task in the analysis above. This is a promising result for a complex action on their first exposure to the system.

Twenty-two (22) participants commented on the “magic pixel” behavior of the blue box as it moved around the page, usually as they were trying to determine the exact spot to outline the content they wanted.

User remarks clearly showed a need for specific improvements in the user interaction. However, there was clear evidence of appreciation for the tool. Ten participants volunteered praise for the tool in the think aloud. Five participants offered general praise for the tool, calling it “real cool” or “pretty neat.” Two participants stated that they would envision using the tool in their daily lives.

Discussion

The overall success rates in Table was promising, suggesting that most people will be able to use the tool to position the parts of the dataset in the spreadsheet interface. Two participants failed to complete any tasks, reinforcing the notion that not every end user has the cognitive skills to participate in wrapper construction. The above findings, however, indicate that programming training is not a necessary prerequisite for constructing wrapper. Programmers do have a higher rate of success.

The overall short amounts of time spent on each task was also encouraging. Extrapolating from the data suggests that about 95% of web pages containing datasets could be wrapped in under 10 minutes. The task of programming a wrapper by hand for a web page, like any nontrivial programming task, is difficult to complete in such a short time. As an additional comparison, manually scraping the task pages into a spreadsheet is likely to take longer too.

There is also a need to iterate on the user interaction design, thereby incorporating lessons drawn from the participants experiences to make SmartWrap easier to teach. In addition to reducing frustration, this will increase success rates and decrease task completion time.

There is also a need to better articulate what SmartWrap is able to accomplish. Improved training videos are a logical first step. Many users, however, will not use or attend to available documentation. Another approach to encouraging complex actions is to have the tool ask the user whether they want certain actions to occur when the user comes close to executing a complex action, making it clear that the tool supports such functionality.

Conclusions

We have shown that approaches like SmartWrap have the potential to rapidly accelerate the gathering of information from the web for use in applications like Tiramisu. The study 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 training in programming can do so. The issue of whether this approach can be extended to support real-life tasks in transit and other transportation applications is a separate issue which would require further study. We hope to demonstrate this by passing extracted information to users through an eventual release of Tiramisu.

Recommendations

In an ideal world, information sources will provide relevant open data to third parties through standardized, software friendly formats (GeoAccess 2011). Likewise, web tables would be formatted and populated using standard techniques with appropriate HTML markup, thus limiting the need for custom web scraping tools and approaches like SmartWrap. Besides the positive impact better HTML practices would have on the retrieval and scraping of data, this would also increase the accessibility of web content for users who are blind and use a screen reader for web surfing.

Unfortunately, we do not have high expectations that best practices will be adopted and maintained. This is based on the widespread lack of adherence to current standards, even for basic web standards like HTML. There is also an ever-growing number of legacy webpages that will not be altered to the extreme cost of modifying existing software. This problem is especially prevalent for cash-strapped transit agencies. Therefore, it is important to develop and deploy solutions like SmartWrap to bridge this gap.

Another important step would be for information about non-standard scenarios (e.g., detour notices, closures, etc) to be provided in machine-friendly structures, not just in narrative text form. This would limit the need for natural language processing and encourage dissemination of this important information to riders by third party app developers. Some agencies are moving in this direction and the new GTFS-Realtime standard includes methods for sharing this kind of data (<https://developers.google.com/transit/gtfs-realtime/>).

Acknowledgements

Additional funds for SmartWrap were supplied by the Center for the Future of Work at Carnegie Mellon University.

The core Tiramisu work is funded by the Rehabilitation Engineering Research Center on Accessible Public Transportation (RERC-APT). The RERC-APT is funded by grant number H133E080019 from the United States Department of Education through the National Institute on Disability and Rehabilitation Research. Additional support for Tiramisu was provided by Traffic21 at Carnegie Mellon University, a program developed with the support of the Hillman Foundation, and a US Department of Transportation SBIR Phase I grant (DTRT57-12-C-10039).

References

- Allen, James, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, and William Taysom. 2007. "PLOW: A Collaborative Task Learning Agent." In *In Proc. Conference on Artificial Intelligence (AAAI)*, 22–26. Springer-Verlag.
- Beemer, Brandon, and Dawn Gregg. 2009. "Mashups: A Literature Review and Classification Framework." *Future Internet* 1 (1): 59–87. doi:10.3390/fi1010059. <http://www.mdpi.com/1999-5903/1/1/59>.
- Bigham, Jeffrey P, Ryan S Kaminsky, and Jeffrey Nichols. 2009. "Mining Web Interactions to Automatically Create Mash-Ups." In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, 203–

212. New York, NY, USA: ACM. doi:10.1145/1622176.1622215.
<http://doi.acm.org/10.1145/1622176.1622215>.
- Bolin, Michael. 2005. "End-User Programming for the Web". MIT.
<http://groups.csail.mit.edu/uid/projects/chickenfoot/mbolin-thesis.pdf>.
- Cypher, Allan. 2010. "End User Programming on the Web." In *No Code Required: Giving Users Tools to Transform the Web*, edited by Allan Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols, 3–22. Burlington, MA, USA: Morgan Kaufmann.
- Cypher, Allen. 1993. "Watch What I Do." In , edited by Allen Cypher, Daniel C Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A Myers, and Alan Turransky, 205–217. Cambridge, MA, USA: MIT Press.
<http://dl.acm.org/citation.cfm?id=168080.168111>.
- Dennett, Daniel C. 1987. *The Intentional Stance* (Bradford Books). Cambridge, MA: The MIT Press.
<http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262540533>.
- Dontcheva, Mira, Steven M Drucker, David Salesin, and Michael F Cohen. 2007. "Relations, Cards, and Search Templates: User-Guided Web Data Integration and Layout." In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 61–70. New York, NY, USA: ACM. doi:10.1145/1294211.1294224.
<http://doi.acm.org/10.1145/1294211.1294224>.
- Dontcheva, Mira, Steven M Drucker, Geraldine Wade, David Salesin, and Michael F Cohen. 2006. "Summarizing Personal Web Browsing Sessions." In *UIST*, 115–124. New York, NY, USA. doi:10.1145/1166253.1166273.
<http://doi.acm.org/10.1145/1166253.1166273>.
- Ennals, Rob, Eric Brewer, Minos Garofalakis, Michael Shadle, and Prashant Gandhi. 2007. "Intel Mash Maker: Join the Web." *SIGMOD Rec.* 36 (4) (December): 27–33. doi:10.1145/1361348.1361355.
<http://doi.acm.org/10.1145/1361348.1361355>.
- Gardiner, S., Tomasic, A., Zimmerman, J., Aziz, R., and Rivard, K. 2011. "Mixer: mixed-initiative data retrieval and integration by example." *Interact*, Springer-Verlag.
- GeoAccess Challenge Team (2011). *Data-Enabled Travel: How Geo-Data Can Support Inclusive Transportation, Tourism, and Navigation through Communities*, <http://geoaccess.org/content/report-data-enabled-travel>

- Horvitz, Eric. 2007. "Reflections on Challenges and Promises of Mixed-Initiative Interaction." *AI Magazine* 28 (2).
<http://www.aaai.org/ojs/index.php/aimagazine/article/view/2036>.
- Huynh, David F, Robert C Miller, and David R Karger. 2007. "Potluck: Data Mash-up Tool for Casual Users." In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, 239–252. Berlin, Heidelberg: Springer-Verlag.
<http://dl.acm.org/citation.cfm?id=1785162.1785181>.
- Kandel, Sean, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. "Wrangler: Interactive Visual Specification of Data Transformation Scripts." In *CHI*, 3363–3372. New York, NY, USA. doi:10.1145/1978942.1979444.
<http://doi.acm.org/10.1145/1978942.1979444>.
- Kawanaka, Shinya, Yevgen Borodin, Jeffrey P Bigham, Darren Lunn, Hironobu Takagi, and Chieko Asakawa. 2008. "Accessibility Commons: a Metadata Infrastructure for Web Accessibility." In *SIGACCESS*. Halifax, Nova Scotia, Canada: ACM. doi:10.1145/1414471.1414500.
<http://dl.acm.org/citation.cfm?id=1414500>.
- Kosbie, David S, and Brad A Myers. 1993. "PBD Invocation Techniques: A Review and Proposal." In *Watch What I Do: Programming by Demonstration*, edited by Allan Cypher, 415–422. Cambridge, MA, USA: MIT Press.
- Lau, Tessa, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. 2003. "Programming by Demonstration Using Version Space Algebra." *Mach. Learn.* 53 (1-2): 111–156. doi:http://dx.doi.org/10.1023/A:1025671410623.
- Leshed, Gilly, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. "CoScripter: Automating & Sharing How-to Knowledge in the Enterprise." In *CHI*, 1719–1728. New York, NY, USA: ACM. doi:10.1145/1357054.1357323.
<http://doi.acm.org/10.1145/1357054.1357323>.
- Lewis, Clayton, and Gary Olson. 1987. "Empirical Studies of Programmers: Second Workshop." In , edited by Gary M Olson, Sylvia Sheppard, and Elliot Soloway, 248–263. Norwood, NJ, USA: Ablex Publishing Corp.
<http://dl.acm.org/citation.cfm?id=54968.54984>.
- Lin, James, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A Lau. 2009. "End-User Programming of Mashups with Vegemite." In *IUI*. Sanibel Island, Florida, USA: ACM. doi:10.1145/1502650.1502667.
http://delivery.acm.org/10.1145/1510000/1502667/p97-lin.pdf?ip=128.237.197.103&id=1502667&acc=ACTIVE SERVICE&key=C2716FEBFA981EF1D8ED4F16102DA82BADDD11EBB600FF97&CFID=244267921&CFTOKEN=56849953&_acm_=1378849687_643d295cde88917143b901eb86486583.

- Malone, Thomas W, Kenneth R Grant, Kum-Yew Lai, Ramana Rao, and David Rosenblitt. 1987. "Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination." *ACM Trans. Inf. Syst.* 5 (2): 115–131. doi:10.1145/27636.27637. <http://dl.acm.org/citation.cfm?id=27636.27637>.
- Maulsby, David, and Ian H Witten. 1993. "Metamouse: An Instructible Agent for Programming by Demonstration." In *Watch What I Do: Programming by Demonstration*, edited by Allan Cypher, 155–181. Cambridge, MA, USA: MIT Press.
- "Mixed-Initiative Interaction." 1999. *IEEE Intelligent Systems* 14: 14–23. doi:<http://doi.ieeecomputersociety.org/10.1109/MIS.1999.10022>.
- Modugno, Francesmary, and Brad Myers. 1993. "Graphical Representation and Feedback in a PBD System." In *Watch What I Do: Programming by Demonstration*, edited by Allan Cypher, 415–422. Cambridge, MA, USA: MIT Press.
- Myers, B., Hudson, S.E., and Pausch, R. 2000. "Past, present, and future of user interface software tools." *ACM Trans. Comput.-Hum. Interact.* 7(1), 3–28.
- Nardi, Bonnie A. 1993. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press.
- Nardi, Bonnie A, and James R Miller. 1990. "The Spreadsheet Interface: A Basis for End User Programming." *Proceedings of the IFIP TC13 Third International Conference on Human-Computer Interaction*. North-Holland Publishing Co.
- Norman, D A, and S W Draper. 1986. "User Centered System Design: New Perspectives on Human-Computer Interaction." In , 31–61. Lawrence Erlbaum Associates. <http://books.google.com/books?id=IR9FAAAAYAAJ>.
- Raman, Vijayshankar, and Joseph M Hellerstein. 2001. "Potter's Wheel: An Interactive Data Cleaning System." In *VLDB*, 381–390. San Francisco, CA, USA. <http://dl.acm.org/citation.cfm?id=645927.672045>.
- Rode, Jennifer A, Eleanor F Toye, and Alan F Blackwell. 2004. "The Fuzzy Felt Ethnography—understanding the Programming Patterns of Domestic Appliances." *Personal Ubiquitous Comput.* 8 (3-4): 161–176. doi:10.1007/s00779-004-0272-0. <http://link.springer.com/content/pdf/10.1007/s00779-004-0272-0.pdf>.
- Steinfeld, A., Zimmerman, J., Tomasic, A., Yoo, A., and Aziz, R. 2012. "Mobile transit rider information via universal design and crowdsourcing." *Transportation Research Record - Journal of the Transportation Research Board* 2217, 95–102.

- Tecuci, Gheorghe, Mihai Boicu, and Michael Cox. 2007. "Seven Aspects of Mixed-Initiative Reasoning: An Introduction to This Special Issue on Mixed-Initiative Assistants." *AI Magazine* 28 (2).
<http://www.aaai.org/ojs/index.php/aimagazine/article/view/2035>.
- Tomasic, Anthony, John Zimmerman, Ian Hargraves, and Roderick McMullen. 2007. "User Constructed Data Integration via Mixed-Initiative Design." In *AAAI Spring Symposium*, 122–123.
- Toomim, Michael, Steven M Drucker, Mira Dontcheva, Ali Rahimi, Blake Thomson, and James A Landay. 2009. "Attaching UI Enhancements to Websites with End Users." In *CHI*, 1859–1868. New York, NY, USA.
doi:10.1145/1518701.1518987.
<http://doi.acm.org/10.1145/1518701.1518987>.
- Tuchinda, Rattapoom, Pedro Szekely, and Craig A Knoblock. 2007. "Building Data Integration Queries by Demonstration." In *IUI '07: Proceedings of the 12th International Conference on Intelligent User Interfaces*, 170–179. New York, NY, USA: ACM. doi:<http://doi.acm.org/10.1145/1216295.1216328>.
- Venetis, P., Halevy, A., Madhavan, J. et al. 2011. "Recovering semantics of tables on the web." *Proc. VLDB Endow.* 4(9), 528–538.
- Wang, Guiling, Shaohua Yang, and Yanbo Han. 2009. "Mashroom: End-User Mashup Programming Using Nested Tables." In *Proceedings of the 18th International Conference on World Wide Web*, 861–870. New York, NY, USA: ACM.
doi:10.1145/1526709.1526825.
<http://doi.acm.org/10.1145/1526709.1526825>.
- Zimmerman, John, Kathryn Rivard, Ian Hargraves, Anthony Tomasic, and Ken Mohnkern. 2009. "User-Created Forms as an Effective Method of Human-Agent Communication." In *CHI*, 1869–1878. New York, NY, USA.
doi:10.1145/1518701.1518988.
<http://doi.acm.org/10.1145/1518701.1518988>.
- Zimmerman, J., Tomasic, A., Garrod, C., Yoo, D., Hiruncharoenvate, C., Aziz, R., Thirunevgadam, N., Huang, Y., and Steinfeld, A. 2011. "Field trial of Tiramisu: Crowd-sourcing bus arrival times to spur co-design," *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*.
- Zimmerman, John, Anthony Tomasic, Isaac Simmons, Ian Hargraves, Ken Mohnkern, Jason Cornwell, and Robert Martin McGuire. 2007. "VIO: a Mixed-Initiative Approach to Learning and Automating Procedural Update Tasks." In *CHI*, 1445–1454.
- Zloof, Moshé M. 1981. "QBE/OBE: A Language for Office and Business Automation." *IEEE Computer* 14 (5): 13–22.