Mixer: Mixed-Initiative Data Retrieval and Integration by Example

Steven Gardiner, Anthony Tomasic, John Zimmerman, Rafae Aziz, Kathryn Rivard

Mixer: Mixed-initiative Data Retrieval and Integration by Example

Steven Gardiner, Anthony Tomasic, John Zimmerman, Rafae Aziz, and Kathryn Rivard

Carnegie Mellon School of Computer Science, 5000 Forbes Ave Pittsburgh PA 15213
{sgardine, tomasic, johnz, raziz, krivard}@cs.cmu.edu

Abstract. Office administrators are frequently asked to create ad hoc reports based on web accessible data. The web contains the desired data but does not allow efficient access in the way the administrator needs, prompting a tedious and labor-intensive task of retrieving and integrating the required data. Mixer is a programming-by-demonstration (PBD) tool empowering administrators to construct ad hoc reports from diverse web sources without tedious piecemeal labor. Mixer's design builds on the exploration into end user conceptualization of data retrieval tasks from our previous Wizard-of-Oz study [39], and incorporates insights from mixed-initiative researchers into collaboration between end users and software agents. This paper justifies the design decisions that drive Mixer, focusing on general lessons for designers of programming-by-demonstration systems targeting nonprogrammers. We evaluate Mixer by performing a user study showing that administrators are able to accomplish programming tasks without needing to understand programming concepts for data retrieval and integration.

Keywords: programming by demonstration, end user programming, mixed initiative, data integration

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 [34]. 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 useable 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 [20, 22]. 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 paper 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.

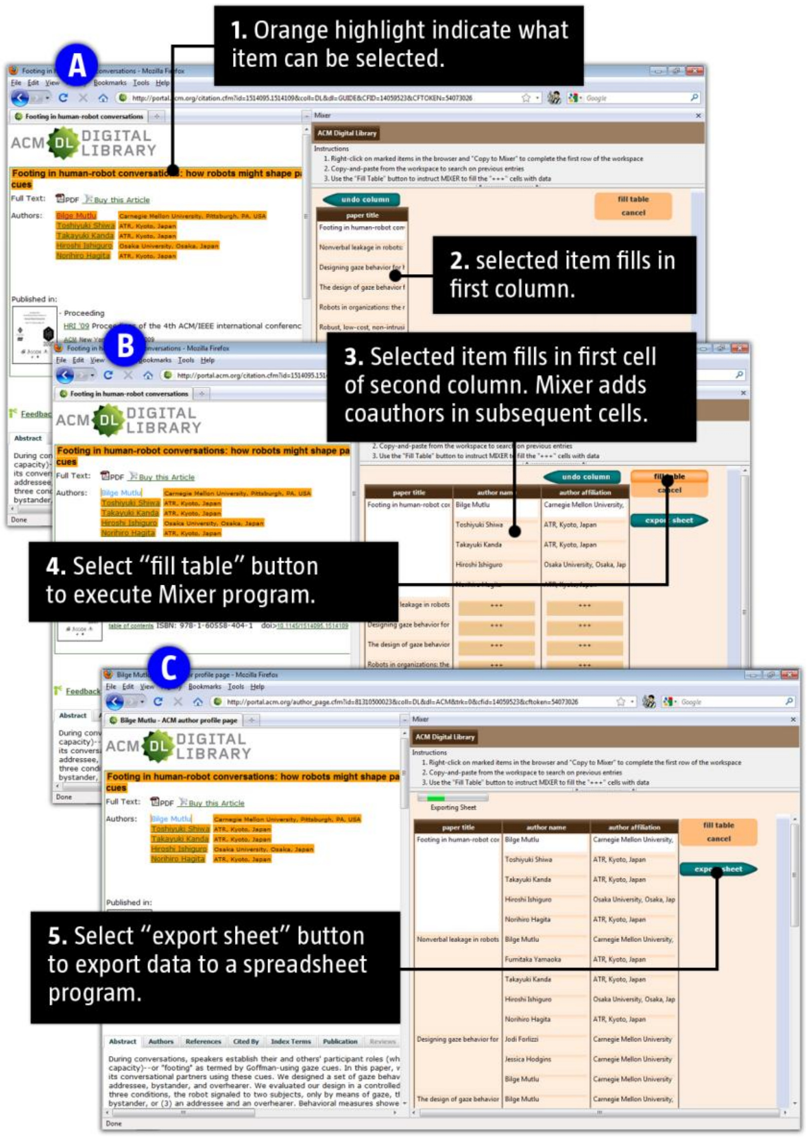
A
1. Orange highlight indicate what item can be selected.
2. selected item fills in first column.
B
3. Selected item fills in first cell of second column. Mixer adds coauthors in subsequent cells.
4. Select "fill table" button to execute Mixer program.
C
5. Select "export sheet" button to export data to a spreadsheet program.
undo column
fill table
cancel
export sheet
paper title
author name
author affiliation

Figure 1: A user interacting with Mixer to extract the coauthors of a researcher based on the researcher's papers on the ACM Digital Library

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 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 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 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 [26].

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 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 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 illuminated the way for the present implementation; however, as noted by Sundström et al [32], 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.

2.1 Wrappers

Mixer augments the target page with orange highlights to indicate selectable elements. The agent performs this augmentation by applying a wrapper to the page. A wrapper is a small piece of code that identifies the types and location of data within the page. Each time a new web page is visited, Mixer consults a database of wrappers, selects the most appropriate wrapper for the page, and applies the wrapper to the page. Based on the application of the wrapper to the page, Mixer highlights the wrapped data on the page, as shown in Figure 1, and adds the appropriate user interaction bindings. The use of wrappers represents a departure from the previous Mixer interface. The wrapper directly encodes the hierarchical relationship among the potential columns, allowing the agent to automatically fill in the first column as soon as the first element has been selected.

An obvious bottleneck for the applicability of Mixer is the coverage of its database of wrappers, raising the question of how this database would be populated. The present work makes the simplifying assumption that the database is pre-populated with all necessary wrappers. For real world usage, several possibilities exist. First, IT shops who are interested in offering Mixer capability to their customers might code wrappers for their internal websites and distribute a version of Mixer with access to those wrappers. Alternately, a crowd of third-party developers might contribute wrappers for web pages to an internet-wide repository for use internet-wide, as is done with public GreaseMonkey [6] or CoScripter [20] scripts. An extension of this approach is to deploy tools (e.g. Mash Maker [13] or reform [35]) enabling end users to participate in this crowd-sourcing construction of the wrapper repository. Mash Maker [13] maintains exactly such a repository of wrappers contributed by end users. The Accessibilities Commons [17] maintains a similar database of web accessibility enhancements, designed to crowd-source a solution to the accessibility problem.

More formally, a wrapper overlays a relation, i.e. a list of one or more tuples, over the page. The wrapper specifies the type of the tuple as an unordered list of text fields and subrelations (with tuples specified in the same way[1]). Each tuple and "leaf" field is located with an XPath expression [7], interpreted relative to its parent tuples; thus the addressing is similar to that in the Accessibilities Commons [17]. To minimally overcome the limitations of identifying fields with DOM elements (see e.g. Dontcheva et al [12]), fields may optionally refer to a regular expression matching some of the tokens within the chosen element. Thus the Mixer wrapper formalism is sufficiently expressive to wrap pages with a uniformly repeating tuple-type where tuples and fields are contiguous within DOM elements. Partly as a tradeoff for this expressivity, Mixer wrappers are nontrivial to specify; thus the sheer number of wrappers required for widespread Mixer usage seems to present a more limiting bottleneck than the limitations of the expressivity of the wrapper formalism.

2.2 Mixer Program Induction

Generally, *program induction* is the task of constructing a program based on a small number of example executions of the program. Mixer performs program induction by observing the browser actions as the user performs the demonstration, and forming a program which will be executed when the user asks to complete the table.

In order to reduce the prohibitively large search space [19] of programs consistent with the demonstration, Mixer leverages relatively strong assumptions about the Mixer domain to decompose the induction problem into several smaller sub-problems[2]. The assumptions about the domain are:

- Universality: All loops are repeated over all instances in their scope
- Quantification: All loops are scoped over the values in a workspace column
- Task Focus: Only actions which change the workspace can affect the program

The learning mechanism records all browser actions in a log of actions. The log contains the user's actions without modification or generalization of any kind. Parallel to the log, Mixer constructs the program, which represents the best guess about the repeatable procedure whose output the user desires. The instruction set contained in

[1] The allowance for subrelations expands the expressivity of Mixer wrappers to nested tables, also known as non-first normal form relations (see e.g. [38])

[2] This decomposition was chosen to be amenable to Machine Learning classification.

the program are the same as the browser actions contained in the log, with the addition of parameterized versions of each action as well as a looping *foreach* structure. Additionally, a program begins with a preamble which loads the starting page where the replayed browsing actions will commence.

The learning mechanism appends each observed browser action to an accumulated list of steps. The first sub-problem is to classify a given sequence of steps as to whether it forms a complete unit of action, which should affect the program in some way. The Mixer implementation presented here uses a simple heuristic, considering only step sequences terminated by a step inserting new content into the workspace.

Given that a unit does affect the program, Mixer next decides how it does so. The sequence of steps in the unit is decomposed into the *transition*, the *query*, and the *selection*. The query determines how data from previous columns drives widgets to perform the web lookup, and the selection selects information to be inserted into the workspace; the transition contains any preparatory steps which are not data-dependent[3]. The steps in the transition are simply appended to the program at the current insertion point, and afterward a new loop is appended, quantified over the entity type used by the query. The steps of the query are parameterized to depend on data from columns in the workspace, then appended inside the new loop. Finally, the insertion point is advanced to the current end of the loop. The current implementation of Mixer makes no effort to track previous positions of the insertion point; as a consequence the "undo" operation only functions back to the insertion point. Beyond the insertion point, the user cannot effect any changes to the underlying program, so in case of error must restart the session.

The selection is converted into a single atomic command for extracting all content from the visited page. While inserting new content into the workspace always appears to the user to change the workspace, only the first piece of data taken from a visited page actually affects the underlying program. The workspace, meanwhile, contains all the data from the page but only displays the pieces the user has demonstrated so far. Upon demonstration of the addition of subsequent pieces of information to the workspace, the program is unchanged; only the visibility flag for the affected column is toggled. This low-level distinction is made invisible to the user.

3 Evaluation

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.

[3] The transition would contain interactions with widgets of a form which are universal to the tuple, but not to any particular value of a field. For example, checking a checkbox choosing to search for people rather than, say, departments, would be included in the transition; the query would continue by specifying which person to search for.

    - 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 [39]. 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 [37] (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").

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, 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 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 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 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.

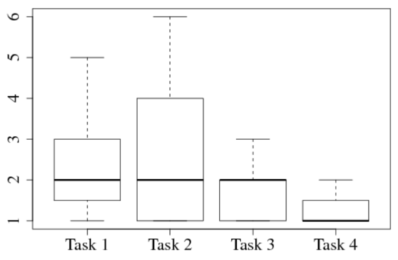

Figure 2. Number of attempts to construct the correct workspace

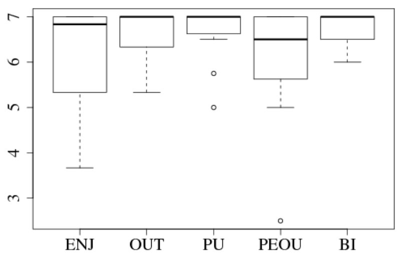

Figure 3. Participants' ratings of Mixer along TAM constructs

5 Discussion

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 struggled 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 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 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.

6 Related Work

Nardi [27] 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 [31] 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 addresses this by specifically disguising the fact that the administrator is programming when interacting with the tool.

Malone et al 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 [23]. 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 [40] 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 [28] build on the work of Lewis and Olson [21] 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 [29] 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 Wizard-of-Oz study of Mixer 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 [22] asks the user to create a set of "VegeTables," each of which corresponds to a script for combining two websites. Karma [36], Dontcheva et al [11] and Mashroom [38] 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 encourages the user to construct the single, unified table that seems to match her 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 [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, 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 agents. The principal actions of a PBD session [18] 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 [8], which assist the user in deciding when to invoke the observed behavior.

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. 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" [10, 24] 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 [4].

As to the second challenge, Modugno and Myers [25] 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.

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 [5] records web actions as general JavaScript. CoScripter [20] chooses a slightly more user-friendly approach, representing the program in a "sloppy" or natural programming language. Query-by-Example and Office-by-Example [41] 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 [3]. 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 [35] or the summaries of Dontcheva et al [12]) or data cleaning (e.g. Potters Wheel [30] or Potluck [15]). Mash Maker [13] 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.

7 Conclusion

Mixer advances mixed-initiative PBD interaction through a novel user-constructed nested table communication method that allows users to declare the outcome they want while implicitly demonstrating how the agent should programmatically perform the task. Mixer specifically allows administrators to automate repetitive web data retrieval and integration tasks they find to be tedious to perform. Our evaluation of the system shows specifying the table to be an effective method for people and the agent to communicate their varying knowledge and needs. The evaluation also reveals a strong likelihood that administrators would use Mixer if it were available to them. The interaction presented in Mixer represents a transition in how office workers engage in computing. Instead of forcing workers to rely on their ability to adapt to the design of IT systems, Mixer empowers workers to leverage their expertise in web data retrieval to train agents to undertake tedious information integration tasks for them.

This work is supported by grant number H133E080019 from the United States Department of Education through the National Institute on Disability and Rehabilitation Research.

References

1. Mixed-Initiative Interaction. In: IEEE Intelligent Systems, vol. 14, pp. 14-23, 1999.
2. Allen, J., Chambers, N., Ferguson, G., Galescu, L., Jung, H., and Taysom, W.: PLOW: A Collaborative Task Learning Agent. In: AAAI, 2007, pp. 22-26.
3. Beemer, B., and Gregg, D.: Mashups: A Literature Review and Classification Framework. In: Future Internet, vol. 1, no. 1, pp. 59-87, 2009.
4. Bigham, J. P., Kaminsky, R. S., and Nichols, J.: Mining web interactions to automatically create mash-ups. In: UIST, 2009, pp. 203-212.
5. Bolin, M., "End-User Programming for the Web," Masters Thesis, MIT, 2005.
6. Boodman, A. "Greasemonkey," http://www.greasespot.net/.
7. Clark, J., and DeRose, S., *XML Path Language (XPath) Version 1.0*, W3C, 1999.
8. Cypher, A.: Eager: Programming Repetitive Tasks by Demonstration. In: Watch what I do: programming by demonstration, MIT Press, 1993, pp. 205-217.
9. Cypher, A.: End User Programming on the Web. In: No Code Required: Giving Users Tools to Transform the Web, Morgan Kaufmann, 2010, pp. 3-22.
10. Dennett, D. C., The Intentional Stance, Cambridge, MA: The MIT Press, 1987.
11. Dontcheva, M., Drucker, S. M., Salesin, D., and Cohen, M. F.: Relations, cards, and search templates: user-guided web data integration and layout. In: UIST, 2007, pp. 61-70.
12. Dontcheva, M., Drucker, S. M., Wade, G., Salesin, D., and Cohen, M. F.: Summarizing personal web browsing sessions. In: UIST, 2006, pp. 115-124.
13. Ennals, R., Brewer, E., Garofalakis, M., Shadle, M., and Gandhi, P.: Intel Mash Maker: join the web. In: SIGMOD Rec., vol. 36, no. 4, pp. 27-33, 2007.
14. Horvitz, E.: Reflections on Challenges and Promises of Mixed-Initiative Interaction. In: AI Magazine, vol. 28, no. 2, 2007.
15. Huynh, D., Miller, R., and Karger, D.: Potluck: Data Mash-Up Tool for Casual Users. In: The Semantic Web, Springer Berlin / Heidelberg, 2007, pp. 239-252.
16. Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J.: Wrangler: Interactive Visual Specification of Data Transformation Scripts. In: CHI, 2011.
17. Kawanaka, S., Borodin, Y., Bigham, J. P., Lunn, D., Takagi, H., and Asakawa, C.: Accessibility commons: a metadata infrastructure for web accessibility. In: Assets, 2008, pp. 153-160.
18. Kosbie, D. S., and Myers, B. A.: PBD Invocation Techniques: A Review and Proposal. In: Watch what I do: programming by demonstration, MIT Press, 1993, pp. 415-422.
19. Lau, T., Wolfman, S. A., Domingos, P., and Weld, D. S.: Programming by Demonstration Using Version Space Algebra. In: Mach. Learn., vol. 53, no. 1-2, pp. 111-156, 2003.
20. Leshed, G., Haber, E. M., Matthews, T., and Lau, T. A.: CoScripter: automating & sharing how-to knowledge in the enterprise. In: CHI, 2008, pp. 1719-1728.
21. Lewis, C., and Olson, G.: Can principles of cognition lower the barriers to programming? In: Empirical studies of programmers: second workshop, Ablex, 1987, pp. 248-263.
22. Lin, J., Wong, J., Nichols, J., Cypher, A., and Lau, T. A.: End-user programming of mashups with vegemite. In: IUI, 2009, pp. 97-106.

23. Malone, T. W., Grant, K. R., Lai, K.-Y., Rao, R., and Rosenblitt, D.: Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination. In: ACM Trans. Inf. Syst., vol. 5, no. 2, pp. 115-131, 1987.
24. Maulsby, D., and Witten, I. H.: Metamouse: An Instructible Agent for PBD. In: Watch what I do: programming by demonstration, MIT Press, 1993, pp. 155-181.
25. Modugno, F., and Myers, B.: Graphical Representation and Feedback in a PBD System. In: Watch what I do: programming by demonstration, MIT Press, 1993, pp. 415-422.
26. Myers, B. A.: Peridot: Creating User Interfaces by Demonstration. In: Watch what I do: programming by demonstration, MIT Press, 1993, pp. 125-154.
27. Nardi, B. A., A small matter of programming: perspectives on end user computing: MIT Press, 1993.
28. Nardi, B. A., and Miller, J. R., *The Spreadsheet Interface: A Basis for End User Programming*, HP Laboratories, 1990.
29. Norman, D. A.: Cognitive Engineering. In: User centered system design: new perspectives on human-computer interaction, Lawrence Erlbaum Associates, 1986, pp. 31-61.
30. Raman, V., and Hellerstein, J. M.: Potter's Wheel: An Interactive Data Cleaning System. In: The VLDB Journal, 2001, pp. 381-390.
31. Rode, J. A., Toye, E. F., and Blackwell, A. F.: The fuzzy felt ethnography--understanding the programming patterns of domestic appliances. In: Personal and Ubiquitous Computing, vol. 8, no. 3, pp. 161-176, 2004.
32. Sundström, P., Taylor, A. S., Grufberg, K., Wirström, N., Belenguer, J. S., and Lundén, M.: Inspirational Bits: Towards a shared understanding of the digital material. In: CHI, 2011.
33. Tecuci, G., Boicu, M., and Cox, M.: Seven Aspects of Mixed-Initiative Reasoning: An Introduction to this Special Issue on Mixed-Initiative Assistants. In: AI Magazine, vol. 28, no. 2, 2007.
34. Tomasic, A., Zimmerman, J., Hargraves, I., and McMullen, R.: User Constructed Data Integration via Mixed-Initiative Design. In: Interaction Challenges for Intelligent Assistants, 2007, pp. 122-123.
35. Toomim, M., Drucker, S. M., Dontcheva, M., Rahimi, A., Thomson, B., and Landay, J. A.: Attaching UI enhancements to websites with end users. In: CHI, 2009, pp. 1859-1868.
36. Tuchinda, R., Szekely, P., and Knoblock, C. A.: Building Mashups by example. In: IUI, 2008, pp. 139-148.
37. Venkatesh, Viswanath, Bala, and Hillol: Technology Acceptance Model 3 and a Research Agenda on Interventions. In: Decision Sciences, vol. 39, no. 2, pp. 273-315, May, 2008.
38. Wang, G., Yang, S., and Han, Y.: Mashroom: end-user mashup programming using nested tables. In: WWW, 2009, pp. 861-870.
39. Zimmerman, J., Rivard, K., Hargraves, I., Tomasic, A., and Mohnkern, K.: User-created Forms as an Effective Method Of Human-agent Communication. In: CHI 2009, pp. 1869-1878.
40. Zimmerman, J., Tomasic, A., Simmons, I., Hargraves, I., Mohnkern, K., Cornwell, J., and McGuire, R. M.: VIO: a mixed-initiative approach to learning and automating procedural update tasks. In: CHI, 2007, pp. 1445-1454.
41. Zloof, M. M.: QBE/OBE: A Language for Office and Business Automation. In: IEEE Computer, vol. 14, no. 5, pp. 13-22, 1981.