Predictive Web Automation Assistant for People with Vision Impairments

Yury Puzis
Charmtech Labs LLC
CEWIT SBU R&D Park
Stony Brook, NY, USA
yury.puzis@gmail.com

Yevgen Borodin
Charmtech Labs LLC
CEWIT SBU R&D Park
Stony Brook, NY, USA
borodin@charmtechlabs.com

Rami Puzis
Information Systems Engineering
Ben-Gurion University
Beer-Sheva, Israel
puzis@bgu.ac.il

I.V. Ramakrishnan
Charmtech Labs LLC
CEWIT SBU R&D Park
Stony Brook, NY, USA
ram@charmtechlabs.com

ABSTRACT

The Web is far less usable and accessible for people with vision impairments than it is for sighted people. Web automation, a process of automating browsing actions on behalf of the user, has the potential to bridge the divide between the ways sighted and people with vision impairment access the Web; specifically, it can enable the latter to breeze through web browsing tasks that beforehand were slow, hard, or even impossible to accomplish. Typical web automation requires that the user record a macro, a sequence of browsing steps, so that these steps can be automated in the future by replaying the macro. However, for people with vision impairment, automation with macros is not usable.

In this paper, we propose a novel model-based approach that facilitates web automation without having to either record or replay macros. Using the past browsing history and the current web page as the browsing context, the proposed model can predict the most probable browsing actions that the user can do. The model construction is "unsupervised". More importantly, the model is continuously and incrementally updated as history evolves, thereby, ensuring the predictions are not "outdated".

We also describe a novel interface that lets the user focus on the objects associated with the most probable predicted browsing steps (e.g., clicking links and filling out forms), and facilitates automatic execution of the selected steps. A study with 19 blind participants showed that the proposed approach dramatically reduced the interaction time needed to accomplish typical browsing tasks, and the user interface was perceived to be much more usable than the standard screen-reading interfaces.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; H.5.4 [**Information Interfaces and Presentation**]: Hypertext / Hypermedia - navigation

WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2035-1/13/05.

Keywords

Web; accessibility; blind; low vision; browser; screen-reader; macro; non-visual; automation; model; prediction; sequence alignment; interface agent; adaptive interface;

1. INTRODUCTION

While browsing, blind users rely on screen-readers [1, 2, 3] which are assistive tools that narrate the screen content. Screen-readers enable sequential navigation over the content and provide numerous shortcuts that can speed up the navigation. In most cases, however, even expert users fall back to the most basic navigation, in which they go through items one by one sequentially. In contrast to sighted users, visually-impaired users cannot just "glance" over a webpage whenever there is a need to quickly find the right button to click or text to read. The sequential mode of interaction with webpages and the inability of screen-reader users to determine whether a piece of content is important before they listen to it often results in high cognitive load that, in turn, impacts users' browsing speed and their overall browsing experience. As a result, visually-impaired users spend significantly more time on seemingly simple online tasks.

Web automation - a process of automating browsing actions on behalf of the user - has the potential to bridge the web accessibility divide between the ways people with and without vision impairments use the Web. The traditional approach to Web automation is via *macros*, pre-recorded sequences of instructions that can automate browsing steps. Macro-based automation tools are finding their way into mainstream technology as part of screen-readers, browser extensions, and browser features (e.g., form filling). The adoption of macro-based Web automation has been stifled by a number of challenges: (a) it requires the user to make an effort to create, manage, and, then, find and replay macros, and (b) it lacks the flexibility necessary to allow the user to deviate from the pre-recorded sequence of steps, or to choose between several options in each step of the macro. Those difficulties make macro-based approaches too limiting to be useful for people with vision impairments; none of our 19 blind participants reported using any kind of web automation technology.

In this paper, we present a novel approach to accessible web automation for people with visual impairments. The salient aspects of the approach, embodied in the prototype system - *Web Automation*

Assistant - are as follows: First, it uses a computational model, based on a modified dynamic programming sequence alignment algorithm, to predict the most probable actions the user can take when browsing. Second, the construction of the model is "unsupervised", continuous, and incremental, i.e., the model is silently updated after every user action. Third, the Assistant uses a novel user interface that focuses the user on the predicted browsing steps and, if necessary, facilitates their automatic execution. This is done by continuously monitoring user activity, and, when requested, guiding the user through browsing tasks step by step, providing relevant contextual suggestions, and enabling the user to confirm each step before the suggestions are executed. The effectiveness of the Assistant was validated in experiments evaluating the prediction accuracy, and in the user study evaluating the Assistant's usability and user experience with 19 blind participants.

It is worthwhile pointing out that the Assistant can be viewed as an *interface agent* [15, 18] - a system that observes user interactions with the UI and interacts with the user or with the UI. This work is also related to *adaptive user interfaces*. Adaptive interface tailors the presentation of functionality to better fit an individual user's tasks, usage patterns, and abilities [14].

2. USE SCENARIO

To illustrate how screen-reader users accomplish browsing tasks, let us consider a simple scenario of finding the "checkout button" to buy some product after adding it to cart. The strategies employed by any given user will depend on the level of expertise of the user and his/her familiarity with the task and/or web page. Beginning users often listen to the content from the top of the page continuously until they hear the button label read to them (if they recognize that it is the button they need). They can also expedite the process by pressing the "Down" key to make the screen-reader skip to the next line. This unfortunately is not much faster because one has to hear at least some of the content before realizing that it is not relevant. More advanced users may employ the search feature to find the button by searching for its label, assuming they know what the label is, and the label is a text string.

Expert users would employ element-specific navigation allowing them to jump back and forth among elements of certain HTML type: buttons, headings, edit fields, etc. Expecting to find a HTML button, they may press "B" to jump only among buttons narrowing down their search space and reducing the amount of information they have to listen to. If they are familiar with the web page, they sometimes find quicker ways to get around, e.g., if the page has many buttons, instead of pressing "B" multiple times to get to the right button, the user may go to the end of the page and go in reverse order, or press "H" to go to next heading, and then "Shift+B" to go to the button that precedes the heading. Unfortunately, if the web page gets a slight make over, or if the user cannot remember the structure of the page, s/he has to fall back to the least efficient navigation with arrows. For instance, the check out button may turn out to be an image-link with the label "Go to Cart", in which case neither pressing "B" nor searching for the label will help.

In contrast, the Web Automation Assistant described in this paper allows the user to examine the most likely action-objects, e.g., a button that needs to be pressed or a form-field to be filled. When the user looks through the suggestions, there may only be the "checkout" and "keep shopping" buttons, because the Assistant will remember that those are the buttons the user usually presses after adding an item to cart. Furthermore, because of the way the actions are recorded in the model, the Assistant is likely to find the button even if its label and/or the position on the page changes. Thus, the Assistant will allow even a novice user to be as efficient as an expert user, while also relieving the expert user from having to remember the web page structure.

In a similar fashion, using the Assistant, the user can just as quickly go through the entire shopping transaction: enter user name and password to log in, select the shipping address, shipping method, billing information, and finally complete the purchase. The Assistant will also help to not only find the form fields to fill, but also enter the values, if necessary. One *could* hypothetically record a macro to help guide the user step by step through the same exact transaction; however, not only does this require some effort on the part of the user, but macros are also a lot less flexible and cannot support the following scenarios: the Assistant lets the user choose from among several possible actions; it can begin guiding from any step of the suggested sequence; it allows the user to diverge from but then continue with the suggested sequence.

3. TECHNICAL PRELIMINARIES

We define the *environment* with which the user and the Assistant interact to be the web browser coupled with the screen-reader. The user, and the Assistant, interact with the environment by executing *browsing actions* ("actions" for brevity). For example, the user can press a keyboard shortcut; the Assistant can simulate a keyboard shortcut press on the user's behalf. In response to the browsing actions, the environment triggers *events*, e.g., browser's JavaScript events, screen-reader's virtual cursor movement, etc.

An *automation instruction* is defined as the instruction that is needed to execute a specific browsing action programmatically on user's behalf. We define 3 types of automation instructions: *Value Change* (change the value of an HTML form field such as textbox, radio button, checkbox, selection list, etc.), *Invocation* (following a link, pressing an on-screen button, except for submit a form), and *Form Submission* (separated for the purpose of combining multiple ways of form submission). Automation instructions are automatically inferred from observed events (e.g., a JavaScript onsubmit event can be used to generate a Form Submission instruction specifying which form needs to be submitted). An action may trigger different events at different times and, hence, may or may not always generate equivalent automation instructions. Similarly, different actions may result in generating equivalent automation instructions. In the remainder of the paper we will use browsing actions and automation instructions interchangeably when there is no confusion in the context.

We define *history* as a sequence of automation instructions that appear in the order in which they were generated from events. An example of history is: <a Value Change for setting the "First Name" textbox to "John", followed by a Value Change for setting the "Last Name" textbox to "Doe", followed by a Form Submission>.

The *query* denotes the set of all suffixes of the browsing history. Observe that suffixes denote recent browsing actions.

4. MODEL-BASED WEB AUTOMATION

4.1 Overview

The basis of our approach is based on the idea that if some subsequence of the history matches the most recent browsing actions, then the action immediately following the matched subsequence can be predicted as the user's next action, and hence it is a possible candidate for suggestion. To illustrate, let

`...ABCD...AECE...ACE...ABECF...`

be a part of the history capturing prior actions, and let `ABC` be one of the suffixes in the current query. Each letter represents a single browsing action and the same letters are used for equiva-

lent actions. Then, we can align ABC to every subsequence of the browsing history and predict the next most likely user action (denoted by the ? symbol). In our example we get the following four local sequence alignments:

ABCD AECE A-CE ABECF ...
ABC? ABC? ABC? AB-C?

First, the suffix may align to more than one subsequence in history. In some cases, different actions may follow each of the aligned subsequences (e.g., D, E, F above). This can be a result of the user executing the same task on a modified webpage, or of the user making different choices within the browsing task, or of the user switching to a different browsing task altogether. Second, the suffix may partially match a subsequence in the history (e.g., AECE, ACE, ABECF), in which case the alignments will have different *edit distances* (minimal number of insertions, deletions, and replacements needed to modify one sequence into another). The larger the edit distance, the smaller is the likelihood that the alignment is useful for making a prediction. However, the ability to find partial matches is very important for the algorithm to be useful whenever the user deviates from historic record.

The time interval between actions in history may have high variance, either due to periods of user inactivity, or because a subsequence of history was deleted (or never recorded). However, it is not obvious that accounting for differences in time intervals will significantly improve the performance of the statistical model, since user's previous actions may be indicative of his/her next action even if they are separated by a large time span. Similarly, a very short (or very typical) time span between two actions is not a guarantee that the user did not make an unpredictable 'switch' to a completely new browsing task. An investigation of usefulness of a sophisticated handling of time intervals may require an in-situ study, and is beyond the scope of this paper.

4.2 Model Construction

The basis of our web automation model (overviewed above) is the Smith-Waterman [30], a dynamic programming sequence alignment algorithm. Smith-Waterman builds a Sequence Alignment Table T of size $m+1$ columns by $n+1$ rows. Each row i, and column j in the Sequence Alignment Table corresponds to a single automation instruction q_i, and h_j respectively. The automation instructions in the rows and columns are generated from the user's browsing history. Column m and row n represent the most recent automation instruction. $T[i][j]$ denotes the score of the alignment of two sequences composed of automation instructions from row 0 to i and column 0 to j.

In building the Sequence Alignment Table, we are interested in the alignment of the query to each subsequence of history. Those alignments are represented by the cells in the bottom row of the Sequence Alignment Table. Given a table of $n+1$ rows, cell $T[n][j-1]$ scores the alignment of the query with the subsequence of history that ends with action h_{j-1}. We say that the subsequent action h_j is predicted by the alignment $j-1$ with the score of $T[n][j-1]$.

Consider the example in Figure 1, in which we align $V_1V_2S_1$ (recent actions) to $\ldots V_1V_2I_1I_2S_1V_3\ldots$ (subsequence of history), where V are Value Change instructions, I are Invocation instructions, S are Form Submission instructions. The score modifier for a match or a mismatch of q_i, and h_j, is 1 and -1, respectively:

The positive scores in the bottom row indicate that there are three equally likely alignments, each scored 1: (a) rows V_1V_2 match to columns V_1V_2, row S_1 is "deleted", and the predicted action is I_1 (b) rows V_1V_2 match to columns V_1V_2, row S_1 is (mis)matched with column I_1, and the predicted action is I_2, and (c) row S_1 is matched to column S_1, and the predicted action is V_3.

	...	V_1	V_2	I_1	I_2	S_1	V_3
	0	0	0	0	0	0	0
V_1	0	1	0	0	0	0	0
V_2	0	0	2	0	0	0	0
S_1	0	0	**1**	**1**	0	**1**	0

Figure 1: Example of a Sequence Alignment Table

4.3 Model Optimization

Our sequence alignment is based upon the following definition, drawn from the standard Smith-Waterman algorithm:

$$T_o[i][j] = \max \begin{cases} 0 & \text{case 1} \\ T_o[i-1][j-1] + S_o[i][j] & \text{case 2} \\ T_o[i-1][j] + S_o[i][j] & \text{case 3} \\ T_o[i][j-1] + S_o[i][j] & \text{case 4} \end{cases}$$

Let $\equiv$ denote a match of q_i and h_j, let $\equiv_{vc}$ denote a match of q_i and h_j such that both are Value Change, and have the same value, and let $\not\equiv$ denote a mismatch of q_i and h_j. As already mentioned, Smith-Waterman defines the score modifier, denoted by S_o, as 1 if $q_i \equiv h_j$ and -1 otherwise. We depart from Smith-Waterman by redefining the score modifier as follows:

$$S_o[i][j] = \begin{cases} 1 & \text{if } q_i \equiv h_j \\ 2 & \text{if } q_i \equiv_{vc} h_j \\ P_o[i][j] & \text{if } q_i \not\equiv h_j \end{cases}$$

where P_o is a progressive penality for a mismatch of two actions, and is defined as follows:

$$P_o[i][j] = \begin{cases} P_o[i-1][j-1] - 1 & \text{if case 2} \\ P_o[i-1][j] - 3 & \text{if case 3} \\ P_o[i][j-1] - 2 & \text{if case 4} \\ 0 & \text{otherwise} \end{cases}$$

Value Change represents a modification to the HTML form fields, and, in some use scenarios, the chosen value of a form field may hint at future user actions. This scenario is accounted for by setting $S_o = 2$ if $q_i \equiv_{vc} h_j$.

The introduction of progressive penalty P_o is done for the following reason. The necessary length of the suffixes in the query is hard to predefine. Ideally, this number should be large enough to include all the actions that can influence the user's choice of the next action, but, at the same time, small enough to exclude the actions that are "irrelevant" when predicting the next action. The local sequence alignment approach taken in this paper allows us to examine the mapping of all possible suffixes to all subsequences of the history simultaneously. For instance, the example in Figure 1 examines the suffixes S_1, V_2S_1, and $V_1V_2S_1$; the length of positively scored alignments is 3 for cases (a) and (b) and 1 for case (c). However, this implies that some of the suffixes might be too long (contain "irrelevant" actions). We deal with this by penalizing the mismatch of two actions on a progressive scale, under the assumption that actions become "irrelevant" when they are followed by too many mismatches.

Finally, we want to prioritize case 2 over case 4, and case 4 over case 3. Therefore, the score modifier for aligning mismatching q_i and h_j (case 2) is smaller than the modifier for inserting an item from history (case 4), which is in turn smaller than the modifier for deleting an item from the query (case 3).

4.4 Generation of the Prediction List

As was discussed above, there may be multiple local sequence alignments between the query and the history. Each such alignment can result in a potential prediction. Let the *prediction list* be defined as a list of predicted browsing actions, or, equivalently, automation instructions, ordered by likelihood. The process of inferring a new prediction list is illustrated in Figure 2.

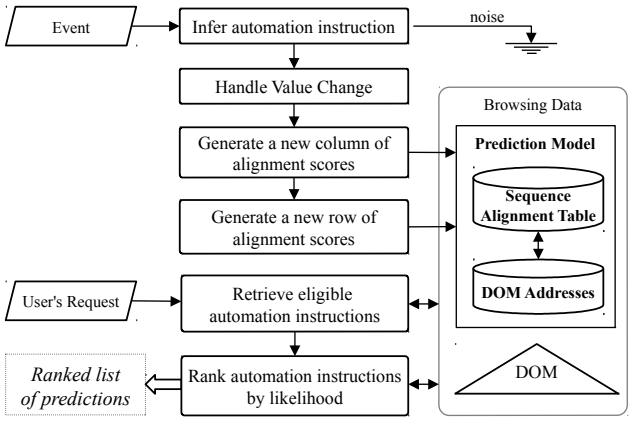

Figure 2: Process Flow for Predicting User Actions

The model is updated on the fly, every time a new environment event is logged. First, unless the event can be ignored (as discussed in Section 4.5), a new automation instruction is inferred. Second, if Value Change was generated, and it has already been executed on the same page since the last page load, then its value is assigned to the first occurrence of the instruction, and the new occurrence is discarded. Third, the model is incrementally updated with the new automation instruction by generating a new column and a new row of alignment scores in the Sequence Alignment Table.

Observe that the model is updated incrementally. This is because the new alignment of the updated query and history is computed by generating a single table row and column (instead of recomputing the Sequence Alignment Table from scratch). This is possible because the Sequence Alignment Table that needs to be computed at step i is a subset of the Sequence Alignment Table that needs to be computed at step $i+1$. Moreover, adding a new row (column), only involves examining the row above (column to the left), and generation of prediction list only involves the examination of the bottom row. Therefore, only the bottom row and the rightmost column of the Sequence Alignment Table need to be stored (see [22] for further details).

The algorithm generating the prediction list (or, equivalently in this context, automation instructions) is executed upon user's request for suggestions (for example, a shortcut). First, each prediction is evaluated for eligibility to be scored. A prediction is considered non-eligible and is discarded if at least one of the following is true: (a) the targeted webpage element (visual primitive of a webpage) does not exist, (b) it is hidden, (c) it is a disabled or a read-only form field, or (d) the action has been executed since the last page load. For instance, actions I_1 and I_2 in Figure 1 would be discarded if the Form Submission action S_1 triggered a new page load. The hidden, disabled, and read-only form fields are discarded because their value is usually assigned by the webpage itself, as a function of user's interaction with the enabled webpage elements, or factors beyond user's control (e.g., the date).

The eligible predictions are scored as follows. A sliding window of size k is used to collect k top scored, *unique* predictions, i.e., the sliding window always contains k most likely predictions so far. Since predictions need to be unique, each prediction's score is defined as the maximum of scores assigned to it by different alignments. If the scores of two different predictions are equivalent, the prediction with the score provided by the more recent (larger column j index) alignment is ranked higher (discussed in Section 4.5). For instance, the ranking order in Figure 1 would be V_3, I_2, I_1. Furthermore, by ignoring the *number* of predictions for each action, and tie breaking, we aim to minimize the effects of data decay.

The time complexity for updating the model and recomputing the prediction list after each action is $O(m \log k + n)$, where k is the size of the sliding window, $O(m)$ is the time to add a new row, $O(n)$ is the time to add a new column, and $O(m \log k)$ is the time to generate a new prediction list. However, since k is a small constant - providing the user with too many suggestions would be counterproductive - the complexity is $O(m+n)$. The space complexity is also $O(m+n)$.

4.5 Technical Details

Automation instructions are generated from environment events. The mapping between events and instructions is:

- Value Change: triggered by JavaScript *onchange*, or *onreset* events: change of value of an HTML form field such as textbox, radio button, checkbox, selection list, etc.
- Invocation: triggered by (a) JavaScript *onclick* event: a mouse click (except on a form submit button), or (b) JavaScript *keypress* event: a press of the 'enter' key (except in a form field, or a form 'submit' button).
- Form Submission: triggered by a JavaScript *onsubmit* event.

We say that two automation instructions are equivalent if and only if they have the same type and:

- Both are an Invocation of a URI element (e.g., following a link), and refer to the same URI. Otherwise,
- Both refer to the same unique webpage element. The HTML elements can be uniquely identified by an XPath [4] (see [21] for XPath addressing resilient to DOM changes).

When comparing automation instructions, we need to consider the semantics (the actual effect) of the instructions, rather than the specific events from which the instructions were inferred. A mouse click and a press on the 'enter' key are assigned the same type because more often than not the semantics of a mouse click and a press on the 'enter' key is the same (for instance, clicking a link with the mouse, or, focusing on the link and then pressing 'enter'). By not making any assumptions about the specific input device, we make it easier for the model to compare and identify equivalent instructions and ensure that the model created by observing keyboard events will be usable when the user switches to the mouse and vice versa. The mouse can be useful for people with low vision and it can enable sighted people to easily create models for people with vision impairments. This generalization will also enable transparent support for other input modalities (such as Braille devices, touch screens, or speech input) in the future.

A Value Change instruction is parametrized with any input in the corresponding form field, such as entering value into a textbox, (un)checking a checkbox, changing the selection in a combo-box, etc. Value Change is recorded only when the user exits the form field, i.e. the value is final. The value of all the subsequent modifications of the same form field (i.e. if the user comes back to update the value) are stored in the original instruction, i.e. no new instruction is created. For instance, if the user (a) sets "First Name" field to "John", then "Last Name" field to "Smith", and then updated the "First Name" field to "Sam", this generates a sequence of (a) Value

Change "Sam" for "First Name", followed by (b) Value Change "Smith" for "Last Name".

Form submission is always related to the specific form being submitted, and therefore the semantics of submitting a form differs from Invocation. Different Invocation instructions that can be used to submit the same form include pressing 'enter' or clicking the mouse on the form 'submit' button (of which there can be more than one for the same form, including outside the form element, as per HTML5 specification), pressing 'enter' inside a form field, and interacting with a custom form submission mechanism. We want to make sure that the semantically equivalent but otherwise different ways of submitting the same form are considered equal.

However, the general case of comparing instructions by their semantics is not reliable (except form submission, invoking a URI, and possibly other special cases). If a webpage has two controls that cause the same effect (e.g., two buttons that open the same shopping cart) by running custom JavaScript, the corresponding automation instructions will not be equivalent. This is mitigated by the fact that if the user chooses to use both controls interchangeably, the system will learn both variations and so it will provide useful suggestions regardless of user's choice.

5. PUTTING IT ALL TOGETHER

Figure 3 provides a high level overview of Assistants' architecture. The Assistant updates the Sequence Alignment Table by listening to events triggered by browsing actions. The user interacts with the Assistant as follows: (a) while browsing, the user requests suggestions (shortcut); (b) the Assistant exposes a set of suggestions by associating each suggestion with the relevant webpage element; (c) the user examines the suggestions, using a combination of shortcuts provided by the browser, the screen-reader, and the Assistant; the Assistant provides voice feedback where appropriate (supplementing, or replacing the screen-reader feedback); For example, if a textbox is labeled "First name", it has no value, and the suggested value is "John" then the system will announce "Textbox 'First name' blank. Suggestion: John" when the user visits this textbox; (d) the user either confirms automatic execution of one of the suggestions (shortcut), or ignores the suggestions; (e) the user (optionally) verifies that the action was executed correctly by listening to the feedback provided by the Assistant (e.g., voicing the new value of the modified textbox), and/or examining the webpage; (f) the user continues browsing.

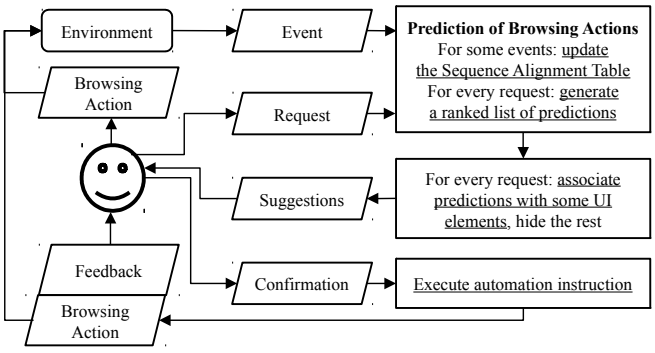

Figure 3: Web Automation Assistant Architecture

There are several important details in this design. First, the user never leaves the webpage to review the suggestions, and the webpage elements are not reordered. Instead, the user is focused on the most important webpage elements and the associated suggested actions. This helps the user to always know the position of the virtual cursor on the webpage, and prevents unexpected context changes. Second, each suggestion is for a *single* action, rather than for a sequence of actions, enabling easy deviation from a scenario after automating one or more actions. Third, confirming one of the suggestions, or interacting with the webpage in any way may automatically refresh the available suggestions, and/or update the model with new data. Fourth, since the system is always monitoring user actions, there is no need to turn it 'on' or 'off'. Fifth, the user is not required to use the Assistant to automate a suggestion s/he finds useful, and can instead use standard screen-reader or browser commands for that purpose: rather than changing his/her workflow, the user can use the Assistant as an "advisor" and, when necessary, an "automator".

6. EVALUATION

We developed a prototype implementation of the Web Automation Assistant. In this section, we describe the evaluation of the model's prediction accuracy, as well as report on the results of a user study with blind participants.

6.1 Model Evaluation

6.1.1 Methodology

The data for model evaluation was collected by observing 8 Firefox users performing 6 different browsing tasks, each associated with a different website. The browsing tasks were chosen to be comparable in difficulty and to be representative use cases for people with vision impairments (according to our accessibility consultants). The 6 chosen tasks were: (1) registering a new website account (ebay.com); (2) searching for and booking a hotel (hilton.com); (3) website registration and product checkout (tigerdirect.com); (4) product checkout (amazon.com); (5) job search (monster.com); and (6) searching for a restaurant and sharing the information with friends (yelp.com).

The data was collected by sighted, rather than blind users, for pragmatic reasons - it takes a blind user on average 5-10 times longer to complete a single task compared to a sighted person, which makes data collection prohibitively expensive. However, as we confirmed in a pilot data collection, the model works the same way for sighted and blind users because they would have to take the same exact steps to complete the tasks. The actions predicted by the model are limited entirely by the functionality of a website; the actions are not specific to non-visual browsing (e.g., movement of the virtual cursor), and are input-modality independent (mouse, keyboard). Hence, the accuracy of the underlying model can be evaluated with sighted, as well as with blind users.

The users generated 5 "noisy" browsing sequences per website by performing a task on each website 5 times, each time varying the browsing actions as they would normally, e.g., choosing a different shipping method while shopping, reviewing product specifications, changing the mind and choosing a different product, filling out form values in different order, re-entering (updating) data in fields, using different methods to submit a form, etc. The users were not constrained by a time limit and completed all the browsing tasks. The resulting dataset contained a total of 240 browsing sequences (30 per person) and 3074 browsing actions (65% Value Changes, 19% Invocations, and 16% Form Submissions).

The goal of the evaluation was to measure the ability of the model to predict a *gold-standard* (minimal) sequences of actions from a noisy browsing history. That is, we considered a scenario in which the user's intention was to complete the remainder of any given browsing task using the Assistant with a minimal number of

steps. A gold-standard sequence was independently handcrafted for each of the 6 websites; the resulting sequences contained a total of 51 actions, of which 72% were Value Changes, 17% were Invocations, and 11% were Form Submissions.

The evaluation was done in iterations, where in each iteration $j = 1...6$: 1) the model was updated with randomly selected noisy browsing sequences one by one (from the same website, and the same subject), 2) a new sequence of prediction lists ξ_j was generated, and 3) the accuracy of the prediction was evaluated by comparing the model prediction to the gold standard sequences. That is, on the first iteration, the model contained a single browsing sequence, on the second - two browsing sequences, and so on. The accuracy for each iteration j, denoted μ_j^{ξ}, is then averaged across all users and websites. The gold-standard sequence was not used to update the model.

Given a gold-standard sequence $h_i \in H$, the prediction list $L_i \in \xi$ was evaluated with the metric *precision at k*, denoted by P@k, and the *mean reciprocal rank* metric, denoted by MRR, of the correct prediction h_{i+1} (sliding window size is unbounded). MRR is defined as follows:

$$MRR(H, \xi) = \frac{1}{|H| - 1} \sum_{i=1}^{|H|-1} \psi_i$$

If $h_{i+1} \in L_i \in \xi$ then the utility function $\psi_i = 1/L_i(h_{i+1})$, and $\psi_i = 0$ otherwise; $L_i(h_{i+1})$ is the rank of h_{i+1} in L_i.

It is important to note that the predictions (in form of suggestions) might not be reviewed by the user in the order produced by the ranking algorithm. In a user study with people with vision impairments (Section 6.2), we presented the suggestions in the screen-reading order (preorder DOM traversal) to ease transitions between reviewing suggestions and regular browsing. However, MRR is useful for comparing prediction algorithms, and for producing a rough estimate of a threshold k below which the predictions can be discarded because they are unlikely to be correct. This threshold is the size of the sliding window used to generate the prediction list (see Section 6.2), and is also used for the P@k metric. The smaller is the value of k the more preferable it is for the user, and the less likely it is that a useful prediction will be part of the list made avaliable to the user. In our experiments we set $k = 5$, as a compromise between the two competing considerations, based on empirical obsevations of system results and on a pilot user study.

6.1.2 Results

Table 1 compares the accuracy of 3 methods for gold-standard sequence prediction: (a) $P(h_{j+1}|h_j)$: bigram frequency count, where each bigram represents two consequitive user actions h_j, h_{j+1}, i.e. predict action $j + 1$ based on a single most recent action j, (b) $T[i][j]$: the unmodified Smith-Waterman [30], and (c) $T_o[i][j]$: the optimized Smith-Waterman. $S(\cdot)$ denotes discarding of non-eligible predictions (as described in Section 4.4).

	P@k		MRR	
Algorithm	μ_1^{ξ}	μ_6^{ξ}	μ_1^{ξ}	μ_6^{ξ}
$P(h_{j+1}\|h_j)$	0.23	0.27	0.19	0.36
$S(P(h_{j+1}\|h_j))$	0.43	0.46	0.45	0.30
$T[i][j]$	0.36	0.27	0.17	0.15
$S(T[i][j])$	0.45	0.33	0.27	0.20
$T_o[i][j]$	0.52	0.54	0.35	0.41
$S(T_o[i][j])$	0.62	0.69	0.45	0.49

Table 1: Accuracy of Predicting the Gold-Standard Action Sequence from User-Generated Action Sequences

The results show that (i) looking back at more than one preceeding user action, (ii) optimization of the Smith-Waterman parameters, and (iii) skipping of non-eligible actions all contribute towards better performance; although not shown here, each optimization was also tested independently. Of course, the disadvantages of making predictions just based on a single most recent action will be erased if the user never deviates from his previous steps, but this is a very uncommon scenario in web browsing.

6.2 User Study

We tested the effectiveness of the Assistant in a user study with 19 blind screen-reader users performing tasks on webpages with and without the help of the Assistant. The following hypotheses were formulated:

H1: Visually impaired users can complete tasks significantly faster when using the Assistant (either A or B) than when using a standard screen-reader.

H2: Usability of non-visual browsing is significantly better when using the Assistant (either A or B) than when using a standard screen-reader.

6.2.1 User Interface

In consultation with web accessibility experts, we have designed several versions of the Assistant's interface with two versions making the final cut in a pilot user study. Since even small details of UI design have the potential to influence results, we decided to test both versions of the Assistant UI.

Assistant A. The S and Shift+S keys are used to move the screen-reader's virtual cursor to respectively the next and the previous webpage element for which an action is suggested. This interface is based on the standard screen-reading interface for navigating among webpage elements of a particular type, e.g., B and Shift+B for buttons, A and Shift+A for links.

Assistant B. A single shortcut is used to toggle on/off the "suggestions" mode, in which the user can use standard screen-reader shortcuts, but navigation is only allowed among the suggestions of the Assistant, making the rest of the content "disappear". If the current screen-reader's virtual cursor position is not associated with a suggestion (because the mode was just turned on or the suggestions changed), the cursor position is moved to the suggestion topologically following the current position. If there is no such suggestion, the user is taken to the suggestion topologically preceding the current position. Otherwise, "no suggestions" is voiced.

It is imporant to note that both Assistant A and B interfaces are designed to coexist in the same system with a screen-reader. If none of the suggestions are useful for the user, he/she can use the standard screen-reader shortcuts to acomplish their immidiate goal, and then continue using the Assistant. Both Assistant A and B interfaces can also coexist in the same system with each other.

For example, imagine that the Assistant is used on a web page containing a large number of elements, of which the 5th one, a textbox, and the 32nd one, a button, are associated with suggestions; the user's current reading position is on the 10th element. With Assistant A, pressing the 'S' key will navigate to the button, while pressing 'Shift+S' will navigate to the textbox. With Assistant B, when the suggestion mode is turned on, the user will be taken to the button, and pressing the 'Up' key will take the user to the textbox. In both interfaces, the content and the suggestions will be read the same way, e.g., "Textbox 'First name' blank, Suggestion: 'John"'. Additional shortcuts can be used to iterate over alternative values, if any. Pressing Ctrl+Space will execute the suggestion, voice "Textbox 'First Name' John", and replace the old set of suggestions with a new one. After a suggestion is executed,

when using Assistant A, the reading cursor will remain on the same element, and the user will have to press 'S' or 'Shift+S' again or navigate normally. When using Assistant B, the user's position will be automatically moved to the appropriate suggestion; the user will have to exit the suggestion mode to navigate normally. It is notable that .

6.2.2 Subjects and Methodology

The 19 participants of our study consisted of 58% males, and were on average 54 ($\sigma = 12$) years old. The age group represents the typical age of a screen-reader user, as many people lose sight later in life. The participants were White/Caucasian (42%), Black/African-American (26%), Latino/Hispanic (12%), Asian (5%), Indian (5%), Central-American (5%), and 5% declined to respond. The participants rated their level of computer experience at "not comfortable" (0%), "mildly comfortable" (0%), "comfortable" (26%), "very comfortable" (58%), and "expert" (16%). About 90% of the participants used Internet Explorer as their primary browser, 5% use Firefox and another 5% use Safari. About 90% of the participants used JAWS as their primary screen-reader, 5% used ZoomText and 5% VoiceOver. All the participants used JAWS before. The number of hours per week that participants regularly spent using the Web was 1-5h (32%), 6-10h (16%), 11-20h (26%), and more than 20h (26%).

The study was conducted using Capti Web browsing application that a screen-reading interface very similar to the standard screen-readers, with the Firefox browser and IVONA [5] Text-To-Speech (TTS) engine with the voice "Eric", speech rate of 180 words per minute.

In this study, we asked the participants to perform 6 browsing tasks, each associated with a different website. The tasks and the websites were the same as in Section 6.1.1. Subjects were asked to perform the tasks using 3 different systems: the baseline screen-reader without automation (**N-A**), Assistant A (**A-A**), and Assistant B (**A-B**). Each system was evaluated with 2 consecutive tasks. We counterbalanced the task order, the system order, and task-to-system assignment.

Prior to performing the tasks, the participants were explained how to use the system they were about to evaluate, and given an opportunity to practice on a sample web site until they felt comfortable to proceed. Since all the participants had prior experience with the JAWS screen-reader, the interface of our Capti screen-reader, which provides the commonly used features and shortcuts of JAWS, was immediately familiar.

The profiles of the statistical models for the Assistant (same for A-A and A-B) were created in advance, i.e. for each user all initial suggestions contained the correct action that the user was expected to execute, or confirm. During the evaluation, the Assistant updated the model and changed its suggestions based on the subject's behavior when s/he deviated from the initial suggestions. In practice, during the study, the participants ended up with at most 2 suggestions to choose from.

For each task, we measured the completion time, or recorded a time-out if the subject exceeded 10 min. (Table 2). The tasks that timed-out were not included in quantitative results computation. After each system evaluation (2 consecutive tasks) the participants answered the System Usability Scale (SUS) [11] questionnaire. After the completion of all tasks, the participants had to compare the 3 systems (Table 3).

6.2.3 Quantitative Results

Task success. All participants were able to complete all tasks with A-A and A-B within the given time constraint of 10 min. Four participants were not able to complete 1 task in the time allotted (a different task in each case, a total of 4 tasks) using N-A.

	A-A		*A-B*		*N-A*	
Task	μ	σ	μ	σ	μ	σ
1	187	87	**173**	33	448	158
2	**69**	3	92	29	476	55
3	214	67	**193**	49	526	89
4	107	26	**101**	26	342	179
5	**94**	35	112	27	411	164
6	174	36	**171**	48	422	127
AVG	153	71	**142**	52	426	132

Table 2: User Interface Evaluation: Task Completion Time (sec.)

Task completion times are shown in Table 2. One-way ANOVA test ($\alpha = 0.001$) shows statistically significant result ($p < 0.0001$) and Tukey's Multiple Comparison Test showed statistically significant difference between A-A and N-A, as well as between A-B and N-A ($p < 0.0001$). The one-tailed t-test ($\alpha = 0.001$) for all tasks showed that both A-A ($t = 8.9, df = 41$) and A-B ($t = 11, df = 45$) provide statistically significant ($p < 0.0001$) speed improvements when compared to N-A. This corroborates hypothesis **H1**.

Of the 6 websites used in the study, only amazon.com was very familiar to most of the participants - 80% used it on a regular basis (the only other familiar website was ebay.com, with 20%). This resulted in an insignificant improvement in the performance time when using screen-reader without automation (Table 2, task 4, column 5). However, even in this case both A-A and A-B provided statistically significant speedups when compared to N-A. This reinforces hypothesis **H1**.

6.2.4 Post-completion Questionnaire

The SUS questionnaire scores are shown in the first two rows of Table 3. According to Bangor et al. [6] the results for A-B (80), and N-A (78) can be considered "good", while the result for A-A (88) can be considered "excellent".

The ANOVA test between all three systems, as well as the one tailed paired t-test between the A-B and N-A ($p = 0.33, t = 0.44, df = 18$), showed no statistically significant difference. The one tailed paired t-test between the A-A and N-A showed that the improvement was statistically significant ($p = 0.0109 < 0.05, t = 2.5, df = 18$). This corroborates hypothesis **H2**.

After evaluating all three systems, the participants were asked to answer 10 questions comparing the three systems to each other (Table 3). The participants could answer by naming one of the evaluated systems or saying "none of the above". The results show that the A-A was strongly preferred by most participants in all questions (shown in bold: the highest values for positive questions 1,3,5,7,9, and the lowest values for negative questions 2,4,6,8,10). This reinforces the results of the SUS questionnaires. It is notable that the participants scored A-A higher than A-B despite the fact that, in most cases, the participants completed tasks with A-B only marginally faster.

The high deviation values (σ_{SUS}) are, mostly likely, a result of counterbalancing the order of systems, and the fact that SUS questions were asked immediately after each system was evaluated (to ensure the experience was still fresh in the participants' minds). For example, if the subject evaluated A-A before N-A then the latter received much lower scores, reflecting participants' tendency to score each system in the context of prior experience. Likewise,

	A-A	*A-B*	*N-A*
μ_{SUS}	**88.16**	80.13	78.03
σ_{SUS}	**11.24**	17.21	22.62
1. Which system would you prefer to use?	**84.2%**	15.8%	0%
2. Which system did you find the most complex?	**0%**	42.1%	31.6%
3. Which system did you find the easiest to use?	**78.9%**	15.8%	5.3%
4. Which system would need the most support from a technical person?	**0%**	31.6%	26.3%
5. Which system was the most well integrated?	**63.2%**	21.1%	15.8%
6. Which system was the least consistent?	**5.3%**	31.6%	15.8%
7. Which system do you imagine most people would learn to use most quickly?	**78.9%**	15.8%	5.3%
8. Which system did you find the most cumbersome to use?	**0%**	42.1%	21.1%
9. Which system did you feel the most confident using?	**78.9%**	10.5%	10.5%
10. Which system did you feel required you to learn the most before using it?	**0%**	42.1%	26.3%

* The answer "none of the above" is omitted for brevity
* The questionnaire is a reformulated for comparison SUS

Table 3: User Interface Evaluation: Post-Completion Questionnaire

when N-A was evaluated first, the participants assigned it very high scores because, in this case, they compared it to the screen-reader they were used to the most: JAWS. A possible reason for favorable comparison is that blind users strongly associate a screen-reader with the synthesized voice, and we used IVONA, a high quality voice. In one specific example of contextual scoring, one of the participants, having already given N-A the highest scores possible, expressed her desire to score A-A higher than was permitted by the Likert scale.

6.2.5 Qualitative Results

Our observations show that, when performing tasks using N-A, the participants adopted one of three strategies: (a) moving through the webpage, top down, until finding the element they searched for, or, if not successful after some time, (b) jumping to the end of the webpage and sequentially moving bottom up, or (in about 20% of all cases) (c) using advanced shortcuts such as search or iterating through predefined element types (e.g., buttons, links). When successful, the last strategy provided significant speed improvement; when not, the subject had to fall back to one of the two former strategies.

When using A-A or A-B, the participants initially showed signs of hesitation, often pausing before pressing the next shortcut. However, by the second task the participants were becoming substantially more confident. A common mistake made by the participants when using the A-B was to try to use the "up" and "down" keys to navigate away from the suggested element (this is impossible by design, since those shortcuts only navigate between suggestions when the suggestion mode is on). When using A-A the participants sometimes used those keys to become familiar with the neighborhood of the suggested element before proceeding with the task.

A number of users explicitly stated that they preferred to remain always in the same mode (i.e. not have a special suggestion mode), because this meant they did not need to learn subtle differences between the way the mode behaved. In other words, introduction of multiple states may be increasing cognitive load, and may be the main reason for significantly higher usability scores of the A-A. At the same time, A-B may still be a better choice for power-users, because it after a suggestion is executed, the A-B moves the cursor to the next suggestion, allowing the user to press fewer shortcuts. Also, in case of multiple suggestions, the user could use element-specific navigation shortcuts in the suggestion mode to iterate over suggestions of a particular type. For example, pressing B in suggestion mode will iterate only over suggested buttons.

The participants made a large number of suggestions and provided ideas for improving the Assistant interface. For example, when suggesting to change the state of a checkbox, voicing the suggested value after the current value may be confusing because, in this case, users always listen to the last spoken word to determine checkbox status (the same however was not confusing for other form fields).

6.3 Testimonials

We received no negative comments from the participants about the evaluated user interface. Below are a few quotations that exemplify subject's overall reaction:

"It's like automatic bookmarking on steroids."

"If all what people were doing is browsing the web, this would be perfect."

"I haven't seen anything as unique as this program."

"With one tap it brings you right to where you want to go."

"I hope it is coming soon, many blind people could use it; especially when you go to college it can help students with research."

"Pretty innovative idea. . . I am wondering why even sighted people wouldn't find this product useful."

7. RELATED WORK

In this overview, we discuss some representative work on web Automation, while a more detailed review of the web automation tools can be found in [27]. The two main approaches to web automation are handcrafting, e.g., [9] and Programming by Demonstration (PBD) [12].

The handcrafting method requires writing a script to customize the behavior of the browser / screen-reader, or hardcoding the automation instructions into a feature in the browser / screen-reader, e.g., JAWS screen-reader has a shortcut to look up a word in an online dictionary. While handcrafting automation instructions is sometimes facilitated by a user interface [21], it typically requires learning to use the tool(s) and the language for creating the automation instructions; for instance, handcrafting scripts for the JAWS screen-reader [1] requires the user to learn to program in a special scripting language and follow the 180+ page macro creation manual, while the instructions for Window-Eyes screen-reader [2] scripting extend to over 1000 pages.

Programming by demonstration (PBD) is a more user-friendly approach that enables the end-user to "demonstrate" to the automation system what actions need to be done and in what order; this is usually accomplished by recording a macro, which can be later replayed to automate the same sequence of steps [17]. To take this further, a recording made for one webpage can sometimes be dynamically adapted to be used for a similar task on a different webpage or even a different website [8]. Typically, to record a macro

the system needs to know the beginning and the end of the instruction sequence. This requires that user explicitly start the recording, execute the browsing steps flawlessly, terminate the recording, and save it under a recognizable name, all of which require both manual and mental effort. So, to put the effort into creating the macro, the user has to feel that the it will be very useful and will save a lot of time in the future. Smart Bookmarks [16] tries to make this process a little simpler by automatically guessing the beginning of the macro once the user indicates the end of the sequence. Unfortunately, the approach is not 100% accurate and it still carries most of the previously mentioned disadvantages of macros. Even if the user takes the time to record a macro, it is a fixed sequence of steps, so to automate slight deviations from this sequence, the user has to record multiple macros.

The interface described in this paper can be categorized as PBD, and was inspired by the ideas developed in [24, 25, 26, 27]. In contrast to the standard PBD approaches described above, our approach does not use macros altogether; instead, it uses the history of past browsing actions to predict future actions. Automatic form-filling in modern web browsers is another example of PBD that learns from what the user types into web forms and then helps automate form-filling. The proposed model-based approach, however, subsumes form-filling and even makes it more powerful; specifically, it can propose different values for the same form-fields depending on the browsing context of the user, e.g., if the user enters the first name, the model can predict the related last name(s). Furthermore, the proposed model-based approach can automate other browsing actions such as clicking links and buttons, and, in constrast to macros, it can suggest several possible alternatives at each browsing step.

The feasibility of a step-by-step automation of browsing tasks without macros was also explored in [19, 20, 28], where web automation relied on process models constructed from sequences of actions using machine learning techniques such as clustering, classification, and automata learning. The resulting process models had browsing states as nodes and actions as transitions. However, the approach used to construct the process models was not incremental, requiring that the model be rebuilt in order to accommodate any changes, such as learning new transition (or unlearning old ones). Moreover, the approach implied the system's ability to split clickstreams into sequences corresponding to sessions, cluster similar session sequences into similar groups, and then learn the automata, potentially introducing errors in each of those steps (e.g., it is not clear what is a good indicator of a splitting point, and if one exists at all in our domain). Finally, to predict the next browsing step, this approach required the exact match between the current browsing state and a state in the model (which represents a first-order Markovian process), thus limiting the predicting power of the approach in case of deviations from the process.

The model described in this paper is modeling a higher-order Markovian process, making it possible to predict the next step even if the user deviates from previously learned sequences of browsing actions. Futhermore, the model learns to automate browsing steps incrementally, making it possible to insert new and to remove outdated data on the fly without rebuilding the model. The model takes the history of browsing steps as they are without the need to segment the history into sessions.

Related work on the prediction of user browsing behavior also exists in other application domains such as prefetching [23, 13], e-commerce [29], content personalization [7], etc. These applications are only somewhat related because they pursue different goals, e.g., prefetching files or adjusting web page layout. These applications often construct probability graphs (similar to process models) and tend to do that offline; they do not handle form filling, etc. Notably, in [10], sequence alignment is used to find the next link the user could click in a website directory (e.g., Yahoo!). Apart from the difference in the application domain, the described approach also differs in that it aligns the current user session to multiple past sessions, that were previously clustered and organized into clickstream trees. The ensuing limitations are similar to those of the process-model approach: errors of session segmentation, as well as in clustering, and inefficiency of off-line model construction. In contrast, the approach proposed in this paper avoids the compexity and errors of clustering and segmentation, constructs the model incrementally, automates both form-filling and clicking, indroduces modifications to the alignment algorithm, and, rather than using the final result of sequence alignment, it uses the inner-workings of the algorithm (i.e. alignment table) to predict multiple possible steps the user could take.

8. CONCLUSION AND FUTURE WORK

In this paper, we proposed Web Automation Assistant, an interface agent with an accessible non-visual user interface for step-by-step automation of repetitive web browsing tasks. By combining a very carefully crafted user interface, and a sequence-alignment-based prediction model, we were able to improve significantly the user experience and the usability of web browsing for people with vision impairments. The fact that one of the two evaluated user interfaces turned out to be much more usable than the other serves as a reminder of the importance of attention to detail in interface design in general, and accessible interface design in particular. With further R&D effort, the Assistant has the potential to morph into an eyes-free, voice-controlled system, benefiting people both with and without vision impairments.

The main directions of work that we will pursue next include addressing the need of people with vision impairments to find non-interactive content (e.g., articles), and *silent*, dynamic page modifications (e.g., alerts, error notifications), enabling the model to learn from the browsing history of multiple users, purging the model from decayed data, investigating the possibility of accounting for time intervals, and improving the user interface based on the feedback from the usability study presented in this paper.

9. AKNOWLEDGMENTS

This work was developed under a grant from the Department of Education, NIDRR grants number H133S110023 and H133S120067. However, contents do not represent the policy of the Department of Education, and you should not assume endorsement by the Federal Government. We are also grateful to our Accessibility Consultant, Glenn Dausch, for his insightful feedback.

10. REFERENCES

[1] Freedom Scientific: JAWS screen-reader. `http://www.freedomscientific.com/`.

[2] GW Micro: Window-Eyes screen-reader. `http://www.gwmicro.com/`.

[3] NonVisual Desktop Access: NVDA screen-reader. `http://www.nvda-project.org/`.

[4] World Wide Web consortium (W3C): XML path language (XPath), 1999. `http://www.w3.org/TR/xpath`.

[5] IVONA software: IVONA multi-lingual speech synthesis system, 2005. `http://www.ivona.com/`.

[6] Bangor, A., Kortum, P. T., and Miller, J. T. An empirical evaluation of the System Usability Scale. *Int. J. Hum. Comput. Interaction* (2008), 574–594.

[7] Bian, J., Dong, A., He, X., Reddy, S., and Chang, Y. User action interpretation for online content optimization. *Knowledge and Data Engineering, IEEE Transactions on PP*, 99 (2012), 1.

[8] Bigham, J. P., Lau, T., and Nichols, J. Trailblazer: enabling blind users to blaze trails through the web. In *Proceedings of the 14th international conference on Intelligent user interfaces*, IUI '09, ACM (New York, NY, USA, 2009), 177–186.

[9] Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, ACM (New York, NY, USA, 2005), 163–172.

[10] Bose, A., Beemanapalli, K., Srivastava, J., and Sahar, S. Incorporating concept hierarchies into usage mining based recommendations. In *Proceedings of the 8th Knowledge discovery on the web international conference on Advances in web mining and web usage analysis*, WebKDD'06, Springer-Verlag (Berlin, Heidelberg, 2007), 110–126.

[11] Brooke, J. SUS: A quick and dirty usability scale. In *Usability evaluation in industry*, P. W. Jordan, A. Thomas, B. Weerdmeester, and I. McClelland, Eds., Taylor and Francis (1996).

[12] Cypher, A., Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B. A., and Turransky, A., Eds. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA, 1993.

[13] Domènech, J., de la Ossa, B., Sahuquillo, J., Gil, J.-A., and Pont, A. A taxonomy of web prediction algorithms. *Expert Syst. Appl. 39*, 9 (2012), 8496–8502.

[14] Findlater, L., and Gajos, K. Z. Design Space and Evaluation Challenges of Adaptive Graphical User Interfaces. *AI Magazine 30*, 4 (2009), 68–73.

[15] Huang, Z., Eliens, A., van Ballegooij, A., and de Bra, P. A taxonomy of web agents. In *Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on* (2000), 765 –769.

[16] Hupp, D., and Miller, R. C. Smart bookmarks: automatic retroactive macro recording on the web. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, ACM (New York, NY, USA, 2007), 81–90.

[17] Leshed, G., Haber, E. M., Matthews, T., and Lau, T. Coscripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, ACM (New York, NY, USA, 2008), 1719–1728.

[18] Lieberman, H. Autonomous interface agents. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '97, ACM (New York, NY, USA, 1997), 67–74.

[19] Mahmud, J., Borodin, Y., Ramakrishnan, I. V., and Ramakrishnan, C. R. Automated construction of web accessibility models from transaction click-streams. In *Proceedings of the 18th international conference on World Wide Web*, WWW '09, ACM (New York, NY, USA, 2009), 871–880.

[20] Mahmud, J., Sun, Z., Mukherjee, S., and Ramakrishnan, I. Abstract web transactions on handhelds with less tears. In *Proceedings of the Workshop MobEA IV - Empowering the Mobile Web* (2006).

[21] Montoto, P., Pan, A., Raposo, J., Bellas, F., and López, J. Automating navigation sequences in ajax websites. In *Proceedings of the 9th International Conference on Web Engineering*, ICWE '09, Springer-Verlag (Berlin, Heidelberg, 2009), 166–180.

[22] Myers, E. W., and Miller, W. Optimal alignments in linear space. *CABIOS 4* (1988), 11–17.

[23] Padmanabhan, V. N., and Mogul, J. C. Using predictive prefetching to improve World Wide Web latency. *SIGCOMM Comput. Commun. Rev. 26*, 3 (July 1996), 22–36.

[24] Puzis, Y. Accessible web automation interface: a user study. In *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility*, ASSETS '12, ACM (New York, NY, USA, 2012), 291–292.

[25] Puzis, Y. An interface agent for non-visual, accessible web automation. In *Adjunct proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST Adjunct Proceedings '12, ACM (New York, NY, USA, 2012), 55–58.

[26] Puzis, Y., Borodin, E., Ahmed, F., Melnyk, V., and Ramakrishnan, I. V. Guidelines for an accessible web automation interface. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*, ASSETS '11, ACM (New York, NY, USA, 2011), 249–250.

[27] Puzis, Y., Borodin, Y., Ahmed, F., and Ramakrishnan, I. V. An intuitive accessible web automation user interface. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, W4A '12, ACM (New York, NY, USA, 2012), 41:1–41:4.

[28] Sun, Z., Mahmud, J., Ramakrishnan, I. V., and Mukherjee, S. Model-directed web transactions under constrained modalities. *ACM Trans. Web 1*, 3 (Sept. 2007).

[29] Vanesa Aciar, S., Serarols-Tarres, C., Royo-Vela, M., and De la Rosa i Esteva, J. L. Increasing effectiveness in e-commerce: recommendations applying intelligent agents. *International Journal of Business and Systems Research 1*, 1 (01 2007), 81–97.

[30] Waterman, M. S., and Smith, T. F. Identification of common molecular subsequences. *J. Mol. Biol. 147* (1981), 195–197.