Automated Construction of Web Accessibility Models from Transaction Click-streams

Jalal Mahmud * Yevgen Borodin I.V. Ramakrishnan C.R. Ramakrishnan

Department of Computer Science
Stony Brook University
Stony Brook, NY 11794, USA
{borodin, ram, cram}@cs.sunysb.edu

ABSTRACT

Screen readers, the dominant assistive technology used by visually impaired people to access the Web, function by speaking out the content of the screen serially. Using screen readers for conducting online transactions can cause considerable information overload, because transactions, such as shopping and paying bills, typically involve a number of steps spanning several web pages. One can combat this overload by using a transaction model for web accessibility that presents only fragments of web pages that are needed for doing transactions. We can realize such a model by coupling a process automaton, encoding states of a transaction, with concept classifiers that identify page fragments "relevant" to a particular state of the transaction.

In this paper we present a fully automated process that synergistically combines several techniques for transforming *unlabeled* click-stream data generated by transactions into a transaction model. These techniques include web content analysis to partition a web page into segments consisting of semantically related content, contextual analysis of data surrounding clickable objects in a page, and machine learning methods, such as clustering of page segments based on contextual analysis, statistical classification, and automata learning. The use of unlabeled click streams in building transaction models has important benefits:

(i) visually impaired users do not have to depend on sighted users for creating manually labeled training data to construct the models; (ii) it is possible to mine personalized models from unlabeled transaction click-streams associated with sites that visually impaired users visit regularly; (iii) since unlabeled data is relatively easy to obtain, it is feasible to scale up the construction of domain-specific transaction models (e.g., separate models for shopping, airline reservations, bill payments, etc.); (iv) adjusting the performance of deployed models over time with new training data is also doable.

We provide preliminary experimental evidence of the practical effectiveness of both domain-specific, as well as personalized accessibility transaction models built using our approach. Finally, this approach is applicable for building transaction models for mobile devices with limited-size displays, as well as for creating wrappers for information extraction from web sites.

*This work was done when Jalal Mahmud was with the CS Department in SUNY, Stony Brook. His current affiliation is with IBM Almaden Research Center, 650 Harry Rd, San Jose, CA; email: jumahmud@us.ibm.com

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Search and Retrieval; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms

Algorithms, Design, Human Factors, Experimentation

Keywords

Context, Web Transaction, Process Model, Machine Learning

1. INTRODUCTION

The Web has evolved into a dominant digital medium for conducting online transactions, a notion that broadly encompasses activities such as shopping, paying bills, making travel plans, etc. Figure 1 is an illustrative example of a web transaction, where the user selects the *SanDisk* MP3 player category (pointed to by the arrow in Figure 1(a)), chooses the first MP3 player from the list of available players in Figure 1(b), then adds the selected item to the cart in Figure 1(c) and, finally, does the checkout, as shown in Figure 1(d).

As can be seen from the above example, web transactions typically involve a number of steps spanning several web pages. With graphical web browsers, sighted users can quickly go through the transaction steps. However, non-visual transactions pose serious challenges to blind individuals, because screen readers (e.g., JAWS [1], Windows Eyes [11]), the dominant assistive technology used by blind users, typically narrate the content of web pages sequentially. Although screen-readers provide shortcuts to move backward and forward within a page, they offer no help in finding relevant information and, thus, expose their users to considerable information overload.

Given the same aural screen-reader interface, the best way to facilitate a non-visual web transaction is to give blind users a way to quickly access page fragments and action-buttons that would be relevant in the corresponding steps of the transaction. We will refer to such fragments and buttons as *concepts*. To illustrate, after a category is selected in the first step of the transaction in Figure 1(a), the page fragments corresponding to the item list (showing a taxonomy of items in that category) and the search form (for doing a new search if needed) are among the most relevant concepts for the next step of the transaction in step (b) of Figure 1.

In [30] we had developed a *transaction model* for web accessibility to identify, extract, and aurally render only the "relevant" page fragments in each step of a transaction. The use of a transaction model for improving the accessibility of web transactions under-

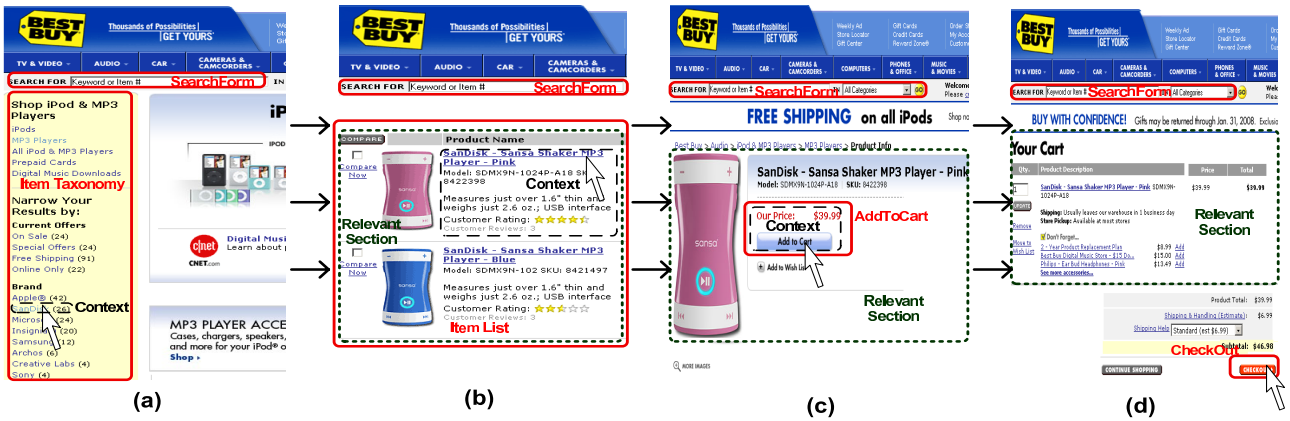

Figure 1: Example Web Transaction in BestBuy.com

lies our previous work in [30] as well as the work reported in this paper.

The transaction model uses a process automaton, in which each state represents a step of a transaction. At each step, the user is presented with a set of concepts relevant to a transaction at that step. A concept classifier is used to identify such instances of concepts whenever they are present in web pages.

Figure 2 illustrates a process automaton for online shopping, where s_1 and s_5 are the start and accept states respectively. Note that each edge connecting a pair of states is a possible action/transition (e.g., select_category, select_item, and submit_searchform in s_1). Each state in the process automaton also specifies the set of transaction-specific concepts it expects to find in any web page that is given as the state's input. For example, to identify the "Item Taxonomy" and the "Search Form" in Figure 1(a), the start state s_1 has to expect the corresponding concepts. The rest of the content will be filtered out.

The transaction example in Figure 1 can be mapped to the process automaton as follows. In s_1 the user is prompted to select a category; choosing SanDisk takes the transaction back to s_1 where all models of SanDisk MP3 players are spoken out. Picking the first item from the list takes the transaction to s_2, where the details of the item are presented. Adding an item to cart takes the transaction to s_4. The checkout action completes the transaction, finishing in state s_5.

In contrast to our current approach (to be described later on), in our previous work [30] we built the automata using preprocessed transaction sequences. In the preprocessing stage every transaction step in the sequence was assigned a user-defined "semantic" label. For instance, $< select\_category, select\_item, add\_to\_cart, check\_out >$ is one such sequence corresponding to the transaction illustrated in Figure 1. Observe that the semantics associated with labels, $select\_category, select\_item$, etc., corresponds to the operations performed on the concept instances. Assigning such semantic labels is by and large a *manual* process. Another place involving manual effort was in the construction of supervised concept classifiers. In particular, we manually collected sample web page fragments corresponding to concept instances from different web sites. Each concept was associated with a user-defined concept name and their instances were assigned the corresponding name (e.g., the fragment within the solid rectangle in fig 1(a) is assigned the concept name: "taxonomy"). We extracted different features (e.g.,, words) from such labeled concept instances to train the concept classifiers.

Having to rely *exclusively* on user-defined semantic labels and

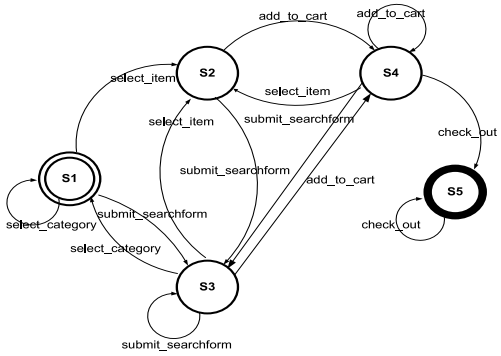

Figure 2: Process Automaton Example

concept names for supervised construction of transaction models has serious shortcomings:

- Firstly, blind users will have to depend on sighted users for such training data, which makes it impractical to create personalized models for the sites that each individual blind user visits regularly.
- Secondly, it is not feasible to update and retrain already deployed models with new training data over time.
- Lastly, because such user-defined labeled data is expensive to obtain, it is not feasible to scale up the construction of domain-specific transaction models (e.g., shopping, airline reservations, bill payments, etc.).

Thus, a model construction process that is not dependent on user-defined labeled training data is highly desirable.

Contributions of this Paper:

In this paper we present such a fully automated model construction process. Specifically, given a collection of raw (i.e. unlabeled) click streams associated with transactions, we construct the transaction model by synergistically combining several techniques that include: web content analysis for partitioning a web page into segments consisting of semantically related content elements, contextual analysis of data surrounding clickable elements in a page, and machine learning techniques such as clustering of page segments based on contextual analysis, statistical classification, and automata learning. Informally, a click-stream is a sequence of browser-generated

$< select\_category(145AB2D1), select\_item(05F354A1), add\_to\_cart(731DA231), check\_out(873A11F2) >$
$< select\_category(165AC2E1), 06F344A2, add\_to\_cart(3A1DB241), 47CA13F5 >$
$< 16AAC2D2, 06F3A4A2, 23AD2232, 8A3111E1 >$

Table 1: Click-stream Examples

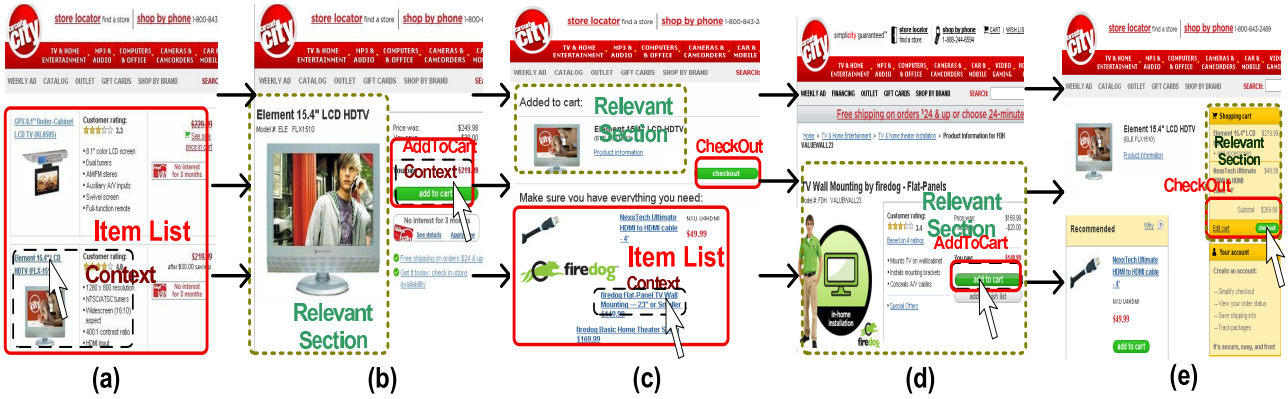

Figure 3: Example Web Transaction in CircuitCity.com

unique IDs belonging to actionable page elements that were acted upon by the user during a web transaction. For example, Table 1 shows a set of partially labeled click-streams associated with the transaction in Figure 1, where a labeled click-stream element is of the form *label(page element's ID)* (e.g., *select_item(05F354A1)*) whereas the unlabeled page elements only have IDs. Observe that all elements are labeled in the 1st row, only one is labeled in the 2nd row, whereas none are labeled in the 3rd row.

A unique aspect of the transformation process – from "raw" clicks streams to accessibility models – is that the click-streams, that serve as training data for the learning methods, need not have any user-defined labels. More generally, the process can operate with *partially labeled* click-stream data, where some (including *all*) user-defined labels could be missing. As a result of using partially labeled data for creating concept classes and process automata, some (including all) classes and state transitions may remain unlabeled. However, assigning user-defined labels to concepts and transitions, although useful, is not critical for conducting transactions non-visually. For example, if the concept class denoting all taxonomies remains unlabeled, the concept classifier for taxonomies will still be able to identify the item taxonomy segment in Figure 1 (a) and read out its content to the user. Not having to rely on (manually) labeled training data makes our method scalable, facilitates the construction of personalized transaction models, and makes it feasible to retrain and adjust the already deployed models, so as to improve their performance over time.

While the focus of this paper is on web accessibility, it is worthwhile pointing out that the transformation process has a broader application scope. Specifically it can be used to build transaction models for mobile devices with limited-size displays, as well as for creating wrappers for information extraction from web sites.

The rest of the paper is organized as follows: Section 2 has the technical preliminaries needed for the development of our construction process in Section 3. An envisioned assistive browser with a transaction model is described in Section 4. Preliminary experimental evidence of the practical effectiveness of both domain-specific and personalized transaction models built using our new framework appears in Section 5; related work is in Section 6 and we conclude the paper in Section 7.

2. TECHNICAL PRELIMINARIES

We use the term *web object* to refer to any element in a web page that has a unique address, e.g., link, button, text elements, etc. We say that a web object is *clickable* whenever clicking on the object results in an action, e.g., clicking on a link results in following the link to another web page.

Although web pages have a lot of content, only some content is important in a transaction, e.g., "product description", "checkout", etc. Over time, the web content has evolved into standard categories, e.g., "taxonomy", "search result", etc. We use the notion of *concept class* to refer to a category of web content relevant for conducting web transactions. A concept class has the same meaning across many different web sites, e.g., the *Add-to-Cart* buttons in Figures 1 and 3 belong to the same concept class.

We assume that each concept class will have a unique concept label and an associated concept operation. Table 3 shows some examples of (user) assigned concept names and their associated operations for the shopping domain (In general, manually assigned concept names and operations are assumed to be user-defined). Suppose C is the label for a concept and O is the name of the operation associated with the concept, then the functions $Opr(C)$ will return O and $Opr^{-1}(O)$ will return C.

The following notions lay the groundwork for identifying concepts in web pages from click-streams.

A *geometric segment* is a web page fragment consisting of content elements that share the same geometric alignment in the page. e.g., the two solid rectangles enclosing the "Item Taxonomy" and "Item List" elements in Figure 1 (a-b). Observe how the elements within them share the same geometric alignment. The alignment may imply "semantic" relationship between the content elements. A number of papers have exploited this observation to partition web pages into geometric segments containing such semantically related elements (e.g., [22, 34]).

We define the *context* of a clickable object *Obj* to be the text around the *Obj* that maintains the same *topic* as the text of the *Obj*. To collect the context, a simple topic-detection algorithm based on cosine similarity is applied to the text surrounding the object within the geometric segment containing the object, as described in [20]. For example, the dashed boxes in Figures 1 (b) and (c) enclose

Click-stream	Website Name
$< select\_category(145AB2D1), 05F354A1, 731DA231, 873A11F2 >$	"BestBuy"
$< select\_item(01A561E2), 81121F12, 02141F34, 02141F11, 023A21A2 >$	"CircuitCity"

Table 2: Training Click-streams

the context of the corresponding links pointed to by the arrows. The page segment enclosed by the dotted rectangle is a *Context Segment*.

Typically, two broad classes of concepts arise in transactions: simple concepts, represented by a single HTML object such as button or link (e.g., "Log out"), and structured concepts, capturing more content that often encapsulates other simple concepts (e.g., "Item Detail" concept encapsulating the "Add To Cart" concept). Structured concepts follow certain patterns, e.g., taxonomies, list of email, list of products, headline news with summary snippets, search results, etc.

We define a *concept segment* to be either a context segment or a geometric segment consisting of a collection of similar context segments forming a pattern. The former is a simple concept and the latter is a structured concept.

A *concept instance* is a concept segment that is classified to a concept class. In Figure 1 the concept instances are all shown using solid rectangles. There are four instances of the "Search Form" concept and single instances of "Item Taxonomy", "Item List", "Add to Cart", and "Check Out" concepts.

A *click-stream* is a non-empty sequence of objects that were clicked during the transaction process. In a *partially labeled* click-stream some objects may be labeled with the corresponding concept operation . Table 1 shows examples of partially labeled click-streams. We call a fully labeled click-stream (e.g., the 1st row in Table 1) a *transaction sequence*.

3. BUILDING TRANSACTION MODELS

The overall objective of the construction process is to transform the set of click-streams into a transaction model. Let us examine the issues involved in this process.

Recall that each state in the process automaton (Figure 2) specifies a set of transaction-specific concepts it expects to find in any web page that is given as the state's input. This means that we have to automatically associate a clicked object in a click-stream with the corresponding concept class and label the object. To solve the problem, we use the context of an object to identify the concept segment containing this object. Objects in click-streams may belong to the same concept class, for example, in Table 1 the clicks on the objects with IDs 731DA231 and 3A1DB241 correspond to doing an "add to cart" operation. Hence, the concept segments containing these two objects should belong to the same "add to cart" concept class. Our solution for identifying the concept classes associated with objects is to cluster the segments containing these objects. Segments with high degree of similarity will belong to the same cluster. Each such cluster will correspond to a concept class. If any one of these concept segments has a user-defined label assigned a priori, the cluster containing those segments is assigned the same label; by definition a cluster cannot have conflicting labels. Otherwise, the cluster is assigned a unique machine-generated label and an operation name. Using these labels we can then transform a click-stream into a transaction sequence, from which we learn the process automata. Using a generic set of features from the segments in a cluster we automatically train a concept classifier. All of these steps are summarized in Figure 4.

We use the following notations to outline the algorithms that

Concept Name.	Operation Name
"Item_List"	"select_item"
"Item_Taxonomy"	"select_category"

Table 3: Initial Table of User-defined Labels

Concept Name.	Operation Name
"Item_List"	"select_item"
"Item_Taxonomy"	"select_category"
"L_1"	"l_1"
"L_2"	"l_2"

Table 4: Updated Table with Machine Generated Labels

correspond to the steps. Let D denote the set of partially labeled click-streams of *completed* transactions, W be the set of web pages containing the objects in the click-streams and T be a table of user-defined labels assigned to some concepts and associated concept operations a priori. Note that T can be empty, corresponding to unlabeled click-streams. The construction process will add newly identified concept classes to this table and assign them unique machine generated labels. We will use *getConop(Obj)* to get the operation label on the object *Obj* in the click-stream. It will return null if there is no label associated with the *Obj*.

The input to the process is the triple $< D, W, T >$. The output is a process automaton and concept classifiers for the concepts constructed from D.

All the steps in Figure 4 are fully automatic. In Step 1 we partition the web pages into geometric segments and extract their features. In Step 2 we identify the concept segments containing the objects in the click-stream, which requires the context of the object and the geometric segment containing the segment (see definitions in Section 2). In this step we also infer the labels of those concept segments whose corresponding click-stream objects are labeled in D. Based on the features extracted in Step 1 we cluster the concept segments into concept classes in Step 3. In Step 4 we assign (machine generated) concept labels to each unlabeled class. The process splits into two separate branches at this point. In Step 5 we train classifiers for each of the concepts using the clustered segments as training data. In Step 6 we use the concept labels from Step 4 to transform the click-stream into a set of transaction sequences. In the Step 7 we use the transaction sequences as training data to learn the process automaton. The technical details of each of these steps now follows. We will use Table 2 and 3 to illustrate the steps.

Algorithm *IdentifyConceptSegment*
Input: Obj: a WebObject in a Web Page
Output: *ConceptSegment*: A Concept Segment
1. $GeometricSegment \leftarrow$ Geometric Segment Containing the WebObject
2. Extract Features from the $GeometricSegment$
3. Identify if any pattern is repeated in the $GeometricSegment$
4. **if** $GeometricSegment$ has Repeated Context Pattern
5. **then** $ConceptSegment \leftarrow GeometricSegment$
6. **else** $ConceptSegment \leftarrow$ Context Segment Containing the WebObject
7. $operationLabel \leftarrow$ getConop(Obj)

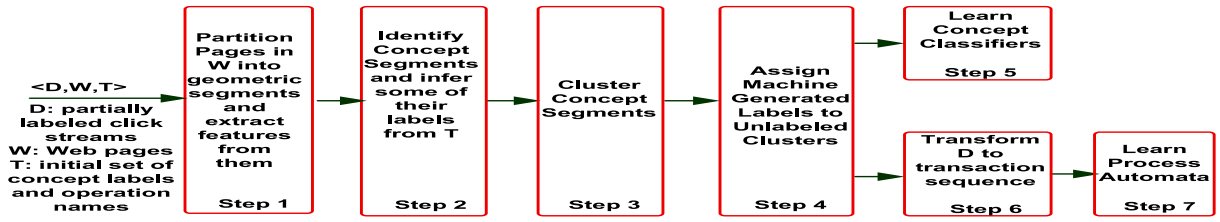

Figure 4: From Click-streams to Models: Transformation Process

Figure 5: Initial Clusters Containing Labeled and Unlabeled Concept Segments

8. $conceptLabel \leftarrow Opr^{-1}(operationLabel)$
9. **if** $conceptLabel \neq$ NULL
10. **then** $ConceptSegment.ConceptLabel \leftarrow conceptLabel$
11. **else** $ConceptSegment.\ ConceptLabel \leftarrow$ NULL
12. **return** *ConceptSegment*

Step 1: Partitioning and Feature Extraction:

This step begins by partitioning every web page in W into geometric segments. Three kinds of features are extracted from each of the segments: **Word Features** are words in the text appearing in the segment. **Linguistic Features** are bigrams, trigrams, and their stemmed [26] counterparts in the segment's text. **Pattern Features** are features representing the visual presentation of the content in the segment. e.g: in Figure 1 (a), some of the pattern features extracted from the geometric segment are "text", "link", "$text(link)^+$", etc. Note that the features computed from concept segment are domain independent and hence generic.

Step 2: Identifying Concept Segments:

In this step the concept segments containing the objects in the click-stream are identified. Algorithm *IdentifyConceptSegment* is a high-level sketch of this process which essentially is an implementation of the definition of a concept segment based on the idea of context of objects. Specifically lines 6 and 5 correspond to simple and structured concepts respectively. To recognize them as such we make use of the pattern features extracted from the segments in the previous step. e.g.,in Figures 1, 3 the sections enclosed by the solid rectangle correspond to concept segments identified in this step.

In line 7, we invoke the function $getConop$ on an object Obj in the click-stream in order to infer its operation label. If it is non-null then we retrieve its associated concept name in line 8 from T and assign it to the segment; otherwise the segment remains unlabeled.

Running the algorithm on the streams in Table 2 will result in labeling two concept segments, namely, 'Item Taxonomy" and "Item List" in Figure 5. The others remain unlabeled.

Step 3: Clustering Concept Segments:

In this step we cluster the concept segments. At the end of this step all the concept segments in a single cluster are said to belong to the same concept class. Towards this goal we use the well known notion of Jaccard's similarity [15] and the notions of intra-cluster, inter-cluster similarity and quality metrics for clusters developed in [29].

Jaccard similarity computes the similarity between any pair of concept segments. Let S_i and S_j be two such segments with feature sets $F(S_i)$ and $F(S_j)$ respectively. This similarity, denoted $J(S_i, S_j)$, is defined as:

$$J(S_i, S_j) = \frac{|F(S_i) \cap F(S_j)|}{|F(S_i) \cup F(S_j)|} \quad (1)$$

Intra-Cluster Similarity computes the measure of similarity of the concept segments in a cluster. Let C be a cluster with concept segments S_1, S_2, ..., S_n. Then $Intra(C)$, the intra-cluster similarity value for C is defined to be:

$$Intra(C) = \frac{1}{(n-1) \cdot n} \sum_i \sum_j J(S_i, S_j) \quad (2)$$

Inter-Cluster Similarity computes the measure of similarity between two clusters. Let C_i, C_j be two clusters and S_{i_1}, S_{i_2}, ..., $S_{i_{n_i}}$ be the concept segments in cluster C_i, and S_{j_1}, S_{j_2}, ..., $S_{j_{n_j}}$ be the concept segments in cluster C_j. Then $Inter(C_i, C_j)$, the inter-cluster similarity value for this cluster pair is defined to be:

$$Inter(C_i, C_j) = \frac{1}{n_i \cdot n_j} \sum_m \sum_n J(S(i_m), S(j_n)) \quad (3)$$

We say that a cluster is *trivial* if it has only one concept segment. We do not fix the total number of clusters a priori. Instead, we use a quality metric to measure the goodness of clustering before merging the most similar clusters and return the set of clusters that yields the highest quality value. Formally, suppose there are n concept segments and C_1, C_2, ..., C_k are k non-trivial clusters. Furthermore, $\forall i, j$ let Intra(C_i) and Inter(C_j, C_j) denote respectively intra- and inter-cluster similarity for C_i and the pair (C_i, C_j). Then *quality of clustering*, denoted ϕ^{Q}, is defined as: [29]:

$$\phi^{\mathrm{Q}} = 1 - \frac{\sum_{i=1}^{k} \frac{n_i}{n-n_i} \sum_{j \in \{1,\ldots,i-1,i+1,\ldots,k\}} n_j . \mathrm{Inter}(C_i, C_j)}{\sum_{i=1}^{k} \mathrm{Intra}(C_i)} \quad (4)$$

For trivial clusters this quality metric is undefined. *ClusterConceptSegments* is a sketch of the clustering algorithm. Lines 1-3 are the initialization steps: Line 1 builds the initial clusters from the set of concept segments. If the concept segment is already labeled in Step 2 with a concept name then the concept segment is added to the cluster with the same label. Otherwise a new cluster is created and the concept segment is added to the new cluster. Figure 5 shows the initial clusters. The initial clusters form the best cluster to start with.

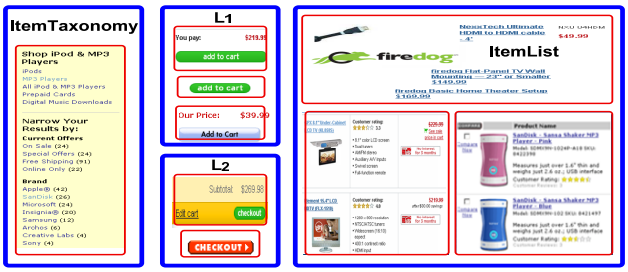

Figure 6: Clusters Returned by the Algorithm

Lines 4 to 19 make up the main clustering loop. In each iteration inter-cluster similarity values are computed and the pair of clusters with the highest inter-cluster similarity value is identified (line 5). For merge to take place at least one of the two clusters in the pair must be unlabeled (line 6). This will ensure that we do not merge two clusters with different labels (e.g. "add to cart" cluster with "check out"). Lines 7-11 does the merge. If one of the cluster is labeled then its label becomes the label of the merged cluster otherwise it remains unlabeled. The quality value of the resulting updated cluster set following the merge is recomputed (line 12). If it is an improvement over the previous best then this cluster set becomes the current best. The algorithm continues until all the clusters are merged into a single cluster or there is no unlabeled cluster (line 19). The algorithm returns the clusters ($BestClusters$) with the highest quality value (line 20). Figure 6 shows the final clusters returned by our algorithm.

Algorithm *ClusterConceptSegments*
Input: $Segments$: A Set of Concept Segments
Output: *Clusters*: A Set of Clusters Containing Concept Segments
1. $Clusters \leftarrow$ Build Initial Clusters from $Segments$
2. $Quality \leftarrow$ Compute Quality of Clustering from $Clusters$
3. $BestClusters \leftarrow Clusters$
4. **repeat**
5. Identify the pair of clusters ($Cluster_1$, $Cluster_2$) with highest inter-cluster similarity
6. **if** ($Cluster_1.Label$ = NULL **or** $Cluster_2.Label$ = NULL)
7. Insert segments from $Cluster_2$ into $Cluster_1$
8. **if** ($Cluster_1.label$ = NULL)
9. **then** $Cluster_1.label \leftarrow Cluster_2.label$
10. Remove $Cluster_2$ from $Clusters$
11. $Clusters.size \leftarrow Clusters.size$ -1
12. $CurrentQuality \leftarrow$ Compute quality of updated $Clusters$
13. **if** ($Quality = UNDEFINED$ and
14. $CurrentQuality \neq UNDEFINED$)
15. **or** ($CurrentQuality > Quality$)
16. **then** $Quality \leftarrow CurrentQuality$
17. $BestClusters \leftarrow Clusters$
18. $NumUnlabeled \leftarrow$ No. of Unlabeled Clusters in $Clusters$
19. **until** $Clusters.Size$ = 1 **or** $NumUnlabeled$ = 0
20. **return** *BestClusters*

From the best cluster set we remove spurious clusters: trivial clusters and those with low intra-cluster similarity values. The threshold is set a priori from experimentation. Of course one can also automatically learn it using validation sets.

Step 4: Labeling of Unlabeled Clusters (Unsupervised):
Each cluster in the final set returned by the algorithm in the previous step is a distinct concept class. These are the concept classes that are generated from completed transaction click-streams. Some or all of the clusters in the final set may be unlabeled. In this step we assign a distinct machine generated concept label to each unlabeled cluster. We add these new labels to the table T and also generate an operation label to be uniquely associated with each of these concept label.

For example, at the end of step 4, the unlabeled clusters in Figure 6 are automatically assigned the labels "L_1" and "L_2". The associated concept operation names are "l_1" and "l_2" respectively. Table T is updated with these machine generated labels as shown in Table 4. Every segment in a cluster is assigned the concept label of its cluster.

Step 5: Learning Concept Classifiers (Unsupervised):
In the previous step all the concept classes relevant for transactions were mined and labeled. In this step we automatically learn SVM-based classifiers [33] for each concept class. The segments in a concept class serve as the labeled training examples. Recall that their features were extracted in Step 1. Note that the labels for the concept segments were either inferred in Step 2 or automatically generated in the previous step.

Step 6: Generating Transaction Sequences:
Recall that in Step 2 the concept segments in web pages containing the click-stream objects in D were all identified and in Step 4 they were all assigned concept names. In this step we label the objects in D with concept operations and thereby transform them into transaction sequences. To label an unlabeled object in a click stream, we apply the function Opr whose parameter is the label of the concept segment containing the object. This will retrieve the corresponding operation name from updated T with which we label the object. Table 5 shows the transaction sequences resulting from applying this step to Table 2.

Step 7: Learning Process Automaton (Unsupervised):
In this step we learn the process automaton from the transaction sequences generated in the previous step. In general, DFA learning requires sizable number of negative examples which are relatively more difficult to obtain especially from logs of user activities (see [23]). Instead, we propose a simple learning algorithm from positive examples only, in particular the completed transaction sequences generated in the previous step. It is motivated by the following observation:

If a substring of operations in a completed transaction repeats consecutively then either deleting or inserting one or more of the repeats will also result in a completed transaction sequence. For example, deleting the repeat of $select\_category.$ $select\_item$ from $select\_category.select\_item.select\_category.$ $select\_item.add\_to\_cart.check\_out$ or inserting additional repeats of $select\_category.select\_item$ to $select\_category.select\_item.select\_category.$ $select\_item.add\_to\_cart.check\_out$ will also represent completed transaction. So given a set of completed transaction sequences the aforementioned insert and delete operations allow a limited degree of generalization. We will learn a process automation to accept these kinds of generalized sequences from a training set T of completed transaction sequences. The details are as follows:

DEFINITION 1 (LANGUAGE OF TRANSACTION SEQUENCES). *Given a training set T, the language of transaction sequences, denoted by $\mathcal{A}(T)$ is the* smallest *set such that:*

- $T \subseteq \mathcal{A}(T)$, *and*
- *for all $x \in \mathcal{A}(T)$ such that $x = pmms$ (p is the prefix,*

s the suffix and m the repeated middle, all possibly empty), $pm^k s \in \mathcal{A}(T)$ for every $k > 0$.

Note that $\mathcal{A}(T)$ generalizes T, and is the language we seek to learn from T. In particular, we generalize T such that any consecutively repeating substring in T is now permitted to repeat an arbitrary number of times. The language $\mathcal{A}(T)$ has an important property: it is closed with respect to training sets in the sense that adding any string in the language to the training set does not change the language. Formally,

THEOREM 1 (CLOSURE). *Let T be a given training set and $\mathcal{A}(T)$ be the corresponding language of transaction sequences. Then, for all S such that $T \subseteq S \subseteq \mathcal{A}(T)$, $\mathcal{A}(S) = \mathcal{A}(T)$.*

The proof of this property follows from the monotonicity of $\mathcal{A}$: i.e. if $T \subseteq S$ then $\mathcal{A}(T) \subseteq \mathcal{A}(S)$. The property of closure indicates "stability" of the learned language since no string in the language could have been added to the original training set to construct a different (more general) language.

The definition of $\mathcal{A}$ is does not directly give a procedure to construct an automaton that accepts $\mathcal{A}(T)$. We now outline such a procedure.

DEFINITION 2. *Let $\mathcal{R}(T)$ be the set of regular expressions over the alphabet of T, defined as follows:*

- $T \subseteq \mathcal{R}(T)$
- $\forall x \in T$ *such that* $x = pmms$, $pm^+s \in \mathcal{R}(T)$.

Note that $\mathcal{R}(T)$ is a finite set of regular expressions (REs); in particular, if the largest sequence in T is of length k, then $|\mathcal{R}(T)| = O(k^2|T|)$, and the largest regular expression in $\mathcal{R}(T)$ is of length k^2 or less.

Let $\mathcal{L}(r)$ denote the language of a regular expression r. The language of a set of regular expressions is the union of the languages of each of its elements: i.e. if R is a set of regular expressions, then $\mathcal{L}(R) = \cup_{r \in R} \mathcal{L}(r)$.

The language of regular expressions $\mathcal{R}(T)$ constructed from the training set is identical to $\mathcal{A}(T)$, the language of transaction sequences learned from T, as formally stated below.

THEOREM 2. *For all sets of training sequences T, $\mathcal{A}(T) = \mathcal{L}(\mathcal{R}(T))$.*

The above theorem can be proved by considering the usual least fixed point (iterative) construction of $\mathcal{A}(T)$, and showing that the least fixed point computation will converge in two steps to $\mathcal{L}(\mathcal{R}(T))$.

Note that $\mathcal{R}(T)$ gives us an effective procedure for constructing the transaction automaton. For each regular expression in $\mathcal{R}(T)$ we construct the corresponding nondeterministic finite automaton (NDFA) using Thomson's construction [13]. The automaton for $\mathcal{L}(\mathcal{R}(T))$ is simply the union of all the individual automata. Based on the argument above on the size of $\mathcal{R}(T)$ and Thomson's construction, it follows that if k is the length of the longest sequence in T, then the automaton for $\mathcal{A}(T)$ thus constructed is of size $O(k^4|T|)$, and can be constructed in time $O(k^4|T|)$. Finding more efficient construction algorithms and building smaller automata are topics of future research.

Running this algorithm on Table 5 results in the REs shown in Table 6. Figure 7 is the automaton for these REs. Either the NDFA constructed from the REs or its equivalent DFA can be used at runtime to guide the transaction. Use of a NDFA will increase the information overload since concept instances corresponding to a set of states will be presented. This can be avoided with a DFA but constructing it from a NDFA can be computationally expensive.

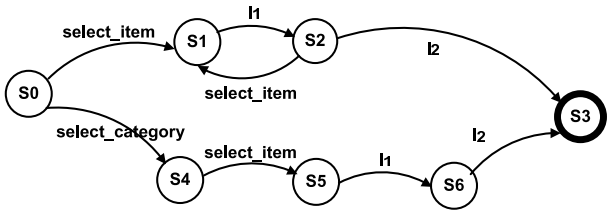

Figure 7: Learned Automata from Sequences in Table 5

$< select\_category, select\_item, l_1, l_2 >$
$< select\_item, l_1, select\_item, l_1, l_2 >$

Table 5: Training Transaction Sequences

4. INTEGRATED ASSISTIVE BROWSER

We have developed HearSay, a context-directed non-visual browser, which identifies and presents relevant information to the users as they navigate from one page to another [20]. The modular design of the HearSay system accommodates plugging in components such as the transaction processing component described in this paper. With this add-on HearSay will transparently identify the concepts in web pages and map them to the most similar state of the process automaton, where appropriate. At any time, HearSay users will be able to turn on the transaction support layer, and, in this way, attain quick access to the concept instances that may be present in their currently opened web pages. It will allow them to navigate between and within concepts using the regular navigation shortcuts. Just as easily, users will be able to return to the original browsing mode and explore the content around the concepts. If the transaction model fails to classify some concept instances or they are labeled with random machine-generated labels, the users will still be able to proceed with the transaction since HearSay reads out the actual contents of the segment. To access the content that is not associated with any operations, e.g., "Item Detail" (detailed description of an item in a web page), which is not represented in our model, the transaction support layer can be combined with the context-browsing layer, which helps identify such segments [21].

Coming back to the example displayed in Figure 1 (a), the transaction support module will match the web page to S_0, the beginning state of the learned process automaton (a fragment of which is shown in Figure 7). The corresponding interface layer will contain the "Item Taxonomy" and the "Search Form" concept instances. The user will then be able to press a special shortcut to navigate between the concepts, or simply use regular navigation shortcuts to skip to and follow the link of interest, indicated by the arrow. On the next page, in Figure 1 (b), the process automaton will go to state S_4 as a result of the *select_category* operation; and, thanks to context-directed browsing module, HearSay will start reading the "Item List". With both context-directed browsing and transaction support layers turned on, the user will be able to access both "Search Form" and "Item List" concept instances.

After following the link to the page shown in Figure 1 (c), HearSay will start reading from the product description, while the underlying process automaton takes the state transition from S_4 to S_5 on "select_item" operation. Since the training data did not contain a

$< select\_category, select\_item, l_1, l_2 >$
$< (select\_item, l_1)^+, l_2 >$

Table 6: Regular Expressions from Table 5

user-defined label for the "Add To Cart" concept, the concept classifier automatically assigned "l_1" to the "add to cart" button. But note that the user will have no problem understanding the meaning of the button, because HearSay always reads out the actual button captions. Finally, by pressing the "add to cart' button', the user shifts the process automaton to state S_6. Figure 1 (d) shows the last page of the transaction, in which HearSay's context-directed browsing algorithm helps find the section containing the content of the cart, and the process automaton helps identify the "Search Form" and the "Check Out" concept instances. Having reached the "Submit" button and having realized that the shipping option is not accessible through the combination of context and transaction layers, the user can momentarily return to the original view and examine the content around the submit button to find and choose the shipping option, and successfully complete the transaction quite quickly.

5. EXPERIMENTAL EVALUATION

Since we had done user studies on the efficiency of transaction models in [30], the evaluation here only focuses on the quantitative performance of the clustering algorithms and the models, namely, concept classifiers and process automaton. Towards that we manually collected and labeled over *300* click-streams from *36* online shopping web sites for books, electronics, and office supplies.

Performance of Clustering

For evaluating the clustering algorithm, we generated 10 data sets $T_0, T_{10}, \ldots, T_{100}$, where the subscript indicates the percentage of manual labels, i.e., dataset T_0 was completely unlabeled, while concept segments in dataset T_{100} were fully labeled. Clusters constructed from T_{100} were used as the validation set to compare the performance of clustering over the other nine data sets. Let $Cluster_{user}$ denote this labeled cluster set constructed from T_{100} (This is straightforward.). We construct all possible pairs of concept segments in $Cluster_{user}$ ($N \cdot (\mathrm{N}-1)/2$, N is the total number of segments).
Let M_{total} be the total number of segment pairs in $Cluster_{user}$ where both of the segments in the pair belong to the same cluster.

Now we do the following for each test data set: Let $Cluster_{algo}$ denote the clusters constructed for a test data set by our algorithm. Let $M_{correct}$ denote the total number of pairs where both segments appear in the same cluster in both $Cluster_{user}$ and $Cluster_{algo}$. Let $M_{incorrect}$ denote the total number of pairs where a pair appears in the same cluster in $Cluster_{algo}$, but in different clusters in $Cluster_{user}$. Then, the recall of the clustering algorithm is $M_{correct}$ / M_{total}, and the precision is $M_{correct}$/ ($M_{correct}$ + $M_{incorrect}$). F-measure is calculated by taking the harmonic mean of the recall and precision. Figure 8 shows the F-measure variation for the data sets.

Performance of Transaction Models

We evaluated the performance of the process automaton and concept classifiers created for each data set. We divided each of them into training (90%) and testing (10%) sets and did a standard 10-fold cross validation. Let N_{total} denote the total number of completed transaction sequences, $N_{correct}$ denote the number of completed transactions accepted by the automaton, $N_{incorrect}$ denote the number of incomplete transactions accepted by the automaton Then, the recall of the process automata learning algorithm is $N_{correct}$ / N_{total}, and the precision is $N_{correct}$/ ($N_{correct}$ + $N_{incorrect}$).

Let $C^i{}_{total}$ denote the actual number of segments which are instances of concept i, $C^i{}_{correct}$ denote the number of correctly labeled concept segments for that concept, $C^i{}_{incorrect}$ denote the number of segments labeled erroneously as instances of that concept, Then, the recall of the concept classifier for concept i is $C^i{}_{correct}$ / $C^i{}_{total}$, and the precision is $C^i{}_{correct}$/ ($C^i{}_{correct}$ + $C^i{}_{incorrect}$). These values are averaged over each concept to get overall recall/precision.

Figure 9 and 10 show the obtained F-measures (harmonic mean of recall/precision) for each of the data set.

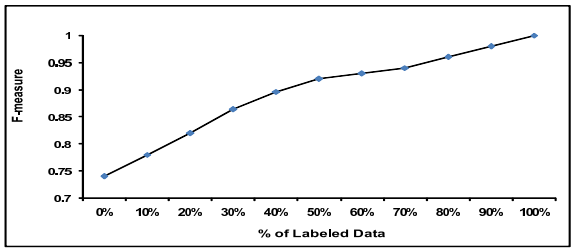

Figure 8: Performance of Clustering

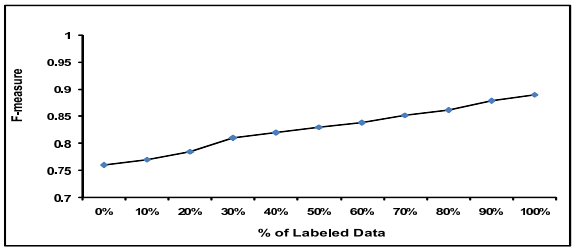

Figure 9: Performance of Process Automata Learning

Performance of Personalized Transaction Models

We constructed personalized transaction models separately for each of "Amazon.com", "OfficeMax.com", and "BN.com" web sites. By repeating the same procedures to determine the performance of clustering and transaction models, we consistently observed 10% improvement (over generalized models) in the performance of clustering algorithm (starting with 84.5% for unlabeled data), 12.5% improvement (over generalized models) of process model learning (starting with 88.5% for unlabeled data), and 9% improvement (over generalized models) of concept identification (starting with 82% for unlabeled data).

Discussion of the Experimental Results

Observe from Figure 8 that with no labeled data at all, the accuracy of clustering was already above 70%. And the performance steadily improved when manually labeled data became available, e.g., 90% accuracy with 40% of labeled data. This is because such labels prevented merging of the cluster containing a segment manually labeled as "add to cart" with another cluster that had a labeled

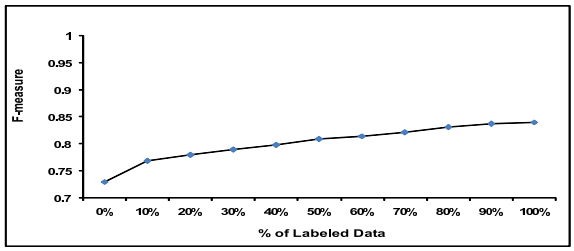

Figure 10: Performance of Concept Classifiers

segment "checkout". Similar trend is seen in Figures 9 and 10 and this was due to improvements in clustering (hence, more accurately labeled concept segments and click-streams) resulting from more user-labeled data. Finally, the performance improvement of personalized transaction models can be ascribed to the low degree of variability in captions and presentation patterns in a single Web site.

6. RELATED WORK

The work described in this paper has broad connections to a number of research areas.

Web Accessibility Research-

Extent assistive technologies, such as screen-readers and non-visual browsers (e.g., JAWS [1], IBM's Home Page Reader [5], Windows Eyes [11], BrookesTalk [35], etc.) provide shortcuts or summaries as a way to combat information overload in a single page. In contrast, our transaction model uses a global view of the content in a set of pages to determine what should be filtered in a state. Process automaton can also be used to rank different concepts occurring in a page.

End User Programming-

The research on end-user programming relating to our work includes programming by demonstration [9, 16], agent learning [3], query from demonstrations [31], etc.

Programming by demonstration allows users to construct a program by simply performing actions in the user interface, with which they are already familiar. CoScripter [19] uses this approach to build a collaborative scripting environment for recording, automating, and sharing web-based processes. Other browser recording and playback tools, e.g.,iMacros (www.iopus.com/imacros), also use this approach. However, these scripts are page-specific, whereas our transaction model is not, because it is scalable across different web sites that share similar content semantics.

PLOW [3] is a collaborative task learning system that learns task models by demonstration, explanation, and dialog. Knoblock et al. describe Karma [31], which allows users to easily build services that integrate information from multiple data sources. All of these enable the user to complete a task without knowing how to program; however, these approaches can be broadly categorized as supervised, requiring user interactions. Some other research projects (e.g.,[3]) also use domain knowledge to learn a task model. In contrast, our construction process is fully automated.

Automatic Information Extraction-

There are several works describing automatic information extraction (e.g. [18, 8]). [18] describes automatic labeling of web service data using a classifier. However, the classifier has to be trained a priori using labeled data. In contrast, we do not depend on user supplied labeled data. The RoadRunner system [8] automatically extracts data from web sites by exploiting similarities in page layouts. It learns the underlying template of Web sites from sample pages and uses it to automatically extract data from Web sites. However, the use of templates will not suffice for constructing a transaction model. Dong et al. [10] describes Woogle, a search engine that supports similarity search for web services through a process of clustering. In our construction process, clustering is just one of several steps.

Click-stream Analysis-

Building Web accessibility models from transaction click-streams is related to research in click-stream analysis and mining [25, 6, 17, 24, 7]. Click-stream data are mined to create user profiles [24], suggest context-aware query [7], predicting the next request of the user as s/he visits the same Web site [25], etc. Clustering and categorizing Web users based on pattern analysis of their click-streams is discussed in [6]. Our work departs from the above-mentioned research in both scope and approach. The construction of our accessibility models requires deeper content analysis based on context and segmentation. Our models are used to combat information overload when performing web transaction non-visually.

Process Automata Learning-

Constructing the transaction model is also related to research in mining workflow process models [27, 12, 28] (see [32] for a survey). However, our definition of a transaction is simpler than the traditional notions of workflows (e.g.,we do not use sophisticated synchronization primitives). Hence, we are able to model our transactions as finite state automata instead of workflows and learn them from example sequences. Learning automata is a thoroughly researched topic (see [23] for a comprehensive survey). However, automata learning requires a sizable number of negative examples, which are relatively difficult to obtain. In our work, we learned a simple process automata from positive examples, which are completed transaction sequences. Specifically the automata accepts strings from a simple, less expressive language for representing transaction sequences.

Contextual Analysis-

The notion of context has been used in different areas of computer science research. For example, [14] defines the context of a web page as a collection of text, gathered around the links in other pages that are pointing to that web page. The context is then used to obtain a summary of the page. Summarization using context is also explored by the InCommonSense system [4], where search engine results are summarized to generate text snippets.

All of these works define the context of the link as an ad-hoc collection of words surrounding it. In contrast, our notion of context, used in our earlier paper [20], is based on topic similarity of words around the link. We identify context with a simple topic boundary detection method [2], confined to geometric segments that have semantically related content. Our earlier paper on context-directed browsing [20] used this notion of context to identify the most relevant segment on the *next* page after following a link [20]. In contrast, here we use the context of a clicked web object to identify the concept segment containing the object in the *same* page. The concept segments are later clustered to label the click-streams and learn concept classifiers.

7. CONCLUSION

In this paper we described a fully automated algorithmic process for constructing transaction models from partially labeled click-streams. A model is comprised of a process automaton and concept classifiers, and has the objective of reducing the information overload experienced by blind users, who use the web for paying bills, shopping online, etc. Our approach reduces (and can even eliminate) reliance on sighted users in constructing such models, thereby bridging the web accessibility divide and further increasing the independence of blind users.

The construction process is even broader in its scope of applicability. The idea of using transaction models for fetching relevant information can be very useful for mobile devices with limited displays. Another application area is content extraction from web sites using wrappers. Our construction process in principle can automatically create such wrappers from click-streams obtained from example runs of content extraction.

There are several avenues for future research. So far, we have focused on the construction of domain-specific transaction models, in particular, for the shopping domain. We have observed that our techniques readily apply to other domains (such as online bill

payments and banking). In our work, the process automaton was constructed to accept a simple language that we defined for representing limited generalizations of training transaction sequences. Defining more expressive languages for transactions is a problem that merits further exploration. While the focus of this paper was on eliminating the exclusive dependency on manually labeled data in constructing the models, the tight integration of these models with our HearSay assistive browser is work in progress.

8. ACKNOWLEDGMENTS

We thank the National Science Foundation (Awards IIS-0534419, 0808678 and CNS-0751083) for supporting the research reported in this paper.

9. REFERENCES

[1] JAWS Screen Reader. http://www.freedomscientific.com.

[2] J. Allan, editor. *Topic Detection and Tracking: Event-based Information Organization*. Kluwer Academic Publishers, 2002.

[3] J. F. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. D. Swift, and W. Taysom. Plow: A collaborative task learning agent. In *Proc. of AAAI*, 2007.

[4] E. Amitay and C. Paris. Automatically summarising web sites - is there a way around it? In *Proc. of the CIKM*, 2000.

[5] C. Asakawa and T. Itoh. User interface of a home page reader. In *Proc. of ASSETS*, 1998.

[6] A. Banerjee and J. Ghosh. Click-stream clustering using weighted longest common subsequences. In *Proc. of the Web Mining Workshop at the 1st SIAM Conference on Data Mining*, pages 33–40, 2001.

[7] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *Proc. of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 875–883, 2008.

[8] V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *Journal of ACM*, 51(5):731–779, 2004.

[9] A. Cypher. Watch what i do: Programming by demonstration. *MIT Press*, 1993.

[10] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *VLDB '04: Proc. of the Thirtieth international conference on Very large data bases*, pages 372–383, 2004.

[11] D. Geoffray. The internet through the eyes of windows-eyes. In *Proc. of Tech. and Persons with Disabilities Conf.*, 1999.

[12] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Mining expressive process models by clustering workflow traces. 2004.

[13] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[14] B. B.-M. J.-Y. Delort and M. Rifqi. Enhanced web document summarization using hyperlinks. In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 208–215, 2003.

[15] P. Jaccard. Bulletin del la soci?t? vaudoisedes sciences naturelles 37. pages 241–272, 1901.

[16] T. Lau. *Programming by Demonstration: a Machine Learning Approach*. PhD thesis, University of Washington, 2001.

[17] J. Lee, M. Podlaseck, E. Schonberg, and R. Hoch. Visualization and analysis of click-stream data of online stores for understanding web merchandising. *Data Min. Knowl. Discov.*, 5(1-2):59–84, 2001.

[18] K. Lerman, A. Plangprasopchok, and C. A. Knoblock. Automatically labeling the inputs and outputs of web services. In *Proc. of AAAI*, 2006.

[19] G. Leshed, E. M. Haber, T. Matthews, and T. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In *Proc. of CHI*, 2008.

[20] J. Mahmud, Y. Borodin, and I. V. Ramakrishnan. Csurf: A context-directed non-visual web-browser. In *Proc. of the WWW*, 2007.

[21] J. Mahmud, Y. Borodin, and I. V. Ramakrishnan. Assistive browser for conducting web transactions. In *Proc. of the IUI*, 2008.

[22] S. Mukherjee, G. Yang, W. Tan, and I. Ramakrishnan. Automatic discovery of semantic structures in html documents. In *Proc. of ICDAR*, 2003.

[23] K. Murphy. *Passively learning finite automata*, 1996.

[24] O. Nasraoui, C. Cardona, and C. Rojas. Mining of evolving web click-streams with explicit retrieval similarity measures. In *Proc. of "International Web Dynamics Workshop", International World Wide Web Conference*, 2004.

[25] M. T. Ozsu. A web page prediction model based on click-stream tree representation of user behavior. In *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–540, 2003.

[26] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.

[27] R. Silva, J. Zhang, and J. Shanahan. Probabilistic workflow mining. 2005.

[28] R. Silva, J. Zhang, and J. G. Shanahan. Probabilistic workflow mining. In *Proc. of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 275–284, 2005.

[29] A. Strehl. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining*. PhD thesis, The University of Texas at Austin, May 2002.

[30] Z. Sun, J. Mahmud, S. Mukherjee, and I. V. Ramakrishnan. Model-directed web transactions under constrained modalities. In *Proc. of WWW*, 2006.

[31] R. Tuchinda, P. Szekely, and C. A. Knoblock. Building data integration queries by demonstration. In *Proc. of the IUI*, 2007.

[32] W. van der Aalst and A. Weijters. Process mining: A research agenda. *Computers and Industry*, 53:231–244, 2004.

[33] V. Vapnik. Principles of risk minimization for learning theory. In *D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, Advances in Neural Information Processing Systems 3*, 1992.

[34] S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma. Improving pseudo-relevance feedback in web information retrieval using web page segnmentation. In *Proc. of Intl. World Wide Web Conf. (WWW)*, 2003.

[35] M. Zajicek, C. Powell, and C. Reeves. Web search and orientation with brookestalk. In *Proc. of Tech. and Persons with Disabilities Conf.*, 1999.