

WAIF_R: Web-Browsing Attention Recorder Based on a State-Transition Model

Dmitrii Zagorodnov *

Lars Brenna *

Cathal Gurrin °*

Dag Johansen *

University of Tromsø, Norway *
{dmitrii, larsb, dag}@cs.uit.no

Dublin City University, Ireland °
cgurrin@computing.dcu.ie

ABSTRACT

This paper presents a model for a software component that monitors a user's interaction with a computer system and distributes summaries of that interaction to remote repositories. We illustrate the model with a prototype, implemented as a Web browser extension, for recording temporal aspects of Web browsing, such as time spent reading a page or keeping it open. To give the user control over privacy exposure and network resource usage that are inherent in remote monitoring, our design allows the user to specify how detailed the summaries are and who they are sent to.

Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements/Specifications;
H.1.2 [Models and Principles]: User/Machine Systems—
Human information processing

General Terms

Design, Human Factors, Performance

Keywords

attention metadata, attention recorder, implicit feedback, browsing model, privacy

1. INTRODUCTION

Many contemporary Internet services rely on feedback from users to improve the quality of the search results they return and the recommendations they generate. By learning more about the users and detecting similarities among them, the services can generate recommendations using collaborative filtering techniques [5]; by learning about what the users like and dislike, the services can adjust the ranking of search results to improve precision [12]. The majority of successful search engines, online merchants, and community sites can all testify to the utility of user feedback.

Although *explicit* feedback can be effective, as in the case of product reviews on merchant sites or votes for a Web page

by an online community, the cost of collecting it – either by hiring experts or by asking users to submit it – is prohibitive in many domains. This is especially the case when many responses are necessary for a meaningful result. Consequently, collection of *implicit* feedback – i.e., inferring users' preferences from their behavior – is becoming common, too. We will refer to the records of user behavior, which are potentially useful as implicit or explicit feedback, as *attention metadata* or just *data*, we will refer to the component that collects and distributes that data as an *attention recorder*, and to the destination for attention data as a *bank*.

In theory, any type of interaction between the user and the system, even such minute details as the number of seconds the user spends scrolling through a document, may constitute useful feedback for the author of the document or for an intermediary that offers personalized search results or recommendations. In practice, however, sending such detailed attention data to the parties that serve the user is both a resource burden and a privacy risk. The latter of these is beginning to gain attention; the former will be a problem if attention gathering becomes more widespread.

Currently, the attention metadata for Internet users is collected either with a client-side extension – the most famous one being the Google toolbar – or by logging their use of a Web-based service, such as search, email, etc. Under this setup, short of discontinuing the use of the service, the user has no control over what types of behavior are logged and at what precision; furthermore, the user is not able to share their metadata with another service. Our goal was to design an attention recorder that is general enough for the needs of many services and that the user can configure to specify what attention metadata each service is allowed to receive. Essentially, the idea is to let the users (and not their service providers) decide on the best trade-off between the loss of privacy and the gain from personalization.

The contributions of this paper are twofold: First, in Section 3, we present a model of an attention recorder that we believe is general enough for the types of attention metadata discussed in the literature to date. (That literature is summarized in the next section.) Second, in Sections 4 and 5 we describe how we applied this model to a Web browsing attention recorder. Although we are not the first to collect temporal attention metadata, such as time spent reading or idling on a page, we are not aware of other recording frameworks that: are capable of capturing and remotely logging the low-level details of user behavior, are locally configurable in terms of the amount and the detail level logged, and are compatible with a contemporary Web browser.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CAMA '06, November 10, 2006, Arlington, Virginia, USA.

Copyright 2006 ACM 1-59593-524-X/06/0011 ...\$5.00.

2. RELATED WORK

To our knowledge, all previous models of user behavior are based on records of *actions* – a set of things that the user can do with the system – along with the *items* that the actions operate on. For example, based on three earlier works, Nichols [9] identified 13 actions, such as *read*, *save*, *refer*, and *mark*, and mentioned several types of items, including academic publications, USENET posts, Web pages, etc.

Adding structure to previous models, Oard and Kim [10] classify actions into four categories – *examine*, *retain*, *reference*, and *annotate* – and assign items to three levels of “scope”: *segment*, *object*, and *class*. For example, reading is in the *examine* category, saving is in *retain*, referring is in *reference*, and marking is in *annotate*. The only explicit feedback action – *rate* – is also in the *annotate* category.

In terms of scope, their classification system allows one to differentiate among viewing a part of a document, viewing a specific document as a whole, and viewing a class of documents (e.g. all Web pages on a host or a domain). Kelly and Teevan [7] add the fifth behavior category, *create*, and a few more action types to the classification, which is then complete enough for them to conduct a survey of much of the prior work on implicit feedback.

In addition to the item that the action operates on, action-specific attributes can be associated with the action. For example, a price accompanies the *purchase* action, while a reference accompanies the *refer* action. Perhaps the most common attribute in literature, investigated by many projects – [3, 11], to mention a couple of recent ones – is time *duration*, which is typically attached to actions in the *examine* category. It can indicate, for example, how long the user spent reading a document, a specific section of it, or a collection of documents (depending on the scope).

The works cited above, as well as many others, including some of the earliest investigations of implicit feedback [8], found strong positive correlation between the reading time and the results of explicit feedback. Some authors [9, 3] reported that a combination of reading time with a measure of scrolling inside the document produces in the most accurate prediction of user interest attainable with implicit feedback. Hence, while there is no universal agreement on the best set of actions to record, implicit feedback *is* considered effective, including time durations and low-level user interface actions like scrolling.

In terms of how much the users have to modify their systems, the related projects run the gamut from requiring the users to install a custom application (e.g., for reading news [8] or browsing the Web [3]), to application extensions [4], to post-processing of user access logs [11] (which can be done with a proxy and requires no installation). The prototype we present in Section 5 fits in the middle of the spectrum, along with the Internet Explorer extension used by Fox et al [4].

3. ATTENTION RECORDER MODEL

Building on the results of the related projects, we based our conceptual model of an attention metadata recorder around a set of actions that operate on items. Actions identify the types of user interaction with the system that the recorder can capture. Items identify the object of that action at some scope. (From now on, we will use the term “item”

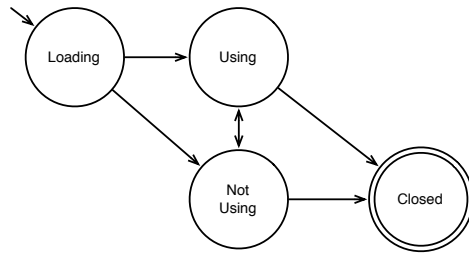


Figure 1: A simple state-transition model for a session with a document. *Loading* is the initial state, *Closed* is the terminal state. *Using* ↔ *Not Using* transitions are based on the UI actions that the recorder captures and aggregates.

in the discussion of the general model and we will use the more specific “document” when the items in question are always documents, such as when recording Web browsing actions at the scope of an entire page.)

An *attention record* is an encoding of an action, an item, and additional fields, which will be discussed shortly. The primary task of an attention recorder is to capture actions and forward them as records to one or more destinations. A remote attention bank can be a destination, and so can a local file.

Motivated by our consideration of privacy and performance, our attention recorder model also allows for filtering of actions: Each captured action is evaluated by a *filtering function* for each potential destination. The function returns a set of zero or more records; if one or more records are returned, they get forwarded to the destination associated with the function.

Such functions allow for the records to be arbitrarily manipulated, multiplied, or discarded altogether (by returning an empty set). For example, such a function can deny to a destination access to attention data associated with a particular Web site or with the HTTPS protocol. A function can “sanitize” a record by changing action attributes or the scope of the item. For instance, instead of reporting about every page that a user sees, a function could enlarge the scope of the item, reporting only the server name or only the domain name.

3.1 Aggregation of actions

Aggregator is a type of filtering function that emits a record in response to multiple discarded actions. The content and the timing of the record emitted by an aggregator may depend on the content, sequence, or timing of the discarded actions. The important point is that an aggregator keeps state (about the items) between actions.

Consider, for example, a recorder that can capture low-level user-interface (UI) actions, such as mouse movement and keyboard input. If the intended use of the attention data does not require such high level of precision, these actions could be “summarized” by an aggregator into less frequent records that state the amount of viewing or editing that a user performed and list the items involved. These summary records can be sent periodically or they can be sent at natural transition points, such as when the user switches focus to or from an item.

We found it helpful to design aggregators of UI actions

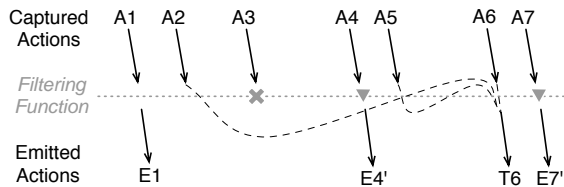


Figure 2: Operation of an attention filtering function (time flows to the right): action A1 is emitted as event E1 without modification; A2, A5, and A6 are aggregated into a transition T6; A3 is discarded; A4 and A7 are modified and emitted as events E4' and E7', respectively.

with the help of a *state-transition model* in which each session of a user with an item is viewed as a sequence of state transitions. Given all the states and all the valid transitions, one can visualize the model with a state-transition diagram, such as the one shown in Figure 1. This model applies to any session that involves the use of a document.

While simplistic, the model is sufficient for the design of a non-trivial aggregator: UI actions that arrive while the document is in the *Using* state are summarized by a record emitted upon a state transition. Thus, when the application window switches to the background and the document transitions to *Not Using*, or when the document is *Closed*, a record describing user’s behavior (e.g., number of keystrokes) is emitted.

With an aggregator based on the state-transition model, it is convenient to distinguish between two types of actions:

1. *Transitions* herald an item state change and are identified by pre- and post-transition states. For example, when a user switches from one application to another, the document open in the former may undergo *reading* \rightarrow *background* transition, while for the document in the latter the reverse will happen.
2. *Events* are actions that are not associated with the state change of the item. For example, after a document is printed, which can be announced with a record, it remains in the open state.

See Figure 2 for an illustration of how a filtering function processes the captured actions and emits transitions and events. As shown, actions may pass through without modification, may be discarded, modified, or aggregated. Not shown – as we have not found it useful so far – is the possibility of emitting multiple records in response to an action.

3.2 Attention record

Table 1 shows the contents of an attention record in our model. *Timestamp* pinpoints the moment when the action occurred on the user’s host and allows for the actions of a user to be ordered. *Action* contains a specification of either an event (e.g., *print*) or a transition (e.g., *browsing* \rightarrow *idle*). *Item* identifies the object of the action. Depending on the scope, that can be a pointer to a document, such as a URI, or a specification that is more narrow or more general.

The last two components of a record are optional. If set, *Instance* allows us to distinguish among multiple windows displaying the same document. This addition to the

model was prompted by our experience with the prototype attention recorder, which had to accommodate multiple Web browser windows and tabs. For example, the instance of a Web page open in a browser tab may be “Firefox window 2, tab 5.” We suspect that instances are not useful at scopes larger than a document.

Attributes may contain properties (key-value pairs) associated with the action. Most notably, the *duration* attribute attached to a transition can specify the time the item spent in the pre-transition state; added together, these values can tell, for instance, how long a user spent viewing a document. The duration attribute captures the temporal aspects of user behavior; the number of times the item passes through a state captures the “repeated use” aspect [9].

4. BROWSING RECORDER MODEL

Building on the concepts introduced in the previous section, we now discuss the attention recorder for monitoring Web browsing behavior. In this section we give it a general treatment and in Section 5 we discuss the details of our implementation.

4.1 Actions and items

An attention recorder that runs as an extension of the browser can capture both high-level application actions and low-level system actions. Some of the former are necessary if high-precision temporal attention metadata is desired. We further categorize the low-level events into *I/O-related actions*, which announce loading and unloading of documents, and *UI-related actions*, which result from mouse movement and keystrokes of the user.

High-level application actions include such activities as printing a document, saving it to disk, or bookmarking it. While such traditional browser functions can be enumerated, a comprehensive listing of high-level actions is a moving target since the browser is continually evolving as it turns into a general application platform. (See Table 2 for the list of actions supported by our prototype.)

Likewise, the types of *items* that contemporary Web applications work with – email messages, photo galleries, annotated maps – is unbounded. However, since most of these items are addressable by a URL, an attention recorder can go a long way by focusing on those. Conveniently, the structure of the URL, as defined by RFC 3986 [1], has natural scope boundaries:

`http://example.com:8080/over/there?key=value#part5`
scheme authority path query fragment

Table 1: Parts of an attention record.

<i>Timestamp</i>	Time on the user’s host when the action occurred.
<i>Action</i>	Specification for an event or a state transition (of the form: <i>previous</i> \rightarrow <i>next</i>).
<i>Item</i>	Identifier for the item that the action was applied to.
<i>Instance</i>	Identifier for the specific window of the application operating on the item.
<i>Attributes</i>	A set of action-specific attributes, such as <i>duration</i> .

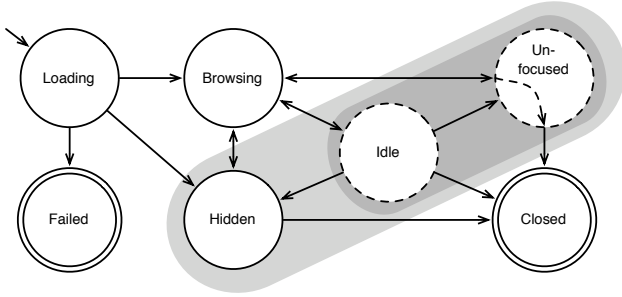


Figure 3: Web browsing state transitions for an instance of a document. Groups indicated by shading can be aggregated into single states, reducing the number of emitted events. Transition $browsing \rightarrow closed$ always passes via $unfocused$.

A complete URL allows a precise specification of the item, down to a section of the document. If a larger scope is desired, parts of the URL can be removed or simplified: the path can be truncated by removing directory names, the authority section can be reduced to the domain name or just the top-level domain.

4.2 State transitions

Figure 3 shows the state transition diagram for an instance of a Web document. Each arrow in the diagram corresponds to a potential state transition and the attention recorder can produce a record upon any of the transitions, as configured by the user. *Loading* is the initial state which can also be reached from any other state through a reload requested by the user or an automatic refresh (we omitted those arrows for clarity).

First, consider the five states shown as solid circles. They capture the general stages during the lifespan of a URL opened in a browser: all documents, whether or not they have been viewed before (and possibly cached), start in the *loading* state; if loaded successfully, the document can be either in the front-most tab of the window (*browsing* state) or in one of the background tabs (*hidden*); the document may move between these two states indefinitely as the user switches among tabs; when the page is unloaded, either by overwriting it with another page or by closing the tab or the entire window, it moves to the terminal *closed* state.

The other terminal state is *failed*, which indicates that the page failed to load, either due to a server error or due to the loading being canceled by the user. Memory-resident information about the documents in the terminal state can be garbage collected by the recorder.

Focus and idleness

We suspect that the five states shown as solid circles are sufficient for many types of attention recording. (Note that without the *failed* state, this diagram is equivalent to Figure 1.) A system that uses implicit feedback, however, may benefit from further detail.

For example, since a browser may have multiple windows open, each with a front-most tab, the analysis of attention data could take into account whether the front-most tab has focus or not. For that reason, we add the *unfocused* state to

the diagram and change the semantics of *browsing* to only apply to the front-most tab with focus. (Also, since a closing window always receives the “blur” event from the browser, we omit the direct $browsing \rightarrow closed$ transition.)

Granted, an unfocused window may still be readable to a user, but short of tracking eye movement there is no way for an attention recorder to tell. A related problem is that we cannot tell whether a focused window is being read. The best we can do is analyze the low-level UI actions to infer when the window is in use.

To avoid the overhead of sending the low-level UI events to the attention banks, we rely on an aggregator, a timer, and an additional *idle* state: Every time an action of interest, such as scrolling or clicking, takes place, the aggregator resets the timer and discards the action. If the timer fires, the recorder emits a transition and places the document into the *idle* state. The document remains there until a UI action indicates user activity or until a window “blur” confirms inactivity or until the document is closed.

5. BROWSING RECORDER PROTOTYPE

Our work on attention recording was motivated by the need for fine-grained user feedback to use in a recommendation system [2] that we are building as part of the Wide-Area Information Filters (WAIF) project [6]¹. The work resulted in WAIF_R (with “R” for “recorder”), an attention-recording extension for the Firefox browser written in Javascript and the backend code (PHP with MySQL) for saving the attention records and analyzing them.

We started with the code of the attention recorder from AttentionTrust², which logs URLs fetched by the browser, and modified it to capture various types of events, manage document state, and measure state durations. The recorder implements the 7 states and 15 transitions shown in Figure 3.

The actions supported by WAIF_R are listed in Table 2, with \star denoting all possible states for the transition. The two left columns list the events and their categories, respectively. The name of the “retain” category used in the last three rows is borrowed from the related literature [10], but all others are low-level events that trigger state transitions. The transitions themselves fall under the “examine” action category. A special “timeout” action, internal to the recorder, is generated by the timer to trigger the $browsing \rightarrow idle$ transition.

Implementation of the state-transition model matched well with the event-driven programming model of a GUI application such as Firefox. The only difficulties with respect to capturing actions resulted from apparent bugs in the Mozilla architecture. Most notably, on one platform (MacOS), the “blur” event was not always delivered by Mozilla. Since an idle window, focused or not, would be eventually detected with a timeout, this only affected the timings by at most the length of the timeout.

Our default timeout is 20 seconds. Although such small value means we often underestimate the amount of time a user spends reading without any detectable input, it also limits the magnitude of our overestimation. Admittedly, the choice of this value requires further empirical investigation.

¹<http://waif.cs.uit.no/>

²<http://attentiontrust.org/>

Table 2: Actions supported by WAIF_R.

Event	Category	Transitions triggered by the Event	Comment
Load started	I/O	$\star \rightarrow \textit{loading}$	
Load done	I/O	$\textit{loading} \rightarrow \textit{browsing}$ $\textit{loading} \rightarrow \textit{hidden}$	if document is in the front-most tab if document is in a background tab
Load failed	I/O	$\textit{loading} \rightarrow \textit{failed}$	
Unload	I/O	$\star \rightarrow \textit{closed}$	
Timeout	Internal	$\textit{browsing} \rightarrow \textit{idle}$	triggered by the attention recorder
Key press	UI	$\textit{idle} \rightarrow \textit{browsing}$	
Mouse click	UI	$\textit{idle} \rightarrow \textit{browsing}$	
Mouse movement	UI	$\textit{idle} \rightarrow \textit{browsing}$	
Blur	UI	$\star \rightarrow \textit{unfocused}$ $\textit{browsing} \rightarrow \textit{hidden} \ \& \ \textit{idle} \rightarrow \textit{hidden}$	if document is in the front-most tab if document is in a background tab
Focus	UI	$\textit{hidden} \rightarrow \textit{browsing} \ \& \ \textit{unfocused} \rightarrow \textit{browsing}$	
Print	Retain		
Save	Retain		
Bookmark	Retain		

5.1 Attention record format

With respect to the conceptual model (from Table 1), the *Item* in a WAIF_R record is identified by its URL (scope varies depending on what parts of the URL are present, as mentioned at the end of Section 4.1) and by the content hash (to detect changes in content, if necessary); the *Instance* is a combination of the application ID and the browsing tab ID; *Timestamp* is in milliseconds since the start of 1970 according to the clock on the user’s host (although the backend also adds a timestamp to allow correlation in time of records from different users).

The *duration* attribute is present in all records and it is measured on the client, also in milliseconds. Table 3 is a summary of the action attributes that WAIF_R currently supports. Again, \star denotes all valid states for a transition. The attributes can help determine the cause of the failure to load the full page, what triggered the HTTP request, how a page was closed.

A filter function can use items or attributes for choosing what to discard. For example, to avoid recording actions associated with secure connections, the filter can discard actions with the “https://” URL scheme. To filter out automatic page reloads, the filter can discard $\star \rightarrow \textit{closed}$ transitions with *how=reload* and the $\textit{loading} \rightarrow \star$ transitions with *refresh=true*.

To tie an attention record to a person we rely on user IDs, which are generated randomly and stored in the user’s Firefox profile when the extension initializes. (If the user has more than one profile, such as one for home and one for office use, and they want the attention data tied to a single identity, they have to manually configure the two profiles to use the same user ID.) Since we do not log IP addresses of users and do not request any personal information from them, only the information available in the attention data can potentially tie a user ID to the person; in our system, the person has control over the attention data.

We retained the XML-based format for records used by the AttentionTrust recorder, adding several fields to it. While there is no clear standard for attention metadata now, we expect that it would be trivial to modify WAIF_R to support the standard format if one emerges.

5.2 Unresolved issues

There are two issues, both specific to Web browsing, that we have ignored in the current WAIF_R implementation:

Some of the pages that fail to fully load can still be usable to the user. For example, many error pages, such as those with HTTP code 404, contain useful information for finding the page that failed to load or for reporting a broken link. To support them in the model of Figure 3, the recorder could make the *failed* state non-terminal and add transitions from it to *browsing* and *background*. Alternatively, error pages can be treated as valid HTML pages.

Online bookmarking services, such as del.icio.us and furl.net, enable the users to bookmark and save Web pages remotely. These are examples of high-level actions that extend the functionality of a traditional browser. To capture such feedback, the recorder would need to allow for custom capture code and filtering functions that would submit such actions to the attention bank.

6. CONCLUSION AND FUTURE WORK

Currently the WAIF_R attention recorder is being tested by the members of our research group, with over 162,000 records from the past nine weeks already “in the bank.”

Our immediate plans include a collaborative filtering experiment to investigate the utility of temporal attention metadata. Plotting users into a vector space will allow for matching users according to the similarity of their browsing behavior. This will then support recommendation of news feeds, whereby a user is recommended news feeds that are important to a similar user, and in this way, the user is introduced to new content, which may not be recommended simply on the basis of their previous browsing behaviour.

Undoubtedly, rich attention metadata has a high value in the contemporary information economy. The top players all collect, store, and analyze the attention of their users, typically without the latter being aware of the extent to which their privacy is compromised. Given the potential for misuse of the attention data, we believe in keeping the user aware of and in control of the trade-off between privacy and personalization, which routinely takes place on the Internet today.

Table 3: Action attributes supported by WAIF_R.

Transition	Attributes	Possible values
<i>loading</i> → <i>failed</i>	<i>how:</i> <i>error code:</i> <i>duration:</i>	server-error user-cancel http-404 http-500 ... τ milliseconds
<i>loading</i> → *	<i>from:</i> <i>predicates:</i> <i>duration:</i>	location-bar click bookmark is-cached, is-reload, is-refresh τ milliseconds
* → <i>closed</i>	<i>how:</i> <i>duration:</i>	window-close click-through reload τ milliseconds

In a step toward the solution, we offer the users of our attention recorder fine-grained control over the attention data that is sent to each attention bank. Since these mechanisms operate at a low level, we plan to look at how a high-level privacy policy may translate into low-level action filters. More generally, we intend to use our prototype to better understand the technological implications of attention recording on privacy.

7. ACKNOWLEDGEMENTS

The authors would like to thank the folks behind AttentionBank, who made the code of their recorder available. We are also thankful to the anonymous reviewers for their helpful comments.

8. REFERENCES

- [1] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (URI): Generic syntax. RFC 3986, Network Working Group, Jan. 2005.
- [2] L. Brenna, C. Gurrin, D. Johansen, and D. Zagorodnov. Automatic subscriptions in publish-subscribe systems. In *Proc. 26th IEEE Intl Conf. on Distributed Computing Systems (ICDCS) Workshops: Distributed Event-Based Systems (DEBS)*, Lisbon, Portugal, July 2006.
- [3] M. Claypool, D. Brown, P. Le, and M. Waseda. Inferring user interest. *IEEE Internet Computing*, 5(6):32–29, Nov. 2001.
- [4] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve Web search. *ACM Trans. Inf. Sys.*, 23(2):147–168, 2005.
- [5] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information Tapestry. *Communications of the ACM*, 35(12):61–71, Dec. 1992.
- [6] D. Johansen, R. van Renesse, and F. Schneider. WAIF: Web of asynchronous information filters. *Lecture Notes in Computer Science: Future Directions in Distributed Computing*, 2584, Apr 2003.
- [7] D. Kelly and J. Teevan. Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum*, 37(2):18–28, 2003.
- [8] M. Morita and Y. Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *Proc. 17th SIGIR*, pages 272–281, 1994.
- [9] D. M. Nichols. Implicit ratings and filtering. In *Proc. 5th DELOS workshop on filtering and collaborative filtering*, pages 221–228, Hungary, Nov. 1997.
- [10] D. Oard and J. Kim. Modeling information content using observable behavior. In *Proc. 64th Annual Meeting of the American Society for Information Science and Technology*, pages 38–45, 2001.
- [11] R. Rafter and B. Smyth. Passive profiling from server logs in an online recruitment environment. In *Proc. IJCAI workshop on intelligent techniques for Web personalization (ITWP)*, pages 35–41, 2001.
- [12] J. Rocchio. *The SMART REtrieval System : Experiments in Auotmatic Document Processing*, chapter Relevance feedback in information retrieval, pages 313–323. 1971.