Application Blogging - Automatic Post-generation and Search

Mayur Hemani
International Institute of Information Technology
Hyderabad, India
hemani@students.iiit.net

ABSTRACT
Weblogs have grown into a popular medium of publishing on the web. People weblog for several different reasons ranging from opinion expression to knowledge sharing. We present a simple framework where computer programs could blog their state on the web, so that users worldwide can use it. Application programs maintain the states of some objects. The paper studies the feasibility of having this state persistently recorded on weblogs. The motivating factor for doing such blogging stems from applications such as online chess where people would like to share, discuss and comment on games played by other players. We look at the issues arising from application-weblogging and present solutions to two problems - automatic weblog-post generation and application-weblog searching.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms
Application weblogging

Keywords
application weblogging, weblog search

1. INTRODUCTION
Weblogs, or blogs, have grown into a powerful medium of expression on the web. A weblog is a frequently updated website consisting of time-stamped entries arranged in a reverse-chronological order so that the most recent entry (post) appears first. People write weblogs[13] to express opinions, to convey news, to share knowledge, and for many other reasons. Weblogs are maintained using software built on top of web-servers. People can upload text (and more recently, pictures and video) to these servers. Each blogger has his or her own mini-website. A user who wishes to post a weblog must log into his/her account and then upload the text and media to the site. Viewers of the weblog can then see the contents appear on the weblog website. Most weblog servers also provide archiving for posts older than a certain amount of time. Weblogs are now looked at as sources of different kinds of information ([8]) including business trends, opinions about products, services, etc. and demographic information. Weblogs are being studied from several different perspectives ranging from its structural aspects ([1]) to dynamics of social networks ([12]). In short, weblogs constitute an important body of information on the web([9]).

Application programs maintain the state of some objects. For example, a chess program maintains a data-structure which represents the state of the chess-board. A chat program maintains the characters typed in by the user in a buffer. A web-browser maintains the state of its control objects. Along with object states, most programs also have events. An application weblog is a persistent record of such objects or such events, on the web. The idea is that programs can log their states or events on the web. This paper studies the feasibility of using weblogs for storing such logs on the web.

Application weblogs hold the states (or events) of a computer application and are stored in a retrievable form. There are several possibilities with having application states recorded on the web. Consider an online game of chess between two chess-players. During the game, if the players feel that the game is quite interesting and that it would be good if others could play the game from the same point onwards, or maybe, discuss about the game. With a weblog of such a game in place, other people could download, discuss[20] and share the game (Figure 1). Consider another example - that of an application help-center. Suppose there is an application program that does support application weblogging, then it could log the state of the program whenever error-conditions occur, and retrieve comments from weblogs having similar error-conditions. These comments could be filtered based on knowledge of the application and refined into context-sensitive help text. In an organization’s network, such a weblog could be a part of its knowledge infrastructure ([14], [17]). The possibilities are boundless.

Several issues arise when implementing application weblogs. First of all, application weblogging would require that the application program itself support weblogging, otherwise, it is difficult to capture the state of the system from outside of the program. Another issue that surfaces from having blogging application states on the web is that the stored states may not be intelligible to the users. For example, storing some variable values as they are, as a weblog-post would be hardly of any use since nobody would make sense of what it means. If it is desired that users download
the object states (as in the chess game), a more human-comprehensible description needs to be produced. Also, once application weblogs are in place, people would like to search for them. Searching in application weblogs could be done with keywords, as is done with normal web-pages and weblogs. However, it could be possible to search with the semantics of the particular application too. There can be several issues with implementing such a system, but we do not consider them in this paper.

This paper studies two problems with respect to application weblogging - weblog-post generation, and indexing of application weblogs for search with the semantics of the application. The next section states the problems in greater detail and definiteness, and relates work on automatic text-generation and weblog-indexing. The section on automatic weblog-post generation describes our solution to the problem of generating weblog-posts from object states. The next section describes the different ways in which application weblogs could be searched and presents a simple technique for searching with application-semantics. The next section illustrates the concepts stated above with the example of a chess application which we developed for testing out the ideas.

2. RELATED WORK

Figure 1. illustrates the manner in which a typical application weblogging system is intended to work. The object referred to, in the diagram is the state (or event description) that the application program submits to the system. From this point onwards, a submitted state is called an object. As mentioned in the previous section, an application weblog system is required to have a mechanism for generating weblog-posts from submitted object states. In this paper, we try to solve this problem with some simplifying assumptions, and for a single application. However, it is not a limiting assumption, since the same framework can be used for multiple applications without much change. It is for the convenience in illustration of ideas that only a single application is assumed throughout the rest of this paper. A second assumption used in this paper is that applications consistently submit (during blogging) the same kind of objects. This assumption too, is merely a convenience, since it is always possible to have the application to specify the structure of the object along with the values of the variables contained in the object.

The problem of generating explanation from data is a well-studied one ([10],[16]). There are several algorithms in the domain of natural text generation which use different planning techniques for generating textual description of things. However, weblog-posts do not need to be natural, in the sense that, as long as they convey the meaning, it doesn’t matter if the text is coherent or not. The need is not to describe the object in question in a human-friendly manner, but to have enough descriptive information present in each weblog-post. This paper presents an extremely simple solution for generating reasonably good, multi-sentential weblog-posts.

The second problem that we deal with is that of searching. Most weblog search engines use the same techniques for searching for weblog-posts as are used for other web-pages. Several weblog search engines exist on the web ([18],[4]). We deal with a slightly different problem - that of searching with semantics of the application in question, and searching with feature-values. There is a lot of work on semantic searching (for example, [5] and [6]), as well as content-based information retrieval (for example,[7]). We attempt to solve the problem of searching with semantics in the limited space of a particular application.

The idea of storing persistent objects on the web (for example, [3]), or using the web for storage ([15]) is not new. In this context, storing of simple blog-posts representing the states of a software system at some time is not really a problem from the storage perspective.

3. AUTOMATIC BLOG-POST GENERATION

Figure 2. depicts the solution that the paper suggests for
3.1 Features

The features used in the rules for text-generation are extracted from the submitted data object using the structure of the object. We earlier mentioned that we expect either a textual (entity-value pair) representation of the object, or a binary representation of the object. In the case of entity-value pair representation, the set of features can be defined by specifying the entities involved in a feature. In case of binary data, the size, the location (byte-offset, for example) and the data-type for each feature can be specified. Primitive data-types such as integers, characters, real numbers, as well as array-types can be easily supported. The extracted feature values need to be stored for use in indices.

3.2 Rule Representation

A rule uses the feature values to decide whether a (corresponding) piece of information should be a part of the weblog-post or not. Each rule comprises of some rule-properties and some information (sentences, images, etc.). If the rule-properties are satisfied by the feature-values extracted from the submitted object, then the corresponding information is added to the weblog-post.

The rule-properties consist of a rule-expression, and a corresponding threshold value. Each rule-expression in the set R of rules consists of variables and constants. The variables in the expressions are the features extracted from the submitted object. Each expression is composed of arbitrary arithmetic, string, and logical operations on the features. They can also contain simple programming constructs such as loops and conditionals. For each expression e in R, there is a threshold value T_e. A rule is considered to have matched if for all e in R, the value obtained from evaluating e is within the threshold T_e of the value V_e. If a rule matches, then the corresponding piece of information (text, in our case) is output as part of the weblog-post. Rules can be stored in a database along with their threshold values and ranks. The rules can be framed by an application expert or the developers of the application, or be generated in some clever way from the application itself.

3.3 Rule Ranking

For some applications, all information contained in the object states may not be equally important. It may be essential to report some things prior to reporting other things. For instance, in a chess game, reporting that a pawn was promoted may not be as important as reporting that the white king is under a check. For this reason, the rules can be given a partial order, categorizing them between important and unimportant. Arbitrary integer ranks can be specified for each rule, giving them a partial order. The rule-matcher then attempts to match the submitted data with the rules in that order.

3.4 Primary and Secondary Rule Matchers

The framework suggested in this paper uses two rule-matchers. The first one, called the primary rule-matcher uses the feature-values and generates descriptive or direct text. This text consists of simple descriptive sentences. The second rule-matcher uses fuzzy-rules to output some inferred text. This text can have more information than is discernible from the object state itself.

3.5 User-Text

In addition to the automatically generated weblog-text, a user can submit some text along with the object-state. This would be appended to the weblog-post generated by the rule-matchers.

3.6 The Algorithm

The algorithm depicted in Figure 3 describes the way the primary rule-matcher works. The input to the algorithm are the feature-values and the direct rules. The output is a set of sentences that represent part of the weblog-post. A suitably large random number N is generated. This represents the maximum number of direct rules, the output of which would be output to the weblog-post. All the rules in the input are executed and the resultant errors are stored in a closeness vector E. Each element E_i represents the error that resulted from attempting to match the rule R_i. If the rule is matched, the corresponding text sentences are appended to the weblog-post. Although, all rules are attempted, only the text corresponding to the top N rules (in the ranking order) is added to the output. The algorithm for the secondary rule matcher is identical, except that there are no error-vector is generated, and that the rules can be based on the feature-values as well as the vector E generated from the primary rule matcher. The fifth section illustrates this kind of rule-matching with an example pertaining to the game of chess.

3.7 Advantages and Disadvantages of the Approach

The primary advantage of using this scheme for generating weblog-text is that it is extremely simple to implement. Another important advantage of the approach is that not
only text but other forms of information too can be used as weblog-posts. For example, simple images can be constructed on the fly from the rules. Other media too, could be linked with weblog-posts when certain rules match. This makes the whole system quite flexible.

Using rule-matching for weblog-post generation has its own drawbacks. The text generated may not be coherent, since there is no planning algorithm involved in its generation. The text can be monotonous if the number of rules is small. But, as was mentioned earlier, these are not exactly drawbacks, since a weblog can hardly be considered to be a coherent piece of text itself.

4. SEARCHING IN APPLICATION BLOGS

Weblog searching has long been treated in the same manner as searching for other web-pages. They are indexed using a set of words arranged in either an inverted index [2] or in other web-page indices such as suffix arrays [11]. With weblogs, search is no different than with other web-pages. However, with application weblogs, the search ought to be different. For one, the posts in weblog-pages have not just text, but objects (or their feature values). Another reason why application-weblogs need to be treated differently is that the text in the posts is taken from a finite set of sentences specified by the rules. One can think of several ways of searching application weblogs; two of them are immediately evident:

1. Searching with keywords (with associated semantics)
   Given a set of query keywords, report those weblog-posts which are semantically similar or related to the words.

2. Searching with feature values
   Given an instance of an object, find weblog-posts that are the descriptions of similar objects

The first problem is solved using what can be called rule-based inverted indexing. The second problem can be solved by simulating the process of post-generation, followed by a search on a simple inverted index.

4.1 Searching with keywords

The search technique described here is based on the premise that the text in the weblog posts is actually taken from the finite set of sentences associated with the rules. The key idea is to have two inverted indices. The first inverted index is on the text associated with each of the rules. Given a set of query keywords S, this inverted index returns the ordered set R of rules that have the words in their text portion. The order is based on the rank of the rules. The second inverted index is on the weblog-posts themselves. A query on this index is a rule ID. The index yields the set P of posts, which were constructed using the query rule. This essentially implies that the query rule was matched during the generation of all weblog-posts in P. These two inverted indices, I1 and I2, are used together to answer search queries on the application weblogs. Figure 4 depicts the algorithm.

There is one problem with the algorithm - the keywords input as the query may not match with those that are in the inverted index for the rule-text. A simple solution to this problem is to first construct an application-specific thesaurus and then use it to get the word used in rule-text for every search keyword. For each interesting word w in the rule-text, we find a set Syn of synonymous words. w is then added to the thesaurus entries for all the words in Syn. Using this construct, one could map query words into those that are actually used in the weblog-posts, and then apply the search algorithm depicted in Figure 4. More sophisticated implementations can also use ontological representation for text as in [19]. This way, we can succeed in searching for weblog-posts based on the application semantics.

4.2 Searching with feature-values

Sometimes, keyword-searching may not suffice the needs of a user. For example, in the context-sensitive help application mentioned earlier, the search has to be based on the feature-values and not on keywords. The problem is that given an object (of the application), report all the weblog-posts which are the descriptions of similar objects. This problem can be solved in a simple manner using the weblog-post generation algorithm itself to generate keywords for searching. The idea is that given a search query object, we pass it to the rule-matcher, which generates the corresponding text. The text is filtered to obtain interesting keywords (the words that would have been used in the inverted index on the weblog-posts. The procedure is depicted in Figure 5. The rationale for searching with feature-values in this manner is that it is essentially a mapping from a high-dimensional space (of the complex object) to a set of keywords by a set of rules. These same rules were also used for generating the blog-posts, and hence it is quite likely that if the objects are similar, so will their descriptions and in turn, the keywords which are stored in the index for the blog-posts.

5. THE CHESS-BLOGGER PROGRAM

In order to test the above ideas, we wrote a simple chess
application which could perform blogging of its states. Additionally, we implemented the rule-matching and indexing algorithms for chess weblogs. The implementation is explained in some detail below, for illustrating the ideas stated in the previous sections.

- **The Client Program** - The chess program parsed PGN (Portable Game Notation) files which have several chess games. It simulated the games based on the moves. There was a Blog It button in the application’s interface, which initiates an HTTP request to send the captured game state to a web-server (the site where the rule-matcher and the weblogs were hosted).

- **Rule Representation** - The rule-matcher and search programs were written in PHP for the ease of coding. Another important reason why PHP was used is that it supports the execution of externally-stored code (eval). Essentially, the rules were written as PHP scriptlets and stored in a database. When a rule had to be matched, the corresponding code was brought into the memory and executed. Note that, this kind of an implementation can always be achieved irrespective of the programming language used (only it may be more difficult). A typical rule would look like this - if there are no pawns alive then there can be no further promotions possible.

- **Working** - Suppose a user clicks the blog-it button, then an HTTP request is initiated which transfers the game state (in this case, the positions of pieces on the game-board) to the weblog rule-matcher. The rule-matcher would store the game state in a local variable and the rules are brought into the memory. The rules are then executed on the feature-values and the corresponding text is generated. The indices and the weblog-post file are updated at this stage, too.

6. EXPERIMENTAL RESULTS

In this section we present some results from experimental implementation of the ideas stated above.

6.1 Chess Blog-Post Generation

Figure 6. illustrates some sample chess blog-posts generated by our chess-blogging program. The blog-posts were created from just 30 rules in the rule-base, each rule corresponding to one sentence of text. Each rule has a piece of code which uses the object state to determine if the rule matches or not. The maximum code size per rule is 512 bytes, which is a reasonable size for simple PHP expressions.

Note that most of the rules used in generating this text are deterministic. In other applications, these rules can be simple fuzzy rules too. In fact, any form of rules can be used in the same framework as long as the corresponding code can be stored in the rule-base database. The text does not appear to be a coherent description of the object state. Nevertheless, the text does convey quite a bit of information about the game. The corresponding object state submitted looks as follows:

```
ENT&Adams, Michael&Dickenson, Neil F

.................wk........
...........wp..br........wp........
........bK....bP............bP..wp..

.................wr..
```

As is clearly visible from the object’s constitution that it is not really intelligible to a normal human user. However, the given description of the game does give some idea of what really happened in the game. Also note that the state of the game is available for download alongside the blog-text. Further examples of blog-posts (text) generated using the same system are shown below:

```
The white king is under a check. Neither player has any knights alive. White has greater number of pieces in Black’s territory than black has in White’s. The black king has cas-
```
tled to a safe position. The black player appears to be in a superior position.

The black king has castled. The game is in its initial position. Both sides have equal number of pawns. White has all pawns alive and can still have promotions. Both players have equal number of pieces but the game may or may not be balanced at this stages.

### 6.2 Search using Feature Values

For testing out searching by feature values, we gave several chess-board objects as queries the result of which were blog-posts which had similar chess-board positions. For instance, the following object representing the chessboard in Figure 7 was given as a query:

**Query:**
```
wR..wBwQ..wRwK..wPwPwP....wPwPwP
.....................wB...wN......
......................bQbPbN....
bPbP......bPbPbPbR..bB..bKbB..bR
```

And the results were the following blog-posts:

**Adams, Michael vs. Dickenson, Neil F**

Both players have equal number of pieces but the game may or may not be balanced at this stage. Black has the greater number of pawns on its own side. A white rook is sitting in the corner. A black rook too is sitting in the corner. The game is still in its initial stages. Black has less number of pieces in the white’s territory than white has in black’s.

Friday, 10 March 2006 {11:34:58}

**Adams, Michael vs. Saeed, Saeed Ahmed**

The white king is under check. White has more number of pawns than black. A white rook is sitting in the corner. A black rook too is sitting in the corner. The white queen has been killed. The black queen has been killed. Black has less number of pieces in the white’s territory than white has in black’s.

Friday, 10 March 2006 {11:35:12}

The corresponding boards are represented in Figure 7. It is evident that mapping complex objects to a lower dimension (that of words) is feasible if the rules for doing so match both in case of querying as well as storage.

### 7. CONCLUSIONS

Weblogging and weblog-viewing as an activity is bound to grow with time. With application-weblogging, organizations can build knowledge bases automatically in a simple and searchable form. However, implementing application weblogging would cause several issues to surface. In this paper we suggested solutions to two such issues. The solutions are in fact an indication that more than problems, application weblogging promises to present opportunities for developing new collaborative applications.

### 8. REFERENCES


Intelligence, Portland, Oregon, 1996.


