Mining Software Usage Data

Mohammad El-Ramly
Department of Computer Science,
University of Leicester, UK
mer14@le.ac.uk

Eleni Stroulia
Department of Computing Science,
University of Alberta, Canada.
stroulia@cs.ualberta.ca

Abstract

Many software systems collect or can be instrumented to collect data about how users use them, i.e., system-user interaction data. Such data can be of great value for program understanding and reengineering purposes. In this paper we demonstrate that sequential data mining methods can be applied to discover interesting patterns of user activities from system-user interaction traces. In particular, we developed a process for discovering a special type of sequential patterns, called interaction patterns. These are sequences of events with noise, in the form of spurious events that may occur anywhere in a pattern instance. In our case studies, we applied interaction pattern mining to systems with considerably different forms of interaction: Web-based systems and legacy systems. We used the discovered patterns for user interface reengineering, and personalization. The method is promising and generalizable to other systems with different forms of interaction.

1. Introduction

An increasing amount of work is published on mining software "development" data, i.e., data produced while developing the software, e.g., CVS archives [16,17]. Also, a lot of work was done by the dynamic analysis community [9,10] on analyzing "runtime" data of software systems, e.g., execution traces of object oriented systems, etc. Little was done to mine software "usage" data, i.e., system-user interaction data collected while the system is running. Usage data is rich of information on how the system is currently used. This position paper makes a case for mining usage data in support of reengineering, reverse engineering and program understanding efforts.

Usage (or system-user interaction) data consists of temporal sequences of events that took place while the users were interacting with the system. We call such sequences "interaction traces". Typically, interaction traces contain interesting patterns of user activities and of the usage of system services in support of these activities. We demonstrate that sequential data mining techniques can be applied to discover these patterns. We used these patterns for interaction reengineering, i.e., reengineering the way the users interact with the system via user interface (UI) reengineering and personalization. Also, such patterns can be used for program comprehension, requirements recovery and reverse engineering tasks.

To validate our argument, we developed a process for mining interaction traces and successfully used it to mine usage data of two types of software systems, with radically different UI styles. The process retrieves a special type of sequential patterns, called interaction patterns. An interaction pattern is a frequently occurring sequence of events that might include up to a certain level of noise in each pattern instance, in the form of spurious events. Noise events can occur anywhere in a pattern instance. A pattern is considered interesting according to a user-defined criterion that defines the minimum pattern length and frequency and the maximum level of noise permitted.

We applied our "interaction pattern mining" process to legacy and Web-based systems. In the first application, we recovered the patterns of the frequent tasks done by the users of a legacy system from traces of its users' dialog with the legacy UI, recorded while the users were doing their regular jobs. These patterns model the currently active and demanded services of the legacy system as accessed via its UI. These service models are used for reengineering the legacy UI and wrapping it with a Web or WAP (Wireless Application Protocol) UI. In the second application, interaction pattern mining was used to discover the interesting navigation patterns in the Web server logs of a focused Web site. A focused Web site supports an ongoing evolving process and its users use it in a consistent way, e.g., navigation activities of different users during a period of time are more or less similar. Examples of such sites are Web sites of university courses. They evolve according to the course schedule and students access them similarly, following to the course-work progress. The discovered patterns can be used in reengineering the Web site UI for faster and easier access. This is done by giving online URL recommendations for current users to make their navigation easier and faster, by suggesting to them the consequent pages accessed by enough recent users who had similar navigation history.

While more case studies are needed, these applications demonstrate the applicability and value of the method and suggest that generalization to other forms of sequential system-user interaction and runtime data is straightforward.

In the following, Section 2 presents the "interaction pattern mining" problem. Sections 3 and 4 describe two applications of it. Section 5 is the summary and conclusion.

2. Interaction Pattern Mining

Mining sequences of data for recurring patterns is a generic problem with instances in a range of domains that are similar in that the data to be mined is represented by sequences, but different in the type of patterns of interest in each domain. Examples include sequential pattern mining of retail industry data [1], discovery of frequent episodes in event sequences [14], and discovering patterns in DNA and protein sequences [4]. Many algorithms to solve these problems emerged from data mining [1,2,3,14] and bio-informatics [11,13] communities.

Interaction pattern mining is similar to other sequential pattern mining problems in that the input data is ordered sequences of event Ids (or URLs or protein labels, etc.), but it is different in terms of the type of patterns sought. This is because interaction pattern mining constrains the maximum level of noise permitted in a pattern instance, but does not care where the noise occurs. This gives flexibility in discovering tasks that include user mistakes, trivial events and/or alternative paths for some subtasks.

To formally define interaction pattern mining,

1. Let A be the ***alphabet*** of events.
2. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of sequences. Each ***sequence*** s_i is an ordered set of Ids drawn from A and represents a recorded trace of the runtime behavior of the system under analysis, e.g., an interaction trace.
3. An ***episode*** e, is an ordered set of events occurring together in a given sequence.
4. A ***pattern*** p is an ordered set of events that exists in every episode $e \in E$, where E is a set of episodes of interest according to some user-defined criterion c. E and e are said to "support" p. The individual Ids in an episode e or a pattern p are referred to using square brackets, e.g., $e[1]$ is the first Id of e. Also, $|e|$ and $|p|$ are the number of items in e and p respectively.
5. If a set of episodes E supports a pattern p, then the first and last Ids in p must be the first and last Ids of any episode $e \in E$, respectively, and all Ids in p should exist in the same order in e, but e may contain extra Ids, i.e., $|p| \leq |e| \; \forall \; e \in E$. Formally,
   - $p[1] = e[1] \quad \forall \; e \in E$,
   - $p[|p|] = e[|e|] \quad \forall \; e \in E$, and
   - $\forall$ pair of positive integers (i, j), where $i \leq |p|$, $j \leq |p|$ and $i < j$, $\exists \; e[k] = p[i]$ and $e[l] = p[j]$ such that $k < l$.

The above predicate defines the class of patterns that we are interested in, namely, ***approximate interaction patterns*** with at most a predefined number of insertion errors. For example, the episodes {2,***4***,3,4}, {2,***4***,3,***2***,4} and {2,3,4} support the pattern {2,3,4} with at most 2 insertions per episode, which are shown in bold italic font.

6. An ***exact interaction pattern*** q is a pattern supported by a set of episodes E such that none of its instances has insertion errors
   - $q[i] = e[i] \quad \forall \; e \in E$ and $1 \leq i \leq |q|$
7. The ***support*** of a pattern p, written as *support* (p), is the number of episodes in S that support p.
8. **A *qualification criterion*** c, or simply ***criterion***, is a user defined quadruplet (*minLen*, *minSupp*, *maxError*, *minScore*). Given a pattern p, the ***minimum length*** *minLen* is a threshold for $|p|$. The ***minimum support*** *minSupp* is a threshold for *support* (p). The ***maximum error*** *maxError* is the maximum number of insertion errors allowed in any episode $e \in E$. This implies that $|e| \leq |p| + maxError \; \forall \; e \in E$. The ***minimum score*** *minScore* is a threshold for the scoring function used to rank the discovered patterns. A scoring function can be defined depending on the application in hand and the nature of the patterns sought.
9. A ***maximal pattern*** is a pattern that is not a sub-pattern of any other pattern with the same support.
10. A ***qualified pattern*** is a pattern that meets the user-defined criterion, c.

Given the above definitions, the problem of interaction pattern mining can be formulated as follows:

Given: (a) an alphabet *A*,
(b) a set of sequences *S*, and
(c) a user criterion *c*

Find all the qualified maximal patterns in *S*.

Solving interaction pattern mining problems is a three steps process. The first is a pre-processing step, needed for cleansing the data in the input sequences, depending on the problem in hand. In the second step, our novel algorithm for interaction pattern mining, Interaction Pattern Miner 2 (IPM2) [8], is applied to the cleansed sequences to discover the patterns that meet a user-defined criterion. The last step is the post processing analysis and comprehension of the discovered patterns. The first and third steps are application and data dependent.

3. Reengineering Legacy User Interfaces Using Interaction Patterns

CelLEST project for semi-automated legacy system UI reengineering [6,12,15] used interaction pattern mining as part of a lightweight automated process for UI reengineering of legacy systems that was designed to automate the then non-automated technology of our

industrial partner Celcorp. [5] CelLEST adopts a semi-automated task-centered lightweight process for wrapping legacy UIs with Web or WAP UIs. The idea is to semi-automatically understand and model the frequently used legacy system services, represented by frequent patterns of user activities. Then, new UIs for the desired platform are generated automatically, that wrap these demanded services. A significant innovation in the project is that the new UI packages an entire legacy system service (or user task) in the suitable UI on the target platform instead of mimicking the legacy interface one by one. For example, a system service that requires accessing 15 legacy screens may be reengineered into one Web form instead of 15 Web pages and forms because this is the natural representation of this task on the Web platform. Hence, the method is described as "task-centered". Service models, which were built from the patterns, are used by the new UI to invoke the legacy system services via the legacy UI as if the new UI is a typical user of the legacy system performing his or her tasks.

In this application, interaction pattern mining was used to recover the frequent patterns of user activities while using the legacy system. These patterns are buried in the huge amount of data exchanged in dialog between the users and the system via its UI. Mining the system-user interaction traces with the legacy UI discovers these patterns. The instances of each pattern are analyzed to infer information about the type and location on the screen of the user inputs. An analyst augments these patterns with extra semantic information to build full-fledged models of the frequent user tasks. Then, these models are automatically translated to abstract UI specifications and then to a UI implementation on the platform of choice: XHTML-enabled (Extensible Hypertext Markup Language) platforms or WAP devices. The automated pattern discovery and analysis process replaced the earlier time-consuming error-prone manual modeling process.

In CelLEST project, we used a host-access middleware that serves as an emulator and recorder to access legacy IBM 3270 systems. Unobtrusively through the middleware, legacy system users can open a session with the legacy application and do their regular jobs. The middleware records their dialog with the system UI in the form of sequences of legacy screen snapshots forwarded to the user' terminal, interleaved with the user actions in response, in the form of keystrokes. Screens and snapshots are like classes and objects; the later are instances of the former. Analysis methods, including feature extraction, clustering and others are used to derive a model of each legacy screen from all its instances in the traces. This model includes, for example, any distinguishing features of the screen like a title or a certain visual distribution of the screen content that occurs on all its recorded instances. In this step, each screen is given a unique Id. So, interaction traces can be abstracted by sequences of Ids as Figure 1 shows. Figure 1 is part of a real interaction trace taken from navigating a legacy library catalog system. Numbers are screen Ids and arrows are user actions.

To explain what type of interaction patterns can be found in these traces, Figure 1 shows two very similar segments of the user dialog with the legacy library system (in the dashed polygons) that occurred apart from each other in the trace. These segments suggest that the user was doing two similar runs of the same task, or alternatively, s/he was using the same legacy system service twice, although the two runs differ in the number of snapshots of screens 6 and 9 accessed. In the actual recorded trace, screen 4 displays the results of issuing a *browse* command to browse the relevant part of the library catalog file. Then, the user decides which items s/he wanted to retrieve from the catalog by issuing a *retrieve* command and s/he receives screen 5. Then, s/he displays brief information about the items using *display* command that displays screen 6. Finally, s/he selects an item using the *display item* command to display its full or partial information (screen 7). After selecting an option from screen 7 (e.g., full details, summary, etc.), screens 8, 9 and 10 display the first, intermediate and last pages of the required details, respectively. The number of snapshots of screens 6 and 9 retrieved varies depending on the item checked. The navigation segment of Figure 1 shows that this task of item information retrieval was done twice.

We applied interaction pattern mining to recorded system-user dialog traces of a number of systems [6]. To further explain the expected outcome of this application, we brief one of the experiments. In this experiment, we collected 5 traces of user interaction with the IBM 3270 connection of a public library system. Each trace represented a session of interaction between a user and the library catalog, during which s/he retrieved information about the library items of interest. The traces had 1657 screen snapshots in total and 27 screens (recall the classes

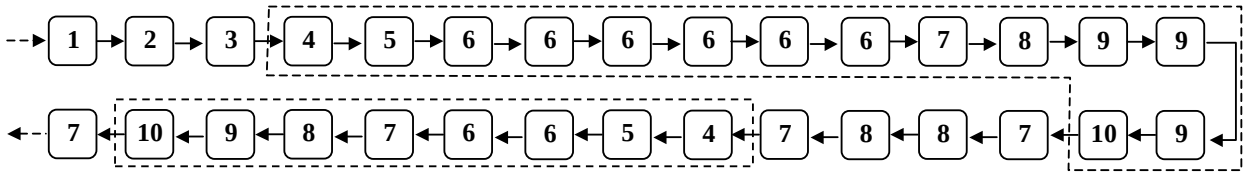

Figure 1. A Segment of a Trace of Interaction with a Legacy Library System

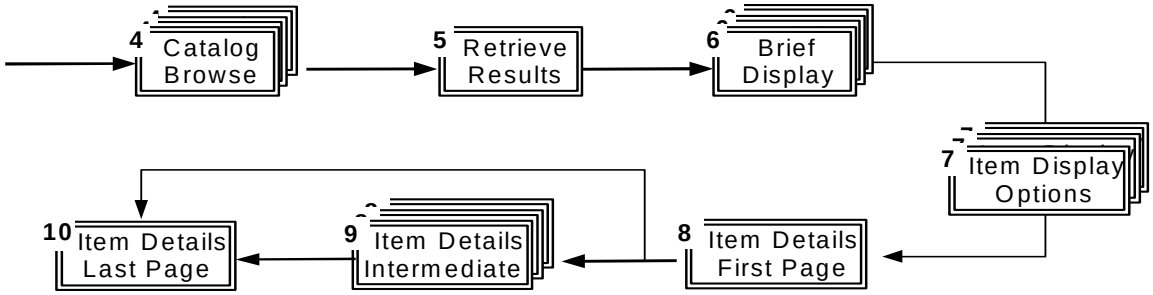

Figure 2. A Diagrammatic Representation of The Pattern 4^{+}-5-6^{+}-7^{+}-8^{+}-9^{*}-10, Corresponding to The Information Retrieval Task Repeated Twice in The Trace Segment of Figure 1.

and object analogy). The segment of Figure 1 is taken from one of the traces after identifying screens and labelling snapshots with their screen Ids. The traces were pre-processed and analyzed using IPM2 algorithm. Then manually, an analyst post-processed the discovered patterns by filtering out trivial patterns. Three patterns of interaction with the library catalog were discovered; each represents an information retrieval task, corresponding to a service of the legacy system. Figure 2 shows one of the interaction patterns discovered, which is $\{4^{+}$-5-6^{+}-7^{+}-8^{+}-9^{*}-10$\}$. n^{+} means one or more snapshots of screen n and n^{*} means zero or more. The two dashed polygons of Figure 1 represent instances of this pattern. Further details of this experiment are in [6].

In CelLEST, these patterns were used for lightweight UI reengineering. Each pattern was enriched with semantic information and then translated to abstract user interface specifications that are executable on multiple platforms, e.g., XHTML-enabled platforms and WAP devices [12]. Hence, a quick UI wrapper can be generated semi-automatically for the legacy services of interest.

Use Case name: Retrieving Information on a Library item
Participating actor: Library System User
Entry condition: The user issues a *browse* command
Flow of events:

1- Flip the catalog pages until the relevant page.
2- Issue a *retrieve* command to construct a results-set for the chosen catalog entry.
3- Display the results set using *display* command and turn its pages until the required item is found.
4- Issue a *display item* command.
5- Specify a display option.
6- Display the item details.

Exit condition: The user retrieves the required Information about the item s/he wants.

Figure 3. A Use Case Model Representation of The Interaction Pattern of Figure 2.

Beyond CelLEST, the enriched patterns can be represented in alternative formats to serve other reverse engineering and reengineering purposes. For example, to integrate the legacy system services with new object oriented applications designed in UML, it could be useful to translate the recovered interaction patterns to use cases to integrate with new UML-based requirements. Figure 3 shows a use case representation of the pattern of Figure 2. These patterns can also be used for re-documentation, and requirement recovery tasks.

4. User Interface Reengineering of Focused Web Sites Using Interaction Patterns

We used interaction pattern mining for Web-usage behavior analysis of focused Web sites and proposed a method for on-the-fly UI reengineering and personalization of such sites. A focused web site supports an ongoing evolving process and is usually navigated in a task-driven as opposed to data-driven way. The users of the Web site usually navigate it to accomplish the same task(s) during a certain period of time. As the process evolves, they shift together to other tasks. Models of these tasks, in the form of interaction patterns, are buried in the Web server logs.

We used a Web site of a computer science university course as an example of focused Web sites. The users of this site usually do similar tasks related to their course work every week, e.g., read and/or download new materials, lab instructions and assignments and related code, etc. Ideally, the discovered interaction patterns correspond to the frequent user tasks of interest, not just to interesting associations of Web pages. In other words, the sequence of navigation matters. Since interaction pattern mining tolerates noise, in the form of insertion errors, it can discover patterns of sequential user navigation with spurious navigation or slight differences. These patterns can serve as basis for online URL recommendations for current users to make their navigation easier and faster, by suggesting to them the further pages accessed by recent users who had similar

navigation sequences. Details of this application are in [7], but highlights are briefed below.

Mining Web logs for interaction patterns is a three steps process. First, preprocessing cleans and standardizes the Web server logs and divides them into sessions. A session is a coherent sequence of Web site navigation activities of the same user. Then, IPM2 is applied to the sequences of URL Ids representing sessions, with a user defined interestingness criterion. Then, post-processing depends on how the extracted patterns will be used.

We designed a method for Web UI reengineering and personalization using these patterns, which generates runtime recommendations for pages that new users of the Web site may want to visit. A runtime infrastructure, capable of user session tracking and relevant pattern selection, is needed in this application. The HTTP protocol is stateless and does not support establishing a long-term connection between the Web server and the client's browser. To address this problem, dynamic page rewriting with hidden fields can be used. When the user first submits a request, the server returns the requested page rewritten to include a hidden field with a session-specific Id. Each subsequent request of the user to the server supplies this Id to the server, enabling it to maintain the user's navigation history. This session-tracking method does not require any information on the client side and can therefore be employed, independent of any user-defined browser settings. Since the server knows the user's Id, it can examine its recent navigation history to identify whether it includes the prefix of any of the collected patterns. If so, the suffixes of the relevant patterns are offered as recommendations for subsequent navigation. Then, page-rewriting technique can easily support the dynamic adaptation of the pages requested by the users with the recommendations on new potential places to visit.

5. Summary and Conclusions

This position paper demonstrated that applying data mining to software usage data reveals valuable information about the system in the form of sequential patterns. Such patterns can be used for a variety of reengineering, program understanding and reverse engineering tasks. In particular, we presented a process for discovering a type of sequential patterns, called interaction patterns and two applications for it. The first is discovering patterns of the frequent user tasks in the recorded traces of system-user interaction with legacy systems. These patterns are used for automated UI reengineering. We also applied interaction pattern mining to discover frequent user navigation patterns from server logs of focused web sites. We proposed a method for lightweight Web site runtime reengineering by introducing on-the-fly URL recommendations based on these patterns.

Interaction pattern mining is a powerful technique for analyzing usage data and can be extended to different types of software runtime sequential data to discover patterns of user and/or system activities.

References

[1] R. Agrawal and R. Srikant, Mining Sequential Patterns. In Proc. of the 11th Int. Conf. on Data Engineering (ICDE), pg. 3-14, 1995.

[2] R. Agrawal and R. Srikant, Mining Sequential Patterns: Generalizations and Performance Improvements. In Proc. of the 5th Int. Conf. on Extending Database Technology (EDBT), 1996.

[3] J. Baixeries, G. Casas and J. Balcazar, Frequent Sets, Sequences, and Taxonomies: New, Efficient Algorithmic Proposals. Report No. LSI-00-78-R, El departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain, 2000.

[4] B. Brejova, C. DiMarco, T. Vinar, S. R. Hidalgo, G. Holguin, and C. Patten, Finding Patterns in Biological Sequences. Unpublished project report for CS798G, University of Waterloo, Fall 2000.

[5] Celcorp, http://www.celcorp.com/

[6] M. El-Ramly, Reverse Engineering Legacy User Interfaces Using Interaction Traces. Ph.D. Thesis, University of Alberta, Canada, 2003.

[7] M. El-Ramly and E. Stroulia, Analysis of Web-Usage Behavior for Focused Web Sites: A Case Study. J. of Software Maintenance and Evolution: Research and Practice, vol.16, no. 1-2, pg. 129-150, 2004.

[8] M. El-Ramly, E. Stroulia and P. Sorenson, Interaction-Pattern Mining: Extracting Usage Scenarios from Run-time Behavior Traces. In Proc. of the 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2002), 2002.

[9] ICSE Workshop on Dynamic Analysis, 2003.

[10] ICSE 2nd Int. Workshop on Dynamic Analysis, 2004.

[11] I. Jonassen, Methods for Finding Motifs in Sets of Related Biosequences. Dr. Scient Thesis, Dept. of Informatics, Univ. of Bergen, 1996.

[12] R. Kapoor, Device-Retargetable User Interface Reengineering Using XML. Tech. Report TR01-11, Dept. of Computing Science, Univ. of Alberta, 2001.

[13] A. Floratos, Pattern Discovery in Biology: Theory and Applications. Ph.D. Thesis, Dept. of Computer Science, New York Univ., 1999.

[14] H. Mannila, H. Toivonen and A. Verkamo, Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery, vol.1, no. 3, pg. 259-289, 1997.

[15] E. Stroulia, M. El-Ramly, P. Iglinski and P. Sorenson, User Interface Reverse Engineering in Support of Interface Migration to the Web. Automated Software Engineering, vol.3, no. 10, pg. 271-301, 2003.

[16] T. Ying, Predicting Source Code Changes by Mining Revision History. M.Sc. Thesis, Dept. of Computer Science, University of British Columbia, 2003.

[17] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, Mining Version Histories to Guide Software Changes. In Proc. of Int. Conf. on Software Engineering (ICSE), 2004.