Understanding the Breadth of the Event Space: Learning from Logan

Danyel Fisher, Steven M. Drucker, Mary Czerwinski, Rob DeLine, Kael Rowan

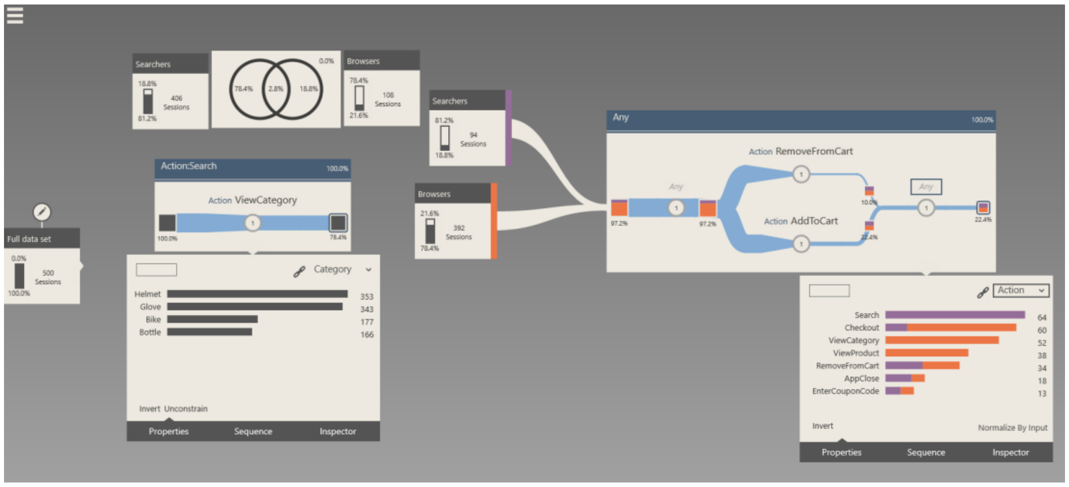

Fig. 1.The Logan Prototype: Workspace from a session exploring shopping behavior at an online store. Cohorts can be extracted, combined, and fed into queries where events and their payloads are viewed as histograms.

Abstract— Event processing, analysis, and visualization are increasingly important, and common, problems as telemetry and log recording become ubiquitous. We are still learning about the space of ways to both query and see the results of sequential queries against event-logged data. In this paper, we discuss our Logan event exploration prototype, which is based on regular-expression queries and result histograms; we then discuss the many factors that vary between tools, domains, and datasets -- tradeoffs of factors such as session size, cardinality of events, and the presence of event arguments.

Index Terms—Sequential Data, Interactive Visualization, Regular Expressions

1. Introduction

Event processing is an exciting new area for data analytics and visualization. With the rise of telemetry, with increasingly ubiquitous logfiles, and with phones that continuously log usage patterns, there are increasing numbers and opportunities for data sources based on events.

It is our belief that sequential events constitute a new structure for data, as different from other visualization types as, say, graph-oriented visualization is from bar chart and line charts. In this new world of sequential event visualization and processing, a number of new types of visualizations will become not just meaningful, but necessary to make sense of the data. There will, of course, be ways to transform these new data types into more familiar visualizations: just as we might view a tree's nodes by size on a bar chart, sequential visualization will still require classic visualization skills.

But there is a new thing going here, too: an event-driven dataset has a notion of *multiple sequences of events*; each sequence has individual events that are *timestamped*, associated with particular *identities* (such as sessions, persons, or applications), and *labelled*. We often want to ask questions about series of events and their interrelations—how often did one event occur after another, or what events followed this first event. We might drive an exploration with questions like:

- What action gets followed by "UNDO" most often?
- When do people abandon a shopping process?
- How do people recover from errors?
- Does a sequence of treatments expose side effects?

Over the last few years, we have been creating a series of systems designed to process events; users query them with a visual regular expression builder. Our systems have been used to explore logfile-based data, from a variety of different applications and usages.

Application domain makes a substantial difference to the analysis: as we explore new types of data, we have learned about new genres of questions which suggest different interactions. The fact that this discovery process continues suggests that as we learn about more problem domains – at this workshop and elsewhere – we will also learn about new types of problems.

In this paper, we try to understand the dimensions of the space of event-driven visualization. Just as a visualization for a network must be very different if the network is dense or sparse; and a tree visualization is different if the tree is deep and narrow, or shallow and wide; so too we wish to understand the dimensions along which event datasets vary. The dimensions we discuss below are the results of explorations from both building prototype designs, and of exploring and using these protoypes on datasets; they represent lessons ruefully learned, and a variety of approaches as we attempted to build systems that provided interesting results from complex event data.

All authors are with Microsoft Research.
Proceedings of the IEEE VIS 2016 Workshop on Temporal & Sequential Event Analysis.
Available online at: http://eventevent.github.io

In this position paper, we introduce the Logan system, a sequential event visualization system based on regular expression queries. We show several screens and describe its core usage. We then proceed to discuss a broader space of event-based datasets, highlighting how varying cardinalities of events and their attributes; lengths of sequences; and other attributes make for very different querying systems. In this new area, there are opportunities to explore many new ways to ask questions of event sequences.

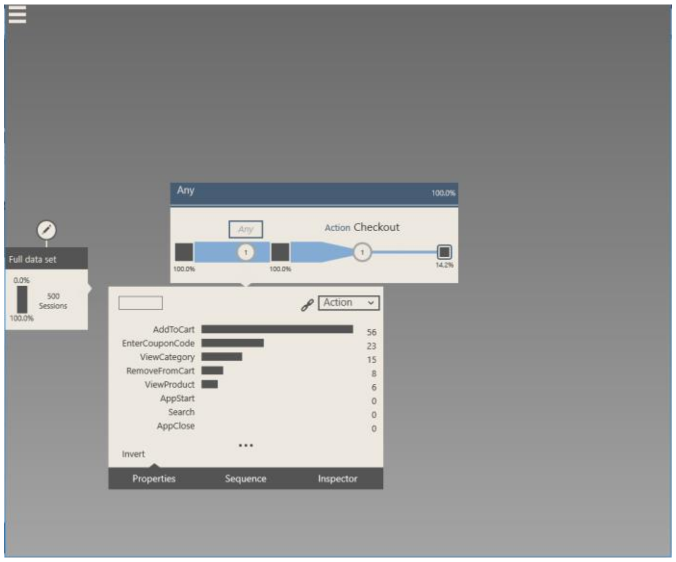

Fig 2: A two event query: Any event that immediately precedes a checkout. The histogram of the resultant actions is shown below.

2. The Logan Prototype

The Logan prototype grew out of our previous work on using regular expressions to query and explore sequential datasets [Zgraggen15]. Logan preserves the basic structure from SQueries; users start exploring by specifying a query with a graphical language that matches primitive symbols with events in a sequence. Primitives are combined together to match longer sequences (and wildcards can be used to match very long sequences).

Many sequences are very complex; as a result, Logan requires that users begin with a generic query, which they can use to refine further—by adding new clauses before or after, or by constraining a result to a single value.

The results of queries (matching sequences) are summarized primarily as histograms and queries can be modified by interacting with the histogram to constrain the query. See figures 1-3. While the SQueries system was very successful for testing hypothesis on small datasets, it suffered from a number of shortcomings that we wished to address in the subsequent Logan prototype. We address these design changes below.

Below, we discuss some examples using regular expressions. Each alphabetic letter will represent a distinct event (with possibly different arguments); thus "ABCD" refers to four different events in a row. "A.*B" uses regular expression notation to mean "an A; any number of other events; a B." In the prototype, users do not need to be aware of regular expression language since the user interface helps hide the complexity associated with specifying regular expressions.

2.1 Scalability

We wished to load datasets many orders of magnitude larger than was handled in the (S|Qu)eries work. We addressed scalability by building the prototype on top of a fast, streaming database engine [Chandramouli14] which gives continuous updates as a query is performed. In this way, we can maintain a responsive interface, updating visualizations of the data as the query is processed through a large file. Separate queries can be started even before a query is finished. This immediate feedback enables much more rapid exploration of the entire dataset [Fisher12] for details on interacting with streaming database engines.

2.2 Workspace Management and UI Enhancement

Several UI enhancements were made to help deal with awkwardness that users experienced when interacting with the (S|Qu)eries prototype. Namely, we never provided a way of working with a GROUP of elements, and each element of a query needed to be moved separately. This made managing a single query awkward, with manual rearrangement required to make room for other elements of the query. In Logan, queries automatically arrange when inserting a part of a query into the rest of the query. Layout rules based on where the query is dropped enable conjunctions and disjunctions. The entire query can then be moved around as a group making organizing the workspace easier.

Regular expressions have some unintuitive aspects for users. For example, our users had no trouble understanding that the query A.*B matches ADECB. However, we chose to allow our regular expression matching to be *greedy*; thus, AAABBB or AABABABBB would also match this sequence. Logan incorporates UI affordances for making it easier to express the query "A[^AB]B", which forces the query to match anything except A or B; this better suits users' expectations.

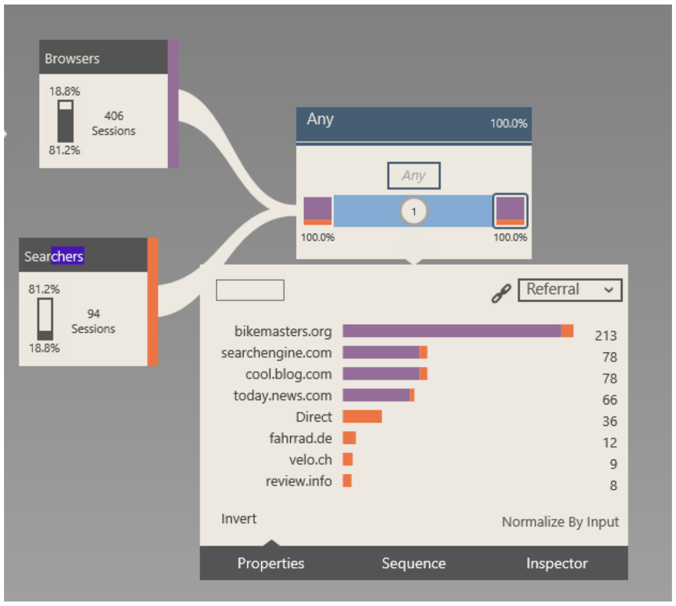

Fig 3: A comparison of 2 cohorts (Searchers and those who did not search – Browsers). We can see how they arrived at the site.

2.3 Cohorts

An important capability desired by users was to compare the behavior between different cohorts of users (or sessions). This is similar to previous work of Krause et al and Malik et al [Krause16, Malik15]. Thus if we want to look at how people in one city compared to those in another city we could create one cohort for the first set and another for the other set. We can then pipe these cohorts into a query and look at a histogram to compare these results. The histograms now include colors to include those exclusively in one cohort, those in another cohort or those in both. See figure 3. We can construct new cohorts by doing Boolean operations of cohorts. When we bring one cohort in close proximity with another cohort, a simple Venn Diagram appears and users can pull from the intersection, or the combined regions. These resultants cohorts are treated as first class cohorts when piped to other queries.

2.4 Blank Slate Problem:

In SQueries, many users were confused about where to start initial investigations, especially if they had not yet formed a hypothesis about which they could explore.

Logan provides a good starting point for people who are reasonably familiar with the events contained in a log. The opening screen presents users with a histogram of known events, making it easy to start a query with an event of interest. However, Logan does not use a full-on "overview, filter, details on demand" approach.

That said, we explored abstracting and representing groups of sessions by generating a finite state machine that could represent them all (and would appropriately leave out sessions that were explicitly excluded) [Gahani92]. By representing these sessions using the same kind of regular expression language that could generate them in the first place, we desired a summary that was amenable to subsequent filtering and narrowing down on different parts of the space. Finding an effective and efficient way of representing sequences and the appropriate level of abstraction is still being explored. Using Logan-like queries along with summary representations such as those in EventFlow [Monroe13a].

3. Points in the Event Analysis Space

In this section of the paper, we discuss the broader questions of some major issues that become clear in our domain. As we have explored different logfiles and sets of events, we have found that certain behaviors of event-driven systems can dominate discussion. This sections acts, in part, as an opportunity to compare Logan with longer-lived systems such as EventFlow [Monroe13a].

Different datasets have very different aspects – and those differences make for demands on interfaces and interactions. In our experience, we have worked with several datasets. One is a series of store transactions from the (fictional) online vendor AdventureWorks; we paid Mechanical Turk users to "buy" items from the store; another is a logfile of user actions in a suite of online visualization tools, "AgaVue"; the last is a logfile of user actions in the "SandDance" tool. [Drucker15].

2.5 Cardinality of events

In dealing with event logs, we have seen very substantial differences between collection methodologies. At one end are highly-focused logs that note very specific events; at the other, logs that have a tremendous number of different sorts of events. For example, the AdventureWorks dataset only keeps eight different events; AgaVue keeps 30; SandDance maintains a similar number. Working with some product groups, we have seen some systems that maintain verbose logs that log each function entered or exited separately.

In a system with a smaller cardinality of events, it makes sense to (for example) color each of them with a separate color in an overview panel. As the number of distinguishable events grows, it become increasingly important to have ways to filter or separate them.

Solution Spaces: EventFlow builds an opt-in system; users can select the subset of events that seem interesting to them. This has the tradeoff that interesting things might be lurking beneath an overview, missed because the event wasn't opted-into; it also allows users to merge similar events. Logan allows users to select events from a menu; it begins to creak under the weight of too many events. It's less clear what to do when there are hundreds or thousands of types of events. Perhaps a separate hierarchy of related events make this more manageable.

2.6 Cardinality of arguments

Sometimes, the arguments to an event are simple: for example, if a medical subject takes a medication at 100 mg or 200 mg daily; or a window is moved to (0,0) or (50,100). Other classes of arguments, however, can get complex. We have encountered:

- Situations where the entire meaning of the event is in the argument. "CLICK(button)", for example – while it's good to know the user has clicked, the real underlying event is the fact that the button was activated. If there are several different buttons, this becomes very important.
- Situations where the meaning of the event is dependent on the argument. For example, in the AgaVue dataset, a "TreeStats" event has a number of arguments showing that a user has just created a tree, and what sort -- unless it shows zero nodes, in which case the user has attempted to create a tree, and failed.
- Situations where the argument cannot be interpreted without another event. For example, the treestats event above means very different things if it has a very specific set of parameters (a tree of depth 2, and 11 nodes), which corresponds to sample data.

Solution spaces: Logan provides parameter linking, so the user can search for occasions when one event shares parameters with a later one. (For example, "addData(rows = <VAR>" then matches with "showData(rows = <VAR>").

2.7 Lengths of sequences

A shopping event sequence is reasonably well-defined: a user goes though the phase of finding events of interest, then of checking out. The sequence for user exploring data might be far longer: a user might play with a dataset, save and load it, or try multiple representations. An analysis that tests for the presence of reasonably short sequences might not be able to scale to a sequence of hundreds or thousands of events in a sequence. (One way to reduce that length, of course, is to filter out unneeded event types, reducing the cardinality of events at the same time.)

2.8 Scale of Dataset

From both a computational and display view, the scale of the dataset is very important: a system oriented around showing each individual sequence will require more screen space then a system that, for example, only counts sequences that match a given query. Indeed, the latter can even be set as a dashboard, with streaming queries showing how many sequences currently match the query; as more events show up, the count gets updated.

2.9 Interval and Point Events

Some datasets have logical durations—durations are a first-class citizen in EventFlow [Monroe13b]. Durations are a useful abstraction for both user operations ("the period during which the chart was open") and for medical scenarios ("the time during which the patient was in the hospital.")

In a point-oriented system like Logan, durations may be expressed as a start-event / end-event pair, and the user can then set a query for that pair of events. However, it needs to be done with some caution: a sequence like start-end-TARGET-start-end will still match the query start-TARGET-end. As a result, a system must have the ability to express concepts like "any event that is not an end event."

2.10 Origins of Events

In any event log, some events come about because the user did a thing; others come about in response to a user carrying out an action. (For example, "checked into hospital" might be started by a user, but "seen by doctor" and "results returned from radiology" are both generated by the system; similarly, "user requested new scene" is user-driven, while "system rendered updated image" is generated by the system.)

Solution space: Knowing whether an event was user-driven or system-driven can help unpack causality. This is a challenge for the log itself: it would be useful to track that "event 23 is a result of previous event 21."

2.11 Timing Events

The notion of a sequence of events is core to this discussion; however, time has not yet entered the discussion. Are two events separated by a second to be represented in the same way as two events separated by an hour? Languages need to have ways to express "no less then K minutes separate these events" or "these events must be no less then K seconds apart."

Solution space: Queries that describe sequences can be tagged with "no more than" and "no less than" constraints; this gets far more complex when reasonably-distant events need to come within a certain amount of time. In addition, "session breaking" can be used to separate pairs of events that come far apart: it may be presumed that rather than the use sitting and waiting for an extended period, they came back later to work on a session.

2.12 Arbitrary Ordering

This issue gets even more complex when we look at examples where timing is not definite: any system that involves multiple computers generating or storing logs, such as a client-server system or a web page, has a risk of events getting out of sequence. An event processing tool that focuses on sequence over timing can risk getting hung up on these events. In particular, in online systems, it is common to see "request" and "response" pairs that happen milliseconds apart. It might be very important to distinguish that ordering – or completely irrelevant.

Conversely, there are times when we are indifferent as to the ordering of two events—we don't care whether a patient got a particular test before or after they went to radiology, as long as the two are "roughly simultaneous."

Solution space: Again, this is an example where tagging the events can be helpful--this issue can be reduced as a data cleaning issue when the events are labelled with sequence numbers. A query language, optimally, would have a way to ask for "roughly simultaneous" events.

4. Discussion

In this paper, we have presented a broad perspective on event-driven visualization. We discussed the (S|Qu)eries and Logan systems, which are designed around a regular-expression approach to queries, and which use flow diagrams and histograms to show the transition of events through nodes in the expressions.

Our explorations of data sets have shown a number of different types of datasets and data issues that must be handled by event querying systems. Some of these issues, such as timestamp issues, require some hard thinking about both how to handle a sequential query, and how to create the most effective possible logs. Others of them reflect on design philosophies of diverse querying systems; various systems will explore these aspects very differently from each other.

Acknowledgments

The authors wish to thank Emanuel Zgaggren, who created the first version of SQueries and the first implementation of Logan; Alper Sarikaya, and Titus Barik have carried out research at Microsoft on event logs; Jonathan Goldstein and Badrish Chandramouli created Trill, the high-speed streaming processor; and Ben Zorn, Sumit Gulwani, and Rishabh Signh, who joined us in research on understanding the challenges involved in data cleaning logfiles.

References

[Chandramouli14] B. Chandramouli, et al. Trill: A high-performance incremental query pro-cessor for diverse analytics. In Proc. VLDB Endow., 8(4), 401–412, 2014.

[Drucker15] S. Drucker, R. Fernandez, A Unifying Framework for Animated and Interactive Unit Visualizations. Microsoft Technical Report, 2015.

[Fisher12] Fisher, Danyel, Igor Popov, and Steven Drucker. "Trust me, I'm partially right: incremental visualization lets analysts explore large datasets faster." Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2012.

[Gehani92] N. H. Gehani, H. V. Jagadish, O. Shmueli. Composite event specification in active databases: Model & implementation. In Proc. VLDB, 327–337. VLDB Endowment, 1992.

[Krause16] Krause, Josua, Adam Perer, and Harry Stavropoulos. "Supporting iterative cohort construction with visual temporal queries." IEEE transactions on visualization and computer graphics 22.1 (2016): 91-100.

[Malik15] Malik, Sana, Fan Du, Megan Monroe, Eberechukwu Onukwugha, Catherine Plaisant, and Ben Shneiderman. "Cohort comparison of event sequences with balanced integration of visual analytics and statistics." In Proceedings of the 20th International Conference on Intelligent User Interfaces, pp. 38-49. ACM, 2015.

[Monroe13a] M. Monroe, et al. Temporal event sequence simplification. IEEE Trans-actions on Visualization and Computer Graphics, 19(12), 2227–2236, 2013.

[Monroe13b] M. Monroe, et al. The challenges of specifying intervals and absences in temporal queries: A graphical language approach. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI), 2349–2358. ACM, 2013.

[Zgraggen15] E. Zgraggen, et al. (s|qu)eries: Visual regular expressions for querying and exploring event sequences. In Proceedings of the SIGCHI Confer-ence on Human Factors in Computing Systems (CHI), 2683–2692. ACM, 2015.