(s|qu)eries: Visual Regular Expressions for Querying and Exploring Event Sequences

Emanuel Zgraggen[1,2], **Steven M. Drucker**[1], **Danyel Fisher**[1], **Robert DeLine**[1]

Microsoft Research[1]
Redmond, WA, United States
[sdrucker, danyelf, rob.deline]@microsoft.com

Brown University[2]
Providence, RI, United States
ez@cs.brown.edu

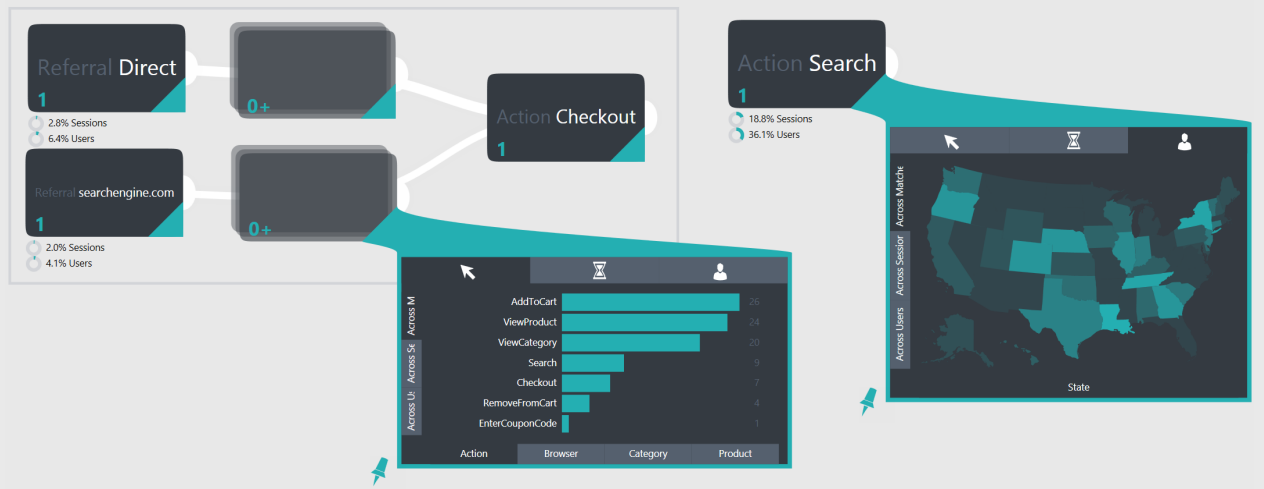

Figure 1. Two queries on a fictional shopping website web log. Left: Query to explore checkout behaviors of users depending on direct referral versus users that were referred from a specific website. Right: Query to view geographical location of customers that used the search feature.

ABSTRACT

Many different domains collect event sequence data and rely on finding and analyzing patterns within it to gain meaningful insights. Current systems that support such queries either provide limited expressiveness, hinder exploratory workflows or present interaction and visualization models which do not scale well to large and multi-faceted data sets. In this paper we present (s|qu)eries (pronounced "Squeries"), a visual query interface for creating queries on sequences (series) of data, based on regular expressions. (s|qu)eries is a touch-based system that exposes the full expressive power of regular expressions in an approachable way and interleaves query specification with result visualizations. Being able to visually investigate the results of different query-parts supports debugging and encourages iterative query-building as well as exploratory work-flows. We validate our design and implementation through a set of informal interviews with data scientists that analyze event sequences on a daily basis.

CHI 2015, April 18–23, 2015, Seoul, Republic of Korea.

ACM 978-1-4503-3145-6/15/04 ...$15.00.
http://dx.doi.org/10.1145/2702123.2702262

Author Keywords

Visual query languages; event sequences; query interfaces; visual analytics; gesture interfaces

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation: Miscellaneous

INTRODUCTION

Event sequence data sets are collected throughout different domains: electronic health records in medical institutions, page hits on websites, personal data from "Quantified Self" technologies, eye tracking and input device logs during usability studies or program execution logs through telemetry systems to name just a few. The ability to efficiently explore and analyze such sequential data can generate deep insights and support informed decision making. Examples range from understanding application usage patterns, optimizing marketing campaigns, detecting anomalies, to testing of research driven hypotheses. HCI researchers and practitioners in particular can benefit from event sequence analysis, since usability issues and user performances can be identified and extracted from telemetry data or other logs [8]. The complexity and size of such data sets is continuously increasing and insights might be hidden within the natural ordering or relationships between events, at different levels of detail and in different facets of the data. For example, consider a smart phone

application that logs user interactions. A usability manager might want to see high-level aggregations of user flows to uncover possible UX related issues, whereas an engineer might be interested in what led to the triggering of an obscure error code in the few specific sessions where it happened.

While the types of questions analysts and researchers might want to pose vary drastically across domains and scenarios, they often can be reduced to finding patterns of events and being able to inspect the results of such queries. Visualizations are a great way of leveraging the human visual system to support pattern finding. However, for large data sets it is hard to detect insights that are hidden within sub-populations or even single instances or to spot patterns with some degree of variation. Other systems that support querying more directly are either not well suited for this type of sequential data (i.e., SQL), offer limited expressiveness, are hard to learn, or discourage fluid, iterative and interactive exploration.

In this paper we introduce (s|qu)eries, a system that transforms the concepts of regular expressions into a visual query language for event sequences. By interleaving query specification with result visualizations, this integrated interface allows users to incorporate the output from an existing query into the parameters for the next query. We developed (s|qu)eries with a touch-first mindset which resulted in a design where visual elements are a direct component of the interface and indirection is kept to a minimum. Queries, such as the ones in Figure 1, can be built and manipulated fluidly with direct manipulation based touch gestures. While our sample application and users focused on telemetry data, the concepts translate to other domains where users need to both spot and investigate patterns.

We started off by analyzing the results of a recent internal study that interviewed 10 data analysts across different functions (UX designers and researchers, program managers, engineers and business stakeholders) from a telemetry group, discussing common formats for log data, understanding the tools that they currently use and, most importantly, finding the types of questions that they typically attempt to answer from their logs. From this point, we iteratively designed the (s|qu)eries system to ensure that it was flexible enough to answer their questions, but also fit into their current workflow.

We validated our approach through detailed interviews with members of five different data science groups within a large data-driven software company. All of the participants analyze event sequences frequently as part of their function within the company. During the interviews we asked the participants to explore their own data through (s|qu)eries. Even though questions, data and backgrounds varied greatly between those teams, we found that all of them were able to uncover previously unknown insights after minimal initial training.

The contributions of this paper are three-fold. First, we present a design for a visual query language for event sequences based on the full power of regular expressions. Second, we incorporated visualizations that allow users to interactively explore the results of both their overall queries as well as individual parts of the queries. These visualizations can be used, in turn, to further add to or refine their queries. Third, we investigate the effectiveness of the system through an informal study with five different data science groups within a large, data-driven software company.

RELATED WORK

We relate and contrast this work to research efforts in the areas of Event Sequence analysis, Temporal Data Visualizations, Query Languages for Event Sequence & Temporal Data, and Touch-based Interfaces for Visual Analytics.

Event Sequence & Temporal Data Visualizations

Much of the work on event sequences visualization is in the domain of electronic health records. Work in this area is applicable to other event sequence domains like telemetry data, where the payloads of events are different but the visualization strategies are not. While in the medical domain, questions might be about what combination of treatments eventually lead to a stroke, in software telemetry, data scientists might pose similar questions about what events lead to a crash of the application. They both require users to find and analyze sequential patterns within event sequences.

LifeLines [20, 21] is seminal work on how to depict the medical history of a single patient. The tool allows users to visually spot trends and patterns within a single patient's history, or a session when looking at telemetry data. Many other systems [13, 9, 3] have used analogical time-line based representations to visualize single event sequences. By ordering pattern-parts from left to right, (s|qu)eries visualizes sequential patterns in a way that is similar to time-lines, but aggregates across multiple event sequences.

LifeLines2 [26] extends this concept to multiple patient records. Spotting patterns within the data can be achieved by aligning all records according to a sentinel event. However, users are still burdened with visually scanning the entire data set, which often makes pattern detection across large data sets, a cumbersome and error prone task. Other systems [27] address this issue by offering time or category based aggregation techniques. While these approaches scale better to moderately sized data sets, they might hide interesting outliers or pattern variations within the aggregations and are susceptible to noisy input. (s|qu)eries aggregates based on patterns and therefore allows users to explore event sequences at different levels of detail (i.e., from high-level aggregates to individual event sequences) and to capture and hide noise within single nodes.

Sankey diagrams [22] statically illustrate flows and sequential dependencies between different states, however aggregation and alignment techniques are needed to make Sankeys useful for spotting patterns and to reduce visual complexity. (s|qu)eries uses aspects of Sankeys (such as scaling the width of links between nodes) to help visualize queries as well as the results.

Commercial products for web log analysis or log analysis in general such as Google Analytics[1] or Splunk[2] are built to

[1] http://www.google.com/analytics

[2] http://www.splunk.com/

scale to big data sets and to accommodate data from various sources. Their aggregated dashboard-like visualizations often hide patterns of sub populations and hinder exploratory and interactive workflows. SessionViewer [15] proposes a multiple coordinate-view approach to bridge between different levels of web log aggregations and supports state-based pattern matching through a textual language. However, SessionViewer's pattern language does not support simultaneous exploration of multiple queries nor is it directly coupled to result viewing.

Query Languages for Event Sequence & Temporal Data

While some systems rely on the human visual system to spot patterns in sequential data, others take a more query driven approach. The expressiveness of query systems varies greatly and is dependent on the type of supported data (point or interval events). Most work in this area is based on the concept of temporal logic proposed by Allen [2], which introduces 13 unique operators that describe temporal relationships between intervals. Currently, unlike other systems, (s|qu)eries focuses on sequences of point events (e.g., "a happens after b") instead of temporal queries (e.g., "a happens within x minutes of b").

Jensen et. al. [10] and Snodgrass et. al. [24] both propose extensions to SQL to support predicates for querying time periods. While offering a high degree of expressiveness, their command-based query languages come with a steep learning curve and are not well suited to exploratory workflows.

Jin et. al [11, 12] have addressed some of those problems by proposing a visual query language that is easily understood and specified by users. However, their comic strip metaphor is not well suited for expressing longer repetitions or parallel (alternate) paths. Numerous efforts [7, 16, 18, 19, 4] originated from work in the domain of electronic health records and present visual query languages with varying degrees of expressiveness and for different tasks. For example, PatternFinder [7] describes temporal patterns through events and time spans and visualizes the result in a ball-and-chain visualization. Others, such as Lan et. al. [16] propose extensions to Monroe's et. al. query language [18], including repetition and permutation constraints on point and interval events and absences thereof. Query specification in all of those systems is done through dialog heavy user interfaces and none of them directly interleaves query specification with result viewing.

In a complementary approach from the data mining community, work by Agrawal and Srikant [1, 25] in this domain introduces algorithms to automatically extract common sequential patterns from logs, rather than using user specified patterns to query the data. Such extracted patterns could serve as a starting points for further interactive exploration within (s|qu)eries.

Touch-based Interfaces for Visual Analytics

While we have not deeply explored the potentials for touch first interaction in this paper, their influence on the design came up numerous times. This 'touch-first' mindset (keep indirection at a minimum, animate changes to the UI, and make sure that interactive visual elements can be interacted with directly) was inspired by the work of Elmqvist et. al. [6] and Lee et. al. [17], who emphasize the importance of interaction for information visualization and visual analytics and propose to investigate new interaction models that go beyond traditional WIMP (windows, icons, menus, pointers) interfaces. "Fluid" interfaces, as proposed by [6], are not only subjectively preferred by users, but also enhance their performance (speed and accuracy) for certain data related tasks [5].

(s|q)ueries's visual query language makes extensive use of the direct manipulation paradigm [23]. Nodes, the main visual element in (s|qu)eries, can be manipulated to build queries as well as to analyze query results - they are direct handles to specify input as well as to interact with output. All interactions within the system are made through simple touch gestures. For example, in order to update a pattern, a user simply adjusts the physical layout of nodes by dragging on the node. Other such interactions include using the visualizations to constrain or build new queries or inspecting nodes to investigate the match set at various positions.

THE (s|qu)eries SYSTEM

We started the design process of (s|qu)eries by analyzing the results of a recent internal study at a large software firm. The study included 10 data analysts across different functions (UX designers and researchers, program managers, engineers and business stakeholders) from a telemetry group. It discuses common formats for log data, tools or combination of tools that are being used, and, most importantly, lists questions that stakeholders would like to answer from their logs. A portion of these are shown in Table 1.

The two key findings we extracted from analyzing questions like the ones in Table 1 are:

- Providing answers to questions is a two step process. First, the pattern of interest must be found across the full data set. Second, the relevant attributes of pattern matches need to be presented in an interpretable way.
- Question come in batches. Questions are often related to each other and / or are built upon answers from the previous ones.

These two findings manifest themselves directly within (s|q)ueries's design. (s|qu)eries exposes the power of regular expressions as a pattern finding language for sequential data through a visual language and interaction design such that users can interactively explore results of matches and use them as starting points for successive queries.

Introductory Use Case

We motivate our approach through an introductory use case that is based on some of the sample questions. Adam is a data scientist at a big e-commerce firm. The company just released a new smart phone application where users can search and browse for products and buy them directly. Adam wants to check if the company's expectations in terms of acquiring new customers were met and if there were any problems with the application. He got a dump of last week's client

Question
What is the typical first interaction when users open the application?
What is the most/least often used component of the application?
What is the frequency with which the different features are accessed?
What is the frequency for feature X per user?
How many users perform workflow X?
How long does it take users to perform workflow X?
What are common technical problems?
What leads to common technical problems?
What are common attributes about users of the system that exploit a particular feature?

Table 1. Sample of common questions gathered by interviewing data scientists who explore software telemetry data.

side telemetry data that the application logs. The sessions, or event sequences, within this data set look something like this:

```
Session,User,TimeStamp,Attributes
4,3,8/1/14 3:01,action=AppStart
4,3,8/1/14 3:02,action=Search&query=Helmets
4,3,8/1/14 3:11,action=ViewProduct&query=Red Bottle
4,3,8/1/14 3:13,action=AppClose
```

Adam starts (s|qu)eries on his touch-based laptop, double-taps the background to create an unconstrained query node that matches any type of events and inspects the node by opening up its result view (Figure 2a). Adam sees a histogram showing the most common actions in the data set. Since he is interested in customers who buy products, he decides to drag out the "Checkout" action (Figure 2b). The system creates a new node pre-constrained on "Action = Checkout". Immediately, he sees that 14.2% of sessions resulted in successful checkouts.

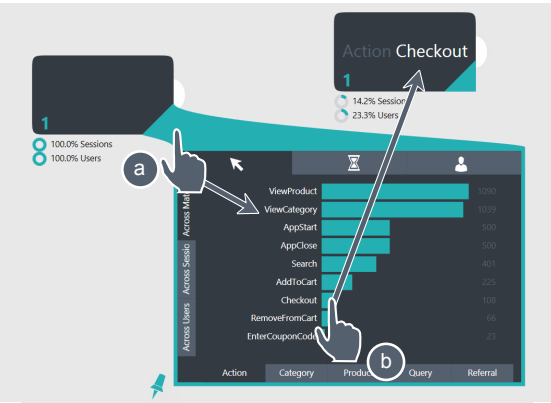

Figure 2. Opening up visualization view of a node and dragging from histogram to create a new constrained node.

Naturally he wants to know what led to users successfully checking out. He creates another node and links the two nodes together. This query allows him to look at all the events that happened immediately before people checked out.

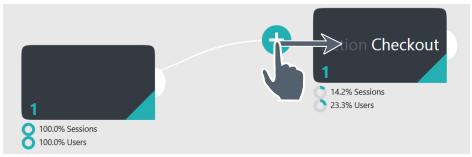

Figure 3. Linking of two nodes.

He opens up the visualization view again. Most commonly people added something to their shopping cart before they checked out which seems reasonable.

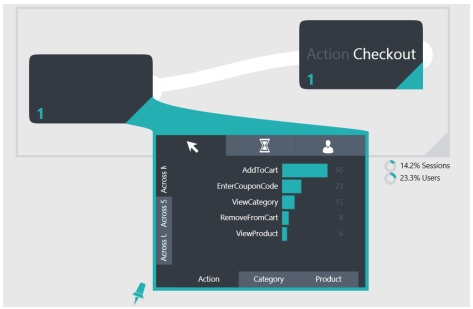

Figure 4. Inspecting part part of a sequential two node query.

But what about after checking out? Did users perform any other actions after they bought something? He switches the ordering of the two nodes and the visualization updates.

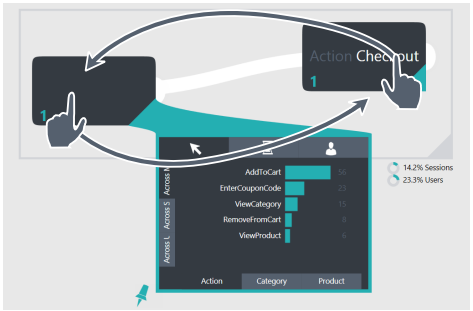

Figure 5. Rearranging nodes to create a new query.

Interestingly enough, there are some people that added other items to their cart even though they already checked out - that seems odd. He selects the "AddToCart" item, which in turn constrains the second node and allows Adam to further explore this unexpected user pattern to check if it is an error in the application or some other anomaly. This leads to more fruitful exploratory analyses.

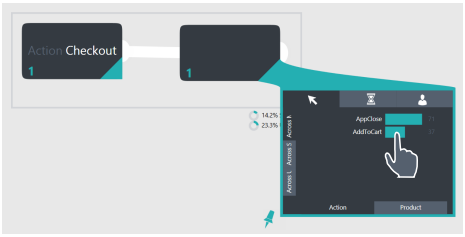

Figure 6. Selecting item in histogram to constrain a node.

Data Model

Like Splunk and other systems, (s|qu)eries operates on event sequences, broken into **sessions** by **user**. A single **user** reflects a logical user of the system: in a telemetry log, for example, a **user** might be a single login to the system. A user, however, may log in and out of the system; it can be

valuable to break a users events into **sessions**, which represent the time from login to logout. A single **session**, then, is made of a sequence of **events**. Each event is associated with a **user id**, a **session id**, a single **time-stamp**, and a series of **attribute-value pairs**, which describe that event.

In the use case above, the event sequence represents all the logged actions on the phone application; sessions begin when the user started the app, and end when they close it. An event holds all the information needed to perform its action. For example, when the user adds an item to the cart, the system might log the following three attribute-value pairs: ("action", "AddItemToCart"), ("timestamp", "12/21/2014 21:12:44"), ("item", "city bike"). (s|qu)eries builds its internal data model on the fly from the input file by aggregating along attribute-value pairs. While our current prototype expects a time-stamp, user and session information, which are particular to telemetry data, the data model is flexible enough to accommodate data from many domains as long as events can be represented as a collection of attribute-value pairs.

Query Language

In order to find answers to questions like those in Table 1 users need an efficient way to query sequential data. Such queries can be thought of as patterns over event sequences, where different parts of the pattern can be more or less constrained. Log data often contains a large amount of noise and a query language therefore needs to provide support to express variations and fuzziness.

The same applies to extracting patterns from character sequences. Regular expressions are widely used for that task. While they present a powerful language for finding sequential patterns, their text-based representation can be hard to understand and they do not offer support to explore results without additional tools.

By defining our patterns in terms of regular expressions, we leverage an extensive literature of what can and cannot be done within this framework and, if necessary, users can use the literature to form particularly complicated expressions. However, we designed the system such that users do not need to be intimately familiar with regular expressions in order to use the system effectively.

In the following sections we formally describe regular expressions and its base operations. We then discuss how these base operations manifest themselves in our visual query language and how users can specify and interact with them. Furthermore, we show how these visual primitives can be combined to create sophisticated queries over event sequences and serve as direct handles to view results and as starting points to incrementally build up and modify queries.

Regular Expressions

Regular expressions are a well known formal language, introduced by Kleene [14]. In popular tools like `grep`, regular expressions are used for matching patterns within text strings. Before describing how (s|qu)eries uses regular expressions to describe events, here is a short review of the constructs of regular expressions, in the familiar domain of text matching:

1. *Concatenation* : A way of specifying that character follows another character. For example, `abcd` matches "abcd", but not "abdc".
2. *Alternation*: A way of expressing choices. For example, `a|b` can match either the string "a" or "b".
3. *Grouping*: A way of defining the scope and precedence of operators. For example, `gray|grey` and `gr(a|e)y` are equivalent patterns which both describe the strings "gray" and "grey".
4. *Quantifiers*: A way of indicating how many of the previous element is allowed to occur. The most common quantifiers are:
   - The question mark `?` matches 0 or 1 of the preceding element, For example, `colou?r` matches both "color" and "colour".
   - The asterisk `*` matches 0 or more of the preceding element. For example, `ab*c` matches "ac", "abc", "abbc" and so on.
   - The plus sign + matches 1 or more of the preceding element. For example, `ab+c` matches "abc", "abbc", "abbbc", and so on, but not "ac".

In addition to these fundamental constructs, tools like `grep` also provide handy shorthand notations for wordy alternations.

1. *Wildcard*: A way of expressing a choice of any character. The wildcard `.` is shorthand for an alternation that includes every character in the alphabet.
2. *Ranges*: A way of expressing a choice based on the order of characters. For example, the wildcard `[a-z]` is shorthand for `a|b|...|z`. Negation (set complement) is also common, e.g. `[^0-9]` represents any character except a digit.
3. *Backreferences*: A way repeating previously matched patterns. For example, the wildcard `([a-c])X\1` is shorthand for `aXa|bXb|cXc`.

For normal text matching, the alphabet contains all characters from a known set like ASCII or Unicode. Sophisticated text matching expressions can be built out of a combination of the above patterns. For example, the following expression matches email addresses:

```
[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+[.][a-zA-Z.]+
```

In (s|qu)eries, instead of directly matching characters in the log file, we represent each **event** as a unique character in a large alphabet based on attribute-value pairs. Each "character" in this alphabet is a fully described event with concrete values for every attribute. For example {("action", "search"), ("query", "city bike")} is a single character in the alphabet. An event's user id and session id are not part of its character representation; rather, they are metadata about that character, much as the line number and column number of a character X in a file are metadata about the X and not used in matching. The events' time-stamps define the ordering of event characters, just as textual characters appear in an order in a text file. Even though the characters of this event alphabet are more complicated than text characters, the regular expression constructs above are well defined for them.

Figure 7. Shows nodes in different configurations, with their similar regular expression syntax. a) Constrained node that matches one event, with attribute Action = Search. b) Unconstrained node that matches none or one event of any type. c) Unconstrained node that matches 0 or more events of any type. d) Constrained node that matches one event, with not (white line indicates negation) attribute Action = Search and attribute Browser = Firefox 32.0.1 or IE 11.0. e) Matches sequences that start with one or more events with Action = Search, followed by a one wild card that matches one event. f) Matches event sequences that start with either an event where Action = Search or Action = ViewPromotion, followed by an event with Action = ViewProduct. This pattern is encapsulated in a group (gray box). g) Backreferencing: the chain icon indicates that this pattern matches sequences where some product was added to the cart and then immediately removed again.

Visual Query Language

Unlike in string-based regular expressions, (s|qu)eries uses a visual language to describe patterns. The main visual elements are nodes, drawn as rounded rectangles. In terms of regular expressions, a node is a character wildcard with a quantifier. The character wildcard is drawn in the middle of the rounded rectangle and the quantifier in the lower-left corner. We call a node *constrained* when some attribute-value pairs are specified and *unconstrained* otherwise. Figure 7b shows an example of an unconstrained node that matches any event; whereas Figure 7a shows a node that matches any event with the attribute-value pair ("Action", "Search"). Nodes can be constrained on multiple attributes and multiple values per attribute (Figure 7d). Constraints on nodes are specified either through a menu-like dialog by tapping on a node or through selections in visualizations.

Nodes can be linked together to create more complex patterns by dragging a node's link handle (white semicircle on the right side of a node) to another node. When linking nodes together, vertical and horizontal placement are both significant: left-to-right placement describes the concatenation of individual nodes; top-to-bottom placement expresses precedence in alternations. The pattern formed by Figure 7f shows both of these.

Each node has a quantifier applied to it, whose notation is intended to be more intuitive than the standard quantifiers: "1" matches exactly one event (E), "0/1" matches none or one event ($E?$), "0+" matches zero or more events ($E*$) and "1+" matches one or more events ($E+$). Quantifiers are represented textually as well as visually (transparency and node-stacks). Users can change a node's quantifier by tapping on the quantifier label in the lower left corner. Figure 7 shows examples of these quantifiers and their visual style.

Additionally, (s|qu)eries supports two common shortcuts: negation and backreference. The former is used to match events that do not comply to the specified constraints (Figure 7d shows an example). The latter is used to find matches where an attribute has the same value across two or more nodes (Figure 7g). Furthermore, nodes that are linked can be arranged into groups[3]. (s|qu)eries displays such groups with gray boxes around the node collections (Figure 7f).

(s|qu)eries features an unbounded pan and zoomable 2D canvas where users can lay out nodes in a free-form fashion. The concepts outlined above, such as constraints on nodes, quantifiers as well as linkage and position of nodes, can be combined to form arbitrary complex regular expressions over the input event sequence data to find patterns.

Result Visualization

The system runs patterns, formed by the visual language, against the underlying event sequence data set and will return event sequences or parts of event sequences that match it. For example, the pattern formed by Figure 7e will match all sub event sequences where one or more sequential events have an attribute-value pair of ("Action", "Search") and are followed by any other event. We will call this retrieved set of sub event sequences the match set.

Most regular expression implementations offer a concept called capturing groups. It enables extraction of the characters that matched a specific part of the entire regular expression pattern. We use this concept to assign events of the match set to nodes within the visual query pattern. Each node (and group) acts as a capture group and therefore knows which events of the match set it was responsible for matching. The nodes thus become visual handles to explore the results of the pattern query they build up. This represents a powerful tool for event sequence analysis, because users can now explore questions like what happens after a sequence of "Searches" by directly inspecting a node (rightmost node in Figure 7e).

The match set of each node (and group) can be inspected by pulling out its bottom right corner through a drag interaction (Figure 8a). Depending on the amount of pullout the user performed, the pullout view either shows a detailed view of match percentages and absolute numbers or it shows a full visualization view (Figure 8b and c).

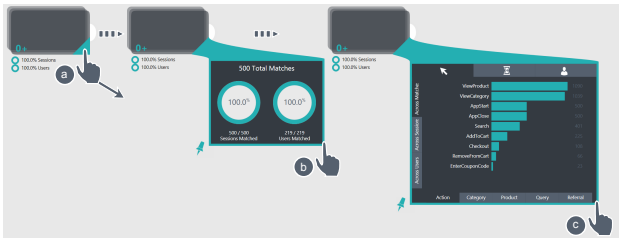

Figure 8. A node can be inspected to analyze its match set.

[3]While our design supports grouping of nodes, it is not fully implemented in our current prototype. Within the prototype, groups around entire queries are supported, while nesting of groups is not.

This second view acts as a portal to explore different dimensions of the match set. It is structured into three main tabs (Figure 9a) which allows users to switch between visualizations of attribute-value pairs, time dependent information and user information. The first one displays histograms of different and specific attribute-value pairs of the match set. The user can toggle between these histograms by using the attributes on the bottom of the view (Figure 9c). The number of attributes that are explorable is dependent on the payload of the matched events. A node that is constrained to the attribute-value pair ("Action", "Search"), will display different attributes than one that is constrained to ("Action", "AddToCart"). Events that match the former might have an associated query term, while the latter might have a product name. From analyzing the questions data scientists posed, we found that results oftentimes needed to be aggregated in different ways. A pattern can occur multiple times within a session and a user can be associated with multiple sessions. Therefore, aggregating either across matches, sessions or users (Figure 9b) can reveal different insights. An application feature might seem to be used frequently only because a few select users constantly use it, whereas across all users, it might not be that popular.

In order to encourage exploratory behavior and to allow users to seamlessly switch between query specification and result, these histograms are interactive. Tapping on an item in the histogram view constrains the node to the selected attribute-value pair, while dragging out an item creates a new pre-constrained node.

The second main tab displays time stamp related information of the match set, such as a heat map of hours of the day and day of the week of matches (Figure 9) or a histogram of match durations (from time-stamp of the first event to time-stamp of the last event). It also shows views of the list of actual events sequences that were matched, grouped by length of the matched sequence. And finally, the third tab visualizes information about the associated users of matches like their home states (Figure 1 right) if geocoded data is available.

Additionally, some high level aggregates of the match set are overlaid right within the visual query language itself. Quick previews of match percentage are displayed at the bottom of nodes as well as line thicknesses are adjusted to show flow amongst branches (Figure 7f).

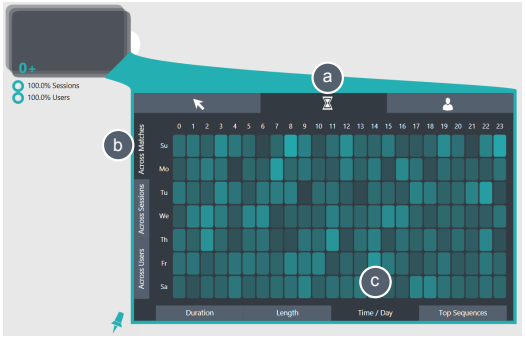

Figure 9. The visualization view has three tab navigators to inspect different attributes and aspects of the match set.

Visualizations are currently hard coded: any attribute can be interpreted as a histogram; "TimeStamp" and "UserLocation" can be interpreted for temporal views, timelines and map-views. For now, the source code must be updated to accommodate new visualization types which might be based on certain types of attribute-value pairs or to incorporate visualizations targeted towards a different data domain (e.g., electronic health records).

Implementation & Scalability

Our prototype is developed in C# and WPF and runs on both touch-enabled as well as mouse and keyboard Windows 8 devices. All computation and pattern matching is done in-memory. The system parses the visual pattern into text based regular expressions and utilizes C#'s regular expression engine to extract pattern matches from the input data set. In order to do so, the system also converts all input sequences into long string representations. After extracting matches and linking them back to the in-memory representation of events, the system computes all aggregated statistics that are used for the visualizations. Pattern matching and aggregation is outsourced to a separate thread to guarantee interactivity while processing queries.

Our prototype works at interactive speeds for real world telemetry logs with up to $30,000$ sessions, with an average length of 150 events, each with payload of around 20 dimensions. While our in-memory approach does not scale to large data sets, the visual language and visualizations do, and the nature of the data makes the required computations an intrinsically good fit for parallelization. These are especially appropriate for map-reduce frameworks, where in the map-step we can compute the regular expression matches and then construct aggregations for the visualizations in the reduce step.

EVALUATION

To evaluate our approach, we invited five people, all of whom analyze event sequences frequently, and encouraged them to bring a colleague along with them for detailed interview sessions. We felt that having colleagues there would help stimulate discussion about the tool. Two of them ended up bringing a colleague along with them. The participants were recruited from five different teams form a large software company. While all of the participants work with telemetry data, we ensured that we had broad coverage with different backgrounds as well as varying goals for potential insights from the tool. To create a familiar scenario and to test (s|q)ueries's ability to operate on different real-world data sets, we asked our participants to send us a sample of their data beforehand. We required the data to be sessionized in a form that made sense to them and limited the number of sessions to $1,500$ with no more than 200 events per session.

We initially gathered some background data on the subjects including their programming expertise, tools they currently exploit in their job and what kinds of insights they were trying to gain from the log data. These are summarized in Table 2.

After a 10 minute introduction and demo of the system on a synthetic test data set, we asked them to answer some questions by creating queries on the same test data set to familiar-

User(s)	Background/experience	Product	Current Techniques	Insights Sought
Alice	A Product Manager comfortable with programming and familiar with regular expressions but not an expert programmer	Online productivity application 1	R, Python, SQL	Test coverage of features within the application and current application shortcomings.
Brigitte and Charlie	Brigitte was a non-programmer while Charlie had a background in programming and Information Retrieval	Online productivity application 2	R, Python, SQL for matching sequences, Excel and R for visualization and modeling	Reproducing crashes from the log data.
Dorinda and Eric	Data scientists without strong programming backgrounds	Online source control repository	SQL Services, SQL query analyzer, dashboard in SQL Server Reporting Services	Understanding end-to-end usage, justifying business cases for separating out features.
Frank	Developer on the backend server system supplying analytics for data scientists	Social Media Analytics Service	SQL, Hadoop, Sumologic	Understanding performance problems in the system from telemetry data.
Gabriella	Product manager with a little programming experience, not deeply familiar with regular expressions	Web Analytics Platform	C#, homegrown visualization tools	Exploring navigation confusion in current offerings, trying to find features to add to their own web analytics platform.

Table 2. Summary of users from evaluation

ize themselves with the tool. Some example questions were of the form: What was the most common event that occurred immediately before "AddToCart"? How many people used "Search" more than once? What was the most common query term for "Search"?

In the second part of the interviews, we encouraged them to use the system to look at their own data. Since we were interested in seeing if they could create queries that reveal previously unknown insights, we instructed our participants to explain to us what kind of queries they were constructing and if the results confirmed some already known fact or not. This also ensured that we could check if their mental model was matching with what the queries actually expressed. All the sessions were voice-recorded and were conducted on a 55" vertical multi-touch display.

Results

Overall

All the users were able to grasp the system and answer questions after the 10 minute introduction. There was universal positive reaction to the tool with quotes like Charlie's, "This has big potential!" or Gabriella's, "Can we ship this now?". While people with strong programming background were quicker at learning the tool, even those without a programming background were able to use it to extract insights from the test data set as well as from their own data. One user (Brigitte) commented that anything that she was able to understand to this extent in 10 minutes was easy enough for her to use more often, unlike formulating raw queries in SQL or other tools which she tended to avoid using.

The users that came in pairs commented that they particularly enjoyed the touch, large display aspects of the tool since it was well suited to collaboratively exploring the data. Typically each would take a turn getting hands-on with the data queries while the other sat back and made suggestions.

All the users needed to be interrupted in the exploration of their own data after 30 minutes, and all requested to use the tool on their own.

Insights

There were several specific insights that users had while working with their own data.

Alice was able to use (s|qu)eries to **find out what preceded or followed events that were important to her**, for instance she noticed that one of the common events that happens before an "undo" was copy. That was not an undoable action, so she was concerned that this was not a path that they had tested for. Another common action after "undo" was another "undo" so she went on to explore what were common circumstances that occurred before multiple "undos". This lead her to realize that pasting was often undone, even after several actions occurred. Both of these situations were surprising insights to her.

Bridget and Charlie **used constraints to isolate an exceptional case** - they showed that one feature crashed only on one version of the product. This confirmed a hypothesis that they had previously made. They explained that this had been recently added to that version of the produt and were not completely surprised.

Dorinda and Eric were able **to explore behaviors across longer, "end-to-end", sequences of actions.** They found that nearly half of the people used the "Work Item Tracker" without using the Version Control System. This was significantly more than they expected and was a good justification for unbundling the two features from each other. They had suspected this, but had not gotten clear proof of this before.

Gabriella **identified a potential broad audience of users** - web analytics managers, who ask questions like these frequently, but were often frustrated in trying to answer things beyond those put into standard, regular dashboards or reports.

Problems and Challenges

There were also some problems or challenges encountered.

Data sets were not always well formed for extracting insight. For instance, Brigitte and Charlie's log data included events that they wished to ignore. While this could have been preprocessed out of the file, they did not realize this until too late. Queries *could* be constructed that specifically ignored those events, but this was somewhat tedious using the current interface. They requested a feature that would allow them to selectively filter out events or combine certain multiple events into a single event.

Occasionally, regular expression rules could lead to some confusion over the results. Frank discovered that when a

wildcard was used between two different events, he assumed that this meant that the event did *not* occur between the two events as opposed to matching the longest sequence possibile between the two events. For instance, if the pattern was A.*B, then this would match AAAAAB or ABBBBB or even ABCDAABCBAAB.

Everybody wanted the system to scale to support bigger event sets since most had data sets hundreds or thousands of times larger than those they explored in the tool.

DISCUSSION & FUTURE WORK

While the current prototype has already proved itself useful both to practitioners in the informal user study and in continuing analysis efforts, there are a number of limitations that can be addressed in future work. Some of which involve small, iterative improvements in the design, and some which would involve more major changes.

Basing the system on regular expressions, which are well-understood in the literature, makes its capabilities and limitations predictable. Regular expressions, unlike push-down automata, can not keep track of state, and therefore certain questions can not be answered if they are not explicitly contained as payloads in the event stream. For instance, while a log might contain events where items are added to the cart, the user can not retrieve the total contents of a cart unless there is an explicit representation of the contents within the payload of the event stream. Furthermore, regular expressions themselves can be notoriously confusing. This is somewhat mitigated in the system by not requiring a cryptic textural stream, (e.g., the email matching regular expression mentioned previously). It also helps that the contents of various nodes in the query can be explicitly examined and the patterns that they match can be explored with the visualizations. In the (s|qu)eries system, the only confusions that came about were because of the greedy nature in the regular expression matching (as discussed in the results above). This also occurred when using alternations - when the top node was a wild card, by definition, no matches were left for nodes lower in the list. One user (Gabriella) did not even realize that the system was based on regular expressions until after we discussed it with her towards the end of the session. She was only vaguely familiar with them and was still completely capable of using the system to answer our test questions and questions on her own data.

The system at this time lacks the ability to add a temporal element to the queries. This would be an exciting and powerful additional capability but has not yet been addressed in the current prototype. Temporal constraints would interact with the regular expressions in an interesting fashion, but could probably be added as a further post-process on the results of a set of matched events.

From the interview with Dorinda and Eric, it became clear that logs often have erroneous or lower level events that need to either be completely filtered out or combined into single, higher level events. Search and replace capability, similar to [16], could add great power to the (s|qu)eries system.

One of the current benefits of the (s|qu)eries system is that the results of the queries can easily be constrained to help create more queries. We want to further expand this interaction so that the system could either automatically display commonly pathways, or so that multiple paths can be interactively constructed by dragging out several results from a node to create separate queries.

Many, small usability improvements could also be made. People did not like starting with a blank canvas, so pre-populating the canvas with at least a single node that shows the most common events in the log would be useful. We wish to allow for saving and loading of queries so that sessions can be shared or continued.

Finally, expanding the capabilities of the visualizations can be done in many different ways. Specific vertical visualizations that explore certain payloads or combine the results of a query with other tables (like geocoding from IP lookup or heat maps of features used in a program) could be added. Several users wanted to contrast the results of multiple queries (e.g., What were the differences between what people added to a cart for those people that used search and those people that didn't?)

CONCLUSION

As sequential logs across numerous domains explode in availability, we see that there is great potential in enabling domain experts who may be non-programmers. Requiring a programmer in the loop greatly limits the exploratory possibilities that the readily available data facilitates. We plan on continuing to expand the capabilities of the (s|qu)eries system as we apply it to other users, other questions, and other domains.

We introduced (s|qu)eries, a touch-based system that features a visual query language building upon the full power of regular expressions. Our system incorporates visualizations that allow users to interactively explore the results of both their overall queries as well as individual parts of the queries. These visualizations can be used, in turn, to further add to or refine their queries. We investigate the effectiveness of the system through an informal study with five different data science groups within a large software company.

ACKNOWLEDGMENTS

The authors wish to thank Dan Zhang for his input on telemetry analysis, the reviewers for their suggestions, as well as all our participants for their time and invaluable feedback.

REFERENCES

1. Agrawal, R., and Srikant, R. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, IEEE (1995), 3–14.
2. Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 11 (1983), 832–843.
3. Bade, R., Schlechtweg, S., and Miksch, S. Connecting time-oriented data and information to a coherent interactive visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM (2004), 105–112.

4. Chittaro, L., and Combi, C. Visualizing queries on databases of temporal histories: new metaphors and their evaluation. *Data & Knowledge Engineering 44*, 2 (2003), 239–264.
5. Drucker, S. M., Fisher, D., Sadana, R., Herron, J., and schraefel, m. Touchviz: A case study comparing two interfaces for data analytics on tablets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 2301–2310.
6. Elmqvist, N., Vande Moere, A., Jetter, H.-C., Cernea, D., Reiterer, H., and Jankun-Kelly, T. Fluid interaction for information visualization. *Information Visualization 10*, 4 (2011), 327–340.
7. Fails, J. A., Karlson, A., Shahamat, L., and Shneiderman, B. A visual interface for multivariate temporal data: Finding patterns of events across multiple histories. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, IEEE (2006), 167–174.
8. Fourney, A., Mann, R., and Terry, M. Characterizing the usability of interactive applications through query log analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 1817–1826.
9. Harrison, B. L., Owen, R., and Baecker, R. M. Timelines: An Interactive System for the Collection and Visualization of Temporal Data. In *Graphics Interface* (1994).
10. Jensen, C. S., and Snodgrass, R. T. Temporal data management. *Knowledge and Data Engineering, IEEE Transactions on 11*, 1 (1999), 36–44.
11. Jin, J., and Szekely, P. Querymarvel: a visual query language for temporal patterns using comic strips. In *Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on*, IEEE (2009), 207–214.
12. Jin, J., and Szekely, P. Interactive querying of temporal data using a comic strip metaphor. In *Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on*, IEEE (2010), 163–170.
13. Karam, G. M. Visualization using timelines. In *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, ACM (1994), 125–137.
14. Kleene, S. C. Representation of events in nerve nets and finite automata. Tech. rep., DTIC Document, 1951.
15. Lam, H., Russell, D., Tang, D., and Munzner, T. Session viewer: Visual exploratory analysis of web session logs. In *Visual Analytics Science and Technology, 2007. VAST 2007. IEEE Symposium on* (Oct 2007), 147–154.
16. Lan, R., Lee, H., Fong, A., Monroe, M., Plaisant, C., and Shneiderman, B. Temporal search and replace: An interactive tool for the analysis of temporal event sequences. *HCIL, University of Maryland, College Park, Maryland, Tech. Rep. HCIL-2013-TBD* (2013).
17. Lee, B., Isenberg, P., Riche, N. H., and Carpendale, S. Beyond mouse and keyboard: Expanding design considerations for information visualization interactions. *Visualization and Computer Graphics, IEEE Transactions on 18*, 12 (2012), 2689–2698.
18. Monroe, M., Lan, R., Morales del Olmo, J., Shneiderman, B., Plaisant, C., and Millstein, J. 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*, ACM (2013), 2349–2358.
19. Monroe, M., Wongsuphasawat, K., Plaisant, C., Shneiderman, B., Millstein, J., and Gold, S. Exploring point and interval event patterns: Display methods and interactive visual query. Tech. rep., Citeseer, 2012.
20. Plaisant, C., Milash, B., Rose, A., Widoff, S., and Shneiderman, B. Lifelines: visualizing personal histories. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM (1996), 221–227.
21. Plaisant, C., Mushlin, R., Snyder, A., Li, J., Heller, D., and Shneiderman, B. Lifelines: using visualization to enhance navigation and analysis of patient records. In *Proceedings of the AMIA Symposium*, American Medical Informatics Association (1998), 76.
22. Riehmann, P., Hanfler, M., and Froehlich, B. Interactive sankey diagrams. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, IEEE (2005), 233–240.
23. Shneiderman, B. Direct manipulation: A step beyond programming languages. In *ACM SIGSOC Bulletin*, vol. 13, ACM (1981), 143.
24. Snodgrass, R. T., Ahn, I., Ariav, G., Batory, D. S., Clifford, J., Dyreson, C. E., Elmasri, R., Grandi, F., Jensen, C. S., Käfer, W., et al. Tsql2 language specification. *Sigmod Record 23*, 1 (1994), 65–86.
25. Srikant, R., and Agrawal, R. *Mining sequential patterns: Generalizations and performance improvements*. Springer, 1996.
26. Wang, T. D., Plaisant, C., Quinn, A. J., Stanchak, R., Murphy, S., and Shneiderman, B. Aligning temporal data by sentinel events: discovering patterns in electronic health records. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM (2008), 457–466.
27. Wongsuphasawat, K., Guerra Gómez, J. A., Plaisant, C., Wang, T. D., Taieb-Maimon, M., and Shneiderman, B. Lifeflow: visualizing an overview of event sequences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 1747–1756.