Modeling and Assessing User Interaction in Big Data Visualization Systems

No Author Given

No Institute Given

Abstract. When users interact with large data through a visualization system, its response time is crucial in keeping engagement and efficacy as high as possible, and latencies as low as 500ms can be detrimental to the correct execution of the analysis. This can be due to several causes: *(i)* for large data or high query rates database management systems (DBMS) may fail to meet the performance needs required to maintain response times below the desired threshold; *(ii)* modeling all the interactions with a visualization system is challenging due to their exploratory nature, where not all of them are equally demanding in terms of computation time; *(iii)* while there exists a variety of optimization techniques to handle this issue in specific scenarios, there is a lack of models for integrating them in a holistic way, hampering consistent evaluation across systems. In response to these problems, we propose a conceptual *interaction-driven framework* that enhances the visualization pipeline by adding a new *Translation* layer between the *Interaction-, Visualization- and Data-* layers, leveraging the modeling of interactions with augmented statecharts. This new layer has the objective of collecting information about queries and rendering computations, linking such values to interactions in the statechart. To make the Translation layer actionable, we contribute a software component to automatically model the user interaction for a generic web-based visualization system through augmented statecharts, in which each interaction is labeled with its latency threshold. We first demonstrate its generality on ten state-of-the-art visualization systems. Then we perform a user study (n=50) collecting traces by asking users to perform already established exploratory tasks on the well-known cross-filter interface. Finally, we replay those traces over its generated statechart, assessing the capability to correctly model the user interactions and describing violations in the latency thresholds.

Keywords: Visualization Application · Visualization Pipeline · User Interaction Modeling.

1 Introduction

When users interact with large data in a visualization system, its response time is crucial in keeping users quick and engaged, particularly for exploratory data analysis (EDA). Even latencies as low as 500 ms can be detrimental [36, 56]. This can be due to several causes: *(i)* for large data or high query rates database

management systems (DBMS) may fail to meet the performance required to maintain response times below the desired threshold; *(ii)* modeling all the interactions with a visualization system is challenging due to their exploratory nature, where not all of them are equally demanding in terms of computation time; *(iii)* while there exists a variety of optimization techniques to handle this issue in specific scenarios, there is a lack of models for integrating them in a holistic way, hampering consistent evaluation across systems.

A visualization system is usually modeled by *(i)* a data layer, *(ii)* an interaction layer, and *(iii)* a rendering layer. Latencies can be introduced in each of these three layers. For example, for large data and complex queries, even commercial database systems (DBMS) may fail to meet the performance needs of interactive visual analysis systems [11], introducing latencies in the data layer. In practice, most visualization systems are designed for small data or resort to ad-hoc mechanisms to try to meet the latency constraints on larger data (e.g., Spotfire [54]) by optimizing only one of the three layers. Also, although we have good models for optimizing the data and rendering layers (e.g., BIRCH [60], DEVise [37] etc.), we lack one which includes the interaction layer. Furthermore, no formal approach exists to connect known performance models in the visualization and HCI communities with optimization strategies at the data management level.

To provide a general model capable of capturing in an integrated way all these aspects, we propose a conceptual interaction-driven framework that enhances the classic visualization pipeline by introducing a *Translation layer* and by modeling interactions using augmented *statecharts* [31]. It allows the automatic collection of user interactions, their translation from low-level events into high-level user actions, and from them to database queries and rendering information that are all annotated in the statechart. The augmented statechart can be explored by the designers to understand which interactions suffer from excessive latency and how to fix them. Exploiting the proposed conceptual framework, it becomes possible to: *(i)* have a complete model of how users interact with the visualization system that can be annotated with information useful for optimization (e.g., latency constraints); *(ii)* know in advance the SQL queries that could be triggered from the current interaction state and *(iii)* derive through them the optimized computations needed to (re)render the results. To make actionable the framework, in this paper we contribute a software component for the *Translation layer* that automatically models the user interaction for a generic web-based visualization system. We first demonstrate its generality on ten state-of-the-art visualization systems. Then we perform a user study (n=50) collecting traces by asking users to perform already established exploratory tasks on the well-known cross-filter interface. Finally, we replay those traces over its generated statechart, assessing the capability to model the user interactions correctly and describing violations in the latency thresholds. In summary, we contribute:

1. a conceptual interaction-driven framework to manage latency of big data visualization systems in a holistic way, better modeling the user interac-

tions through the new *Translation layer*, and using them to allow flexible optimizations at the data and rendering levels;
2. the implementation of a module for automatic statechart generation (Statechart generator), a mandatory component of the *Translation layer*, allowing to automatically describe the interaction space of a generic visualization system and through user interaction collection identify latency violations and problematic parts of a visualization system;
3. The validation of the statechart generator through its application on a set of state-of-the-art visualization systems and a user study.

2 Motivating Scenario

Figure 1 shows an example of big data visualization system, crossfilter interface, for exploring airline delays (similar to Falcon [41]) using the Flights dataset, containing nearly 120 million tuples. Changing the selection of the flight distance, by brushing the histogram in the upper row, causes all other histograms to update in real time. Battle et al. proposed a benchmark [11] for evaluating how DBMSs support real-time interactive querying of large data and used the same crossfilter interface to collect performance data. They find that crossfilter interactions can generate hundreds of queries per second, with high latencies introduced while users expect near-immediate results. We use the *exploratory*

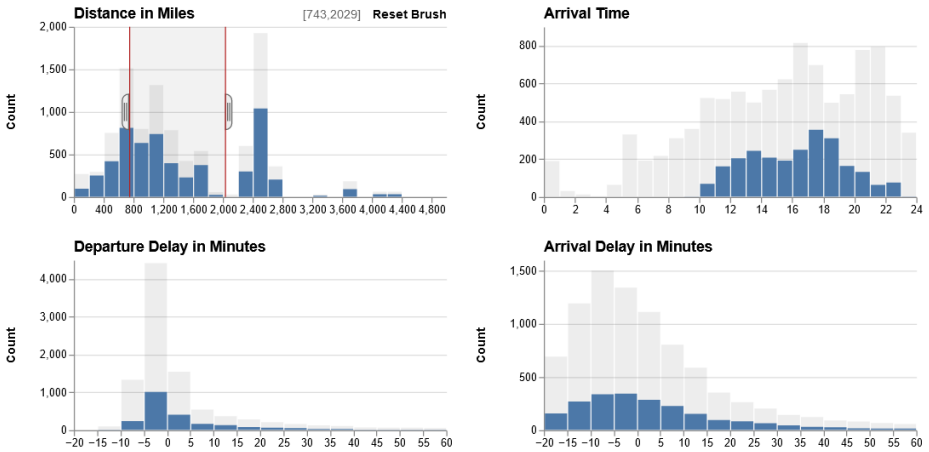


Fig. 1: *A user exploring flight performance data via a crossfilter interface. Brushing on one bar chart filters the others in real time.*

tasks from this benchmark to motivate our framework, which requires the user to explore all columns to respond to the following question: "Which factors appear to have the greatest impact on departure delays?". A user would likely brush several times on each histogram of the interface, to observe the effect on the departure delays histogram. Furthermore, this user may need more than one iteration to verify which column has the greatest impact, generating a huge number of queries.

While the user interacts with the crossfilter interface, not all interactions would suffer from the same latency. This will be due to (1) the type of interactor triggered (e.g., slider, button), (2) the underlying data query operations, and (3) the rendering time needed to update the visualization. Additionally, it will also depend on the current status of the analysis, e.g., the sequence of analysis steps triggered by the user through interactors that led to the current situation. This means that the same interaction step (e.g., the movement of a slider window on a histogram) could suffer or not from excessive latency depending on what happened before this action. For example, in scenario A where the user previously selected a big interval of flights to consider this would mean, when moving the slider, to fastly querying million to tens of millions of tuples, creating very high latency. In Scenario B, where the initial window size is very small(and much smaller than in scenario A), the same interaction would not be affected by any latency. These problems are not captured by classic benchmarks for databases, due to the stateless nature in which they are applied that does not take into consideration the user actions, nor the ones for interactive visualization systems, due to their high dependency on tasks to support and user interactions, as an obvious sequence of operations leading to the optimal result does not exist.

Our conceptual framework is then aimed at capturing in an automatic way these latency problems and providing them as input for optimization strategies at the database management system (DBMS) level and at the rendering level.

3 Related Work

As our proposal deals with modeling and assessing user interaction in big data visualization systems, we organized the related work into four main areas: existing models for big data visualization systems, models for user interactions collection and analysis, latency limits for effective data exploration, and finally data management.

3.1 Modeling Big Data Visualization

An interactive visualization system is designed following state-of-the-art guidelines for information visualization [17,52] and Visual Analytics [34,49]. It can be modeled with three main blocks: a data management block, a visualization (visual rendering) block, and an interaction block. Visualization systems managing small data implement each part in an ad-hoc way, storing everything in memory not worrying about optimizations. For big data visualization systems, data cannot be assumed to fit in memory. Several works exist that at different stages analyzed the literature to provide an overview of Big Data Visualization [1,35,40,48], but only a few coped with proposing frameworks for its modeling and management. Conner et al. [20] provide an analysis of how the visualization aspects of Big Data Visualization have been coped with from the born of the Information Visualization discipline to the present day, and how this term and related solutions evolved over time. Qin et al. [47] propose DeepEye, an automatic big data visualization framework for recommending the best suitable visualizations for the data at hand, a set of automatic analyses on these data coupled with

preservation of interactivity through database optimization techniques. While sharing with our proposal the goal of modeling the full pipeline, they do not have a specific focus on the user interaction as we propose, and they do not consider exploiting user interaction for optimizations. Similar considerations are valid for Erraissi et al. [24] and Golfarelli and Rizzi [29]. Both works try to model Big Data Visualization, with the former focused on proposing a meta-modeling for the visualization block and the latter augmenting it with automated visualization recommendations. Galletta et al. [28] coped with requirements coming from users in modeling Big Data Visualization. However, they target a specific domain, telemedicine, and put their focus on the interpretability and ease of use of the visualizations by physicians, not on capturing and modeling user interactions to support the identification of latency problems. Finally, a new branch of Visual Analytics, named *Progressive Visual Analytics* [9] or *Progressive Data Analysis and Visualization* [25], proposed models for managing latency through the continuous visualization of intermediate partial results computed on data samples or approximated versions of highly costly algorithms, capable of keeping the visual rendering and user interaction below latency constraints. Although providing sound solutions, none of the contributions in this field considers user interaction modeling as a driving factor to produce the intermediate partial results, limiting their intervention on the data, the algorithms, or the visualization rendering separately.

3.2 Modeling Interaction

Interactions are used for various tasks in visualization systems, modeled in a range that covers from low-level description [2] to high-level user intent [22, 55]. High-level tasks are abstract and their concrete implementation can take radically different forms. For example, selection can be done by clicking, using a lasso, or typing in a search box. A popular method for modeling interactions consists in using *widgets* that encapsulate a visual representation and interactive behavior (e.g., [5]). These widgets are connected to the main application through callbacks or other communication mechanisms. A major issue with widgets is exactly that they encapsulate both a graphical representation and a behavior, making it hard to modify one aspect independently of the other [15, 23, 32, 45]. To address this limitation and provide more structure to interactions, Meyers introduced Interactors [42], using state machines to specify and implement the interactions. While using Interactors allows for separating the visual appearance of interaction components from their behavior, the actions performed during interactions are open-ended and cannot be fully modeled in general. However, in visualization, the roles of interactions are more specific so modeling them is possible in most cases.

In addition to the work of Meyers [42], other works modeled interactions as state machines [26, 27, 33, 46]. For our framework, we rely on *statecharts* [31], which allow us to simultaneously run interactions while also reasoning about them. Statecharts are high-level specifications translated into a simple state machine through standardized semantics. Furthermore, they are mature, standardized, well-documented, and implemented by several libraries in multiple lan-

guages. Also, since their syntax is declarative, it can be easily extended.

A statechart is (conceptually) a simple state machine, equivalent to a directed graph with nodes denoting states, and directed edges labeled with a triggering event and pointing to a target state. Running the state machine consists in navigating its directed graph. It starts from a specified state A. When an event Y occurs in that state, it searches for an edge labeled with event Y having a guarding condition that is true. When it finds one, it then runs its transition action and moves to target state B which becomes the current state. Statecharts can also manage a *data model*, that can be tested and updated during transitions. We use this feature to manage the data model and internal states of interactions' statecharts.

3.3 Latency Limits

Big data exploration makes it hard to guarantee that latency will remain under well-specified limits. The *latency limit* is the maximum system response time before the user's cognition starts degrading. Latency limits are defined by studying threshold values of response time in various contexts. During the interaction, if the system does not show any result before the time limit, users could lose attention, be surprised, and feel that the system is unresponsive. While many latency thresholds have been proposed, it is still unclear which values should be considered for specific interactions, making it difficult to determine how and when to optimize interactions for big data exploration. The initial works of Miller [39], Shneiderman [51], and Nielsen [44] are not directly supported by empirical studies. Dabrowski and Munson [21] conducted an experiment with the aim of discovering new latency thresholds supported by empirical results but tied these limits to entire widgets (button, menu, dialog). Nah [43] conducted a State of The Art analysis by gathering information from Miller [39], Nielsen [44], and Shneiderman [51], concluding that delay of 2s for type 1 latency is not acceptable. Shneiderman et al. [53] introduce an additional limit of 3s for *"common tasks"* that is useful to determine the effect that waiting has on users. After 3 seconds, users feel like the system is slow, and start losing focus on their task. Waloszek and Kreichgauer [56] revisited Nielsen's thresholds by relaxing them into ranges, introducing the 3 seconds category from Shneiderman et al. [53] and extending the upper limit to 15 seconds. We will refer to the categories from this work as four latency levels, which can be seen in Table 1, of which we will mainly use *"level 0: 0.2s"* and *"level 1: 1s"*. Liu and Heer [36] show the impact that the 800 ms gap between level 0 and level 1 latency has on the user by providing an empirical study on macro interaction types, which they call *operations*: brush & link, select, pan & zoom. Zgraggen et al. [59] show the same impact but on the gap between level 1 and level 2 latency types through an empirical study that contained two exploratory interfaces, exploiting progressive data computation and visualization, with a simulated delay of 6 seconds and 12 seconds.

3.4 Data Management for visualization systems

When designing visualization systems, it is necessary to map each element of the target visualization component(s) to an operation on the underlying data,

Table 1: *Revised version, found in Waloszek and Kreichgauer [56], of the latency thresholds table introduced in Card et al. [18] ,with the addition of the 3 s category from Shneiderman et al. [53], and additional modifications.*

Level	***Latency Threshold***	***Description***
Level 0	0.1 (0–0.2) seconds	Perceptual Level: feedback after UI input involving direct manipulation/hand-eye coordination (e.g., mouse click, mouse movement, keypress).
Level 1	1.0 (0.2–2) seconds	Dialog Level: finishing simple tasks (e.g., opening a window or dialog box, closing a window, completing a simple search).
Level 2	3.0 (2–5) seconds	Cognitive Level: finishing common tasks, such as logging in to a system.
Level 3	10 (5–15) seconds	Cognitive Level: completing complex tasks, or a complex search or calculation.
Level 4	>15 seconds	

from simple filters over a data column to grouping and binning operations involving multiple columns. While those operations are supported by libraries such as D3.js [16] or Vega for small data scenarios, they become inefficient or even impossible to use in big data scenarios due to memory limits. When we couple the sheer amount of data processed by the DBMS with the fast pace of user interactions, the result is often slow response times [11].

Focusing on a concrete example, In the motivating example shown in section 2, the interface contains six histograms showing the distributions of six data columns. A filter interaction can be performed on any one of these histograms by dragging a range slider (i.e., a brush). Each histogram first filters the dataset with all the ranges specified by the brushes of the other histograms, and then the counts are updated to reflect the new bin heights. This means that every interaction will trigger six different aggregation queries, one to perform a COUNT operation on the filtered dataset, and five others to compute the bins for each histogram. In particular, by considering the exploratory task in section 2, we will have the following SQL Statement:

```
SELECT
   ROUND(FLOOR(departureDelays/5)*5,1) AS bin0,
   ROUND(FLOOR(departureDelays/5)*5+5,1) AS bin1,
   COUNT(*) AS height
FROM flights
WHERE arrivalTime BETWEEN 8 AND 18
GROUP BY 1
```

Query 1.1: *Query to compute bins for the departure delays histogram, having each bin uniformly distributed with size 5, with a filter applied to the arrival time column*

Changes in brush filters correspond to triggering six new queries that add or update the BETWEEN operators in the WHERE clause, which is difficult to support at larger dataset sizes [11].

Generalizing from the Crossfilter example, a visualization system exposes one or more visual components coupled with interactions. Each of them generates

a SQL fragment that is sent to the data layer. These fragments are combined by a data manager to issue the final database queries, but these queries can be optimized in multiple ways and queried at different times, either proactively with prefetching, reactively for real-time querying, and progressively when increasing the visual or interactive resolution over time. For example, ForeCache [10] loads data tiles proactively whereas Falcon [41] loads interaction results with increasing resolutions.

To improve the performance of big data exploration applications, many data management techniques have been proposed, which are classified into six categories by Battle and Scheidegger [12]: *Materialized Views*, *Approximate Query Processing*, *User Modeling & Query Prediction*, *Multi-Query Optimization & Shared Work*, *Lineage/Provenance*, and *Indexing* techniques. A proposal alongside the lines of our approach, while less general, is the DIEL system [58], which helps manage the mapping between visualization components and the underlying DBMS. It focuses on managing distributed data across different remote database engines so the visualization system can be back-end agnostic, and ensures asynchronous operations are done in the right order. It is aware of the columns used by the application so it does not force visualization components to refresh when an executed query has no influence on it, a simple but effective optimization. While each of these approaches can be integrated into our proposed framework, none of them copes with this problem from a general modeling perspective. Our framework provides the infrastructure to reason about and combine all the described interactions, queries, and modeling user behavior deriving optimization. For example, through user interaction modeling, our framework can extend optimizations with data prefetching and progressive refinement of the indexes driven by user actions.

4 Conceptual Interaction-Driven Framework

The proposed conceptual framework, visible in Fig. 2, is built on the classic visualization pipeline, modeled into three different layers: *(i)* a *data layer*, which collects the relevant data through selection and applies optional binning and aggregation operations; *(ii)* a *rendering layer* that uses the binned and aggregated data to generate visualization components and computes the extent of each rendered column to produce axes and scales and *(iii)* an *interaction layer* in charge of issuing queries either from dynamic query widgets or direct manipulation of the visualization components. Big data exploration is generally made possible through the cooperation of these layers, but each application is designed to address separate parts of the pipeline. At the same time, each of these layers, taken in isolation, can introduce problems. Concerning the data layer, for large data or high query rates DBMSs may fail to meet the performance needs required to maintain response times below the desired threshold, introducing latency and violating near real-time or real-time interaction (**issue 1**). Looking at the interaction layer, it usually simply captures the user interaction at a low level (e.g., events of a browser) and directly pass them to the data layer, contributing to creating high query rates even for simple interactions (e.g., brushing operation)

(**issue 2**). Additionally, modeling all the interactions with a visualization system is challenging due to their exploratory nature, where not all of them are equally demanding in terms of computation time (**issue 3**). Finally, while there exists a variety of optimization techniques to handle latency limits in specific scenarios (see section 3), there is a lack of models for integrating them in a holistic way, making it possible to identify causes of latency limits violations and intervene in a coordinated way at different stages of the visualization pipeline (**issue 4**). For example, we lack modules to systematically map interaction events at the rendering level into data processing operations at the database level. At the rendering layer, fully optimized query results that must be visualized by an aggregate visualization (e.g., a heatmap) could produce a new rendering that does not differ from the previous visualized state, wasting precious computation resources without providing any advantage to the user (**issue 5**). Without a connecting framework through the layers (that normally are organized as a stack, see the blue blocks in Fig. 2), it becomes difficult to understand how a new DBMS or rendering optimization strategy applies to new and existing visualization systems. As a result, optimizations developed in the database community run the risk of being point solutions at best, and irrelevant at worst in solving the performance issues observed by the visualization community.

One of the key ideas behind the proposed framework is the addition of a fourth layer, *Translation layer*, to integrate critical information coming from the other three and effectively connect them. It is managed separately from the visualization system design, providing benefits that the other layers of the system can not provide in isolation. It is composed of three components: *(i)* an augmented statechart (AS), *(ii)* an SQL translator (ST), and *(iii)* a rendering translator (RT). Through AS, the interaction between users and the system is modeled with a statechart in which nodes represent the set of possible contexts (i.e., the state of the visual component during the interaction, like the cursor position, HTML focus) and edges represent transitions between states caused by events triggered on the visual components. Furthermore, AS labels the edges of the statechart with desired latency thresholds. Then, ST communicates with the data layer to translate user interactions into the corresponding queries to be performed on the DBMS and adds this information to the labels in the statechart, allowing it to furtherly describe the relation between user interactions and DBMS queries. Finally, RT translates those queries into rendering computations by communicating with the visualization layer. As can be seen in Fig. 2, once the augmented statechart is built, the *Translation layer* can receive in input a high-level query fragment generated by the interaction layer, representing a user activity on the visualization system (e.g., the selection of a set of points in a scatterplot). Then, the fragment is used to fetch from AS an SQL-prepared statement, representing parametrized queries that should be instantiated and then performed on the DBMS when such an activity is performed. The statement is then sent to the data layer to get the actual SQL queries that are going to be triggered. Knowing those queries, the *Translation layer* can finally communicate with the visualization layer to pair them with rendering computations through RT. In this way,

our unified framework enables more versatile management of query load driven by the user's actions and provides a centralized layer that allows us to exploit, rather than modify, the existing data management, visualization, and interaction layers. this property makes it adaptable to any existing visualization system designed using the classic visualization pipeline. Additionally, it can be directly used to design a new system (effectively providing optimization capabilities) or it can be put on top of an existing one (working only as a descriptor/detector of what happens in the classicly implemented system).

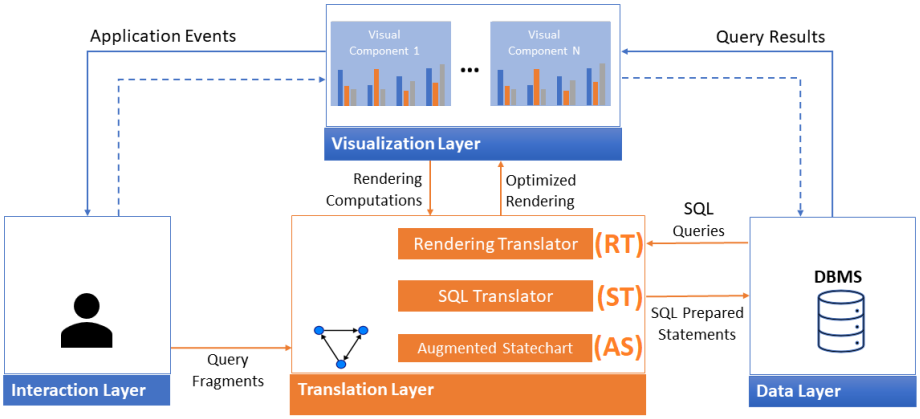


Fig. 2: *High-level description of how the conceptual interaction-driven framework enhances the classical visualization pipeline by introducing the* Translation layer.

The process carried out by the *Translation layer* is strictly bound only to the presence of AS, making its implementation mandatory to exploit the benefits of the framework. While ST and RT can be used to build additional layers of exploitable information on the edges of the statechart for implementing query or rendering optimization, AS can be used in isolation to model the interaction between the user and the system and link it to latency thresholds. We refer to this scenario as the *basic scenario*, capable of describing the user interaction space and identifying violations to latency constraints, both locally (i.e. subparts of the statechart) or globally (i.e., global efficacy score). This scenario provides mitigations for issues 3 and 4.

Moving a step forward, ST can be used not only for translating user interactions into queries but also to apply optimization on the DBMS based on the SQL-prepared statements. It is possible for example to count the frequency of similar or equal statements and prioritize them in the DBMS or cache their answers. Or it is possible to use them as inputs for classic DBMS optimization techniques, taking into account that they come directly from the usage of the visualization systems. This step allows an interaction-driven optimization on the data layer. We refer to this scenario as the *intermediate scenario*. This scenario additionally mitigates issues 1 and 2, inheriting the previous mitigations for issues 3 and 4.

Finally, the intermediate scenario would not consider the inefficiency that even an optimized query could produce on the visualization layer. To mitigate this problem, RT can provide optimizations taking into account the query result from ST and the information from AS. In this way, it could exploit the first to efficiently manage the rendering and the second to prioritize rendering in specific areas more used by the users or prone to stronger violations in cases in which multiple areas of the visualization part must be re-rendered. We refer to this scenario as the *advanced scenario*. This is the best scenario possible, in which also issue 5 got mitigated, providing solutions for all the reported issues. Overall, the resulting architecture is modular, allowing for a variety of possible implementations that exploits only a combination of the other components to better fit a specific context. Due to space constraints and for self-containment reasons, from this point on we will consider as the target scenario of this paper the *basic scenario*, leaving to future works the ST and RT components' advanced behaviors.

5 The Statechart Generator

The first step to make the *Translation layer* actionable is building a mechanism to automatically model the interaction with visualization systems implementing AS. With such a mechanism, the first part of the framework becomes easily applicable to a generic visualization system to validate its efficacy. AS was implemented through a software component, namely the *Statechart Generator*, developed following a three phases iterative approach: *(i)* requirements definition and state of the art analysis; *(ii)* prototyping; *(iii)* validation. In the following, the result of the final iteration of the approach is described, while source code, documentation, and detailed description of validation activities are available in an OSF project[1]. After a thorough investigation of the possible implementation solutions and a deep State-of-the-Art analysis, we ended up targeting visualization systems with the following characteristics: *(i)* designed for desktop devices; *(ii)* working properly on a web browser with Chromium command line interface (CLI); *(iii)* implemented through event-based techniques (the best results can be achieved with the JQuery or D3.js libraries).

5.1 Design

The final design of the software component completely revolved around capturing the interaction events, avoiding the case in which some events could be masked by high-level frameworks or libraries used. This goal has been achieved by instrumenting a module that, through the Chromium CLI, retrieves all non-masked interaction information for a specified document object model (DOM) object in the visualization system rendering code. Given an object of the DOM in input, it will output a list of the event listeners registered on it, alongside their description and properties (e.g., event functions). Thus, it can be exploited by calling it on each object of the starting DOM to get the contextless *Root Statechart*, modeling all the states reachable from the root, which we call *Rest State*

[1] `https://osf.io/79hsw/?view_only=be4e5107a18145e6b86a7eaf6109cb60`

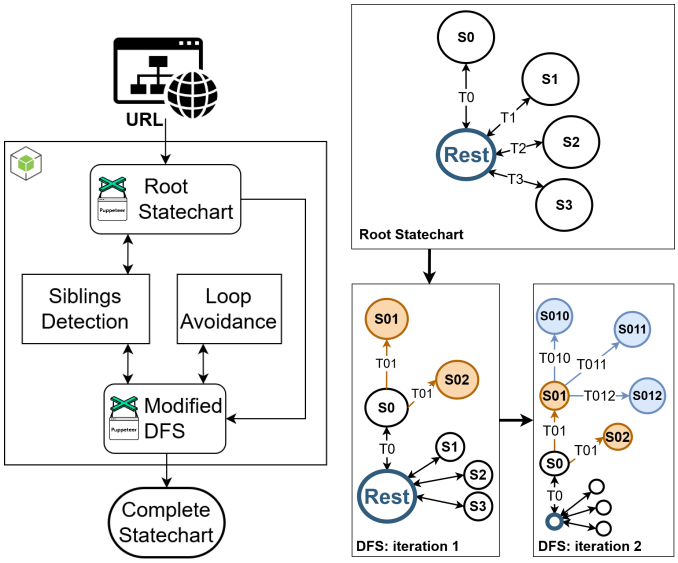


Fig. 3: *(a) Architecture of the Statechart Generator. (b) During execution, the Root Statechart is expanded by triggering events and looking for new states in a DFS fashion.*

(e.g., when the visualization system has just been started). Then, exploiting the *Puppeteer* Node.JS library (for the automatic simulation of human actions with a web application), a modified depth-first-search (DFS) exploration of the visualization system interactions can be started by automatically triggering each detected event on the web browser and checking which events can be triggered from the new context (e.g., cursor position), to reconstruct all the possible interaction paths in the application. To avoid infinite loops (e.g., selecting the same bar of a histogram over and over) the *getEventListeners* function can be invoked after triggering each event, and by comparing the new result with the last one it is possible to understand if new information has been discovered (e.g., if there are different numbers of listeners) or if the exploration reached a leaf and should backtrack to a new branch. Furthermore, to avoid the explosion of states that can be caused by specific implementations of some visualization components (e.g., a scatterplot with thousands of selectable dots) a mechanism to cluster together objects sharing the same interaction behavior and parents, which we call *Siblings*, has been implemented. This design drastically reduces the time required to build the statechart and makes it much easier to read for human users by showing only semantically different states. The architecture of the Statechart Generator can be seen in Fig. 3: *(i)* the URL of the visualization system is given in input to a NodeJS module; *(ii)* the Root Statechart is built exploiting Puppeteer and a siblings detection mechanism; *(iii)* the Root Statechart is used to perform a customized DFS that outputs the *Complete Statechart* exploiting

Table 2: *Latency thresholds assigned to each interaction.*

Interaction	*Transition*	*Latency Threshold*	*Source*	*Notes*
Zoom	*in/*out	Level 1	[56]	*=zoom
	*in/*out	74-106 ms	[30]	
Hover	*leave	Level 1	[56]	*=mouse
	*leave/*over	74-106 ms	[30]	
	*over	Level 0	[56]	
Drag/Pan/Brush	*start	Level 0	[56]	*=drag/pan/brush
	*end	Level 1	[56]	
	*end/*start/mousemove	74-106 ms	[30]	
	mousemove	Level 0	[56]	
Click	onclick	197.56 ms	[21]	
	onclick	Level 0	[56]	
	onclick	74-106 ms	[30]	

both the siblings detection and a loop avoidance mechanisms. Leveraging on the literature analysis (see Sec. 3.3) on latency limits, we propose a mapping between each atomic interaction and a latency threshold, which can be seen in Tab. 2. In this way, whenever a new transition is added to the statechart, it is automatically labeled with its mapped latency level.

6 Validation

In the following, it is first discussed how the contributed software component generality was validated. Then, a user study conducted on one of the most challenging systems to assess its capability of correctly modeling user interaction is described.

6.1 Generality

To validate the capability of the Statechart Generator to be executed on a generic web-based visualization system, a set of ten systems was taken from the InfoVis literature to be used as input to the software component. These ten systems were subjected to formal scrutiny, which took both intrinsic and extrinsic information in order to estimate the level of complexity of the candidate visualization system. This information was then used to make sure that the systems were chosen to spread equally across the complexity spectrum and cover the range of existing visualization systems. The established factors to assign a level of complexity to a visualization system can be seen in Tab. 4. The software ran on a virtual machine on the cloud with an eight-core i7 CPU, 32 Gb of RAM, an HDD, and Ubuntu 22.04 (Jammy Jellyfish). The ten chosen visualization systems with the associated level of complexity and time required to execute the Statechart Generator can be seen in Tab. 3 Finally, to assess a potential correlation between the time required to build the complete statechart and the factors that we used to assess their complexity level, we computed the Pearson correlation coefficient. From the result of this process, which is visible in Tab. 4, we can highlight a strong correlation between computation time and: *(i)* the number of states in the complete statechart; *(ii)* the number of events in the visualization system.

Table 3: *List of the ten visualization systems selected to validate the Statechart Generator, with details about their level of complexity and the execution time required to obtain the Complete Statechart.*

Visualization System	***Complexity***	***Time (minutes)***
DataVis [57]	568	30
Crumbs [8]	889	65
CrossWidget [5]	2532	100
Ivan [7]	2649	420
Nemesis [3]	5052	90
IDMVis [61]	5168	705
Wasp	5918	1140
Radviz [4]	6458	1860
InfluenceMap [50]	8581	420
Summit [19]	16427	30

Table 4: *Factors making up the complexity level of the ten candidate visualization systems alongside their Pearson correlation coefficient with respect to execution time.*

Intrinsic		***Extrinsic***	
Factor	***Pearson Coefficient***	***Factor***	***Pearson Coefficient***
#Elements	-0.205	#Siblings	-0.045
#Events	0.858	Execution Time	-
#Attributes and Data Fields	0.439	#Generated States	0.892
#Peculiar Events	-0.281	#Generated Edges	0.645
#Peculiar Contexts	0.636		

Summarizing, having not encountered any major problems in the generation of the statecharts for the ten representative visualization systems we conclude that the implemented Statechart generator is general enough to be applicable to a generic visualization system. We report results on the correctness of the generated statecharts in the next section.

6.2 User Study

To assess the capability of the Statechart Generator to correctly model the user interactions and describe violations in the latency thresholds, we performed a user study (n=50) to collect real user traces from the user interaction with a visualization system and replay them over the Complete Statechart to find potential discrepancies (e.g., missing states or transitions, illegal paths) highlighting, in the process, response times exceeding the latency thresholds. In the following, we discuss the formulated theses and how we designed the study.

Method The Complete Statechart needs to be able to cover the entire space of interaction between users and the visualization system to be trusted. Being automatically generated by exploiting a software component exploring the visualization system by triggering events in a deterministic way, the risk of not

being able to capture patterns of interaction typical of human behavior can not be neglected. Furthermore, labeling the transitions of the Complete Statechart with latency thresholds to easily highlight violations while replaying user traces on top of it requires validation with users. For these reasons, we structured the design of the user study around two main theses:

T1: the Complete Statechart generated by the Statechart Generator completely models the user interaction with the visualization system used.
T2: labeling the Complete Statechart with latency thresholds allows for correctly identifying violations when a user trace is replayed on it.

To accomplish *T1*, we decided to: *(i)* compute the Complete Statechart of a candidate web-based visualization system through the Statechart Generator; *(ii)* collect the traces of the user interaction with the chosen system and *(iii)* replay the collected traces over the Complete Statechart to check their compliance [13,38] (e.g., missing states, illegal paths). To accomplish *T2*, we decided to collect the system response time for each interaction while replaying user traces. In this way, we can easily compare such measures with the latency thresholds labeled on the Complete Statechart and highlight violations. The chosen candidate visualization system was the Crossfilter interface shown in Sec. 2, since, as already established, is the most challenging scenario and comes with exploratory tasks already defined and validated [11]. In particular, we chose to instrument it with the flight dataset sized at 7M entries, to make sure that the system could struggle while brushing, but without making the interaction unbearable, as it could happen with the 120M version. Since the main goal of the study was to collect user traces, we opted for a remote setup. This allowed us to reach a pool of n=50 participants in approximately one month. To implement the study we exploited *Stein* [6], a framework that makes it possible to execute task-based evaluations on visualization systems and to quickly extract relevant data types and events from the target system after embedding it by specifying its URL. To host the experiment online and collect the results, we used *PythonAnywhere*. The users were first asked to answer demographic questions, then they were asked to perform a simple tutorial; finally, they were asked to solve four exploratory tasks on the system. These tasks made the user interact with one or multiple visual components in order to explore data as requested. At the end of the experiment, the user traces were automatically uploaded to a remote repository. Then, those traces have been automatically replayed on the Complete Statechart using *Selenium*, that: *(i)* counted how many times each edge of the state chart has been traversed; *(ii)*recorded how much time was required to reproduce each interaction; *(iii)*labeled latency violations. Finally, the violations have been counted and classified, both in total and split for each exploratory task. In particular, we assigned to each violation a severity, that we call *Level Distance (LD)*, as follows: $\boldsymbol{LD} =$ *measured latency level - threshold latency level.*

Subjects Of the 50 participants, 27 were in the age range 18-24, 17 between 25-34, 3 in the range 35-44, and 3 ranging from 45 to 54. The majority of the pool consisted of males, with 34 participants, while 14 were females, 1 answered "Prefer not to say" 1 answered "Other". With respect to education, three had

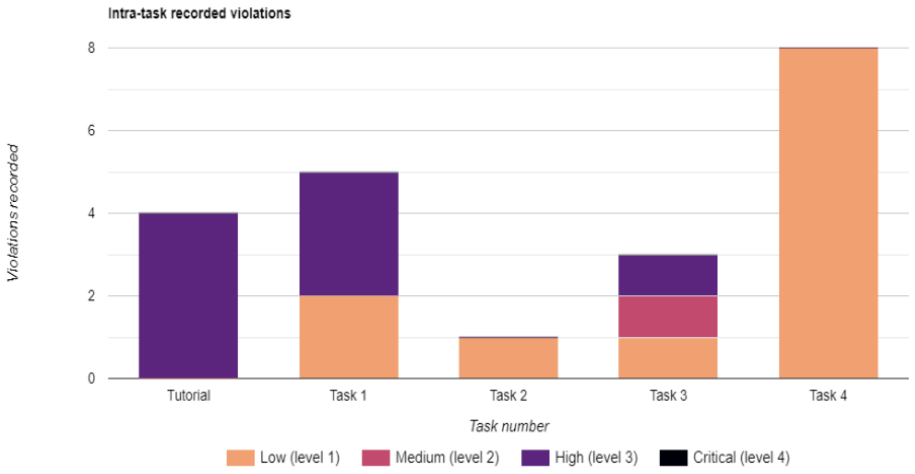


Fig. 4: *Stacked bar chart showing how many violations were highlighted by the user study, divided by task and by Level Distance.*

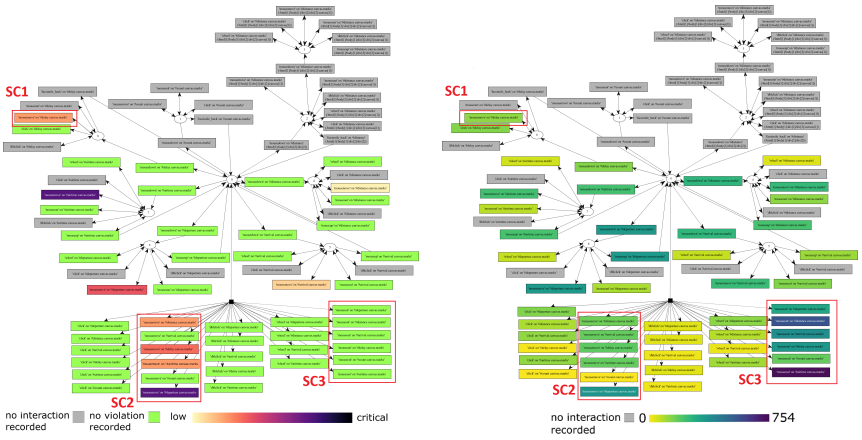
a Ph.D., 23 had a master degree, 14 a Bachelor, and 10 a high school degree. Knowledge about the IT field was distributed with 11 having advanced knowledge, 14 participants having good knowledge, 8 having intermediate one, 9 with low knowledge, and 8 with no knowledge at all in the field.

Results The first result that can be highlighted is that no missing states/-transitions or illegal paths were encountered while replaying 8481 interactions contained in the user traces on the Complete Statechart. This supports ***T1***. Joining this result with the generality testing, they allow us to conclude that the contributed Statechart Generator can model correctly the user interaction with web-based visualization systems.

Fig. 4 shows the violations in latency thresholds found while replaying the user traces, split by task and Level Distance. We highlight that task 4: *"How do distance, departure delays, and both distance and departure delays together appear to affect arrival delays"* introduced the majority of violations, while the tutorial introduced the most critical ones. This was expected, since task 4 was the most difficult one, making users explore data deeper than in the other tasks, while in the tutorial not having a clear goal made users interact with the system in a more free and highly variable way (e.g., rapidly brushing back and forth). All the recorded violations were only caused by two types of interaction: *mousemove* and *brush mousemove*. This is easily explainable by the nature of the Crossfilter interface, which exploits brush for selections and mouseover for inspection of precise values. As can be seen in Tab. 5, the *Departure Time* and *Airtime in Minutes* visual components were the ones that introduced the most violations, with 6 each, while the *Flights Selected* visual component did not introduce any violation, as it is the only non-brushable visual component of the entire system. Finally, Fig. 5 highlights which paths in the Complete Statechart produced more violations while replaying the user traces *(a)*, and how many

Table 5: *The violations found while replaying the user traces over the Complete Statechart, divided by triggering event and visual component.*

Visual component	***mousemove***	***brush mousemove***
Departure Time	4	2
Distance in Miles	1	1
Arrival Time	0	2
Airtime in Minutes	1	5
Arrival/Departure Delay	2	3
Flights Selected	0	0



(a) (b)

Fig. 5: *The Complete Statechart (rectangles are edge labels) of the Crossfilter interface chosen to perform the user study mapped with: (a) discovered latency violations; (b) paths frequency. Three scenarios are highlighted: SC1, in which medium violations were discovered in a path not so frequently taken by users; SC2, in which violations were discovered in a path frequently taken; SC3, in which no violations were discovered in a path frequently taken.*

times each interaction was performed *(b)*. Interestingly, the grey portion of the statechart identifies paths never taken by any user. We explained this result as a consequence of the task formulation, which asked to focus on visual components outside the grey area. This information can be helpful for optimization strategies that can prioritize the other areas of the statechart. In particular, three scenarios can be highlighted: *(SC1)* in which latency violations with medium LD were discovered in an interaction path not frequently taken by participants; *(SC2)* in which violations were discovered in a path frequently taken and *(SC3)* in which no violations were discovered in a path frequently taken.

This thorough analysis of latency violations clearly shows that by labeling the Complete Statechart with latency thresholds it is possible to easily highlight violations when replaying user traces, thus supporting ***T2***.

7 Discussion and Conclusions

In this paper, we presented a novel conceptual interaction-driven framework enhancing the classical visualization pipeline through the addition of the *Translation Layer*. This layer allows to model the user interaction space of a generic visualization, collects data about experienced latencies, and identifies latency violations for future optimization. We actioned the framework, in its basic scenario, through the implementation of the Statechart generator component, tested on a set of ten representative visualization systems, and with a user study (n=50) demonstrated the capability to correctly model the space of interaction between the user and a visualization system and to correctly identify and describe violations in latency threshold when a user trace is replayed on top of it. By using the proposed conceptual framework we identify as the first benefit the *easy modeling of the interaction* behind big data exploration interfaces, which does not depend on low-level implementation details. By exploiting the statechart description provided by the generator component it is possible to apply *programmatic inference of valid interaction paths*, which can enable big data exploration systems to conserve computational resources by ignoring interaction transitions that the user will never take, as shown for the Crossfilter interface used in the user study. It is also possible, by exploiting the annotated latency thresholds, to distinguish between latency-sensitive and latency-insensitive interactions or paths automatically. In this way, the visualization system can focus on deploying optimizations only for the interactions that truly need it. Moreover, the generated statechart can be used for statistical analysis of post-mortem event-based traces to compute the probability of alternative interaction paths, which could support predictive optimizations in real time.

Looking at limitations, while the statechart generator reached a *flexible support* for any web-based visualization design that can be specified using existing visualization languages such as D3 [16] (allowing the coverage of the majority of web-based visualization systems), we experienced from our testing activities some minor incompatibilities with HTML Canvas, moving objects (e.g., Origraph [14]), the native Javascript apparatus alert and Vega that will be further investigated in future activities. Additionally, currently the identification of violations can happen only during the current usage of a system by users, requiring the designer to recruit test users whose traces will be replayed to populate the statechart, as described in the user study. We are working on an automatic violations detector to allow the simulation of user behavior, informed by the traces collected during the user study, to relax this limitation. Finally, We intend to continue working on the remaining components, SQL translator and Rendering translator, to include optimizations that will allow the conceptual framework to express its full potential.

References

1. Ali, S.M., Gupta, N., Nayak, G.K., Lenka, R.K.: Big data visualization: Tools and challenges. In: 2016 2nd International Conference on

Contemporary Computing and Informatics (IC3I). pp. 656–660 (2016). https://doi.org/10.1109/IC3I.2016.7918044
2. Amar, R.A., Eagan, J., Stasko, J.T.: Low-Level Components of Analytic Activity in Information Visualization. pp. 111–117. IEEE (2005). https://doi.org/10.1109/INFVIS.2005.1532136
3. Angelini, M., Blasilli, G., Farina, L., Lenti, S., Santucci, G.: Nemesis (network medicine analysis): Towards visual exploration of network medicine data. (2019)
4. Angelini, M., Blasilli, G., Lenti, S., Palleschi, A., Santucci, G.: Towards enhancing radviz analysis and interpretation. In: 2019 IEEE Visualization Conference (VIS). pp. 226–230 (2019). https://doi.org/10.1109/VISUAL.2019.8933775
5. Angelini, M., Blasilli, G., Lenti, S., Palleschi, A., Santucci, G.: CrossWidgets: Enhancing Complex Data Selections through Modular Multi Attribute Selectors (2020), https://doi.org/10.1145/3399715.3399918
6. Angelini, M., Blasilli, G., Lenti, S., Santucci, G.: STEIN: Speeding up Evaluation Activities With a Seamless Testing Environment INtegrator. In: Johansson, J., Sadlo, F., Schreck, T. (eds.) EuroVis 2018 - Short Papers. The Eurographics Association (2018). https://doi.org/10.2312/eurovisshort.20181083
7. Angelini, M., Catarci, T., Santucci, G.: Ivan: An interactive herlofson's nomogram visualizer for local weather forecast. Computers **8**(3) (2019). https://doi.org/10.3390/computers8030053
8. Angelini, M., Lenti, S., Santucci, G.: Crumbs: A cyber security framework browser. In: 2017 IEEE Symposium on Visualization for Cyber Security (VizSec). pp. 1–8 (2017). https://doi.org/10.1109/VIZSEC.2017.8062194
9. Angelini, M., Santucci, G., Schumann, H., Schulz, H.J.: A review and characterization of progressive visual analytics. Informatics **5**(3) (2018). https://doi.org/10.3390/informatics5030031
10. Battle, L., Chang, R., Stonebraker, M.: Dynamic Prefetching of Data Tiles for Interactive Visualization. In: Proceedings of the 2016 International Conference on Management of Data. pp. 1363–1375. SIGMOD '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2882903.2882919
11. Battle, L., Eichmann, P., Angelini, M., Catarci, T., Santucci, G., Zheng, Y., Binnig, C., Fekete, J.D., Moritz, D.: Database Benchmarking for Supporting Real-Time Interactive Querying of Large Data. pp. 1571–1587. SIGMOD '20, New York, NY, USA (Jun 2020). https://doi.org/10.1145/3318464.3389732
12. Battle, L., Scheidegger, C.: A structured review of data management technology for interactive visualization and analysis. IEEE Trans. Vis. Comput. Graphics pp. 1–1 (2020). https://doi.org/10.1109/TVCG.2020.3028891
13. Benvenuti, D., Buda, E., Fraioli, F., Marrella, A., Catarci, T.: Detecting and explaining usability issues of consumer electronic products. In: Human-Computer Interaction – INTERACT 2021. pp. 298–319 (2021)
14. Bigelow, A., Nobre, C., Meyer, M., Lex, A.: Origraph: Interactive network wrangling. In: 2019 IEEE Conference on Visual Analytics Science and Technology (VAST). pp. 81–92. IEEE (2019)
15. Blanch, R., Beaudouin-Lafon, M.: Programming rich interactions using the hierarchical state machine toolkit. pp. 51–58. AVI '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1133265.1133275
16. Bostock, M., Ogievetsky, V., Heer, J.: D^3 data-driven documents. IEEE Trans. Vis. Comput. Graphics **17**(12), 2301–2309 (2011). https://doi.org/10.1109/TVCG.2011.185

17. Card, S., Mackinlay, J.: The structure of the information visualization design space. In: Proceedings of VIZ '97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium. pp. 92–99 (1997). https://doi.org/10.1109/INFVIS.1997.636792
18. Card, S.K., Robertson, G.G., Mackinlay, J.D.: The information visualizer, an information workspace. In: Proc. SIGCHI Conf. Human Factors Comp. Sys. pp. 181–186. CHI '91 (1991)
19. Class, B.: Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations
20. Conner, C., Samuel, J., Garvey, M., Samuel, Y., Kretinin, A.: Conceptual frameworks for big data visualization: Discussion of models, methods, and artificial intelligence for graphical representations of data. In: Handbook of Research for Big Data, pp. 197–234. Apple Academic Press (2021)
21. Dabrowski, J.R., Munson, E.V.: Is 100 milliseconds too fast? In: CHI'01 extended abstracts on Human factors in computing systems. pp. 317–318 (2001)
22. Desolda, G., Esposito, A., Lanzilotti, R., Costabile, M.F.: Detecting emotions through machine learning for automatic ux evaluation. In: Human-Computer Interaction – INTERACT 2021. pp. 270–279. Springer International Publishing, Cham (2021)
23. Edwards, M., Aspinall, D.: The synthesis of digital systems using asm design techniques. Computer Hardware Description Languages and their Applications pp. 55–64 (1983)
24. Erraissi, A., Mouad, B., Belangour, A.: A big data visualization layer meta-model proposition. In: 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO). pp. 1–5 (2019). https://doi.org/10.1109/ICMSAO.2019.8880276
25. Fekete, J.D., Fisher, D., Nandi, A., Sedlmair, M.: Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411). Dagstuhl Reports **8**(10), 1–40 (2019). https://doi.org/10.4230/DagRep.8.10.1, http://drops.dagstuhl.de/opus/volltexte/2019/10346
26. Ferrentino, A., AB, F.: State machines and their semantics in software engineering (1977)
27. Feyock, S.: Transition diagram-based cai/help systems. International Journal of Man-Machine Studies **9**(4), 399–413 (1977)
28. Galletta, A., Carnevale, L., Bramanti, A., Fazio, M.: An innovative methodology for big data visualization for telemedicine. IEEE Transactions on Industrial Informatics **15**(1), 490–497 (2019). https://doi.org/10.1109/TII.2018.2842234
29. Golfarelli, M., Rizzi, S.: A model-driven approach to automate data visualization in big data analytics. Information Visualization **19**(1), 24–47 (2020). https://doi.org/10.1177/1473871619858933, https://doi.org/10.1177/1473871619858933
30. Han, F., Xu, T., Tian, C., Hou, Z.: Investigation on human visual response latency. In: 2010 International Conference On Computer Design and Applications. vol. 1, pp. V1–602. IEEE (2010)
31. Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming **8**(3), 231–274 (1987). https://doi.org/https://doi.org/10.1016/0167-6423(87)90035-9, https://www.sciencedirect.com/science/article/pii/0167642387900359
32. Huot, S., Dumas, C., Dragicevic, P., Fekete, J.D., Hégron, G.: The MaggLite Post-WIMP Toolkit: Draw It, Connect It and Run It. In: Proc. ACM Symp.

on User Interface Soft. and Tech. pp. 257–266. UIST '04, ACM, New York, NY, USA (2004). https://doi.org/10.1145/1029632.1029677, https://doi.org/10.1145/1029632.1029677
33. Jacob, R.J.: Using formal specifications in the design of a human-computer interface. Communications of the ACM **26**(4), 259–264 (1983)
34. Keim, D., Andrienko, G., Fekete, J.D., Görg, C., Kohlhammer, J., Melançon, G.: Visual Analytics: Definition, Process, and Challenges, pp. 154–175. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70956-5$_7$, https://doi.org/10.1007/978-3-540-70956-5_7
35. Keim, D., Qu, H., Ma, K.L.: Big-data visualization. IEEE Computer Graphics and Applications **33**(4), 20–21 (2013). https://doi.org/10.1109/MCG.2013.54
36. Liu, Z., Heer, J.: The Effects of Interactive Latency on Exploratory Visual Analysis. IEEE Trans. Vis. Comput. Graphics **20**(12), 2122–2131 (Dec 2014). https://doi.org/10.1109/TVCG.2014.2346452
37. Livny, M., Ramakrishnan, R., Beyer, K., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J., Wenger, K.: Devise: Integrated querying and visual exploration of large datasets. p. 301–312. SIGMOD '97, Association for Computing Machinery, New York, NY, USA (1997). https://doi.org/10.1145/253260.253335, https://doi.org/10.1145/253260.253335
38. Marrella, A., Catarci, T.: Measuring the learnability of interactive systems using a petri net based approach. In: Proceedings of the 2018 Designing Interactive Systems Conference. p. 1309–1319. DIS '18 (2018). https://doi.org/10.1145/3196709.3196744
39. Miller, R.B.: Response time in man-computer conversational transactions. In: Proceedings of the December 9-11, 1968, fall joint computer conference, part I. pp. 267–277 (1968)
40. Mohammed, L.T., AlHabshy, A.A., ElDahshan, K.A.: Big data visualization: A survey. In: 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). pp. 1–12 (2022). https://doi.org/10.1109/HORA55278.2022.9799819
41. Moritz, D., Howe, B., Heer, J.: Falcon: Balancing Interactive Latency and Resolution Sensitivity for Scalable Linked Visualizations. CHI '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3290605.3300924
42. Myers, B.A.: Separating application code from toolkits: Eliminating the spaghetti of call-backs. In: Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology. pp. 211–220. UIST '91, ACM, New York, NY, USA (1991). https://doi.org/10.1145/120782.120805, https://doi.org/10.1145/120782.120805
43. Nah, F.F.H.: A study on tolerable waiting time: how long are web users willing to wait? Behaviour & Information Technology **23**(3), 153–163 (2004)
44. Nielsen, J.: Usability Engineering. Morgan Kaufmann (1993)
45. Oney, S., Myers, B., Brandt, J.: Interstate: A language and environment for expressing interface behavior. In: Proc. ACM Symp. on User Interface Soft. and Tech. pp. 263–272. UIST '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2642918.2647358, https://doi.org/10.1145/2642918.2647358
46. Parnas, D.L.: On the use of transition diagrams in the design of a user interface for an interactive computer system. In: Proceedings of the 1969 24th national conference. pp. 379–385 (1969)

47. Qin, X., Luo, Y., Tang, N., Li, G.: Deepeye: An automatic big data visualization framework. Big Data Mining and Analytics **1**(1), 75–82 (2018). https://doi.org/10.26599/BDMA.2018.9020007
48. Raghav, R.S., Pothula, S., Vengattaraman, T., Ponnurangam, D.: A survey of data visualization tools for analyzing large volume of data in big data platform. In: 2016 International Conference on Communication and Electronics Systems (ICCES). pp. 1–6 (2016). https://doi.org/10.1109/CESYS.2016.7889976
49. Sacha, D., Stoffel, A., Stoffel, F., Kwon, B.C., Ellis, G., Keim, D.A.: Knowledge generation model for visual analytics. IEEE Transactions on Visualization and Computer Graphics **20**(12), 1604–1613 (2014). https://doi.org/10.1109/TVCG.2014.2346481
50. Shin, M., Soen, A., Readshaw, B.T., Blackburn, S.M., Whitelaw, M., Xie, L.: Influence flowers of academic entities. In: 2019 IEEE conference on visual analytics science and technology (VAST). pp. 1–10. IEEE (2019)
51. Shneiderman, B.: Response time and display rate in human performance with computers. ACM Computing Surveys (CSUR) **16**(3), 265–285 (1984)
52. Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations. pp. 364–371. Interactive Technologies (2003). https://doi.org/https://doi.org/10.1016/B978-155860915-0/50046-9
53. Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., Diakopoulos, N.: Designing the User Interface: Strategies for Effective Human-Computer Interaction. Pearson, 6th edn. (2016)
54. Tico spotfire. `https://www.tibco.com/products/tibco-spotfire` (1995), accessed: 2019-07-20
55. Strahl, J., Peltonen, J., Floréen, P.: Directing and combining multiple queries for exploratory search by visual interactive intent modeling. In: Human-Computer Interaction – INTERACT 2021. pp. 514–535. Springer International Publishing, Cham (2021)
56. Waloszek, G., Kreichgauer, U.: User-centered evaluation of the responsiveness of applications. In: IFIP Conference on Human-Computer Interaction. pp. 239–242. Springer (2009)
57. Woodburn, L., Yang, Y., Marriott, K.: Interactive visualisation of hierarchical quantitative data: an evaluation. In: 2019 IEEE Visualization Conference (VIS). pp. 96–100. IEEE (2019)
58. Wu, Y., Chang, R., Wu, E., Hellerstein, J.M.: DIEL: transparent scaling for interactive visualization. CoRR **abs/1907.00062** (2019), `http://arxiv.org/abs/1907.00062`
59. Zgraggen, E., Galakatos, A., Crotty, A., Fekete, J.D., Kraska, T.: How Progressive Visualizations Affect Exploratory Analysis. IEEE Trans. Vis. Comput. Graphics **23**(8), 1977–1987 (Aug 2017). https://doi.org/10.1109/TVCG.2016.2607714
60. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: An efficient data clustering method for very large databases. p. 103–114. SIGMOD '96, Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/233269.233324, `https://doi.org/10.1145/233269.233324`
61. Zhang, Y., Chanana, K., Dunne, C.: Idmvis: Temporal event sequence visualization for type 1 diabetes treatment decision support. IEEE transactions on visualization and computer graphics **25**(1), 512–522 (2018)