Vistrates: A Component Model for Ubiquitous Analytics

Sriram Karthik Badam, Andreas Mathisen, Roman Rädle,
Clemens N. Klokmose, and Niklas Elmqvist, *Senior Member, IEEE*

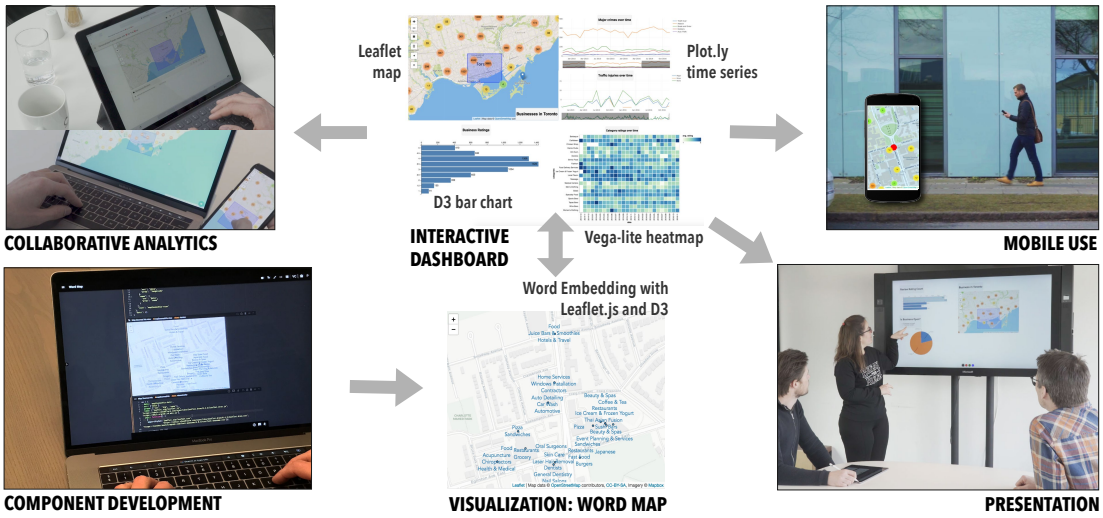

Fig. 1: Vistrates transcend the traditional tool boundaries of analysis activities, analyst expertise, input and output devices, and modes of collaboration to enable a truly ubiquitous visualization and analytics workflow. Each vistrate (center) is a shareable dynamic media [40] where components encapsulating existing web technologies—such as D3 [20], Vega-Lite [64], Leaflet, and Plot.ly—can be made to interoperate seamlessly. The document can be accessed from multiple settings, using heterogeneous devices, and by multiple concurrent users in activities ranging from data wrangling and exploration, to development and presentation.

Abstract—Visualization tools are often specialized for specific tasks, which turns the user's analytical workflow into a fragmented process performed across many tools. In this paper, we present a component model design for data visualization to promote modular designs of visualization tools that enhance their analytical scope. Rather than fragmenting tasks across tools, the component model supports unification, where components—the building blocks of this model—can be assembled to support a wide range of tasks. Furthermore, the model also provides additional key properties, such as support for collaboration, sharing across multiple devices, and adaptive usage depending on expertise, from creating visualizations using dropdown menus, through instantiating components, to actually modifying components or creating entirely new ones from scratch using JavaScript or Python source code. To realize our model, we introduce VISTRATES, a literate computing platform for developing, assembling, and sharing visualization components. From a visualization perspective, Vistrates features cross-cutting components for visual representations, interaction, collaboration, and device responsiveness maintained in a component repository. From a development perspective, Vistrates offers a collaborative programming environment where novices and experts alike can compose component pipelines for specific analytical activities. Finally, we present several Vistrates use cases that span the full range of the classic "anytime" and "anywhere" motto for ubiquitous analysis: from mobile and on-the-go usage, through office settings, to collaborative smart environments covering a variety of tasks and devices.

Index Terms—Components, literate computing, development, exploration, dissemination, collaboration, heterogeneous devices.

1 INTRODUCTION

Visualization, for all its success within academia, industry, and practice, is still very much a fragmented area with no common, unified method that applies in all (or even most) situations [10, 13]. For any given visualization project, the choice of tool, technique, and approach depends heavily on the dataset, the goals of the analysis, and the expertise of the analyst and audience. Many additional factors come into play: At what stage is the visualization going to be used: during initial analysis or presentation of results? Is the analyst alone, or is there a team consisting of multiple people, each with their own roles and expertise? Are there special devices or equipment, such as smartphones, tablets, display walls, or tabletops, that should be integrated?

All of these questions give rise to specific choices among the available tools and techniques in the visualization field today. For example, in terms of expertise, a novice may go for a template-based visualization tool such as Excel, a financial analyst may choose a shelf configuration tool such as Tableau [71], and a data scientist may opt for Jupyter Notebooks [55]. Early on in the sensemaking process, a developer may choose Observable [53] to interactively code their analyses and see immediate results, whereas a more mature project may call for a custom-designed web visualization built in D3 [20] or Vega [65], and a final report for communication may require a narrative visualization tool such as Graph Comics [6], Data Clips [3], or even Microsoft PowerPoint. Graph data may influence a designer to pick NodeXL [31] or

- *Sriram Karthik Badam and Niklas Elmqvist are with the University of Maryland in College Park, MD, USA. E-mail: {sbadam, elm}@umd.edu.*
- *Andreas Mathisen, Roman Rädle, and Clemens N. Klokmose are with the Aarhus University in Aarhus, Denmark. E-mail: am@cs.au.dk, roman.raedle@cc.au.dk, clemens@cavi.au.dk.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

Gephi [12], whereas tabular data may require Spotfire [1], and event sequences may mean using EventFlow [50] or EventPad [21]. Obviously, there are currently few synergies between these five determining criteria that we identify—expertise, analysis stage, data, single/multi-user, and device—and committing to one typically means disregarding the others. Furthermore, this fragmentation takes its toll on as participants need to make a "collective compromise," negotiate a common software denominator [51], and expend additional effort to share information, import and export artifacts, and work across visualization systems. The core issue is essentially one of *interoperability*: how to combine functionality from multiple available platforms?

In this paper, we apply the vision of *shareable dynamic media* [40] as well as recent advances in conceptualizing and implementing software as *information substrates* [16] to the field of data analysis. Information substrates blur the traditional distinction between *application* and *document*, they embody content, computation and interaction, and can evolve and be repurposed over time. We propose a *component model* for assembling visualization and analytics pipelines into such information substrates to increase their analytical scope. In this model, a component is a unit module with internal state, inputs, and outputs. Components provide visual analytics functionality and are reusable, replaceable, and extendable. This allows them to become building blocks for data analysis systems. Following the philosophy of information substrates, these systems can be integrated in media such as slideshows, interactive whiteboard canvases, reports, or interactive applications. Designing a comprehensive component framework is outside the scope of this paper, but we propose a basic library of components based on a few examples. While the component design space is huge, our model provides a starting point for future advances.

To demonstrate this model, we introduce VISTRATES (visualization substrates), a web-based collaborative platform for visualizations manifested as dynamic media (Figure 1). Vistrates provides a document-based framework of cross-cutting components for data visualization and analysis that **transcends** (i) both single-user and collaborative work, (ii) the full spectrum of data analysis activities, (iii) all levels of expertise, and (iv) a menagerie of devices. The platform provides a data layer for managing data, a pipeline model for controlling data flow, and a canvas for organizing visualization views. It is well suited to the type of "anywhere" and "anytime" sensemaking characterized by Elmqvist and Irani as *ubiquitous analytics* [9, 28, 76].

Components in Vistrates are maintained in a component repository as prototypes, from where they can be instantiated and composed based on the target analytical activity, or modified, or even refactored into new components. To this end, Vistrates follows a literate computing inspired approach,[1] by providing an integrated browser-based programming environment using the Codestrates framework [58]. While analysts and developers with programming expertise can instantiate, customize, or build components from scratch, Vistrates also supports a point-and-click interface for novices (or for use on mobile devices with limited screen size), where a pipeline can be assembled using a graphical user interface. Meanwhile, all vistrate documents are collaborative, and can be viewed on any device with a web browser.

In this paper, we contribute the Vistrates platform, which applies the concept of information substrates specifically to the activities in visualization and data analytics, while supporting multiple expertise levels, users, and devices. We also illustrate Vistrates and its capabilities using an in-depth motivating scenario on travel journalism in the next section and the accompanying video. Furthermore, we showcase features of the platform in additional examples on (1) wrapping existing toolkits such a D3, Vega, and Plot.ly as components, (2) offloading heavy computations from mobile devices, and (3) using the Vistrate canvas to present the results of an analysis to an audience.

2 MOTIVATING SCENARIO

Vergil is an experienced freelance travel writer. He has been commissioned by a new internet-based travel guide company called "TraLuver" that is trying to "disrupt" the travel guide industry by providing customized travel plans for their clients. Their business idea is to use data science to find an optimal match. TraLuver is rolling out their service to a select few North American cities, and Vergil has been tasked with curating and preparing the dataset for Toronto, Canada.

Prior to starting his field work, Vergil uses his laptop to familiarize himself with the TraLuver platform, which is built on top of a Vistrates installation. Vergil is not a data scientist, so he connects with Daria, an analyst in the data science team at TraLuver's headquarters. Using videoconferencing and a single vistrate document, Daria takes Vergil on a tour of the basic datasets available including Yelp! businesses and reviews and open data provided by the City of Toronto. She constructs a simple visualization interface, where a map of businesses in Toronto can be filtered to see their ratings in a bar chart, by putting together available components in the vistrate's graphical interface without any programming. Since Vergil knows he will be restricted to mobile devices when he is out in the field, he installs a mobile view following Daria's example, which shows one of the available views at a time.

After learning the system, Vergil heads out to the 553-meter CN Tower, a major landmark of the city. He installs a GPS component to center the map in the vistrate to his location. This helps him visit surrounding restaurants and access their reviews on the TraLuver platform. He realizes that the single map view would be more useful if it also incorporated relevant review keywords. He pulls out his tablet, sketches this idea, and discusses it with Daria, who provides a temporary solution by adding a simple word cloud. Daria gets in touch with Sam, a developer in TraLuver, who starts building the new component in a copy of Vergil's vistrate. After this, Sam calls Vergil and Daria and adds the new Word Map component to their vistrate.

A month later, after hard work by Vergil and his TraLuver team, Daria can finally present the finished Toronto project to the company's board. Basing her presentation on the same vistrate that she and Vergil created weeks back, she has created a slideshow of multiple canvases that show each feature, dataset, and visualization of the final product.

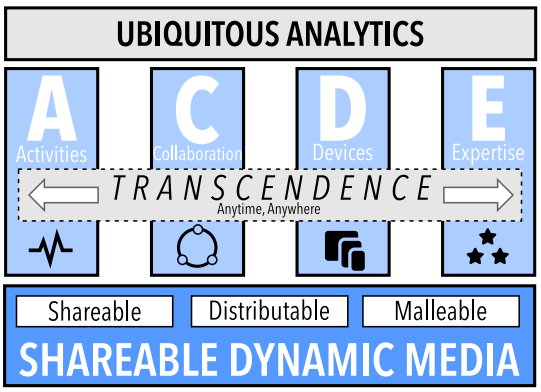

Fig. 2: Transcending individual silos of activities, collaboration, devices, and expertise to support the vision for truly ubiquitous analytics.

3 VISION FOR UBIQUITOUS ANALYTICS

The motivating scenario showcases a unique series of activities across multiple users with different expertise and using different devices. The users—Vergil, Sam, and Daria—seamlessly transition between analytical tasks including exploration of multiple datasets, presentation and sharing of ideas and insights, and development of new visualizations. They performed collaborative activities from distributed locations, with Vergil situated in the field as a mobile user. While doing so, they were even able to involve the expert developer—Sam—to meet an analytical need that arose during mobile use of the platform.

The common thread in the scenario is a notion that we coin *transcendence*. Data analysis in the scenario goes beyond individual contexts and applications to encompass a wide variety of analytical activities (A), modes of collaboration (C), types of devices (D), and levels of expertise (E). Transcendence across all four of these "ACDE axes" captures the essence of *ubiquitous analytics* [28], where people use

[1] Akin to interactive programming in Jupyter [55] and Observable [53].

multiple networked devices to analyze data anytime and anywhere. Central to realizing this vision is the ability to transcend each particular silo and consider the analysis as a whole (Figure 2).

However, achieving the vision cannot simply be to build a new tool that incorporates all four of the ACDE components [10]. Consider, for example, if Tableau added robust functionality in support of all four. In doing so, all we are left with is a yet another tool that while it has a wider purview than the others, still is a disconnected island separate from the rest of the data science ecosystem. We believe that instead of focusing on applications, the solution is to focus on *dynamic media* as the core building block of data analysis. Such "dynamic media" consist of information substrates that are *shareable* in that they intrinsically support collaboration, *distributable* in that they can be accessed using any device, and *malleable* in that they can be fully adapted by the user [40]. They transcend the classical application boundaries by allowing the user to mix and match tools and representations that typically are siloed in applications for particular domains.[2]

How would such a "dynamic media" approach to data visualization and analytics look in practice? First and foremost, such an approach would most likely be *based on the web* (C1), which at its core is the closest we have to dynamic media [40]. Second, to capitalize on the large and growing ecosystem of web-based visualization resources, the approach must be *open* (C2) and *extensible* (C3) to allow for integrating existing technologies into the platform without duplicating effort. Third, to enable recycling and magnifying prior efforts, components of the platform should also be *reusable* (C4) and *composable* (C5) so that new ideas can build on existing work. Finally, to truly transcend all contexts of activity, collaboration, device, and expertise, the tool should support *multiple, adaptive, and simultaneous layers of access* (C6), enabling teams consisting of the gamut of novices through programmers to analyze data together anytime and anywhere.

4 Background

Here we outline the background work in visualization platforms, collaborative visualization, and visualization across heterogeneous devices to define their scope in terms of our design considerations.

4.1 Visualization Toolkits, Platforms, and Languages

Data visualizations can be created with many toolkits and platforms, which mainly differ in terms of their target users. Visualization tools for novice users, such as Excel, support basic charting and data transformations. Shelf-based visualization tools such as Tableau [72] support easier configuration of visualizations by drag-and-drop into "shelves" of visual variables. For visualization design, iVoLVER [49] uses visual programming to help non-programmers extract data from multiple sources and transform it into animated visualizations. iVisDesigner [59] supports creation of visualizations by utilizing template-based mappings. More recently, Data Illustrator [46] and DataInk [79] explore the concept of lazy data bindings for better expressiveness in visualization design for non-programmers. Faceted visualization browsers such as Keshif [80] generate predefined facets representing data to support novice users focus on visual exploration.

For development, visualization toolkits such as Protovis [19] and D3 [20] support web visualization with a data model that maps data items to visual marks for SVG-based interactive graphics. D3 also binds the data to the Document Object Model (DOM) of web browsers and supports basic extract, transform, load (ETL) operations to create data objects for custom visualizations. More recently, high-level visualization specification grammars have been developed, such as Vega [63], Vega-lite [64], and Atom [54]. These platforms are oriented towards analysts with technical expertise in visualization development. They offer preliminary means for reusability by maintaining online examples and promoting open source contributions [17].

4.2 Supporting Collaboration

Previous collaborative visualization platforms have covered all quadrants of the collaboration matrix [11, 36]. The asynchronous collaboration model is most common for visual analytics, exemplified by the Sense.us [34] and ManyEyes [75] platforms that leverage crowd intelligence through view sharing, annotation, discussion, and social navigation [27]. Methods for maintaining group awareness [29]—understanding of collaborators' work—and coordination [47, 73] have been considered through history mechanisms and notifications [32] as well as data coverage representations and widgets [38, 62, 77].

Specialized techniques for supporting collaborative visualization have also been explored. Collaborative brushing [30, 37] helps build upon the group's interactions through highlighting of the user selections. Branch-explore-merge [48] presents a co-located group analytic technique for transition between coupled and decoupled work across devices. For co-located collaboration, Lark [74] allows multiple users to work together by directly manipulating the visualization pipelines. Frameworks have also been created for developing distributed visual analytic interfaces, such as Munin [9] and PolyChrome [7]. These efforts support requirements for networking devices and connecting multiple users in specific scenarios, but do not offer concrete means for transcendence on the ACDE axes described in Section 3.

4.3 Heterogeneous Devices for Visual Analysis

Another significant effort in visualization has been to utilize heterogeneous devices [43]. SketchStory [44] uses touch input and sketching as a means for storytelling. VisPorter [23] enables cross-device interactions for visual exploration between large displays and smartphones. GraSp [39] promotes flexible workflows involving physical navigation and remote interaction between handheld devices and large displays. VisTiles [41] presents the styles in coupling multiple small-screen devices to explore data. Recently, a conceptual framework for combining multiple devices—large displays and smartwatches—for visual analysis has also been developed [35]. These advances exemplify the fact that visual analysis has outgrown traditional desktop scenarios [60] to support advanced applications involving heterogeneous devices.

4.4 Interactive Notebooks

Interactive notebooks have recently gained popularity in data science and visual analytics, as they promote collaboration by sharing. They adopt a literate computing-based approach to programming, where executable code is interweaved with text and images to create interactive narratives. Jupyter Notebook [55] is a web-based interactive notebook that connects to a kernel capable of executing code in languages such as Python, R, Ruby, JavaScript and many more. Jupyter provides integrations with analytics and visualization frameworks such as SciPy [52] for analytics and Altair [2] for declarative visualizations. However, Jupyter does not support real-time collaboration out of the box. Google's Colaboratory [24] is a Jupyter implementation using the Google Drive backend for real-time collaboration. However, collaboration is on the level of editing and not interaction.

Observable [53] is a JavaScript-based interactive notebook primarily designed for creating interactive visualizations. It provides a reactive programming model where re-execution of a code-block will result in a re-execution of any code block that depends on it. While notebooks written in Observable are easy to fork and share, they do not yet support real-time collaboration neither in editing nor interacting with the produced visualizations. Furthermore, they are not focused on analytical work across heterogeneous devices and activities in the sensemaking beyond visualization development.

4.5 Webstrates and Codestrates

Vistrates is built on top of Webstrates and Codestrates. Webstrates [40] is a web framework where webpages are made collaboratively editable in real-time. Changes made to the DOM of a webpage (called a *webstrate*) are persistent and synchronized to all clients of the webstrate.

[2] This is consistent with the principles of *instrumental interaction* [14, 15] from HCI, where instruments transcend individual applications and instead take a document-centric view of computing as documents manipulated using general instruments. In instrumental interaction, you do not open Microsoft Word to edit a document; instead you simply use the text editing instrument to modify the document object, and you can use the same instrument to edit a webpage or a tweet—activities that would normally require separate applications.

Table 1: Classification of existing tools and platforms for supporting data analysis based on the ACDE axes from Figure 2. A checkmark (✓) signifies that the corresponding tool/system/framework has dedicated support for the dimension. Note that Codestrates is the only framework here not developed for data analysis, but it is added here since Vistrates is built on top of it. (ETL implies Extract, Transform, Load.)

Systems/Frameworks	**Collaboration**		Cross-**Expertise**	Responsive to **Devices**			**Activities**			
	Sync	Async		Input	Output	Resources	ETL	Analysis	Presentation	Sharing
Jupyter [55]							✓	✓		✓
Observable		✓					✓	✓		✓
Colaboratory	✓	✓					✓	✓		✓
Tableau			✓					✓	✓	✓
D3 [20]							✓	✓		
VisPorter [23]	✓			✓	✓			✓		
ManyEyes [75]		✓	✓					✓		
PolyChrome [7]	✓	✓		✓	✓			✓		
Codestrates [58]	✓	✓		✓	✓				✓	✓
Vistrates	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Codestrates [58] is an authoring environment built on top of Webstrates. A codestrate is a webstrate that includes tools for editing its own content, including writing and executing code, following a literate computing approach similar to interactive notebooks. Individual codestrates contain their implementation, which means they can be reprogrammed from within. A codestrate is structured in sections consisting of paragraphs of code, data, styles, and web content. Sections can be turned into packages of functionality that can be shared between codestrates [18]. Similar to Observable, Codestrates is JavaScript-based and all execution happens in the run-time of the browser. This also means that it integrates well with existing web libraries.

4.6 Summary of Related Work

Table 1 contrasts the related work against our Vistrates platform. Overall, Vistrates is inspired and influenced by many related works. It is capable of leveraging frameworks such as D3, Vega, and Plot.ly [57], while promoting reusability and extensibility in development. It uses Codestrates and Webstrates as the underlying infrastructure specifically to (1) support interactive programming for development/extension of vistrate components, (2) synchronize documents across devices and users, and (3) utilize package management to maintain reusable components. By doing so, it extends the related work to ubiquitous visual analytic scenarios by defining a visualization component structure and enabling many activities in the sensemaking spectrum [56].

5 Designing Vistrates

Vistrates is a design proposal for a component model for ubiquitous analytics, but also a proof-of-concept implementation. It is a realization of a set of design choices that together form the vision of a holistic and sustainable data analysis and visualization environment.

The design of Vistrates is rooted in the principles from Webstrates that software should be malleable, shareable, and distributable. With Vistrates we assume that all software is expressed as (one or more) webstrates, and hereby we inherit the quality of Webstrates that shareability is a fundamental premise; webstrates can be edited in real-time by remote users, and easily shared by passing around a URL. As webstrates are webpages, they can be accessed from any device with a browser, which means that distributability is ensured.

To realize a visualization environment that can enable the motivating scenario and adhere to the ideals for ubiquitous analytics, we introduce the following six design principles to support analytics.

5.1 Component-based Pipeline Architecture

The typical architecture to go from data to visualization is through a visualization pipeline [22]. We propose a component-based architecture, where components (Figure 3(a)) are connected together in reconfigurable pipelines (Figure 3(b)). A component can be a data source (e.g., serving a file, connecting to a database or API, or providing coordinates from a phone's GPS module), a computation on data (e.g., filtering, aggregating, or analyzing), or a visualization (e.g., a bar chart, scatterplot, heatmap, etc.). Visualizations do not have to be end points in the pipeline, but can be interactive and hereby serve as data sources as well. Components should be executable blocks of code with an optional input, output, state, and view (Figure 3(d)). The pipeline should be reactive, so when the output of a source component changes, it will trigger updates of components that have the output of the source component as input. Components should adhere to a minimalistic interface for connecting them together, and what a component does and what third-party software libraries it uses should be up to the developer.

5.2 Collaborative Pipeline

It should be possible to modify, run, and interact with components in the pipeline *collaboratively* from different clients. However, for most computations it is faster and simpler to execute them locally than to distribute data across a potentially high-latency network. We therefore propose a design where each client executes its own instance of a pipeline, but synchronizes state locally to components between them. State includes the configuration of a component and any application state that should be synchronized or persisted, e.g., interactions. An example of the latter could be a rectangular selection on a map-based visualization, or a URL to a data file that a data source component should load into memory. It should be up to the developer to specify what application state is synchronized, allowing, e.g., the developer of the aforementioned map component to specify that selections should be synchronized but not, e.g., zoom levels. Components should synchronize execution between clients, i.e., rerunning a component on one client should trigger reruns on all other clients as well.

In other words, the collaborative pipeline principle consists of (1) a reactive data flow, (2) a shared execution flow, and (3) shared component state between clients of the same vistrate.

5.3 Prototype-based Components

Components should be prototype-based, and components should be instantiated by copying from a prototype and configuring the instance into the pipeline. An instance of a component should contain its own implementation and its state. This is a deliberate violation of the software architectural principle to avoid code duplication. However, by having components contain their own code they become directly reprogrammable, allowing a user to reprogram a single component and potentially turn it into a prototype for new components.

5.4 Multiple Levels of Abstraction

Users should be able to work on multiple levels of abstraction: from programming components, to configuring components in a pipeline, to creating presentations of the visualizations and to interact with said visualizations. At the lowest level of abstraction, all aspects of components should be manipulable as code. At a higher level of abstraction, components and their pipeline should be reconfigurable in an interactive fashion, allowing for even non-programmers to reconfigure without programming (Figure 3(b)). At an even higher level of abstraction,

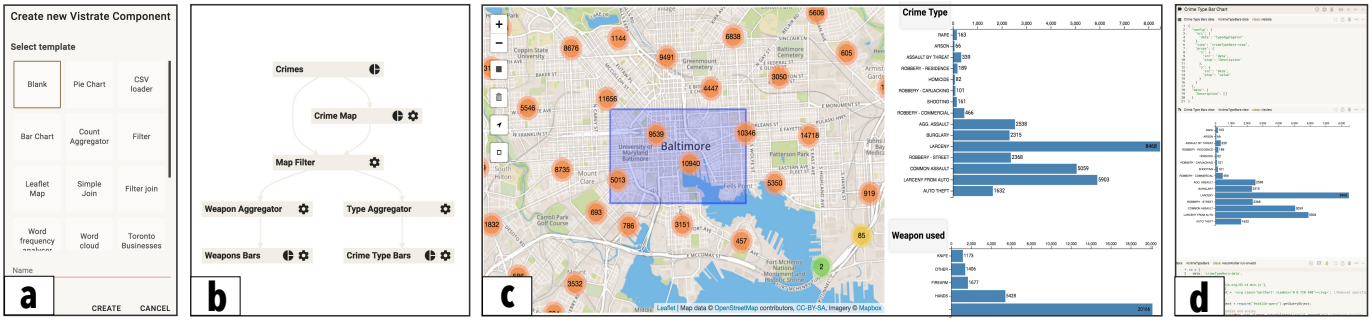

Fig. 3: (From left to right) **Component instantiation**: Existing component prototypes are available to be edited, thus, promoting reusability and extensibility. **Pipeline view:** This view supports configuration through interaction—drop-down selection—to create visualizations and interactively explore data without programming. In this example, a crime dataset from Baltimore, MD is visualized through a map and bar charts for crime type and weapons by aggregation. A filter component is added to filter the bar charts based on the selection on the map. **Dashboard view**: This vistrate view creates a grid layout for visual exploration of the data and annotation using rich text. **Development view**: The lowest level of abstraction for a vistrate in which a programmer can edit the code and create new visual analytic components.

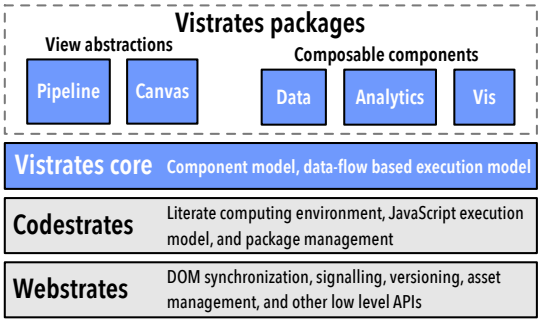

Fig. 4: Vistrates architecture and relation to Codestrates/Webstrates.

visualizations should be treated as content that can be composed, e.g., in the form of a slide deck, a document, or a dashboard (Figure 3(c)).

Collaboration should be possible on each level of abstraction—from writing the code to interacting with the visualizations—but it should also be possible to collaborate on different levels of abstraction *at the same time*. That is, while one user is interacting with a component, it should in principle be possible for another user to reprogram it and re-execute it without requiring the first user to restart their client.

5.5 Component Repository

It should be possible for a "programming literate" user to develop their own components or redevelop other people's components. However, we also wish for a non-programming-savvy user to be able to construct a visualization pipeline using components made by others. Components should, therefore, be shareable through a common component repository where users can publish their components, as well as retrieve components made by others.

5.6 Transcending Application Boundaries

It should be possible to integrate visualizations directly into other types of media (e.g., presentations or reports). The components and pipeline should therefore co-exist in an open-ended software ecology with tools not only designed for analytics and visualization work.

6 Implementation

Vistrates[3] is implemented using standard modern web technologies—JavaScript, HTML, and CSS—as well as Codestrates [58] and Webstrates [40]. It uses Codestrates' literate computing and package management features [18]. Vistrates consists of a core framework package and individual components implemented as packages (Figure 4).

[3]Vistrates: `https://github.com/karthikbadam/Vistrates`

In Codestrates, software is implemented in paragraphs grouped into sections. There are four basic types of paragraphs: *code paragraphs* containing JavaScript that can manually be executed by the user, toggled to run on page load, or be imported into other code paragraphs; *style paragraphs* containing CSS rules; *body paragraphs* that can contain any web content expressible as a DOM subtree; and *data paragraphs* containing JSON formatted data that can manually be edited by users, or programmatically through JavaScript. Every paragraph can be given a human readable name, a unique identifier, and a list of class names. Every paragraph can be toggled to be shown in full-screen either only local to a particular client, or for all clients. Vistrates utilizes these abstractions—paragraphs and sections—and defines an update logic and data flow between them, turning them into building blocks for visualization components and analytical activities.

6.1 The Core Framework

At its core, the Vistrates framework governs the control flow through component pipelines using the principles of inversion of control and dependency injection of components. The backbone of Vistrates is a singleton that registers all components in a vistrate, a component class that implements an observer pattern for connecting the input and output of components together, and an execution model for executing user-provided component code.

On load, the Vistrates singleton registers all existing components and registers observers between them. When components are updated or new components are created, the singleton also updates observers accordingly. All components have a controller that implements an observer pattern, such that the appropriate components in the pipeline are notified when the output of a component changes. A component consist of three paragraphs grouped in a section: a code paragraph, a data paragraph, and an optional view paragraph (web content paragraph).

The **code paragraph** of a component includes the definition of specific methods and properties following the format shown in Listing 1, which include the fields `data`, `src`, `props`, and `libs` as well as the methods `init`, `destroy`, and `update`. All fields and methods in a component are optional, but defines how a certain component can function within the pipeline. Vistrates uses regular DOM IDs as references, and the `data` property contains the ID of the component's data paragraph containing configuration and stored state. The `src` property defines named sources that can be referred in its methods, and `props` refers to named properties of the sources that can be remapped dynamically through its configuration data. If a component does not have source references, it can only function as an entry node in the pipeline, which is typical for components that load data into the vistrate. `libs` is a list of references to JavaScript libraries that the component depends on. The references can either be URLs to external files, or file names of files uploaded as assets to the webstrate.

The first time the code paragraph is executed, a controller object is

```
vc = {
  data: "id-of-vis-data",
  src: ["mySourceName_1", ..., "mySourceName_n"],
  props: ["myProp_1", ..., "myProp_m"],
  libs: ["myLibraryStoredAsAsset.js", "https://somecdn.com/anotherLibrary.js"],
  init: function() { /* code goes here */ },
  destroy: function() { /* code goes here */ },
  update: function(source) { /* code goes here */ }
}
```

Listing 1: The code paragraph template.

```
{
config: {
    src: {"mySourceName_1": "source_1_id", ..., "mySourceName_n": "source_n_id"},
    props: {
      "myProp_1": { "src": "mySourceName_1", "prop": "somePropOnSource"},
      ...,
      "myProp_m": { "src": "mySourceName_n", "prop": "someOtherPropOnSource"}
    },
    view: "id-of-vis-view"
  },
  data: { /* The data field of the component for storing state */ }
}
```

Listing 2: The data paragraph template.

instantiated from the controller class, and the properties and methods defined in the code paragraph are evaluated and copied to the controller object. If the controller references anything in `libs`, these are downloaded and evaluated before the `init` method is run. After `init` the `update` method is called, and it is subsequently called any time any of the component's sources update their output. Code paragraphs can be rerun by pressing the play button, and whenever this happens, the previous `destroy` method is executed (to, e.g., remove event listeners) and the newly defined properties and methods are hotswapped on the controller object. Updating the `output` property of a component will trigger the `update` method on any observing components.

The **data paragraph** contains the configuration of the component and the shared state, encoded as JSON. The data paragraph template has the format shown in Listing 2. Besides observing the sources of a component, a Vistrates controller also observes its data paragraph and changes to the configuration will trigger changing dependencies to be hotswapped and changes to the state will trigger the `update` function, and thereby immediately be reflected in the component views on all clients. The chosen configuration of a component includes the mapping between source/property reference names and the actual ids in the vistrate, which allows users to change the mapping on the fly. This format was chosen to be able to reference specific data items in the output of a source component without forcing developers to follow a rigid output convention. As an example, the source with reference name `mySourceName_1` that is currently mapped to source id `source_1_id` in Listing 2 can be changed to refer to another source id simply by changing this mapping. Similarly, the property `myProp_1` is mapped to a specific data item in `mySourceName_1`. The configuration also includes a reference to the view paragraph. The shared state of a component can be encoded in the `data` field of the data paragraph, which for instance can be the interaction state of a visualization. Webstrates will by default synchronize the DOM elements outside `transient` HTML tags, hence the state encoding will also be synchronized across clients. We have chosen an open format for the declarative state specification, which means that it is left to the developer to define this encoding and how to behave accordingly in the `update` method. The data paragraph can be edited by the user, or updated programmatically, e.g., by the pipeline view described below.

The **view paragraph** is a body paragraph containing the visual output of a component. The content can be any standard web content, which is then wrapped in a `transient` element. Transient elements are Webstrates-specific DOM elements that do not have their state synchronized nor persisted. This means that the content of views are not shared across clients. Clients share code and data paragraphs, but clients are executing their own pipeline and thereby creating the content of their own views. This makes it possible to have views where not all interactions are shared between clients. As an example, a map component can share area selections by writing those selections to the data paragraph, while at the same time allowing each user to define their own viewbox and zoom level. In the controller code, the view

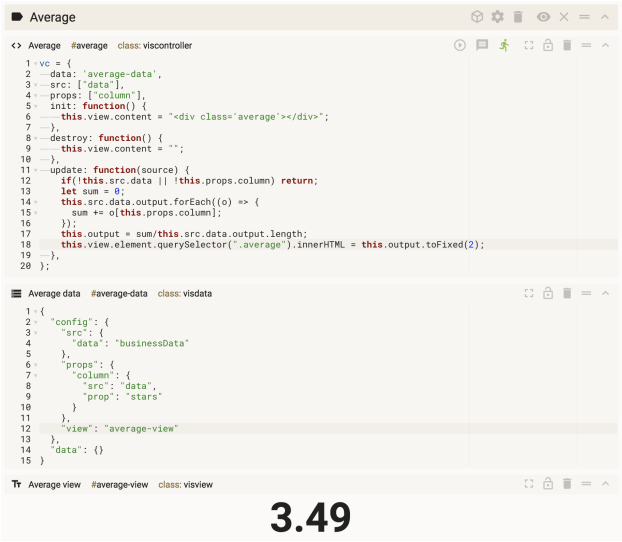

Fig. 5: A simple Vistrates component that calculates the average of a single numeric data column. The component is easily reconfigurable by changing the mappings in the data paragraph. Other components can observe the output and act accordingly when it changes.

can be referred through the `view` property and its content can be replaced by setting the `view.content` property either to an HTML string or a DOM node reference, or by referring to the root DOM element of the view using the `view.element` property. Finally, style paragraphs can be added to define the appearance of a vistrate view.

Component updates in Vistrates are triggered in two ways: (1) when the output of a source is updated, or (2) when the configuration or the state in the data paragraph is updated. The cause of an update is encoded in the first argument in the `update` method, which can be a specific source from the `src` list, the configuration, or the state. We chose this design as it allows the developer to update a visualization differently based on the type of update; if the data input changes the visualization needs to be redrawn, but if only the interaction state changes the visualization can be updated in a different manner: say, by highlighting specific visual marks. It is possible to create update cycles between components, but it is up to the developer to ensure that these cycles are finite. For instance, such update cycles are currently used to develop coordinated multiple views with brushing-and-linking [61].

In essence, Vistrates adapts these paragraphs to visualization and analytics. In contrast to Codestrates, paragraph definitions in Vistrates have an analytical value—the code captures the underlying logic for processing and visualizing data, the data paragraph captures the declarative specification to map properties to visual variables, and the view contains the analytical outcome of the component. Furthermore, Vistrates explicitly defines the update logic and control flow across components made from these paragraphs. Figure 5 shows an example component including controller (code), data, and view paragraphs that calculates the average of a data column and views the result.

6.2 The Pipeline

Vistrates components are composable through the configuration specification in the data paragraph. The pipeline view is an abstraction layer on top of the textual specification, which provides graphical access to the configuration and composition of the components in a vistrate. In the pipeline view, components can be reconfigured and recomposed at any time, and changes are immediately reflected in their output, which also triggers updates of connected components. The components' views can be inspected within the pipeline view to immediately observe the effects of a reconfiguration or recomposition. The pipeline view is itself a component that observes the state of the pipeline through an observer installed on the Vistrate singleton. This means that the core of the pipeline view also follows the standard com-

ponent template with a code paragraph, a data paragraph, and a view paragraph. The pipeline is easily reprogrammable or replaceable with another abstraction layer. Our current pipeline view is a basic graph implemented in D3 with unfoldable nodes that can either contain the view or the configuration of a component. The pipeline can be accessed in a vistrate through a keyboard shortcut or by pressing the pipeline button in the global toolbar.

6.3 The Component Repository

The component repository is implemented using Codestrates' package management features [18]. Prototype components can be pushed to or installed from a repository. New instances of an installed component can be created through the "Create new Vistrate Component" dialog accessible through the global toolbar (Figure 3(a)). Instantiating a component will copy the selected prototype, insert the given name and ids, and add it to the vistrate document.

Any component can be turned into a reusable prototype by making it a package and pushing it to the repository. This adds metadata to the component including a short description, a list of assets (e.g., images, JavaScript libraries, or CSS files), dependencies to other packages, and a changelog. This approach allows for reappropriation and customization of existing components. Components in the repository are also ready to use, meaning that an instance can immediately be configured using the pipeline view and, therefore, allows users to create visualization pipelines without programming. The current Vistrates component repository contains components for standard visualizations such as the bar chart, pie chart, line chart, geographical map, scatterplot, parallel coordinates, etc., components to analyze, transform, combine, and filter data, as well as utility components to load data, spawn headless browsers, and offload heavy parts of the pipeline to strong peers.

6.4 Component Canvas

Space is an important cognitive resource; we think and work in physical space [4]. Our implementation of the "Vistrates Component Canvas" package, therefore, allows for such spatial arrangements of component views on a 2D canvas. This can facilitate the sensemaking process to externalize thoughts and for distributed cognition during collaborative work, or it can become a dashboard for interacting with the visualizations (Figure 3(c)). In addition, users can add rich text and other media supported by HTML5 and annotate the canvas with a digital pen. Any content on the canvas—including component views—can be moved, scaled, and rotated. When installed, the Vistrates interface has a button in the global toolbar to add a component canvas paragraph. A canvas paragraph is styled to look like a whiteboard.

6.5 Mobile List View

In contrast to the component canvas, the "Vistrates Component List View" displays a single (selected) component view at a time. It provides a responsive component container that scales component views according to a device's available screen real-estate, e.g., to fully show a component view on a smartphone (Figure 7). Any available component view can be picked from an action menu.

7 Vistrates In Action

Here we explain the workflow when using Vistrates and describe how it can be used to realize the motivating scenario (Section 2).

7.1 The Workflow

A fresh Vistrates workflow begins with a user creating a codestrate with the Vistrates package installed. It can be copied by appending `?copy` to the URL. Then different visualization and analytics components can be installed by accessing the Vistrates component repository.

The first component a user will need is a component for loading data. The user will, e.g., create an instance of the *CSV loader* component from the component menu (Figure 3(a)). The user can now use its user interface (in its view) to upload a CSV file that will be stored as an asset on the webstrate layer. For larger datasets, a database component can be used that uses Webstrates' searchable CSV asset API to store data [25]. Additional visualization and analytics components can be instantiated and connected in the pipeline view (Figure 3(b)), or by manually editing the configuration in their data paragraphs.

To create a new component from scratch, the user instantiates a blank component and edits the code paragraph. A new component can also be created by modifying an instance of an existing component. To share a new component, it has to be turned into a package. Finally, there is no clear boundary between development and deployment in Vistrates since all changes are immediately reflected in the document. To get an application-like state, developers can simply use the persisted fullscreen functionality. For instance, to create a dashboard, the user can install the canvas package, create a canvas, expand it to fullscreen, and add components to it through a context menu.

7.2 Realizing the Scenario

In the following, we describe how Vistrates can realize the motivating scenario (Section 2). The accompanying video to this paper showcases a realization of the motivating scenario built with Vistrates.

7.2.1 Starting Out

To begin with, Daria can use a CSV loader component as a prototype for creating components containing the datasets needed by Vergil, and add these datasets to the component repository. Daria can create a blank codestrate with the Vistrates package installed and share its URL with Vergil. She and Vergil can use a WebRTC-based communication and remote pointers from Codestrates to communicate while Daria explains how Vistrates works. Together they can add existing components from the Vistrates component repository, instantiate them, and use the pipeline view to connect them together to, say, connect a ratings bar chart to a map component, so that Vergil can filter and choose businesses based on their ratings. The pipeline view helps transcend user expertise as pipelines can be configured by non-programmers such as Vergil through a graphical user interface.

7.2.2 Extension

The developer, Sam, can create a new component for the custom visualization (a Word Map) following the component structure (Listings 1 and 2) and add it to the component repository to enhance the analytical tools available to Vergil. Sam can integrate external libraries such as Leaflet [42] for drawing maps. Other components are based on graphing libraries such as Plot.ly [57] and Vega-Lite [64] (see details in Section 8). To test his component in Vergil's pipeline, Sam creates a copy of the Vergil's vistrate and integrates his component.

7.2.3 Mobile Use

Vergil can install a list view that allows for navigating the views of components on a mobile device. He can also connect the GPS data from his phone—provided by the JavaScript geolocation API—to a map component to center the map on his current geographic location. The Word Map that Sam has developed relies on computing tf-idf scoring [68] and word embedding, which is too heavyweight for a mobile device. Daria can add a component to the beginning and end of the heavy part of the pipeline so that mobile devices will offload that part of the computation to a desktop client or the server (see Section 8.1).

7.2.4 Transcending Activities

Vergil can use Codestrates' rich-text editing capabilities to write reviews and take notes. He can use the canvas component for sketching, and also arrange selected component views in a 2D layout. This allows Vergil to explore the data in a dashboard on his personal computer. Finally, the view paragraphs can be turned into slides allowing Daria to create a presentation (see Section 8.3).

7.2.5 Shareability and Collaboration

Vistrates is developed for collaborative analytical workflows. Vergil, Daria, and Sam utilize the video conferencing components to talk to each other during the analytical process. Interaction is synchronized across instances of a vistrate, while presenting remote pointers for collaborators, to support synchronous collaboration between the actors. For instance, using this, Vergil is able highlight a region of Toronto to

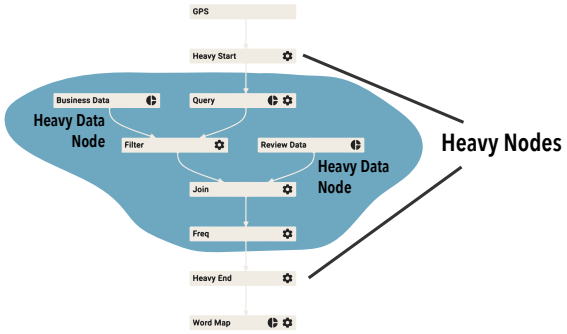

Fig. 6: A pipeline where *heavy* nodes are used to offload part of the pipeline when the client of a vistrate is a weak peer, e.g., a phone. The large datasets in the marked region—business and reviews data from the scenario—are also not loaded.

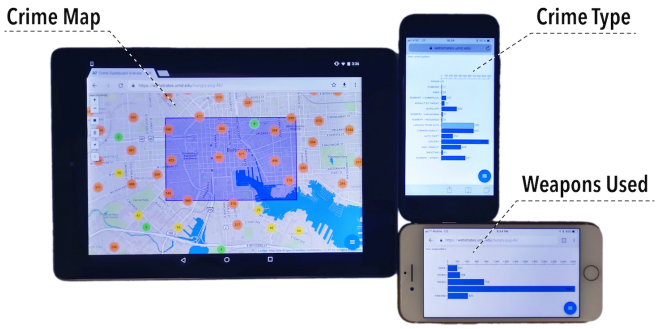

Fig. 7: A physical dashboard using two phones and a tablet for the crime analysis vistrate in Figure 3. The vistrate is opened on the web browser with mobile list view component installed to fit to the screen.

Daria as a developing area that can be explored. Vistrates applies a relaxed WYSIWIS model [69] to collaboration, which means a client can have, e.g., a canvas in full screen while another shows the code or pipeline view while their content and execution is kept synchronized.

8 Additional Examples

In this section, we will elaborate on a few examples that showcase the expressive power of the Vistrates platform.

8.1 Computation Offloading

When all clients of the same vistrate execute their own code, it is possible for weaker peers to offload heavy computations to stronger peers. This can even be an entire subpart of the pipeline, as the example in Figure 6 shows, where the highlighted part of the pipeline is offloaded to stronger peers. Two components called *Heavy Start* and *Heavy End* handle the offloading. The start node will signal *help* to other clients if it is executed on a mobile device and pick one of the stronger clients that offers help. The communication between clients is realized using the Webstrates signaling API [67]. The chosen client will then execute their pipeline using the input of the weak client. When the heavy end node is reached, the strong client will provide the weak client with the result, and the weak client can then continue its own execution. This principle also works for multiple heavy start and end nodes. If no strong peer is available, a service implemented on the Webstrates server can be called through an HTTP request to spawn a headless browser instance pointing to the given vistrate. Beyond this, a *Heavy Data* component is also available to avoid attempting to load large datasets on weaker clients. This way, a client can present interactive visualizations without having to load the dataset.

This approach to computation offloading is implemented purely as new components without any changes to the core of Vistrates. This means that components for different approaches to distributed computing could be created, e.g., to support the kind of peer-to-peer distributed computation provided by VisHive [26].

8.2 Cross-Device Visualization

A vistrate can be opened on any device with a web browser. This provides an opportunity to create physical dashboards across multiple devices and for mobile use of vistrates. In Figure 7, we showcase a physical dashboard with two mobile phones and a tablet (inspired by VisTiles [41]). This dashboard is created by installing a Mobile List View on the vistrate from Figure 3 and selecting a single component view on each device. The phones show visualizations of aggregated crime data—crime type and weapon used—and the tablet show a geographical map of all the crimes in Baltimore. Filtering a view by interaction will trigger updates to synchronize views on other devices owing to the collaborative pipeline of Vistrates. By introducing heavy nodes, the phones never execute any aggregation, filtering, or analysis, but they can show the visualizations and support interaction.

```
vc = {
  ...
  libs: ["plotly.min.js"],
  init: function() {
        this.plotDiv = document.createElement("div");
        this.view.content = this.plotDiv;
        this.update_view = () => {
          let layout = {"xaxis.range": this.data};
          this.Plotly.relayout(this.plotDiv, layout);
        };
        ...
        this.draw_plot = () => {
          ...
          this.plotDiv.on('plotly_relayout', (eventdata) => {
              this.data = [eventdata['xaxis.range[0]'], eventdata['xaxis.range[1]']];
          });
        };
  },
  update: function(source) {
        ...
        this.update_view();
  }
};
```

Listing 3: Creating a Vistrate component from a Plot.ly line chart.

8.3 Integrating Visualizations in a Slideshow

As Vistrates is built on top of Codestrates, it is out-of-the-box compatible with, e.g., a slideshow package from Codestrates. This package wraps a view paragraph into a slide as part of a slideshow—including cross-device presenter notes and remote pointing (Figure 1). Each slideshow can be styled by creating a theme—a set of stylesheets—that can specify how the visualizations and text appear in the presentation. Visualizations in a vistrate, when wrapped in a slide, are still interactive as their event handlers are retained, and this enables interactive and dynamic presentations when utilizing Vistrates.

8.4 Creating Vistrate Components from Existing Libraries

With Vistrates, it is easy to create new components by wrapping visualizations built in existing libraries such as Plot.ly, Vega, or D3 (e.g., Figure 1). This provides developers with a powerful set of tools to reuse developments not made within Vistrates, but wrap these in the Vistrates data and synchronization flow. It further enables developers to utilize existing charting and interaction primitives in these libraries. As an example, Vistrates allows for making Plot.ly visualizations collaborative so that they can be used to create filters on the data, which can be piped to other analyses. The code in Listing 3 showcases how Plot.ly can be used to initialize a line chart, save the line chart selection in the data paragraph, and update selections across clients.

9 Discussion

A unified approach to transcend analytical activities, device ecologies, levels of expertise, and modes of collaboration has the potential to bridge the gap between data science and visualization [13] as well as the gap between research and real-world adoption. Vistrates is a proof-of-concept implementation that shows the technical feasibility of such an approach. While it is a step towards fulfilling our vision for ubiquitous analytics [28], the vision will not be accomplished by

a single group of researchers, and as such the current framework and implementation also comes with limitations.

9.1 Limitations and Future Work

Scalability. All code in Vistrates is currently executed within a browser, which has at least two limitations: (1) reliance on JavaScript and (2) the available computational power. Data scientists often use languages such as Python and R that provide multiple efficient libraries for data transformation and machine learning, which JavaScript does not provide to a similar extent. As a next step, we are making Vistrates a mixed environment, where, e.g., data analysis components developed in Python can be executed in the cloud and interleaved with visualization components developed in JavaScript. Beyond this, it can be challenging to deal with large datasets and execute complex computations driving the data analysis in a web browser. Our computational offloading functionality showcases a first step to support mobile devices. In fact, this feature enables viewing the word cloud and Word Map visualizations on mobile devices by offloading the tf-idf analysis and layout computation on the large Toronto datasets to a computer (see supplementary video). However, this offloading is still limited since it just executes the code on a different device with better capabilities. A future work is to extend Vistrates to support distributed and parallel methods (cf. imMens [45]) and progressive visual analytics [70] for better scalability. Finally, the downside of a vistrate containing its entire source code is that the web browser needs to download all the required resources on page load to execute the vistrate. This can be slow on low bandwidth connections. In the future, we plan to add a lightweight mode to Vistrates where resources are loaded on demand.

Usability. Another limitation of the current implementation is its usability. Vistrates supports multiple levels of abstraction through development, pipeline, and canvas views. However, the current proof-of-concept is centered on a linear development view based on literate computing. More abstraction levels should be available to support certain activities and users: (1) shelf-based configuration such as in Tableau and Polestar to assemble components, (2) provenance tracking using interaction and insight histories [33] for visual exploration, and (3) better mobile interfaces driven by responsive visualization [5, 8].

Flexibility. The core framework in Vistrates is designed to be flexible, with only a few constraints such as the usage of predefined `src`, `props`, and `output` properties described in Section 6.1. Therefore it is difficult at this stage to provide guidelines for a good component design since the components can be defined at multiple granularities. For instance, an aggregation algorithm can be developed to be a standalone component as in our current approach or integrated into a visualization component. The former is better suited for extensibility and recomposability, while the latter leads to a simpler pipeline view that is easier to use. This tradeoff between flexibility and usability exists when creating new components within Vistrates. To answer this challenge, we are currently developing visual analytic templates—groups of components for specific data types and tasks—that are more meaningful and easily extensible by target users.

Another key property of the Vistrates framework is that once users start to gain expertise, the environment allows them to exercise that expertise by transcending abstraction levels. This property along with the component repository offers flexibility to develop new abstraction levels. However, leveraging this flexibility can be challenging, especially for non-programmers. To answer this, solutions at multiple levels need to be investigated including documentation standards, design patterns, and community support to share knowledge in Vistrates.

9.2 Implications

With Vistrates, we hope to advance a trend of creating software environments that can support *computational thinking* [78]. We believe that there is not only a pedagogical and educational challenge to heighten computational literacy in our society, but also a need for tools that make it easy to exploit the power of programming and bridge the skills of experts and novices. Vistrates can also support the design of learning environments for analytics and visualization, where the complexity level can be increased gradually as students improve their skills. Vistrates continues a trend set by the work on interactive notebooks for creating reproducible data analysis and science [66]. The principles applied in the design of Vistrates allows for the creation of digital artifacts that can include interactive visualizations and allow the user or reader to experiment with the data, but also to peek behind the scenes and tinker with the implementation.

9.3 Universal Component Model for Ubilytics

Our work on Vistrates have given us a unique perspective on how future component models for ubiquitous analytics should be designed:

Standardized Integration and Embedding. Vistrates provides a framework to make most visualizations collaborative and composable, but existing libraries should also have some key functionality to make them easy to integrate. All libraries already have methods to (1) translate data to visualization, and (2) capture user interactions. But they should also contain methods to (3) extract the predicates for user interactions [65], and, most importantly, methods to (4) update visualizations based on the interaction predicates, such that they can be synchronized across clients. The Plot.ly line chart is an example of a specific visualization that fulfills all these requirements.

Declarative State Specification. The Vega [65] grammar showcased the power of declarative specifications for creating interactive graphics. Similar ideas are needed to specify the state of a visualization, such that the state can be synchronized across clients. In Vistrates, this is currently left entirely up to the developer, which has its merits and drawbacks. It would be interesting to study whether Vistrates could support developers in creating declarative specifications of visualization state and interpret them again.

10 Conclusion

We have presented Vistrates, a holistic and sustainable data analysis and visualization environment designed using the principles of malleability, shareability, and distributability. Built on top of the Webstrates [40] and Codestrates [58] platforms, Vistrates provides a collaborative pipeline that supports the full range of visualization and analysis activities—including data management (ETL), exploration, sensemaking, and presentation—for teams of collaborators of diverse expertise—including developers, analysts, and laypersons. Using the basic Vistrate platform, we have built an initial component model that enables bundling disparate visualization and analytics functionality into reusable prototype-based components. These components can easily be instantiated and plugged into the data flow pipeline, even using drag-and-drop on a mobile device, and can be optionally customized down to the actual source code itself. We have demonstrated the utility of Vistrates in a scenario involving data-driven travel planning, as well as in three examples involving server-side computation, wrapping existing web components, and cross-device visualization.

In the future, we envision using Vistrates as a platform for a multitude of visualization projects. The fact of the matter is that the core Vistrate features are simply too convenient to give up, and they come at minimal cost; building a Vistrate component instead of freestanding D3 or Vega code will make the result collaborative, shareable, and persistent. However, beyond such convenience arguments, an important future research activity is to look deeper into the broad topic of component models for visualization and analytics. The component framework we have built using the basic Vistrate platform for this paper is merely a suggestion, and we claim by no means that it is a final, definite component model. More work on this is necessary to realize the grand vision of a universal visualization platform [10].

Acknowledgments

We are grateful to the anonymous reviewers for their feedback. We also thank Brian Ondov for the voice-over of the accompanying video. This work was supported by the U.S. NSF award IIS-1539534, the DABAI project (IFD-5153-00004B), and the Aarhus University Research Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the respective funding agencies.

REFERENCES

[1] C. Ahlberg. Spotfire: An information exploration environment. *SIGMOD Record*, 25(4):25–29, 1996. doi: 10.1145/245882.245893

[2] Altair. `http://altair-viz.github.io`, accessed March 2018.

[3] F. Amini, N. H. Riche, B. Lee, A. Monroy-Hernandez, and P. Irani. Authoring data-driven videos with DataClips. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):501–510, 2017. doi: 10.1109/TVCG.2016.2598647

[4] C. Andrews, A. Endert, and C. North. Space to think: large high-resolution displays for sensemaking. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 55–64. ACM, 2010. doi: 10.1145/1753326.1753336

[5] K. Andrews and A. Smrdel. Responsive Data Visualisation. In *EuroVis 2017 - Posters*. The Eurographics Association, 2017. doi: 10.2312/eurp.20171182

[6] B. Bach, N. Kerracher, K. W. Hall, S. Carpendale, J. Kennedy, and N. H. Riche. Telling stories about dynamic networks with graph comics. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 3670–3682. ACM, 2016. doi: 10.1145/2858036.2858387

[7] S. K. Badam and N. Elmqvist. PolyChrome: A cross-device framework for collaborative web visualization. In *Proceedings of the ACM Conference on Interactive Tabletops and Surfaces*, pp. 109–118, 2014. doi: 0.1145/2669485.2669518

[8] S. K. Badam and N. Elmqvist. Visfer: Camera-based visual data transfer for cross-device visualization. *Information Visualization*, 2017. doi: 10.1177/1473871617725907

[9] S. K. Badam, E. Fisher, and N. Elmqvist. Munin: A peer-to-peer middleware for ubiquitous analytics and visualization spaces. *IEEE Transactions on Visualization and Computer Graphics*, 21(2):215–228, Feb 2015. doi: 10.1109/TVCG.2014.2337337

[10] S. K. Badam, R. Rädle, C. N. Klokmose, and N. Elmqvist. Towards a unified visualization platform for ubiquitous analytics. In *Proceedings of the ACM CHI Workshop on Mobile Visualization*, 2018.

[11] R. M. Baecker. *Readings in Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration*. Elsevier, 1993.

[12] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *Proceedings of the International Conference on Weblogs and Social Media*. The AAAI Press, 2009. doi: 10.13140/2.1.1341.1520

[13] A. Batch and N. Elmqvist. The interactive visualization gap in initial exploratory data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):278–287, 2018. doi: 10.1109/TVCG.2017.2743990

[14] M. Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 446–453, 2000. doi: 10.1145/332040.332473

[15] M. Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the ACM Conference on Advanced Visual Interfaces*, pp. 15–22, 2004. doi: 10.1145/989863.989865

[16] M. Beaudouin-Lafon. Towards unified principles of interaction. In *Proceedings of the Biannual Conference on Italian SIGCHI Chapter*, pp. 1:1–1:2. ACM, New York, NY, USA, 2017. doi: 10.1145/3125571.3125602

[17] Bl.ocks. `https://bl.ocks.org/`, accessed June 2018.

[18] M. Borowski, R. Rädle, and C. N. Klokmose. Codestrates packages: An alternative to "one-size-fits-all" software. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems*, pp. LBW103:1–6. ACM, New York, NY, USA, 2018. doi: 10.1145/3170427.3188563

[19] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, 2009. doi: 10.1109/TVCG.2009.174

[20] M. Bostock, V. Ogievetsky, and J. Heer. D^3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[21] B. C. M. Cappers and J. J. van Wijk. Exploring multivariate event sequences using rules, aggregations, and selections. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):532–541, 2018. doi: 10.1109/TVCG.2017.2745278

[22] E. H.-h. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of the IEEE Symposium on Information Visualization*, pp. 69–75. IEEE, 2000. doi: 10.1109/INFVIS.2000.885092

[23] H. Chung, C. North, J. Z. Self, S. L. Chu, and F. K. H. Quek. VisPorter: facilitating information sharing for collaborative sensemaking on multiple displays. *Personal and Ubiquitous Computing*, 18(5):1169–1186, 2014. doi: 10.1007/s00779-013-0727-2

[24] Google Colaboratory. `https://colab.research.google.com/`, accessed Feb 2018.

[25] Webstrates CSV Assets API. `https://webstrates.github.io/userguide/api/assets.html#creating-searchable-csv-assets`, accessed June 2018.

[26] Z. Cui, S. Sen, S. K. Badam, and N. Elmqvist. VisHive: Supporting web-based visualization through ad hoc computational clusters of mobile devices. *Information Visualization*, 2018. doi: 10.1177/1473871617752910

[27] P. Dourish and M. Chalmers. Running out of space: Models of information navigation. In *Short paper presented at HCI*, vol. 94, pp. 23–26, 1994.

[28] N. Elmqvist and P. Irani. Ubiquitous analytics: Interacting with big data anywhere, anytime. *IEEE Computer*, 46(4):86–89, 2013. doi: 10.1109/mc.2013.147

[29] C. Gutwin and S. Greenberg. Design for individuals, design for groups: Tradeoffs between power and workspace awareness. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pp. 207–216, 1998. doi: 10.1145/289444.289495

[30] A. H. Hajizadeh, M. Tory, and R. Leung. Supporting awareness through collaborative brushing and linking of tabular data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2189–2197, 2013. doi: 10.1109/TVCG.2013.197

[31] D. Hansen, B. Shneiderman, and M. A. Smith. *Analyzing Social Media Networks with NodeXL: Insights from a Connected World*. Morgan Kaufmann Publishers, Sept. 2010.

[32] J. Heer and M. Agrawala. Design considerations for collaborative visual analytics. *Information Visualization*, 7(1):49–62, 2008. doi: 10.1057/palgrave.ivs.9500167

[33] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, 2008. doi: 10.1109/TVCG.2008.137

[34] J. Heer, F. B. Viégas, and M. Wattenberg. Voyagers and voyeurs: supporting asynchronous collaborative information visualization. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 1029–1038. ACM, New York, NY, USA, 2007. doi: 10.1145/1240624.1240781

[35] T. Horak, S. K. Badam, N. Elmqvist, and R. Dachselt. When David meets Goliath: Combining smartwatches with a large vertical display for visual data exploration. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 19:1–19:13. ACM, 2018. doi: 10.1145/3173574.3173593

[36] P. Isenberg, N. Elmqvist, J. Scholtz, D. Cernea, K.-L. Ma, and H. Hagen. Collaborative visualization: Definition, challenges, and research agenda. *Information Visualization*, 10(4):310–326, 2011. doi: 10.1177/1473871611412817

[37] P. Isenberg and D. Fisher. Collaborative brushing and linking for co-located visual analytics of document collections. *Computer Graphics Forum*, 28(3):1031–1038, 2009. doi: 10.1111/j.1467-8659.2009.01444.x

[38] K. Kim and N. Elmqvist. Embodied lenses for collaborative visual queries on tabletop displays. *Information Visualization*, 11(4):319–338, 2012. doi: 10.1177/1473871612441874

[39] U. Kister, K. Klamka, C. Tominski, and R. Dachselt. GraSp: Combining spatially-aware mobile devices and a display wall for graph visualization and interaction. *Computer Graphics Forum*, 36(3):503–514, 2017. doi: 10.1111/cgf.13206

[40] C. N. Klokmose, J. R. Eagan, S. Baader, W. Mackay, and M. Beaudouin-Lafon. Webstrates: Shareable dynamic media. In *Proceedings of the ACM Symposium on User Interface Software & Technology*, pp. 280–290. ACM, 2015. doi: 10.1145/2807442.2807446

[41] R. Langner, T. Horak, and R. Dachselt. VisTiles: Coordinating and combining co-located mobile devices for visual data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):626–636, 2018. doi: 10.1109/tvcg.2017.2744019

[42] Leaflet. `https://leafletjs.com/`, accessed June 2018.

[43] B. Lee, P. Isenberg, N. H. Riche, and S. Carpendale. Beyond mouse and keyboard: Expanding design considerations for information visualization

interactions. *IEEE Transactions of Visualization and Computer Graphics*, 18(12):2689–2698, 2012. doi: 10.1109/TVCG.2012.204

[44] B. Lee, R. H. Kazi, and G. Smith. SketchStory: Telling more engaging stories with data through freeform sketching. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2416–2425, 2013. doi: 10.1109/TVCG.2013.191

[45] Z. Liu, B. Jiang, and J. Heer. imMens: Real-time visual querying of big data. In *Computer Graphics Forum*, vol. 32, pp. 421–430. Wiley Online Library, 2013. doi: 10.1111/cgf.12129

[46] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, J. Stasko, and U. Lehi. Data Illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 123:1–123:13, 2018. doi: 10.1145/3173574.3173697

[47] N. Mahyar and M. Tory. Supporting communication and coordination in collaborative sensemaking. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1633–1642, 2014. doi: 10.1109/TVCG.2014.2346573

[48] W. McGrath, B. Bowman, D. McCallum, J. D. Hincapié-Ramos, N. Elmqvist, and P. Irani. Branch-explore-merge: Facilitating real-time revision control in collaborative visual exploration. In *Proceedings of the ACM Conference on Interactive Tabletops and Surfaces*, pp. 235–244. ACM, 2012. doi: 10.1145/2396636.2396673

[49] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 4073–4085, 2016. doi: 10.1145/2858036.2858435

[50] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal event sequence simplification. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2227–2236, 2013. doi: 10.1109/TVCG.2013.200

[51] M. Nouwens and C. N. Klokmose. The application and its consequences for non-standard knowledge work. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 399:1–399:12. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3173973

[52] SciPy. `http://scipy.org`, accessed March 2018.

[53] Observable, 2018. `https://beta.observablehq.com/`, accessed Feb 2018.

[54] D. G. Park, S. M. Drucker, R. Fernandez, and N. Elmqvist. ATOM: A grammar for unit visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2018. To appear. doi: 10.1109/TVCG.2017.2785807

[55] F. Pérez and B. E. Granger. IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, 2007. doi: 10.1109/MCSE.2007.53

[56] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of the International Conference on Intelligence Analysis*, vol. 5, pp. 2–4, 2005.

[57] Plot.ly. `https://plot.ly/`, accessed June 2018.

[58] R. Rädle, M. Nouwens, K. Antonsen, J. R. Eagan, and C. N. Klokmose. Codestrates: Literate computing with webstrates. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 715–725. ACM, 2017. doi: 10.1145/3126594.3126642

[59] D. Ren, T. Höllerer, and X. Yuan. iVisDesigner: Expressive interactive design of information visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2092–2101, 2014. doi: 10.1109/TVCG.2014.2346291

[60] J. Roberts, P. Ritsos, S. K. Badam, D. Brodbeck, J. Kennedy, and N. Elmqvist. Visualization beyond the desktop—the next big thing. *IEEE Computer Graphics and Applications*, 34(6):26–34, Nov 2014. doi: 10.1109/MCG.2014.82

[61] J. C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the International Conference on Coordinated and Multiple Views in Exploratory Visualization*, pp. 61–71. IEEE, 2007. doi: 10.1109/cmv.2007.20

[62] A. Sarvghad and M. Tory. Exploiting analysis history to support collaborative data analysis. In *Proceedings of the Graphics Interface Conference*, pp. 123–130, 2015. doi: 10.20380/GI2015.16

[63] A. Satyanarayan and J. Heer. Authoring narrative visualizations with Ellipsis. *Computer Graphics Forum*, 33(3):361–370, 2014. doi: 10.1111/cgf.12392

[64] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030

[65] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 669–678. ACM, 2014. doi: 10.1145/2642918.2647360

[66] H. Shen. Interactive notebooks: Sharing the code. *Nature News*, 515(7525):151, 2014.

[67] Webstrates Signaling API. `https://webstrates.github.io/userguide/api/signaling.html`, accessed June 2018.

[68] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972. doi: 10.1108/eb026526

[69] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Information Systems*, 5(2):147–167, Apr. 1987. doi: 10.1145/27636.28056

[70] C. D. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014. doi: 10.1109/TVCG.2014.2346574

[71] C. Stolte and P. Hanrahan. Polaris: a system for query, analysis and visualization of multi-dimensional relational databases. In *Proceedings of the IEEE Symposium on Information Visualization*, pp. 5–14, 2000.

[72] Tableau. `https://www.tableau.com/`, accessed June 2018.

[73] J. C. Tang. Findings from observational studies of collaborative work. *International Journal of Man-Machine Studies*, 34(2):143–160, 1991. doi: 10.1016/0020-7373(91)90039-A

[74] M. Tobiasz, P. Isenberg, and S. Carpendale. Lark: Coordinating co-located collaboration with information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1065–1072, 2009. doi: 10.1109/tvcg.2009.162

[75] F. B. Viégas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon. ManyEyes: A site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007. doi: 10.1109/TVCG.2007.70577

[76] M. Weiser. The computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991. doi: 10.1145/329124.329126

[77] W. Willett, J. Heer, and M. Agrawala. Scented widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1129–1136, 2007. doi: 10.1109/TVCG.2007.70589

[78] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006. doi: 10.1145/1118178.1118215

[79] H. Xia, N. Henry Riche, F. Chevalier, B. De Araujo, and D. Wigdor. DataInk: Direct and creative data-oriented drawing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 223:1–223:13, 2018. doi: 10.1145/3173574.3173797

[80] M. A. Yalcin, N. Elmqvist, and B. B. Bederson. Keshif: Rapid and expressive tabular data exploration for novices. *IEEE Transactions on Visualization and Computer Graphics*, 2017. doi: 10.1109/TVCG.2017.2723393