Interactivity is a key concern in system design. Judiciously designed interaction provides a more cohesive user experience and enables people to steer algorithmic processes — for example, guiding predictions when the input or output domains are unbounded. Yet, despite decades of research in human-computer interaction, reasoning about interaction remains difficult. Automated design and retargeting, for instance, requires more tractable representations of interaction than the imperative callbacks used today. My research develops **new declarative models of interaction**, which describe *what* an interaction should accomplish rather than *how* it should be computed, and leverages them in **novel interactive design systems**.

In my thesis, I study these issues in the domain of **interactive data visualization**. Data visualization provides an opportunity for broad impact as its use has gone mainstream — from business intelligence to data-driven journalism, society has embraced visualization to record, analyze, and communicate data. It presents an attractive petri dish for my research as interaction is critical to effective visualization, allowing users to iteratively refine their mental models by engaging data in dialogue. While visualization offers a well-constrained design space that has been modeled formally, existing models account for interaction poorly, if at all. Moreover, these models have only been instantiated in textual programming languages, and significant opportunity exists to exploit them within higher-level systems that accelerate analysis and design.

My work has focused on the design and development of the **Reactive Vega stack** (Figure 1). It features two new declarative interaction models: Reactive Vega, an expressive low-level representation that extends techniques from both Functional Reactive Programming and streaming databases [3]; and Vega-Lite, a higher-level grammar that enables *concise systematic enumeration* of interaction techniques [1]. These components build one on top of the other, providing *platforms for further research*. I have investigated the implications of declarative interaction on the architecture of visualization systems [2], and how it enables Lyra, a direct-manipulation visualization design environment [4].

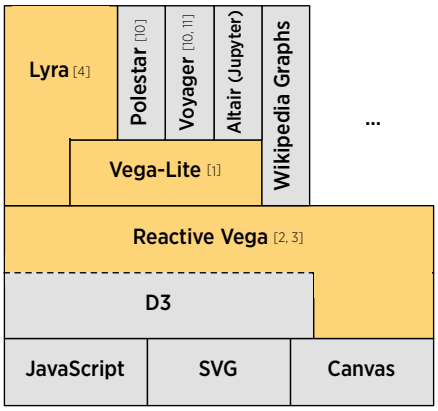
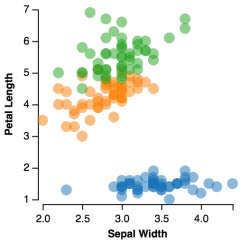


Figure 1: The Reactive Vega stack provides a platform for interactive visualization research, and has been integrated into production systems such as Wikipedia and the Jupyter Notebook. Highlights are my thesis contributions.

These systems have been released as open-source projects, have been widely adopted, and have given rise to an *ecosystem* of interactive visualization tools. Users can author an exploratory visualization in the **Jupyter Notebook**, export it to Lyra via Vega-Lite and add an explanatory annotation layer, and then embed the resultant Reactive Vega visualization within a **Wikipedia** article. As a result, rather than a single monolithic system, the Reactive Vega stack facilitates development of targeted applications, and allows users to work at the level of abstraction most suited for the task at hand.

REACTIVE VEGA: DECLARATIVE INTERACTIVE VISUALIZATION + A STREAMING DATAFLOW ARCHITECTURE

Declarative specification has become the dominant means of authoring data visualizations, but declarative support for interaction techniques remains weak. At best, users can use simple palettes of common techniques (e.g., brushing, panning, etc.) but customization is limited. For custom interaction, users must program *imperative* event handling callbacks which undo the benefits of declarative design. The complexity of state management and interleaved execution falls to the user, not the underlying system, and has provoked the colloquialism "callback hell."




```
{
  "data": [{"name": "iris", "url": "data/iris.json"}],

  "scales": [{
      "name": "sx", "type": "linear", "range": "width",
      "domain": {"data": "iris", "field": "sepalWidth"}
    },
    {"name": "sy", ...}, {"name": "sc", ...}
  ],

  "axes": [{"type": "x", "scale": "sx"}, ...],

  "marks": [{
    "type": "symbol",
    "from": {"data": "iris"},
    "properties": { "update": {
      "x": {"scale": "sx", "field": "sepalWidth"},
      "y": {"scale": "sy", "field": "petalLength"},
      "fill": {"scale": "sc", "field": "species"},
      "fillOpacity": {"value": 0.5},
    }}
  }]
}
```

Figure 2: Vega's declarative visual encoding abstractions. Data is imported from a URL. Scales are defined to transform data values to visual values. Properties of graphical marks (here, symbols) are determined by scale mappings. Guides (here, axes) are instantiated as well.

In response, I designed Reactive Vega: a declarative language for *interactive* data visualization that models user interaction as *streaming data* [2, 3]. Alongside existing declarative visual encoding primitives (Figure 2), Reactive Vega introduces *event selectors* and *signals*, two Functional Reactive Programming constructs. Event streams, defined with a novel CSS-inspired *selector syntax*, abstract the complexity of capturing and sequencing input events. For example `[mousedown, mouseup] > mousemove` defines a stream of drag events — a sequence that would previously require three callbacks to modify external state. Event streams drive signals: dynamic expressions that are automatically reevaluated when new events fire. Signals parameterize visual encoding primitives, thereby endowing them

with reactive semantics. These declarative specifications are expressed with JavaScript Object Notation (JSON), and are parsed to construct a dataflow graph. By adapting techniques from streaming databases, the dataflow operators perform computations on *changesets* of tuples or scenegraph elements, and are *replayed* if signal parameter values change.

Supporting expressive interactive visualizations required extending both reactive and data stream methods. As they are backed by input events, signal values are, by default, defined in pixel space. Reactive Vega introduces *scale inversions* and *predicates* to lift signal values to the data domain and define data queries respectively (Figure 3). These abstractions generalize an interaction technique, allowing it to coordinate multiple visualizations or be reused across them. Moreover, to support data-driven multi-view displays, Reactive Vega's dataflow is conditioned on data values. As data is unobserved at compile-time, the dataflow graph *dynamically rewrites itself* at runtime by extending or pruning branches based on varying input data or events.

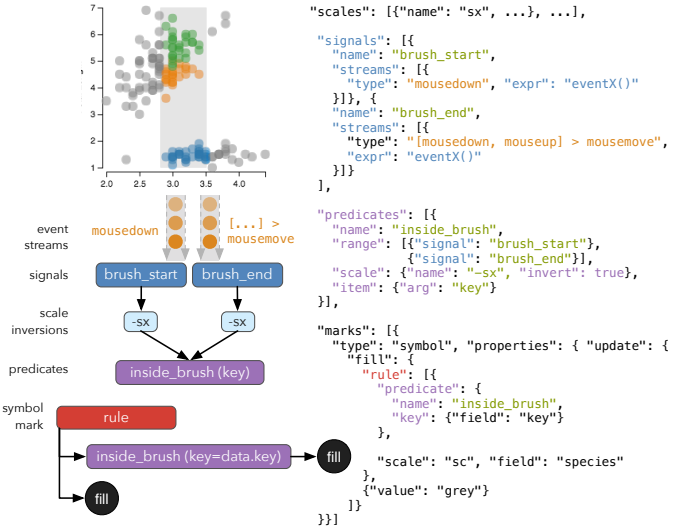


Figure 3: A declarative Reactive Vega specification for a brushing interaction. Signals capture mousedown and drag events, and calculate the brush extents. The extents feed a range predicate, expressed over data space via a scale inversion, to determine the symbol mark's fill color.

Together, Reactive Vega's primitives and architecture extend the benefits of declarative specification to interaction design. With event streams and signals, users need only describe the relationship between input events and interactive state respectively. State no longer needs to be manually maintained, but is automatically updated when new events occur. This decoupling also facilitates retargeting an interaction (e.g., to support mouse and touch) as alternate source event streams can be specified for a signal without affecting any downstream logic. The system runtime is entirely responsible for managing the complexity of execution and can even unobtrusively optimize processing. Thus, Reactive Vega is both sufficiently expressive to author a diverse range of interactive visualizations (Figure 4) and often more performant that D3 [2].

VEGA-LITE: A GRAMMAR OF INTERACTIVE GRAPHICS

As its abstractions are relatively low-level, it is apt to describe Reactive Vega as an "assembly language" for interactive visualization. While this approach is useful for custom *explanatory* visualization, the verbose specifications it yields can impede the rapid authoring process crucial for *exploratory* visualization. Analysts instead prefer concise, high-level visualization grammars (e.g., R's ggplot2) which resolve ambiguous specifications by applying smart default values. A smaller language surface area also makes higher-level reasoning and inference tasks more tractable — for example, providing fewer visual encoding properties to enumerate and rank has enabled novel visualization recommender systems [10, 11].

To meet this goal of concision, I developed the formalism for a novel *grammar of interaction* within the higher-level Vega-Lite language [1]. Interaction techniques in Vega-Lite are composed of *selections*: an abstraction over input event processing, the visual elements or data points a user has chosen, and a predicate function for inclusion testing (each of these are separate constructs in Reactive Vega). To concisely instantiate common defaults, three *selection types* are offered: *point*, *list*, and *interval*, to select a single point of interest, a discrete list of points, or a continuous region of points respectively. Users can manually override the constituent components of a selection, for example populating a point selection on *hover* rather than the

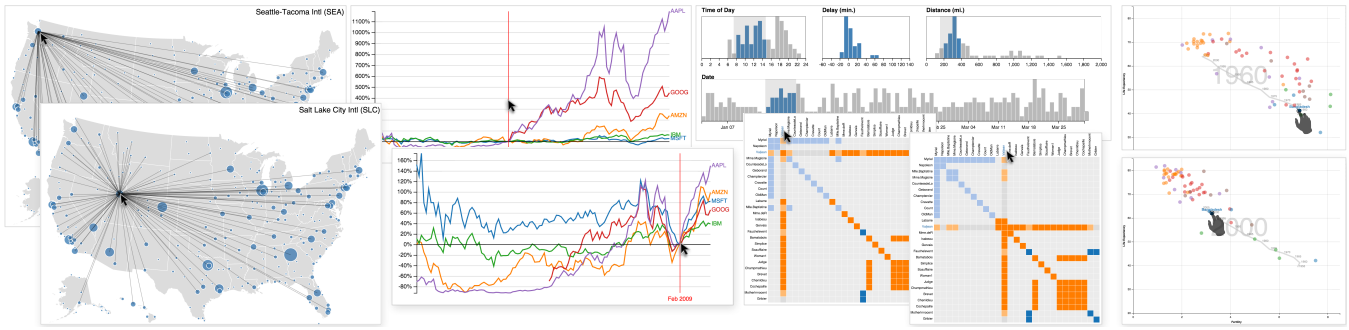


Figure 4: Example interactive visualizations constructed with Reactive Vega. (left-right) U.S. airport connections; interactive normalization of time-series stock prices; cross filtering multiple coordinated histograms; reordering rows and columns of an adjacency matrix; and, DimpVis [13], a touch-based technique for browsing time-series data that emulates Hans Rosling's narrative style.

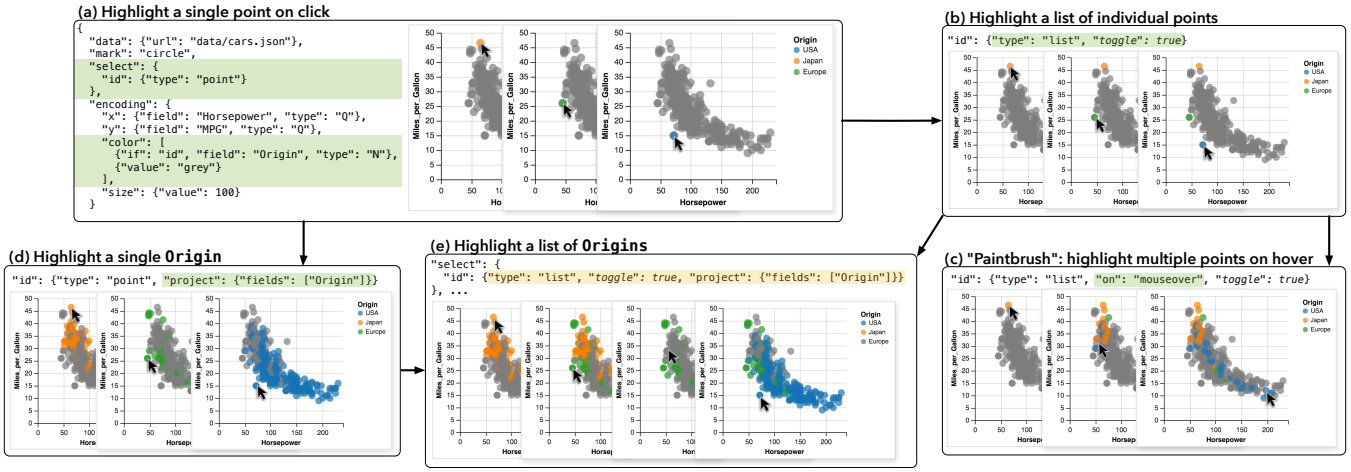


Figure 5: A partial depiction of the space of Vega-Lite interaction techniques that can be systematically enumerated for a scatterplot.

default *click*. To further promote rapid specification, I identified recurring design patterns and encapsulated them within a set of *selection transformations*. For instance, a *toggle* transform (Fig 5(b)) adds or removes data points from a list selection when events occur, whereas a *translate* transform (Fig 6(c)) offsets a selection's spatial properties (or corresponding data values) via a sequence of events. Once defined, selections parameterize visual encodings by serving as input data, defining scale extents, or driving conditional logic. The Vega-Lite compiler ingests these higher-level specifications, also expressed as JSON, and emits lower-level Reactive Vega.

While selections are necessarily less expressive than Reactive Vega's primitives, they can nevertheless be used to construct both common and custom interaction techniques including panning, zooming, linked selection, and interactive index charts. This reduction in expressivity, however, is offset by significant gains in concision. A signature result is cross filtering as shown in Figure 6(c). In 2012, this interaction required a custom-built library, with its own idiosyncratic API; with Vega-Lite, a general-purpose language, cross filtering can now be realized with only 35 lines of JSON. Moreover, selections decompose an interaction method into semantic units that can be systematically varied as shown in Figures 5 and 6. As a result, not only can designers rapidly explore alternate points in the design space, but interaction techniques can now be systematically enumerated and generated as part of higher-level applications.

LYRA: VISUALIZATION DESIGN BY DIRECT-MANIPULATION AND DEMONSTRATION

Reactive Vega and Vega-Lite offer JSON syntaxes to facilitate programmatic generation of interactive visualizations within higher-level interactive applications. In my thesis, I explore this nascent space with Lyra: an interactive visualization design environment (VDE) [4]. Recognizing that Reactive Vega and Vega-Lite present a fundamental mismatch in representations — using *textual* languages to express *visual* output — Lyra instead enables authoring visualizations via direct-manipulation. Through drag-and-drop interactions, such as those shown in Figure 7, users bind data values to mark properties. A data

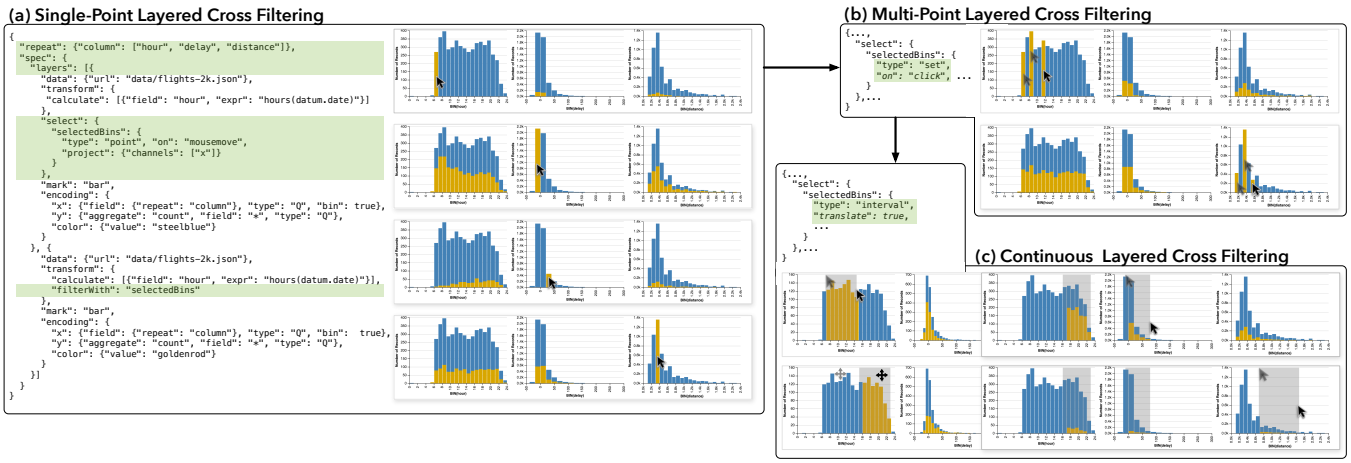


Figure 6: Cross filtering is an interaction technique popularized in 2012 by Mike Bostock's *custom-built* CrossFilter.js library. In 2015, I recreated it as a Reactive Vega specification (Figure 4) and in 2016, as an order-of-magnitude shorter Vega-Lite specification. Moreover, Vega-Lite's high-level specification language facilitates rapid exploration of alternative cross filtering interactions.

pipeline interface allows users to visually specify and inspect data transformations and layout algorithms. In ongoing work, I am extending Lyra to support interaction design through demonstration.

As a higher-level graphical interface, Lyra allows users to fluidly move between the different levels of abstraction. Direct-manipulation interactions and interactive demonstrations generate statements in Vega-Lite, which are compiled, and merged into a backing Reactive Vega specification; the visual inspectors provide complete control over the latter. Thus, users can iterate between rapidly creating recognizable output and making fine-grained customizations. This approach yields a diverse range of visualizations (Figure 8) without writing a single line of code and, in evaluative studies, users remarked that Lyra *"made [them] feel more in control"* and that *"there is a real joy in using Lyra."* Lyra has been noted as a "significant development" [8], nominated for an "Information is Beautiful" award [9], and is used by approximately **1,500 users-per-month** including journalists and educators.

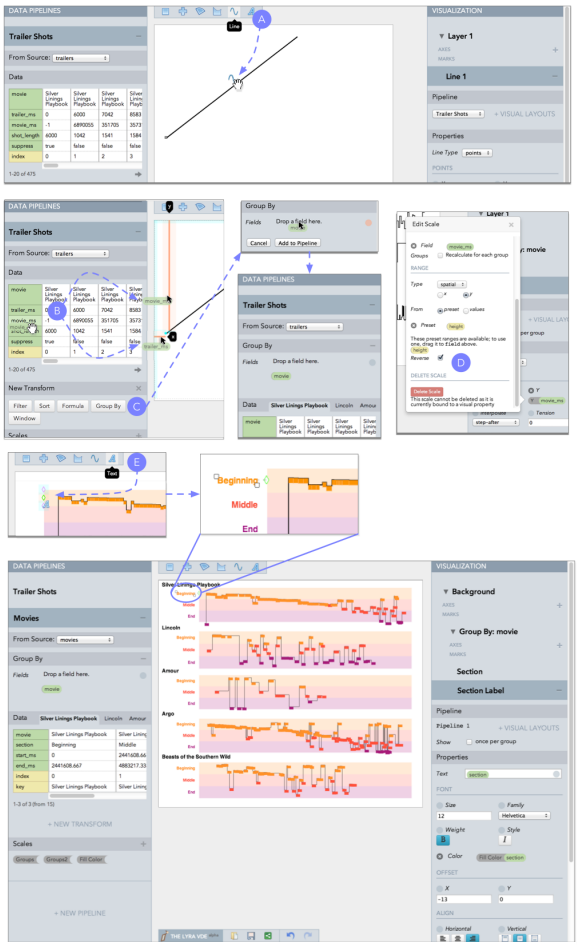


Figure 7: Using Lyra to recreate *Dissecting a Trailer* from the New York Times. (a) Add a line mark to the canvas. (b) Drag a field from a pipeline's data table to a drop zone to map it to a mark property. (c) Add a "group by" data transform to create a hierarchy. (d) Edit a scale definition to reverse the range. (e) Anchor text marks to rectangles using connectors.

FUTURE RESEARCH AGENDA

I plan to continue studying interaction through the Reactive Vega stack as it provides both a platform for developing novel interactive systems and, critically, a growing and engaged community to study their use with.

A Science of Interaction

Developing a generalized theory of interaction — one that answers questions such as what makes an interaction technique more effective than another, or what are principles for combining multiple techniques that preserves their individual advantages — has been difficult because existing empirical evaluations of interactions have been conducted largely in an ad-hoc manner. This is due, in part, to representations of interaction that have obscured how to isolate properties of an interaction technique as experimental variables. As Herbert Simon notes in *The Science of Design*, *"solving a problem simply means representing it so as to make the solution transparent,"* and my interaction models offer a promising way forward: for a constant Vega-Lite visualization, we can not only systematically generate interaction techniques, but also vary their constituent properties. Testing these alternative designs with human subjects will allow us to understand the costs and benefits associated with interactions and formulate design guidelines, as graphical perception studies have done for visual encodings.

Further study of these new representations themselves is also needed. In particular, we need to determine what cognitive loads they impose on users, as well as what new scaffolding [6] is needed as a result to support the authoring process.

Automated Design & Inference for Interactive Visualization

Performing inference over Reactive Vega and Vega-Lite specifications has already yielded new systems for data exploration [10] and visualization recommendation [11], and is driving my ongoing work on interaction design by demonstration. These applications, however, are only an initial exploration of this design space, and there is fertile ground to study how inference

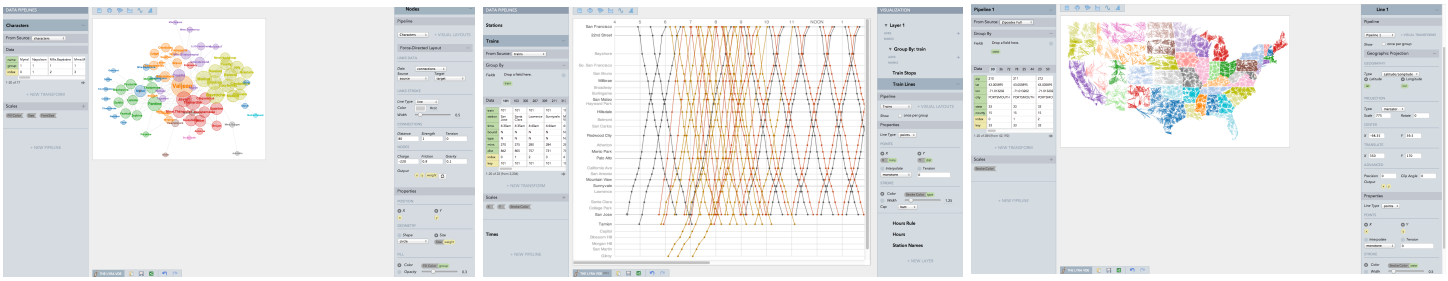


Figure 8: Example visualizations created with Lyra. (left-right) Character co-occurrences in Les Misérables; the schedule of the San Francisco Bay Area's CalTrain service in the style of E. J. Marey; ZipScribble by Robert Kosara.

procedures can be used to accelerate interactive analysis and visualization. For example, mining corpora of interactive visualization designs [7] can codify best practices and identify trends; this information can then be leveraged in a system like Lyra to provide "auto-complete" suggestions, or improve an existing design with a visualization "linter." As these systems are deployed, they should be capable of analyzing user interaction over time, and refining their inference models as needed.

Collaborative Analysis & Visualization

Data analysis and visualization are inherently collaborative processes, and support for multiple users will be an important ongoing consideration. For example, in studies I conducted with journalists, they reported feeling empowered by a system that allowed them to prototype narratives alongside the visualizations developed by their team members — a process they were previously locked out of [5]. However, to tighten iteration between collaborators, several open questions remain including how awareness and common ground is established between users, and how version control workflows may be adapted within graphical systems to support visual and interactive artifacts. The Reactive Vega ecosystem contains several such systems that can be readily built on to answer these questions and, as each system targets a different set of tasks, I aim to relate recurring design patterns to the shared underlying representations of interactive visualizations.

Moreover, in multi-user settings, it is especially unlikely that the same application satisfies all users. I am interested in studying how system architectures should evolve to enable users to create "mashups" — custom or one-off environments that repurpose and recombine existing interfaces and interactive functionality. Studying these issues within the Reactive Vega ecosystem is a natural extension of my work — for example, how might Lyra's direct-manipulation and demonstration capabilities be extracted into a standalone module (or *interactor* [3]) such that users could leverage it within Voyager [10] to annotate recommended visualizations or make them interactive?

Toolkits for Domain-Aware User Interfaces

I believe insights from conducting the above work in the Reactive Vega ecosystem will have ramifications for user interface toolkits more broadly. Generalizing these findings will rely on developing representations for a variety of task domains that mimic the role of Reactive Vega and Vega-Lite for visualization. *Information substrates* [12], developed by my collaborators at INRIA and Aarhaus University, offers a promising methodology to do so and I am interested in applying it to data science tasks such as data modeling and statistical analysis. By performing this work across several domains, I hope to identify recurring patterns and processes for constructing these domain-specific representations, and encapsulate them within a new genre of toolkits for domain-aware user interfaces. These toolkits would allow developers to prototype new domain-specific representations, synthesize simple interfaces in order to assess their expressiveness and effectiveness, provide standardized inference procedures, and would automatically instrument applications to learn from user interaction over time.

REFERENCES

1. Vega-Lite: A Grammar of Interactive Graphics. Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, Jeffrey Heer. *Proc. InfoVis 2016*. **Best Paper Award**.
2. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. Arvind Satyanarayan, Ryan Russell, Jane Hoffwell, Jeffrey Heer. *Proc. InfoVis 2015*.
3. Declarative Interaction Design for Data Visualization. Arvind Satyanarayan, Kanit Wongsuphasawat, Jeffrey Heer. *Proc. UIST 2014*.
4. Lyra: An Interactive Visualization Design Environment. Arvind Satyanarayan and Jeffrey Heer. *Proc. EuroVis 2014*.
5. Authoring Narrative Visualizations with Ellipsis. Arvind Satyanarayan and Jeffrey Heer. *Proc. EuroVis 2014*.
6. Visual Debugging Techniques for Reactive Data Visualization. Jane Hoffswell, Arvind Satyanarayan, Jeffrey Heer. *Proc. EuroVis 2016*.
7. Webzeitgeist: Design Mining the Web. Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, Jerry O. Talton. *Proc. CHI 2013*. **Best Paper Award.**
8. 10 Significant Visualization Developments: January to June 2014. Andy Kirk, *Visualizing Data*, *August 2014*. http://tinyurl.com/hyxy3lg
9. Information is Beautiful Award Shortlist. *Kantar*, *November 2015*. http://tinyurl.com/gkvsum8
10. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer. *Proc. InfoVis 2015*.
11. Towards a General-Purpose Query Language for Visualization Recommendation. Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, Jeffrey Heer. *HILDA 2016*.
12. Webstrates: Shareable Dynamic Media. Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, Michel Beaudouin-Lafon. *Proc. UIST 2015*. Best Paper Award.
13. DimpVis: Exploring Time-Varying Information Visualizations by Direct-Manipulation. B. Kondo and C. Collins. *Proc. InfoVis 2014*.