Jupyter on Steroids

ABSTRACT

Despite the plethora of recently proposed tools and frameworks, common data analysis tasks, such as data retrieval, curation and visualization remain laborious and non-trivial. The limitations of existing frameworks, pushes code-literate data analysts to interactive notebooks such as Jupyter.

In this work, we argue that interactive notebooks are suboptimal for commonly performed tasks, with regards to their ease of use and interactivity. We propose ViDeTTe, a framework designed to facilitate data analysis through the utilization of a template language. We implement a sample data analysis work flow and present the benefits of using ViDeTTe instead of commonly used imperative language in iPython and Jupyter notebooks.

ABSTRACT

This abstract is way too long. Most of it should be in the introduction. We should try to keep it under half column with a flow like: issue -> "This paper proposes..." -> Results if any.

A plethora of tools and frameworks has been proposed, by the database and systems community, for streamlining the admittedly arduous tasks of data retrieval, data curation and data visualization. Since these tasks, typically, have to be carried out in most data science projects, a significant portion of the afore-mentioned tools, focuses on providing a user-friendly way for performing them. Despite the very promising results, however, these tools often seem to limit flexibility as they only focus either on a predetermined set of use/analysis-cases or on only a small fraction of tasks of a typically much bigger pipeline. This lack of flexibility often pushes code-literate data scientists to the tested and proven path of interactive notebooks such as Jupyter.

Interactive notebooks allow the use of popular, high-level and highly expressive imperative languages, such as python, for analyzing data and composing the results into an easily readable notebook-like interface. Due to the wide popularity of such languages, there is also a huge collection of third-party libraries that can be used by data scientists as building blocks of a much bigger analytical process. Furthermore, the web environment of notebooks enables collaboration between data scientists, since it allows them to directly interact with the user interface in order to develop and run code, process data, generate visualizations, and lastly, compose their findings into an interactive (and re-runnable) report-like page, that contains code, visualizations and textual description of the analysis.

However as we show in this work, interactive notebooks are still suboptimal with regard to ease of use and interactivity. Setting up notebook environments and dependencies, obtaining and combining data and generating the respective visualizations, requires technical knowledge that often exceeds the skill-set of a typical data scientist. Lastly, while such notebooks support the generation of interactive visualizations, this interactivity is not an integral part of the data analysis process. To address these issues, we propose a set of extensions on interactive notebooks...

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

1. INTRODUCTION

User-friendly data analysis tools and frameworks often provide limited flexibility, as they usually focus on a predetermined set of use/analysis cases or a small fraction of the typically large data analysis pipelines. This lack of flexibility often pushes code-literate analysts towards the use of interactive notebooks such as Jupyter.

Allowing the use of high-level expressive imperative languages such as Python, interactive notebooks are able to assist data analysis, as well as composing results into easily readable notebook-like interfaces. In conjunction with the large number of third party libraries, make interactive notebooks a complete solution for developing, documenting and communicating code and visualizations to other analysts.

However as we show in this work, interactive notebooks are still suboptimal with regard to ease of use and interactivity. Setting up notebook environments and dependencies, obtaining and combining data and generating the respective visualizations, requires technical knowledge that often exceeds the skill-set of a typical data scientist. Lastly, while such notebooks support the generation of interactive visualizations, this interactivity is not an integral part of the data analysis process.

We address these issues, by extending interactive notebooks with a template language called ViDeTTe. The main contributions of this extension are:

- *Declarative semantics:* ViDeTTe implements formal

declarative *Model-View-View-Model* (MVVM) semantics.

> *Fill in why this is a good thing. I have no idea.*

- *Expressive template language:* Prior database work, treats a page as a database view. Building on that, our template language goes beyond SQL query and view definition in both style and fundamental expressiveness. It is a mixture of query as well as web templating language that works on ordered (arrays) and semi-ordered (JSON) data.

- We allow in-line declarative code directly in JSON...

In this work, we demonstrate the use of ViDeTTe via a walkthrough example. Specifically, we want to use website access data, plot an access count histogram, as well as the recorder user demographics (age groups). We then want to interact with the histogram plot and select a time region. We want this action to automatically update the second plot with the user demographics in the selected time window. We assume a Jupyter server, where the analysts develop their notebooks and a different database server where data is stored. To retrieve the entirety of the required data, we have to query two different databases and join the returned JSON files. Figure ?? shows how our databases are organized. Our fictional analyst will perform the following tasks:

- Data retrieval from remote databases.

- Data curation: Join data and prepare for visualization.

- Data visualization.

2. INTRODUCTION

Contributions We describe the following contributions of ViDeTTe:

> *We spend 2 full pages on the introduction (with comments but still an issue), which is not viable in a 6-page paper. On top of running out of space, it makes the section feel way too long and boring.*

> *We also waste too much space using these lists. We either change the formatting to start from the beginning of the line, or figure out a different way to write.*

> *My suggestion is: Move some of the abstract text here and present the list of contributions with minimal text and as little detail as possible. When we then describe the architecture/design/concept, we refer back to how it affects these challenges.*

- *Declarative Semantics:* We present a formal declarative MVVM semantics for ViDeTTe. To the best of our knowledge, this is the first formal presentation of MVVM declarative semantics by an MVVM framework. *YannisP: What extra do we offer in light of CIDR11 to the semantics?*

> *"MVVM" used without previous declaration*

Costas: Fundamentally, they are both diff driven application frameworks... Some differences are that the CIDR11 and SIGMOD10 version: 1.the template language was written in XML which was more verbose and therefore needed more effort to be written (more characters per line and more total lines of code for the same template instance). 2.Additionally, since template functions were not supported, the app developer often had to inline complex SQL++ queries in the template to "massage" the data appropriately. 3. Delta functions were not supported. Overall templates appeared more convoluted compared to the current version and therefore they were harder to follow. For more differences check respective file in Notes folder (at the root of the repo) *YannisP: A large difference between CIDR11 and current work is the aim to easily interface with JavaScript. This is behind the functions in the template language, the interfacing of JavaScript components, the JavaScript actions and (the missing) easy programming model for accessing JavaScript data (the latter can stay for journal or OOPSLA)* Hence, the declarative semantics of ViDeTTe can also serve as an introduction to the semantics of MVVM at large, and to the semantics of AngularJS and other MVVM frameworks, after one accounts for their limitations.

- *Expressive Template Language:* While we draw from prior database research work, which treats the page as a database view, the template language goes beyond SQL query and view definitions in both style and fundamental expressiveness *YannisP: Good point for a journal publication but on VLDB suffers from CIDR comparisons. The reason behind the higher expressiveness is new.* : It is a mixture of query language and web templating language that works on ordered data (arrays) and semi-structured data (JSON) *Costas: JSON encompasses the concept of arrays/ordered data* and is expressive enough to cover many common transformations without requiring the use of additional imperative JavaScript code. Notably, it is more expressive than the language of AngularJS, due to reasons pertaining to differences in their incremental rendering algorithms. *YannisP: Cannot be quickly backed up by the example. Two nested loops may be hard but a for/if combo is doable.* *Costas: How would it be backed up by example? We would have to introduce Angular and ViDeTTe templates in the intro, wouldn't we?* Furthermore, besides visualization, the templates can also input data and catch events/activate actions *Costas: actions have not been defined* , as a complete web application development framework should do.

- *Incremental Rendering:* The incremental rendering problem has similarities to Incremental View Maintenance (IVM) but also poses distinct new challenges due to (1) template language features that have not received attention by the database community, and (2) unlike IVM whose goal is to update a target materialized view, the incremental rendering algorithm is a *diff propagation* algorithm that results into a sequence of calls to the rendering methods of (the usually 3rd party) components. *YannisP: A diff and its resulting rendering method call should have been demonstrated in the context of the example.* Pertaining to Item (1) ViDeTTe solves the following diff propagation problems:

  1. *Handling Arrays:* Arrays are prime citizens of the data model. Therefore we reconsidered what is the appropriate language for describing diffs, so that diffs on arrays are included. *Costas: What do we mean when we say "language" for describing diffs* *YannisP:*

Current example has no motivation for insertion in the middle of array, which is where problems happen. Second we developed algorithms for propagating such diffs.

2. *Black-box functions in Template:* *YannisP: no example of a function either. It need not be the first example but I think there is just no example.* In the rare cases where the out-of-the-box expresiveness of the template language is not enough, the developer can include Javascript function invocations in her template. ViDeTTe offers a pay-as-you-go approach to the efficient diff propagation through JavaScript functions that may be part of the template: The developer may provide nothing (but the function itself), in which case the diff propagation algorithm will still work but may be slow *Costas: Let's avoid the word slow, let's just say it wouldn't work "incrementally", which is the more efficient.* , as it will be re-evaluating the function. Alternately, the developer may provide one or more diff management functions, which are suggested to her by ViDeTTe and lead to more efficient diff propagation. *Costas: We should also mention that we provide some of these functions out-of-the-box (e.g sortby, groupby etc...) YannisP: IMPORTANT: we need to discuss how this differs from angular's pay-as-you-go* *Costas: Angular does not provide something similar to ViDeTTe's delta functions, so there's nothing to pay as you go*

3. *Loops in Templates* ViDeTTe provides diff propagation over loop structures. *Costas: Why is that a big deal? This bullet doesn't illustrate why that's challenging or interesting...* *YannisP: would be far more convincing with fors and ifs*

The ViDeTTe incremental rendering also solves the following problems pertaining to its need to translate diffs into rendering method calls. *YannisP: as said above, problems should have memorable names and do not repeat the whole description continously* *Costas: refer to dictionary spreadsheet* https://docs.google.com/spreadsheets/d/1pn6EPZUnz_3Tc3e7FnveRxQXcs2uRqD0znERjI0-g0o/edit#gid=0

1. *Easy wrapping of 3rd party components* A novel triggering algorithm enables the easy and pay-as-you-go wrapping of Javascript visualization rendering methods. In particular, the developer in charge of wrapping a visualization component is only tasked with specifying the single diff that a rendering method handles. *Costas: This sentence doesn't make sense: "(The developer specifies the) single diff that a rendering method handles". The developer does not specify diffs, he specifies renderer wrappers and diff signatures (refer to dictionary). The number of renderer wrappers he needs to specify is anything between two (construct-destruct) and the total number of renderers supported by the visualization library* *YannisP: Needs example. Recall, earlier we introduced a diff and a corresponding renderer. Here we should change the assumption on renderer or the diff so that we can show simulation.* Therefore the size of the specification *Costas: what specification?* is proportional to the number of rendering methods she wishes to wrap, as opposed to the number of the possible kinds of diffs, which is typically far larger. *Costas: the latter number is: supported-diff opcodes X potential targets in unit instance* *YannisP: tell the number that shows how many diffs exist in our example. You don't have to substantiate yet why so many. The substantiation will come much later once we have shown the unit's "schema" and have explained how many kinds of diff exist. The most impressive introductions say "without my technique it would be N and with my technique it is M that is an order of magnitude below" Costas: I don't follow. Why do we need to specify the number of possible diffs. When reading this section it feels like we're comparing our system with a system similar to ours that does not support simulation. Angular does not work with diffs, so this is somewhat pointless.* When a diff is produced that does not correspond directly to a rendering method, ViDeTTe discovers how to indirectly support it by *simulating* the missing rendering method with existing wrapped rendering methods. *Costas: we should add something like: "This automation significantly simplifies the process of unit wrapping. Since competing frameworks do not provide a similar feature, unit developers are often required to manually perform it manually. This negatively impacts both the amount of time that needs to be spent in unit wrapping and the amount of lines of code that need to be written"* Lines-of-code experiments show the advantage of ViDeTTe in 3rd party wrapping productivity.

2. *Managing Direct Updates and Provenance Heterogeneity* ViDeTTe treats update diffs as first-class citizens, as opposed to simulating updates by insert-delete combinations. *Costas: Not sure why we mention this, Angular directives don't need to simulate updates diffs with insert-delete diffs. Is this something that other IVM algorithms do not support?* This feature leads to higher performance and also superior UI behavior, when the visualization components provide update renderer methods. It also creates a need for keeping track of the provenance of each part of the visualization, so that it can be identified and updated. Notice that provenance has to be converted into terms that the 3rd party component understands, creating a need for special conversion data structures and algorithms. *YannisP: Needs support of the example, both on the update and on the provenance. (The current example is an insertion and does not have any provenance problem.)*

- *Superior performance over other MVVM frameworks:* A net effect of the algorithms is that the client-side incremental rendering algorithm achieves better Big-O algorithmic complexity than the incremental rendering algorithm of AngularJS and also ouperforms it in the presented experiments.

Paper Outline. *YannisK: Revise after the sections have been finalized* The paper is structured as follows: In Section 3 we explain the structure of a ViDeTTe application as seen by an application developer. Section **??** describes ViDeTTe's internal architecture, while Sections **??-??** describe the incremental view maintenance algorithms that power the framework. Section **??** compares the performance and productivity gains of the proposed system against the state of the art. Finally, Sections **??** and 4 discuss related work and conclude the paper, respectively.

3. ViDeTTe APPLICATION DEVELOPMENT

YannisK: This section is currently disconnected from the introduction. After we decide on the main contributions in the introduction, we should connect it with the introduction. We present the structure and semantics of a ViDeTTe application as seen by its application developer. The internal execution of the application by the framework, along with the automatically applied optimizations, is the topic of Section **??**. *YannisP: There are two kinds of developers: The database-oriented developer and the JS developer. A JS developer can be an application developer. Hence, this is not the defining difference.*

Running example. We show how a developer can use ViDeTTe to create a temperature monitoring dashboard. The dashboard, shown in Figure **??**, consists of a chart showing the temperature readings as they arrive from a backend database. *YannisK: We should be careful how we describe the source of the new readings* The readings are represented by data points that are color-coded according to their value: Normal temperatures are shown in yellow, while lower and higher temperatures are shown with gradients of green and red, respectively. The normal temperature can be set by the user by utilizing the slider shown on the right side of the chart.

The dashboard uses a combination of visualization components, allowing us to demonstrate how ViDeTTe interfaces with complex visualization libraries. In particular, the application employs two visualization components; a HighCharts component [1], displaying the chart and a Slider component for setting the normal temperature. *YannisK: Should we explain that we build the slider component in house? I vote for now, as it may raise questions.* *Costas: We could say that we've wrapped all input types (see table: "Attribute Values" in: http://tinyurl.com/h5ktgqa) into units but it still does not explain why the range is color coded...*

Costas: Running example will be replaced. Revisit similar issues after it's finalized

YannisP: About Figure: 1: instead of a virtual database we can go for just a series of active wrappers ("active" in the sense that they catch modifications)

YannisP: There are two kinds of developers and the introduction never made the distinction: A db-oriented developer, who relies on the expressiveness of templates, and a Javascript developer who can write code, writes functions, interfaces components, writes complex actions. The distinctions need to become clear because some contributions pertain to the second kind of the developer. *Costas: I disagree with this distinction. Both developers need to write actions (in JS) in order to handle events. However, I do buy the argument that a naive application developer may not be able to write his own (delta) functions. Perhaps we should introduce the "template function developer" (that makes reusable template functions) or simply mention that any "advanced developer" can do everything a naive application developer does plus unit wrapping, (delta) function declaration and perhaps source wrapper declaration*

Application structure. Figure 1 shows the conceptual, developer-oriented architecture of a ViDeTTe application. It consists of: (a) a *unified application state (UAS)* *YannisP: See above...* , abstracting access to the data sources that the application uses (e.g., the sensor readings in our running example), (b) *template instances*, which abstract the structure of the *visual pages* that are shown in the browser (e.g., describing in our running example the data points that will be shown and the visualization libraries that will be used to display them), (c) declarative *templates*, specifying how the data in the UAS are transformed to template instances, and (d) *actions*, specifying the procedures that should be executed in response to events. The procedures may decide which template should be displayed and they may modify the UAS.

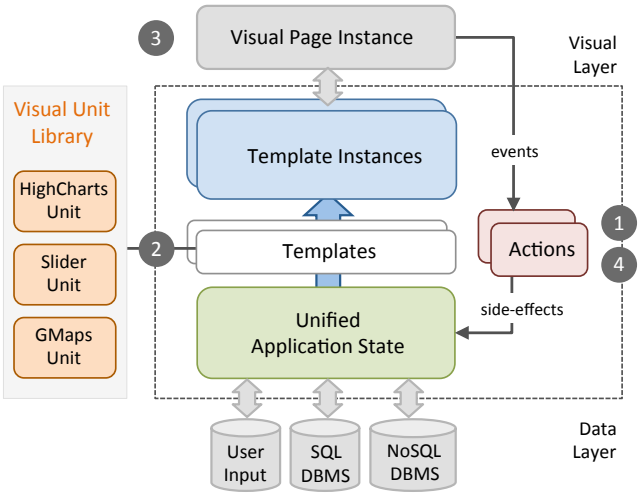

Figure 1: Programming model for an application developer. Actions, templates and UAS specifications are the pieces that are provided by the developer.

During the runtime, an application follows an *action-template cycle* in which actions activate templates that produce visual page instances. As new events occur they trigger respective actions again thus restarting this cycle. In particular, an action-template cycle consists of the following steps (denoted by numbers in Figure 1): An action is activated (step 1) and chooses the template that will be displayed. At that time the action can also cause side-effects to the application state by interacting with the UAS instance. Subsequently, ViDeTTe evaluates the chosen declarative template on the UAS (step 2), which produces an abstraction of the visual page in the form of a template instance. Then ViDeTTe automatically produces a new visual page instance from the template instance (step 3). Finally, user interface events are caused as the user interacts with the visual page instance, thus triggering actions (step 4), that lead to repetition of the action-template cycle. We next describe each of the pieces of a ViDeTTe application in detail. *Costas: It is possible for an action to not trigger a template (for instance an action could simply collect data from the page and store them locally or transmit them over-the-wire to a remote source)*

YannisP: About Figure 2a: Why the importing is on the template? The actions may not be template specific. Same for the functions.

Costas: Yes, the template needs to be changed.

YannisP: About Figure 2a: The total absence of html and/or any form of unit testing hurts the idea that we do pages.

Costas: The two units should be wrapped by an HTML unit. I

[1] http://www.highcharts.com/

```
<% template temp_view() %>
  <% import functions %>
  <% import actions %>

  <% let readings = select t.temp
                    from db.temperature as t
                    order by timestamp %>

  <% unit highcharts %>
  {
    title: { text: 'Temperature monitor' },
    series: [{
      data: [
        <% for reading in readings %>
          {
            y    : <% print reading.temp %>,
            color: <% print toHex(reading.temp, threshold) %>
          }
        <% end for %>
      ],
      lineWidth: 1
    }],
    <% event onSelection redrawSelected() %>
  }
  <% end unit %>

  <% unit slider %>
  {
    min  : 0,
    max  : 10000,
    value: <% bind threshold = 65 %>
  }
  <% end unit %>
<% end template %>
```

(a) Template `temp_view`

```
<% unit highcharts %>
{
  title: { text: 'Temperature monitor' },
  series: [{
    data: [
        { y: 55, color: '#359435' },
        { y: 57, color: '#359823' },
        { y: 56, color: '#359533' },
        { y: 53, color: '#359220' }
    ],
    lineWidth: 1
  }]
}
<% end unit %>

<% unit slider %>
{
  min : 0,
  max : 10000,
  value: 65
}
<% end unit %>
```

(b) Template instance

```
  sources : [ {
    driver   : "postgres",
    host     : "localhost",
    port     : 5432,
    aliases  : [{db : "sensorDb"}]
    username : "dbadmin"
  }]
```

(c) UAS configuration file

Figure 2: Template, template instance, and UAS configuration file for the running example

believe this was done like that to avoid an extensive description on fragmentation.

YannisP: About Figure 2a: The query suspiciously lacks any window

Costas: The application shows historic data, it doesn't show a particular window. The user however can use the chart to zoom into some time window, to explore the data.

YannisP: About Figure 2a: Query almost covers the provenance heterogeneity problem behind time stamp.

YannisP: About Figure 2b slider unit: May be good to have a non-initial value and talk about it.

3.1 Unified Application State (UAS)

Modern web applications retrieve data from a variety of sources, including backend databases (both SQL and NoSQL), browser storage (such as IndexedDB), web services and user input (through forms). ViDeTTe abstracts all data sources into a *unified application state* (in short *UAS*), which serves two purposes: First, it provides unified distributed accesses to multiple sources, in the spirit of virtual databases. Second, it efficiently observes modifications to data sources, which is an important step to ViDeTTe's incremental computation ability and a major topic of this paper. A UAS instance is a tuple of the form $\langle x_1 : v_1, \ldots, x_n : v_n \rangle$, where each x_i is a variable bound to the value v_i.

Costas: The UAS instance corresponds to the (Client-Side) Application Model

YannisP: if we abolish the UAS and go with the functions approach, this section says that the results of the functions are JSON++ TBD whether we need functions that also input JSON++. If we do, then it can be the template language itself. It only lacks recursion. At any rate it is probably journal version material.

UAS values are represented through the JSON++ data model: an extension of the (common in web applications) JSON data model with support for unordered collections (bags). This extension is crucial to support propagation of changes that happen to bags of tuples (typically produced by relational DBMSs) as we will see in Section **??**. A JSON++ value can be a scalar (i.e., string, number or boolean), tuple of attribute/value pairs, array of values (i.e., ordered collections), or bag of values (i.e., unordered collections). *Costas: Since bags and JSON tuples inherently use keys, perhaps they can be both encoded as tuples thus simplifying the data model that has to be understood by unit developers - Note, we agreed to do it this way during our last meeting.* Complex values (i.e., tuples, arrays, and bags) recursively contain other JSON++ values, thus allowing the representation of nested data. Finally, internally each value may also contain a *provenance annotation* (in short *provenance*), which is leveraged by the change propagation algorithm. Note, the application developer need not be aware of it.

The following table shows the notation used to represent the different types of JSON++ values:

JSON++ Value	Type
"string" | x | true/false	Scalar (string/number/boolean)
$\langle x_1 : v_1, \ldots, x_n : v_n \rangle$	Tuple of attribute/value pairs
$[v_1, \ldots, v_n]$	Array of values
$\{\{v_1, \ldots, v_n\}\}$	Bag of values
$\#(k_1, \ldots k_n)\ v$	Value with provenance $(k_1, ..., k_n)$

1	*template_instance*	→	*unit_instance*
2	*unit_instance*	→	**<% unit** *unit_class* **%>** *value* **<% end unit %>**
3	*value*	→	*jsonpp_value*
4		|	*unit_instance*

Figure 3: BNF Grammar for Template Instances

To associate the UAS instance with data from a particular source, the developer has to specify the type of the source (e.g., SQL database) and, when applicable, provide the parameters required for connecting to the chosen source type (e.g., host and account information for the database server). This information has to be provided in the form of a UAS configuration file. For instance, Figure 2c shows the UAS configuration file for the running example. In the case of browser resident JSON++ objects, the developer simply needs to access them via the UAS interface. *YannisP: Kosta, is the previous sentence fair? Costas: Not really, you're probably talking about accessing variables of the client-side Model. The client-side Model is simply a collection of Plain Old JS Objects. This collection of variables is encompassed by the client-side (JavaScript) VDB/Model variable. Since they are all POJO's no API is required, the developer interacts with them as she would with any other JS Object.* As we will see in Section 3.3, the template may also introduce new variables to the UAS. Some of these variables are associated with derived data (i.e., they are tantamount to database views) and some other ones are associated with page input. *YannisK: We should give more details on how the configuration file affects the UAS instance. We could extend the UAS instance to contain non-JSON++ values corresponding to sources (e.g., the instance may contain a variable* db *corresponding to a relational database) but this breaks the nice abstraction that we want to create. Any ideas/suggestions? YannisP: Don't we need to extend the JSON++ with arbitrary enriched types? Is string, number and boolean enough?*

3.2 Template Instance

YannisK: Double-check line numbers for all figures

The main challenge at the visual layer is the variety of visualization libraries that may be used in a web application, along with the fact that tedious low-level programming is required to perform incremental rendering with the APIs provided by those libraries. Modern web applications commonly include a huge variety of visual components, such as charts, maps, and sliders, each with their own APIs.

Visual Units *YannisP: Header "Visual Units"*

To shield developers from low-level programming, ViDeTTe abstracts out each visual component as a ViDeTTe *visual unit* (or simply *unit*). In the eyes of an application developer a visual unit is a black box that takes as input a JSON++ value *Costas: I believe we mean JSON value here, I don't see why the application developer should learn about bags and why they are useful* containing the data to be visualized and parameters potentially supported by the visualization and uses the API of the underlying visualization library to render the visual component. As such a particular instantiation of the unit (denoted as *visual unit instance* or simply *unit instance*) can be described as **<% unit** U **%>** v **<% end unit %>**, where U the type of the visual unit and v the JSON value corresponding to the input of the unit. For instance, lines 1-14 of Figure 2b show a unit instance of type `highcharts`. The unit instance specifies both the data that should be shown on the chart (as an array of tuples assigned to the `data` attribute) and the values of the parameters accepted by the unit (such as the chart's title).

YannisP: better to have HTML in the example but do not spend too much time on it in the abstract discussion

As we will discuss in Section **??**, visual units are generated by visual unit developers who wrap popular visualization libraries (such as Google Maps, HighCharts, etc) and make them available to application developers through ViDeTTe's visual unit library. In addition to the units provided by unit developers, ViDeTTe also provides a special unit of type *HTML*. In contrast to other units, which expect JSON values, an HTML unit takes as input a sequence of HTML tags, allowing developers to easily create simple HTML content. For ease of exposition we focus in this paper on JSON units. Our results can be easily extended to the HTML unit. *Costas: So far we've been talking about JSON++ if we keep that, we should make it clear why in this paragraph we only talk about JSON and HTML YannisK: Removed HTML unit encompassing the two JSON units from our running example*

An entire visual page instance (potentially containing multiple visual components) can be described through a composition of unit instances. We refer to this logical description of a visual page as *template instance*, as it is not written directly by the developer but is instead the result of instantiating a *template*, as we will see next. Figure 2b shows a template instance for our running example. It consists of two unit instances; a highcharts unit instance (lines 1-14) and a slider unit instance (lines 16-22). Note that a unit instance may recursively contain nested unit instances *YannisK: Example? YannisK: What does it mean for a unit to be nested within another unit? Can we nest arbitrary units or only units that are designed as such by the unit developer? YannisP: Arbitrary. Costas: The child unit is Arbitrary. However, a child unit can only be attached at predefined locations of the parent unit (i.e When the parent unit is google maps, a child unit can only be attached under the infowindow location since that's the only attachment point that is supported by the Google Maps API)* Figure 3 shows the grammar for template instances. Finally, to simplify the architecture, template instances are internally represented as JSON++ values. This is achieved by encoding a list of unit instances **<% unit** U_1 **%>** v_1 **<% end unit %>** $\ldots$ **<% unit** U_n **%>** v_n **<% end unit %>** as a JSON++ tuple $\{U'_1: \ v_1, \ \ldots, \ U'_n: \ v_n\}$, where U'_i is a unique identifier assigned by ViDeTTe to the i-th unit instance. *YannisK: OK or too cryptic?*

YannisP: We need reserved keywords for the attribute names that correspond to units. Eg, _myunit Costas: How can we have a predifined set of reserved keywords given that there's an arbitrary number of possible units each with a unique name?

3.3 Template

YannisP: About Figure 4: the syntax allows a single top level unit, which is probably correct but not what the example does. Is the top level unit always html?

```
1  template   →  <% template template_name (param_list) %>
                    let*
                    unit
                  <% end template %>
2  param_list →  ( var_name (, var_name)* )?
3  unit       →  <% unit unit_class %>
                    value
                  <% end unit %>
4  value      →  jsonpp_value
5             |  unit
6             |  print
7             |  [ for ]
8             |  < for >
9             |  if
10            |  bind
11            |  { event*
                    ("string" : value
                    (, "string" : value)* )? }
12 let        →  <% let var_name = expr %>
13 print      →  <% print expr %>
14 for        →  <% for var_name in expr %>
                    let*
                    value
                  <% end for %>
14 if         →  <% if expr %>
                    value
                    (<% elif expr %>
                    value)*
                    (<% else %>
                    value)?
                  <% end if %>
15 bind       →  <% bind var_name = expr %>
16 event      →  <% event event_name action_name %>
17 expr       →  js_expression
18            |  source_expression
19            |  json_path
```

Figure 4: BNF Grammar for Templates

YannisK: To do: Modify template BNF grammar to make sure that we can have directives appear inside JSON values *Templates* are declarative specifications describing the template instances as a function of UAS data. A template specifies this function through a set of *template directives*, so that it provides syntax similar to well-known template languages, while it is essentially an expression of a functional programming language without recursion. A template also specifies collecting data from the page and catching events *Costas: perhaps we mean linking event with corresponding actions?* .

There are five template directives: **<% print %>**, **<% for %>**, **<% let %>**, **<% bind %>**, and **<% event %>**. These are used to describe computation, define variables, set up data collection and specify events. We next describe each of them in detail.

YannisP: From an expressiveness point of view, let does not add anything. The interesting consideration is whether it has a meaning about IVM: materialized vs virtual intermediate result.

Defining variables. A template may define variables that are added to the UAS instance so that they can be used in subsequent computation. Variable definition is facilitated by the **let** directive.

The **<% let** x **=** E **%>** directive defines variable x in the UAS, and assigns to x the result of evaluating the expression E. The expression E can be a JavaScript expression, a source-specific language (such as a SQL query in the case of relational database sources) or a JSON++ path. For example, the template of our running example (lines 6-8) employs a **let** directive to create a variable `readings` containing all temperature readings (retrieved from a relational DBMS through a SQL query).

Reporting syntax and semantics. Computation in a template is specified using the **print** and **for** directives. The **<% for** x **in** E **%>** B **<% end for %>** directive specifies that variable x iterates over the result of the expression E. In each iteration, the body B of the **for** loop is instantiated. For example, the template of our running example (lines 14-19) uses a **for** directive to iterate over the sensor readings (stored in the `readings` variable). For each reading, it generates a new JSON tuple of the form `{y:..., color:...}`, which is the data format expected by the HighCharts unit for each data point.

The **<% print** E **%>** directive instantiates the result of expression E. For example, the template of the running example uses two **print** directives to generate the values of the `y` and `color` attributes of each JSON tuple produced by the **for** directive. *YannisP: The part between the "{}" could be removed* {The value of `y` is created by printing the value of the `reading` variable (line 16), while the value of `color` is generated by calling the `toCSS` function, which takes as input the current reading and the normal temperature (set through the slider) and produces a CSS color code according to the coloring schema explained above (line 17).}

Collecting data. In addition to specifying how to compute a template instance, the template's **bind** directive allows the developer to specify how data are collected from user input on the page.

The **<% bind** x $=$ E **%>** directive specifies that the template instance attribute value in whose position the directive appears will be assigned to variable x. This allows UAS variables to become bound to input received by visual units. *YannisP: If UAS has not been defined we need to say what is the JavaScript target.* *Costas: We should define a Model/VDB variable, that encompasses every client-side variable used in an application. The target in this case would be the Model.x variable* For instance, the template of our running example (line 30) uses a **bind** directive to assign to variable `threshold` the current value of the slider (which is returned by the slider unit as the value of the `value` variable). The **bind** directive also allows the developer to specify an expression E, whose value will be assigned to the variable when the template is first instantiated. For example, the slider of our running example is initialized with the value 65.

Specifying event handlers. Finally, a template allows also the invocation of actions in response to events. The **<% event** e a **%>** directive specifies that whenever event e of the enclosing unit instance occurs, action a is executed. For instance, *YannisK: Costa, please add event to template and add description here.*

Costas:
Finally, each template may also accept parameters that are passed to it by its caller by value. When the template is instantiated, ViDeTTe creates for each parameter an identical-named UAS variable and then sequentially scans the template instantiating its directives.

3.4 Actions

An action is a Javascript procedure that is invoked in response to events and can also read and write to the UAS, cause side-effects in external systems, such as charge

a credit card through a REST service, and specify which template should be displayed next. In visualization applications, such as the running example, actions are very simple, since they only specify the next template to be instantiated. For example, see *YannisP: Kosta, please put a super simple action. Costas: Will do after we replace the running example Costas: I think it makes sense to introduce a system-provided action (named redisplay_template()), that simply redisplays the current template, when an event occurs. This would allow eager application view maintenance without requiring extra imperative code from the app developer.*

3.5 Application Specification Summary

YannisP: Could be shortened to one sentence at start To specify an application, the developer thus has to provide the following components: (a) a specification of the UAS describing how data from different sources are mapped into variables, (b) a set of templates describing how to transform the UAS to visual pages, and (c) a set of actions, describing how the application should react to events. *Costas: (a) assumes that ViDeTTe is used as a full stack framework, perhaps we should also mention that if the developer wishes to only use ViDeTTe as a client-side framework, instead of (a) she has to simply provide the JSON objects that constitute the model. Additionally, when we describe the source wrappers it might make sense to introduce a REST/Websocket wrapper that works with 3rd party services. This wrapper would provide diffing capabilities without requiring the app developer to manually diff the model (this diffing would most likely have to take place on the client)*

4. DISCUSSION & CONCLUSION

5. REFERENCES

[1] Angular leaflet directive. http://tombatossals.github.io/angular-leaflet-directive/.

[2] AngularJS. http://angularjs.org/.

[3] Apache mahout. https://mahout.apache.org/.

[4] A. Bhardwaj, A. Deshpande, A. J. Elmore, D. Karger, S. Madden, A. Parameswaran, H. Subramanyam, E. Wu, and R. Zhang. Collaborative data analytics with datahub. *Proceedings of the VLDB Endowment*, 8(12):1916–1919, 2015.

[5] J. A. Blakeley, N. Coburn, and P.-Å. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. Database Syst.*, 14(3):369–400, 1989.

[6] J. A. Blakeley, P.-Å. Larson, and F. W. Tompa. Efficiently updating materialized views. In *ACM SIGMOD*, pages 61–71, 1986.

[7] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics*, 15(6):1121–1128, 2009.

[8] Catel. http://www.catelproject.com/.

[9] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *VLDB*, 1991.

[10] Y.-J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, 1992.

[11] R. Chirkova and J. Yang. Materialized views. *Foundations and Trends in Databases*, 4(4):295–405, 2012.

[12] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: interactive analytics through pen and touch. *Proceedings of the VLDB Endowment*, 8(12):2024–2027, 2015.

[13] D3: Data-driven documents. https://d3js.org/.

[14] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. In *Workshop on Algorithms and Data Structures*, pages 265–276. Springer, 1993.

[15] M. Derthick, J. Kolojejchick, and S. F. Roth. An interactive visual query environment for exploring data. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 189–198. ACM, 1997.

[16] Ember. http://emberjs.com/.

[17] M. F. Fernández, D. Florescu, A. Y. Levy, and D. Suciu. Declarative specification of web sites with Strudel. *VLDB J.*, 9(1):38–55, 2000.

[18] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251–281, 2000.

[19] Y. Fu, K. Kowalczykowski, K. W. Ong, Y. Papakonstantinou, and K. K. Zhao. Ajax-based report pages as incrementally rendered views. In *ACM SIGMOD*, 2010.

[20] Y. Fu, K. W. Ong, Y. Papakonstantinou, and M. Petropoulos. The sql-based all-declarative forward web application development framework. In *CIDR*, pages 69–78, 2011.

[21] ggvis: Interactive grammar of graphics for r. http://ggvis.rstudio.com/.

[22] Google maps api, 2009. http://code.google.com/apis/maps/.

[23] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[24] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *ACM SIGMOD*, pages 157–166, 1993.

[25] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[26] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The madlib analytics library: Or mad skills, the sql. *Proc. VLDB Endow.*, 5(12):1700–1711, Aug. 2012.

[27] highcharts. http://www.highcharts.com/.

[28] highcharts-ng. https://github.com/pablojim/highcharts-ng/.

[29] N. Kamat, E. Wu, and A. Nandi. Trendquery: A system for interactive exploration of trends. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 12:1–12:4, New York, NY, USA, 2016. ACM.

[30] Y. Katsis, K. W. Ong, Y. Papakonstantinou, and K. K. Zhao. Utilizing ids to accelerate incremental view maintenance. In *ACM SIGMOD Conference*, pages 1985–2000, 2015.

[31] Knockout. http://knockoutjs.com/.

[32] C. Koch, Y. Ahmad, O. Kennedy, M. Nikolic, A. Nötzli, D. Lupei, and A. Shaikhha. Dbtoaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB J.*, 23(2):253–278, 2014.

[33] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, 9(12):948–959, Aug. 2016.

[34] Leafletjs. http://leafletjs.com/.

[35] E. Liarou and S. Idreos. dbtouch in action database kernels for touch-based data exploration. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1262–1265, 2014.

[36] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. Devise: integrated querying and visual exploration of large datasets. In *ACM SIGMOD Record*, volume 26, pages 301–312. ACM, 1997.

[37] Mllib. http://spark.apache.org/mllib/.

[38] Mvvm light. http://www.mvvmlight.net/.

[39] Mvvmcross. https://mvvmcross.com/.

[40] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Trans. Knowl. Data Eng.*, 3(3):337–341, 1991.

[41] React. https://facebook.github.io/react/.

[42] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2017, 2015.

[43] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2016.

[44] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran. zenvisage: Effortless visual data exploration. *CoRR*, abs/1604.03583, 2016.

[45] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[46] Vega, a visualization grammar. http://trifacta.github.io/.

[47] H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer Science & Business Media, 2009.

[48] L. Wilkinson. *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[49] Y. Wu, J. M. Hellerstein, and E. Wu. A devil-ish approach to inconsistency in interactive visualizations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 15:1–15:6, New York, NY, USA, 2016. ACM.

[50] F. Yang, J. Shanmugasundaram, M. Riedewald, and J. Gehrke. Hilda: A high-level language for data-driven web applications. In *ICDE*, page 32, 2006.

[51] K. Zoumpatianos, S. Idreos, and T. Palpanas. Rinse: interactive data series exploration with ads+. *Proceedings of the VLDB Endowment*, 8(12):1912–1915, 2015.