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.

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.* Sec-

ond 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 **??** 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 3 discuss related work and conclude the paper, respectively.

3. DISCUSSION & CONCLUSION

4. REFERENCES