Leveraging Traits for Highly Interactive Computational Tools in Jupyter

Nicole Brewer
Research Computing
Purdue University
West Lafayette, USA
brewer36@purdue.edu

Abstract

We describe our experience designing and implementing a highly interactive, online computational tool in Jupyter Notebooks. The lessons learned and subsequent design choices can be applied to similar tools in various domains. This tool, Superpower [1], is a graphical interface for a set of functions designed to help users to perform power analysis on their study design in psychology. The supported statistical functions are computationally non-trivial in that each power analysis function requires many multidimensional parameters that unavoidably make use of lists or numpy arrays. It is also highly interactive such that user manipulation of any single value or element may require a cascade of updates to others.

We chose an MVC design pattern to separate computational logic from the view. We used ipywidgets for view components such as selection boxes and numeric parameter settings, as it is the de facto standard for interactive components in Jupyter. Additionally, we used Interactive Matplotlib (ipympl) to create features such as a click and drag bar chart. ipywidgets is built on a traits library called traitlets [2]. Traits are object attributes that are designed to create and send a notification upon an event or a change in value. Correspondingly, event listeners are functions or methods designed to respond to these notifications. For example, the *IntText* widget is frequently used to manipulate the number of subjects in the user interface (UI). An *observe* function can be used to update the corresponding value in the model upon change of this number in the view (UI).

However, traitlets proved insufficient for our use case due to the lack of support for "container traits". Container traits are important in our models because of the need to monitor changes in multi-dimensional lists and arrays. For example, a grid of IntText and FloatText widgets were used to represent a set of related multidimensional parameters represented in the model as numpy arrays. Changing any value in this grid would require updates to others. Therefore, it was imperative to use lists observable at the item level. Consequently, we used Enthought's traits library [3] to implement our models; traitlets is, in fact, a lightweight implementation of traits that is missing support for "container traits (list, dict, tuple) that can trigger notification if their contents change" [2].

traits is syntactically similar to traitlets, with a few additional complexities that may make it more difficult to learn for someone not already familiar with a traits library; for example, traits does not raise errors that occur in observe functions, making debugging more difficult. Furthermore, unlike traitlets, traits does not support "links" that simplify the setup of bi-directional mapping between the view and the model. Traditional ipywidgets is likely to be sufficient for tools that do not have multilevel data structures or heavily interactive components. For tools with these complexities, we recommend the use of the traits to monitor changes in a model class. Our design choices can serve as a guide for developers looking to address complex interactivity in a Jupyter-based gateway tool.

Keywords—gateways, traits, Jupyter Notebook, interactive tools

References

[1] Nicole Brewer; Rob Campbell; Jaewoo Shin; Lan Zhao; Erin P Hennes; Sean P Lane (2021), "SuperPower," https://mygeohub.org/resources/superpower

[2] M. Dickinson, I. Tziakos, K. Choi, C. Webster (2021) traits [Software] Enthough, Inc. Austin, TX. Available: https://github.com/enthought/traits

[3] M. Bussonnier. (2021) Traitlets [Software]. The IPython Development Team Available: https://github.com/ipython/traitlets/blob/main/traitlets/traitlets.py