2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) | 978-1-6654-4214-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/VL/HCC53370.2022.9832910

Chaldene: Towards Visual Programming Image Processing in Jupyter Notebooks

Fei Chen* Philipp Slusallek*† Martin Müller† Tim Dahmen*
*German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
†Saarland University, Saarbrücken, Germany
{fei.chen, philipp.slusallek, tim.dahmen}@dfki.de
martin.mueller1@uni-saarland.de

***Abstract*—Jupyter Notebook [1] is an open source, interactive computing platform widely used in the scientific computing and artificial intelligence community [2], [3], [4], [5]. The popularity of the platform is a consequence of the generated single notebook document combining source code, markdown, and visualizations (Fig.1). This makes the platform ideal for tasks such as data analysis and scientific image processing, where repeatability and transparency of analysis tasks are just as important as functionality and performance. However, the obligatory use of code is an obstacle to acceptance of the platform in scientific communities where programming is not generally taught in the curriculum. Consequently, many experimental communities rely on manual image processing using graphical user interfaces [6], [7], [8]. The obvious disadvantages are the lack of repeatability, transparency, and precision in image processing and data analysis tasks. To solve these issues, we propose to extend Jupyter Notebook with visual programming cells. In each visual programming cell, users can create the program by assembling graphical nodes that represent computational instructions, and the textual program is automatically generated and executed by the environment. Cells will support version control aware serialization and deserialization. The core innovation of our proposed work lies in a change of workflow and the adaption of a jupyter-based workflow in experimental communities that have no culture of working with source code. The system can be adapted to multiple applications and domains by integrating new node types. We hereby present an early version of the system and provide one use case from microscopy image processing to demonstrate the integration of existing non-Python software.**

***Index Terms*—VPL, visual programming language, graphical programming, Jupyter Notebook, image processing**

I. Design and Implementation

A. Visual Programming Language

Our visual Programming language follows the dataflow paradigm and is partly implemented using the litegraph.js library [9]. The programs are modeled as directed acyclic graphs (DAG) of data, flowing between vertices. In DAG, a vertex represents a node in visual programming, and a directed edge connecting two vertices signifies the data flow from one to the other. Therefore, our visual programming contains five elements: nodes, connectors, graphs, canvas, and nodes box.

Nodes are the basic building blocks in the visual language and they represent data structures, function calls, class instantiations, or other statement representations in the textual programming language. To define a node, we declare two parts of information: one part is the description of the visual signature such as title, input, output, property. For input arguments and output values, the type could be provided for type matching. The other part is the source code this node represents and the import library. The **connector** is a special node and a connector linking the input port with the output port from two separate nodes represents the data flow from input to output. The **graph** is built by linking the nodes and can be visualized on the canvas. The **canvas** is the graphical panel that contains visual programming language nodes. It organizes the graphical representation of languages elements spatially and visualizes the graph. Additionally, it allows users to create nodes from the node box as well as navigate around the nodes by panning and zooming. The **node box** is the container holding all the node classes for use and allows users to select one of the nodes to create an instance on the canvas. The node class can be added to the node box by registration.

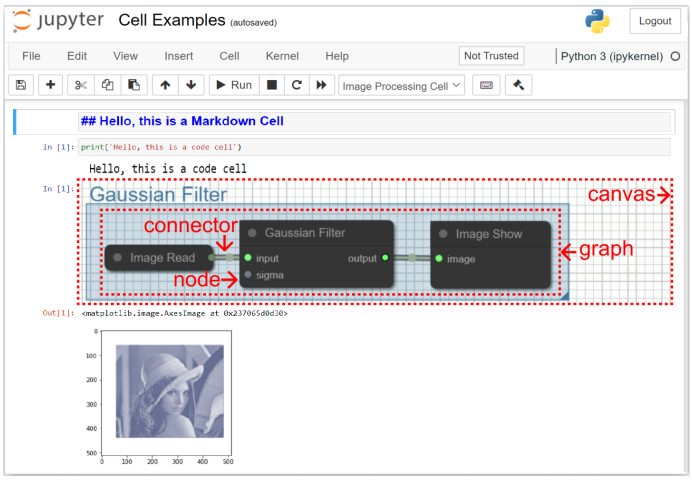

Fig. 1. Markdown cell, Code cell, Visual Programming cell from top to bottom in a single document in Jupyter Notebook platform. Node instance, connector, graph and canvas examples are shown in Visual Programming cell.

B. Integrate Visual Programming in Jupyter Notebooks

The document in the Jupyter Notebook user interface consists of a sequence of cells (See Fig.1). A cell is a block of input fields and allows users to edit and run the command

978-1-6654-4214-5/22/$31.00 ©2022 IEEE

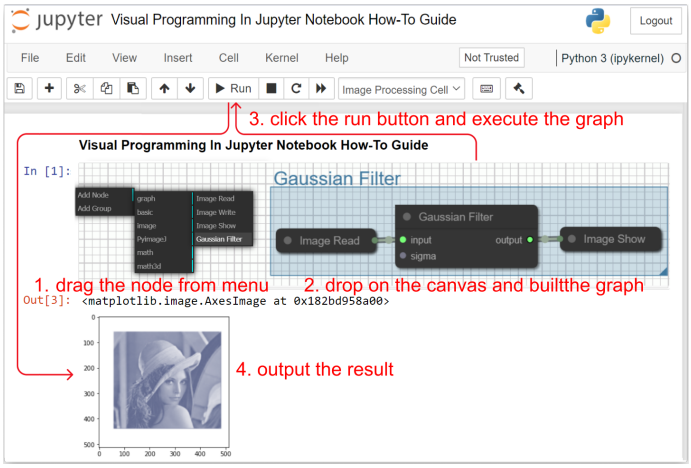

Fig. 2. The user interaction of visual programming in Jupyter Notebooks.

inside the cell. To integrate the visual programming language without inordinate amounts of change to the basic Jupyter Notebook user interface, we created a new type cell, named visual programming cell. (See bottom cell in Fig.1). This cell supports serialization and deserialization and is easy for sharing. Furthermore, it is compatible with basic functionalities in the Jupyter Notebook including cell creation, manipulation, and cell execution.

Cell Creation and Manipulation. The Jupyter Notebook exhibits a modal user interface including edit mode and command mode. The keyboard behaves differently, depending on which mode the Notebook is in and our visual programming cell fully supports these two modes. If the cell is in edit mode, the user can drag and drop nodes and build a computation graph on the canvas. Whereas if the cell is in command mode, the cell is edited as a whole: there is no typing into individual cells, the whole of the cell can be created and manipulated like cut, copied, pasted, deleted, moved up and down, as well as affected with operations from the menu bar, toolbar or shortcuts.

Cell Execution. Jupyter Notebook is a web-based application and adopts a client-server infrastructure. The client is the notebook user interface which allows users to edit and send code to the web server via HTTP requests, after which the code is passed to the computational engine named kernel and executed. Therefore, the textual source code needs to be translated from our visual language. The program in visual programming language is modeled as an directed acyclic graph. Actually, the translation from visual programming language to text-based language is the topological sorting problem of DAG. The Kahn's algorithm is implemented for topological sorting.

C. User Interaction

In the Section A, we describe the definition of visual programming language, especially new visual nodes types declaration from textual code. When building the visual program, the user directly drag the nodes instance from the node box and drop them on the canvas and then connect these nodes. An example is shown in Fig.2.

II. Use Case

ImageJ/Fiji [6] is an open-source image processing software in wide use, for example, in the microscopy community for image processing tasks. ImageJ is written in Java and provides a graphical user interface for interactive use. The key strength of the software package is the existence of a large number of plugins for various tasks in scientific image processing. To utilize the rich image processing ecosystem of ImageJ/Fiji in Jupyter Notebook, we integrate ImageJ/Fiji into our visual language by creating visual node types that represent a set of python wrapper functions and these nodes can be connected to perform several image processing tasks. An image processing use case utilizing the visual language is a real-world task from the scanning electron microscopy (SEM) domain and is presented in Fig.3.

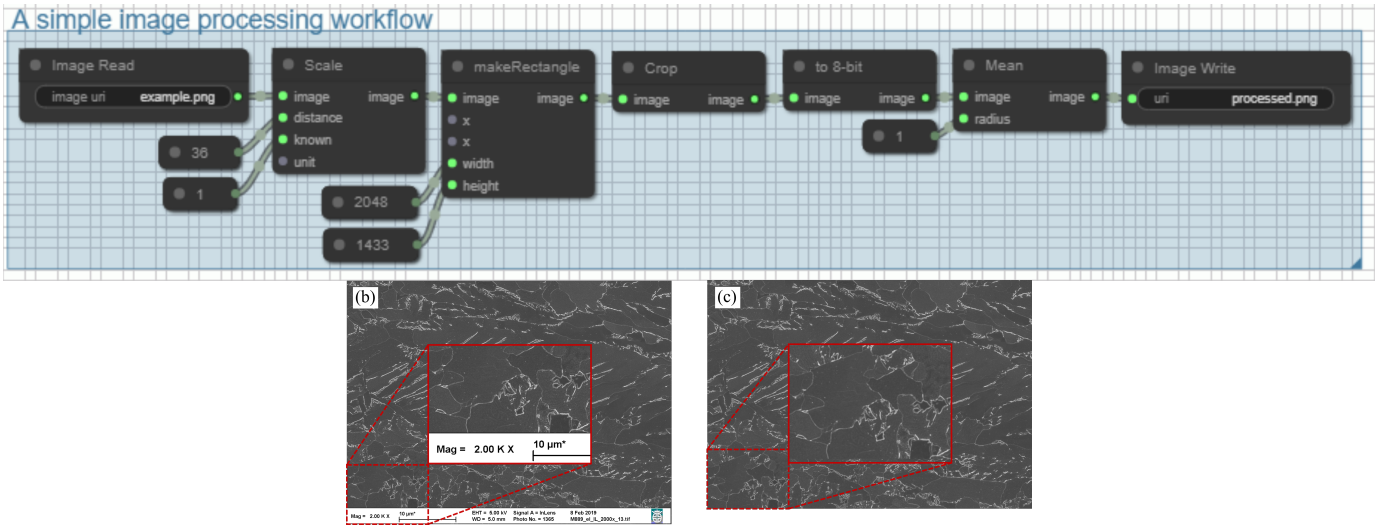

Fig. 3. (a) Workflow for a simple data processing task in electron microscopy. (b) Input of the workflow. (c) Output of the workflow.

ACKNOWLEDGMENT

This work was supported by German Research Foundation (DFG) under the Grant NFDI4MatWerk (NFDI).

REFERENCES

[1] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C. & Team, J. Jupyter Notebooks - a publishing format for reproducible computational workflows, Positioning And Power. *In Academic Publishing: Players, Agents And Agendas*. pp. 87-90, 2016.

[2] Milicchio, F., Rose, R., Bian, J., Min, J. & Prosperi, M. Visual programming for next-generation sequencing data analytics. *BioData Mining*. Vol. **9**, pp. 1-17, 2016.

[3] Janssen, J., Surendralal, S., Lysogorskiy, Y., Todorova, M., Hickel, T., Drautz, R. & Neugebauer, J. pyiron: An integrated development environment for computational materials science. *Computational Materials Science*. Vol. **163**, pp. 24-36, 2019.

[4] Dehaye, P., Iancu, M., Kohlhase, M., Konovalov, A., Lelièvre, S., Müller, D., Pfeiffer, M., Rabe, F., Thiéry, N. & Wiesing, T. Interoperability in the OpenDreamKit project: the math-in-the-middle approach. *International Conference On Intelligent Computer Mathematics*. pp. 117-131, 2016.

[5] D. Ozturk, A. Chaudhary, P. Votava and C. Kotfila, GeoNotebook: Browser based interactive analysis and visualization workflow for very large climate and geospatial datasets. *AGU Fall Meeting Abstracts*. Vol. **2016**, pp. IN53A-1876, 2016.

[6] Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B. & Others. Fiji: an open-source platform for biological-image analysis. *Nature Methods*. Vol. **9**, pp. 676-682, 2012.

[7] The GIMP Development Team GIMP. https://www.gimp.org 2019,6,12.

[8] Dahmen, T., Marsalek, L., Marniok, N., Turoňová, B., Bogachev, S., Trampert, P., Nickels, S. & Slusallek, P. The Ettention software package. *Ultramicroscopy*. Vol. **161**, pp. 110-118, 2016.

[9] litegraph.js. https://github.com/jagenjo/litegraph.js 2022.