NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance

Lucas A. M. C. Carvalho[1], Regina Wang[2], Yolanda Gil[2], Daniel Garijo[2]
[1]University of Campinas, Institute of Computing, Campinas, SP, Brazil
[2]University of Southern California, Information Sciences Institute, Marina del Rey, CA, U.S.A

lucas.carvalho@ic.unicamp.br, gil@isi.edu, dgarijo@isi.edu

ABSTRACT

Interactive notebooks are increasingly popular among scientists to expose computational methods and share their results. However, it is often challenging to track their dataflow, and therefore the provenance of their results. This paper presents an approach to convert notebooks into scientific workflows that capture explicitly the dataflow across software components and facilitate tracking provenance of new results. In our approach, users should first write notebooks according to a set of guidelines that we have designed, and then use an automated tool to generate workflow descriptions from the modified notebooks. Our approach is implemented in NiW (Notebooks into Workflows), and we demonstrate its use by generating workflows with third-party notebooks. The resulting workflow descriptions have explicit dataflow, which facilitates tracking provenance of new results, comparison of workflows, and sub-workflow mining. Our guidelines can also be used to improve understandability of notebooks by making the dataflow more explicit.

CCS CONCEPTS

• Information systems → Artificial intelligence; Knowledge representation and reasoning

KEYWORDS

Scientific Workflows; Workflow Design; Electronic Notebooks.

1 INTRODUCTION

Interactive notebooks have become very popular in science to capture computational experiments [14]. These notebooks include code, visualizations, and explanations, and can be easily shared and re-run.

As scientists carry out their research, they may need to compare the results and methods of different experiments. This involves comparing final results, comparing intermediate results, comparing steps of the method, and comparing parameter values. Since notebooks contain raw code, it can be hard to understand how new results are generated, as well as to compare notebooks. In contrast, workflows offer modular components to run code, and have an explicit dataflow. This can facilitate provenance capture, as well as automated mining of reusable workflow fragments [4]. Workflows also facilitates understanding and performing comparisons, particularly for non-programmers [6].

This paper presents an approach for converting notebooks into workflow descriptions by mapping various aspects of notebook cells into workflow components and dataflow. Our approach is implemented in NiW, a prototype tool to convert Jupyter Notebooks [1] into WINGS workflows [7]. Based on the assumptions of our approach, we propose a set of guidelines for designing notebooks that facilitate the conversion and can be used by notebook developers to improve the understandability of their notebooks.

2 DATAFLOW AND PROVENANCE IN WORKFLOWS AND NOTEBOOKS

This section discusses general issues for identifying dataflow and tracking provenance in notebooks, compared with simple dataflow in workflows. Our work to date has focused on mapping Jupyter Notebooks to workflow descriptions that can be used in the WINGS workflow system [6], but many of the issues will be common for other notebook and workflow systems.

2.1 Dataflow in Workflows

Workflows capture explicitly the dataflow across software components. We describe here a very simple dataflow representation and workflow structure that we assume in the rest of the paper. This approach is used in several workflow systems, including WINGS [6], Pegasus/Condor [4], and Apache Taverna [12].

Each software *component* (or *step*) of a workflow may have multiple datasets as *inputs*, multiple datasets as *outputs*, and multiple *parameters*, which are provided as simple numeric or Boolean values. A dataset generated by a component can be input to another component, thereby indicating the flow of data (i.e., the dataflow) from a component to another.

A workflow management system can run a workflow if the software components can be executed and the respective input datasets and parameter values are provided. Because the dataflow is explicitly captured in the workflow, the system can record the provenance of each new dataset generated by the workflow.

[1] http://jupyter.org

2017 Workshop on Capturing Scientific Knowledge (SciKnow), held in conjunction with the ACM International Conference on Knowledge Capture (K-CAP), December 4, 2017, Austin, TX.

The dataflow of a workflow is often shown as a graph. Workflows can be compared as graphs. Indeed, graph algorithms have been used to query workflow repositories [1], and to mine workflow repositories to find commonly occurring sub-workflows [5]. Visual user interfaces that show the dataflow graph in a workflow are easy to use for non-programmers [8].

2.2 Overview of Computational Notebooks

Notebooks aggregate text and code, grouped into a sequence of containers or *cells*. Cells can be *code* cells, *markdown* cells, and *raw* cells. Code cells have running code usually with one programming language such as Python, R, Java, etc. Code cells are the heart of notebooks. Markdown cells are comments and documentation, so users can add graphics, formatting, etc. These cells are not linked with any other cells and run without interfering other cells. Raw cells display raw text without any conversions, and are much less used. Unlike raw and markdown cells, code cells are linked with each other when the notebook runs, though code cells run like a single unit of code. Thus, when a code cell performs any activity (e.g. initialize a variable), the next code cell to be run carries it on as though there were no other cells in between. Outputs are shown in the notebook when a cell code includes a plot or print statement.

Jupyter Notebook [2] is one of the most popular notebook platforms. They were originally named IPython Notebooks since they are primarily used with Python, but expanded with *kernels* for several other programming languages other than Python. A kernel is a program that runs the notebook's code.

2.3 Understanding Dataflow of Notebooks

We analyzed a diversity of Jupyter Notebooks to understand their dataflow and the provenance of their results. Common problems that we found include:

1. **Processing:** A user may not have a clear understanding of what are the main processing cells of a notebook. For example, cells for assigning values to variables or importing libraries are not processing units and should be placed in the same cell that uses those variables.
2. **Dataflow:** Input file names are either implicit in the code or defined as parameters through method calls in previous cells. This represents implicit dependencies between cells, and therefore make it difficult to understand the dataflow among them. In addition, users may also have difficulty figuring out what files were generated by a given cell.
3. **Inputs:** Input files may contain pointers to other files that are opened and used as inputs inside of a cell. This creates an implicit dependency that is difficult to detect.
4. **Outputs:** When a cell does not have an explicit output it is very difficult to understand what kind of process that cell performed. A cell may overwrite a file with the same name generated by other cells, so it can be hard to track the provenance of newly generated files. Notebooks can generate visualizations, but those do not necessarily generate output files.
5. **Files:** It is very difficult to understand how the files in notebook folders correspond to the cells that used or generated the files.
6. **Data:** Some notebooks are available in repositories without any test data. Therefore, this makes it hard to understand the expected data format of the input files and the outputs generated by the notebook. It also makes it hard to rerun the notebooks.

In summary, many problems arise in trying to understand what is the dataflow across the cells of a notebook and how they use or generate notebook files. This makes it very hard to figure out the provenance of any results. This also makes it hard to understand a notebook, as well as comparing different notebooks.

3 MAPPING NOTEBOOKS TO WORKFLOWS

When mapping notebooks into workflows, many issues must be addressed. We discuss here those issues, and our approach to tackle them. We start with general issues. After that, we focus on issues specific to Jupyter Notebooks and Python, since that has been the focus of our work so far. Then we discuss issues specific to the WINGS workflow system, which is the target of our mappings.

3. 1 Components

3.1.1 Executable Code

Differences: Each component in a workflow must have some running code within it. In notebooks, cells may contain solely value assignment to variables, function declarations or library imports, or documentation, which are not executable code by themselves and cannot be easily mapped into a workflow component. Another difference is that in notebooks even though code is split cell by cell, most of the splits only exist to benefit human readability and do not actually affect the code itself. Cells are used just to modularize the code. As a result, different users may break up the code at different places and it will not matter much if the sequential order to run these cells is preserved when there are any data dependencies between them. In contrast, in workflows the code is split into components which are isolated from one another and that individually carry out a meaningful function. An example of this is that if a variable is created in a component and used in a second component, the latter will not have access to this variable unless it is generated by the former as an output and explicitly consumed as an input by the second component.

Approach: Each notebook cell with running code will be mapped to a workflow component. If a notebook cell does not have running code and only has library imports or method declarations, it will become part of a cell that requires that information.

[2] http://jupyter.org/

3.1.2 Libraries and Methods

Differences: A notebook only needs to import a library or state a method once. Since workflows are componentized, the imports and method declarations need to be done in each workflow component that uses them.

Approach: Every library used in the notebook will be imported into all workflow components created. A method will be included in a component only if the method is used in it.

3.1.3 Open Files

Differences: A notebook can open a file and use it in any subsequent cell. In a workflow, a file can be used only inside the component that has that file as input.

Approach: If a file is opened and used across many cells, those cells will be merged into a single component. Note that an alternative approach might be to create separate components for each of the cells and open and close the file in each of those components, but this would result in inefficiencies if the data is written to files and read from files too many times.

3.1.4 Markdown Cells

Differences: Markdown cells in notebooks do not contain running code, but need to be included in the workflow as documentation so that the information that they contain is not lost. Workflow components can have documentation. There can be more markdown cells than code cells. In addition, the relationship between markdown cells and code cell is not explicit. A markdown cell may be related to either its previous or its subsequent cell.

Approach: Since the relationship between markdown cells and code cells is unknown, the assignment is made in the following way: a markdown cells' information will be attached to the documentation of the component created for its subsequent code cell.

3.1.4 Component Naming

Differences: In a notebook, a cell does not have a name. In a workflow, a component has a name that generally describes the function of the component is in the workflow.

Approach: A name will be generated for each component of the workflow, starting with "Component" followed by the ordering number from the cell (e.g., Component1, Component2 and Component3).

3.2 Data and Parameters

3.2.1 Parameters

Differences: In notebooks, method parameters are set through program variables. In workflows, parameters are inputs to components and provided by users. In workflows, if parameters are coming from other components, these parameters must be passed explicitly through a file.

Approach: Variables of primitive types (i.e., Boolean, string, integer, float, date, etc.) that are given constant values in notebooks will be mapped to parameters in workflows, and they will be given the name that was used in the notebook.

3.2.2 Input Files

Differences: A notebook may be given input data once at the beginning, and there is no need to pass data through files from cell to cell. In a workflow, a component must output a data file that is then an input to another component.

Approach: A data file will be explicitly generated from the notebook code in order to be passed to another component in the workflow. Code will be added to the component that generates the data so that the resulting data is written into a file that can be passed to the next component. Although this may not be an efficient approach, it facilitates provenance tracking which is very important in scientific analysis.

3.2.3 Output Files

Differences: In a notebook, cells can write results directly into files in the local file system. Other cells show results in visualizations. In contrast, workflows generate results in output files.

Approach: When a notebook cell writes into files in the local file system, the corresponding workflow component will have an output corresponding to that file. When a notebook cell shows a visualization, a workflow component will be created to generate that visualization as an output file.

3.3 Workflow Structure

3.3.1 Dataflow

Differences: In a notebook, although cells are specified sequentially they might be executed in any order or a cell may not be executed at all. In a workflow, the flow of data among components must be specified, and all components are executed.

Approach: The identifiers of the files generated and consumed by components generated for a given notebook will be used to obtain the dataflow between the components, and the dataflow will be explicitly stated in the workflow structure. We assume that notebooks run the cells sequentially, so we do not consider any parallelism in the execution.

3.4 Python-Specific Differences

A few mappings are challenging because of the specific way that Python is used in Jupyter Notebooks. The IPython kernel allows notebooks to use special functions that the standard Python interpreter does not support. Since a workflow component would be executed using standard Python, these functions cannot be directly mapped. Notebooks are also designed for human readability and are, as a result, much more documented and aggregate more resources than plain Python code. In addition, notebooks include Python commands to generate visualizations (e.g., graphs), which are executed and the results shown in the notebook but not necessarily saved.

3.4.1 Visualizations

Differences: Notebooks show visualizations which may not be saved into a file. Workflow components would generate visualizations and save them in an output file.

Approach: If the notebook does not save a visualization, the workflow will automatically save the visualization in a file.

3.4.2 Magic Commands

Differences: Notebooks have IPython kernel commands know as *magic commands*. They start with "%" and can list all variables, return the directory being used, etc. Magic commands only work on IPython kernels. The Python standard interpreter does not recognize these commands since it is not Python code.

Approach: Magic commands will be replaced with pre-defined Python code that implements them. For magic commands where no code is available, they will be ignored and not mapped.

3.4.3 Automatic Output

Differences: In a notebook, if a variable is printed (using the print statement), it will appear as an output. In a software component, that code would not generate an output.

Approach: A workflow component that includes print commands will have an extra output with all the results from the print statements.

3.5 WINGS-Specific Differences

A few mappings are specific to requirements in the WINGS workflow system. In WINGS, the code for each software component has an associated script that indicates the command line invocation for the software, but notebooks do not have this. Input files are also treated differently from notebooks. In WINGS, input files are classified into a hierarchy of data types. Semantic metadata properties can be specified as well for all input files. Notebooks do not have either.

3.5.1 Software Components

Differences: Each workflow component has an associated script that invokes the code to be executed in that component. The script for a workflow component in WINGS must specify the invocation command, number of inputs, parameters and outputs that the code for the component expects. Notebooks do not have this information.

Approach: After mapping the code from the cells of the notebook into components, a script will be generated for each component indicating the invocation command together with the total number of inputs, outputs and parameters.

3.5.2 Input Data Files

Differences: In WINGS, each input file is assigned a data type. In notebooks, there are no data types or metadata for the files.

Approach: All input files will be considered to be of the same general data type.

3.5.3 Workflow Components

Differences: In WINGS, each workflow component is assigned to a type in a component type hierarchy based on its function.

Approach: All workflow components will be given a general component type.

3.6 Usability Requirements

Our approach requires that users make changes to their notebooks in order to facilitate the conversion of notebooks into workflows. We took into account additional requirements to reduce the burden to users and maximize the utility of the changes required:

- The user should have to make minimal changes to a notebook to allow the conversion tool to generate a workflow.
- Any changes made to a notebook should improve its readability and documentation as well as facilitate its conversion into a workflow.
- Any changes made to a notebook should be independent of the target workflow system.
- Any changes made to a notebook should improve the understanding of the dataflow.
- The workflows should include all the documentation of the original notebooks.
- All the results generated by a notebook should be generated by the workflow as well, even if they are not explicit in the notebook.
- A conversion tool should automate the process as much as possible, and some manual intervention may be needed after running it.

4 GUIDELINES TO DEVELOP NOTEBOOKS

Based on our approach to map notebooks into workflows, we designed a set of guidelines that users can follow to facilitate the conversion of notebooks into workflows. Users who are not creating workflows will be able to use these guidelines to create notebooks that have more explicit dataflow, which will facilitate understanding, comparisons, and reuse by others.

We list here the set of guidelines, each with a justification.

1. **Provide at least one cell with running code:** a workflow component must have running code within it to be created and a workflow must be composed of at least one component.
2. **Write into files any newly generated data:** the code in a cell should write to files with the intermediate and final data generated, so that other cells can use those files. This will make the dataflow across cells more clear. Here we have a trade-off between input/output (I/O) performance in disk and provenance capture.
3. **Keep code that uses the same file in the same cell:** if files are opened and used across many cells, all those cells should be merged into a single cell, making the notebook cells and workflow components more modular.
4. **Keep the notebook clean and working:** the cells that are not needed to run the notebook should be removed and the code in cells must be running correctly to create the workflow components.

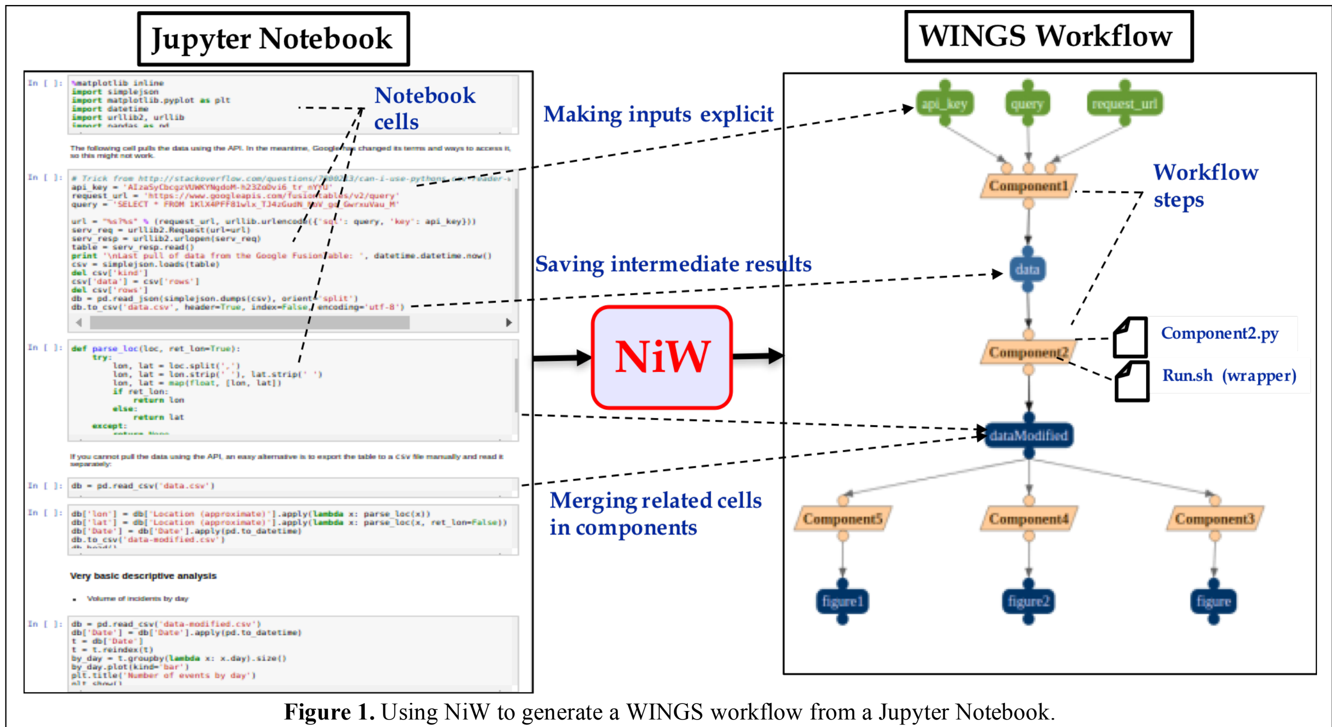

Figure 1. Using NiW to generate a WINGS workflow from a Jupyter Notebook.

5. **Ensure that the notebook produces correct results when running its cells from top to bottom:** the notebook cells are considered sequentially (i.e., from top to bottom) to create the workflow structure. This makes it easier to capture the dataflow between cells and understand the notebook.
6. **Provide meaningful names for variables and files:** these names should make clear what kind of data the files contain. Avoid names such as "load" or "data05". Instead, use names like "PluviometricCalculation" or "SensorReadings". This makes the visual presentation of notebooks and workflows more readable.

These guidelines aim to facilitate the automated conversion of notebooks into workflows. They also improve the understandability of notebooks by making the dataflow more clear.

5 NiW: A Tool for Converting Notebooks into Workflows

NiW (Notebooks into Workflows) is a prototype system that implements our approach to convert notebooks into workflows. Our current NiW prototype creates workflows for WINGS system from Jupyter Notebooks. The software is available online [3].

NiW takes as input a notebook file and generates: 1) a zip file for each workflow component (e.g., Component1.zip), containing the component code as a Python script (e.g., Component1.py), a script file (named io.sh) to handle the inputs and outputs of the component, and a script (named *run*) to execute the component; 2) a file with a list of the names of the components and their inputs, outputs, and parameters; and 3) the workflow structure. NiW generates first (1) and (2) and uses those files to automatically create (3). NiW also creates the data type "File" and associates all data files (inputs, intermediate, and results) to this data type. NiW uses the notebook's filename to name the workflow.

Figure 1 illustrates how the notebooks are converted into workflows by NiW using the approach outlined in Section 3.

5.1 Current Limitations of NiW

The following are limitations of our current NiW implementation. Python is the only programming language supported. The use of magic commands is restricted, currently only the magic command %matplotlib (which allows visualizations to be generated) is supported. The only methods supported for opening files are the built-in method "open" and the method "read_csv" from Pandas,[3] a well-known data analysis library in Python. Only Matplotlib can be used to generate visualizations. Finally, the notebook should run fully without errors. This is because if an error occurs while executing a notebook, it would be difficult to identify how data are generated and used throughout all the cells. Moreover, errors in code might be propagated to the workflow components.

5.2 Using NiW

To demonstrate how NiW works we have chosen a Jupyter Notebook for computational journalism taken from http://nbviewer.jupyter.org/gist/darribas/4121857. This notebook was created by journalists at The Guardian newspaper and uses real world data to analyze and map the incidents during the 2012 Gaza-Israel crisis, exploiting the spatial as well as the temporal

[3] http://pandas.pydata.org

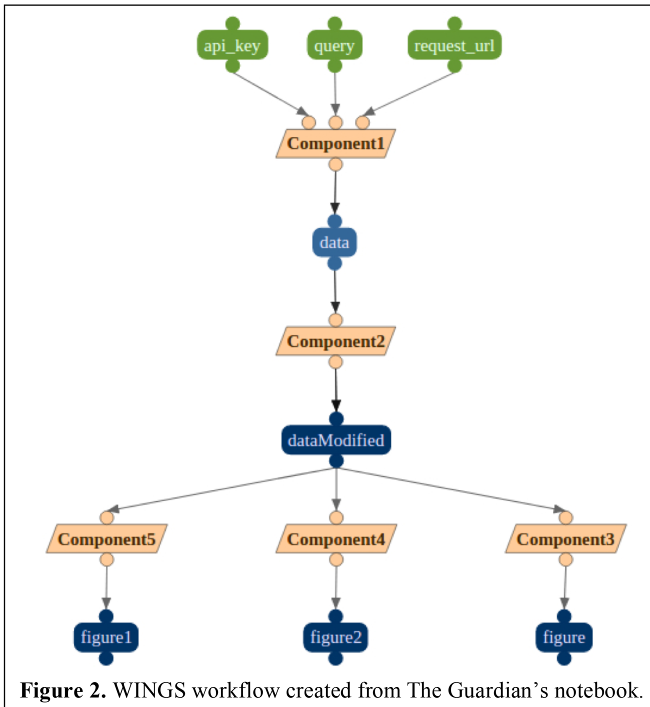

Figure 2. WINGS workflow created from The Guardian's notebook.

dimension of the data. The modified version of the notebook, the WINGS workflow, and the workflow execution are available at [3].

We modified the notebook based on the guidelines presented in Section 4, and to address the limitations of our current implementation of NiW mentioned above. The only changes required by our guidelines in the notebook code were related to guideline #3 – to write newly generated data into files: (1) saving the data retrieved online in a local file, instead of loading it in memory to be used in subsequent cells; (2) saving changes made to the data in each cell into a new file; (3) opening the updated data file saved by (2) in subsequent cells.

The nine code cells in the original notebook resulted in five workflow components. The code cells containing only library imports were merged with other components as well as the cells containing the declaration code of the function *parse_loc*. The inputs of the workflow are the parameters *api_key, request_url* and *query*, variables with assignment to string values in the original cell. After retrieving the data, it is saved as a CSV file by *Component1*. Components 3, 4 and 5 save the graphs generated for future inspection, originally showed inline in the notebook.

The modified notebook is improved for use by scientists with respect to the original version in several respects, such as making inputs explicit, saving intermediate results, merging related cells into meaningful components, and making outputs explicit.

Figure 2 shows the workflow created by NiW using the modified notebook as input. Note that even if notebooks always have a sequential structure, workflows do not. In this case, there are three components at the bottom that could be run in any order because there are no data dependencies across them. This helps a user see how the different steps are related. Unfortunately, the workflow is not more understandable because the components and datasets do not have very meaningful names. A user could easily edit the workflow in WINGS to change those names.

One benefit of the workflow is to support comparison and provenance when run with new datasets. Journalists from The Guardian created the input dataset in collaboration with Internet users. If the input data is updated, the workflow could easily be executed again, and its results can be compared. Since all the intermediate results are stored as provenance information, they may also be compared to previous executions. Another benefit of the workflow is to compare the results when the code changes. In this case, the notebook is collaborative and can be extended by users via GitHub. When the notebook is changed, NiW can be re-run and a new workflow would be generated and executed. The workflows for different notebook versions can be easily compared.

6 RELATED WORK

There are several related approaches to expose the dataflow within scripts and/or to map scripts into structures that support provenance tracking.

NoWorkflow [11] captures provenance information from scripts to help scientists understand the script execution. However, this approach does not simplify the understanding of the script specifications for non-programmers. YesWorkflow [10] enables scientists to make explicit the dataflow in scripts by providing special tags that scientists use to annotate the scripts. These annotations split the script into steps and clarify the inputs and outputs of each step as well as the structure of the workflow. It enables the creation of a visualization based on these annotations, helping scientists to understand the dataflow within the script. However, the scientist still has a script which is difficult to reuse compared to workflows.

W2Share [2] focus on the conversion of scripts into scientific workflows. This approach automatically generates workflows from annotated scripts. However, this work does not consider peculiarities of notebooks.

[13] proposes an approach to capture provenance from notebooks automatically allowing the analysis of provenance information within the notebook, both to reason about and to debug their work. [9] captures dataflows from notebooks by specifying a unique and persistent identifier for each cell and its outputs which can be referred in other cells. In our work we provide guidelines to notebook designers to improve the understanding of the notebooks by scientists, and then converted the modified notebooks into workflows automatically.

7 CONCLUSIONS

We presented an approach to map notebooks into workflows, addressing many issues that arise because of the implicit dataflow in notebooks. We introduced a set of general guidelines for notebook developers that help make dataflow more explicit, which improves understandability and provenance tracking. We

implemented NiW, a prototype tool that can convert notebooks that follow those guidelines into workflows, in particular Jupyter Notebooks into WINGS workflows.

An important area of future work is to make workflows more understandable. Users could edit the names of the workflow components and datasets to make them more meaningful. Another approach would be to use metadata tags for notebooks to facilitate the creation of components and workflows. For example, a metadata tag could be added in the notebook to give each cell a meaningful name, so that NiW would use that name rather than a synthetic one. Another example is the assignment of markdown cells to workflow components, could also be addressed by using metadata tags provided by the notebook creator.

Future work also includes the automatic generation of notebooks from workflows. This would be useful for users who reuse workflows written by others, as it would enable them to use the notebook environment to inspect the code that implements the workflows. In addition, a notebook diagram could be included in the notebook to make the role of each cell clearer.

An interesting direction for future work is to explore the use of workflows for tracking provenance of notebook results and for comparing different notebooks. Workflows can provide provenance records for all the newly generated results. In addition, the structure of workflows makes it easier to compare notebooks because they expose the similarities, the common sub-workflows, and the differences in implementations. There are many opportunities to explore the interplay between notebooks and workflows in terms of alternative user interfaces, execution paradigms, and provenance tracking, and comparison and reuse.

ACKNOWLEDGMENTS

This work was supported in part by a grant from the US National Science Foundation under award ICER-1440323 and ICER-1632211 (EarthCube RCN IS-GEO), and in part by the Sao Paulo Research Foundation (FAPESP) under grants 2017/03570-3, 2014/23861-4 and 2013/08293-7. We would like to thank many collaborators for their feedback on this work, in particular Jeremy White and Zachary Stanko.

REFERENCES

[1] Bergmann, R.; and Gil, Y. Similarity Assessment and Efficient Retrieval of Semantic Workflows. Information Systems Journal, 40. 2014.

[2] Carvalho, L. A. M. C.; Malaverri, J. E. G.; Medeiros, C. B. Implementing W2Share: Supporting Reproducibility and Quality Assessment in eScience. In Proceedings of the 11th Brazilian e-Science Workshop, São Paulo, Brazil, 2017.

[3] Carvalho, L. A. M. C, Wang, R and Garijo, D. (2017, December 9). KnowledgeCaptureAndDiscovery/niw: Notebooks into workflows 0.0.1 (Version 0.0.1). Zenodo. http://doi.org/10.5281/zenodo.1098344

[4] Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman,C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C. and Katz, D. S. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. (2005) Scientific Programming Journal, vol. 13, pp. 219-237.

[5] Garijo, D.; Corcho, O.; Gil, Y.; Gutman, B. A.; Dinov, I. D.; Thompson, P.; and Toga, A. W. FragFlow: Automated Fragment Detection in Scientific Workflows. In Proceedings of the IEEE Conference on e-Science, Guaruja, Brazil, 2014.

[6] Garijo, D.; Corcho, O.; Gil, Y.; Braskie, M. N.; Hibar, D.; Hua, X.; Jahanshad, N.; Thompson, P.; and Toga, A. W. Workflow Reuse in Practice: A Study of Neuroimaging Pipeline Users. In Proceedings of the IEEE Conference on e-Science, Guaruja, Brazil, 2014.

[7] Gil, Y.; Ratnakar, V.; Kim, J.; Gonzalez-Calero, P. A.; Groth, P.; Moody, J.; and Deelman. Wings: Intelligent Workflow-Based Design of Computational Experiments. E. IEEE Intelligent Systems, 26(1). 2011.

[8] Hauder, M.; Gil, Y.; Sethi, R.; Liu, Y.; and Jo, H. Making Data Analysis Expertise Broadly Accessible through Workflows. In Proceedings of the Sixth Workshop on Workflows in Support of Large-Scale Science (WORKS'11), held in conjunction with SC 2011, Seattle, Washington, 2011.

[9] Koop, D., and Patel, J. Dataflow Notebooks: Encoding and Tracking Dependencies of Cells. In 9th USENIX Workshop on the Theory and Practice of Provenance (TaPP 17). USENIX Association. 2017.

[10] McPhillips, T., Song, T., Kolisnik, T., Aulenbach, S., Belhajjame, K., Bocinsky, K., Cao, Y., Chirigati, F., Dey, S., Freire, J. and Huntzinger. YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts., D. International Journal of Digital Curation 10, no. 1 (2015): 298-313.

[11] Murta, L., Braganholo, V., Chirigati, F., Koop, D. and Freire, J. noWorkflow: capturing and analyzing provenance of scripts. In *International Provenance and Annotation Workshop* (pp. 71-83). Springer. 2014.

[12] Oinn, T., M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences. Concurrency and Computation: Practice and Experience, 18(10), 2006.

[13] Pimentel, J.F.N., Braganholo, V., Murta, L. and Freire, J. Collecting and analyzing provenance on interactive notebooks: when IPython meets noWorkflow. In *Workshop on the Theory and Practice of Provenance (TaPP), Edinburgh, Scotland* (pp. 155-167), 2015.

[14] Shen, H. Interactive notebooks: Sharing the code. Nature, 05 November 2014.