IJDC | *General Article*

YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts

Timothy McPhillips[1], Tianhong Song[2], Tyler Kolisnik[3], Steve Aulenbach[4], Khalid Belhajjame[5], R. Kyle Bocinsky[6], Yang Cao[1], James Cheney[13], Fernando Chirigati[7], Saumen Dey[2], Juliana Freire[7], Christopher Jones[8], James Hanken[16], Keith W. Kintigh[17], Timothy A. Kohler[6,18], David Koop[9], James A. Macklin[15], Paolo Missier[10], Mark Schildhauer[8], Christopher Schwalm[11], Yaxing Wei[12], Mark Bieda[3], Bertram Ludäscher[1,14]

[1]Graduate School for Library and Information Science (GSLIS), University of Illinois at Urbana-Champaign (UIUC); [2]Department of Computer Science, University of California, Davis; [3]Department of Biochemistry and Molecular Biology, University of Calgary; [4]University Corporation for Atmospheric Research (UCAR) and U.S. Global Change Research Program (USGCRP); [5]Paris Dauphine University, LAMSADE; [6]Department of Anthropology, Washington State University, Pullman; [7]New York University; [8]University of California, Santa Barbara; [9]University of Massachusetts, Dartmouth; [10]University of Newcastle, UK; [11]Northern Arizona University; [12]Oak Ridge National Laboratory; [13]University of Edinburgh; [14]National Center for Advanced Supercomputing Applications (NCSA), UIUC; [15]Agriculture and Agri-Food Canada; [16]Museum for Comparative Zoology, Harvard; [17]School of Human Evolution & Social Change, Arizona State University, Tempe; [18]Santa Fe Institute.

Abstract

Scientific workflow management systems offer features for composing complex computational pipelines from modular building blocks, executing the resulting automated workflows, and recording the provenance of data products resulting from workflow runs. Despite the advantages such features provide, many automated workflows continue to be implemented and executed outside of scientific workflow systems due to the convenience and familiarity of scripting languages (such as Perl, Python, R, and MATLAB), and to the high productivity many scientists experience when using these languages. YesWorkflow is a set of software tools that aim to provide such users of scripting languages with many of the benefits of scientific workflow systems. YesWorkflow requires neither the use of a workflow engine nor the overhead of adapting code to run effectively in such a system. Instead, YesWorkflow enables scientists to annotate existing scripts with special comments that reveal the computational modules and dataflows otherwise implicit in these scripts. YesWorkflow tools extract and analyze these comments, represent the scripts in terms of entities based on the typical scientific workflow model, and provide graphical renderings of this workflow-like view of the scripts. Future version of YesWorkflow will also allow the prospective provenance of the data products of these scripts to be queried in ways similar to those available to users of scientific workflow systems.

Submitted 20 January 2015 | *Revision received* 2 March 2015 | *Accepted* 2 March 2015

Correspondence should be addressed to Timothy McPhillips, Graduate School for Library and Information Science (GSLIS), University of Illinois at Urbana-Champaign (UIUC), email: tmcphillips@absoluteflow.org; or Bertram Ludäscher, GSLIS & National Center for Advanced Supercomputing Applications (NCSA), UIUC, email: ludaesch@illinois.edu

An earlier version of this paper was presented at the 10th International Digital Curation Conference.

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

1 Introduction

Many scientists use scripts (written in Python, R, or MATLAB, for example) or scientific workflow environments for data processing, analysis, model simulation, result visualization, and other scientific computing tasks. In addition to the widespread use in the natural sciences, computational automation tools are also increasingly used in other domains; for example, for data mining workflows in the digital humanities (Van Zundert, 2012), or to implement data curation workflows for natural history collections (Dou et al., 2012). One advantage of using scientific workflow systems (e.g., Galaxy [Goecks, Nekrutenko & Taylor, 2010], Kepler [Ludäscher et al., 2006], Taverna [Oinn et al., 2004], VisTrails [Bavoil et al., 2005], RestFlow[1] [Tsai et al., 2013]) is that they often include capabilities to track data as it is being processed. By capturing and subsequently sharing such provenance information, scientists can provide a detailed account of how their results were derived from the given inputs via intermediate results, workflow steps, and parameter settings, thereby facilitating transparency and reproducibility of workflow products (Stodden, Leisch & Peng, 2014). In addition to this external use, provenance information can also be used internally; for example, to allow scientists to trace sources of errors and to debug their workflows.

The data provenance captured by workflow environments is sometimes called *retrospective provenance* to distinguish it from another form called *prospective provenance* (Clifford, Foster, Voeckler, Wilde & Zhao, 2008; Lim, Lu, Chebotko & Fotouhi, 2010). The former consists of data dependencies and lineage information recorded at runtime, which can then be used later for retrospective exploration and analysis (also known as "querying provenance" [Davidson & Freire, 2008]). In contrast, prospective provenance is a description of the computational process itself; that is, the workflow specification is considered a form of provenance information, describing the method by which analysis results and other data products are obtained. Scientific workflow systems therefore naturally support both forms of provenance: prospective provenance by visually presenting a workflow as a directed graph with data and process steps, and retrospective provenance by capturing and subsequently exporting runtime provenance.

Despite these and other advanced features of workflow systems, a vast number of computational workflows continue to be developed using general purpose or specialized scripting languages such as Python, R, and MATLAB. This is true in particular for the "long tail of science" (Wallis, Rolando & Borgman, 2013; Heidorn, 2008), where advanced features such as provenance support are rarely available. At the time of writing, for example, provenance libraries for R have only recently been announced (Lerner & Boose, 2014), while for Python a new tool called noWorkflow has just been developed (Murta, Braganholo, Chirigati, Koop & Freire, 2014). The noWorkflow (***n**ot **o**nly **w**orkflow*) system uses Python runtime profiling functions to generate provenance traces that reflect the processing history of the script. Thus, noWorkflow gives users the advantage of automatically captured retrospective provenance information in a manner similar to workflow systems, but allows them to continue working in their familiar Python scripting environment without adopting a new system.

In the following, we describe a new tool called YesWorkflow that complements

[1] RestFlow wiki: http://restflow.org/

 doi:10.2218/ijdc.v10i1.370

noWorkflow by revealing prospective provenance in scripts; that is, YesWorkflow makes latent workflow information from scripts explicit. In particular, dataflow dependencies that are often "hidden" inside a script and not easily understood by outsiders are extracted from simple user annotations; they can then be exported and visualized in graph form. The main features of YesWorkflow, abbreviated here to YW, are as follows:

- YW exposes prospective provenance (workflow structure and dataflow dependencies) from scripts based on simple user annotations.
- YW annotations are embedded inside of comments, so they are *language independent* and can be used for example in Python, R, and MATLAB.
- YW annotations and the underlying model are deliberately kept simple to allow scientists a very low entry bar for adoption.
- The YW toolkit is a grass-roots, agile, open source effort, whose simple and modular architecture and underlying UNIX philosophy facilitates interoperability and extensibility.
- The current YW prototype generates different, easily reusable output formats, including three different views of the extracted workflow graph in Graphviz/DOT form: *process-centric*, *data-centric*, and *combined*.

We discuss YW limitations and plans for future development in Section 7.

2 YesWorkflow Model and Annotation Syntax

In order to use YesWorkflow a script author marks up scripts using a simple keyword-based annotation or tagging mechanism, embedded within the comments of the host language. YW annotations are expressions of the form `@`*tag*␣*value*: `@`*tag* is one of the recognized YW keywords, after which a *value* follows, separated by one or more whitespace characters. Thus, the YW annotation syntax mimics the syntax of conventional documentation generators such as Javadoc and Doxygen.

The YW tool then interprets the embedded, structured comments and builds a simple workflow model of the script. This model represents scripts in terms of scientific workflow entities: programs, workflows, ports, and channels.

- A *program block* (abbreviated to *program* or *block*) represents a computational step in the script that receives input data and produces (intermediate or final) output data. A program is designated in a script by bracketing the relevant code between a pair of `@begin` and `@end` comments. Program blocks are usually visualized as boxes. A block that contains other programs is considered a *workflow*.
- A *port* represents a way in which data flows into or out of a program or workflow. Ports are identified by `@in` and `@out` annotations in the source code comments.
- A *channel* is a connection between an `@out` port of a program and an `@in` port of another (or, in case of feedback loops, the same) program. YW infers channels by matching the names of `@in` and `@out` ports within the same workflow.

Figure 1 depicts a workflow view extracted from a sample Python script for standardizing Net Ecosystem Exchange (NEE) data in the MsTMIP project, described in Section 4.2.

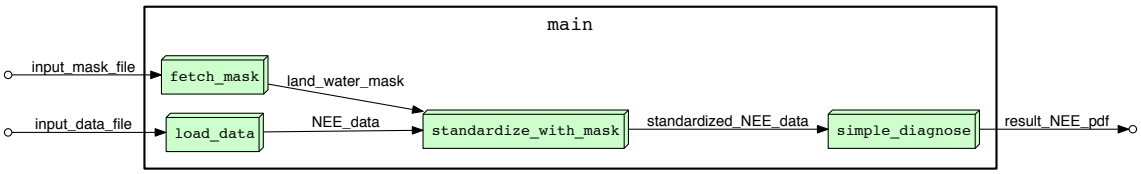

Figure 1. Process-oriented workflow view of a script: boxes represent *programs* (code blocks); edges represent dataflow *channels*; edge labels indicate *data* elements.

2.1 Alternative Workflow Views

The process-oriented view in Figure 1 is the default YW view shown to the user, as it emphasizes the overall block structure given by the script author using `@begin` and `@end` markers. The extracted YW model can however be rendered in other forms. For example, Figure 2 depicts a *data-oriented view*, where data elements (i.e., dataflow channels obtained from `@in` and `@out` tags) are shown as nodes, while programs are only mentioned in edge labels. Finally, Figure 3 shows a *combined workflow view* in which both programs and data channels are represented as nodes.

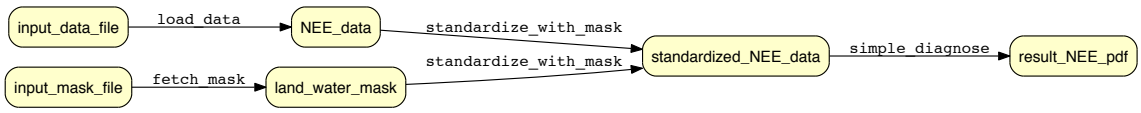

Figure 2. Data-oriented workflow view: program blocks are mentioned in edge labels only, while data channels are exposed as proper graph nodes.

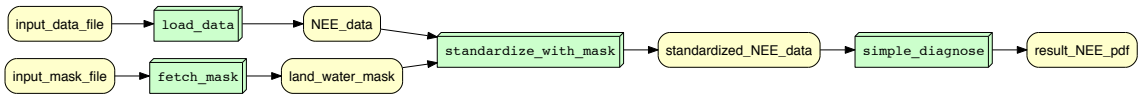

Figure 3. Combined workflow view of a script: both programs and data are nodes.

3 Querying YesWorkflow Models

The workflow structure of large scripts can be difficult to interpret fully even when represented graphically. While the YW prototype is limited to such graphical views, the YW comments and model are sufficient to support queries that reveal specific aspects of the script in workflow terms. Example workflow-structure queries that will be supported by YesWorkflow include the following:

- List all of the code blocks defined in the script along with any description given for each.
- List the code blocks nested (directly or indirectly) within a particular code block.
- List the code blocks that invoke a particular function or external program.
- List the code blocks that contain a particular block (directly or indirectly).

- List the code blocks that receive inputs derived (directly or indirectly) from the outputs of a particular upstream code block.
- List the code blocks affected (directly or indirectly) by a particular parameter value provided to the script.

3.1 Prospective Data Provenance Queries

In addition, YesWorkflow will allow scripts marked up with YW comments to be queried from a data provenance perspective. Because YesWorkflow analyzes the definition of a workflow (the script plus YW comments) rather than information recorded during a run of the script, YesWorkflow will support queries against *prospective* provenance. Example prospective provenance queries include:

- Given the name of an output of the script, list the inputs to the script that the output depends on (directly or indirectly).
- List the computational steps (code blocks) involved in deriving a particular output of the script, or of a named intermediate data product.
- For a particular computational step reveal where each input to the step comes from: an input to the script, a constant in the script, or a value produced by a different step, for example.
- Reveal the complete derivation of a particular script output. That is, list the sequence of code blocks and input and intermediate data products leading to the output. Results of queries of this kind optionally may be rendered graphically.

3.2 Inference of Retrospective Data Provenance

As described above, YesWorkflow will allow prospective provenance to be inferred from scripts marked up with YW comments. In addition, we foresee that combining the information extracted from a marked-up script with references to data files corresponding to a run of that script will in some cases allow the retrospective provenance of those files to be inferred (see also Bowers, McPhillips & Ludäscher, 2012, and Zinn & Ludäscher, 2010). That is, in cases where the entire sequence of data derivation steps for a particular output can be determined unambiguously from YW annotations, YesWorkflow will support queries of the following kind even in the absence of a run-time data-provenance recorder:

- Given a file output by a run of a script, indicate the input files from which it was derived or by which it was affected.
- Given an input file to a script, indicate which output files were derived from or affected by the data contained in that file.
- Indicate which of the parameter values applied to a run of the script affected which of its output files.

4 YesWorkflow Examples

In the following we show YW views extracted from real-world scientific use cases. The scripts were annoted with YW tags by scientists and script authors, using a very modest training and mark-up effort.[2] Due to lack of space, the actual MATLAB and R scripts with their YW markup are not included here. However, they are all available from the yw-idcc-15 repository on the YW GitHub site.[3]

4.1 Analysis of Gene Expression Microarray Data

Bioinformatics workflows commonly possess a pattern of large numbers of incoming parameters and outputs at each stage of computation. In addition, analysis of even a single bioinformatics dataset tends to yield a large number of different output files. Hence, bioinformatics pipelines are attractive candidates for workflow systems, which can capture this complexity (Bieda, 2012). Figure 4 shows a YW representation of an R script performing a classic, complex bioinformatics task: analysis of Affymetrix gene expression microarray data. This R script was modeled on our previous workflows developed in the Kepler environment (Stropp, McPhillips, Ludäscher & Bieda, 2012). The script analyzes experimental designs comprised of two conditions (e.g., microarrays from control-treated cells vs microarrays from drug-treated cells) with the option to use multiple replicates for each condition. The R script employs a set of standard Bioconductor (Gentleman et al., 2004) packages mixed with custom programming. The workflow consists of four fundamental tasks: normalization of data across microarray datasets (`Normalize`), selection of differentially expressed genes (`DEGs`) between conditions (`SelectDEGs`), determination of gene ontology (`GO`) statistics for the resulting datasets (`GO_Analysis`), and creation of a heatmap of the differentially expressed genes (`MakeHeatmap`). Each module produces outputs, and each module (aside from `MakeHeatmap`) requires external parameter inputs. Importantly, this graphical representation clearly indicates the dependence of each module on datasets and parameter inputs. This example demonstrates that YesWorkflow can provide informative visualizations of bioinformatics workflows, especially workflows involving large numbers of inputs and outputs.

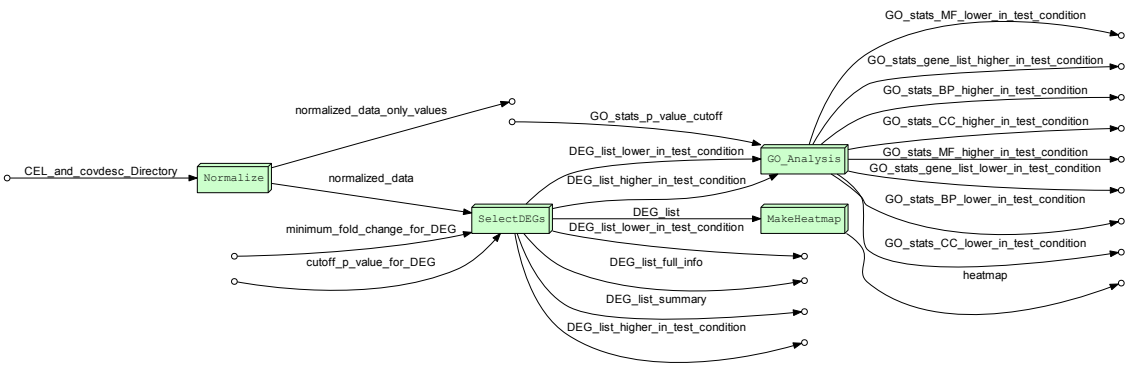

Figure 4. Process workflow view of an Affymetrix analysis script (in R).

[2] For all of these scripts, learning the YW model and annotating the scripts was done in a few hours.
[3] YesWorkflow GitHub repositories: https://github.com/yesworkflow-org

4.2 Terrestrial Biospheric Modeling

In the Multi-scale Synthesis and Terrestrial Model Intercomparison Project (MsTMIP)[4], climate scientists primarily use MATLAB scripts to standardize terrestrial biosphere model output across multiple models and simulation runs for intercomparison purposes and to facilitate diagnosis and attribution. MsTMIP is a large, collaborative effort, aimed at harmonizing a number of complex terrestrial biospheric models for the purposes of comparing these model outputs (Huntzinger et al., 2013). There is a strong need to standardize many aspects of the MsTMIP process, to assure greater uniformity in the treatment of the codes and outputs of the disparate models in the intercomparison analyses. Current practice in MsTMIP, however, is representative of many scientific investigations in that researchers develop their codes with a specific focus on functionality and efficiency. Comments are added primarily as bookmarks to assist with accessing appropriate code areas for debugging, optimization, or discussion. In the more general case, depending on whether the codes are developed in a collaborative context, structured in-code documentation may be recommended or required by the project. Nevertheless, the mechanisms for these code annotations are typically unformalized and unstructured, and rely primarily on the ability to insert non-executable comment statements in the code.

As the complexity of code grows, and the number of variants and alternative approaches increases, MsTMIP researchers need a clear and consistent way to document, review, and share their model intercomparison scripts. This provides a compelling use case for YesWorkflow, in that MsTMIP brings together models from a number of independent efforts that require harmonization into a single framework for evaluating their

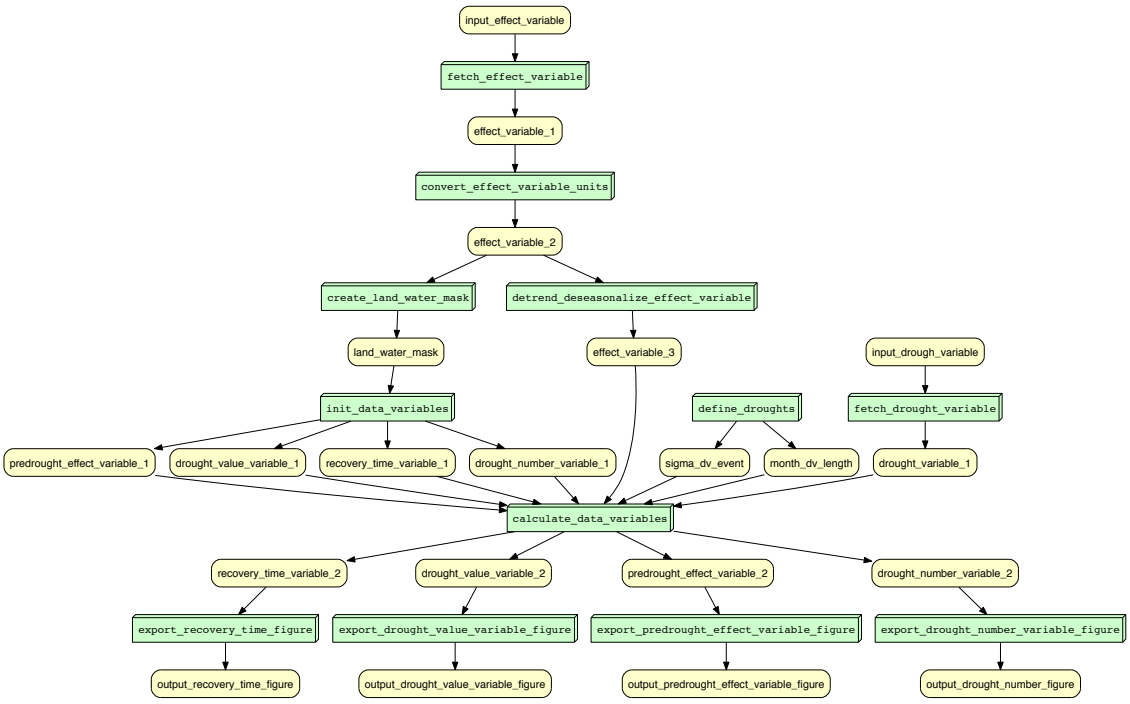

Figure 5. Combined workflow view of a MsTMIP script (in MATLAB). YW views can be easily tweaked via Graphviz properties in the generated DOT files: here, a "Taverna-style" (Oinn et al., 2004) top-down layout is used, as opposed to the default left-to-right display.

[4] MsTMIP website: http://nacp.ornl.gov/MsTMIP.shtml

relative capabilities to predict critical earth system features, such as global Net Ecosystem Exchange (NEE) data from terrestrial biogeographic realms.

A YW representation of a MATLAB script from MsTMIP is shown in Figure 5.

4.3 Paleoclimate Reconstruction

As another working example from a different field, we have used the YW markup syntax to analyze the paleoclimate reconstruction workflow presented by Bocinsky and Kohler (Bocinsky & Kohler, 2014). Their reconstruction method takes as input a spatial interpolation of contemporary weather data, the long-term record of climate held in regional tree-ring chronologies, and a handful of parameters. It then uses a novel regression-based analysis method to generate spatial reconstructions of climate extending 2000 years or more back in time. Figure 6 shows that the YW system nicely exposes the prospective provenance hidden in the underlying R script, even for scripts whose workflow views are highly non-linear.

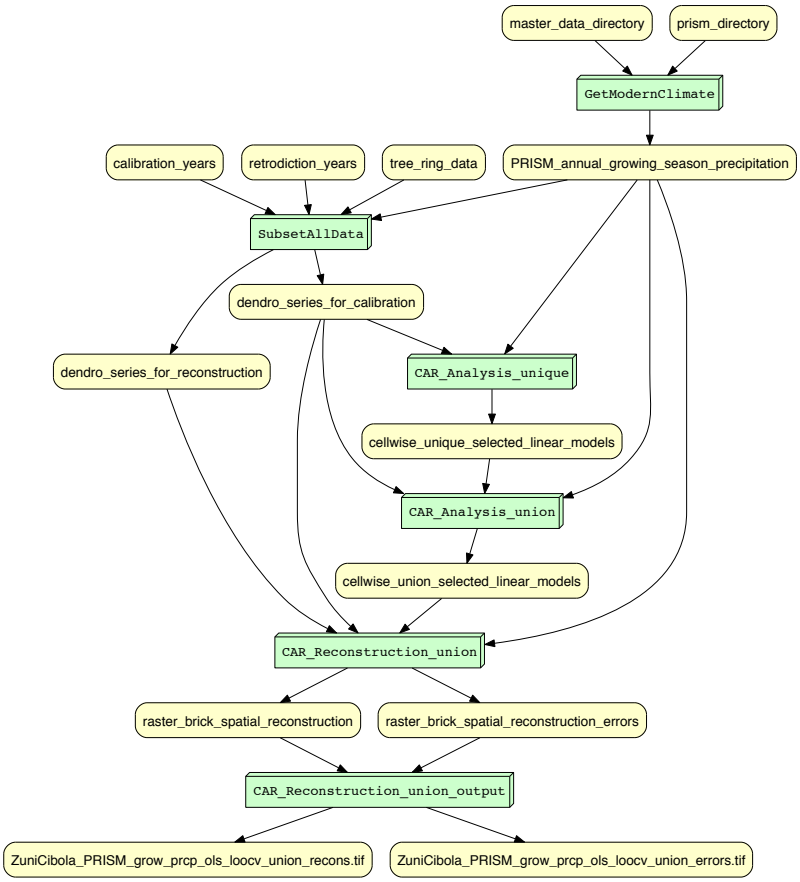

Figure 6. Combined workflow view of a paleoclimate reconstruction R script (Bocinsky & Kohler, 2014).

 doi:10.2218/ijdc.v10i1.370

5 YW Architecture

The YesWorkflow software distribution is envisioned as a set of standard modules that can be used together or separately. The primary goal of this modularity is to enable YW users and developers independently to implement alternatives to any module, as needed, to solve problems particular to their research domain. It will be possible to develop these alternative implementations and extensions in any programming language. One way we plan to facilitate such easy replacement of YW modules is to require that each standard module optionally input and output files – with well-defined formats – representing the expected inputs or outputs of that module. Any program that produces or consumes these file formats can then function as an alternative to one or more standard YW modules and can provide identical, overlapping, or completely different capabilities (e.g., the current YW prototype is primarily implemented in Java, but also contains some alternative YW modules implemented in Python).

Six standard modules (implemented in Java) are currently implemented or planned. The `YW-Extract` module identifies YW comments in a script and produces a language-independent representation of the script and the YW annotations. `YW-Model` interprets the comments identified by `YW-Extract` and builds a model of the script in terms of entities analogous to the components of a traditional scientific workflow as described in Section 2, while `YW-Graph` operates on the outputs of `YW-Model` to produce the dataflow graphs discussed in that same section. As described in Section 3, the planned `YW-Query` module will allow users to probe the structure of a complex script without having to inspect a visual representation of it. An envisioned `YW-Validate` module will ensure that YW comments in a script are consistent both with the other YW comments in the script and with the script itself. Finally, the `YW-CLI` module enables a user to execute sequences of the standard modules, starting from an input file with format appropriate to the first module in the executed sequence.

6 Related Work

The YW approach can be seen in the tradition of programming code annotation, which is widely used for facilitating code understanding and for generating documentation (e.g., Doxygen[5], Epydoc[6], Javadoc[7]). YesWorkflow builds on programming code annotation to provide a higher level of abstraction by revealing the dataflow that underlies the interactions between the different pieces of a script or program.

YesWorkflow is also related to ideas from literate programming[8] and available in tools such as Knitr (Xie, 2013) and IPython (Pérez & Granger, 2007). In literate programming, a script is decomposed into snippets of macros, which are interspersed within documents

[5] Doxygen website: http://www.doxygen.org/

[6] Epydoc website: http://epydoc.sourceforge.net/

[7] Javadoc documentation home page: http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

[8] Don Knuth has argued (Knuth, 1984) that we should change our traditional attitude to programming: "Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do".

that are written in natural language to explain the scripts and eventually analyze the results it generates upon execution. While borrowing ideas from literate programming, YesWorkflow is primarily targeted at developers who are using traditional pure scripting environments to edit their scripts and programs. YesWorkflow aims at providing a consistent interpretation and visualization of codes wherever the language provides for insertion of non-executable comments.

YesWorkflow can also contribute to the area of reproducible computational research (Stodden et al., 2014), which seeks to provide scientists with sufficient information to understand and eventually validate the results claimed by their peers. For instance, the SOLE system (Pham, Malik, Foster, Di Lauro & Montella, 2012) allows linking articles with science objects, which can be source code, a dataset, or a workflow. SOLE allows the reader (curator) to specify human-readable tags that link the paper with science objects, and it transforms each tag into a URI that points to a representation of the corresponding object. While in SOLE the scientific article is the main object that contains links to other (science) objects, we focus on the scripts produced by the scientists, and aim to facilitate the understanding of their dataflow logic. Gavish and Donoho (2011) present the notion of a *Verifiable Computational Result* (VCR), where every result is assigned a unique identifier, and results produced under the exact same conditions have the same identifier to support reproducibility.

Various tools have been proposed to capture the runtime provenance of scripts. Mechanisms that capture provenance at the operating system level (Frew, Metzger & Slaughter, 2008; Guo & Seltzer, 2012; Muniswamy-Reddy, Holland, Braun & Seltzer, 2006) monitor system calls to track the data dependencies between computational processes. Some tools (Bochner, Gude & Schreiber, 2008; Davison, 2012; Huq, Apers & Wombacher, 2013; Murta et al., 2014) have been developed to capture runtime provenance for Python scripts: while Bochner et al. (2008) and Davison (2012) propose Python libraries and APIs that need to be added to the code to capture the execution steps, ProvenanceCurious (Huq et al., 2013) and noWorkflow (Murta et al., 2014) are transparent and do not require changes to the scripts. Similarly, RDataTracker (Lerner & Boose, 2014) captures provenance from the execution of R scripts, and the approach taken by Tariq, Ali and Gehani (2012) supports all programming languages allowed by the LLVM compiler framework. We note that the YW approach is complementary to these tools, since it captures prospective provenance of scripts. We argue that YesWorkflow, along with retrospective provenance approaches, provide a low-effort entry point for scientists who want to reap some of the benefits of scientific workflow systems while still using their familiar scripting environments.

7 YesWorkflow Development Roadmap

In the following we list some limitations of the current YesWorkflow prototype and highlight features planned for future releases of the software.

7.1 Visualization of Nested Code Blocks

The `YW-Extract` and `YW-Model` modules support nesting of code blocks. Any pair of `@begin` and `@end` comment lines can enclose code that contains any number of other code blocks delimited with `@begin` and `@end` comment lines. The workflow model constructed

for a script reflects such nesting: the top-level workflow corresponding to the script as a whole may contain one or more programs (code blocks), and any of these programs can in turn be a sub-workflow that contains further nested programs and workflows. Future versions of `YW-Graph` will reveal these nested code blocks and render sub-workflows graphically.

7.2 Functions and Function Calls

`YW-Extract` currently expects nested code blocks to be defined in-line. However, many scripts are structured as functions (or classes) with a top-level script that calls these functions (or methods on objects). These functions can in turn call other functions. Future versions of YesWorkflow will allow function declarations to be marked up with YW comments in a manner similar to that supported by Javadoc and Doxygen. Calls to these functions also will be annotated with YW markup. The result will be that `YW-Extract` and `YW-Model` will be able to represent function calls as nested code blocks.

7.3 Interactive Graphs

`YW-Graph` currently produces static graphical views (in the well-known Graphviz/DOT format). An interactive viewer for YW graphical output will make these graphs easier to explore and interpret. In the planned graphical user interface, clicking on a data item in the combined or data views optionally will highlight the (prospective) direct and indirect data dependencies for that data item (the data from which it will be derived when the script is run). Features for expanding and collapsing nested subworkflows also will facilitate exploration of these graphs.

7.4 Live Graph View

Although the primary function of YesWorkflow is to reveal workflow-like structure in existing scripts, YesWorkflow also can be used as a *design tool* when developing new scripts (or even before a script is written). Future versions of YesWorkflow will better support such applications by providing live-update features to the interactive graph capabilities described above. Given a set of script files, the live-graph feature will monitor these files for changes and update the chosen graphical view automatically. Users of this feature will continue to be able use their favorite text editor or IDE for developing their scripts.

7.5 Distinguished Data and Parameters

The inputs to scripts for processing scientific data often can be viewed either as data (the data to be processed by the scripts) or as parameters (values that control how that data is processed). Planned versions of the YW comment vocabulary will allow data and parameters to be distinguished. `YW-Graph` optionally will emphasize graph edges, nodes, and labels representing data over those representing parameters.

7.6 Validation of Comments

The future `YW-Validate` module will perform extensive validation of YW comments in light of the actual code in the script. This capability will help guide users adding YW comments to their script. Perhaps more importantly, automatic validation will help prevent initially correct YW comments from becoming stale (i.e., incorrect) when the underlying script is changed or refactored. `YW-Validate` will perform validity checks including the following:

- Confirm that data names used in `@in` and `@out` comments actually appear in the code bracketed by associated `@begin` and `@end` comments.
- Confirm that the names of functions referred to in YW comments for function declaration or for function calls match the names of the functions actually declared or called.
- Confirm that continuous data dependency chains exist from each script output all the way back to script inputs (and embedded constants).

8 Conclusions

YesWorkflow is an agile, grass-roots effort that aims to bring workflow modeling and analysis features to scientific workflows that are defined in script form. Through simple user-annotations in the comments of scripts, dataflow and workflow structure are revealed by the YW toolkit. The user can thus exploit prospective provenance information from scripts by, for example, visualizing, querying, and analyzing this information.

Our early YW prototype[9] has been used by scientists from different domains to mark up complex, real-world scientific scripts with ease. Encouraged by the enthusiastic response of the early adopters, a number of researchers will be incorporating YesWorkflow into their projects, thereby guiding and driving the future development of the toolkit.

MsTMIP researchers plan to annotate their scripts such that authors, as well as reviewers and potential new users, will be able to click on the workflow steps in the interactive YW graph viewer and inspect the corresponding code-blocks in the original script. When clicking on data elements, they will be taken to a folder containing the data instances that were used in the various runs of the script (provided these have been shared). Since the YW approach is language independent, it will also facilitate code migration from MATLAB to R, say, or from R to Python.

In the Kurator project[10] we plan to enable collection managers to author their own data curation workflows using both an Akka-based workflow system and via scripting languages such as Python and R. In the latter case, Kurator tool users will annotate their scripts with YW comments to enable provenance queries to span script-based curation workflows. The Kurator team also plans to use the `YW-Graph` and `YW-Query` tools to graphically render workflows defined using the Kurator-Akka workflow system and to query the prospective provenance of products of these workflows.

[9] YesWorkflow GitHub repositories: https://github.com/yesworkflow-org
[10] Kurator Project public wiki: http://wiki.datakurator.net/

Finally, DataONE is planning a number of enhancements to the YW annotation language. For example, in addition to the currently supported, simple user-defined vocabulary for program blocks and data elements, controlled vocabularies from shared ontologies may be used with these extensions. Similarly, to improve YW interoperability within the DataONE infrastructure, PROV (Moreau & Missier, 2013) and ProvONE (Cuevas-Vicenttín et al., 2015) compatible vocabulary extensions may be used in YesWorkflow in the future.

Acknowledgments

This material is based upon work supported by the National Science Foundation under grants DBI-1356751, ACI-0830944, SMA-1439603, SMA-1439591, SMA-1439516, and IIS-1118088. Juliana Freire and Fernando Chirigati were supported in part by the Moore-Sloan Data Science Environment at NYU, Sloan Foundation, and NSF awards CNS-1229185 and CNS-1405927. Mark Bieda and Tyler Kolisnik were supported by University of Calgary startup funds.

References

Bavoil, L., Callahan, S. P., Crossno, P. J., Freire, J., Scheidegger, C. E., Silva, C. T. & Vo, H. T. (2005). VisTrails: Enabling interactive multiple-view visualizations. In *Visualization 2005 (VIS '05)* (pp. 135–142). IEEE. doi:10.1109/VISUAL.2005.1532788

Bieda, M. (2012). Kepler for 'omics bioinformatics. *Procedia Computer Science*, *9*, 1635–1638. doi:10.1016/j.procs.2012.04.180

Bochner, C., Gude, R. & Schreiber, A. (2008). A Python library for provenance recording and querying. In J. Friere, D. Koop & L. Moreau (Eds.), *Lecture Notes in Computer Science: Vol. 5272. Provenance and Annotation of Data and Processes* (pp. 229–240). doi:10.1007/978-3-540-89965-5_24

Bocinsky, R. K. & Kohler, T. A. (2014). A 2,000-year reconstruction of the rain-fed maize agricultural niche in the US southwest. *Nature Communications*, *5*, Article 5618. doi:10.1038/ncomms6618

Bowers, S., McPhillips, T. & Ludäscher, B. (2012). Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. In P. Groth & J. Frew (Eds.), *Lecture Notes in Computer Science: Vol. 7525. Provenance and Annotation of Data and Processes* (pp. 82–96). doi:10.1007/978-3-642-34222-6_7

Clifford, B., Foster, I., Voeckler, J.-S., Wilde, M. & Zhao, Y. (2008). Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience*, *20*(5), 565–575. doi:10.1002/cpe.1256

Cuevas-Vicenttín, V., Ludäscher, B., Missier, P., Belhajjame, K., Chirigati, F., Wei, Y., ... Leinfelder, B. (2015, January 15). *ProvONE: A PROV extension data*

model for scientific workflow provenance. Retrieved from https://purl.dataone.org/provone-v1-dev

Davidson, S. B. & Freire, J. (2008, June). Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 1345–1350). New York, NY: ACM Press. doi:10.1145/1376616.1376772

Davison, A. (2012). Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, *14*(4), 48–56. doi:10.1109/MCSE.2012.41

Dou, L., Cao, G., Morris, P., Morris, R., Ludäscher, B., Macklin, J. & Hanken, J. (2012). Kurator: A Kepler package for data curation workflows. *Procedia Computer Science*, *9*, 1614–1619. doi:10.1016/j.procs.2012.04.177

Frew, J., Metzger, D. & Slaughter, P. (2008). Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experience*, *20*(5), 485–496. doi:10.1002/cpe.v20:5

Gavish, M. & Donoho, D. (2011). A universal identifier for computational results. *Procedia Computer Science*, *4*, 637–647. doi:10.1016/j.procs.2011.04.067

Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., ... Zhang, J. (2004). Bioconductor: Open software development for computational biology and bioinformatics. *Genome biology*, *5*(10), R80. doi:10.1186/gb-2004-5-10-r80

Goecks, J., Nekrutenko, A. & Taylor, J. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, *11*(8), R86. doi:10.1186/gb-2010-11-8-r86

Guo, P. J. & Seltzer, M. (2012). BURRITO: Wrapping your lab notebook in computational infrastructure. In *4th USENIX Workshop on the Theory and Practice of Provenance (TaPP '12).* Berkeley, CA: USENIX Association. Retrieved from https://www.usenix.org/conference/tapp12/workshop-program/presentation/guo

Heidorn, P. B. (2008). Shedding light on the dark data in the long tail of science. *Library Trends*, *57*(2), 280–299. doi:10.1353/lib.0.0036

Huntzinger, D. N., Schwalm, C., Michalak, A. M., Schaefer, K., King, A. W., Wei, Y., ... Zhu, Q. (2013). The North American Carbon Program Multi-Scale Synthesis and Terrestrial Model Intercomparison Project—Part 1: Overview and experimental design. *Geoscientific Model Development*, *6*(6), 2121–2133. doi:10.5194/gmd-6-2121-2013

Huq, M. R., Apers, P. M. G. & Wombacher, A. (2013). ProvenanceCurious: a tool to infer data provenance from scripts. In *EDBT '13: Proceedings of the 16th International Conference on Extending Database Technology* (pp. 765–768). New York, NY: Association for Computing Machinery. doi:10.1145/2452376.2452475

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, *27*(2), 97–111.

Lerner, B. & Boose, E. (2014). RDataTracker: Collecting provenance in an interactive scripting environment. In *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*. Berkeley, CA: USENIX Association. Retrieved from https://www.usenix.org/conference/tapp2014/agenda/presentation/lerner

Lim, C., Lu, S., Chebotko, A. & Fotouhi, F. (2010, July). Prospective and Retrospective Provenance Collection in Scientific Workflow Environments. In *2010 IEEE International Conference on Services Computing* (pp. 449–456). IEEE. doi:10.1109/SCC.2010.18

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., . . . Zhao, Y. (2006). Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, *18*(10), 1039–1065. doi:10.1002/cpe.994

Moreau, L. & Missier, P. (2013). *PROV-DM: The PROV data model*. Retrieved from W3C website: http://www.w3.org/TR/prov-dm/

Muniswamy-Reddy, K.-K., Holland, D. A., Braun, U. & Seltzer, M. (2006). Provenance-aware storage systems. In *Proceedings of the USENIX '06 Annual Technical Conference* (pp. 43–56). Berkeley, CA: USENIX Association. Retrieved from https://www.usenix.org/legacy/events/usenix06/tech/muniswamy-reddy.html

Murta, L., Braganholo, V., Chirigati, F., Koop, D. & Freire, J. (2014). noWorkflow: Capturing and analyzing provenance of scripts. In B. Ludäscher & B. Plale (Eds.), *Lecture Notes in Computer Science: Vol. 8628. Provenance and Annotation of Data and Processes* (pp. 71–83). doi:10.1007/978-3-319-16462-5_6

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., . . . Li, P. (2004). Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, *20*(17), 3045–3054. doi:10.1093/bioinformatics/bth361

Pérez, F. & Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, *9*(3), 21–29. doi:10.1109/MCSE.2007.53

Pham, Q., Malik, T., Foster, I., Di Lauro, R. & Montella, R. (2012). SOLE: Linking research papers with science objects. In P. Groth & J. Frew (Eds.), *Lecture Notes in Computer Science: Vol. 7525. Provenance and Annotation of Data and Processes* (pp. 203–208). doi:10.1007/978-3-642-34222-6_16

Stodden, V., Leisch, F. & Peng, R. D. (Eds.). (2014). *Implementing reproducible research*. Boca Raton, FL: CRC Press.

Stropp, T., McPhillips, T., Ludäscher, B. & Bieda, M. (2012). Workflows for microarray data processing in the Kepler environment. *BMC Bioinformatics*, *13*, Article 102. doi:10.1186/1471-2105-13-102

Tariq, D., Ali, M. & Gehani, A. (2012). Towards automated collection of application-level data provenance. In *4th USENIX Workshop on the Theory and Practice of Provenance (TaPP '12)*. Berkeley, CA: USENIX Association. Retrieved from https://www.usenix.org/conference/tapp12/workshop-program/presentation/tariq

Tsai, Y., McPhillips, S. E., González, A., McPhillips, T. M., Zinn, D., Cohen, A. E., ... Soltis, S. M. (2013). Autodrug: fully automated macromolecular crystallography workflows for fragment-based drug discovery. *Acta Crystallographica Section D: Biological Crystallography*, *69*(5), 796–803. doi:10.1107/S0907444913001984

Van Zundert, J. (2012). If you build it, will we come? Large scale digital infrastructures as a dead end for digital humanities. *Historical Social Research/Historische Sozialforschung*, *37*(3), 165–186. Retrieved from http://www.jstor.org/stable/41636603

Wallis, J. C., Rolando, E. & Borgman, C. L. (2013). If we share data, will anyone use them? Data sharing and reuse in the long tail of science and technology. *PLoS ONE*, *8*(7), e67332. doi:10.1371/journal.pone.0067332

Xie, Y. (2013). *Dynamic documents with R and knitr*. Boca Raton, FL: CRC Press.

Zinn, D. & Ludäscher, B. (2010). Abstract provenance graphs: anticipating and exploiting schema-level data provenance. In D. L. McGuinness, J. R. Michaelis & L. Moreau (Eds.), *Lecture Notes in Computer Science: Vol. 6378. Provenance and Annotation of Data and Processes* (pp. 206–215). doi:10.1007/978-3-642-17819-1_23