Requirements for Facilitating the Continuous Creation of Scientific Workflow Variants

Lucas A. M. C. Carvalho[1], Daniel Garijo[2], Bakinam T. Essawy[3], Claudia Bauzer Medeiros[1], Yolanda Gil[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
[3]University of Virginia, Department of Civil and Environmental Engineering, Charlottesville, VA, U.S.A

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

ABSTRACT

Workflow systems support scientists in capturing computational experiments and managing their execution. However, workflow systems do not support scientists as they create many related workflows for an experiment. Scientists try different software implementations and different ways to process data, deciding what to do next by looking at workflow execution results. As a result, an initially created workflow will be changed to create many new variants of that initial workflow that differ in one or more steps. Our goal is to create tools that support scientists during the design of computational experiments by assisting them to create variants of an initial root workflow. In this paper, we present several use cases for creating workflow variants and describe their requirements. Finally, we discuss major research directions to address these requirements.

KEYWORDS

Scientific workflows, workflow variants, requirements

1 INTRODUCTION

Scientific workflow systems play a major role supporting computational scientists to design, document and execute their experiments, and automatically collect data provenance during the workflow execution [Wolstencroft et al., 2013][Altintas et al., 2006]. However, workflows break; thus, trying to rerun workflows may not often be possible [Zhao et al. 2012]. Another problem when trying to rerun experiments is that this is often impossible due to changes in data (e.g. when datasets are updated) or software (e.g., software is no longer available, licenses expired, projects are no longer maintained, new version are better). These problems motivate updates in workflows.

In current workflow systems, scientists create workflow variants manually. These workflow variants may be necessary, for instance, due to changes in the software used to perform the component computation, the release of new versions, or the scientists' need to explore distinct functions/methods in an experiment. Updating a workflow is a complex and time-consuming task that may involve several steps and information such as software component interfaces and software components to convert files to data types expected by each component in a workflow. The update process may create new variants for workflows and components, in addition, it may require information about versions of software used in the workflow.

In this paper, we describe use cases and their requirements to support scientists in the process of creating workflow variants and discuss research directions to address these requirements. These scenarios result from our discussion with domain scientists.

2 BACKGROUND AND RELATED WORK

Workflows describe a set of computations as well as an order for these computations. Workflow *templates* describe a general kind of analysis that can be more easily reused, providing a high-level structure of the workflow in a representation that is independent from actual data instantiation. Workflow templates are composed of several workflow *components* which are used to process data and have well-defined input and output ports as well as parameters. Workflow components are connected to other components using the input and output ports. A workflow component relies on interfaces to communicate with third-party libraries and software. Here, we use the term interface to refer to functions and their inputs, outputs, parameters, and data types. Workflow *instances* specify what data and software components are to be used in the workflow execution. Workflow executions are the result of computing the workflow and are associated with provenance of the data products generated by the workflows. WINGS [Gil et al., 2011] assists users during workflow instantiation offering an automatic mechanism to suggest components, parameters and input data based on semantic constrains and workflow templates. However, this assistance focuses on specialize an abstract workflow, rather than on suggesting variants to the workflow template.

There have been several efforts to keep track and/or manage workflow updates and versions. Vistrails [Freire et al. 2006] was the first system to track the changes in workflow instances using a change-based provenance model that records information about modifications to workflow components, inputs, outputs and parameters. They compare results of workflow executions using a visualization approach, showing to the scientist the output results for each execution (i.e., the graphics derived from each change in the workflow). [Koop 2016] proposes a versioning model for workflow version trees allowing the definition of changes that affect multiple workflow versions. Ogasawara et al. [Ogasawara et al. 2009] provide mechanisms for versioning of scientific workflows, including workflow comparison and merging capabilities. However, these approaches are only interested in capture and compare versions, while we are interested in suggesting and supporting the creation of workflow variants.

Koop et al. [Koop et al. 2011] focuses on the problem of maintaining and reusing the provenance of workflow updates when newer software versions are released and suggest how this provenance might be used in future updates. While they are only interested in the ability to update workflows when routines or software interfaces are upgraded, we are interested in variants of workflows regarding use of distinct functions, methods or software to perform an analysis or simulation in a component. Also, we are interested in suggesting possible automatic changes to the workflow structure based on component interfaces and storing information about differences between software versions and their interfaces.

Work related to capture of software information include, e.g., OntoSoft [Gil et al., 2015] that proposes an ontology to describe metadata for scientific software. However, the ontology is designed considering how scientists would approach the reuse and sharing of software, thus, there is not much information about different versions such as features, data types, inputs and outputs.

3 MOTIVATION SCENARIOS AND REQUIREMENTS FOR FACILITATING THE CONTINUOUS CREATION OF WORKFLOW VARIANTS

This section describes the main scenarios we are interested in addressing. They are the first step towards understanding how to better facilitate scientist to continuously create workflow variants. Scientists need to create variants for their experiments due to several reasons, e.g., to test new or different models using distinct functions and software and new releases of versions of software used in the experiment implementation.

Our scenarios consider a single kind of update to create a workflow variant - that of replacing the software that implements a component (e.g., a new version, or a different implementation of the same computation). This, in turn, requires considering the notion of *interface mismatch*.

There are two kinds of *interfaces* that have to be considered when updating a workflow component – the interface of the component (i.e., how it interacts with other components) and the interface of the piece of software that implements that component (i.e., its inputs, outputs, parameters and data types). A component may not need all parameters, inputs or outputs of the software; thus, for implementation purposes, these two interfaces are not necessarily the same.

Software interface updates may require workflow template updates. Suppose, for instance, that a piece of software S that implements component C is updated, and requires a new input. This, in turn, will change C's interface in the workflow template, and eventually require adding a new component to the template. By the same token, if S now requires fewer inputs, it may well be that some components will be deleted from the template.

This being said, for simplicity's sake, we will not distinguish in this text between software interface and component interface, and will refer to both as *interface*. Thus, for the rest of this discussion, we will consider two kinds of updates of workflow components:

(1) those that will not require adding or removing other components from the template;
(2) those that will require adding or removing other components, to solve the issues of interface mismatch.

We adopt the hydrology domain in our scenarios to illustrate the requirements. However, we claim that our requirements are domain-independent.

3.1 Pre-update and post-update steps

Before updating a workflow, scientists need to understand the advantages of such an update – e.g., whether changing the software used to implement a component is worth the effort, and consequences of doing so. Here, they require detailed information about software interface and new functionalities that would be introduced trough the update.

Moreover, after updates scientists want to compare the previous results with the ones obtained after the updates (always recalling that the workflow template may also have been changed). These comparisons should include results from executing the component and executing the entire workflow, so the scientists may understand how the update influenced the workflow result, and the differences between the two components. These comparisons should also include and relate software implementing workflow components, parameters and data inputs used in the execution.

These steps motivate the following requirements:

- **R1 –** Scientists should be able to specify the interfaces of a specific software version (i.e., inputs, outputs, data types, data formats and functions/methods).
- **R2 –** Scientists should be able to compare the difference between the interface of two software versions to understand the differences between them.
- **R3 –** Scientists should be able to provide relevant information that influences software operation (e.g., what is the solver used for solving equations in the system).
- **R4 –** Scientists should be able to inform what software, software version, function or method are used in workflow component implementations.

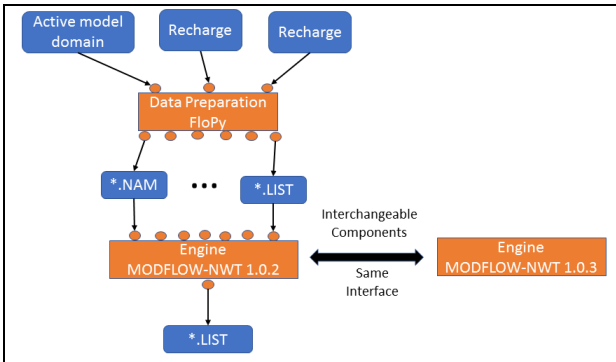

Figure 1. Replacing a workflow component with another one using the same component interface.

- **R5** – Scientists should be able to compare workflow results between different workflow executions to assess the updates performed in workflows components and templates.

3.2 Scenario 1: different software versions without changes in component interfaces

In this scenario, a scientist becomes aware of a new release of a software that he/she uses in an experiment. After assessing the differences between the versions, the scientist decides to replace the workflow component with another that uses the new software version.

Figure 1 shows a workflow whose Engine component performs an analysis using MODFLOW-NWT [1], which is a software intended for solving problems involving groundwater-flow equation. This component uses MODFLOW-NWT version 1.0.2 which is replaced with a component that uses version 1.0.3. Since these two software versions have the same interface (i.e., same data types and same number of input and output files), no changes are required to the interface of the new component. Thus, the files used in the last execution of this workflow can be reused after updating the workflow.

This scenario involves a change in the implementation of the workflow component, thus creating a new version for that component. The workflow template remains the same in case that task has been defined as an abstract component in the template. Thus, during the planning for the execution, the new component can be used in the workflow instance. Otherwise, the workflow component must be replaced in the workflow template, creating a new version for the template.

This scenario motivates the following requirements:

- **R6** – Scientists should be able to define workflow templates using abstract components and give an abstract type to workflow components according to the kind of computations performed by them.

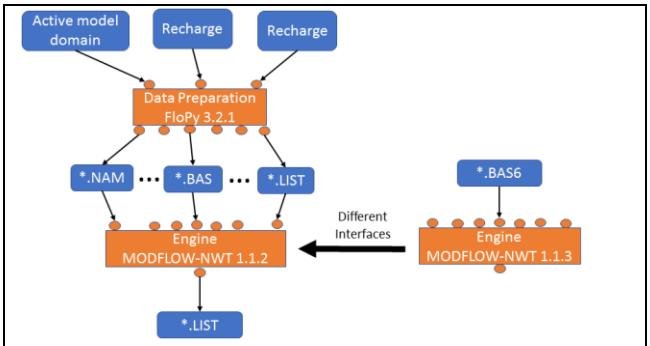

Figure 2. Replacing a component with another one using a different component interface.

- **R7** – Given a piece of software, scientists need to find workflow components implemented with newer versions or similar software or functions/methods.
- **R8** – Scientists should be able to update workflow components, creating a new version for them.
- **R9** – Scientists should be able to update existing workflow templates, creating a new version for them.

3.3 Scenario 2: different software versions with partially-compatible interfaces

In this scenario, a scientist decided to replace a workflow component with one that is implemented using a newer software version. However, the workflow components had different interfaces (i.e., use a different function, different types for the inputs, outputs or parameters). Thus, this scientist needs to add data conversion components before or after the replacing component to convert data types into the expected one.

Figure 2 shows the Engine component, implemented using MODFLOW-NWT version 1.1.2, which is replaced with a component implemented using version 1.1.3. Because the software interfaces of these two versions are a bit different in the inputs, a component must be added to the workflow template to convert a file from the BAS data type into the BAS6 data type.

This scenario motivates the additional requirements:

- **R10** – Scientists should be able to define the data types of inputs, outputs and parameters of workflow components. This would help to identify whether distinct workflow components have compatible interfaces, also helping to find components by input, parameter and output types.
- **R11** – Scientists need to find workflow components for specific data type conversions.

[1] https://water.usgs.gov/ogw/modflow-nwt/

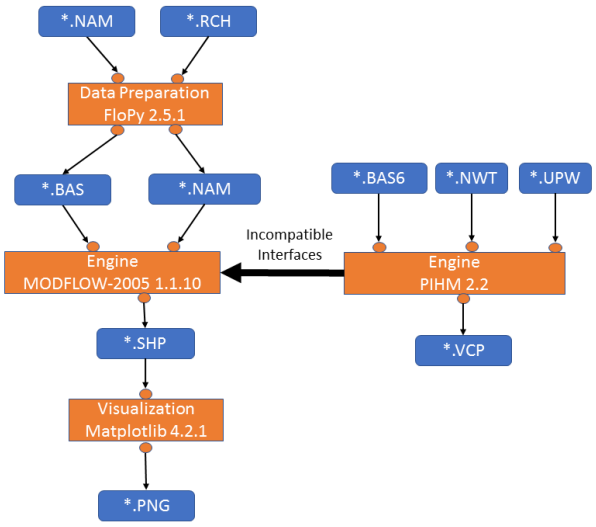

Figure 3. Replacing a component with another component with incompatible interface.

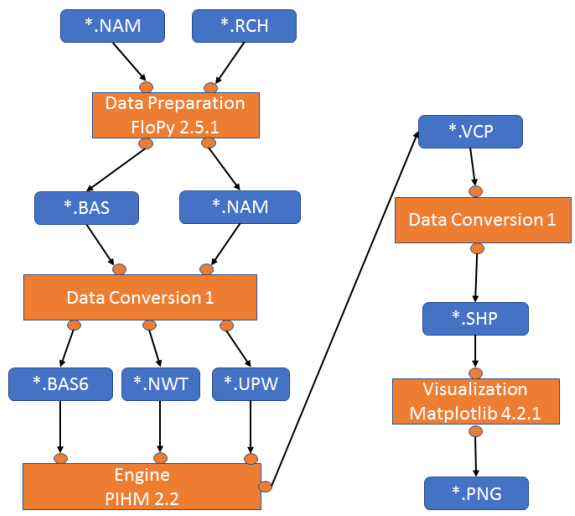

Figure 4. Workflow variant after addition of new components for analysis and data conversion.

3.4 Scenario 3: different software with incompatible interfaces

In this scenario, a scientist needs to replace the software used in a workflow component with a distinct software which performs equivalent functionality.

Here, the interfaces are incompatible due to differences in data types, but also due to difference in the number of input or outputs between the components. Thus, the scientist needs additional components in the workflow template to create and prepare the required files for this replacing component. Figure 3 shows a workflow with a component implemented using MODFLOW-2005 [2] version 1.1.10 that is replaced with a component that uses PIHM[3] version 2.2. The replacing component is incompatible with the previous interface, since the previous component has as data types BAS and NAM for inputs and SHP for output, while the new component has data types BAS6, NWT and UPW as inputs and VCP as output. Figure 4 shows the workflow variant with the new component and additional components for data conversion.

A key challenge regarding this scenario is to identify critical components from the ones that can be removed when updating the workflow to create the workflow variant shown in Figure 4.

Besides all the previous requirements, this scenario motivates the following ones:

- **R12** – Scientists need to know which are critical components in workflows and non-critical components that could be removed if they are no longer needed after in the workflow variant.
- **R13** – The system should be able to verify and remove redundant components or no longer needed inputs, outputs or parameters after updates in workflows.

3.5 Requirements summary

The requirements of the previous scenarios can be grouped into five main categories:

- **Software and version information requirements**, which tackle the representation and metadata of software regarding its interface (i.e., functions/methods, data inputs, data outputs, parameters and data types) and functionalities.
- **Semantics requirements,** which address the representation and identification of data types for inputs, outputs and parameters and abstract types for workflow components and workflow templates.
- **Workflow update requirements**, which address the replacement or change of a workflow component in the workflow template, and the propagation of changes needed to keep the workflow valid.
- **Versioning requirements,** which address the representation and capture of versions for software and representation and creation of versions for workflow components and workflow templates.
- **Comparison requirements**, which address the comparison between distinct software, software versions, workflow template versions and workflow executions.

Table 1 summarizes the requirements pointing the categories, scenarios and steps they belong to.

[2] https://water.usgs.gov/ogw/modflow/mf2005.html
[3] http://www.pihm.psu.edu/

4 DISCUSSION AND RESEARCH DIRECTIONS

Considering the gaps in the literature regarding workflow variants and the requirements from the scenarios, we outline the following possible research directions for future work:

Ontology development. This includes the creation and adaptation of existing ontologies to capture information about software versions and variants, including software interfaces and features. OntoSoft [Gil et al., 2015] is an ontology that might be extended to capture relevant information about software versions and variants. Also, it worth the investigation of how to integrate these ontologies with workflow systems to help in managing experiments and supporting a suggestion mechanism for creation of workflow variants.

Comparison explanation. This includes how to compare software versions and variants regarding information about features and interfaces, and present these results in a useful way to scientists understand the difference between them. A possible approach might be the use of narratives to explain in a human-readable way the reasons why two software versions, software variants, two workflow versions, or two functions/method are different, related or similar. More importantly, these narratives should be easily customized to the reader's level of expertise and interest. As a starting point our approach may be based in an approach for data narratives [Gil and Garijo 2017].

Workflow variants. This includes how to leverage workflow reuse and composition to support the creation of workflow variants. For example, given a set of components and a component that need to be changed in a workflow, how to suggest components to change this workflow. This can be used in a suggestion mechanism for workflow variants. Other investigation is about how to analyze workflow variants to remove duplicate or no longer needed components, inputs, outputs and parameters, and is associated with investigation to define a semi-automatic

Nº	Category	Requirement	Scenarios	Step
R1	Software and version information	Scientists should be able to inform what are the software interfaces (i.e., inputs, outputs, data types, data formats and functions/methods.	All	Pre-update
R2	Comparison	Scientists should be able to compare the software function/method interfaces with other versions and software to understand the differences between them.	All	Pre-update
R3	Software and version information	Scientists should be able to inform relevant information that influence the software operation (e.g., solver used).	All	Pre-update
R4	Software and version information	Scientists should be able to inform what software, software version, function or method are used in workflow component implementations.	All	Pre-update
R5	Comparison	Scientists should be able to compare workflow results between different workflow executions to assess the updates performed in workflows components and templates.	All	Post-update
R6	Semantics	Scientists should be able to define workflow templates using abstract components and give an abstract type to workflow components according to the kind of computations performed by them.	All	Pre-update
R7	Workflow update	Given a software or function/method, scientists need to find workflow components implemented with newer versions or similar software or functions/methods.	All	Post-update
R8	Versioning	Scientists should be able to update existing workflow components, thus, creating a new version for them.	All	Update
R9	Versioning	Scientists should be able to update existing workflow templates, thus, creating a new version for them.	All	Update
R10	Semantics	Scientists should be able to define the data types of inputs, outputs and parameters of workflow components.	All	Pre-update
R11	Workflow update	Scientists need to find workflow components for specific data type conversions.	S2, S3	Update
R12	Workflow update	Scientists need to identify the relevant components in the workflow and the ones that could be removed if they are no longer needed after updating the workflow.	S3	Update
R13	Workflow update	The system should be able to verify and remove redundant components or no longer needed inputs, outputs or parameters after updates in the workflow.	S3	Update

Table 1. Summarization of requirements from scenarios.

mechanism to identify critical and non-critical components in workflows. The critical and non-critical components could be associated to abstractions defined as motifs [Garijo et al. 2014].

It is clear from this paper that work need to be done to easy the continuous creation of workflow variants, for example, to use a newer software release or a distinct function/method in a component of an existing workflow. Tackling this problem will encourage the adoption of scientific workflow systems by scientists, since it may help them to explore new methods and software versions and variants in their experiments.

ACKNOWLEDGMENTS

This work was supported in part by a grant from the US National Science Foundation under award ICER-1440323, and in part by the Sao Paulo Research Foundation (FAPESP) under grants 2017/03570-3, 2014/23861-4 and 2013/08293-7.

REFERENCES

[Altintas et al., 2006] Altintas, I., Barney, O., and Jaeger-Frank, E. (2006). Provenance collection support in the kepler scientific workflow system. In International Provenance and Annotation Workshop, pages 118–132. Springer.

[Freire et al., 2006] Freire, J., Silva, C. T., Callahan, S. P., Santos, E., Scheidegger, C. E., and Vo, H. T. (2006). Managing rapidly-evolving scientific workflows. In Provenance and Annotation of Data, pages 10–18. Springer.

[Freire et al., 2008] Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. Computing in Science & Engineering, 10(3):11–21.

[Garijo et al. 2014] Garijo, D., Alper, P., Belhajjame, K., Corcho, O., Gil, Y., & Goble, C. (2014). Common motifs in scientific workflows: An empirical analysis. Future Generation Computer Systems, 36, 338-351.

[Garijo and Gil, 2011] Garijo, D. and Gil, Y. (2011). A new approach for publishing workflows: abstractions, standards, and linked data. In Proceedings of the 6th workshop on Workflows in support of large-scale science, pages 47–56. ACM.

[Gil and Garijo, 2017] Gil, Y. and Garijo, D. (2017). Towards Automating Data Narratives. In Proceedings of the Twenty-Second ACM International Conference on Intelligent User Interfaces (IUI-17), Limassol, Cyprus.

[Gil et al., 2011] Gil, Y., Ratnakar, V., Kim, J., González-Calero, P. A., Groth, P., Moody, J., and Deelman, E. (2011). Wings: Intelligent workflow-based design of computational experiments. IEEE Intelligent Systems, 26(1):62–72.

[Gil et al., 2015] Gil, Y., Ratnakar, V., & Garijo, D. (2015). OntoSoft: Capturing scientific software metadata. In Proceedings of the 8th International Conference on Knowledge Capture (p. 32). ACM.

[Koop et al. 2011] Koop, D., Scheidegger, C. E., Freire, J., & Silva, C. T. (2011). The Provenance of Workflow Upgrades. In Provenance and Annotation of Data and Process: Third International Provenance and Annotation Workshop, Troy, NY, June 15-16, 2010, Revised Selected Papers (Vol. 6378, p. 2). Springer Science & Business Media.

[Koop 2016] Koop, D. (2016). Versioning Version Trees: The provenance of actions that affect multiple versions. In International Provenance and Annotation Workshop (pp. 109-121). Springer International Publishing.

[Ogasawara et al., 2009] Ogasawara, E., Rangel, P., Murta, L., Werner, C., and Mattoso, M. (2009). Comparison and versioning of scientific workflows. In Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models, pages 25–30. IEEE Computer Society.

[Wolstencroft et al., 2013] Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., Bhagat, J., Belhajjame, K., Bacall, F., Hardisty, A., Nieva de la Hidalga, A., Balcazar Vargas, M. P., Sufi, S., and Goble, C. (2013). The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Research, 41(W1):W557–W561.

[Zhao et al. 2012] Zhao, J., Gomez-Perez, J. M., Belhajjame, K., Klyne, G., Garcia-Cuesta, E., Garrido, A., ... & Goble, C. (2012, October). Why workflows break—Understanding and combating decay in Taverna workflows. In E-Science (e-Science), 2012 IEEE 8th International Conference on (pp. 1-9). IEEE.