DistillFlow: removing redundancy in scientific workflows

Jiuqiang Chen, Sarah Cohen-Boulakia, Christine Froidevaux, Carole Goble, Paolo Missier, Alan Williams

DistillFlow: Removing redundancy in Scientific Workflows

[Demonstration proposal]

Jiuqiang Chen
School of Information Science and Engineering
Lanzhou University, China
LRI CNRS UMR 8623
Université Paris Sud
AMIB group, INRIA Saclay
91405 Orsay, France
chenj@lri.fr

Sarah Cohen-Boulakia
LRI CNRS UMR 8623
Université Paris Sud
AMIB group, INRIA Saclay
91405 Orsay, France
cohen@lri.fr

Christine Froidevaux
LRI CNRS UMR 8623
Université Paris Sud
AMIB group, INRIA Saclay
91405 Orsay, France
christine.froidevaux@lri.fr

Carole Goble
University of Manchester
United Kingdom
carole.goble@manchester.ac.uk

Paolo Missier
University of Newcastle
United Kingdom
paolo.missier@ncl.ac.uk

Alan R Williams
University of Manchester
United Kingdom
alanrw@cs.man.ac.uk

ABSTRACT

Scientific workflows management systems are increasingly used by scientists to specify complex data processing pipelines. Workflows are represented using a graph structure, where nodes represent tasks and links represent the dataflow. However, the complexity of workflow structures is increasing over time, reducing the rate of scientific workflows reuse. Here, we introduce DistillFlow, a tool based on effective methods for workflow design, with a focus on the Taverna model. DistillFlow is able to detect "anti-patterns" in the structure of workflows (idiomatic forms that lead to over-complicated design) and replace them with different patterns to reduce the workflow's overall structural complexity. Rewriting workflows in this way is beneficial both in terms of user experience and workflow maintenance.

Availability: DistillFlow is available for use at http://www.lri.fr/∼chenj/DistillFlow

General Terms

Scientific workflows structures; Refactoring; Anti-patterns

1. INTRODUCTION

Scientific workflows management systems [5, 3, 4] are increasingly used to specify and manage in-silico experiments to easily design complex data processing pipelines. However, while the number of available scientific workflows is increasing, workflows are not (re)used and shared as much as they could be [6, 2]. Several causes for the limited workflow reuse have been identified (e.g., [7]). In the present work we focus on the complexity of workflow structure – that involves the number of nodes and links but is also related to intricate workflow structure features – as the important cause that we want to consider.

In recent prior work [1], we have proposed a workflow refactoring approach that aims at automatically detecting parts of the workflow structure which can be simplified by removing explicit redundancy (we call these idiomatic structures "anti-patterns"). The workflow rewriting process which reduces the complexity is then performed without altering the original workflow semantics (defined as the ability of both workflows to provide the same outputs given the same inputs). Our main contention is that such a reduction in complexity can be performed automatically, and that it will be beneficial both in terms of user experience and operational efficiency. Extensive comparison of our approach to related work is provided in [1].

In this companion demo paper, we introduce the DistillFlow system which implements a full refactoring approach of the workflows from the Taverna workflow management system [5], which for the past ten years has been popular especially within the bioinformatics community.

In the remainder of this paper we summarize the main principles of the refactoring approach we propose (Section 2); Section 3 introduces the architecture of DistillFlow while Section 4 provides the main functionalities of the tool that will be highlighted during the demonstration. We conclude the paper in Section 5.

2. REFACTORING

The aim of our approach is to provide a refactoring procedure to reduce the complexity of workflows. More precisely, our goal is to replace several occurrences of the same processor with one single occurrence whenever possible, while

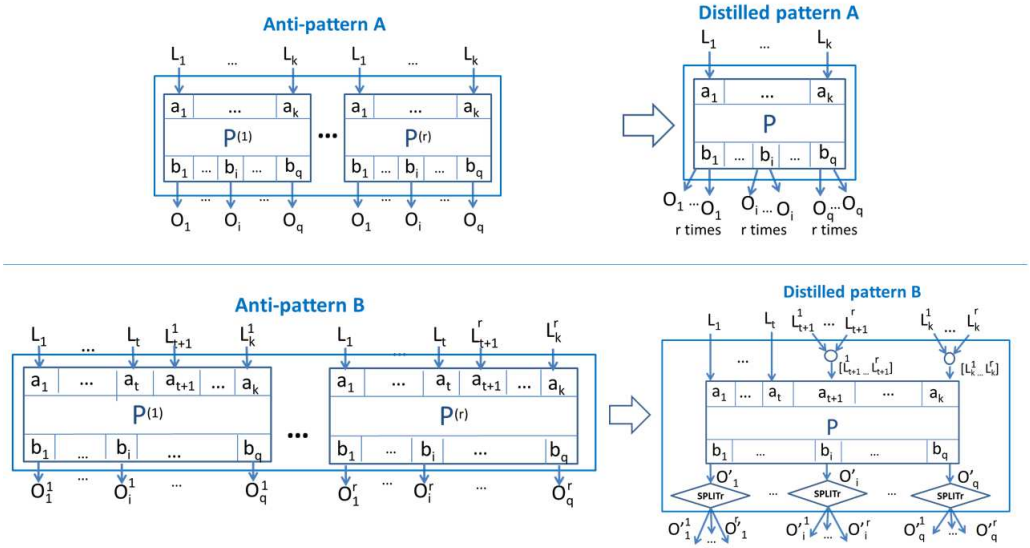

Figure 1: Rewritting rules for anti-patterns of kind A and B.

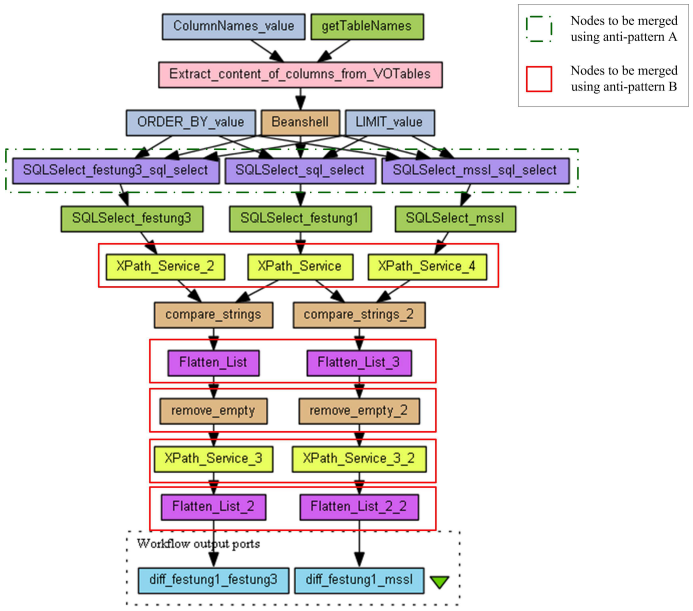

Figure 2: Example of workflow where anti-patterns of kind A and B have been highlighted

following several assumptions: (i) the processors we consider are deterministic (the same output is produced given the same input); (ii) only processors implemented using the exact same code can be merged (in our setting, two processors are equivalent if they represent identical web service calls, or they contain the same script, or they are bound to the same executable Java program). In practice, condition (ii) is often realized, because processors are duplicated during workflow design by means of a graphical "copy and paste" operation. Figure 1 introduces the two generic patterns and the transformation rules we have so far considered ; these rules allow the rewritting of anti-patterns into patterns without redundancy. Figure 2 provides an example of workflow where equivalent processors have been highlighted.

More precisely, anti-pattern A deals with simple redundancy where the same processor P appears with r occurrences (denoted $P^{(1)}...P^{(r)}$ in the figure) and each occurrence takes the exact same input (and can be thus replaced by one single occurrence). Anti-pattern B depicts a more complex case where the various occurrences of the same processor have only part of their inputs in common. In the rewritten pattern, input data that differ from one occurrence to another (L^l_{t+1} to L^l_k) have been merged using the merge processors provided by Taverna (the circle icon) to construct lists of data from the original data items to exploit the implicit it-

erative process of Taverna. As a consequence, the rewritten pattern contains a *list split* processor called $SPLIT_r$ to decompose the list obtained as output into r pieces to ensure that the downstream fragment of the workflow remains unchanged.

The DistillFlow algorithm removes as many anti-patterns as possible, following a recursive process. This paper omits the details about the way we carefully apply rewriting rules and how we consider the various strategies followed by the processors when dealing with lists of input data (cross and dot strategies, see [1] for more information on this point).

3. DISTILLFLOW ARCHITECTURE

The architecture of DistillFlow, implemented in Java, is introduced in Figure 3. The process of transforming a Taverna workflow is described as follows. The user provides the specification of the workflow to be considered. The *TavernaLoader* module is then responsible for loading the workflow into the DistillFlow internal graph structure. The *Anti-pattern Checker* module determines whether or not the workflow taken in contains anti-patterns and provides a report with graph features, including the identification of anti-patterns (if any) and a list of anti-patterns that the system recommends to remove. The user can interact (using the *UserCollaboration* module) with each item of such a list to visualize the corresponding information on the specification graph. If the workflow contains anti-patterns, then the list of anti-patterns selected by the user (possibly all anti-patterns) is sent to the *Anti-pattern Remover* module which removes them. The user can visualize the workflow obtained and decide to stop considering additional anti-patterns. When the user is fine with the workflow obtained, the *TavernaLoader* module produces the rewritten workflow into the Taverna XML format.

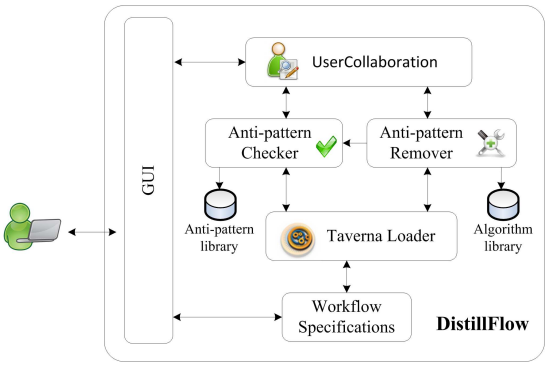

Figure 3: Architecture of DistillFlow

Our implementation is flexible and includes both a library of anti-patterns and a library of rewriting rules (algorithms) making it possible and easy to consider more anti-patterns and rewriting rules in the future.

4. DEMONSTRATION SKETCH

Users communicate with DistillFlow by loading and interacting with original and rewritten workflows. The demonstration will present the following functionalities of the system, based on the workflow of Figure 2.

Loading Data. Users start using DistillFlow by loading a workflow specification into the system (see Figure 4). The original picture of the workflow from myExperiment is displayed by DistillFlow if available (panel (4)), together with a report on graph features (metadata on the workflow, panel (2)). The anti-patterns are determined and the list of anti-patterns displayed in panel (3) while using colors to help users easily distinguish different anti-patterns on the workflow graph ("brown" is related to anti-pattern A while "purple" is related to anti-pattern B). The same color is systematically used on the workflow (to display the nodes involved in an anti-pattern) and on the text listing the anti-patterns (panel (3)).

Refactoring the workflow. Anti-patterns can be removed either all-at-once or in a step-by-step process (selected by the user).
Refactoring all-at-once: DistillFlow allows to automatically remove all the anti-patterns by clicking on the "Remove All anti-patterns" main button (top of Figure 4).
Selecting anti-patterns in collaboration with the user: DistillFlow allows the selection of which anti-patterns users want to be removed. By clicking on the anti-patterns information (panel (3)), DistillFlow highlights the corresponding processors on the graph (panel (1)). Then an operation menu (panel (3)) allows the user to perform the remove operations on the anti-patterns.
Once all the selected anti-patterns have been removed, the user can click on the *Result Overview* button (top of Figure 4). The window entitled "Result Overview" depicted in Figure 5 appears. The original (panel (a1)) and distilled workflows (panel (b1)) are displayed.

Interacting with the initial and distilled workflows. To understand which changes have been done between the initial and distilled workflows, users can interact with the two workflows depicted on Figure 5. By clicking on a node of one graph (initial or transformed graph), DistillFlow highlights the "corresponding" processors in the other graph (it shows the correspondence between a set of occurrences of a given processor P in the original graph and a set of occurrences of the processor P in the (possibly partly) distilled workflow). A detailed report of all the anti-patterns is also displayed (panel (d)). By clicking on the items of the anti-pattern information panel, the corresponding processors will be highlighted on the initial workflow and on the rewritten workflow. As an example in Figure 5, the user has clicked on anti-pattern "7-8-9" in the anti-pattern information panel (d) which has automatically highlighted the corresponding anti-pattern both in the initial workflow (panel (a1), in which three nodes are involved) and in the distilled workflow (panel (b1), in which the anti-pattern has been removed by the system, merging nodes "7", "8" and "9", resulting in only one vertex, numbered "7").

***Running the distilled workflow*:** Any workflow distilled by DistillFlow can be opened and run with Taverna. In our demonstration, we will show that the original and transformed workflows have the same *semantics*: given the same input, they provide the same output using the Taverna engine.

5. CONCLUSION

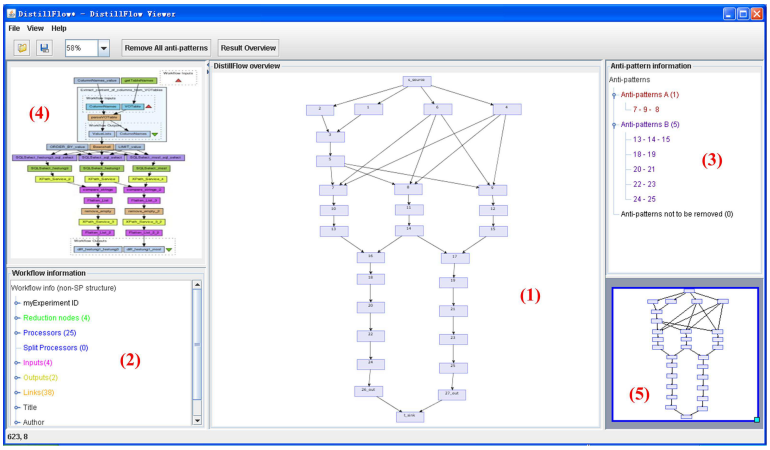

Figure 4: Loading a workflow in DistillFlow and visualizing the set of anti-patterns detected.

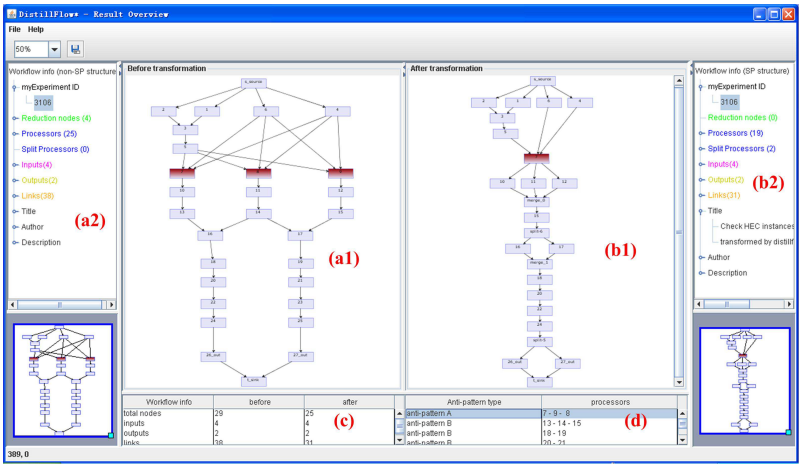

Figure 5: Visualizing both initial workflow and distilled workflow.

The paper introduces a demonstration proposal for DistillFlow, a tool able to aid workflow designers in the minimization of explicit redundancy in scientific workflows. Our tool is available to the community and it is effective in that it is able to take in workflows from the Taverna system and produces to its turn runnable workflows in the Taverna system.

Recent keynote talks, tutorials and research papers from the scientific database community show that *scientific workflows* play a crucial role in data integration and *graph-structured* data are increasingly complex to deal with. Techniques to reduce the complexity of graph structures and improve the readability and maintenance of scientific workflows are thus of increasing interest. This demonstration is at the intersection of these topic areas, and provides a refactoring tool that is interesting to the scientific database community.

6. REFERENCES

[1] S. Cohen-Boulakia, J. Chen, P. Missier, C. Goble, A. R. Williams, and C. Froidevaux. Distilling structure in taverna scientific workflows: a refactoring approach. *BMC Bioinformatics*, 15(Suppl 1):S12, 2014.

[2] S. Cohen-Boulakia and U. Leser. Search, adapt, and reuse: the future of scientific workflows. *SIGMOD Record*, 40(2):6–16, 2011.

[3] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):438–462, 2011.

[4] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. B. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurr Comput*, 18(10):1039–1065, 2006.

[5] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, reloaded. In *Proc. of SSDBM*, volume 6187 of *LNCS*, pages 471–481, 2010.

[6] J. Starlinger, S. Cohen-Boulakia, and U. Leser. (re)use in public scientific workflow repositories. In *Proc. of SSDBM*, volume 7338 of *LNCS*, pages 361–378, 2012.

[7] J. Zhao, J. M. Gómez-Pérez, K. Belhajjame, G. Klyne, E. García-Cuesta, A. Garrido, K. M. Hettne, M. Roos, D. D. Roure, and C. A. Goble. Why workflows break - understanding and combating decay in taverna workflows. In *Proc of e-Science*, pages 1–9. IEEE Comp. Society, 2012.