Implicit Provenance Gathering through Configuration Management

Vitor C. Neves, Vanessa Braganholo, Leonardo Murta
Instituto de Computação
Universidade Federal Fluminense - UFF
Niterói, Rio de Janeiro, Brazil
{vcneves, vanessa, leomurta}@ic.uff.br

***Abstract*—Scientific experiments based on computer simulations usually consume and produce huge amounts of data. Data provenance is used to help scientists answer queries related to how experiment data were generated or changed. However, during the experiment execution, data not explicitly referenced by the experiment specification may lead to an implicit data flow missed by the existing provenance gathering infrastructures. This paper introduces a novel approach to gather and store implicit data flow provenance through configuration management. Our approach opens some new opportunities in terms of provenance analysis, such as identifying implicit data flows, identifying data transformations along an experiment trial, comparing data evolution in different trials of the same experiment, and identifying side effects on data evolution caused by implicit data flows.**

***Index Terms*—implicit data flows, provenance, scientific workflow, scientific experiment, configuration management.**

I. Introduction

Scientific workflows have been extensively used to represent experiments that are based on complex computer simulations. These scientific workflows consume and produce huge amounts of data. In this context, data provenance [1] helps scientists answer queries related to experiment data transformation (e.g., "How were data generated or changed?"). Similarly to art artifacts provenance, data provenance is the historical information about data ownership and transformations. Such information can be gathered through three main strategies: (1) inspecting the operating system to capture all system calls during workflow execution (not direct related with the workflow notation), (2) inspecting the workflow during its execution to register produced and consumed data, or (3) instrumenting each workflow activity to correlate produced data with the processes responsible for the data production.

However, during the workflow execution, data not explicitly referenced by the workflow specification can be created, changed, or accessed, leading to an implicit data flow [2]. Provenance of such information can be useful to help scientists to identify and understand the influence of these implicit data flows. Unfortunately, most of the existing provenance gathering approaches are not able to capture this kind of provenance [1]–[4]. The few exceptions that capture such kind of provenance do not contextualize it with the activities in the workflow specification [5], [6].

This paper introduces a novel approach to gather and store provenance data related to implicit data flow through Configuration Management (CM). CM is a discipline used for controlling software evolution [7]. It is capable of identifying and registering changes on configuration items (i.e., artifacts under CM), relating these changes with the issue that motivated them. By seeing experiment data as configuration items, it is possible to identify some similarities between provenance management and CM. Indeed, provenance is about identifying changes on experiment data (configuration items, in the CM terminology) that were motivated or generated by experiment activities (issues, in the CM terminology). Looking through this perspective, CM becomes a promising approach to provenance management.

Our approach goes in this direction, using the workflow specification as a source of issues to be tracked, and a distributed version control system (VCS) as a CM tool for gathering implicit data flow provenance. This allows tracking changes on files, for each experiment execution, identifying which activities were responsible for these changes. This way, tracked changes include not only explicit data flows, but also the implicit ones, which are our main motivation.

This paper is structured as follows. Section 2 presents a discussion about provenance and implicit data flow. Then, Section 3 defines implicit provenance and presents our vision of provenance management through CM. Section 4 describes our approach, introducing a usage example. Section 5 presents related work. Finally, Section 6 concludes this paper with a discussion about future work.

II. Background and Problem Statement

There are two forms of provenance in the context of scientific experiments [1]: prospective and retrospective provenance. Prospective provenance refers to the workflow specification. It specifies the activities (i.e., steps or tasks) to be executed to generate the expected results. On the other hand, retrospective provenance refers to the workflow execution. It captures executed activities and information about the

environment used for data transformation. It works as an experiment execution log.

Provenance gathering mechanisms can be classified in three different classes [1]: workflow-based, activity-based, or operation system-based (OS). Workflow-based mechanisms are the built-in infrastructure for provenance gathering in Scientific Workflow Management Systems (SWfMS). In this case, the SWfMS becomes responsible for gathering provenance information. By knowing the workflow specification and controlling its execution, these approaches are capable of capturing both prospective and retrospective provenance. However, they are SWfMS dependent, making it difficult to integrate provenance collected from different workflows that compose the same experiment. Moreover, these approaches can only capture provenance of data that were explicitly specified in the workflow, thus missing implicit data flows.

Activity-based mechanisms adapt workflow activities, making them able to gather their own provenance. These approaches are capable of gathering both prospective and retrospective provenance and can be SWfMS independent. The cons are the activity instrumentation difficulties and overhead. Some activities are not easily adapted and are wrapped as black boxes. In such situation, the content of an activity and its input and output data may not be explicitly specified, also leading to implicit data flows misses during provenance gathering.

Finally, OS-based mechanisms rely only on the OS environment's ability to capture data and dependencies among processes and data. These approaches are SWfMS independent, do not require any workflow adaptation, and are able to capture provenance even when implicit data flows are in place. However, they only capture retrospective provenance. Gathering provenance trough OS leads to fine-grained information about all system calls and files touched during the workflow execution. The large amount of retrospective data can make these approaches prohibitive when scientists want to understand the experiment trial in terms of the activities specified in the workflow (i.e., prospective provenance).

Despite the best efforts of scientists to specify all the experiment details into the workflow, there are some situations where workflow activities specification are not complete in terms of consumed and produced data. In such situations, some activities' inputs and outputs that are not declared in the specification manifest during the workflow execution. Fig. 1 illustrates an example of such situation.

According to Fig. 1, the workflow activity *UnzipFile* transmits data to activity *ApplyFilter* through a shared file (Img.bmp). The workflow specification, however, does not register this information. Activities *ApplyFilter* and *CutInterestingArea* also exchange information through the same file. Again, the workflow specification does not register this information. Instead, it only specifies that activity *ApplyFilter* sends a pair of values to activity *CutInteresingArea* ({"circle",2}). In this case, other problem arises: the consecutive changes over the same file overwrite data generated by the previously executed activities. Finally, activities *CutInteresingArea* and *IdentifyPhenomenon* illustrate the ideal scenario for provenance management, since the file they use is explicitly defined in the workflow specification ("c:\wksp\ImgCut.bmp"). Even in this last case, some problems may still arise. Again, if the file name does not change for each workflow execution, all the data generated in previous executions will be lost or altered.

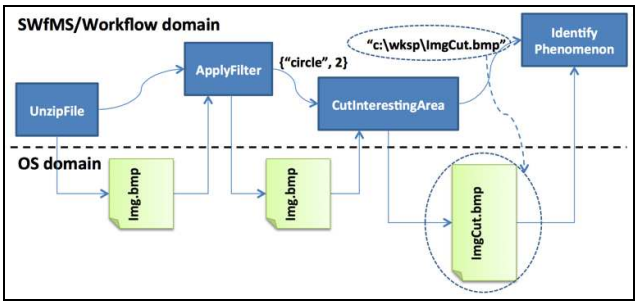

Fig. 1. Implicit provenance example: different domains data flows [2].

Furthermore, Fig. 1 illustrates that two data flows occur in parallel when the workflow specification does not explicitly declare all related data. The first one occurs in the SWfMS/workflow domain, presenting data explicitly declared by the workflow that the SWfMS can collect. The other data flow occurs in the operation system (OS) domain. It is an implicit data flow, since it is not declared in the workflow specification, but it still influences the results of the experiment.

III. Implicit Provenance Management through CM

Implicit data flows can influence experiment results. Since they occur implicitly, their effects may remain hidden from scientists, leading to misleading analyses. Implicit data flow provenance, or simply implicit provenance (ImP), can be useful to help scientists identify or understand possible hidden influence of implicit data flows. In addition, it serves as an evidence of implicit dependencies amongst workflow activities. Identify such dependencies can provide hints on how to distribute an experiment in different sites or even on how remodel it to eliminate this dependence.

The perception of implicit data flows relies on the knowledge of the explicit data flow. In fact, ImP can be defined as the retrospective implicit data flow provenance related to its prospective provenance. Thus, an ImP gathering mechanism must rely on both workflow and OS domains.

According to Koop et al. [4], a tighter integration between scientific workflows and file management (OS domain) is necessary to enable the systematic maintenance of data provenance. Provenance maintenance entails avoiding provenance loss in multiple consecutive trials that overwrite files. An example of such multiple consecutive trials is parameter sweeping, where the same workflow is executed repeatedly with minor changes in its parameters. At the end, the best execution is chosen, and provenance helps to explain the obtained results. This reinforces the necessity of preserving data generated during the workflow execution.

In the CM field, VCS are responsible for managing the different versions of configuration items. Koop et al. [4] state that VCS capture the changes, but not why these changes occurred. Indeed, CM uses another system responsible for

storing such information: issue-tracking systems (ITS). Usually, each issue is associated with the configuration item versions produced during the issue fix. This integrated CM infrastructure can lead to a complete perception of when, how, where, what, why, and by who a configuration item version was created [8]. In the context of workflow provenance data, the workflow specification defines the issues that motivate changing artifacts. Relating provenance data with the workflow activity that generated it is equivalent to relating a VCS check-in with its respective issue in the ITS.

There are four main steps for provenance management through CM: identify the issues to be tracked (prospective provenance register), generate an experiment execution identifier (i.e., trial id), prepare the workspace for the execution, and gather the retrospective provenance, relating it to correspondent prospective provenance. First, prospective provenance must be gathered. Once identified, workflow activities can be treated as issues by retrospective provenance. Then, an experiment identifier must be generated to be possible to identify the tuple {trial id, activity id} that produced the retrospective data during execution. After that, the workspace must be prepared to allow ImP gathering. This preparation involves creating a new workspace or cloning an existing one from a central repository, and generating a branch for the execution. Finally, at the end of each activity a check-in must be triggered to register the new version of every changed artifact, relating it to the tuple {trial id, activity id}.

IV. The ProvMonitor Approach

This vision of provenance management through CM motivated us to conceive a novel approach called *ProvMonitor*. This provenance gathering mechanism uses Git [9] VCS to capture and store retrospective provenance, including ImP. The ProvMonitor implementation is integrated with ProvManager [3], an activity-based provenance gathering mechanism that works through automatic workflow instrumentation. This way, each activity becomes responsible for gathering its own provenance.

A. ProvManager

During the workflow instrumentation, the ProvManager approach collects prospective provenance and automatically wraps the workflow activities into composite activities. These composite activities consist on the original activity and some provenance gathering activities (PGA). The PGAs are responsible to gather retrospective provenance during the workflow execution.

It works as follows: the scientist uploads the workflow specification to the ProvManager. The prospective provenance is gathered while workflow activities are automatically wrapped with PGAs. Then, scientists download the instrumented workflow specification, which is used to execute the experiment. During the workflow execution, the wrapped activities gather the retrospective provenance through the PGA and send it via Web Services to the ProvManager central provenance repository. As discussed, ProvManager alone is not capable of gathering ImP. Actually, even the content of files referenced in the workflow specification is missed. Our approach introduces ImP aware PGAs in ProvManager.

B. ProvMonitor

During prospective provenance gathering, performed by ProvManager, ProvMonitor instruments the workflow with CM commands embedded in the PGAs. This allows ProvMonitor to work during the workflow execution, gathering the retrospective provenance and associating it with the previously collected prospective provenance.

Fig. 2 illustrates how ProvMonitor gathers retrospective provenance. At the beginning of the workflow execution, ProvMonitor generates a trial id and prepares the workspace by cloning the needed input files from a central repository and by generating a branch for the current workflow execution. Then, at the end of each activity, ProvMonitor identifies the accessed files and triggers a check-in, relating the accessed and changed files with the activity that triggered the PGA. At the end of execution, the branch is pushed back to the central repository, making it available for further analysis. As ProvMonitor works managing the workspace, every change in the configuration items are caught by the check-ins, even if it is not explicitly defined in the workflow specification.

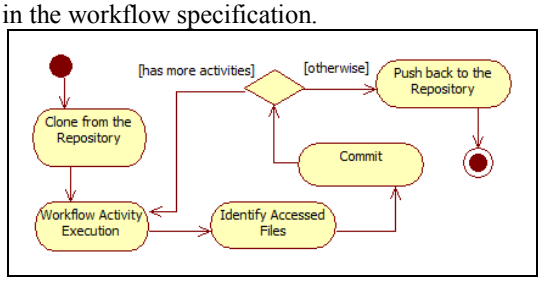

Fig. 2. ProvMonitor gathering process

Fig. 3 presents a usage example of ProvMonitor approach based on the workflow depicted in Fig. 1. As previously discussed, this workflow has implicit data flows that could lead the scientist to a misleading analysis. However, at the end of each activity execution, ProvMonitor catches all changes inside the workspace and identifies each file accessed during the activity execution. For instance, it is possible to observe that the UnzipFile activity created Img.bmp and, subsequently, ApplyFilter changed the same file. This opens opportunity to use specialized diff algorithms [10]–[13] and precisely identify the differences among versions. This kind of analysis is useful to identify the effects of a specific activity over the data.

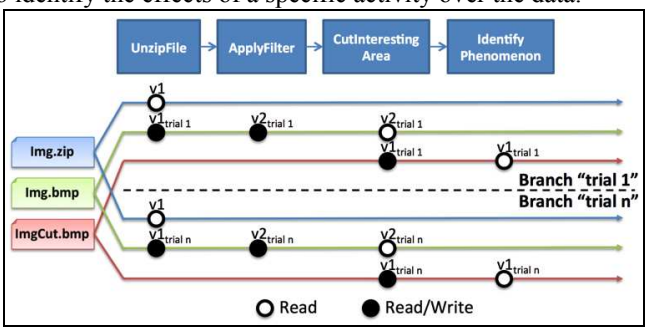

Fig. 3. Activities executions changes through trials

Additionally, the ImP can also be observed across trials. For instance, the before mentioned specialized diff algorithms can be used to compare $v1_{trial\ 1}$ and $v1_{trial\ n}$ of ImaCut.bmp file, both produced by CutInterestingArea activity, but in different experiment trials and, consequently, stored in different branches. This kind of analysis is useful to comprehend the effects of parameter sweeping on intermediate data.

V. Related Work

Until now, ImP has received low attention in the literature. Actually, ImP is missed by existing provenance approaches, except the OS-based gathering approaches [5], [6]. However, they do not relate prospective and retrospective provenance. Furthermore, the fine granularity of their gathering mechanism leads to a very large amount of provenance data. Our approach tries to minimize the amount of data by working into a delimited workspace and being triggered by workflow activities. Doing so, it works at the OS domain, but surrounded by the workflow activities (Workflow domain).

The ProvManager [3] approach, as previously discussed, works by adapting workflow activities. It minimizes the overhead of activities instrumentation via an automatic adaptation process. However, it cannot capture ImP and misses even referenced file content. ProvMonitor is an extension of the ProvManager gathering mechanism that fixes such limitations.

The Strong link approach [4] presents a framework that uses VCS to create strong links between the experiment and its provenance data. It is able to capture referenced file content, but misses ImP when workflow specification does not explicitly reference the files. Strong Links adopts a cache mechanism based on Git to optimize workflow executions. ProvMonitor also uses Git for storing file content separated from the workspace structure, optimizing persistence and leading to file moves detection. The main difference is that we capture retrospective provenance even when it is not specified in the prospective provenance. Furthermore, we do not provide a cache system. Cache systems are interesting to optimize executions in deterministic environments. However, it is risky to suppose determinism in experiments with implicit data flows, since different executions can lead to different results due to implicit data flow side effects.

VI. Conclusion and Future Work

In this paper, we observed the provenance management problem through a new perspective: CM. Moreover, we presented a discussion about implicit data flows and defined the ImP concept. Going beyond, we presented a novel approach to gather ImP using CM.

Our approach opens some new opportunities in terms of provenance analysis, such as identifying implicit data flows, identifying side effects on data evolution caused by implicit data flows, and identifying activities implicit dependencies. Furthermore, as provenance information is gathered relating each change with its correspondent workflow activity on each execution (i.e., trial), it allows intra-trial and inter-trial analysis. Intra-trial analysis compares the data produced by different activities in a specific trial. Inter-trial analysis compares the data produced by a specific activity in different trials.

As future work we plan to conduct experiments on real scientific workflows to better evaluate the use of our gathering mechanism in practice. Then, we plan to develop a query and analysis mechanism to help scientists to explore, on a useful fashion, all the provenance captured by our mechanism. Furthermore, our approach gathers information of every execution and of each execution step. Therefore, we plan to introduce a filtering mechanism to clean up the unnecessary provenance information and focus on the ImP that really contributed to the experiment results.

Acknowledgement

The authors would like to thank CNPq (305276/2010-7 and 305283/2011-1) and FAPERJ (E-26/101.512/2010 and E-26/103.253/2011) for the financial support.

References

[1] J. Freire, D. Koop, E. Santos, and C. Silva, "Provenance for Computational Tasks: A Survey," *Computing in Science & Engineering*, vol. 10, no. 3, pp. 11–21, May 2008.

[2] A. Marinho, C. Werner, M. L. Q. Mattoso, V. Braganholo, and L. G. P. Murta, "Challenges in managing implicit and abstract provenance data: experiences with ProvManager," in *USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, Heraklion, Creta, Grécia, 2011.

[3] A. Marinho, L. Murta, C. Werner, V. Braganholo, S. M. S. Cruz, E. Ogasawara, and M. Mattoso, "ProvManager: a provenance management system for scientific workflows," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1513–1530, Oct. 2011.

[4] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva, "Bridging workflow and data provenance using strong links," in *International Conference on Scientific and Statistical Database Management (SSDBM)*, Berlin, Heidelberg, 2010, pp. 397–415.

[5] J. Frew, D. Metzger, and P. Slaughter, "Automatic capture and reconstruction of computational provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 485–496, 2008.

[6] K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Annual Conference on USENIX*, Boston, MA, 2006, pp. 4–4.

[7] S. Dart, "Concepts in Configuration Management Systems," in *International Workshop on Software Configuration Management (SCM)*, Trondheim, Norway, 1991, pp. 1–18.

[8] C. R. Dantas, L. G. P. Murta, and C. M. L. Werner, "Mining Change Traces from Versioned UML Repositories," in *Brazilian Symposium on Software Engineering (SBES)*, João Pessoa, Brazil, 2007, pp. 236–252.

[9] S. Chacon, *Pro Git*, 1st ed. Berkeley, CA, USA: Apress, 2009.

[10] J. W. Hunt and M. D. McIlroy, "An Algorithm for Differential File Comparison," Bell Laboratories, Murrray Hill, New Jersey, Computing Science Technical Report, 1976.

[11] G. Cobena, S. Abiteboul, and A. Marian, "Detecting Changes in XML Documents," in *International Conference on Data Engineering (ICDE)*, San Jose, CA, USA, 2002, pp. 41–52.

[12] D. Ohst, M. Welle, and U. Kelter, "Differences between versions of UML diagrams," in *European Software Engineering Conference (ESEC)*, Helsinki, Finland, 2003, pp. 227–236.

[13] J. R. Silva Junior, T. Pacheco, E. Clua, and L. Murta, "A GPU-based Architecture for Parallel Image-aware Version Control," in *European Conference on Software Maintenance and Reengineering (CSMR)*, Szeged, 2012, pp. 191 –200.