Managing Provenance in Scientific Workflows with ProvManager

Anderson Marinho[1], Leonardo Murta[2], Cláudia Werner[1], Vanessa Braganholo[2], Sérgio Manuel Serra da Cruz[1], Eduardo Ogasawara[1,3], Marta Mattoso[1]

[1]PESC/COPPE – Universidade Federal do Rio de Janeiro
Rio de Janeiro – RJ – Brazil

[2]Instituto de Computação – Universidade Federal Fluminense
Niterói – RJ – Brazil

[3]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
Rio de Janeiro – RJ – Brazil

{andymarinho,werner,serra,ogasawara,marta}@cos.ufrj.br,
{leomurta,vanessa}@ic.uff.br

Abstract. Running scientific workflows in distributed environments is motivating the definition of provenance gathering approaches that are loosely coupled to the workflow systems. We have proposed a provenance gathering strategy that is independent from workflow system technology. This strategy has evolved into a provenance management system named ProvManager. The main principle is that each workflow activity should collect its own provenance data and publish them in a repository which scientists can access to make their queries. In this paper we show how provenance is captured along distributed heterogeneous systems. Two main strategies are used to capture provenance: using Prolog predicates to register provenance, and using an API for the communication between the wrapped activity and the ProvManager.

1. Introduction

Provenance provides historical information about data manipulated in a workflow (Freire et al. 2008). This historical information tells us how data products were generated, showing their transformation processes from primary input and intermediary data. The management of this kind of information is important since it provides to the scientists a variety of data analysis applications. For instance, from provenance information it is possible to verify data quality of generated data products, since one can look at its ancestral data and determine if they are reliable or not. Other examples are: possibility to audit trails to verify what resources are being used; data derivation capability; experiment documentation; among other applications (Simmhan et al. 2005).

Provenance data must be gathered, modeled, and stored for further queries. Provenance management is an open issue that is being addressed by several workshops and the Provenance Challenge series (ProvChallenge, 2010). One of the open problems is related to which provenance data should be gathered and how. Provenance gathering becomes more complex when the workflow is executed among distributed and heterogeneous execution environments, such as clusters, P2P, grids and clouds. Some scenarios of workflow execution in a distributed environment can be listed (Marinho et al. 2009). Each one has its own characteristics that contribute to the complexity of provenance management. In this paper we focus in a scenario where pre-existing

workflows were conceived independently, using different scientific workflow management systems (SWfMS). However, these independent workflows are integrated to generate a complex experiment, which may entail some additional manual activities to link the workflows. In this scenario, each SWfMS manages provenance information in a decentralized and isolated way, meaning that each system considers provenance in a specific granularity, stores the information on a specific language, or even worse, some SWfMS may not even provide a provenance solution at all.

A solution to manage this heterogeneity is to transfer the provenance management responsibility to an independent system. This system is responsible for capturing, modeling, storing and providing queries to an experiment provenance in an integrated way. Some related work (Cruz et al. 2008, Groth 2006, Simmhan et al. 2006) have the same concerns and share the agnostic strategy of a provenance management system that is independent on a workflow system technology. Besides, there is an initiative (Moreau et al. 2007) working towards a standard model, the Open Provenance Model (OPM), which defines a generic representation for provenance data.

The main difficulty of SWfMS agnostic strategy is that the SWfMS and the provenance management system should communicate to exchange information. In order to make this communication possible, some researches (Simmhan et al. 2006, Munroe et al. 2006, Lin et al. 2008) propose a series of manual activity adaptations over the workflow. However, this solution introduces additional overhead to scientists, which should not have this computational burden. Furthermore, some workflow activities used by scientists are from third-parties, which make their adaptation more complex.

For that reason, in our previous work (Marinho et al. 2009) we have proposed a provenance gathering strategy for experiments that are executed in distributed environments. This strategy is independent on workflow system technology and also tries to address some of the aforementioned problems. This strategy has evolved into a provenance management system named ProvManager. ProvManager leverages the provenance management from an individual workflow to the whole experiment, which may be composed of multiple workflows executed in different SWfMS. In this paper, we contribute by detailing the provenance gathering mechanism of ProvManager.

This paper is organized as follows. Section 2 presents the ProvManager approach, describing its main characteristics. Section 3 presents the ProvManager architecture, characterizing some components and functionalities. Section 4 presents ProvManager in action. Finally, Section 5 concludes the paper.

2. ProvManager

The ProvManager system main focus is to manage provenance into distributed environments. In such scenarios, an integrated analysis of the workflow provenance is complex, since several SWfMS and/or machines may be used and the provenance information is generated and stored in a heterogeneous decentralized way. In order to implement a provenance management system that works in a distributed environment, ProvManager should have the characteristic of being independent of workflow-related technology*, i.e.* independent of any SWfMS, activities scheduler, execution engines, etc. In the literature, there are three provenance gathering approaches in heterogeneous decentralized environments (Freire et al. 2008): workflow, operating system (OS), and activity. At the workflow level approach, a SWfMS is responsible for gathering all the

provenance information. Even though this is the most popular approach, this level has the disadvantage of being dependent on the SWfMS. At the operating system level approach, the provenance mechanisms use OS functionalities for gathering provenance information (*e.g.*: file-system, system call tracer). The advantage of this approach is the SWfMS independency. However, the data collected by these mechanisms are fine-grained and need to be post-processed. At the activity level approach, the provenance mechanism tries to merge the best characteristics of the other two levels. At this level, each workflow activity is responsible for gathering its own provenance information. The advantage of this level is the SWfMS independency, just like the OS level, and more precise information is gathered, just like in workflow level gathering. The problem of this level is the need of adaptation of preexisting activities to incorporate provenance gathering functionalities.

Analyzing the three strategies, we have opted to define our gathering mechanism to be located at the activity level (Marinho et al. 2009). At this level, the independency is preserved but we still have the workflow activity adaptation issue. To solve this, ProvManager provides an automatic mechanism to adapt the workflow activities to support the provenance gathering mechanism. The objective of ProvManager is to isolate the provenance management from the SWfMS and focus on the provenance of the whole experiment. This is possible by means of the integration of provenance data obtained from different workflows that are executed in a distributed environment. Figure 1.a presents an overview of ProvManager in operation. After composing the experiment (*i.e.*, composing all workflows that are part of the experiment), scientists have to configure them to work with the provenance system. This configuration is made automatically by ProvManager at workflow design time, which adapts each workflow activity to support the provenance gathering mechanism. After that, whenever the scientist runs the experiment, all provenance data are transferred from each activity to ProvManager, which is responsible to store them at a provenance repository. Once the experiment is executed, scientists may visualize and analyze provenance data directly through ProvManager.

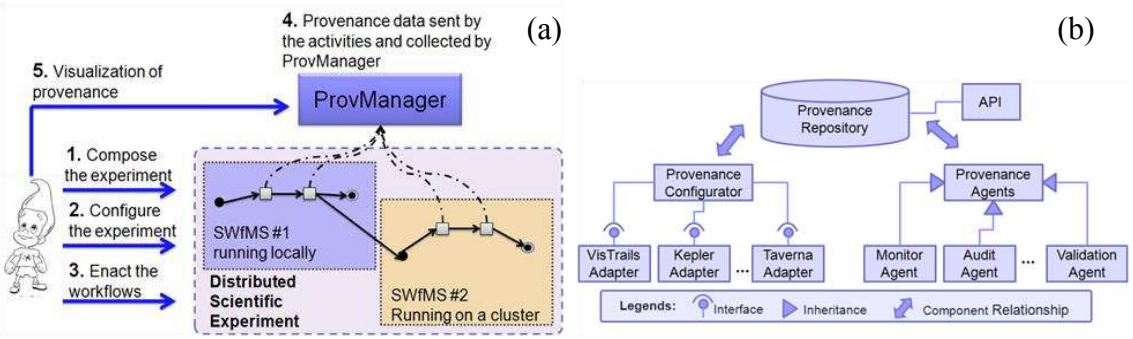

Figure 1. ProvManager in operation (a); Architecture of ProvManager (b).

3. ProvManager architecture

Figure 1.b shows the architecture of ProvManager. It comprises three main components: *provenance repository, provenance configurator,* and *provenance agents*. In this paper, we focus only on the provenance gathering process, which includes the *provenance repository* and *provenance configurator* components. The *provenance agents* are related to the provenance query process, which is outside the scope of this paper.

3.1. Provenance repository

The provenance repository is the component responsible for storing the provenance information. This repository deals with both prospective and retrospective provenance. Prospective provenance is provided by the provenance configurator component, during the workflow adaptation (see Section 3.2), whereas retrospective provenance information is provided by the provenance gathering mechanism that sends information about the workflow execution during runtime.

The provenance data is stored in the provenance repository as Prolog predicates. The choice of Prolog was related to the adoption of OPM, because this model specification indicates that a series of provenance information can be retrieved from inference. Therefore, the use of Prolog is natural, since it natively provides full support for inference. In addition, the provenance repository provides a web services API to allow the communication with the workflow activities, which can be located at different SWfMS or machines. This API helps to make the provenance gathering mechanism more independent on the provenance repository, since the first does not need to know about the provenance repository's implementation and vice versa. The provenance repository API provides the information defined in the provenance model by means of service operations. The service operation signatures are defined at high level, abstracting how the data are stored in the Prolog database. However, internally, each operation maps its functionality into Prolog predicates in order to store the data received in the provenance repository. Figure 2 illustrates the provenance repository API, in which the operations are divided into prospective and retrospective operations.

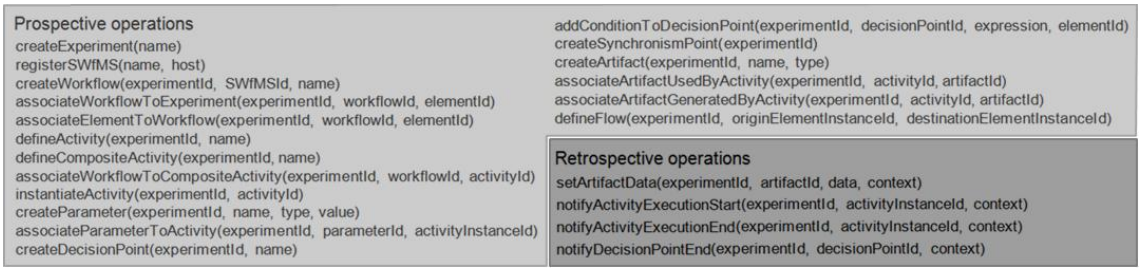

Figure 2. Provenance repository API

3.2. Provenance configurator

The Provanance configurator is responsible for automatically configuring the workflow to support the ProvManager provenance gathering mechanism. It is inspired by the weaving mechanisms of Aspect-oriented Programming (Popovici et al. 2002). The provenance configuration process consists of three steps: (1) the scientist provides the workflow specification to the provenance configurator; (2) the provenance configurator instruments the workflow with the provenance gathering mechanism; (3) during the instrumentation, the provenance configurator extracts the prospective provenance information from the workflow specification and publishes in ProvManager; and (4) a new specification of the adapted workflow is returned to the scientist, which must be reloaded back to the SWfMS.

Analyzing this process, some issues arise. The first one regards the workflow specification, which may be written in different languages, according to the SWfMS being used. Each SWfMS has its own characteristics, such as workflow language specification, execution mechanism restrictions, etc. For that reason, the provenance configu-

rator should be prepared to configure workflows from different SWfMS. Another issue is on how to adapt an activity to support the provenance mechanism. Once again, the answer depends on the SWfMS. Some SWfMS write the activity code in the workflow specification, others work with just services or executables. In other words, it is hard to directly adapt the workflow activity. A solution that is more independent from SWfMS implementation is needed.

To solve the first problem, the provenance configurator architecture was designed to be extensible, supporting SWfMS variations. The provenance configurator component supports the connection of SWfMS adapters. These adapters have the duty of configuring a workflow that was developed in a specific SWfMS. Therefore, to cover a variety of SWfMS, it is necessary to implement specific adapters for each one. In the ProvManager architecture (Figure 1.b), we have, for example, adapters for VisTrails (Callahan et al. 2006), and others SWfMS.

Regarding the activity adaptation problem, a solution is to make it indirectly by means of the construction of wrappers that encapsulate the original activities (Marinho et al. 2009). The wrapper adds the provenance gathering mechanism, which is responsible for gathering the information from the original activity and sending it to the provenance repository via web services API. This API is provided by the provenance repository, as shown in Figure 1.b. Figure 3 shows the structure of an adapted activity, which is composed of the provenance gathering mechanism module and the original activity. It is possible to observe that the adapted activity input and the original activity output data are collected by the provenance gathering mechanism and then stored at the provenance repository.

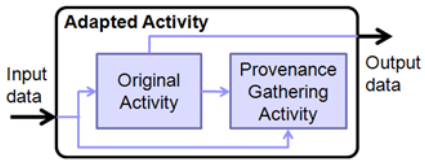

Figure 3. Wrapper structure of an adapted activity

In order to implement the wrapper solution, we opted to use the composite activity concept, which is provided by most SWfMS. A composite activity aggregates activities, forming a sub-workflow. Thus, for each workflow activity, we create a composite activity that contains the original activity and provenance gathering activities (PGA), which perform the provenance capturing. Each PGA is responsible for publishing retrospective provenance information, such as the beginning and ending of an activity execution, the produced artifacts, etc. The retrospective provenance API provided by the provenance repository (illustrated in Figure 2) is used for this purpose by PGA.

4. ProvManager in action

This section shows a small example of ProvManager in action. Figure 4.a illustrates the experiment structure that we use as an example for describing the ProvManager's functionalities. This experiment is segmented in two workflows: one workflow is instantiated in VisTrails, and the other in Kepler (Ludascher et al. 2006). Figure 4.b shows the workflow in VisTrails with more details. The fragment is composed of three activities: *GetData*, *Validate*, and *Simulate,* running on a remote host with IP address 192.168.0.5. In order to capture provenance data from this workflow, the scientist has to publish it in ProvManager, uploading the workflow specification (in the VisTrails case, a .VT file).

At this moment, ProvManager configures the workflow, by automatically adding special activities (PGA) that are responsible for capturing and publishing provenance data in ProvManager during the workflow execution. This process of adding provenance components into the workflow is called instrumentation.

Currently, ProvManager can only instrument workflows executed in Kepler and VisTrails. However, ProvManager works with the concept of plugin to be able to support future extensions to other SWfMS instrumentation mechanisms. Finally, at the end of the instrumentation, a new .VT file is returned to the scientist to be reloaded in VisTrails. During both the instrumentation and execution of the workflow, ProvManager captures provenance data from the workflow and publishes this data in the repository. This repository is a Prolog database, so provenance data are mapped into Prolog predicates. Figure 4.b shows the .VT file mapped into prolog predicates.

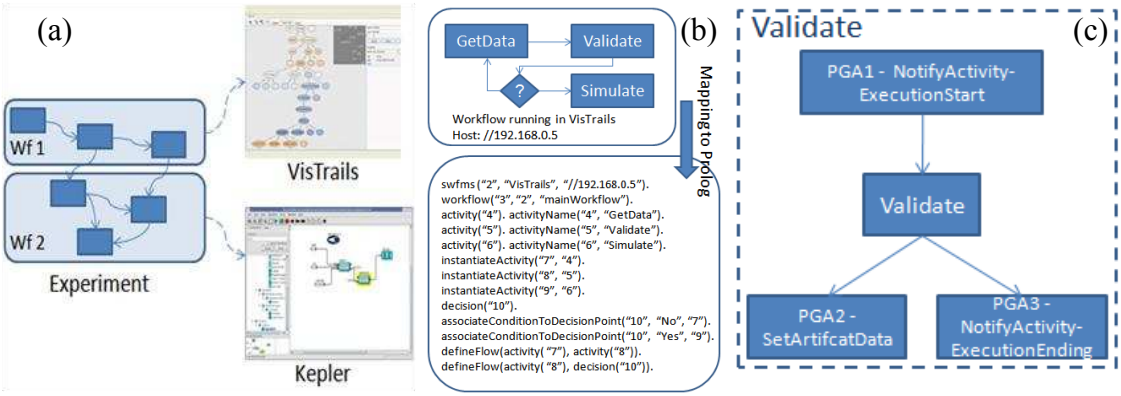

Figure 4. Experiment example (a), VisTrails workflow mapped to Prolog predicates (b), and workflow adaptation in VisTrails (c).

ProvManager has a web interface (http://reuse.cos.ufrj.br/provmanager) which allows scientists to register experiments and analyze their provenance data. To register an experiment in ProvManager the first step the scientist has to do is to create an experiment entity. An experiment entity can be associated with one or more workflows. A workflow in ProvManager represents an experiment segment that is executed by a SWfMS. Workflows are created by the scientist according to the experiment structure. In the example of Figure 4.a, the two workflows, i.e., VisTrails and Kepler, represent the experiment segments. To publish a workflow in ProvManager, the scientist should provide the workflow specification to be adapted. When the workflow is published, the scientist can download the adapted specification in order to actually execute it in the SWfMS, as shown in Figure 5.a. Figure 4.c illustrates some operational details about how the workflow activity *Validate* is adapted using composite activities in VisTrails. A composite activity can be created by using the VisTrails functionality called *Group*. Notice that some PGA are placed before the activity execution (PGA1), and others are placed after it (PGA2 and PGA3). This decision depends on the type of provenance that needs to be gathered. For instance, the PGA agents that use the API operation *notifyActivityExecutionStart* have to be executed before the original activity in the sub-workflow. The opposite happens to the PGA that uses the API operation *notifyActivityExecutionEnd*.

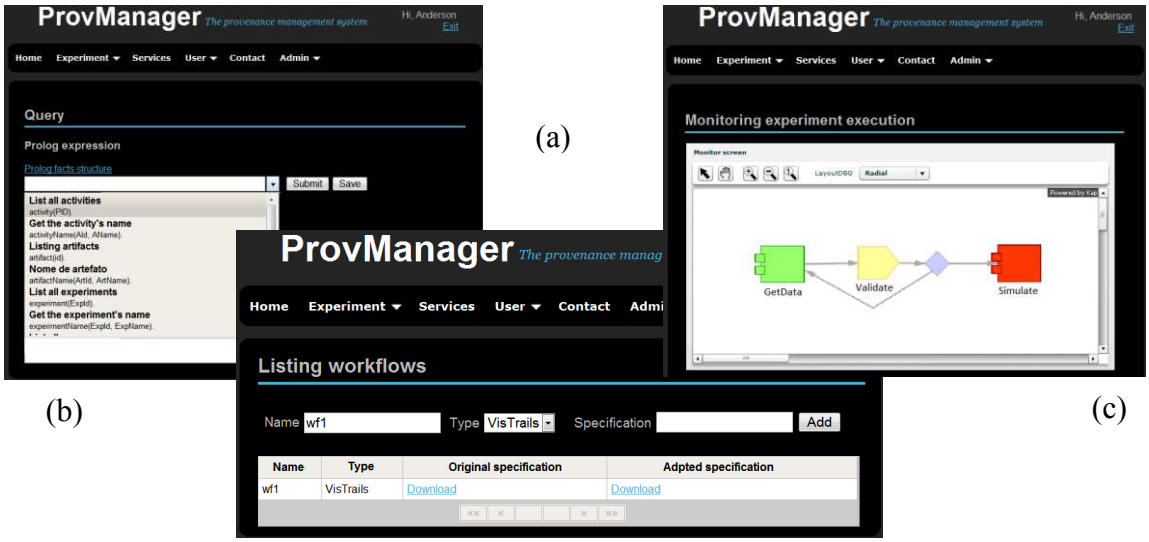

Figure 5. ProvManager's screens: (a) List of workflows from one experiment; (b) Query interface; (c) Execution monitoring.

ProvManager provides a basic query mechanism to visualize the collected provenance data. It uses Prolog as the underlying query language. The query mechanism is illustrated in Figure 5.b. The scientist types a query expression in the input text and the result is returned in the text area below. This mechanism also helps the scientist to type the expression by means of a recomendation mechanism that exhibits possible queries according to what the scientist is typing. These suggestions can be augmented with more expressions since the query mechanism allows for the scientist to save each query (clicking in the *save* button). Besides, this mechanism allows for the scientist to store high level query expressions. Finally, ProvManager provides a mechanism for monitoring the experiment execution. This mechanism is interesting since it provides a global view of the experiment execution to the scientist, saving the scientist's time in visiting each SWfMS to verify whether an experiment fragment was executed or not, as is shown in Figure 5.c.

5. Conclusions

There are a number of provenance systems that manage scientific experiments composed of one or more workflows running either locally or distributed. As far as we are concerned, ProvManager is the only system that benefits from gathering provenance at activity level, offering a loosely coupled approach to SWfMS. Our proposal operates outside the SWfMS. Thus, it does not require modifications at the hosting SWfMS source-code and it can collect either prospective or retrospective provenance data. Besides, further provenance analysis may be achieved by reasoning due to the use of Prolog rules. Furthermore, ProvManager helps to diminish the scientist efforts on configuring provenance gathering mechanism.

The strategy of working with the provenance gathering mechanism at the activity level can cause an overhead on workflow processing since new activities (PGA) are inserted. In addition, depending on the gathered information, these PGA may also consume high network bandwidth, since they have to transfer all generated data from the workflow activities to the provenance repository. An alternative to address this limita-

tion is to store data locally and publish only a pointer to the data at the repository, as suggested by Groth et al. (2006).

ProvManager is currently under development. The provenance repository, the provenance model, and the provenance publishing API are already developed. We are currently working on a higher level query interface, so the scientist will not need to have any knowledge about Prolog. As future work, once ProvManager development finishes, we plan to evaluate our architecture in real scenarios, using bioinformatics and petroleum engineering examples that are being developed at our research group. This evaluation will bring us feedback for future improvements.

Acknowledgments. The authors would like to thank CAPES and CNPq for the financial support.

References

Callahan, S. P., Freire, J., Santos, et al., (2006), "VisTrails: visualization meets data management". In: *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, p. 745-747, Chicago, USA.

Cruz, S., Silva, F., Gadelha Jr, L., et al., (2008), "A Lightweight Middleware Monitor for Distributed Scientific Workflows". In: *CCGrid'2008*, p. 693-698

Freire, J., Koop, D., Santos, E., Silva, C., (2008), "Provenance for Computational Tasks: A Survey", *Computing in Science & Engineering*, v. 10, n. 3, p. 11-21.

Groth, P., Jiang, S., et al., (2006), *An Architecture for Provenance Systems*, Technical report, Electronics and Computer Science, University of Southampton.

Lin, C., Lu, S., Lai, Z., et al., (2008), "Service-Oriented Architecture for VIEW: A Visual Scientific Workflow Management System". In: *Services Computing 2008 (SCC '08)*, p. 335-342, Honolulu, USA.

Ludascher, B., Altintas, I., et al., (2006), "Scientific workflow management and the Kepler system", *Concurrency and Computation*, v. 18, n. 10, p. 1039-1065.

Marinho, A., Murta, L., Werner, C., et al., (2009), "A Strategy for Provenance Gathering in Distributed Scientific Workflows". In: *IEEE 2009 Third International Workshop on Scientific Workflows*, p. 344-347, L.A., USA.

Moreau, L., Freire, J., Myers, J., et al., (2007), *The Open Provenance Model*, Technical report, Electronics and Computer Science, University of Southampton.

Munroe, S., Miles, S., Moreau, L., et al., (2006), "PrIMe: a software engineering methodology for developing provenance-aware applications". In: *Proceedings of the 6th international workshop on Software engineering and middleware*, p. 39-46, Portland, USA.

Popovici, A., Gross, T., Alonso, G., (2002), "Dynamic weaving for aspect-oriented programming". In: *Proceedings of the 1st international conference on Aspect-oriented software development*, p. 141-147, Enschede, The Netherlands.

ProvChallenge, (2010), "Fourth Provenance Challenge, http://twiki.ipaw.info/bin/view/ Challenge/FourthProvenanceChallenge".

Simmhan, Y., Plale, B., Gannon, D., (2005), *A Survey of Data Provenance Techniques*, Technical report, Computer Science Department, Indiana University.

Simmhan, Y., Plale, B., Gannon, D., (2006), "A Framework for Collecting Provenance in Data-Centric Scientific Workflows". In: *International Conference on Web Services*, p. 427-436, Chicago, USA.