Towards Provenance-Enabling ParaView

Steven P. Callahan[1,2], Juliana Freire[1,2], Carlos E. Scheidegger[2], Cláudio T. Silva[1,2], and Huy T. Vo[2]

[1] VisTrails, Inc.
[2] Scientific Computing and Imaging Institute, University of Utah
{stevec,cscheid,csilva,hvo}@sci.utah.edu, juliana@cs.utah.edu

Abstract. Currently, there are no general provenance management systems or tools available for existing applications. Our goal is to develop provenance technology that is flexible and adaptable to the wide range of requirements of software applications. By consolidating provenance information for a variety of applications, we can provide a uniform environment for querying, sharing, and re-using provenance in large-scale, collaborative settings. In this paper, we describe our framework for provenance-enabling existing applications. Our approach is applicable to a variety of software systems that are process driven. As a concrete example, we describe a working plug-in for an open source application in scientific visualization.

1 Introduction

Computers are now extensively used throughout science, finance, engineering, and medicine. Advances in data mining, computational geometric modeling, imaging, and simulation allow researchers, engineers, and artists to build increasingly complex models and generate unprecedented amounts of data. Hedge funds use simulations to construct accurate risk and return assessments for portfolios. Oil & Gas companies heavily depend on simulations for various tasks, including exploration and pipeline transport. Clinical medicine has become increasingly dependent on procedures that include simulations from data acquired directly from the patient through magnetic resonance imaging (MRI), Computed Tomography (CT), and other computerized exams. Even areas of the entertainment industry have been greatly impacted by the use of computers to design complex computer models and scenes for movies and video games. A major problem that these disciplines face is the management of this data and the processes that were used to generate the data.

Currently, ad-hoc approaches for capturing the provenance of exploratory computational tasks are used in the scientific and engineering community. For example, laboratory notebooks are commonly used to track changes in parameters or processes. However, ad-hoc approaches have serious limitations. In particular, scientists and engineers need to expend substantial effort managing data and recording provenance information. The absence of detailed provenance makes it hard (and sometimes impossible) to reproduce and share results, to solve problems collaboratively, to validate results with different input data, to understand

J. Freire, D. Koop, and L. Moreau (Eds.): IPAW 2008, LNCS 5272, pp. 120–127, 2008.

the process used to solve a particular problem, and to re-use the knowledge involved in the data analysis and generation processes. In addition, it limits the longevity of the data products—without precise and sufficient information about how the data product was generated, its value is greatly diminished. The growing demands for compliance to varying industry and governmental regulations and standards also requires detailed audit trails of data sources and workflows (tasks) executed.

Originally motivated by the needs in the scientific domain, the VisTrails provenance technology [3] and the infrastructure it provides is general and applicable to a wide range of applications that involve complex computational processes. Whereas our initial development focused on provenance management for tasks developed within a workflow system, our goal in this paper is to show that the same infrastructure can be used to *provenance-enable existing applications*, without requiring them to be integrated within a workflow system. One of the major advantages of this approach is that users will be able to leverage provenance using the same applications and environments that they are used to.

1.1 Related Work

There are important distinctions that set our work apart from previous approaches to provenance. Notably, our focus is on interactive applications that provide graphical user interfaces. Although there has been previous works on provenance-enabling such applications, these have proposed application-specific solutions (see e.g., [1]). In contrast, the plug-in infrastructure is general and can be integrated with any application that exposes its undo-redo stack.

There has also been work proposing general provenance solutions that can be combined with arbitrary systems. The Earth System Science Workbench (ESSW) uses scripts to wrap legacy systems so that their inputs and outputs can be transparently gathered [4]. The Provenance-Aware Service-Oriented Architecture (PASOA) was designed to support provenance capture in a service-oriented environment [5]. It requires that services be instrumented to produce assertions which detail, for example, how different services interact and which data item they manipulate and derive. Like PASOA and ESSW, our approach also requires applications to be instrumented, however the purpose of this instrumentation is to obtain access to existing applications' undo-redo capabilities. Furthermore, the approaches used in PASOA and ESSW were designed for services and batch-oriented programs. In contrast, our infrastructure can be combined with both interactive and batch oriented system.

2 A Process-Driven Provenance Model

VisTrails introduced a change-based model to capture provenance and display it in a history tree called a *vistrail* [2]. Here we describe a generalized version of this provenance model that is adaptable to a variety of settings.

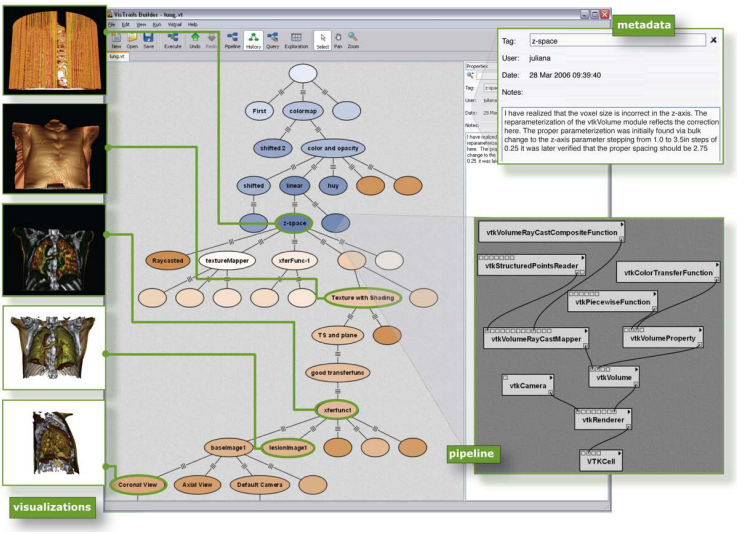

Fig. 1. The version tree stores the complete history of the actions performed by a user. Each node corresponds to a state in the application, the edges show how the actions are ordered to achieve these states.

2.1 Change-Based Provenance

In an application, as the user makes changes to the state of the application through a user interface, the provenance mechanism records those changes. Instead of storing a set of application states, the change-based model stores the operations, or actions, that are applied to the application (e.g., slicing a volume or editing a parameter in a scientific visualization system). This representation is both simple and compact—it uses substantially less space than the alternative of storing multiple instances or *versions* of the state. In addition, it enables the construction of an intuitive interface that allows users to both understand and interact with the history of the application states through these changes. A tree-based view allows a user to return to a previous version in an intuitive way, to undo bad changes, to compare different workflows, and to be reminded of the actions that led to a particular result. Figure 1 shows an example of a vistrail created through computational workflows.

The change actions are represented as a rooted tree VT in which each node corresponds to a *version* of the application state, and each edge between nodes d_p and d_c, where d_p is the parent of d_c, corresponds to the action applied to d_p which generated d_c. This is similar to the versioning mechanism used in Darcs [8]. More formally, let DF be the domain of all possible states of the application, where $\emptyset \in DF$ is a special empty state. Also, let $x : DF \rightarrow DF$ be a function that transforms one state into another, and $\mathcal{D}$ be the set of all such functions. A vistrail node corresponding to a workflow d is constructed by composing a sequence of actions, where each $x_i \in \mathcal{D}$:

$$d = (x_n \circ (x_{n-1} \circ \ldots \circ (x_1 \circ (\emptyset))...))$$

This change based representation is general in that the actions can be captured at different granularities and they can be made to match the semantics of a specific application. In particular, it can be readily applied to create Provenance Explorer plug-ins for existing applications.

3 Capturing, Representing, and Re-playing Provenance

Our change-based representation of provenance is easily incorporated into existing applications that provide a mechanism for controlling the actions that are being performed by a user via a graphical interface. The model-view-controller paradigm [6] is an architectural pattern used in software engineering that decouples the user interface (view) from the domain-specific logic and access (model) using an event processor (controller). This software engineering paradigm is frequently used in large projects to increase the flexibility and reuse of code. As the user interacts with a view that is generated based on the current model, a registered handler or callback is triggered in the controller. The controller then updates the model so that the view can be recreated. Since all the events that are generated by the application pass through one event handler, capturing and replaying then is performed either by modifying this controller directly, or by intercepting and fabricating the events via the callback mechanism.

Our Provenance Explorer is an application that runs along-side the main application. Provenance is captured during user interactions with the main application using a custom solutions for the application. This provenance is passed to and from the Provenance Explorer via a Communication API. The details of these steps are provided in more detail in this section.

3.1 Capturing Actions

The implementation of the action-based provenance in the VisTrails system is specific to the actions that occur while creating and editing workflows in the VisTrails Builder. These actions include adding and deleting modules and connections, and changing parameter values. For other applications, our Provenance Explorer needs to be able to handle a more general action type. Conceptually, the model supports actions at varying granularities or semantic levels, from basic mouse button presses to complex sets of operations (such as copying and pasting a set of actions). The level of granularity that an action may take needs to be application specific.

In general, applications that take advantage of the model-view-controller paradigm have a mechanism for storing and re-using actions: the undo and redo operations. In a scientific visualization system, for instance, with *undo* a user should be able to walk through the steps they took to create an image, albeit backwards. Although undo does not capture the complete exploration process nor does it persist across sessions, it provides valuable context for granularity of actions. The designers of the software have already determined the granularity

of actions by designing the undo stack. The undo stack of an application may individually capture single mouse events or keyboard strokes if they are needed to recreate of the state. Furthermore, interactions performed by the user may cause multiple actions to be performed, which the undo stack will store as one step. We capture actions at the same granularity in which the undo stack does. In fact, in practice it is simpler to capture actions as they are being added to the undo stack instead of where they are handled by the controller. Obviously, this depends on the completeness and availability of the undo/redo mechanism in the application.

In some applications, access to the controller is limited, the undo mechanism captures state instead of actions, or the undo mechanism does not provide the actions that are required for full reproducibility. In these cases, it is necessary to compute actions based on the previous and next states, s_p and s_n, respectively. Using the application's model of the state, the difference $s_p - s_n$ can easily be computed as the set of changes that take s_p to s_n. These changes can then be stored much more efficiently and uniformly as actions in our provenance model.

3.2 Representing Actions

Once the actions have been captured from the application, we use our Communication API to pass them on to our Provenance Explorer, which is an independent application running on its own thread. The Communication API uses sockets to send and retrieve actions from the application's controller to Provenance Explorer's controller. These actions that move across the socket are simply strings that represent the commands that have been captured or are to be executed by the main application. When the Provenance Explorer receives a new command, it creates an action that contains the command along with additional metadata that is either automatically and manually created. Automatically created metadata includes the date and time the command was executed, the user who created it, a unique identifier for the action, and the identifier for the action that preceeds it. Other metadata such as annotation notes or a tag to label the action can be added by the user in the Provenance Explorer interface.

The set of actions stored in the Provenance Explorer, or vistrail, is represented in XML as is described by the following partial schema given in a terse form:

```
type Vistrail =
 vistrail [@version, @id, @name, Action*, annotation? ]
type Action =
 action [ @date, @user, @id, @parentId, @command, tag?,
          annotation? ]
```

This is a more general form of the original VisTrails schema [2] that was used to capture the limited number of actions that are available within the VisTrails Builder (i.e., adding/deleting workflow modules, adding/deleting connections, and changing parameter values). This schema has also been extended to store vistrails in a variety of available relational database management systems as well.

Visually, a vistrail is shown in the Provenance Explorer as a history tree of actions that can be tagged, annotated, and queried using our graphical user interface.

3.3 Re-playing Actions

When the user interacts with our history tree by selecting a version, the Provenance Explorer uses the Communication API to send actions back to the main application. The set of actions to reproduce a version in the tree are serialized by compiling all the commands in each action from the top of the tree to the current selected node. The main application receives these actions, clears the current state, and uses the actions either as a series of events that are executed by the controller or as direct updates to the model state. By returning to a previous version in the history tree, then making changes in the main application, it is possible to branch the tree. In this way, the actions performed by a user are never lost, even though they would be with a normal undo stack.

During interaction with the main application, the user may still want to use undo/redo as is provided by that application. It is important to allow this interaction so that we minimize disruption to the normal workflow of the user. The undo and redo operations can be hijacked so that they trigger the current version in the Provenance Explorer to change by walking up (undo) or down (redo) the history tree. This allows a complete history tree of the provenance to be captured even if the user has visual component of the Provenance Explorer interface disabled.

4 Case Study: ParaView

ParaView [7] is an open-source, multi-platform application designed to visualize data sets of size varying from small to very large. The project started in 2000 as a collaboration between Kitware and Los Alamos National Laboratories. The current version, ParaView 3.0, was released in May 2007. ParaView is quite popular, and is downloaded over 10,000 times a month. The system is used by researchers and engineers in both industry and academia.

Figure 2 shows ParaView together with the Provenance Explorer, transparently capturing the complete exploration process. This Provenance Explorer was implemented by inserting monitoring code in ParaView's undo/redo mechanism, which captures changes to the underlying pipeline specification. Essentially, the action on top of the undo stack is added to the vistrail in the appropriate place, and undo is reinterpreted to mean "move up the version tree". The current version of the Provenance Explorer captures all of the changes to the pipeline. However, some changes of state are not related to the pipeline and ParaView does not store these in the undo stack. For example, the position of the camera is not stored there. In fact, it is quite common for 3D applications to not store navigation in the undo/redo stack (just like word processors typically do not store which page the user is looking at in undo stacks). In this sense, it would

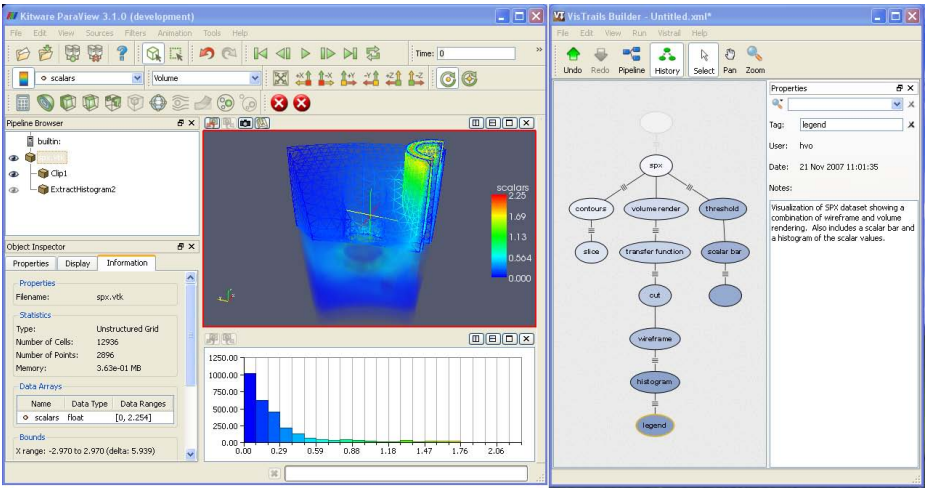

Fig. 2. A screenshot of ParaView (left) with the provenance captured by VisTrails and displayed as a version tree in a separate window (right). This preliminary prototype taps into ParaView undo/redo mechanism to capture the exploration process.

arguably be incorrect to interpret view changes as actions that generate new versions.

If, however, capturing these interactions is really required, more sophisticated approaches are necessary. The latest version of ParaView introduced "Lookmarks", which capture the complete underlying pipeline of a visualization. Unlike in VisTrails, however, Lookmarks need to be manually set by the user during the exploration process. Lookmarks can be serialized, allowing a visualization to be reproduced at a later time. This mechanism for capturing the pipeline and state of the application exposes a wider class of actions for our Provenance Explorer. We are currently implementing a version of the infrastructure that combines the undo/redo stack inspection with Lookmark information, in order to capture this potentially missing information.

5 Discussion

In the VisTrails system, provenance is used for more than version tracking and persistence. Specifically, there are some operations on particular version that can be cast as operations over the set of stored actions. For example, VisTrails allows users to compare two different workflows by looking at a sequence of actions that takes one workflow into the other [3]. This sequence of actions is presented analogously to a workflow, which allows users to look at the result in the same way they look at regular workflows. It would be interesting to extend this principle to third-party applications. For example, the difference between

two visualizations in ParaView should be presented as a single visualization, superimposing and highlighting the differences between the two versions.

VisTrails also allows users to build workflows by analogy [9]. The technique involves identifying the differences between two workflows a and b and remapping this sequence of actions so it can be applied to a different workflow c. It originally involves computing an approximate graph matching between a and c. In a general case, the remapping would have to be specifically tailored for each application, but the general algorithm would still apply.

Finally, our broader goal is to provide a uniform platform for capturing, querying, and reusing provenance from many applications. To this end, we intend to develop the infrastructure that allows other developers to quickly and easily incorporate our Provenance Explorer as a plug-in to their own applications.

Acknowledgments. This work was funded by the Department of Energy grant FG02-08ER85157 and SciDAC (VACET and SDM centers), the National Science Foundation (grants IIS-0746500, CNS-0751152, IIS-0713637, OCE-0424602, IIS-0534628, CNS-0514485, IIS-0513692, CNS-0524096, CCF-0401498, OISE-0405402, CCF-0528201, CNS-0551724, IIP-0712592), and IBM Faculty Awards (2005, 2006, 2007, and 2008).

References

1. Becker, R.A., Chambers, J.M.J.M.: Auditing of data analyses. SIAM Journal of Scientific and Statistical Computing 9(4), 747–760 (1988)
2. Callahan, S., Freire, J., Santos, E., Scheidegger, C., Silva, C., Vo, H.: Managing the Evolution of Dataflows with VisTrails (Extended Abstract). In: IEEE Workshop on Workflow and Data Flow for Scientific Applications, SciFlow (2006)
3. Freire, J., Silva, C.T., Callahan, S.P., Santos, E., Scheidegger, C.E., Vo, H.T.: Managing rapidly-evolving scientific workflows. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 10–18. Springer, Heidelberg (2006)
4. Frew, J., Bose, R.: Earth system science workbench: A data management infrastructure for earth science products. In: Proceedings of SSDBM, pp. 180–189 (2001)
5. Groth, P., Jiang, S., Miles, S., Munroe, S., Tan, V., Tsasakou, S., Moreau, L.: An architecture for provenance systems. Technical report, ECS, University of Southampton (2006)
6. Krasner, G.E., Pope, S.T.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of Object-Oriented Programming 1, 26–49 (1988)
7. Paraview, `http://www.paraview.org`
8. Roundy, D.: Darcs, `http://abridgegame.org/darcs`
9. Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J., Silva, C.T.: Querying and creating visualizations by analogy. IEEE Transactions on Visualization and Computer Graphics 13(6), 1560–1567 (2007)