Managing the Evolution of Dataflows with VisTrails
Extended Abstract

Steven P. Callahan Juliana Freire Emanuele Santos Carlos E. Scheidegger
Cláudio T. Silva Huy T. Vo

University of Utah
vistrails@sci.utah.edu

1 Introduction

Scientists are now faced with an incredible volume of data to analyze. To successfully analyze and validate various hypotheses, it is necessary to pose several queries, correlate disparate data, and create insightful visualizations of both the simulated processes and observed phenomena. Data exploration through visualization requires scientists to go through several steps. In essence, they need to assemble complex workflows that consist of dataset selection, specification of series of operations that need to be applied to the data, and the creation of appropriate visual representations, before they can finally view and analyze the results. Often, insight comes from comparing the results of multiple visualizations that are created during the data exploration process. For example, by applying a given visualization process to multiple datasets; by varying the values of simulation parameters; or by applying different variations of a given process (*e.g.*, which use different visualization algorithms) to a given dataset. Unfortunately, today this exploratory process is far from interactive and contains many error-prone and time-consuming tasks.

Visualization systems such as Paraview[1] and SCIRun[2] allow the interactive creation and manipulation of complex visualizations. These systems are based on the notion of dataflows, and they provide visual interfaces to produce visualizations by assembling *pipelines* out of modules that are connected in a network. However, these systems have important limitations which hamper their ability to support the data exploration process. First, there is *no separation between the definition of a dataflow and its instances*. In order to execute a given dataflow with different parameters (*e.g.*, different input files), users need to manually set these parameters through a GUI—clearly this process does not scale to more than a few visualizations. And second, modifications to parameters or to the definition of a dataflow are destructive—*no change history is maintained*. This places the burden on the scientist to first construct the visualization and then to remember the values and the exact dataflow configuration that led to a particular image.

[1] http://www.paraview.org
[2] http://software.sci.utah.edu/scirun.html

At the University of Utah, we have started to build VisTrails, a visualization management system. VisTrails provides a scientific workflow infrastructure which can be combined with existing visualization systems and libraries. A key feature that sets VisTrails apart from previous visualization as well as scientific workflow systems is the support for data exploration. By maintaining *detailed provenance of the exploration process*—both within and across different versions of a dataflow—it allows scientists to easily navigate through the space of dataflows created for a given exploration task. In particular, this gives them the ability to return to previous versions of a dataflow and compare their results. In addition, in VisTrails there is a clear separation between a dataflow definition and its instances. A dataflow definition can be used as a template, and instantiated with different sets of parameters to generate several visualizations in a scalable fashion—allowing scientists to easily explore the parameter space for a dataflow. Finally, by representing the provenance information in a structured way, the system allows the visualization provenance to be queried and mined.

Although the issue of provenance in the context of scientific workflows has received substantial attention recently, most works focus on data provenance, *i.e.*, maintaining information of how a given data product was generated [5]. This information has many uses, from purely informational to enabling the regeneration of the data product, possibly with different parameters. However, while solving a particular problem, scientists often create several variations of a workflow in a trial-and-error process. These workflows may differ both in the parameter values used and in their specifications. If only the provenance of individual data products is maintained, useful information about the relationship among the workflows is lost.

To the best of our knowledge, VisTrails is the first system to provide support for tracking workflow evolution. In this paper, we describe the provenance mechanism used in VisTrails, which uniformly captures changes to parameter values as well as to workflow definitions.

Outline. The rest of this paper is outlined as follows. In Section 2, we describe our motivating example. Section 3 gives a brief overview of the VisTrails architecture. Our new

approach for tracking dataflow evolution is presented in Section 4. Although our focus is on dataflows for visualization, our ideas are applicable to general scientific workflows. We discuss this issue in Section 5.

2 Motivating Example: EOFS

Paradigms for modeling and visualization of complex ecosystems are changing quickly, creating enormous opportunities for scientists and society. For instance, powerful and integrative modeling and visualization systems are at the core of Environmental Observation and Forecasting Systems (EOFS), which seek to generate and deliver quantifiably reliable information about the environment at the right time and in the right form to the right users. As they mature, EOFS are revolutionizing the way scientists share information about the environment and represent an unprecedented opportunity to break traditional information barriers between scientists and society at large [1]. However, the shift in modeling paradigms is placing EOFS modelers in an extremely challenging position, and at the risk of losing control of the quality of operational simulations. The problem results from the breaking of traditional modeling cycles: tight production schedules, dictated by real-time forecasts and multi-decade simulation databases, lead even today to tens of complex runs being produced on a daily basis, resulting in thousands to tens of thousands of associated visualization products.

As an example, Professor Antonio Baptista, the lead investigator of the CORIE[3] project prepares figures for presentations showing results of simulations that he has designed. The component elements of his figures (whether static or animated) are generated over a few hours by a sequence of scripts, activated by a different staff member in his group. To create the composite figure, Baptista has to request, by e-mail, information on which runs have been concluded. He then draws the composite figure for a particular run in PowerPoint using cut-and-paste. This process is repeated for similar and complementary runs. Because element components are static and visually optimized for each run, cross-run synthesis often have scale mismatches that make interpretations difficult.

The process followed by Baptista is both time consuming and error prone. Each of these visualizations is produced by custom-built scripts (or programs) manually constructed and maintained by several members of Baptista's staff. For instance, a CORIE visualization is often produced by running a sequence of VTK[4] and custom visualization scripts over data produced by simulations. Exploring different configurations, for example, to compare the results of different versions of a simulation code, different rendering algorithms, or alternative colormaps, requires the scripts to be modified, and/or the creation of new scripts. Since there is no infrastructure to manage these scripts (and associated data), often, finding and running them are tasks that can only be performed by their creators. This is one of main reasons Baptista is not able to easily produce the visualizations he needs in the course of his explorations. Even for their creators, it is hard to keep track of the correct versioning of scripts and data. Since these visualization products are generated in an ad-hoc manner, data provenance is not captured in a persistent way. Usually, the figure caption and legends are all the metadata available for this composite visualization in the PowerPoint slide—making it hard, and sometimes impossible, to reproduce the visualization.

[3] http://www.ccalmr.ogi.edu/CORIE

[4] http://www.vtk.org

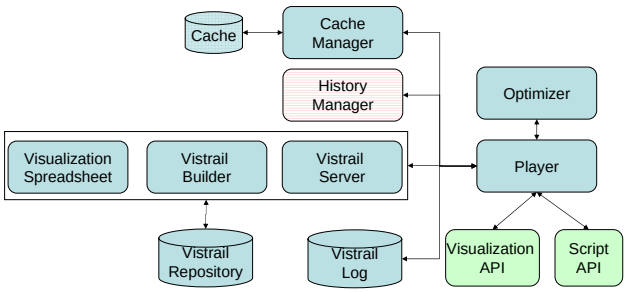
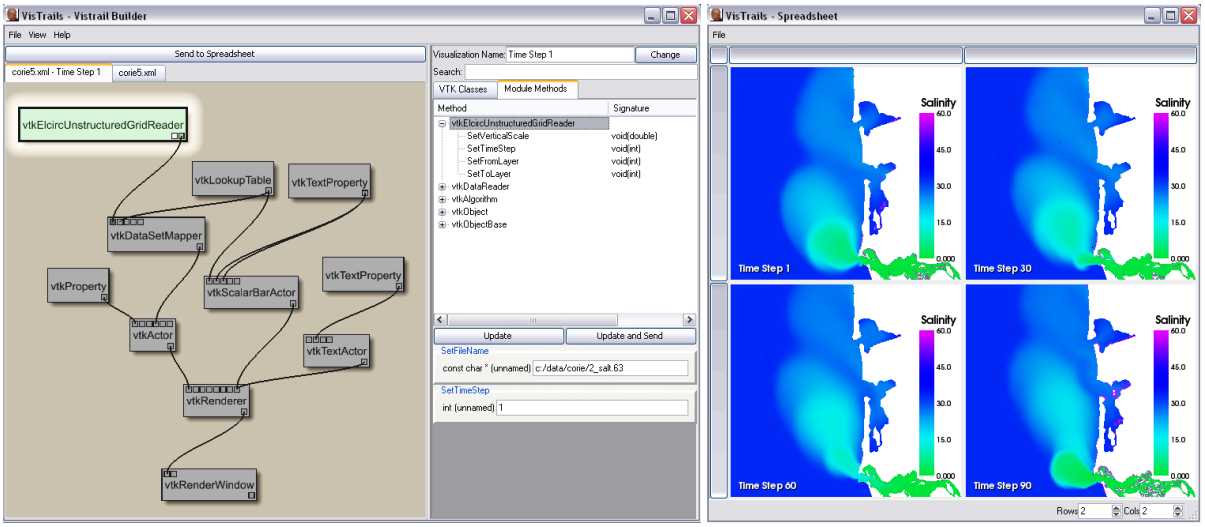


Figure 1. VisTrails Architecture.

3 The VisTrails System

With VisTrails, we aim to give scientists a dramatically improved and simplified process to analyze and visualize large ensembles of simulations and observed phenomena. VisTrails *manages* both the data and metadata associated with visualization products. The high-level architecture of the system is shown in Figure 1. Users create and edit dataflows using the *Vistrail Builder* user interface. The dataflow specifications are saved in the *Vistrail Repository*. Users may also interact with saved dataflows by invoking them through the *Vistrail Server* (*e.g.*, through a Web-based interface) or by importing them into the *Visualization Spreadsheet*. Each cell in the spreadsheet represents a view that corresponds to a dataflow instance; users can modify the parameters of a dataflow as well as synchronize parameters across different cells. Dataflow execution is controlled by the *Vistrail Cache Manager*, which keeps track of operations that are invoked and their respective parameters. Only new combinations of operations and parameters are requested from the *Vistrail Player*, which executes the operations by invoking the appropriate functions from the Visualization and Script APIs. The Player also interacts with the *Optimizer* module, which analyzes and optimizes the dataflow specifications. A log of the vistrail execution is kept in the *Vistrail Log*. The different components of the system are described below. Since our emphasis in this paper is on dataflow evolution and history management, we only sketch the main features of the system here, for further details see [2].



(a) (b)

Figure 2. The Vistrail Builder (a) and Vistrail Spreadsheet (b) showing the dataflow and visualization products of the CORIE data.

Dataflow Specifications A key feature that distinguishes VisTrails from previous visualization systems is that it separates the notion of a dataflow specification from its instances. A dataflow instance consists of a sequence of operations used to generate a visualization. This information serves both as a log of the steps followed to generate a visualization—a record of the visualization provenance—and as a recipe to automatically regenerate the visualization at a later time. The steps can be replayed exactly as they were first executed, and they can also be used as templates—they can be parameterized. For example, the visualization spreadsheet in Figure 2 illustrates a multi-view visualization of a single dataflow specification varying the time step parameter. Operations in a vistrail dataflow include visualization operations (*e.g.*, VTK calls); application-specific steps (*e.g.*, invoking a simulation script); and general file manipulation functions (*e.g.*, transferring files between servers). To handle the variability in the structure of different kinds of operations, and to easily support the addition of new operations, we defined a flexible XML schema to represent the general dataflows. The schema captures all information required to re-execute a given dataflow. The schema stores information about individual modules in the dataflow (*e.g.*, the function executed by the module, input and output parameters) and their connections—how outputs of a given module are connected to the input ports of another module. The XML representation for vistrail dataflows allows the reuse of standard XML tools and technologies. An important benefit of using an open, self-describing, specification is the ability to share (and publish) dataflows.

Another benefit of using XML is that the dataflow specification can be queried. Users can query a set of saved dataflows to locate a suitable one for the current task; query saved dataflow instances to locate anomalies documented in annotations of previously generated visualizations; locate data products and visualizations based on the operations applied in a dataflow; cluster dataflows based on different criteria; etc. For example, an XQuery query could be posed by Professor Baptista to find a dataflow that provides a 3D visualization of the salinity at the Columbia River estuary (as in Figure 2) from a database of published dataflows. Once the dataflow is found, he could then apply the same dataflow to more current simulation results, or modify the dataflow to test an alternative hypothesis. With VisTrails, he has the ability to steer his own simulations.

Caching, Analysis and Optimization Having a high-level specification allows the system to *analyze and optimize dataflows*. Executing a dataflow can take a long time, especially if large datasets and complex visualization operations are used. It is thus important to be able to analyze the specification and identify optimization opportunities. Possible optimizations include, for example factoring out common subexpressions that produce the same value; removing no-ops; identifying steps that can be executed in parallel; and identifying intermediate results that should be cached to minimize execution time. Although most of these optimization techniques are widely used in other areas, they have yet to be applied in dataflow-based visualization systems.

In our current VisTrails prototype, one optimization we have implemented is memoization. VisTrails leverages the dataflow specification to identify and avoid redundant operations. For complete details, see [2]. Caching is especially useful while exploring multiple visualizations. When variations of the same dataflow need to be executed, substantial speedups can be obtained by caching the results of overlapping subsequences of the dataflows.

Playing a Vistrail The Vistrail Player (VP) receives as input an XML file for a dataflow instance and executes it using the underlying Visualization or Script APIs. Information

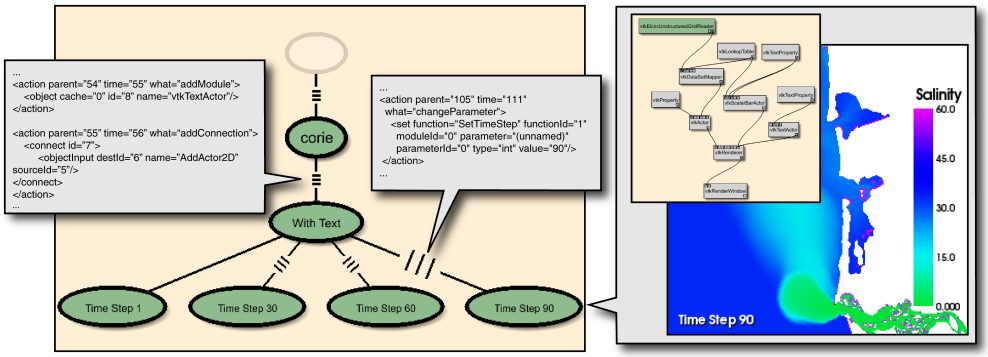


Figure 3. A snapshot of the VisTrails history management interface. Each node in the vistrail history tree represents a dataflow version. An edge between a parent and child nodes represents to a set of (change) actions applied to the parent to obtain the dataflow for the child node. The image and dataflow instance corresponding to the node labeled "Time Step 90" are shown on the right.

pertinent to the execution of a particular dataflow instance is kept in the Vistrail Log (see Figure 1). There are many benefits from keeping this information, including: the *ability to debug* the application—*e.g.*, it is possible to check the results of a dataflow using simulation data against sensor data; *reduced cost of failures*—if a visualization process fails, it can be restarted from the failure point. The latter is especially useful for long running processes, as it may be very expensive and time-consuming to execute the whole process from scratch. Logging *all* the information associated with all dataflows may not be feasible. VisTrails provides an interface that lets users select which and how much information should be saved.

Creating and Interacting with Vistrails The Vistrail Builder (VB) provides a graphical user interface for creating and editing dataflows (Figure 2(a)). It writes (and also reads) dataflows in the same XML format as the other components of the system. It shares the familiar nodes-and-connections paradigm with dataflow systems. To allow users to compare the results of multiple dataflows, we built a Visualization Spreadsheet (VS). The VS provides the user a set of separate visualization windows arranged in a tabular view. This layout makes efficient use of screen space, and the row/column groupings can conceptually help the user explore the visualization parameter space. The cells may execute different dataflows and they may also use different parameters for the same dataflow specification (see Figure 2(b)). To ensure efficient execution, all cells share the same cache. Users can also synchronize different cells using the VS interface.

4 Capturing Dataflow Evolution

Vistrail: An Evolving Dataflow To provide full provenance of the visualization exploration process, we introduce the notion of a visualization trail—a vistrail. *A vistrail captures the evolution of a dataflow—all the trial-and-error steps followed to construct a set of visualizations*. A vistrail consists of a collections of dataflows—several versions of a dataflow and its instances. A vistrail allows scientists to explore visualizations by returning to and modifying previous versions of a dataflow.

An actual vistrail is depicted in Figure 3. Instead of storing a set of related dataflows, we store the operations or actions that are applied to the dataflows. A vistrail is essentially a tree in which each node corresponds to a *version* of a dataflow, and each edge between nodes P and C, where P is the parent of C, corresponds to one or more actions applied to P to obtain C. This is similar to the versioning mechanism used in DARCS[5]. More formally, let DF be the domain of all possible dataflow instances, where $\emptyset \in DF$ is a special empty dataflow. Also, let $x : DF \rightarrow DF$ be a function that transforms a dataflow instance into another, and $\mathscr{D}$ be the set of all such functions. A vistrail node corresponds to a dataflow constructed by a sequence of actions:

$$vt = x_n \circ x_{n-1} \circ \ldots \circ x_1 \circ \emptyset$$

where each $x_i \in \mathscr{D}$.

An excerpt of the XML schema for a vistrail is shown in Figure 4.[6] A `visTrail` has a unique id, a name, an optional annotation, and a set of actions. Each `action` is uniquely identified by a timestamp (`@time`), which corresponds to the time the action was executed. Since actions form a tree, an action also stores the timestamp of its parent (`@parent`). The different actions we have implemented in our current prototype are described below. To simplify the retrieval of particularly interesting versions, a vistrail node can optionally have a name (the optional attribute `tag` in the schema).

Dataflow Change Operators. In the current VisTrails prototype, we implemented a set of operators that correspond to common actions applied in the exploratory process, including: adding or replacing a module, deleting a module, adding a connection between modules, and setting parameter values. We also have an import operator that adds a dataflow to an empty vistrail—this is useful for starting a new exploration process.

[5] http://abridgegame.org/darcs

[6] Due to space constraints, we only show subset of the schema and use a notation that is less verbose than XML Schema.

```
type VisTrail =
 visTrail [ @id, @name, Action*, annotation? ]
type Action =
 action [ @parent, @time, tag?, annotation?,
          (AddModule|DeleteModule|ReplaceModule|
   AddConnection|DeleteConnection|SetParameter)]
```

Figure 4. Excerpt of the vistrail schema.

This action-oriented provenance mechanism captures important information about the exploration process, and it does so through a very simple tracking (and recording) of the steps followed by a user. Although quite simple and intuitive, this mechanism has important benefits. Notably, it uniformly captures both changes to dataflow instances (*i.e.*, parameter value changes) and to dataflow specifications (*i.e.*, changes to modules and connections).

In addition, the action-oriented model leads to a very natural means to *script* dataflows. For example, to execute a given dataflow f over a set of n different parameter values, one just needs to apply a sequence of set parameter actions to f:

$$(setParameter(id_n, value_n) \circ \ldots (setParameter(id1, value1) \circ f) \ldots)$$

Or to compare the results of different rendering algorithms, say R_1 and R_2, a *bulk update* can be applied that replaces all occurrences of R_1 with R_2 modules.

User Interface. At any point in time, the scientist can choose to view the entire history of changes, or only the dataflows important enough to be given a name (*i.e.*, the tagged nodes). The history tree in Figure 3 shows a set of changes to a dataflow that was used to generate the CORIE visualization products shown in Figure 2. In this case, a common dataflow was created from scratch to visualize the salinity in a small section of the estuary for all time steps. This common dataflow (tagged "With Text"), was then used to create four different dataflows that represent different time steps of the data and are shown separately in the spreadsheet. Note that in this figure, only tagged nodes are displayed. Edges that hide untagged nodes are marked with three short perpendicular lines. We are currently investigating alternative structures and techniques to display the history tree. One issue with the current interface is that it does not convey the chronological order in which the versions were created—the structure only shows the dependencies among the versions. Thus, it can be hard to identify the most current branch one has worked on. To aid the user in this task, we plan to use visual cues, *e.g.*, to use different saturation levels to indicate the age of the various dataflows. In addition, a vistrail is often used in a collaborative setting, where several people can modify a given vistrail. For shared vistrails, it is also important to distinguish nodes created by different people, and we plan to use color to identify different users.

Related Work. Kreuseler *et al.* [3] proposed a history mechanism for exploratory data mining. They use a tree-structure, similar to a vistrail, to represent the change history, and describe how undo and redo operations can be calculated in this tree structure. They describe a theoretical framework that attempts to capture the complete state of a software system. In contrast, in our work we use a simpler model and only track the evolution of dataflows. This allows for the much simpler action-based provenance mechanism described above.

5 Conclusion and Future Work

In this paper, we proposed a novel provenance mechanism that uniformly captures changes to parameters as well as to dataflow definitions. This mechanism has been implemented in VisTrails, a visualization management system whose goal is to provide adequate infrastructure to support data exploration through visualization. Although our focus in the VisTrails project has been on dataflows for visualization, the techniques we have developed are general and have been adopted in other domains. For example: the VisTrails cache management infrastructure was implemented in Kepler, a scientific workflow system [4]; and our provenance mechanism is being used in the Emulab testbed, to track revisions of experiments[7].

An alpha release of VisTrails (available upon request) is currently being tested by a select group of domain scientists.

Acknowledgments. António Baptista has provided us valuable input on our system, as well as several CORIE datasets for our experiments. This work is partially supported by the National Science Foundation (under grants IIS-0513692, CCF-0401498, EIA-0323604, CNS-0514485, IIS-0534628, CNS-0528201, OISE-0405402), the Department of Energy, an IBM Faculty Award, and a University of Utah Seed Grant. E. Santos is partially supported by a CAPES/Fulbright fellowship.

References

[1] A. Baptista, T. Leen, Y. Zhang, A. Chawla, D. Maier, W.-C. Feng, W.-C. Feng, J. Walpole, C. Silva, and J. Freire. Environmental observation and forecasting systems: Vision, challenges and successes of a prototype. In *Conference on Systems Science and Information Technology for Environmental Applications (ISEIS)*, 2003.

[2] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *IEEE Visualization 2005*, pages 135–142, 2005.

[3] M. Kreuseler, T. Nocke, and H. Schumann. A history mechanism for visual data mining. In *IEEE Information Visualization Symposium*, pages 49–56, 2004.

[4] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 2005.

[5] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[7] http://www.emulab.net