SIMPLIFYING THE CREATION AND DEPLOYMENT OF COLLABORATIVE DATA ANALYSIS AND VISUALIZATION TOOLS

by

Emanuele Marques dos Santos

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

August 2010

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of _____

has been approved by the following supervisory committee members:

_____ , Chair _____
Date Approved

_____ , Member _____
Date Approved

_____ , Member _____
Date Approved

_____ , Member _____
Date Approved

_____ , Member _____
Date Approved

and by _____ , Chair of

the Department of _____

and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

Managing and understanding the large amounts of scientific data is undoubtedly one of the most difficult research challenges scientists are facing today. Data exploration through visualization is an effective means to understand and obtain insights from large collections of data. Although several visualization tools are available, including tools with sophisticated visual interfaces, they are out of reach for users who have little or no knowledge of visualization techniques and/or who do not have programming expertise. In addition, as interdisciplinary groups work together, the ability to generate a diversified collection of analyses for a broad audience in an ad-hoc manner is essential for supporting effective scientific data exploration. Science portals and visualization web sites have been used to simplify this task by aggregating data from different sources and by providing a set of predesigned analyses and visualizations. However, such portals are often built manually, and are not flexible enough to support the vast heterogeneity of data sources, analysis techniques, data products, and user communities that need to access this data.

The goal of this dissertation is to provide a complete framework for streamlining the creation of customized visualization applications and to facilitate collaboration and sharing among scientists and visualization experts. The framework is composed of three main components: VisMashup, for creating customized visualization applications based on dataflow specifications; Provenance-Rich Publications, a component that allows users to create documents (web pages, presentations or pdf files) whose digital artifacts (*e.g.,*figures) include detailed provenance information (workflow and associated parameters) used to produce the artifact; and CrowdLabs, a system that adopts the model used by social Web sites and that combines a set of usable tools and a scalable infrastructure for providing a rich collaborative environment for scientists and that also takes into account the requirements of computational scientists, such as accessing high-performance computers and manipulating large amounts of data.

To demonstrate how users can benefit from this framework, a series of use case scenarios is also included.

In Memory of My Dear Grandmother, Paulina Maria dos Santos

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

There are many people I would like to thank for making this dissertation possible. I would like to thank Prof. Cláudio Silva and Prof. Juliana Freire for giving me the opportunity to pursue this research and to work with them at the University of Utah. I am really grateful for their guidance and support throughout these five years that we worked together. My appreciation also extends to Prof. Creto Vidal, my advisor and friend in my earlier studies who persuaded me to pursue a doctorate degree. I would like to thank my committee members for their valuable feedback for this dissertation: Cláudio Silva, Juliana Freire, Chris Johnson, Valerio Pascucci and James Ahrens. I would like to thank Prof. Joel Tohline for believing in this work since the beginning.

It was a pleasure working with my co-authors: Lauro Lins, James Ahrens, David Koop, Joel Tohline, Huy Vo, Phillip Mates, Erik Anderson, Brad Grimm, Wendel Silva, Patricia Hendricks, Ayla Khan, Julien Tierny, Steve Callahan, Carlos Scheidegger and all the VisTrails team.

Special thanks to Huy, Lauro, Tiago, Gustavo, Phillip and Joel for the insightful discussions usually accompanied by a cup of coffee at the SCI cafeteria. Thanks to the wonderful staff of the SCI Institute and School of Computing at the University of Utah, specially to Karen Feinauer and Magali Coburn for their assistance with all the necessary paperwork to finish this dissertation. Thanks to Ed Cask and Chems Touati for the incredible multimedia and videoconference support. Thanks to Liz Jordan and Deb Zemeck for pampering us students, even when we did not appreciate them enough.

I would like to thank CAPES and Fulbright for giving me a scholarship. Thanks to Glayna Braga and her colleagues at the Fulbright Commission in Brazil, and to Sarah McCormick at the IIE for their kind support. My work has also been funded by the National Science Foundation, the Department of Energy SciDAC (VACET and SDM centers) and Los Alamos National Laboratory.

I would like to thank my dearest friends who made my life away from home more bearable: Bea, Luciano, Anúsia, Nanda, Todd, Lauro, Sofia, Tiago, Karina, Cris, Gustavo, Karane, Douglas and Ronaldo. Thanks to Kathy and Linda for welcoming me so warmly.

Thanks to Frei Wellington for being the best of the listeners, to Kel, my number-one fan and to all my friends in Brazil, who always presented me with a few words of encouragement.

Finally, I would like to thank all my family, especially my grandparents, Elça, Fragoso and Paulina (who died while I was in Utah), my parents Jaubeth and Raimundo and my sister, Isabela. Neta, I am sorry if I made you cry with *saudades* so many times. If it were not for their endless patience and support, I would not have succeeded.

CHAPTER 1

INTRODUCTION

Data exploration through visualization is an effective means to understand and obtain insights from large collections of data. Not surprisingly, visualization has become a mature area with an established research agenda [51], and several systems have been developed that support the creation of complex visualizations [38, 37, 15, 49, 80, 52, 35, 77]. But even using systems that have sophisticated visual programming interfaces, such as AVS, DX, SCIRun and VisTrails, the path from the raw data to insightful visualizations is laborious and error-prone. And this has hampered a wider adoption of visualization techniques.

Visual programming interfaces expose computational components as *modules* and allow the creation of complex visualization pipelines which combine these modules in a dataflow, where *connections* between modules express the flow of data through the pipeline [41]. They also provide built-in constraint checking mechanisms (*e.g.,* that disallow a connection between incompatible module ports) that simplify the creation of pipelines. Recently, new techniques have been proposed to simplify the creation and refinement of these pipelines, including recommendation systems [70, 40] and a mechanism to refine pipelines by analogy [62]. Notwithstanding, without detailed knowledge of the underlying computational components, it is difficult to understand what series of modules and connections ought to be added to obtain a desired result. Consequently, the generation of visualizations is a task that is best carried out by experts who are familiar with visualization techniques and tools.

Recognizing this problem, there have been a number of efforts to create simpler visualization applications (*e.g.,* [77, 24, 71]) that are customized to a particular task (*e.g.,* [36, 44, 11]) and that use concepts that are more meaningful to domain experts than visualization primitives. Some systems [77, 24, 71, 44] provide customized interfaces that allow users to control the parameters of a pipeline. These simplified interfaces are easier to use than the general systems and can be designed specifically for domain experts that do not have expertise in visualization. Nonetheless, they are expensive to create and maintain. The life-cycle for developing custom applications is often long, requiring several iterations until the application does what the end-user needs. Whereas such solutions can be (and have been)

successfully applied in big scientific projects, they are out of reach for smaller-scale efforts. The costs involved in creating these applications make them unsuitable for one-of-a-kind, exploratory visualizations. Figure 1.1 graphically illustrates how visualization tools behave with relation to flexibility and ease-of-use. Usually, the more flexible a visualization tool is, the more difficult it is to use. Expert users favor more flexible tools, while end users prefer easy-to-use tools.

This problem is compounded as interdisciplinary groups collaborate and need to perform a wide range of analyses targeted to multiple audiences. Recently, social Web sites have been created that enable users to collaboratively visualize data [78, 67, 68]. They allow users to create and discuss visualizations of a wide range of data sets. However, they fail to cater to important requirements of scientific exploration. In particular, they were designed for small data sets and only provide a limited set of visualizations.

Science portals [14, 53, 34], on the other hand, have focused on simplifying data exploration by aggregating data from different sources and by providing a set of canned analyses and visualizations. However, they have important limitations. They are insufficient for handling large volumes of heterogeneous data and the diversity of stakeholders and their needs; it is simply not possible for IT personnel to anticipate all necessary analyses and different ways to correlate and integrate data. Furthermore, while some analyses that are used regularly can be canned, others are ground-breaking and need to be created, altered on-the-fly and improved as part of a collaborative effort. Last, but not least, many small research groups do not have the necessary resources to create such portals.

1.1 Thesis Statement

Effective exploration of scientific data requires a combination of novel software and frameworks that provide users with different goals and levels of expertise the ability to create, modify and share their own analysis and visualization workflows and tools.

1.2 Dissertation Objectives

This dissertation describes a complete framework that simplifies the creation and deployment of collaborative data analysis and visualization tools. The system is composed of three main components:

- **VisMashup**, which streamlines the creation of customized visualization applications that can be deployed on desktop computers, Web or portable devices [57, 58, 75]. While this infrastructure simplifies the job of an application designer, it gives an

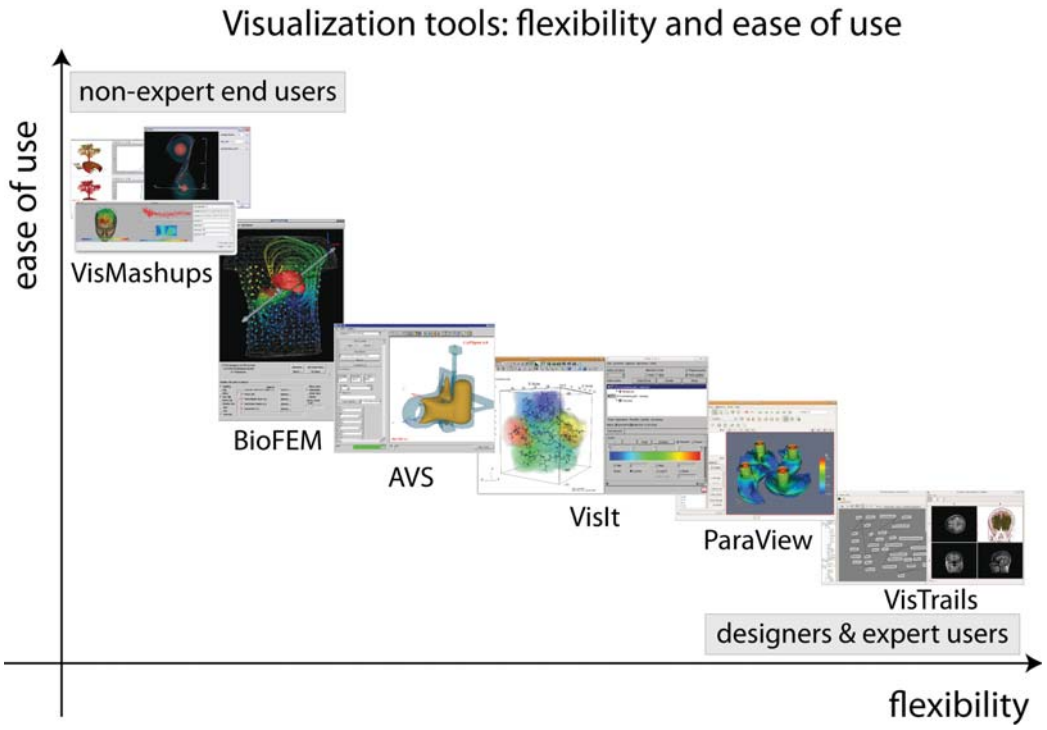

Figure 1.1. A graphical display of how visualization tools behave with relation to flexibility and ease-of-use. Expert users prefer visualization tools that are more flexible, such as VisTrails, ParaView and VisIt. End users, on the other hand, lean towards more easy-to-use tools, which are frequently customized to their needs..

end-user the flexibility to create visualizations through a simple interface that hides unnecessary details of the underlying pipelines and how they are integrated.

- **Provenance-Rich Publications**, a component that allows users to create documents (web pages, presentations or pdf files) whose digital artifacts (*e.g.,* figures) include detailed provenance information (workflow and associated parameters) used to produce the artifact [56].

- **CrowdLabs**, a component that adopts the model used by social Web sites and Web-based communities (groups, friends, tags, comments, rates, *etc.*) to enable social analysis of scientific data [61, 60]. The goal is to facilitate collaboration and sharing among scientists and visualization experts. Shared repositories of analysis and visualization workflows will expose users to a large number of tasks that provide examples of (sophisticated) uses of tools. This will help scientists to effectively perform their own analyses, and to create and publish new data products with minimal intervention of expert programmers and IT personnel.

The remainder of this dissertation is organized as follows. Chapter 2 reviews related work and provides background information on the system used by the framework. Chapter 3 describes the VisMashup system and Chapter 4 describes the CrowdLabs system. Chapter 5 presents our approach for enabling reproducible research. Finally, the application scenarios are given in Chapter 6 and conclusions and future work are presented in Chapter 7.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Visualization Systems

A number of visualization systems have been proposed that provide sophisticated visual programming interfaces. These interfaces aim to simplify the creation of visualizations by allowing users to create pipelines which combine computational components (modules) in a dataflow [77, 35, 52, 80]. However, without detailed knowledge of the underlying computational components, it is difficult to understand what series of modules and connections ought to be added to obtain a desired result. Other systems, such as ParaView [37], go a step further and not only hide the details of the pipelines, but also provide abstractions that simplify the creation of visualizations. However, these systems are still very general and require detailed knowledge of visualization techniques. Similar to these tools, VisMashup (described in Chapter 3) relies on dataflows which describe a series of visualizations, and like ParaView, the derived applications hide the underlying pipelines. However, with VisMashup, customized (specific) applications can be derived that are targeted to end-users who do not necessarily have knowledge of visualization. VisMashup is built on top of the VisTrails system [80, 4, 27], which is described below.

2.1.1 The VisTrails System

VisTrails [80] is an open-source provenance management and scientific workflow system that was designed to support the scientific discovery process. VisTrails provides unique support for data analysis and visualization, a comprehensive provenance infrastructure, and a user-centered design. The system combines and substantially extends useful features of visualization and scientific workflow systems. Similar to visualization systems [35, 77, 37, 79], VisTrails makes advanced scientific visualization techniques available to users, allowing them to explore and compare different visual representations of their data; similar to scientific workflow systems, VisTrails enables the composition of workflows that combine specialized libraries, distributed computing infrastructure and Web services. As a result, users can create complex workflows that encompass important steps of scientific discovery, from data

gathering and manipulation, to complex analyses and visualizations, all integrated in one system.

Workflows have been traditionally used to automate repetitive tasks; however, for applications that are exploratory in nature, such as simulations, data analysis and visualization, very little is repeated—change is the norm. As a scientist generates and evaluates hypotheses about data under study, a series of different, albeit related, workflows are created as they are adjusted in an iterative process. VisTrails was designed to manage these rapidly-evolving workflows. Another distinguishing feature of VisTrails is a comprehensive provenance infrastructure that maintains detailed history information about the steps followed and data derived in the course of an exploratory task; VisTrails maintains provenance of data products (*e.g.,* visualizations, plots) of the workflows that derive these products and their executions (see Figure 2.1).

The system also provides extensive annotation capabilities that allow users to enrich the automatically captured provenance. This information is persisted as XML files or in a relational database. Besides enabling reproducible results, VisTrails leverages provenance information through a series of operations and intuitive user interfaces that aid users to collaboratively analyze data. Notably, the system supports reflective reasoning by storing temporary results, by providing users the ability to reason about these results and to follow chains of reasoning backward and forward. Users can navigate workflow versions in an intuitive way, undo changes but not lose any results, visually compare multiple workflows and show their results side-by-side in a visual spreadsheet, and examine the actions that led to a result. In addition, the system has native support for parameter sweeps, whose results can also be displayed on the spreadsheet.

VisTrails has been modified to provide access to workflow provenance in a client-server mode in order to be used by VisMashup and CrowdLabs. The same mechanism is also used to create provenance-rich documents.

2.2 Simplifying Application Development and Use

One approach that has been used to simplify data exploration through visualization is the creation of custom applications. CDAT, for example, is a tool designed for climatologists that supports visual exploration of climate data [11]. Because such applications are costly to develop, this approach is not scalable. Instead of building applications from scratch, some systems (*e.g.,* AVS [77], IRIS Explorer [24], VISSION [71] and SCIRun [44]) allow the creation of customized interfaces for visualization pipelines. In SCIRun, these interfaces are called PowerApps. BioFEM, for example, is a PowerApp built atop of a SCIRun

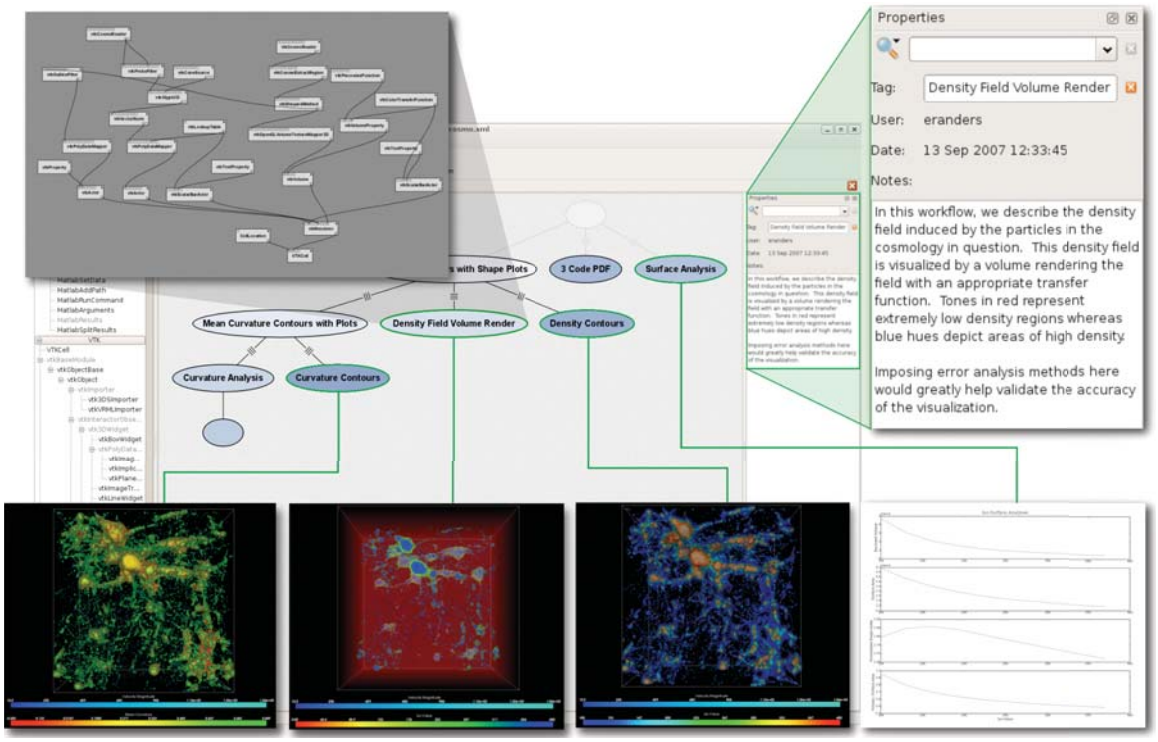

Figure 2.1. Exploratory visualization for studying celestial structures derived from cosmological simulations using VisTrails. Complete provenance of the exploration process is displayed as a vistrail (history tree), where each node represents a workflow that generates a unique visualization. Detailed meta-data are also stored including free-text notes made by the scientist, the date and time the workflow was created or modified, optional descriptive tags and the user that created it.

pipeline that solves a bioelectric forward problem using the finite element method provided by Biological Problem Solving Environment (BioPSE) tools [44]. BioFEM consolidates all necessary user controls in a single UI, hiding other generic controls and the associated dataflow network. Unfortunately, a SCIRun PowerApp needs to be manually crafted for a given pipeline. IRIS Explorer provides a GUI to facilitate the layout and design of widgets representing parameters, but users need to manually create the interfaces. Similar to VISSION, VisMashup automatically derives the application interface from a dataflow network. But VisMashup goes a step further; it provides a comprehensive infrastructure that allows the application developer to explore collections of pipelines and combine them to create the applications. Another important distinction between VisMashup and previous systems is that it captures detailed provenance of both application development and use.

Several approaches have been proposed whose goal is to simplify application development, from high-level programming languages such as Ruby[1] and Python[2], to visual programming tools such as Visual Basic. These, however, still require programming expertise. More recently, there have been a number of proposals for tools targeted to nonprogrammers. For example, Yang et al. [84] proposed a WYSIWYG tool that aids in the development of data-driven Web applications, and Yahoo! Pipes [83] provides a visual interface through which users can assemble pipelines for constructing Web mashups. Although VisMashup shares some of the same goals, our approach targets the construction of applications which use dataflows as basic building blocks.

Reduced-functionality interfaces, such as training wheels [10], have been used to help first-time and novice users of complex systems. In these interfaces, advanced features are disabled on the GUI to prevent user errors and confusion. Shneiderman [65] proposed the design of layered interfaces, which provide a set of increasingly complex interfaces, allowing the user to select the interface that best suits her needs. VisMashup naturally supports both techniques—the designer can easily create several variations of an application through a series of pipeline views. In addition, because provenance is maintained, detailed information about how the different application versions were created and their differences is readily available.

2.3 Reproducible Research

In 1994, in a National Research Council meeting on High Performance Computing in Seismology, Jon Claerbout [16] stated that, "In engineering, a published paper is an advertisement of scholarship but the electronic document can be the scholarship itself. Forty years ago data were *pencil marks on paper* and theory was some Greek symbols. Then paper documents were adequate. No more. Now we need electronic documents." Claerbout was one of the first to realize that the printed medium is not adequate for reporting complex computation results. When publishing scientific results, it is important to describe the lineage of a result. Providing data sets and computer code allows scientists to verify and reproduce published results and to conduct alternate analyses. In the past few years, interest in this subject has increased in different communities [50, 22] which led to different approaches for publishing scientific results.

[1] http://www.ruby-lang.org

[2] http://www.python.org

2.3.1 Reproducible Documents

ReDoc [63], one of the earliest approaches created by Claerbout and his students, consisted of a system for document preparation using concepts similar to make files. By using its special viewer, the readers had access to the complete computational environment (code and data) that generated the results in the article. The open source Madagascar project[3] currently extends this environment with modern tools such as Python-based SCons integrated with LaTex [23]. Other examples in the statistical community are WaveLab [8] and Sweave [42]. WaveLab focuses on expressing all the details underlying the data sets, simulations, figures and tables uniformly in Matlab's standard language and computing environment. Wavelab was the first effort to provide a toolbox specifically designed to reproduce results in a series of computational science papers. Sweave [42] is an extension of Donald Knuth's literate programming concept [39] applied to the creation of reproducible statistical documents. It combines the LaTex document formatting language with the R programming language to divide documents into either text or code chunks, processing each in a different way. Gentleman and Temple Lang [29] proposed a paradigm where documents are mixtures of code and text. Such documents may be self-contained or they may be a component of a compendium, which provides the infrastructure for accessing the data and supporting software. These documents, or compendiums, can be processed in a number of different ways. For example, one transformation will be to replace the code with its output, providing the static document. The usefulness of the compendium concept is demonstrated in [28], where the paradigm was applied to a seminal paper in bioinformatics by Golub *et al.* entitled "The Molecular Classification of Cancer" [30].

2.3.2 Sharing Scientific Data on the Web: Challenges and Opportunities

The success of Web 2.0 technologies, such as wikis, blogs, and social-networking sites, together with the proliferation of Internet-enabled wireless portable devices has opened up new opportunities to improve collaboration and information sharing among scientists. Recently, the idea of Science 2.0 has started to gain attention. Science 2.0 entails the use of the open-access Web 2.0 technologies for carrying out scientific activities [81]. It can also be viewed as a new kind of science, which introduces new methods for carrying out scientific research [64].

---

[3] `http://reproducibility.org`

By democratizing science, Science 2.0 has the potential to benefit both the scientific community and the general public. The Web is open and for this reason, anyone can participate, publish and consume information, regardless of whether one is a member of a large established research group with plentiful resources, an independent researcher or a high-school student.

Because the Web provides virtually unlimited space, scientific results can be described at much greater detail. Whereas a scientific peer-reviewed publication represents a snapshot of a given problem and solution, on the Web, scientists can essentially publish their notebooks and the many different alternatives they tried, as well as the data sets and analysis scripts they used. With all this information, readers can better understand not only the results but also the exploratory process that led to those results. In some cases, they may also be able to reproduce and validate the results. Besides better information dissemination, Science 2.0 also leads to improved collaboration. As a scientist posts her results (and questions) online, she can receive immediate feedback as well as discuss (or blog) with a number of researchers throughout the world [31, 78].

2.3.2.1 Science 2.0 Today: Opportunities

2.3.2.1.1 Collaborative content creation and curation. Wikis have become a popular means to share data and scientific findings [82]. Scientific wikis have been used in different ways. OpenWetWare[4] is a wiki for sharing electronic lab notebooks in biological sciences and engineering. Started in 2005, today it serves more than 7,700 registered users. WikiPathways[5] has been used by the biology community for sharing and maintaining a pathway database [53]. It provides the infrastructure for a mass collaboration approach to curate the pathways. To facilitate the participation in pathway curation, WikiPathways extended the popular MediaWiki software to include a custom graphical pathway editing tool and integrated databases comprising major gene, protein and small-molecule systems.

Wikigenes[6] is a portal that provides the access to gene, protein and chemical compound databases [34]. Like any wiki, WikiGenes consists of thousands of articles collaboratively edited by users. Most wiki engines keep track of every modification made to a page, but these modifications are rarely exposed to users because after a large number of changes,

[4] http://openwetware.org

[5] http://wikipathways.org

[6] http://wikigenes.org

the information can be very hard to understand. A key difference between WikiGenes and other wikis is that it links every piece of text and every word directly to its author. It also allows registered authors to rate other authors' contributions, helping to build reputation within the community.

2.3.2.1.2 Accessing high-performance computing (HPC) resources. Being able to run compute-intensive pipelines or simulation jobs and interact with them by changing parameter values and observing their effects is rapidly becoming essential for most scientists. HUBzero Platform for Scientific Collaboration [48] is a cyberinfrastructure developed at Purdue University that lets researchers access and share scientific simulation and modeling tools. It was created to support nanoHUB.org [66], an online community for the Network for Computational Nanotechnology (NCN), funded by the NSF since 2002 to connect simulation developers with experimentalists and educators, who can then access the resulting tools using an ordinary Web browser and launch simulation runs on the national Grid infrastructure, without having to download or compile any code. Nowadays, HUBzero leverages a cloud of computing resources to support hubs that serve different communities. When a user launches a live session in nanoHUB.org, a Java applet embedded in the Web page connects to a Web server with a middleware responsible for running the simulation on a cluster of hosts and for projecting the results back to the user's browser using VNC. Each simulation tool runs in a restricted lightweight virtual machine. To publish a tool, developers have to connect to a special workspace machine, compile their code there and use NCN's open source toolkit Rappture to create friendly GUIs. In order to encourage the improvement of the tools and the participation of the users, nanoHUB provides forums with a reward point system and a bug-report/wishlist mechanism that is sent directly to the tool developers.

CrowdLabs (described in Chapter 4) is similar to HUBzero, as it also enables the use of HPC resources and by using VisMashup, it also provides an easy way to deploy applications on the Web. The key difference is that our computations are based on pipelines, which make it much easier for developers and other users to contribute them and for keeping track of changes to the computations. Our approach is flexible enough to be extended to use virtual machines to use cloud resources.

2.3.2.1.3 Social data analysis and visualization pipelines. While there have been several efforts whose goal is to share scientific data, relatively little work has gone into sharing analysis and visualization specifications (pipelines). To this end, most closely related to our approach is myExperiment [54]. myExperiment is a collaborative environment for

sharing pipelines and other digital objects. myExperiment supports versioning of pipelines and can execute certain types of pipelines. However, because its focus is on pipelines that integrate bioinformatics-related Web services, myExperiment does not support data- and compute-intensive pipelines. On the other hand, a number of sites have recently come online which aim to support social analysis and visualization of data. Tableau Public [68] provides infrastructure for users to publish interactive visualizations on the Web and it is free to use with public data. Using a desktop client (available for Windows only) users can import and explore the data using different types of 2D visualization tools (*e.g.,* plots, maps and coordinate views) and when they are ready to share the data and the visualization, they are hosted on Tableau's servers. Swivel [67] and Many Eyes [78] are public social data analysis Web sites, where users can upload data, create visualizations of that data and leave comments on either visualizations or datasets. Many Eyes Wikified [45] is a Web site based on Dashiki [47] which is a wiki-based Web site for collaboratively building visualization dashboards. In Many Eyes and Many Eyes Wikified, the visualization is provided in the browser by using Java applets, where all the processing is done. In Tableau Public and Swivel, the client side is done in JavaScript and the processing is done on the servers, requiring lightweight clients to display the visualizations.

Although the social features of CrowdLabs were inspired by these sites, as discussed above, our goal is to support scientific data exploration. Whereas these approaches focus on small, tabular data and provide a small set of predefined visualizations, CrowdLabs was designed to support large scientific data sets as well as complex, user-defined analyses and visualizations. Users can create data products and visualization by defining and uploading new pipelines to the server, or by modifying existing pipelines in the CrowdLabs repository. They can use different libraries to build the pipelines and the stored provenance information can be used for credit and attribution. Furthermore, CrowdLabs provides a general framework to create social sites such as Swivel and Many Eyes. Consider for example, Many Eyes: if a data set is modified, visualizations using the original data are automatically updated. In CrowdLabs, since detailed provenance is maintained for datasets and pipelines, this behavior can be customized. CrowdLabs also allows the customization of access permissions—sites can be created where pipelines can be open and accessible to all, or restricted to a group of users.

2.3.2.2 Challenges

Although the openness of Science 2.0 has clear benefits and has the potential to make science more efficient, it also presents important challenges. For example, if everyone can publish and edit information on a wiki, can we trust that the information is accurate? If unpublished results are posted on a wiki, can we prevent others from stealing that work? Will scientists spend all the extra time required to publish their notebooks and associated data online? And if they do, will we be flooded with information and unable to find what is really important? If a document links to information in a wiki, will that information be there one year from now? We discuss these issues below.

2.3.2.2.1 The importance of provenance. Provenance, from the Latin word *provenire*—"to come from," means the origin, or the source, of something, or the history of the ownership or location of an object. Because Science 2.0 applications facilitate collaboration and allow potentially large groups of people to create and modify artifacts (*e.g.,* documents, databases and workflows), maintaining detailed provenance of these artifacts is crucial. The provenance information can be used to determine authorship, enforce intellectual property rights, validate the integrity of artifacts and assess their quality, and in some cases to reproduce the artifact.

Existing Science 2.0 applications, however, provide little or no support for provenance. Wikis, for example, do track changes to documents, but they fail to capture modifications to artifacts included in the documents. In myExperiment, it is not possible to track sequences of modifications to a workflow by multiple users, unless explicitly stated by the users.

2.3.2.2.2 Integrating Science 2.0 applications. If we examine existing Science 2.0 applications, we can see that each application addresses specific aspects of the scientific process, for example, data curation, result reproducibility and information sharing. To support the scientific process more fully, it is necessary to integrate these tools and the information they hold. One alternative is to use *mashups* that combine components that come from multiple Web sites [13]. For example, this would be useful for a scientist adding an entry on his lab notebook on OpenWetWare about an experiment he did on finding biological pathways. He could add the new pathway he found to WikiPathways and link to the workflow he used to do that on myExperiment. However, achieving this integration can be challenging given the large number of tools used, data formats, and the lack of standard and reliable linking mechanisms [25].

2.3.2.2.3 Usable tools. Part of the success of Web 2.0 technologies can be attributed to the fact that they provide usable tools and interfaces for users to share informa-

tion and collaborate. But to share (reproducible) scientific results, substantial work may be required, *e.g.,* to organize and upload source code and data sets. Scientists will be less inclined to generate reproducible publications or to publish lab notebooks online if great effort and time are required. Thus, there is a need for tools that facilitate this process and that make the process of information sharing more transparent and better integrated with the tools scientists already use for carrying out their experiments. For example, a scientist that uses VisTrails to analyze and visualize data can easily publish her results in a single step, using a single copy-and-paste operation (see Chapter 5 for a description of our approach for publishing scientific results. A video illustrating the technique is also available [55]).

2.3.2.2.4 Finding and making sense of information. Although Science 2.0 sites can help make sense of the scientific data tsunami, they also add to the information overflow problem. An important challenge is thus how to quickly find the information one is looking for. Generic search engines such as Google or Yahoo index attempt to maximize the coverage of their index and they are very good for broad searches. For specific queries, however, they are often ineffective. For example, if a biologist needs to locate online databases related to molecular biology and searches on Google for the keywords "molecular biology database" over 19 million documents are returned. Among these, she will find pages that contain databases, but the results also include a very large number of pages from journals, scientific articles, personal Web pages, etc. Domain-specific (vertical) search engines will be fundamental to help users more effectively search for information in Science 2.0 sites [3, 12]. More specific and structured queries may be required for some of the information available in these sites. Handling such queries and attempting to integrate the information on-the-fly is an open research problem [25].

2.3.2.2.5 Data longevity. An important problem that needs to be addressed in Science 2.0 is data longevity. Since data and applications live in distributed and autonomous sites, there is no guarantee that they will exist forever. Even when an artifact is published together with its provenance, the quality of the artifact is diminished if part of the provenance comes from a site that disappeared, or was produced by workflow that no longer runs (*e.g.,* uses old libraries or defunct Web services). This problem of data preservation has attracted substantial attention recently and it is the main topic of the National Foundation DataNet program [18].

2.3.2.2.6 Authorship attribution. Many scientists are concerned with the openness provided by Science 2.0. Tenure cases and patents are dependent on being the first

to publish a new discovery, and by publishing results on the Web, one risks having ideas *stolen* [81]. Keeping detailed provenance, such as what is done in WikiGenes [34], is essential for keeping track of authorship. myExperiment also allows users to give credit to other users. However, this nontraditional form of credit assignment is far from being accepted by the scientific community.

The adoption of Science 2.0 has the potential to transform the way science is done, both by improving the process of information sharing and by enabling new forms of collaboration. However, its success depends heavily on our ability to solve challenging computer science problems. In order to tackle these problems, our approach leverages the provenance and metadata captured at different levels by providing easy-to-use tools to allow scientists to produce provenance-rich documents without having to spend too much extra effort and time.

CHAPTER 3

VISMASHUP: STREAMLINING THE CREATION OF CUSTOM VISUALIZATION APPLICATIONS

Visualization is essential for understanding the increasing volumes of digital data. However, the process required to create insightful visualizations is involved and time consuming. Although several visualization tools are available, including tools with sophisticated visual interfaces, they are out of reach for users who have little or no knowledge of visualization techniques and/or who do not have programming expertise. In this Chapter, we present VisMashup, a new approach for simplifying the creation, maintenance and use of customized visualization applications (or mashups). Because these applications can be customized for very specific tasks, they can hide much of the complexity in a visualization specification and make it easier for users to explore visualizations by manipulating a small set of parameters. We first introduce basic concepts and describe the main components of the VisMashup model. Then we present the implementation details of the VisMashup components, their interfaces and illustrate how the system can be deployed on portable devices.

3.1 Model

In what follows, we present the basic concepts underlying the model we propose for generating customized visualization apps.

3.1.1 Dataflows

We assume that the dataflow [41] model is used to specify visualization pipelines. In this model, a pipeline is represented as a directed acyclic graph (DAG) where nodes (modules) represent computations and arcs (connections) denote data dependencies. If there is an arc from node m to node n, n requires the output of m to execute. Besides the outputs of earlier computations, nodes can also have parameters (data values) as inputs. The dataflow model is widely-used in visualization systems, including ConMan [32], AVS [77],

SCIRun [52] and VTK-based systems such as Paraview [37], VisIt [79] and VisTrails [80]. Basing VisMashup on dataflows enables our framework to be combined with virtually any system that uses dataflows to specify pipelines. A schematic representation of the model is depicted in Figure 3.1, and we describe its components below.

Definition 1. A *pipeline* (or dataflow) d is defined by a tuple (M, C), where M is a set of *modules* and C a set of *connections*. Each module $m \in M$, in turn, is associated with a tuple (I_m, O_m, P_m), where I_m corresponds to a set of *input ports*, O_m corresponds to a set of *output ports* and P_m is a list of *parameters*. Each parameter $p \in P_m$ is associated with a value v. In a *connection* $(o, i) \in C$ connecting module m to module n, $o \in O_m$ and $i \in I_n$, o is called the *source port* and i is called the *target port*. *Sources* are modules that do not have target ports and *sinks* are modules not containing source ports. Additionally, ports and parameters have a type; if there is a connection $(o, i) \in C$, the types of o and i must be compatible. ■

Definition 2. A *subpipeline* $d_s(M_s, C_s)$ of a pipeline $d(M, C)$ is the pipeline induced by the set of modules $M_s \subset M$. For each connection $c \in C$, if c connects two modules m_1 and m_2, and $m_1, m_2 \in M_s$, then $c \in C_s$. ■

3.1.2 Pipeline Operations

The two basic operations our model applies to pipelines are *run*, which executes a pipeline; and *substitution*, which manipulates pipeline components (*i.e.,*parameter values and modules). These operations are used both during application design and to manipulate the pipeline through the application.

3.1.2.1 Run

A *pipeline run* is the execution of a pipeline in the order determined by the network of modules and connections, in a *demand-driven* fashion. Each sink module ensures that its input ports have up-to-date data from incoming connections, causing the modules of the source ports of those connections to do the same. These data requests are propagated up (recursively) to the sources of the pipeline which will, in turn, execute and produce the output values required by the downstream modules. The downstream modules are executed, in turn, until the sinks located at the end of the pipeline are reached.

3.1.2.2 Substitution

Substitution allows parameter values and modules to be replaced within a pipeline. Let $d = (M, C)$ and $d' = (M', C')$ be two pipelines. The operation $substituteParameter_d(p, v)$

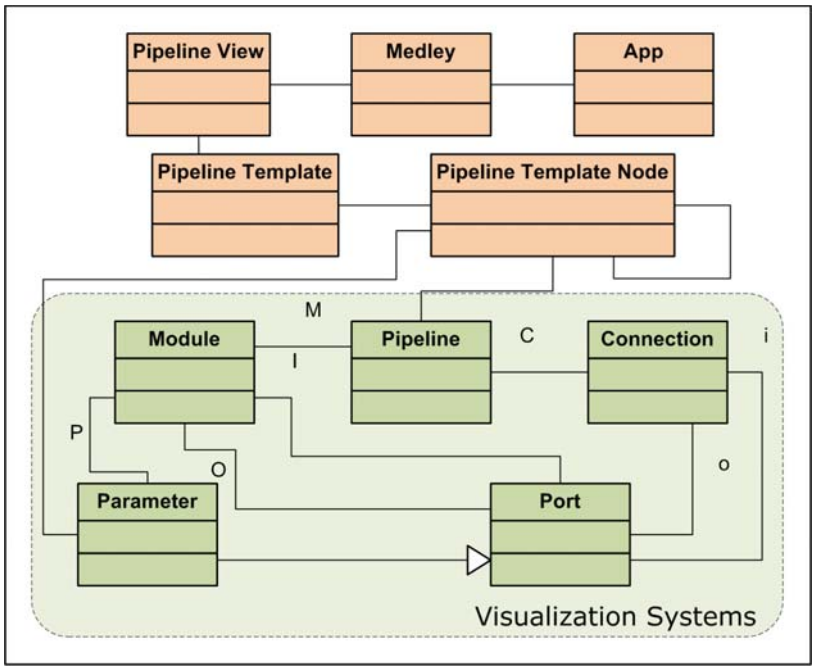

Figure 3.1. Conceptual schema of the main concepts in VisMashup.

assigns value v to parameter p in pipeline d, provided that the types of v and p are compatible.

The operation $substitutePipeline_d(M_s, d', f)$ replaces the subpipeline induced by the modules in $M_s \subset M$ by the pipeline d', using the mapping function f to rebuild the connections from d' to M. Let $C_s \subset C$ be the set of connections which connect modules in M to modules in M_s. Given a connection $(o_s, i_s) \in C_s$, where o_s is an output port of a module m_{source} *in* M, and $m_{source} \notin M_s$, i_s is replaced by the appropriate port in d', $f(i_s)$, provided the types are compatible. For connections $(o_s, i_s) \in C_s$, where i_s is an input port of a module m_{source} *in* M, and $m_{source} \notin M_s$, o_s is replaced by the appropriate port in d', $f(o_s)$, provided the types are compatible.

Figure 3.2 shows an example of the *substitutePipeline* operation.

3.1.3 Templates, Views, Medleys and VisMashups

Visualization pipelines can be very complex, making it difficult for users other than the original developer to modify them. Consider, for example, the relatively simple pipeline shown in Figure 3.3(a). This pipeline has 6 modules and tens of parameters. To explore the visualizations that can be derived by this pipeline, the user must be able to identify the relevant parameters that relate to a particular visualization feature, know the ranges

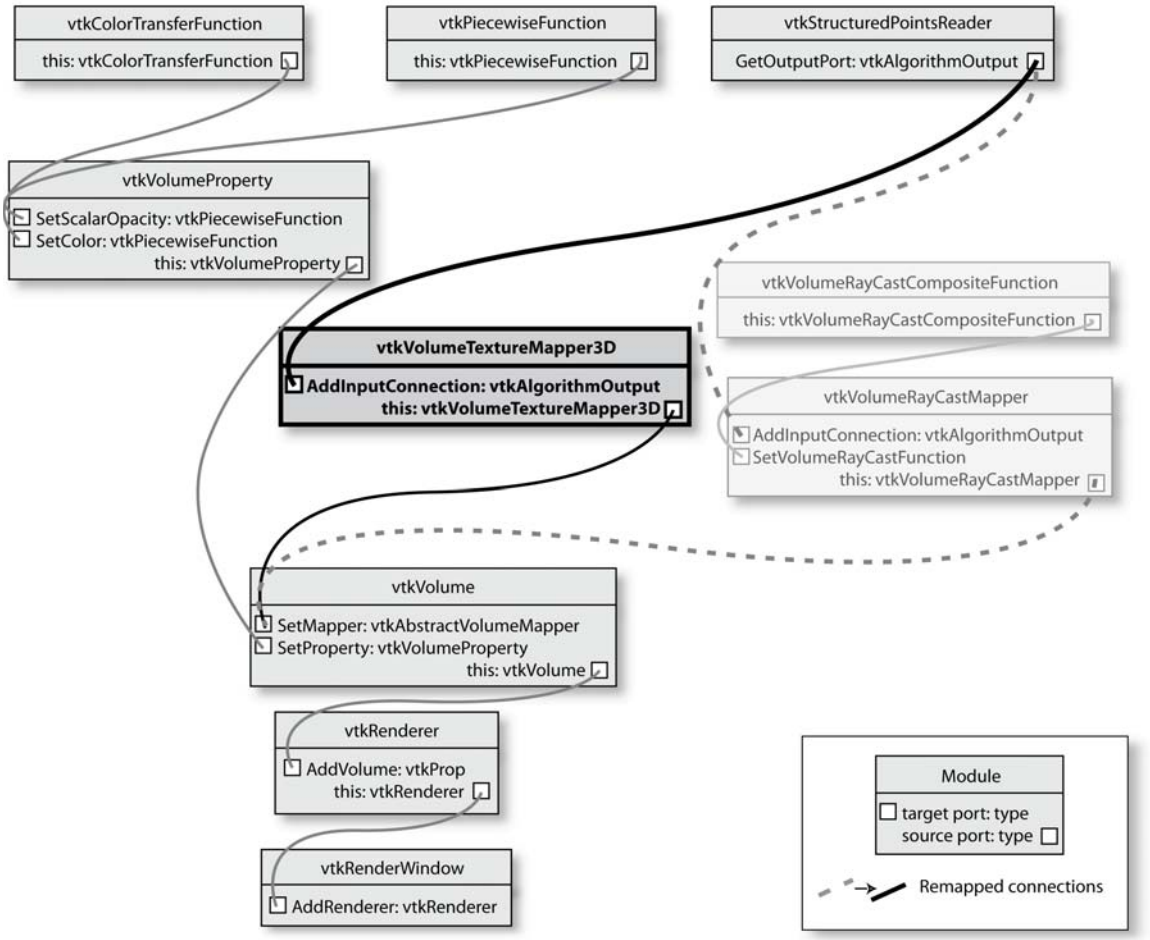

Figure 3.2. Example of applying substitution for changing the volume rendering technique of a VTK pipeline. The modules implementing a ray casting technique (*vtkVolumeRayCastMapper* and *vtkVolumeRayCastCompositeFunction*) are replaced by a module that uses 3D texture mapping (vtkVolumeTextureMapper3D). Dashed connections are remapped to the bold connections.

of values that can be used for individual parameters as well as valid combinations of values for different parameters. This can be very challenging, in particular for users that do not have detailed knowledge of visualization techniques and of the individual VTK modules. An important goal of VisMashup is to simplify this task and provide the end-user the flexibility to explore different visualizations in a more intuitive way. Below, we introduce concepts and operations that allow a pipeline designer to create simplified views of pipelines and pipeline collections (medleys) which abstract away details that are not important for a given task. Using these operations, the designer may also select and expose configurable

components of the pipelines which will guide the end user in providing valid inputs as well as experiment with different pipeline variations.

3.1.3.1 Pipeline Templates

We introduce the notion of a *pipeline template* as an extension of a pipeline that allows users to define reconfigurable pieces of the pipeline in a hierarchical way. Users can select and label parameters or subpipelines using a nomenclature that is meaningful for a given application or task. Figure 3.3(b) shows a pipeline template generated for the VTK pipeline in Figure 3.3(a). In order to give the end user the ability to modify the colors and rotations, the designer selected to expose the colors in the `vtkProperty` module with the parameter names `color_R`, `color_G` and `color_B`; and the X and Y rotations in `vtkActor` as `rotateX` and `rotateY`, respectively. In addition, selecting the subpipeline containing the module `vtkCylinderSource` allows this subpipeline to be replaced by other pipelines through the customized application.

The root of the template hierarchy represents the pipeline, and its children and descendants correspond to configurable parameters and subpipelines. We refer to each element in the template hierarchy as a *pipeline template node*. Nodes that correspond to subpipelines are represented as rectangles and parameters as ellipses. Note that labels are unique in a given hierarchy level. By representing the template as a hierarchy, our approach is able to handle arbitrary nesting of pipelines.

Pipeline template nodes provide the same operations of a pipeline as well as other specific operations for labeling and removing labels, for creating, adding and removing child nodes, creating and removing connections between template nodes and between template nodes and modules, and for materializing a pipeline.

3.1.3.2 Pipeline Views

Whereas in a pipeline template, important and configurable elements (*i.e.,* parameters and subpipelines) are selected, in a *pipeline view*, a new (abstract) module is created that hides all details of the underlying pipelines, including modules, connections and parameters that are deemed as unnecessary for deriving a set of visualizations required by an end user.[1]

As we describe below, we use *medleys* as a mechanism to synchronize pipeline parameters and combine multiple views.

---

[1]Pipeline views in VisMashup are similar to macro pipelines in AVS [77].

More formally, a *pipeline view* d_v is a projection of a pipeline d, where only a subset of the pipeline elements (modules, parameters and subpipelines) are exposed for direct interaction. For example, a user cannot change a parameter if that parameter is not exposed by the pipeline view. However, a pipeline view still keeps a reference to the original pipeline so the operations applied to the view can also be applied to the pipeline, including execution. We refer to the exposed elements as *variables*.

Notice that a pipeline view can be naturally built on top of a pipeline template. In fact, the view parameters and configurable subpipelines are also represented as a hierarchy that mirrors the one for the template hierarchy. Besides setting the visibility of template nodes and parameters, a developer can also specify a list of *suggestions* for the values a variable can take—these suggestions are used to derive application widgets that enable users to conveniently select a value from the list and *bind* it to the variable (see Sec. 3.2). Figure 3.3(c) shows a view (`DrawShape`) derived from the template in Figure 3.3(b). In addition, the view also contains suggestions for alternative subpipelines that can be used to bind the `input` variable, *i.e.,* `vtkCylinderSource` and `vtkCubeSource`. When defining these suggestions for template nodes, the developer also specifies how these suggestions connect in the parent template. It is possible that a given template node can be bound in different ways and the presence of some variables in a pipeline view is determined by the bindings used for their parents. In the example shown in Figure 3.3(c), the developer defined different types of sources, each containing their own set of parameters. There are two possible bindings suggested for the variable `input`: the subpipelines `Cylinder` and `Cube`. In the pipeline view shown, where `Cylinder` is used, its parameter (*i.e.,* `resolution`) is added to the view. If `Cube` were chosen, its parameters would be included instead.

3.1.3.3 Medleys

For exploratory visualization tasks, a user often needs to create and manipulate a set of pipelines. For example, to compare different isosurface extraction algorithms, several pipelines need to be created, including one pipeline per isosurface algorithm, and a pipeline that produces a histogram which accumulates the quality information for each mesh. Figure 3.4 illustrates two visualizations and their corresponding quality histograms.

In order to support the construction of applications that require the combination of multiple pipelines, we introduce the notion of a medley: a *medley* $\mathbf{M}$ is a collection of (related) pipeline views. VisMashup supports a set of operations for manipulating the views in a medley. One such operation is *synchronization*. A variable x in a pipeline view $d_v \in \mathbf{M}$ can be synchronized with any variable x' in another view $d'_v \in \mathbf{M}$ if x and x' have

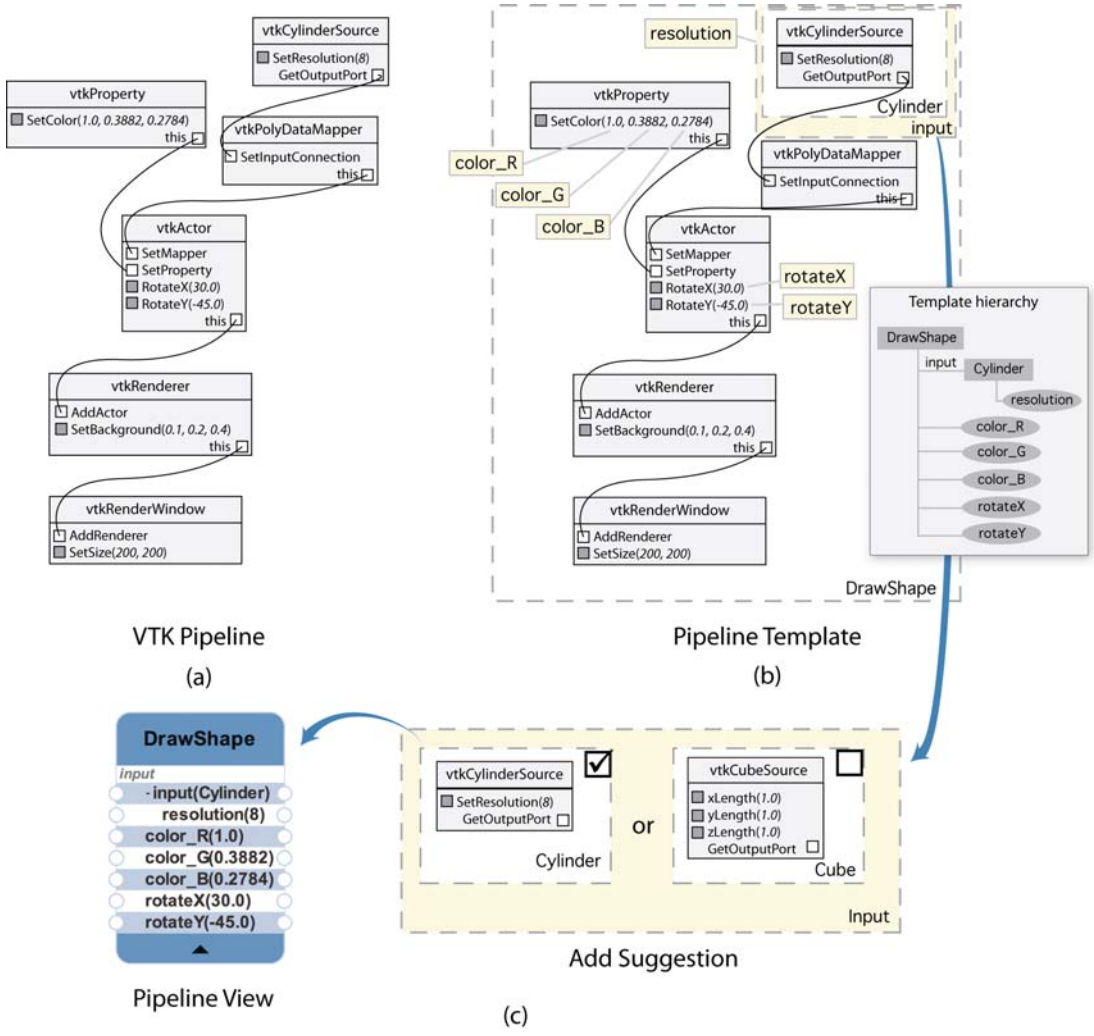

Figure 3.3. Example of how pipeline views are created. (a) Original VTK pipeline that draws a simple cylinder. (b) The user marks configurable pieces of the pipeline to create a template. (c) A pipeline view is created based on the pipeline template. The user defines template node *Cube* as another suggestion for variable input and *Cylinder* as the default value.

the same type. If x and x' are synchronized, binding either to a value v sets both variables to v.

The ability to synchronize variables is useful, in particular, for tasks such as comparative visualization, since it helps ensure that all parameters across different pipelines whose values should be the same are indeed the same. Consider, again, the isosurface example. Given a medley that contains the views for the pipelines that use the different isosurface algorithms, by synchronizing their input file and contour values, in a single step, the user could set these parameters and they would be automatically propagated to the two pipelines. Furthermore,

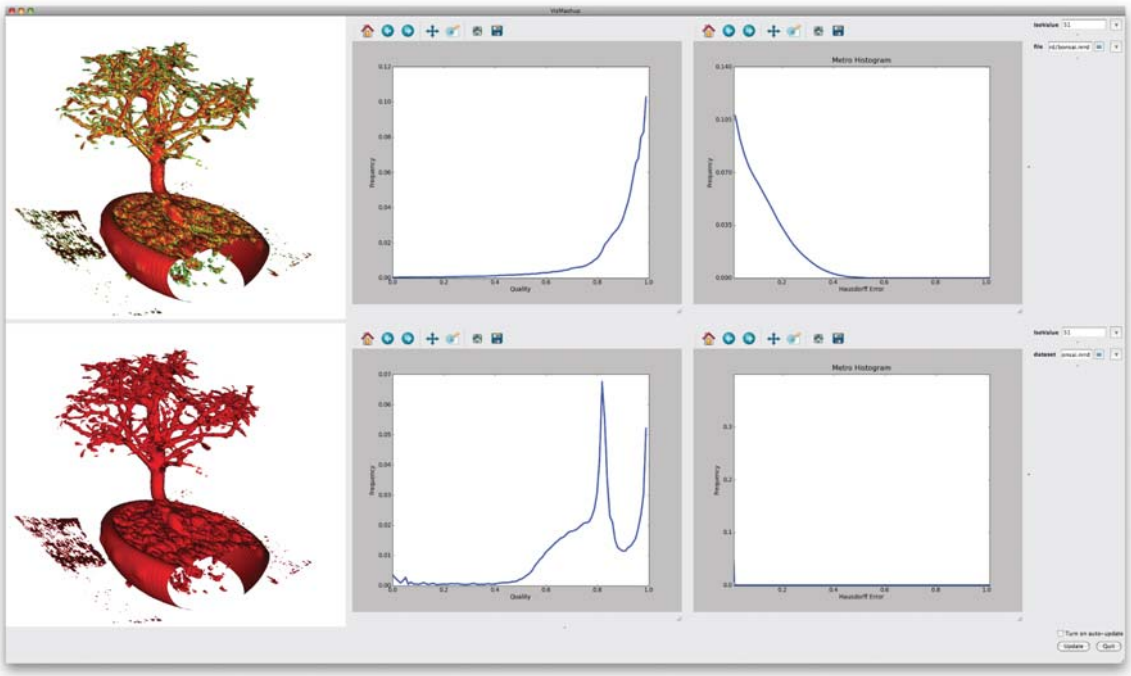

Figure 3.4. Comparing visualizations derived by two different isosurface algorithms and their associated quality histograms.

synchronization enables a user to efficiently try out different configurations. Instead of setting values for each pipeline individually—which can be both time consuming and error prone, the value for a parameter is set only once and it is automatically propagated to all synchronized variables in multiple views. Another possible operation that can be applied to a medley is *composition*. Composition can be achieved by synchronizing an output port of a view to the input port of another. In our example, composition could be used to pass the mesh derived by the two isosurface views to the view that derives the quality histograms. Note that because pipeline views reduce the number of components that are exposed for modification, they make it easier to identify how pipelines can be integrated and synchronized.

3.1.3.4 VisMashups

A *vismashup* (application) is a flexible GUI automatically generated from a medley specification. Instead of interacting directly with a dataflow network or a very general and complex GUI, users manipulate and execute a set of pipelines in a medley through a small number of graphical widgets (see Chapter 6 for examples of vismashups). Application maintenance is simplified, since when the underlying medley changes, the GUI can be

automatically updated. Furthermore, besides setting parameter values and synchronizing parameters, users can customize a mashup; they can hide, show and move widgets around. Changes made to the pipeline views through the mashup are propagated down to the pipeline level and sent to the visualization system for execution.

An important component of the mashup generation and execution subsystems is that they maintain detailed provenance information. VisMashup maintains provenance of the mashup design process—it keeps all versions and variations of a mashup created as the visualization expert interacts with a user, using a model similar to the change-based provenance proposed in [27]. Besides, log information of the use of a mashup is also kept (*e.g.,* when the mashup was executed, the values provided). The log information can be used to further refine the views and medleys.

3.2 The VisMashup System

As shown in Figure 3.5, the VisMashup system consists of a set of components that support tasks an application designer has to perform to construct a custom application. The *Pipeline acquisition and analysis* component allows designers to query and mine pipeline collections and their provenance [26]. This enables them to identify relevant pipelines, discover interesting parameters that should be exposed in the application, as well as a set of valid values that can be used for these parameters. Through a point-and-click interface, the *Template/pipeline view creation* component allows a designer to manipulate pipelines and create simplified views which expose only the pipeline components that are relevant for a given task. Pipeline views can then be combined into a medley (see Sec. 3.1.3) to create more complex, exploratory tasks (*Medley creation*). The *App generation* component uses the medley specification to automatically generate the application and associated user interface. VisMashup also maintains detailed provenance information of the application development process and use (*Medley Provenance*). As we discuss in Section 3.1, this information can be used to further refine and improve the derived apps.

We have built a prototype VisMashup system using Python, Qt, PyQt and VisTrails as the underlying dataflow system. VisTrails supports a flexible set of libraries, including VTK, ITK, matplotlib, SciPy and Web services. The implementation of VisMashup's main components and their user interfaces are described in detail below.

3.2.1 Pipeline Acquisition and Analysis

Although the pipelines for a mashup can be built from scratch, often, as visualization experts (designers) interact with domain experts (users), they construct a series of pipelines

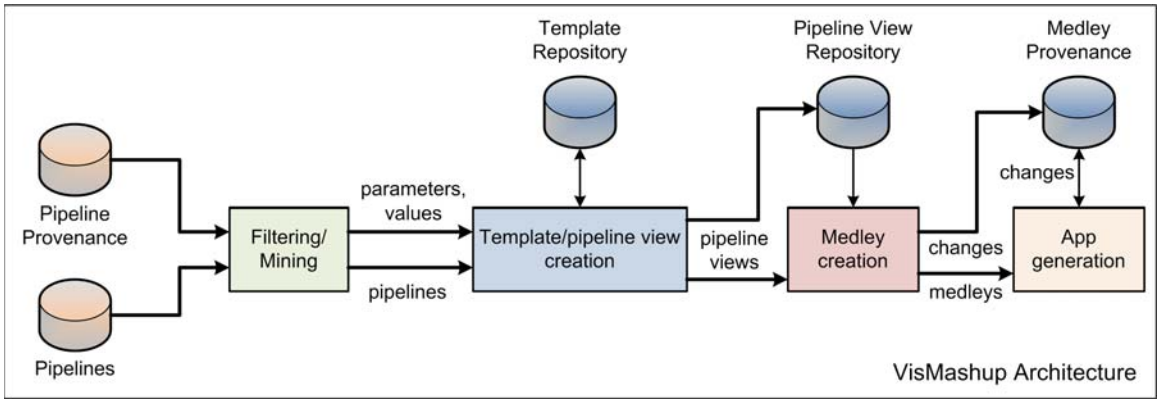

Figure 3.5. The VisMashup Architecture.

to explore different aspects of a data set, or different ways to visualize the data. The actual pipelines and their provenance, including information about the different runs, parameters set and their values, can be useful for designing the views that serve as the basis for the custom application. The Pipeline Acquisition and Analysis component allows the designer to explore a pipeline collection to identify relevant pipelines as well as identify important parameters and associated values.

Based on the metadata and provenance associated with a pipeline, we provide a filtering mechanism for developers to select pipelines satisfying simple conditions (see Figure 3.6). These conditions can be based on *time* (when the pipeline was created), the *user* that created the pipeline, *execution* (whether a pipeline was successfully executed and when), *tags* and *notes* associated with a pipeline. These rules are compiled into a simple regular expression matching over the text descriptions of the pipelines. Note that it is also possible to integrate more advanced interfaces that allow structural queries over the pipelines (*e.g.,* [62, 5]). Besides filtering, this component also provides an algorithm to help organize the pipeline collection into groups and extract information about parameter changes (see below). If there is provenance information about parameters that have been set across different executions of a pipeline and their values, this information can be explored both to suggest which parameters to expose in the pipeline view as well as to create a range of values that can be used as suggestions for the exposed parameters (Sec. 3.1.3).

Given a set of pipelines D, the procedure Group-and-Mine-Parameters initially groups together pipelines that have the same structure (modules and connections), *i.e.,*iso-morphic graphs (line 2).

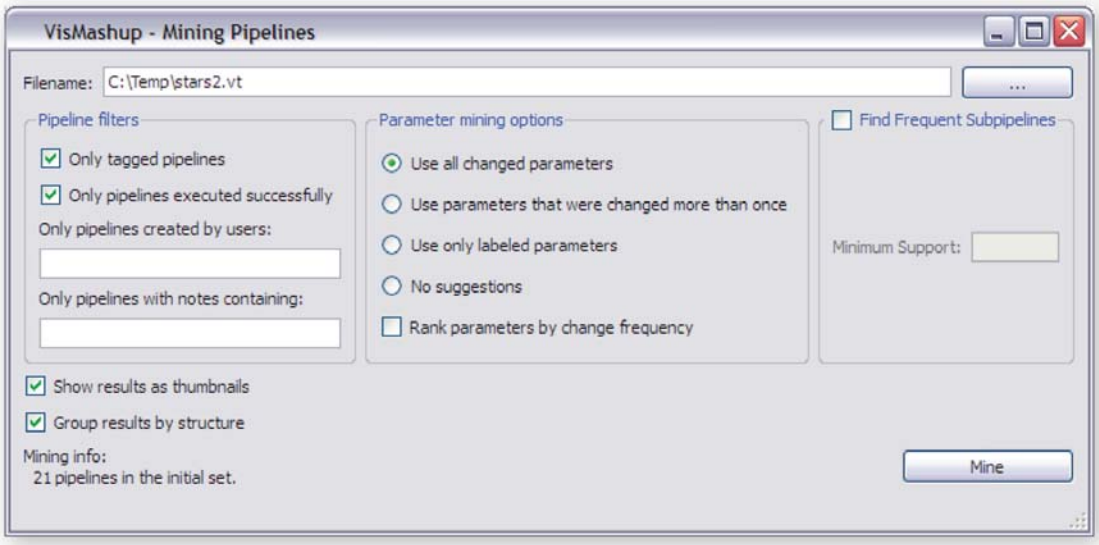

Figure 3.6. Window to control the mining options and filtering conditions to select pipelines.

For each group, the parameters and values stored in the pipelines and in their provenance (if available) are extracted and attached to the group (lines 3-12). Finally a *Hasse diagram* [6, 76] of the groups is generated by calling procedure BUILD-HASSE-GRAPH. The Hasse diagram is a tree where each node corresponds to a group of isomorphic graphs, and edges between a parent and a child node indicate containment. It shows how pipelines are related with respect to structure: the graph of a parent node is contained in the graph of the child node. Figure 3.7(a) shows an example of a Hasse diagram for a collection of 21 pipelines. Each pipeline is identified by a label and a thumbnail of the visualization it generates. Although the Hasse diagram resembles an image graph [43], the latter only captures the relationship between visualization parameter changes and are created for a single pipeline.

```
GROUP-AND-MINE-PARAMETERS(D)
 1  hasse-nodes ← MAKE-NODES-FROM-PIPELINES(D)
 2  hasse-nodes ← MERGE-WITH-SAME-STRUCTURE(hasse-nodes)
 3  for n in hasse-nodes
 4     n . param-list ← DICTIONARY()
 5     for d in n . pipelines
 6        params ← GET-PARAMS-FROM-PROVENANCE(d)
 7        params ← params + d . parameters
 8        for p in params
 9           if p . name not in n . param-list .KEYS()
10              n . param-list[p . name] = [ ]
11           n . param-list[p . name] ← p . value)
12  BUILD-HASSE-GRAPH(hasse-nodes)
```

To discover the containment relationships, it is necessary to compute subgraph isomorphism between all pipeline group pairs. For our prototype, we implemented the algorithm proposed by Ullman [76]. We note that, although computing subgraph isomorphism is computationally hard for general graphs,[2] we have observed that, in practice, this computation can be efficient for pipelines represented as dataflows. Just to give an idea, for a collection of over 7,000 pipelines, it took 140 seconds to generate their Hasse diagram. Besides, even if this computation takes long, it can be done in an offline process once for a collection, and be re-used multiple times.

The interface for the Hasse diagram is interactive, with zoom and pan functionality. This diagram gives a developer an overview of the collection and shows which pipelines share the same structure but have different parameter values. By clicking on a node of the diagram, the developer has access to another screen that provides details about all the pipelines in that group. The pipeline is displayed together with all the parameters that were changed, the number of times they were changed, and the set of values used to bind them. If the pipeline stores information about its executions, a plot of the execution times is also shown. This information is then used to help the construction of pipeline templates and views (Figure 3.7(b)).

3.2.2 Template and Pipeline View Creation

After selecting the interesting pipelines and extracting information about important parameters and values, the developer can use the *Pipeline Template Editor* to define the configurable pieces of the pipelines, by labeling subpipelines and parameters. To select a

[2]This problem is trivially reducible from the MAX-CLIQUE problem, a well-known NP-complete problem [33].

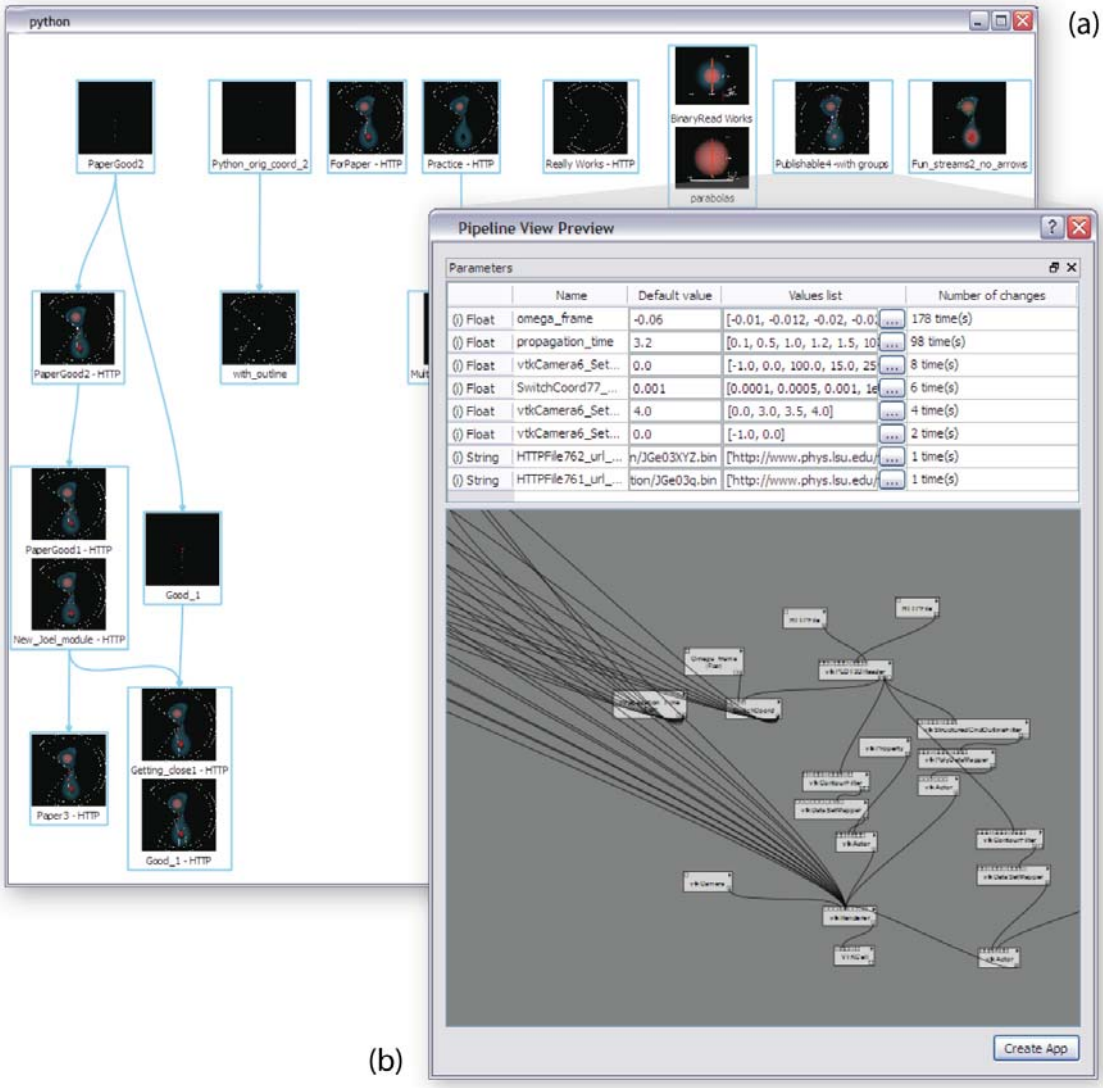

Figure 3.7. VisMashup Mining Interface: (a) Relevant pipelines organized in a Hasse Diagram; (b) Examining the parameters returned from the mining process using the Template/Pipeline View Editor.

pipeline to be used as a template, the designer simply clicks on the corresponding node in the Hasse diagram. This causes the Template Editor to be invoked. When the developer labels a parameter or a subpipeline, all the relevant information associated with these components that had been collected during the mining process is automatically added to the template (*e.g.,* all the values associate with the parameter). This is illustrated in Figure 3.7(b), which shows a set of parameters extracted from one of the pipelines in Figure 3.7(a).

Once a template is created, one of the operations supported by the Template Editor is view creation. While configuring a view, users can set the visibility of the parameters and configurable subpipelines, as well as select suggestions from the list stored in the template. Note that the information about both values as well as change frequency is very useful for configuring a view. Clearly, parameters that have had many different values are good candidates for being exposed. As shown in Figure 3.7(b), the value for parameter `omega_frame` was modified 178 times, while the parameter `propagation_time` changed 98 times. The values for these parameters can be used to create a list of suggestions for variables in the view. As we describe below, suggestions are included in the mashup to guide the user interaction, *e.g.,* besides typing a value in the interface, the user can also select a value from a list of suggestions. Note that both templates and views are stored in a repository where they can be accessed at a later time.

3.2.3 Medley Creation and Mashup Generation

To combine pipeline views in a medley, the designer uses the *Medleys Editor*. The views stored in the Pipeline View Repository are displayed on a panel and they can be dragged and dropped on a canvas. Once on the canvas, the medley operations (synchronization and composition) can be applied to the views. A screenshot of the Medleys Editor is shown in Chapter 6.

Each variable in a view has an associated handle—the circles on the left and right of each variable name in the pipeline view in Figure 3.3(c). By connecting the handles for two variables in distinct views, their values are synchronized. To simplify the task of identifying variables to be synchronized, when the designer starts a connection, all the variables that are compatible with that variable are highlighted.

VisMashup uses the change-based provenance model [27] to capture all the changes made to a medley. We designed and implemented a change algebra that captures the actions applied to a medley (*e.g.,* synchronize variables) as well as to its components (*e.g.,* hide/expose view variables).

A mashup is generated from a medley using the following algorithms:

GENERATE-APP(*medley*)

```
1  widgets = TREE()
2  window = CREATE-WINDOW(medley)
3  for pipeline-view in medley. pipeline-views
4     BUILD-GUI-WIDGETS(pipeline-view .root, window, widgets)
5  for sync in medley .syncs
6     LINK-WIDGETS-FROM-SYNC(sync)
```

```
BUILD-GUI-WIDGETS(template-node, parent, node)
1  widget = CREATE-CONTAINER(parent, template-node)
2  child = MAKE-TREE-NODE(widget)
3  node .ADD-CHILD(child)
4  for var in template-node .children
5     BUILD-GUI-WIDGETS(var, widget, child)
6     for par in var .parameters
7        widget-type ← GET-FROM-REGISTRY(par . type)
8        wp ← CREATE-WIDGET(widget-type, widget, par)
```

GENERATE-APP automatically derives a user interface for a given medley. The interface consists of a set of widgets represented as a tree whose root corresponds to the mashup window. For each view in the medley, the algorithm iterates through the view components and derives their widgets by invoking BUILD-GUI-WIDGETS (lines 3-4). After the widgets are created, the widgets that correspond to synchronized variables in the medley are linked (lines 5-6) —this ensures that when the value is set for a widget w, the same value is propagated to all widgets linked to w.

BUILD-GUI-WIDGETS generates the widgets for a pipeline view. It first creates a container for the view (line 1), and then recursively builds a widget tree following the view template tree (Sec. 3.1.3). The widget types associated to the types of the variables are stored in a *Registry*. A widget is selected for a variable based on the mapping stored in the Registry. In the current implementation, the Registry is populated based on the types available in VisTrails, where designers can define their custom types and widgets. A planned extension to VISMASHUP is to allow the designer to edit the Registry and configure the choices independently of the underlying dataflow system. In addition to the widgets, canvases are created to display each of the outputs of a view. Note that the algorithms above are used to generate both client applications as well as Web-based mashups. The main differences between the two kinds of apps is that the communication between a Web app and a medley is done via a Web service and it provides a more limited interaction with the visualizations.

3.2.4 Mashup Interaction

When a mashup is created, a default layout is selected for the widgets. By activating the mashup edit mode, the user can move the widgets around, hide and show widgets, enable and disable synchronizations. The new custom layout is saved as an attribute of the medley associated with the mashup, and the next time the mashup is invoked, the custom layout is loaded.

When a container widget changes its value (*e.g.,* changing from Cylinder to Cube in the pipeline view described in Figure 3.3), the mashup window needs to be updated accordingly. As the widgets in the window follow the hierarchy of the pipeline template, updates can be applied locally to that node—it is not necessary to update the other widgets in the window. The update procedure is shown below.

```
UPDATE-APP(window, tree − node)
1  parent ← tree-node .parent
2  template-node ← tree-node .widget. template-node
3  BUILD-GUI-WIDGETS(template-node, parent.widget, parent)
4  REMOVE-LINKS-FROM-WIDGETS()
5  for sync in window.medley.syncs
6          LINK-WIDGETS-FROM-SYNC(sync)
```

Note that before UPDATE-APP is called, the medley associated with that window detects the change events and stores the corresponding provenance information. Since variables can be removed, the synchronization information may need to be updated as do the associated links between widgets. Finally, executing a mashup causes all pipeline views to update their corresponding pipelines using the values on the medley (the same as in the GUI) and trigger their execution.

3.3 MobileMashups for Portable Devices

The explosion in the use of programmable portable devices and the ability they give people to input and access information anytime and anywhere has the potential to fundamentally change scientific applications. Not only are these devices becoming ubiquitous, but they also provide advanced functionality not generally available in traditional desktop computers. In particular, they provide mobile access to the Internet, have touch screens, accelerometers, and GPS capabilities. Although several consumer applications have been deployed, there are many challenges that need to be overcome for their use by scientists.

While applications can be built from scratch and designed specifically for portable devices, doing so is expensive and does not scale. In this Section, we explore MobileMashups, a more cost-effective alternative: the use of VISMASHUP to simplify the creation of customized mobile applications. The infrastructure partially automates the generation of interfaces suitable for small devices through which users can steer the underlying workflows. Below we discuss the details of the implementation.

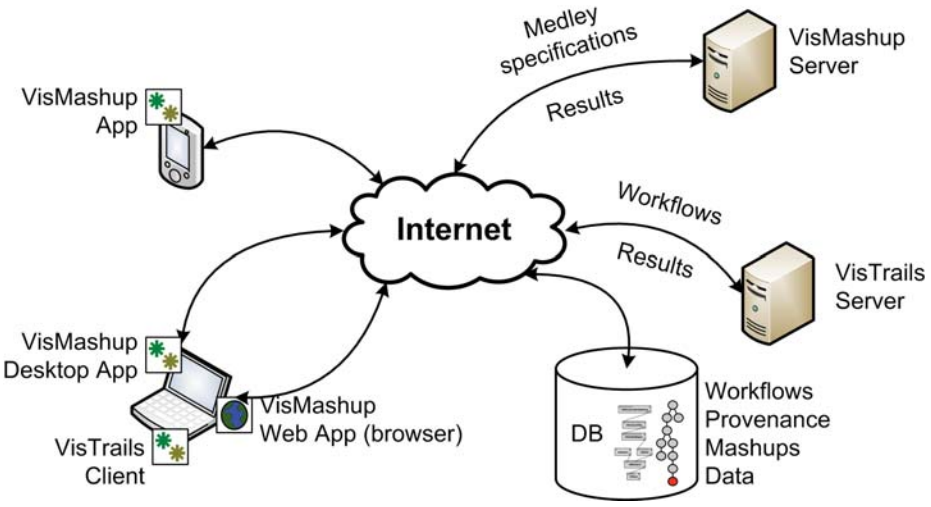

Figure 3.8. Overview of MobileMashups. Mashups can be deployed in three different types: Desktop App, App and Web App. The VisMashup server exchanges medley specifications and results with the VisMashup clients and workflows and results with the VisTrails server. Mashups, workflows and provenance data are stored on a database accessible to servers and VisTrails clients.

Both VisTrails and the VisMashup system were extended in order for them to work on portable devices. Our idea is to use VisTrails to create the workflows and VisMashup to deploy them on the portable devices. Figure 3.8 displays an overview of the system.

We first added to VisTrails a package specific to portable devices, called *mobile*. On the VisTrails side, each module in the mobile package works as a surrogate for another module that will be invoked depending on the device where the vismashup is deployed. This allows a package that is general and therefore can be deployed on different portable devices. The modules and their description are listed on Table 3.1.

We also added to VisMashup a set of surrogate widgets corresponding to the new modules in the mobile package. These surrogate widgets will be used in the medley specifications and therefore will be implemented by the VisMashup app running on the portable device.

In Chapter 6 we describe how the Location, Audio and Image widgets were implemented on Apple's iPhone. The *Location* widget can automatically get information from the device's position with the user's permission or it can be manually entered by the user. The *Image* widget can access images existing on the user's albums or take new pictures using the camera. The *Audio* widget allows users to record a voice note or any other sound.

An example of a medley specification is shown in Figure 3.9. Medleys specifications are sent back and forth between clients and server. The clients populate the *val* attribute of

Table 3.1. List of modules in the mobile VisTrails package and their descriptions.

Module	Description
Audio	Represents graphical content encoded in Base64 scheme.
Image	Represents image content encoded in Base64 scheme.
Location	Represents geographical coordinates, altitude of the portable device along with values indicating the accuracy of the measurements and when those measurements were made. In devices with a GPS, this module also reports information about the speed and direction in which the device is moving.
LocationCoordinate2D	Represents a geographical coordinate (latitude and longitude) using the WGS 84 reference frame.

each parameter which will be used by the VisMashup server to configure the workflows for execution. The full process is summarized below:

1. Workflow designers create workflows using VisTrails with the *mobile* package.
2. Workflow designers use the VisMashup to create simplified views, generate medley specifications and store them on the VisMashup server.
3. Users install the VisMashup app on their portable devices.
4. The VisMashup app requests through the Internet a medley specification and downloads it from the VisMashup server.
5. The medley specification is loaded on the device. The surrogate widgets are replaced by actual widgets, the GUI is displayed on the device and it is ready to have data input.
6. After entering the requested information, the VisMashup app sends the medley specification back to the VisMashup server to be executed.
7. The VisMashup server instantiates the VisTrails workflows populating them with the included parameter values and dispatches them to the VisTrails server for execution.

```
<?xml version="1.0" encoding="UTF-8"?>
<medley id="17" name="Register Observation"
type="vistrail">
<view name="Observation" version="45" vtid="14">
<alias id="73" name="username">
<component ctype="Parameter" id="73" maxVal="" minVal="" parent=""
pos="1" seq="False" spec="string" stepSize="" val="john"
valueList="" widget="text"/>
</alias>
<alias id="76" name="image">
<component ctype="Parameter" id="76" maxVal="" minVal="" parent=""
pos="4" seq="False" spec="image" stepSize="" val="" valueList=""
widget="image"/>
</alias>
<alias id="74" name="species">
<component ctype="Parameter" id="74" maxVal="" minVal="" parent=""
pos="2" seq="False" spec="string" stepSize="" val="hawk sp."
valueList="duck sp.,hawk sp.,swan sp.,waterfowl sp."
widget="combobox"/>
</alias>
<alias id="75" name="location">
<component ctype="Parameter" id="75" maxVal="" minVal="" parent=""
pos="3" seq="False" spec="location" stepSize="" val="" valueList=""
widget="geolocation"/>
</alias>
<alias id="77" name="audio">
<component ctype="Parameter" id="77" maxVal="" minVal="" parent=""
pos="5" seq="False" spec="audio" stepSize="" val="" valueList=""
widget="audio"/>
</alias>
<alias id="74" name="notes">
<component ctype="Parameter" id="78" maxVal="" minVal="" parent=""
pos="6" seq="False" spec="string" stepSize="" val="" valueList=""
widget="multiline"/>
</alias></view></medley>
```

Figure 3.9. Medley specification for the mashup described in Section 6.4

8. The VisTrails server executes each workflow and sends the results back to the VisMashup server.

9. The VisMashup server forwards the results to the VisMashup app that made the request.

10. The results are displayed on the VisMashup app.

CHAPTER 4

CROWDLABS: SOCIAL ANALYSIS AND VISUALIZATION FOR THE SCIENCES

Managing and understanding the large amounts of scientific data is undoubtedly one of the most difficult research challenges scientists are facing today. As large interdisciplinary groups work together, the ability to generate a diversified collection of analyses for a broad audience in an ad-hoc manner is essential for supporting effective scientific data exploration. Science portals and visualization web sites have been used to simplify this task by aggregating data from different sources and by providing a set of predesigned analyses and visualizations. However, such portals are often built manually, and are not flexible enough to support the vast heterogeneity of data sources, analysis techniques, data products and user communities that need to access this data. In this chapter, we describe CrowdLabs, a system that adopts the model used by social Web sites and that integrates a set of usable tools and a scalable infrastructure to provide a rich collaborative environment for scientists.

CrowdLabs combines benefits of social Web sites and science portals while at the same time addressing their limitations. Similar to social Web sites, CrowdLabs aims to foster collaboration, but unlike these sites, it was specifically designed to support the needs of computational scientists, including the ability to access high-performance computers and manipulate large volumes of data. By providing mechanisms that simplify the publishing and use of analysis pipelines, it allows IT personnel and end users to collaboratively construct and refine portals. Thus, CrowdLabs lowers the barriers for the use of scientific analyses and enables broader audiences to contribute insights to the scientific exploration process, without the high costs incurred by traditional portals. In addition, it supports a more dynamic environment where new exploratory analyses can be added on-the-fly.

Another important feature of CrowdLabs is the provenance support [19, 26]. Publishing scientific results together with their provenance, the details of how the results were obtained, not only makes the results more transparent, but it also enables others to reproduce and validate the results. CrowdLabs leverages provenance information (*e.g.,* workflow/pipeline specifications, libraries, packages, users, datasets and results) to provide a richer sharing

experience: users can search and query this information. In addition, provenance is made accessible through the Web site and an API. This allows users to connect results published in an article or wiki page to the pipelines and data served by CrowdLabs, greatly simplifying the process of creating provenance-rich publications.

The remainder of this chapter is organized as follows. We describe the main components of CrowdLabs in Section 4.1. Information on deploying CrowdLabs is given in Section 4.2.

4.1 System Overview

CrowdLabs was designed to enable large interdisciplinary groups to work together by enabling the generation of a diversified collection of analyses for a broad audience in an ad-hoc manner. It makes heavy use of provenance, workflow and visualization capabilities to create a unique scalable infrastructure for providing a rich collaborative environment for scientists. We take into account the requirements of computational scientists, such as accessing high-performance computers and manipulating large amounts of data. CrowdLabs architecture is depicted in Figure 4.1. The description of the main system components follows.

4.1.1 Client API

CrowdLabs provides a Web-based interface for sharing visualizations and collaborating with peers. The system includes a repository of visualizations, datasets and libraries, and while the CrowdLabs Web site provides a useful platform for sharing and collaboration, the social and provenance data can be useful in other contexts. In order to expose CrowdLabs resources to a diverse set of clients, the Web site employs a RESTful HTTP API [21]. This API identifies visualization and social resources, providing uniform resource identifiers (URIs) for clients to retrieve, add, update and delete them.

The data analysis and visualization resources defined by the system are vistrails, workflows, vismashups, packages and datasets, while the social resources include profiles, projects, groups and blogs. Adhering to the RESTful architecture, each of these resources has various different representations associated with them. Visualization resources might include provenance data, meta-data (modules, documentation, *etc.*), application files (vistrails, datafiles), as well as visualization results. Social resources might include blog posts, discussion topics and notices, along with ratings, tags and comments, which are linked to the visualization resources.

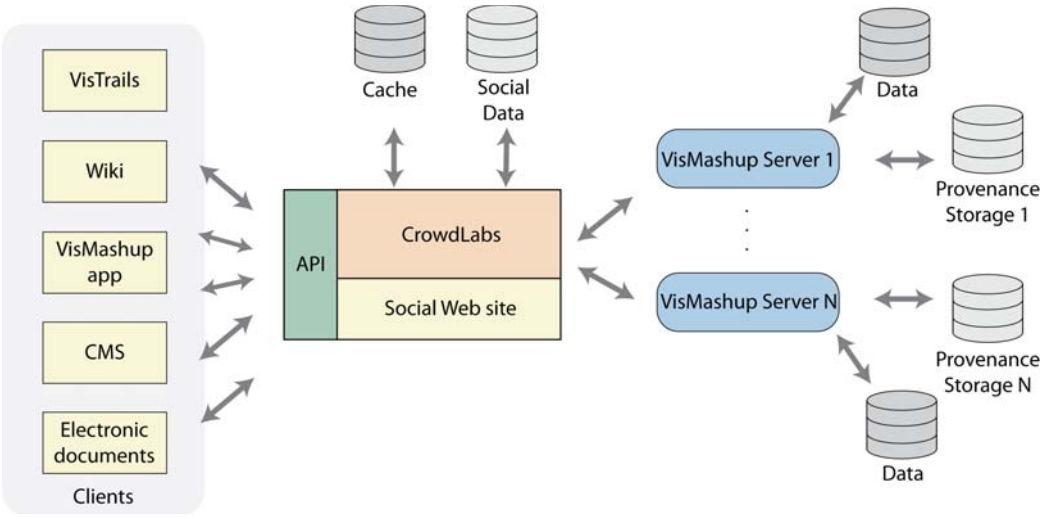

Figure 4.1. CrowdLabs system architecture.

For example, for a client to access the XML representation corresponding to the workflow with id 117, they would have to make a HTTP GET request to the URI *http://www.crowdlabs.org/vistrails/workflows/get_xml/117/*.

The RESTful API has enabled basic CrowdLabs functionality to be integrated into the VisTrails desktop application and other extensions. From the desktop, users can login, add and update vistrails and datasets (see Section 4.2.3) to CrowdLabs through the *Web Reposiotry Options* dialog from within the VisTrails application. The Latex extension described in Chapter 5 also uses the RESTful API for embedding workflows in PDF documents.

Following the example of myExperiment's Google Gadget and Facebook App [54], providing an API encourages developers to extend functionality and creates an open development environment.

4.1.2 Social Web Site

In the spirit of fostering collaboration, CrowdLabs incorporates a social Web site that is based around user-created content and social networking tools. All of the basic social networking capabilities available on other social Web sites, such as Many Eyes [78] and myExperiment [54], are present to users, who can make friends, join groups, write blogs and create projects, topics and wikis. In addition, users can add, edit and delete VisTrails-related data such as vistrails, workflows, vismashups, packages and datasets. Tied to each of these VisTrails related objects are ratings, tags, comments and projects. This social data not

only encourages an environment of user discussions and interaction, but enables the use of crowd sourcing, which can be used to find good-quality visualizations and improve documentation through user ratings, better categorize datasets and workflows by user tagging, and troubleshoot problems through comments and discussion topics. We also enable users to share their work off-site by providing syntax to embed interactive vismashups on Web sites as well as static visualization results on the Web, wikis and within LaTex documents (see Chapter 5).

4.1.3 Cache

CrowdLabs is set up to generate content dynamically. This creates potential efficiency issues, since some workflows can take a long time to run. Also, it is very important to avoid delays when presenting pages to users, otherwise, they can get discouraged, and avoid using the site. We use different forms of caching to speed up common operations.

In CrowdLabs, there are two types of cache: the results cache and the provenance cache. The results cache is used to store images and other files generated by workflows and vismashups. When there is a request for executing a workflow, for example, by accessing a page containing the details of a workflow or a page containing a vismashup, the system first checks if that workflow was executed before. If yes, it uses the files already in the cache, otherwise it will forward the request to the proper VisTrails server instance.

The provenance cache stores information about the workflows and vistrails uploaded to the CrowdLabs Web site. This information currently includes the named workflows in a vistrail, who created them and the packages and modules referenced in the workflows together with their documentation. The system takes advantage of the fact that VisTrails change-based provenance model records information about workflow evolution [27]: all the workflows are versioned. However, VisTrails client users might change a vistrail (*e.g.,* by adding new workflows) by connecting to the database directly. In this case, cache coherence is maintained by updating the cache whenever the workflow was updated in the database, which can be done by a simple query to the VisTrails server.

4.1.4 VisMashup Server

The *VisMashup server* is one of the most important components of CrowdLabs. It provides the link between the workflow provenance and the rest of the system. The VisMashup server is a multithreaded server that uses the XML-RPC protocol to answer client requests. The most common requests are the following: execute a workflow or a

vismashup, add or remove a vistrail or a vismashup from the database, get the packages and modules used in a vistrail or workflow and other information associated with the workflows.

One of the key features of the VisMashup server is that it maintains its own cache (separate from CrowdLabs results cache) for keeping the results of executed workflows or vismashups. When both components are in the same machine, CrowdLabs can be configured to use the VisMashup server cache to avoid redundant storage. Another key feature of the VisMashup server is the ability to start and to communicate with other VisMashup server instances over the same XML-RPC protocol. This allows the creation of clusters of servers that work transparently with the rest of CrowdLabs.

4.2 Deploying CrowdLabs

Depending on the particular application, it is possible to use different deployment configurations for CrowdLabs. For instance, see Figure 4.2. Here we will describe how the system is currently deployed on `www.crowdlabs.org`. We encourage readers to access the site, but bear in mind that it is constantly under development.

4.2.1 System Configuration

CrowdLabs is currently deployed as shown in Figure 4.2(a). The core system and the four instances of the VisMashup server share a 8-core Intel Xeon 2.66 GHz machine with 24 GB RAM running Linux. The system data and workflow provenance are stored in a MySQL database server also running on the same machine. The CrowdLabs Web site is implemented using Django Python Web framework and the VisMashup server is implemented in Python.

4.2.2 Projects and Servers

It is possible that users would like to organize their content into different projects. An example is the CMOP project (see Section 6.3), which contains all the vistrails, vismashups, workflows and the information they use together with the discussions and blog posts created about them. This allows for defining different levels of visibility: groups can have discussions and upload workflows that are only visible to the people involved in the project. The ability to selectively disclose information for people outside the group is extremely important for scientists, who may work for many years before deciding to release certain types of data.

Another advantage is that a project can have its own dedicated VisMashup server. This creates the possibility of having specialized servers for different types of workflows. For example, for the CMOP project, we are in the process of deploying a VisMashup server on one of their machines so users on CrowdLabs Web site can execute workflows and

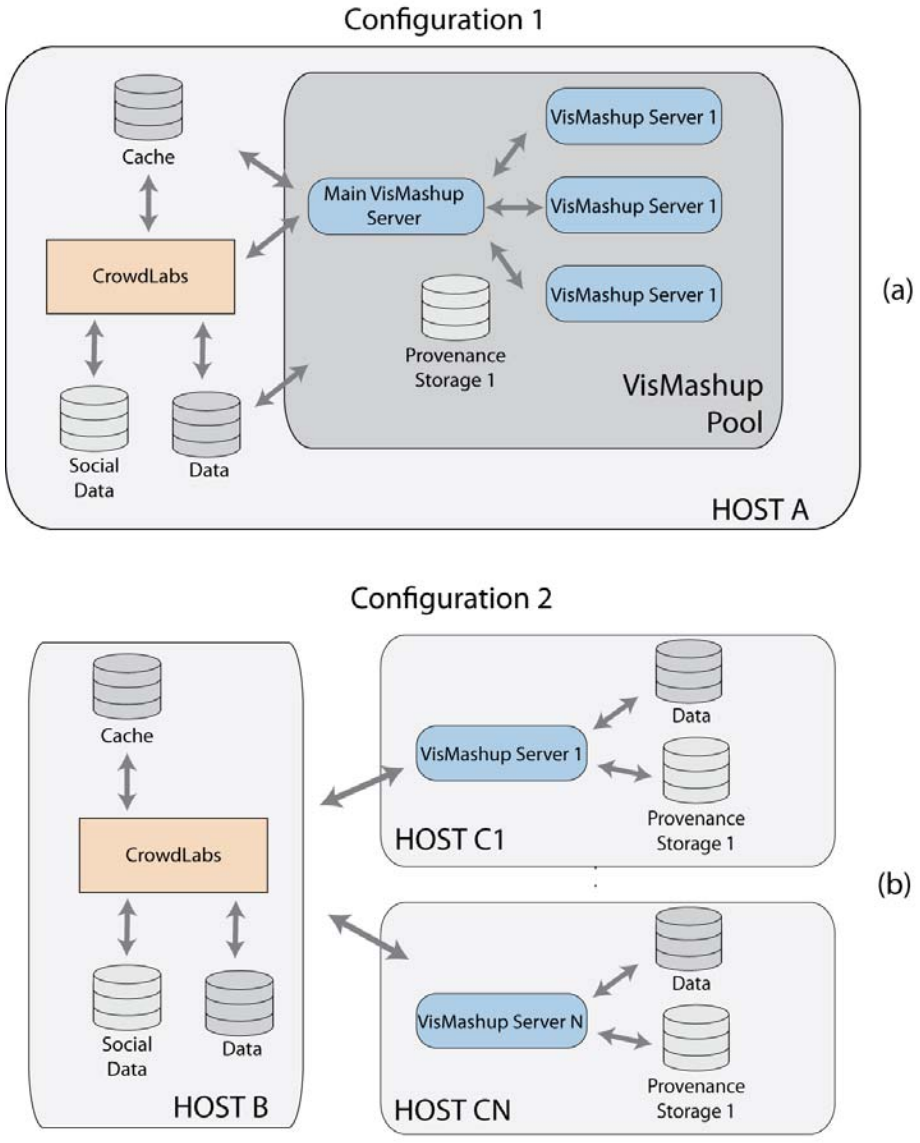

Figure 4.2. Different configurations of deploying CrowdLabs. (a) All the components are located on the same machine. (b) VisMashup servers execute on dedicated machines.

vismashups that access all the hindcasts and forecasts on CMOP's file servers, which are only accessible locally. These different servers allow the system to grow in functionality without compromising the overall performance.

4.2.3 Designing Shareable Workflows

Ideally, the workflows that belong to CrowdLabs would be executable anywhere, that is, they would be "shareable workflows." One of the key requirements to make this possible

is to abstract away references to the input data as to allow other users to execute those workflows.

In CrowdLabs, there are two different ways of making input files accessible. If the files are already available on the Web and can be downloaded from a url, users can use the *HTTPFile* module, available in VisTrails (see Figure 4.3(a)). The first time that users execute a workflow with a *HTTPFile*, VisTrails will download the file to their file system and cache it locally, so it can be reused the next time the workflow is executed. Another option is by uploading the input file to CrowdLabs, and this is done by either visiting CrowdLabs Web site and using the upload form or when designing the workflow in VisTrails, using the *RepoSync* module. By connecting the *RepoSync* between an input file and the module that uses it (see Figure 4.3(b)), VisTrails will ask the user to log-in on CrowdLabs and upload the input file, if it is not already on CrowdLabs. This is verified by comparing the checksum of the contents of the file, which is independent of file names. After uploading a file, the checksum is stored in the workflow specification. This way, when other users download the workflow and try to execute it on their computers, VisTrails will automatically download the file based on the stored checksum from CrowdLabs, if it is not available on the file system.

If a user decides to try a workflow that had a *RepoSync* module with a different input file available on the file system, VisTrails will ask the user how it should deal with the new file; if the user indicates that it is a completely different file, VisTrails will upload as a different file. If it is a new version of the previous file, VisTrails will upload as a new version of the file identified by the checksum in the *RepoSync* module.

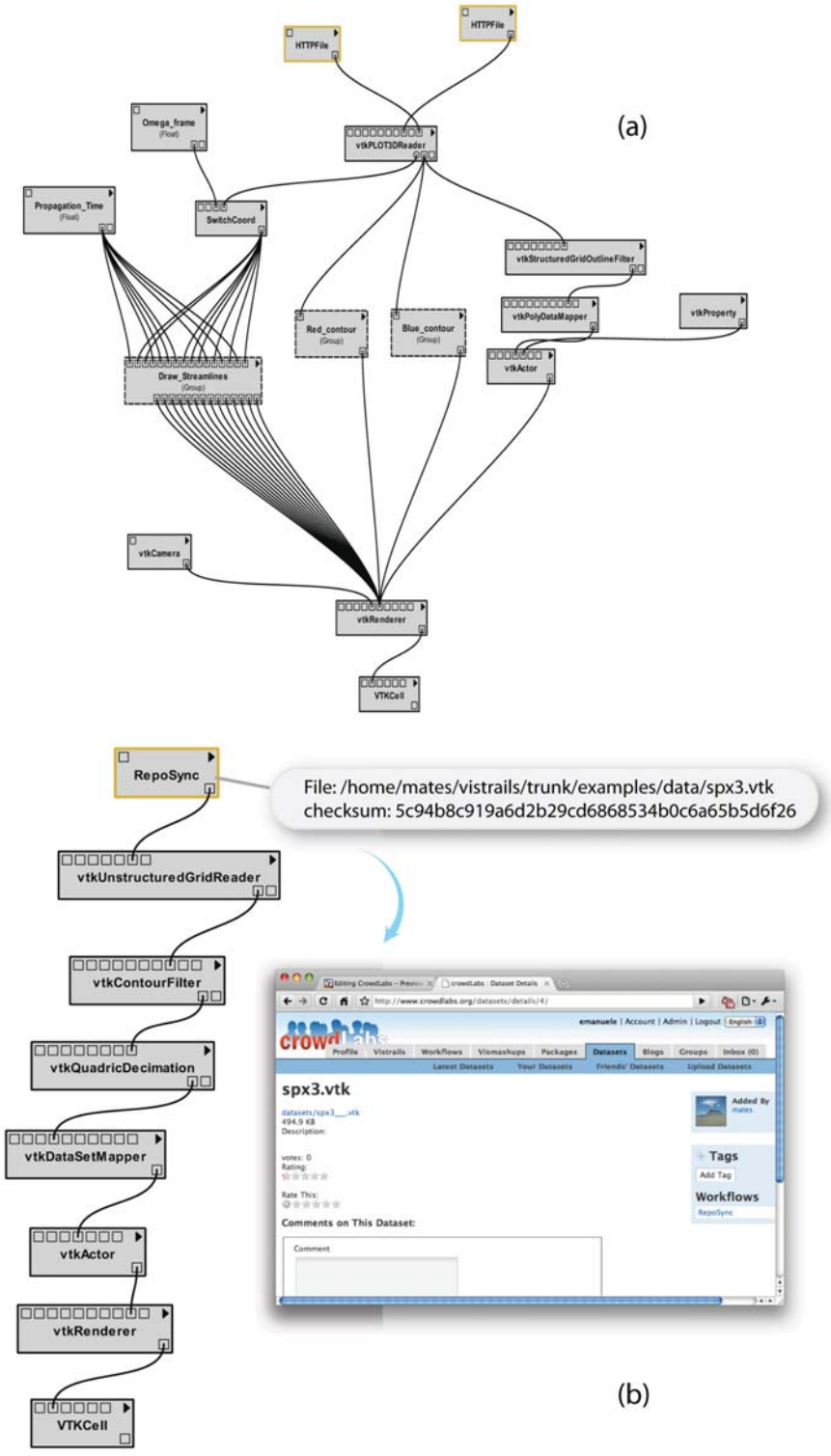

Figure 4.3. Making input files accessible in workflows: (a) Using the HTTPFile module (the input file is already available on the Web) or (b) Using the RepoSync module to upload the input file to CrowdLabs.

CHAPTER 5

PROVENANCE-RICH PUBLICATIONS

One of our goals with creating CrowdLabs has been to make it easier for scientists to share information and make their work more transparent. When publishing scientific results, it is important to describe the lineage of a result. Providing data sets and computer code allows scientists to verify and reproduce published results and to conduct alternate analyses. Unfortunately, providing data sets and computer code in a disjoint manner brings a burden to the readers: besides having to navigate between code, data and text, they have to manually enter the parameter values (when specified in the text). In the past years, interest in this subject has increased in different communities which led to different approaches for publishing scientific results (see [22] for an overview). CrowdLabs greatly simplifies the process of packaging workflows and results for publication. It supports the publication on wikis, other Web sites, or scientific documents. Our work allows users to create documents whose digital artifacts (*e.g.,* figures) include a deep caption: detailed provenance information which contains the specification of the computational process (or workflow) and associated parameters used to produce the artifact. An overview of the many different ways the provenance information can be used is shown in Figure 5.1.

Next, we describe the different mechanisms that CrowdLabs supports for sharing content.

5.1 Interactive Versus Static Content

CrowdLabs is able to support the generation of static content (e.g., images, animations, tables, XML pages) as well as interactive ones. Static content is generated directly from the workflow specification. In particular, we use VisTrails flexible spreadsheet infrastructure to generate the different types of output. Anything that can be displayed in a spreadsheet cell can be rerouted through CrowdLabs. Due to limitations of current browsers, it is hard to provide fully interactive content; in particular, this is quite hard to do for 3-D visualizations. Instead, we use the capabilities of VisMashup that allows for interactive

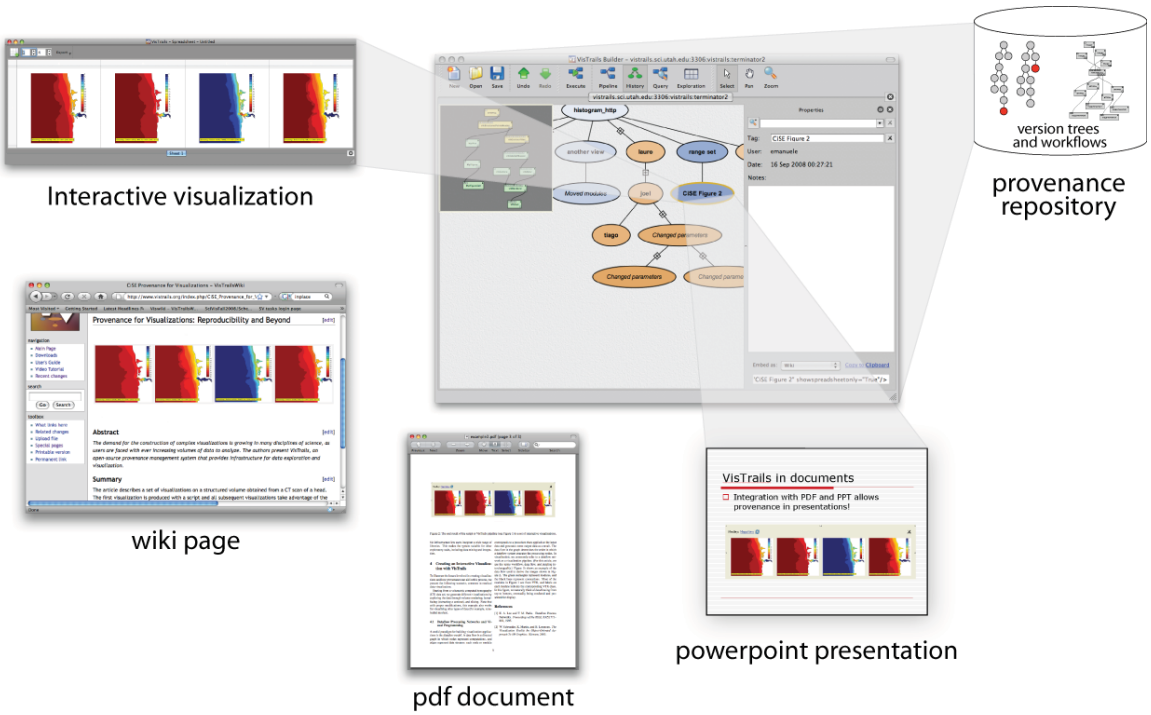

Figure 5.1. Overview of our approach for Provenance-Rich Publications. Digital artifacts include the workflow provenance information used to generate them.

widgets to be placed next to an output on a browser. As the user modifies the exposed inputs, the system computes the resulting visualizations, and makes it available.

5.2 Publishing on Wikis

CrowdLabs has been integrated with Wikis. Basically, one can embed either a static visualization or a VisMashup on a Wiki page. An example of this is shown in Figure 5.2(a). One of the very nice features of wikis is that they version all the changes to any particular page, allowing for the maintenance of a record of authorship. Our work fills an important gap in this infrastructure. Existing wikis are not able to version visualizations or other scientific results. In our work, we have integrated a provenance-enabled workflow system in a way that is fully compatible with the way wikis work, thus making it possible to create fully-versioned and reproducible Wiki pages. As we see later, this is already being used by one of our collaborators for writing a new book on astrophysics.

CrowdLabs uses the extension mechanism, available in most of the existing wiki engines, that allows defining and handling specific tags in the wiki markup language. In our case, when we find a $< vistrail/ >$ or a $< vismashup/ >$ tag, we build an XML-RPC call with the information provided in the tag, such as the vistrail and workflow identifiers, and submit

it to the VisMashup server for execution and replace the tag with the generated images. By using the *Embed this Workflow in a Wiki* link on any workflow page in CrowdLabs, users can copy and paste the wiki tags directly.

5.3 Publishing on Web Sites

Integration with other Web sites is even simpler than integrating with Wikis because it does not require modifications on the other Web sites for supporting this functionality. In the case of a workflow, CrowdLabs generates html markup for a static image with a link back to the workflow page and in the case of a vismashup, it generates html markup for an embeddable object containing the vismashup (in a similar way as YouTube videos can be embedded in Web sites). This is illustrated in Figure 5.2(b).

5.4 Publishing Scientific Documents

We believe that one of the most interesting applications of a system like CrowdLabs is the impact that it can have on printed media. So often, we see published scientific results and we wonder how they were generated, or we would like to compare the results on one type of data or another, or even simply try a type of analysis or visualization on our own data. Right now, this is nearly impossible. Captions are clearly inadequate for communicating the complexity of picture making even with relatively simple libraries, much less with the powerful data analysis and visualization tools available to scientists these days. Every time an image is cut and pasted from an advanced visualization tool to a paper, most lineage information is lost, and valuable information disappears. In CrowdLabs, we advocate a direct linkage to the provenance information, and we provide mechanisms where any visualization generated on the system can be directly embedded in a document. For instance, Figure 5.3 looks like a normal figure, but it is embedded using a custom LaTex command and by clicking on it, the reader will be taken to the original workflow that creates that image.

For making this possible, CrowdLabs uses a technique analogous to the one used to extend the wiki. We defined a *\vistrail* command that takes in the information necessary to identify and execute the workflow and include the images produced in-place. A LaTeX macro file parses the information inside the command, sends to a python script that builds the http request and sends to CrowdLabs. The images are then downloaded and included as a hyperlinked regular *\includegraphics* command. We also allow users to specify layout options (as they would normally do for the *\includegraphics* command) inside the *\vistrail* command, so these options are passed directly to the generated *\includegraphics*.

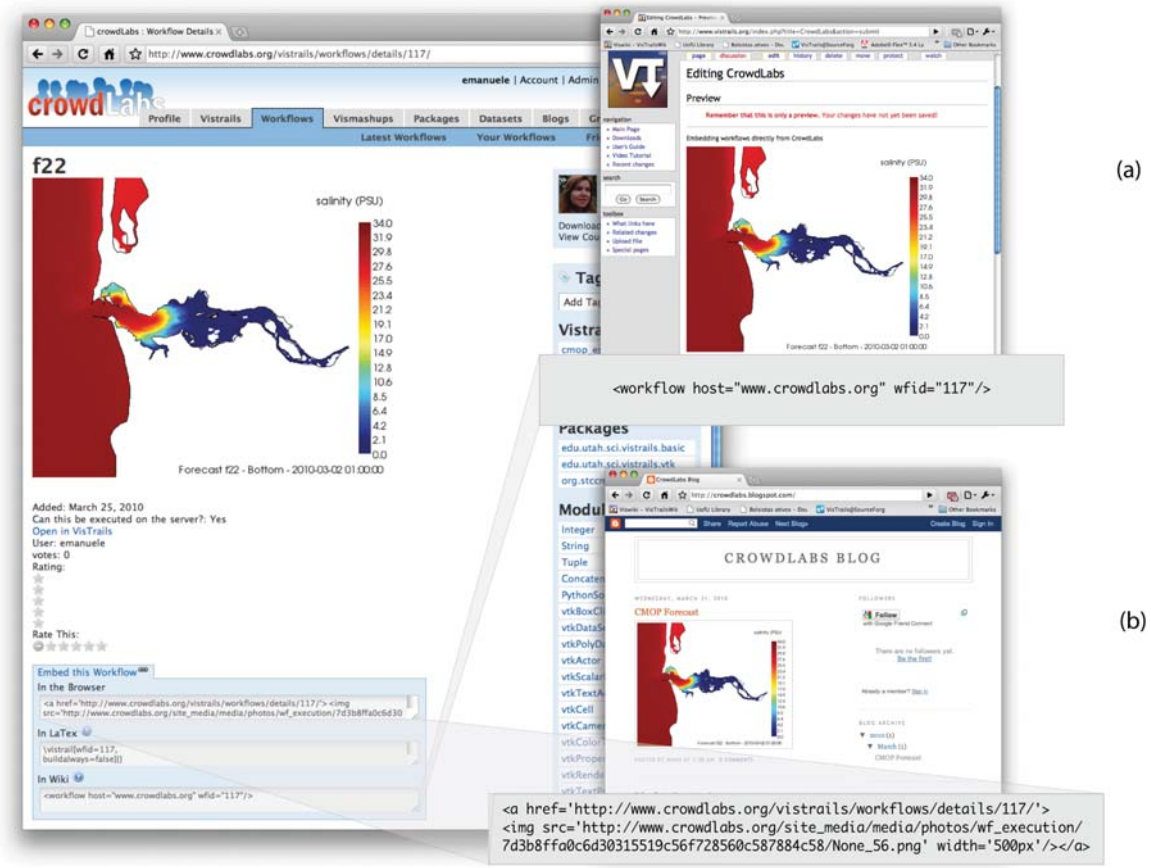

Figure 5.2. Sharing visualizations on the Web: Embedding a workflow as a hyperlinked image in a blog (a) and in a Wiki (b).

5.5 Sharing Content from Other Tools

The techniques presented here are easily extensible to any other system that supports provenance and that is capable of producing result images or files. In particular, it would be easy to share in CrowdLabs visualizations produced in ParaView enabled with the VisTrails Provenance Explorer plugin [9].

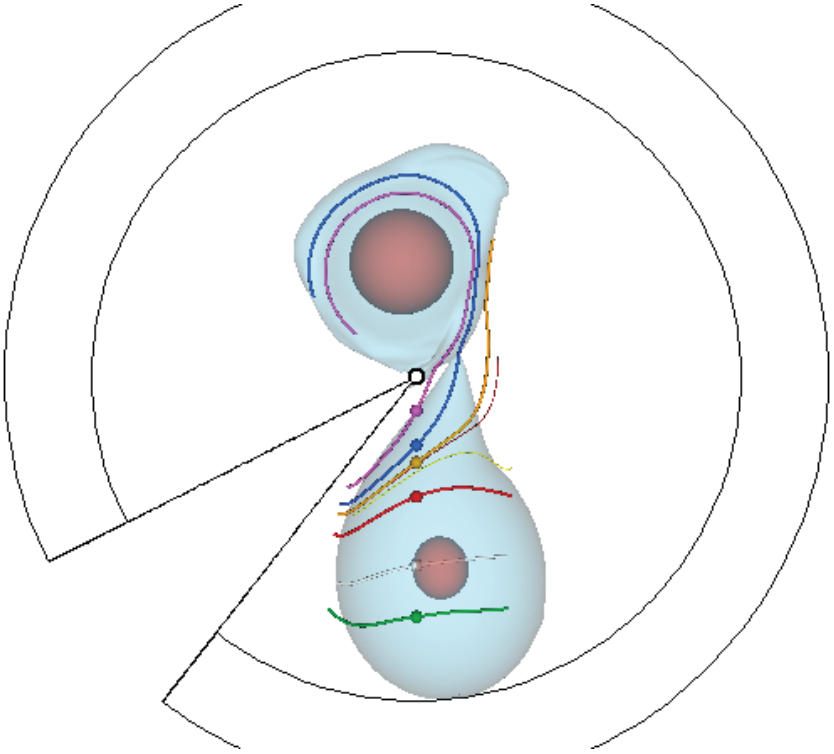

Figure 5.3. Visualizing a binary star system simulation. This is an image that was generated by embedding a workflow directly in the text. The original workflow is available at http://www.crowdlabs.org/vistrails/workflows/details/119/.

CHAPTER 6

APPLICATION SCENARIOS

In this chapter, we present a few application scenarios in which we applied the framework presented in this dissertation: enabling the publication and user interaction with astrophysics simulation results; a mashup that allows neuroscientists to explore and compare complex visualizations for a study on the effects of repetitive Trans-cranial Magnetic Stimulation (rTMS) on working memory; how this framework has been used by scientists in the context of an ocean observatory and finally; how it can be applied by bird observers in the field.

6.1 Sharing Astrophysics Analyses

Professor Joel Tohline's group at Louisiana State University (LSU) has been developing computational fluid dynamics (CFD) techniques to model various astrophysical phenomena, including simulation of mass-transfer instabilities in binary star systems [20]. They use scientific visualization tools and techniques to help them explore the results of these complex CFD simulations. They use VisTrails to construct complex pipelines that involve VTK modules to generate streamlines within each binary mass-transfer simulation, and custom analysis modules for identifying values for important parameters associated with the flows [74, 72].

A binary star system consists of two stars orbiting around a common center of mass with an orbital period P. When the system is viewed from a frame that is rotating with an angular frequency $\Omega_{frame} = 2\pi/P$, the system will appear to be stationary. During a simulation, as mass and angular momentum are transferred from one star to the other, the binary system's orbital period together with the value of Ω_{frame} is expected to vary. Using the most recent pipelines they developed [74], the scientists are able to examine the properties of binary mass-transfer flows from reference frames having a range of different angular frequencies $\Omega_{frame} = (\Omega_0 + \Delta\Omega)$, where Ω_0 is the frame of reference's angular frequency at the beginning of the simulation. As a result, they are able to determine which value of $\Delta\Omega$ provides the best measure of the true orbital period of the binary star system.

The LSU group has published these pipelines on their Web site [74], where they encourage people to try the different values of Ω_{frame} and examine the behavior of the streamlines. Together with the pipelines, they provide annotations where they describe the relevant parameters and values. But to try these pipelines, users must download and install VisTrails, and also learn to use the system. In contrast, using VisMashup, a simple mashup can be created based on the LSU pipelines and deployed as a Web application. Users can then conveniently try the different Ω_{frame} values using a simple form-based interface in a Web browser.

The first mashup we developed was based on the pipeline shown in Figure 6.1. A limitation of this mashup is that, since it contains a single 3D view, it does not easily support the comparison of different parameters. To address this problem, we added more views of this same pipeline to the medley and synchronized the parameters ρ_{min} and *propagation time* in all of them (see Figure 6.2(a)). The new mashup is more suitable for the exploration of the Ω_{frame} parameter, allowing multiple visualizations to be displayed and compared side by side. The app, shown in Figure 6.2(b), has three 3D views with a set of widgets for manipulating the parameters of each view. Users can then set a different value for Ω_{frame} in each window and compare the streamlines. This comparison is facilitated by the synchronization of the 3D view cameras.

6.1.1 An Interactive Astrophysics Book

As we described in Section 2.3, as a venue for reporting computational science results, the printed medium is less than optimal. For most complex computations, it is nearly impossible to fully describe them as printed words. As briefly discussed above, even publishing source

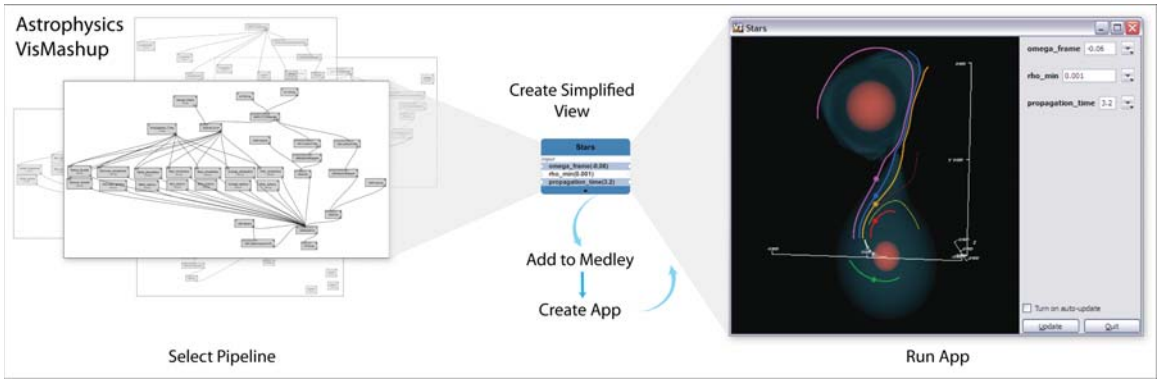

Figure 6.1. Creating an Astrophysics VisMashup to explore a binary star system.

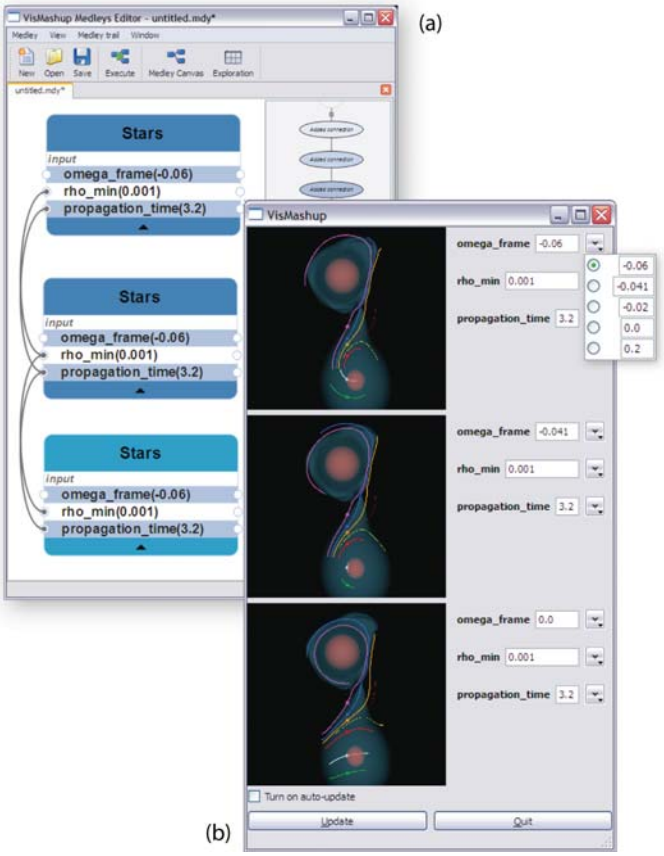

Figure 6.2. Astrophysics VisMashup for comparing different visualizations. (a) Medley created for the VisMashup using 3 instances of a pipeline view with parameters ρ_{min} and *propagation time* synchronized. (b) Using the VisMashup interface, the user explores different values for Ω_{frame}. From top to bottom, the values used are -0.06, -0.041 and 0.0.

code and data is not really enough, since paper does not provide an interactive medium. Many scientists have been advocating for the development of new media that would utilize computer technology to provide a richer experience.

Professor Tohline also believes that the printed journal format, already in use for hundreds of years, is not adequate for readers to fully verify the algorithms and analyze the results published in scientific articles, and that, in the simplest case, a scientific article should make use of the dynamic presentation tools currently available on the Web [73].

Recently, in collaboration with CrowdLabs's project members, Tohline has been undergoing an effort to design a more suitable medium for publication in the computational sciences [75]. He recently resumed working on a project that he first tried 15 years ago: the creation of an interactive astrophysics book. One way to think of this effort is like a

Wikipedia++. Here, all the computational results are modeled by workflows that link back to all the underlying data and code. All the visualizations would also have corresponding descriptions, and can be run interactively on a user's computer or over the Web. One of the hopes is that with such a medium, one can keep the science "current" by updating this single master source. Because the medium tracks changes, it will be possible to keep proper ownership of ideas, and to give proper credit to contributors.

6.2 Visualization in Neuroscience

The study of cognition and memory in humans has been an area of intense study in recent years. Current literature shows that a wide variety of psychological disorders have a marked decrease in working memory performance and that a primary indicator of working memory is the spectral properties of the alpha band (8-12 Hz) in the dorsal lateral prefrontal cortex (DLPFC) [69]. One method under development to help improve working memory performance is repetitive Trans-cranial Magnetic Stimulation (rTMS). rTMS has been shown to be capable of positively manipulating the underlying spectral dynamics of human alpha rhythms to influence working memory performance [2].

In order to adequately measure the effects of rTMS on individual alpha-band spectral dynamics and explore signal patterns in the desired region of the brain, Electroencephalography (EEG) and Magneto-encephalography (MEG) are employed in conjunction with Magnetic Resonance Imaging (MRI). Commonly in collaborative studies, the data processing and visualization experts are unable to adequately analyze the results and implications of the analysis pipelines they create. Likewise, the scientists who can analyze the spectral properties of cognitive dynamics do not have the experience required to write the robust pipelines needed in the study.

VisMashup supports collaboration among the different experts. Experts in data processing and visualization generate a series of pipeline views which expose *only* the functionality the scientists need. The scientists analyzing the results can then explore these views without having to deal with complexities of the corresponding pipelines.

Figure 6.3 shows a mashup constructed from a medley that consists of two pipelines: one that generates a plot of an EEG and another that creates a volume rendered image of a patient's brain. By using vismashups like this, the psychiatrist conducting this study was able to explore the patients' data more freely and more extensively without having to learn how to use the underlying visualization system. The visualization experts working on this study also benefited from using VisMashup because it allowed them to generate the final

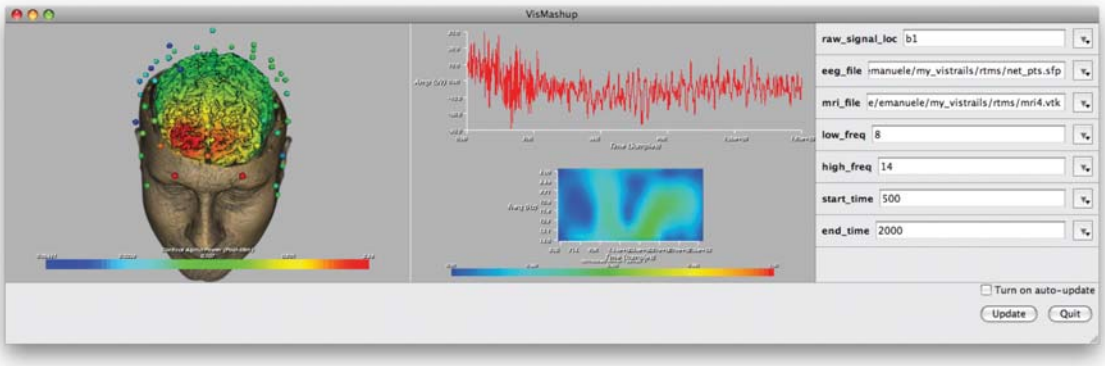

Figure 6.3. Neuroscience VisMashup. This mashup combines two pipelines—one that produces a plot for an EEG and another that creates a volume rendered visualization. To ensure that the plot and the visualization are derived for the same patient, the variables in the pipeline views corresponding to the patient (*i.e.,*the input data set) are synchronized.

apps more quickly, requiring less time spent training the scientist to manipulate complex pipelines.

Before having access to VisMashup, the expert responsible for creating the multimodal visualizations needed in this study had designed had a single pipeline consisting of several disjoint subpipelines. Each of these subpipelines handled a single data modality or processing technique. But for medleys to be effective, it is important that the pipelines be modularized. Since medleys are composed by distinct pipeline views, a medley consisting of composite pipelines is inherently difficult to manipulate. By enforcing stricter modularization on the development of pipelines, medley-enabled systems can benefit not only from more concise pipeline descriptions, but from more flexible display of the results.

6.3 Supporting a Dynamic Ocean Observatory Web Portal

The NSF Science and Technology Center for Coastal Margin Observation & Prediction (CMOP) [17] aims to facilitate interdisciplinary research, technology development, education and knowledge transfer in order to achieve a better understanding of physical, chemical and biological processes regulating river-to-ocean ecosystems. The project has amassed a data inventory that includes millions of data products and sensor measurements, and tens of thousands of results from numerical models. These cover more than 10 years of experiments and occupy over 100 TB of storage. In addition, they have accumulated numerous computational processes used to run simulations, analyze the data and generate

visualizations. CMOP also maintains the Science and Technology University Research Network (SATURN) observatory: a large network of heterogeneous observation platforms coupled with large-scale simulation models of ocean circulation.

Powerful and integrative modeling and visualization systems are at the core of CMOP, 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. To derive and calibrate their models, CMOP scientists need to assemble complex workflows which execute a series of simulations (*e.g.,* running their own code); gather and process data retrieved from sensors; and then create visualizations (*e.g.,* using the VTK library [38]) to compare the results of the models against the sensor data. Because these tasks require expertise in different areas, several people need to be involved in the creation of a new visualization. In CMOP, modelers run the simulations while a visualization expert creates the necessary visualizations. After the results are analyzed and the scientists agree on how to calibrate the model, the process must be repeated. Significant effort is required to build new workflows and to manually modify existing workflows to cater to new requirements (*e.g.,* to use different simulation parameters or different visualization techniques).

The complexity of managing the data, analyses and visualizations in a large center like CMOP has been a key driver of the development of CrowdLabs. While previously, it was extremely hard to design, develop and deploy new visualization data products inside CMOP, CrowdLabs has been greatly facilitating the interaction of the scientists in the center. An interesting example is to consider the process to updating forecasts and hindcast, and adding new plots.

As soon as a simulation run of a forecast or a hindcast is completed and processed, the analysis and visualization results need to be made available on the *Virtual Columbia River* Web site[1] (see Figure 6.4), which is designed in such way that it directly accesses the images generated in these runs. With new data comes the necessity to generate improved visualizations. As we say above, CMOP's production environment is very complicated: it requires the cyber-team to modify dozens of scripts that are executed for each run and to make the Web site aware of the new data product. On-the-fly generation of visualization data products is often avoided, since creating new analyses and publishing their results is time-consuming, often requiring programming expertise and a trial-and-error cycle, demanding intense and off-hours interactions of IT staff with scientists. Essentially all the existing CMOP visualizations are *canned*, and do not support any form of interaction.

[1] http://www.stccmop.org/datamart/virtualcolumbiariver

Figure 6.4. CMOP's Virtual Columbia River Web site: users can access analyses and visualizations of daily forecasts and retrospective simulations.

VisMashup and CrowdLabs are helping to solve that. With CrowdLabs, it is not only easier to manage the overall process, but the system also streamlines the interaction between scientists and visualization experts. The fact that visualizations can be made available immediately without the need for the IT staff to be directly involved in performing manual configurations facilitates deployment. Also, interactive visualizations (in the form of VisMashups) are quickly generated. Figure 6.5 shows an example of how a mashup for a climatology of different properties collected from the Columbia River was created. These mashups allow for parameters to be changed, so that scientists can generate new visualizations on-the-fly and be able to inspect each image individually. These are extremely

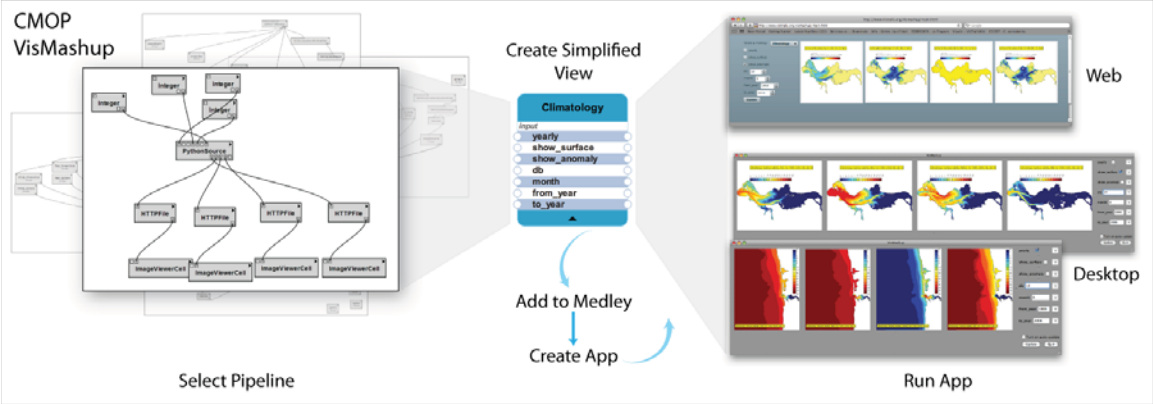

Figure 6.5. Creating a vismashup for a CMOP climatology.

useful to detect and fix errors in the simulations, and to explore the data. The ability to maintain a permanent record of conversations on the design of the visualizations helps everyone stay in sync with decisions. One of these records as a blog post is shown in Figure 6.6.

6.3.1 Comparing real time data with simulation data

Some of these data products need to be accessed in the field. Research cruises are an integral part of SATURN. These cruises are used to measure hydrographic (temperature, salinity and pressure), bio-optical and chemical parameters. They can also collect water samples for microbial and bacterial analyses. Often the scientists on a cruise compare the collected data with the data generated by the simulation models for quality control. This is usually done as a postprocess phase with report generation. However, sometimes a faulty sensor on the ship collects invalid data that will only be detected when it is too late to take other readings in the same region. Being able to quickly compare the acquired data with the one generated by the forecast at the time the data would help to detect and fix the faulty sensors.

The scientists easily created a workflow that for a given time, location, depth value and parameter type, it would return the value generated by the forecast. This was because they reused another workflow that generated visualizations using the forecast data and adapted to obtain the data for a given location, which they accomplished by using the mobile package. When asked to be executed from a portable device, the workflow would get the location automatically and the user can enter the other parameters. The workflow will query the data from the simulation model and return the value back to the user. Figure 6.7(a) shows

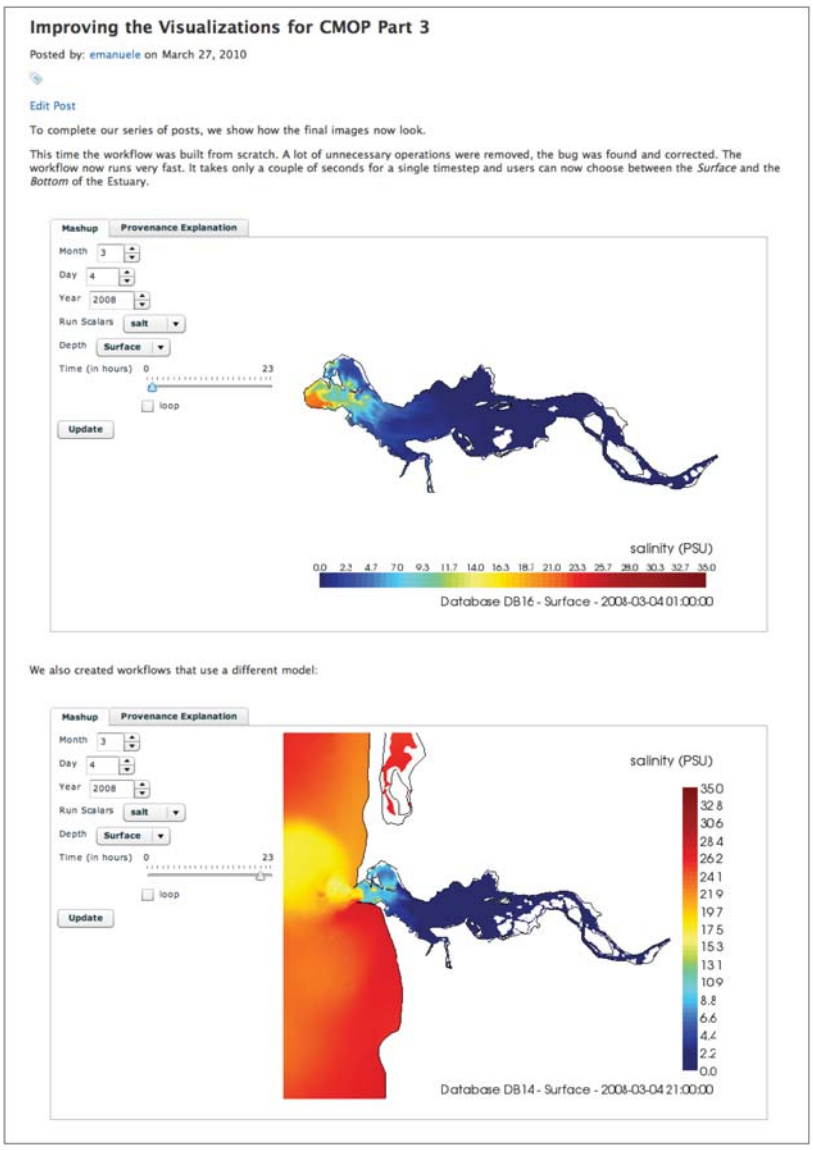

Figure 6.6. Using the blog to document processes: A visualization expert created a series of blog posts to explain the problems found when generating the visualizations for CMOP.

the modified workflow. Figure 6.7(b) displays the mashup being displayed on an iPhone and Figure 6.7(c) shows the results. Notice that the quite complex workflow is executed on the CMOP server and the results are sent back to the device.

6.4 Observing Birds

The birding community is a very large and active group. Nowadays, there are many Web portals (ebird.org, birdforum.net, birdingcommunity.com) dedicated to help birders to share and discuss their observations. Most of these portals require an account in order to

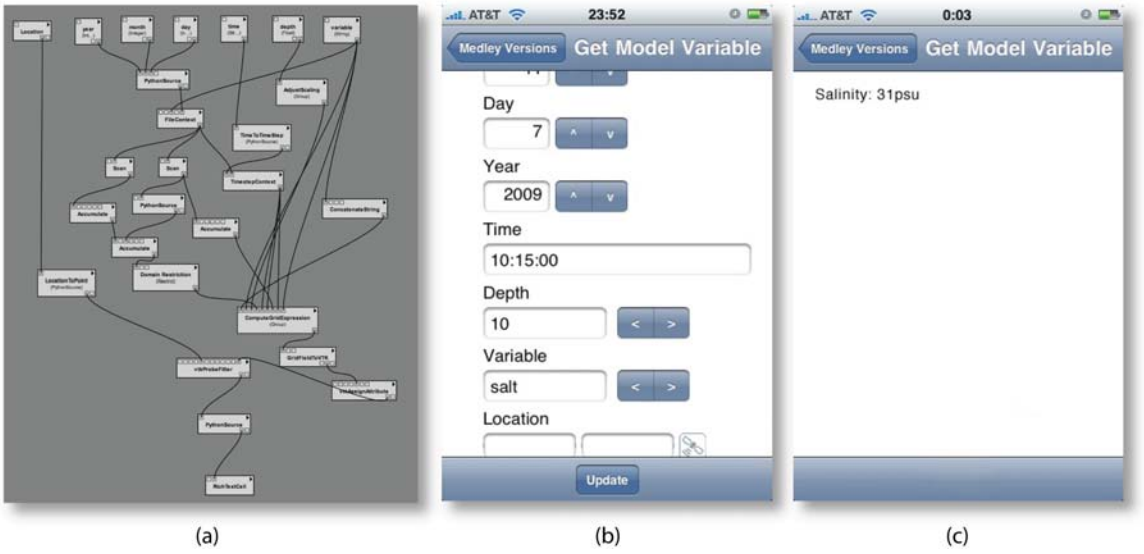

Figure 6.7. Example of a mashup for displaying the value of a hidrographic parameter running on the iPhone. (a) The workflow created on the VisTrails system using the modules from different libraries. (b) The main window of the Mashup. (c) Results of running the mashup.

have access to their services. After logging in, users can enter when, where and how they went birding, find birds spotted in their region and help other users identify unknown birds.

The eBird project (ebird.org), which was introduced by the Cornell Lab of Ornithology and National Audubon Society, is a large-scale citizen science project[2]. Citizen science has been shown to be particularly useful for gathering scientific information from diverse locations. Some projects have been able to mobilize large numbers of volunteers to collect immense amounts of data from many different locations spanning long periods of time. Data stored on eBird is shared among a global community of educators, land managers, ornithologists and conservation biologists in order to obtain a better understanding of bird distribution across the western hemisphere and beyond. Therefore, data quality is a critical issue for the data to be useful [7].

One of the most common sources of errors in the data collected by birders is location data reported manually. Since many portable devices are currently equipped with GPS, being able to record a observation in the field would alleviate this problem. Recently, BirdsEye (www.getbirdseye.com) was released as a new birding app for the iPhone that provides

[2]The Wikipedia definition of citizen science is "a term used for projects or ongoing program of scientific work in which individual volunteers or networks of volunteers, many of whom may have no specific scientific training, perform or manage research-related tasks such as observation, measurement or computation."

real-time access to bird observations submitted to eBird. Although it has many features, such as being able to find birds nearby and specific birds, it does not allow birders to send observation data back to eBird, storing all observations locally.

To demonstrate how easily this can be done without developing a dedicated application, we developed a workflow for collecting data from bird observations that uses the location provided by the GPS on the portable device. The workflow can also collect an image from the library of images on the device or taking a new picture using the device's camera and record a voice note. This useful feature can also be used to record the birds' sounds. The data collected by the workflow is sent to a database on the Internet where the data verification process can be run. Figure 6.8(a) shows the workflow created in VisTrails using the basic and mobile packages. Figure 6.8(b) displays the parameters that were exposed on the mashup: username, species, location, image, audio and notes. The user can select an image from one of his photo albums, shown in Figure 6.8(c), or take a new picture. A voice note or the bird sound can be recorded using the widget displayed on Figure 6.8(d).

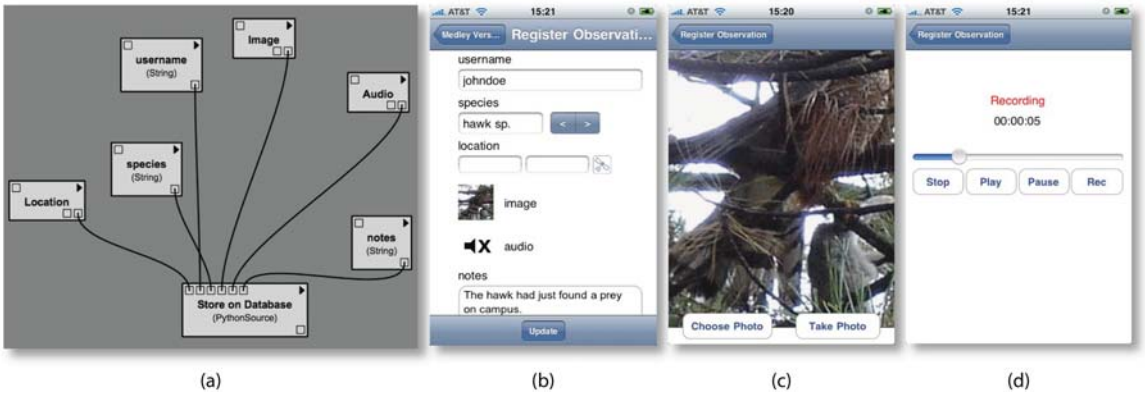

Figure 6.8. Example of a mashup for registering a bird observation running on the iPhone. (a) The workflow created on the VisTrails system using the modules from the Basic and Mobile packages. (b) The main window of the Mashup. (c) Widget for image selection. (d) Widget for recording audio.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This dissertation has presented a new framework for simplifying the creation and deployment of collaborative data analysis and visualization tools. It also presented a series of application scenarios that illustrate the benefits of the framework and how it can be used in different contexts.

Chapter 3 described VisMashup, a system that simplifies the creation of custom visualization applications. Using VisMashup, an application developer can quickly assemble custom applications by leveraging an existing collection of visualization pipelines and their provenance. The mashups produced by VisMashup are not a substitute for more comprehensive, domain-specific applications such as CDAT. Instead, our targets are simpler, exploratory applications.

Although VisMashup simplifies the development of one-of-a-kind, domain-specific visualization applications, it has some limitations. The integration of different libraries can sometimes be complicated by a number of practical issues, such as compatibility of the underlying interface toolkit (e.g., Tk vs Qt). Also, while we try to make sensible choices on the automatic layout of the interface, and we allow the users to fine tune it, this does not guarantee that the most appropriate and intuitive interface for the task at hand is created. There are several areas of future work that we would like to pursue. First and foremost, we need to perform a more detailed evaluation of the effectiveness and usability of VisMashup and CrowdLabs. To do so, we will carry out user studies. To provide more flexibility in filtering and selecting relevant pipelines, we plan to integrate more sophisticated interfaces that allow structural queries to be specified by example [62, 5]. We also plan to explore alternative techniques for mining the pipeline collection. For example, it can be useful to support different grouping/clustering strategies [59]. We also would like to investigate environments where users can share and collaboratively develop applications. Such environments have been successfully deployed for Web mashups (see *e.g.,* http://www.programmableweb.com). But we believe that the ability to share visualization

applications, in addition to data, will help advance existing efforts on collaborative data analysis [78] and it also has the potential to broaden the use of visualization techniques.

The CrowdLabs system presented in Chapter 4 builds on top of VisTrails and VisMashup and it provides the "last mile" to the scientists. For its most basic use, it does not require any installation of tools in the scientists machine, and we see this as a huge advantage. The barrier of entry is quite small, and it is possible to do a good amount of data analysis and visualization tasks without ever having to install any tools. Besides being deployed on `www.crowdlabs.org` and on the VisTrails Wiki (`www.vistrails.org`), the system is also being deployed at the CMOP site [17] at the Oregon Health & Science University, and the ALPS Project [1] site (`http://alps.comp-phys.org`) at ETH in Zurich. We believe that many small research groups that do not have all the resources and infrastructure they would need for doing data analysis and visualization tasks for their research can possibly benefit from CrowdLabs and VisMashup.

CrowdLabs is an evolving system, and it will evolve substantially as we add more users. Besides VisTrails workflows, we also plan to support other provenance-enabled tools, such as ParaView [37], VisIt [79], Maya [46] and other workflow systems. In order to accommodate a growing community, we expect that will be adding substantial new functionality:

7.1 Provenance Analytics

The problem of mining and extracting knowledge from provenance data has been largely unexplored. By analyzing provenance data, scientists can debug their tasks and obtain a better understanding of their results. Mining this data may also lead to the discovery of patterns that can potentially simplify the notoriously hard, time-consuming process of designing and refining scientific workflows. We would also like to use these techniques to extend the system with the following capabilities: a workflow search engine, recommendations based on the types of file the user wants to visualize, the ability to connect people based on the types of workflows they are interested in, etc.

7.2 Improved Web-Enabled Interfaces and Graphics

Although it is possible to use CrowdLabs completely from a web browser, some advanced functionality is only available by installing the VisTrails application. We would like to develop novel interfaces that would allow us to eliminate this need. One of the big challenges is that Web 3D graphics is not well developed or standardized at this moment, creating a major hurdle to support high-end visualization over the web. Due to some data being

remote, we believe that we will need to also add on streaming and multiresolution techniques to our data analysis and visualization workflows.

7.3 System Improvements

There are a number of overall system improvements that are needed, including improved scalability in terms of the size of provenance information and data; improvements to our security model, as to allow for fine grain security; better workflow synchronization capabilities, and so on.

In Chapter 5, we described our approach for reproducible documents. In Chapter 6, various application scenarios that illustrate the benefits of the framework are presented. In the first scenario, we showed how to enable the publication and user interaction with astrophysics simulation results. The second scenario allowed neuroscientists to use VisMashup to explore and compare complex visualizations for a study on the effects of repetitive Trans-cranial Magnetic Stimulation (rTMS) on working memory. The third scenario illustrated the use of the complete framework in the context of an ocean observatory. Finally, in the fourth scenario, the framework was applied to bird observers working in the field.

7.4 Software

The software components generated from this work, although they are robust to an acceptable degree, they are not ready for a general public release yet. We plan to have a public release on `www.crowdlabs.org` in the next few months.

REFERENCES

[1] A.F. Albuquerque, F. Alet, P. Corboz, P. Dayal, A. Feiguin, S. Fuchs, L. Gamper, E. Gull, S. Gürtler, A. Honecker, R. Igarashi, M. Körner, M. Kozhevnikov, A. Läuchli, S.R. Manmana, M. Matsumoto, I.P. McCulloch, F. Michel, R.M. Noack, G. Pawlowski, L. Pollet, T. Pruschke, U. Schollwöck, S. Todo, S. Trebst, M. Troyer, P. Werner, and S. Wessel. The ALPS project release 1.3: Open Source Software for Strongly Correlated Systems. *J. Mag. Mag. Mat.*, 310:1187, 2007.

[2] Erik Anderson, Gilbert Preston, and Claudio Silva. Towards Development of a Circuit Based Treatment for Impaired Memory: A Multidisciplinary Approach. *IEEE EMBS Conf. on Neural Engineering*, pages 302–305, 2007.

[3] Luciano Barbosa and Juliana Freire. An Adaptive Crawler for Locating Hidden-Web Entry Points. In *Proceedings of WWW*, pages 441–450, 2007.

[4] 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.

[5] Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying Business Processes. In *Proceedings of VLDB*, pages 343–354, 2006.

[6] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, RI, Revised edition, 1948.

[7] Rick Bonney, Caren Cooper, Janis Dickinson, Steve Kelling, Tina Phillips, Kenneth Rosenberg, and Jennifer Shirk. Citizen Science: A Developing Tool for Expanding Science Knowledge and Scientific Literacy. *BioScience*, 59(11):977–984, Dec 2009.

[8] J. Buckheit and D. L. Donoho. WaveLab and Reproducible Research. In A. Antoniadis, editor, *Wavelets and Statistics*, pages 55—81. 1995.

[9] Steven P. Callahan, Juliana Freire, Carlos E. Scheidegger, Cláudio T. Silva, and Huy T. Vo. *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised Selected Papers*, chapter Towards Provenance-Enabling ParaView, pages 120–127. LCNS 5272. Springer-Verlag, 2008.

[10] John M. Carroll and Caroline Carrithers. Training Wheels in a User Interface. *Communications of the ACM*, 27(8):800–806, 1984.

[11] Climate Data Analysis Tools (CDAT). `http://www-pcmdi.llnl.gov/software-portal/cdat`.

[12] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[13] Bryan Chan, Leslie Wu, Justin Talbot, Mike Cammarano, and Pat Hanrahan. Vispedia: Interactive Visual Exploration of Wikipedia Data via Search-Based Integration. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1213–1220, Nov.-Dec. 2008.

[14] Chemical blogspace. `http://cb.openmolecules.net/`.

[15] Hank Childs, Eric S. Brugger, Kathleen S. Bonnell, Jeremy S Meredith, Mark Miller, Brad J Whitlock, and Nelson Max. A Contract-Based System for Large Data Visualization. In *IEEE Visualization*, pages 190–198, 2005.

[16] Jon Claerbout. Seventeen years of super computing and other problems in seismology. `http://sepwww.stanford.edu/sep/jon/nrc.html` (accessed May 27, 2010), 1994.

[17] NSF Center for Coastal Margin Observation and Prediction (CMOP). `http://www.stccmop.org`.

[18] Sustainable digital data preservation and access network partners (DataNet). http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503141. Accessed on 19 February 2009.

[19] Susan B. Davidson and Juliana Freire. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of SIGMOD*, pages 1345–1350, 2008.

[20] M.C.R. D'Souza, P.M. Motl, J.E. Tohline, and J. Frank. Numerical Simulations of the Onset and Stability of Dynamical Mass transfer in Binaries. *The Astrophysical Journal*, 643(1):381–401, 2006.

[21] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[22] S. Fomel and J.F. Claerbout. Guest Editors' Introduction: Reproducible Research. *Computing in Science Engineering*, 11(1):5 –7, jan.-feb. 2009.

[23] S. Fomel and G. Hennenfent. Reproducible Computational Experiments Using SCons. In *Proceedings of IEEE Int'l Conf. Acoustics, Speech and Signal Processing*, volume 4, pages 1257–1260. IEEE CS Press, 2007.

[24] D. Foulser. IRIS Explorer: A Framework for Investigation. *ACM SIGGRAPH Computer Graphics*, 29(2):13–16, 1995.

[25] Michael J. Franklin, Alon Y. Halevy, and David Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4):27–33, 2005.

[26] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T. Silva. Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, 10(3):11–21, May-June 2008.

[27] Juliana Freire, Cláudio Silva, Steve Callahan, Emanuele Santos, Carlos Scheidegger, and Huy Vo. Managing Rapidly-Evolving Scientific Workflows. In *International Provenance and Annotation Workshop (IPAW)*, LNCS 4145, pages 10–18. Springer Verlag, 2006.

[28] Robert Gentleman. Reproducible Research: A Bioinformatics Case Study. *Statistical Applications in Genetics and Molecular Biology*, 4(1), 2005. Article 2.

[29] Robert Gentleman and Duncan Temple Lang. Statistical Analyses and Reproducible Research. Bioconductor Project Working Papers, 2004. Working Paper 2.

[30] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, 286(5439):531–537, 1999.

[31] Timothy Gowers and Michael Nielsen. Massively Collaborative Mathematics. *Nature*, 461(7266):879–881, 2009.

[32] Paul E. Haeberli. ConMan: A Visual Programming Language for Interactive Graphics. In *Proceedings of SIGGRAPH'88*, pages 103–111, 1988.

[33] J. Hastad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[34] Robert Hoffmann. A Wiki for the Life Sciences Where Authorship Matters. *Nature Genetics*, 40(9):1047–1051, 2008.

[35] IBM. OpenDX. `http://www.research.ibm.com/dx`.

[36] C.R. Johnson, R.S. MacLeod, S.G. Parker, and D.M. Weinstein. Biomedical Computing and Visualization Software Environments. *Communications of the ACM*, 47(11):64–71, 2004.

[37] Kitware. Paraview. `http://www.paraview.org`.

[38] Kitware. The Visualization Toolkit. `http://www.vtk.org`.

[39] Donald E. Knuth. Literate Programming. *The Computer Journal (British Computer Society*, 27(2):97—111, 1984.

[40] David Koop, Carlos Scheidegger, Steven Callahan, Juliana Freire, and Cláudio Silva. VisComplete: Data-driven Suggestions for Visualization Systems. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1691–1698, 2008.

[41] Edward A. Lee and Thomas M. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[42] F Leisch. Sweave: Dynamic Generation of Statistical Reports using Literate Data Analysis. In W. Härdle and B. Rönz, editors, *Proceedings of Computational Statistics Compstat 2002*, pages 575—580. Physika Verlag, 2002.

[43] Kwan-Liu Ma. Image Graphs - A Novel Interface for Visual Data Exploration. In *Proc. of IEEE Visualization*, pages 81–88, 1999.

[44] R.S. Macleod, D.M. Weinstein, J.D. de St. Germain, D.H. Brooks, C.R. Johnson, and S.G. Parker. SCIRun/BioPSE: Integrated Problem Solving Environment for Bioelectric Field Problems and Visualization. In *Proceedings of the Int. Symp. on Biomed. Imag.*, pages 640–643, 2004.

[45] Many Eyes Wikified. `http://wikified.researchlabs.ibm.com`.

[46] Maya. `http://www.autodesk.com/maya`.

[47] Matt McKeon. Harnessing the Web Information Ecosystem with Wiki-based Visualization Dashboards. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1081–1088, 2009.

[48] M McLennan and R Kennell. HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering. *Computing in Science & Engineering*, 12(2):48 – 53, 2010.

[49] Mercury Computer Systems. Amira. `http://www.amiravis.com`.

[50] J Mesirov. Accessible Reproducible Research. *Science*, Jan 2010.

[51] Tamara Munzner, Chris Johnson, Robert Moorhead, Hanspeter Pfister, Penny Rheingans, and Terry S. Yoo. NIH-NSF Visualization Research Challenges Report Summary. *IEEE Computer Graphics and Applications*, 26(2):20–24, 2006.

[52] Steven G. Parker and Christopher R. Johnson. SCIRun: a Scientific Programming Environment for Computational Steering. In *Proc. of Supercomputing*, 1995.

[53] A. R. Pico, T. Kelder, M. P. van Iersel, K. Hanspers, B. R. Conklin, and C. Evelo. WikiPathways: Pathway Editing for the People. *PLoS Biology*, 6(7), 2008.

[54] David De Roure, Carole Goble, and Robert Stevens. The Design and Realisation of the Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems*, 25(5):561 – 567, 2009.

[55] E. Santos and H. Vo. Reproducible and interactive documents. Online video clip, September 2008. Accessed on 31 October 2008. "http://www.sci.utah.edu/ emanuele/movies/vistrails-publishing-new.mov".

[56] Emanuele Santos, Juliana Freire, and Claudio Silva. Information Sharing in Science 2.0: Challenges and Opportunities. CHI Workshop on The Changing Face of Digital Science: New Practices in Scientific Collaborations, 2009.

[57] Emanuele Santos, David Koop, Huy T. Vo, Erik W. Anderson, Juliana Freire, and Cláudio T. Silva. Using Workflow Medleys to Streamline Exploratory Tasks. In *Proceedings of SSDBM 2009*, pages 292–301, 2009.

[58] Emanuele Santos, Lauro Lins, James Ahrens, Juliana Freire, and Cláudio Silva. VisMashup: Streamlining the Creation of Custom Visualization Applications. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1539–1546, 2009.

[59] Emanuele Santos, Lauro Lins, James P. Ahrens, Juliana Freire, and Cláudio T. Silva. A First Study on Clustering Collections of Workflow Graphs. In *International Provenance and Annotation Workshop (IPAW)*, pages 160–173, 2008.

[60] Emanuele Santos, Phillip Mates, Erik Anderson, Bradley Grimm, Juliana Freire, and Claudio Silva. Towards Supporting Collaborative Data Analysis and Visualization in a Coastal Margin Observatory. CSCW 2010 Workshop on The Changing Dynamics of Scientific Collaboration, 2010.

[61] Emanuele Santos, Phillip Mates, Juliana Freire, and Cláudio Silva. crowdLabs: Social Analysis and Visualization for the Sciences, 2010. (Submitted).

[62] Carlos Scheidegger, David Koop, Huy Vo, Juliana Freire, and Cláudio Silva. Querying and Creating Visualizations by Analogy. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1560–1567, 2007.

[63] Matthias Schwab, Martin Karrenbach, and Jon Claerbout. Making Scientific Computations Reproducible. *Computing in Science & Engineering*, 2(6):61—67, 2000.

[64] B Shneiderman. Science 2.0. *Science*, 319(5868):1349–1350, Jan 2008.

[65] Ben Shneiderman. Promoting Universal Usability with Multi-layer Interface Design. In *Proceedings of the Conf. on Universal Usability*, pages 1–8, 2003.

[66] A Strachan, G Klimeck, and M Lundstrom. Cyber-Enabled Simulations in Nanoscale Science and Engineering. *Computing in Science & Engineering*, 12(2), March/April 2010.

[67] Swivel. `http://www.swivel.com`.

[68] Tableau Public. `http://www.tableausoftware.com/public/`.

[69] H.Y. Tan, W.C. Choo, S.L. Fones, and W.L. Chee. Neuropsychological Performance in Schizotypal Personality Disorder: Importance of Working Memory. *The American Journal of Psychiatry*, 162(10):1896–1903, 2005.

[70] A. Telea and J.J. van Wijk. SMARTLINK: An Agent for Supporting Dataflow Application Construction. In *Proceedings of IEEE VisSym 2000*, pages 189–198, 2000.

[71] R. Telea and J.J. Van Wijk. VISSION: An Object Oriented Dataflow System for Simulation and Visualization. *Proceedings of IEEE VisSym 1999*, pages 95–104, 1999.

[72] J. Tohline, Jinghya Ge, W. Even, and E. Anderson. A Customized Python Module for CFD Flow Analysis within VisTrails. *Computing in Science & Engineering*, 11(3):68–73, 2009.

[73] J. E. Tohline. Scientific Visualization: A Necessary Chore. *Computing in Science & Engineering*, 9(6):76–81, 2007.

[74] Joel Tohline. LSU using VisTrails. `http://www.phys.lsu.edu/~tohline/vistrails/`.

[75] Joel Tohline and Emanuele Santos. Visualizing a Journal that Serves the Computational Sciences Community. *Computing in Science & Engineering*, 12(3), 2010. To appear.

[76] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42, 1976.

[77] Craig Upson, Jr. Thomas Faulhaber, David Kamins, David H. Laidlaw, David Schlegel, Jefrey Vroom, Robert Gurwitz, and Andries van Dam. The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[78] Fernanda B. Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. ManyEyes: A Site for Visualization at Internet Scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.

[79] VisIt Visualization Tool. `https://wci.llnl.gov/codes/visit`.

[80] The VisTrails Project. `http://www.vistrails.org`.

[81] M. Mitchell Waldrop. Science 2.0. *Scientific American*, 298(5):p68 – 73, May 2008.

[82] Mitch Waldrop. Wikiomics. *Nature*, 455:22–25, September 2008.

[83] Yahoo! Pipes. `http://pipes.yahoo.com`.

[84] Fan Yang, Nitin Gupta, Chavdar Botev, Elizabeth F Churchill, George Levchenko, and Jayavel Shanmugasundaram. WYSIWYG Development of Data Driven Web Applications. *Proceedings of the VLDB Endowment.*, 1(1):163–175, 2008.