COLLABORATIVE SCIENTIFIC GRAPHICS AND WEB SERVICES

Eliot Feibush

Princeton Plasma Physics Laboratory

Princeton, NJ

Matthew Milano

Brown University, Providence RI

Benjamin Phillips

Cornell University, Ithaca NY

Princeton Plasma Physics Laboratory

Princeton, New Jersey

August 28, 2010

ABSTRACT

The global fusion energy program is distributed across many institutions. Physicists at different locations often work on the same projects and need to collaboratively share and manipulate data. ElVis, an interactive graphics program for exploring scientific data, is often used by these physicists for fusion research. ElVis interfaces with legacy codes and data sources, making it a natural container for visualizing data. Our project was to support efficient, unconstrained collaborative editing of scientific graphs. Previous efforts in unconstrained collaboration focused on Operational Transform (OT), which is ineffective for ElVis. We wished to create a new design which allows an arbitrary number of users to work on the same graphs at the same time, without reducing ElVis's current functionality. Furthermore, we wanted users to be able to modify graphs in real-time without having to wait on the network every time they want to execute an operation. All of these goals needed to be accomplished while leaving the existing code base mostly untouched so as to avoid introducing bugs into ElVis' scientific analysis tools.

With these goals in mind, we adopted a replicated architecture for our collaboration framework whereby each user's ElVis instance maintains its own copy of the collaboration data. Collaborative operations were implemented on top of ElVis' existing architecture thereby allowing network communication and synchronization issues to be resolved before executing any legacy code. This posed a unique challenge, and lead to cases where execution order is ambiguous or undefined. A popular and heavily-researched method for resolving these issues is OT. OT was designed with the assumption that interactions execute and sycn against other interactions quickly, invariants that are untrue in ElVis and others. This led to a new method of collaboration, our implementation of a dynamic floor control model that functions as a hybrid between a traditional floor control model and operational transformation. The model allows fast interactions to be executed using OT and slower interactions to be executed using floor control.

Our dynamic floor control model satisfies our design requirements. It supports real-time collaboration and can be used by an arbitrary number of scientists on an arbitrary number of projects. It is also convenient for users and effective at handling the types of interactions in scientific graphics and visualization applications.

INTRODUCTION

The global fusion energy program is distributed across many institutions. Physicists at different locations often work on the same projects and need to collaboratively share and manipulate data. While existing software packages such as TeamViewer provide support for multi-user collaboration, they cannot take advantage of application specific features and are often slower and less flexible than application specific collaboration implementations. This project was to implement collaborative features for ElVis, an existing interactive graphics

program for exploring scientific data (Figure 1) that is currently used by physicists for fusion research.

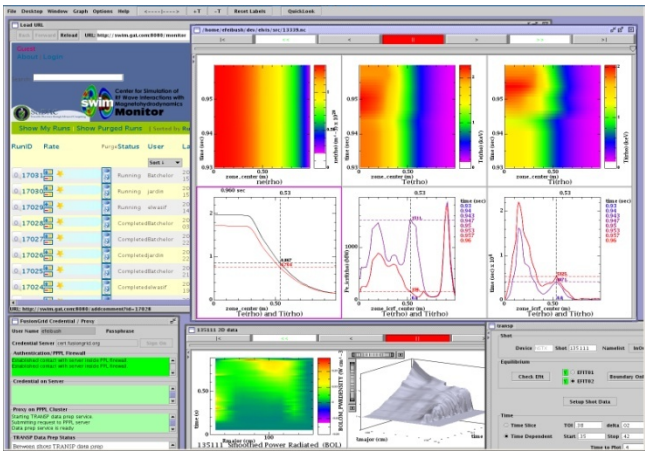

Figure 1: The scientific graphics display program, ElVis, has evolved into a client for creating input, running, monitoring, and visualizing the results from fusion simulation codes and data servers on the web.

ElVis provides numerous display and animation tools for interactively exploring scientific data. Input data is typically stored in netCDF files, which are a common output from fusion codes [6]. A novel extension to ElVis has been integrating user interfaces to legacy codes and data sources. Thus ElVis functions as a natural container for visualizing the input and output of the codes.

ElVis is portable across platforms and can be accessed from anywhere on the Internet. Its architecture enables scientists to run large simulations as a service and access large amounts of data on a server. On the server side, ElVis uses scripts and Java servlets implemented within the Tomcat [2] container to manage running simulations. On the user side, the display client is written in Java making it portable to Windows, Mac OS, and Linux, the three main operating systems run by users.

A unique sociological aspect of ElVis development is that some of the desired collaboration will be between developers of legacy code and their users. As ElVis facilitates running legacy code via a modern user interface and web service (Figure 2), it has become a platform upon which a variety of projects depend. ElVis is unique in this; ElVis developers must directly support feature requests of legacy code developers, who in turn support users of the legacy code. This enables developers to connect to a user's session and see their graphs, trouble-shoot the user's situation, and demonstrate new techniques.

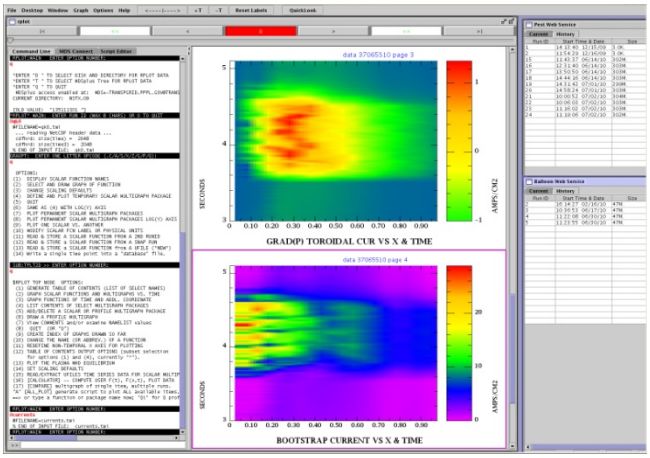

Figure 2: Collaborative web service user interface.

The primary design goal of the project was to support efficient, unconstrained collaborative editing of scientific graphs. We wished to allow an arbitrary number of users to work on the same graphs at the same time while still supporting the extensive range of operations that ElVis provides for work that is not collaborative. Furthermore, we wanted users to be able to modify graphs in real-time without having to wait on the network for every operation. These goals needed to be accomplished while leaving the existing code base mostly untouched so as to avoid introducing bugs into ElVis' scientific analysis tools.

The design was also motivated by two additional constraints. We needed to support multiple, unique groups of collaborators within one client, and we needed a framework well-suited to both presentation and friendly collaboration settings.

MATERIALS AND METHODS

We adopted a replicated architecture for our collaboration framework whereby each user's ElVis instance maintains its own copy of the collaboration data. Collaborative operations were implemented on top of ElVis' existing architecture thereby allowing network communication and synchronization issues to be resolved before executing any legacy code.

Implementation

We developed a system of interactions and groups. Interactions are small, atomic changes to program state which are applied to interaction targets. Each interaction target exists in an interaction group, which determines if and how it is synchronized. Interaction targets which are not currently collaborating are in a "local group." Each interaction group may maintain a connection to an interaction and, if a connection exists, distributes synchronized interactions to other collaborators. All conflict resolution is dealt with by the individual clients. This architecture requires minimal changes to the existing codebase.

ElVis makes use of a global interaction dispatcher to forward all valid interaction reference and execute calls to the appropriate interaction group. The dispatcher allows objects to be targeted without explicitly knowing which group they are in.

Upon instantiation, an interaction target is automatically added to the local group. When an object is brought into collaboration, it and all its children are automatically switched from the local group to the collaboration's group. Similarly, moving objects out of a collaboration consists

only of moving the objects back to the local group. All variables which are interaction targets must be annotated as such. This was a critical feature as it allowed existing code to function with the insertion of a few annotations.

Our interface for interactions is straightforward; it consists only of an execute() method and the interaction target's handle. This form is similar to a Java Runnable [10]. To execute, the interaction is passed to the global dispatcher's execute function. It will then be dispatched to the correct group, which will determine if and how to run the interaction.

Our implementation of handles is similarly straightforward. If a handle is retrieved from an interaction group (whether via the global dispatcher or directly), it remains valid for the life of the program, regardless of object mutation. It contains a getObject() method, which returns the interaction target for which it is a handle. Using handles is a simple matter of retrieving them once, storing them in an interaction, and calling getObject() within the interaction's execute() method.

Synchronization and conflict resolution

Synchronizing each user's local copy of the collaboration posed a unique design challenge. Allowing users to modify their own copy of the data in real-time leads to cases where execution order is ambiguous or undefined. In ElVis, many interactions are not invertible; this could cause a user's local state to become out-of-sync with the collaboration, as described in [11]. Operational transformation (OT) has been a popular modern method for dealing with this problem [3, 4]. In OT, operations are transformed against previously executed operations in order to guarantee consistency within each collaboration instance. Traditionally, OT has been designed to support collaborations in which operations execute quickly; in ElVis, operations are

varied in execution times. Some operations, such as highlighting, zooming, translating, and adding labels to a graph, execute very quickly; other operations, such as changing the presentation style, can be very slow. Slow operations pose a problem for a pure OT model; having to transform or undo a large operation could lead to unacceptable performance for end users. Some operations cannot be realistically transformed without one being completely undone, such as zooming on independent regions of a graph.

ElVis has a legacy API for receiving data from application programs, but the API does not extend to interactions; it is not feature-complete. Additionally this API uses a local socket connection for communication; due to firewall restrictions at target institutions, a client cannot open an arbitrary socket and listen for incoming commands. These issues made the Transparent Adaptation [11] approach for implementing collaboration impossible to directly implement.

We designed a dynamic floor control model that functions as a hybrid between traditional floor control and operational transformation. Synchronized operations are classified as either free or restricted. Free operations are those that either require no transformation against other operations or that can be efficiently transformed against other operations. An example of these operations is ElVis's collaborative whiteboard for annotating graphs (Figure 4). Users can mark up a graph with labels, shapes, and text and can modify graph colors and text colors. These operations execute very quickly and affect isolated parts of the collaboration. As such, users are allowed to perform them at any time without fear of reaching an invalid or inconsistent state. Annotations that are added simultaneously can be resolved by using their creation time.

Restricted operations generally effect large-scale state and need to be executed in isolation. An example of this interaction type is changing the presentation style of a graph,

panning, and zooming. These interactions cannot be transformed because they are mutually exclusive and can require long computation times.

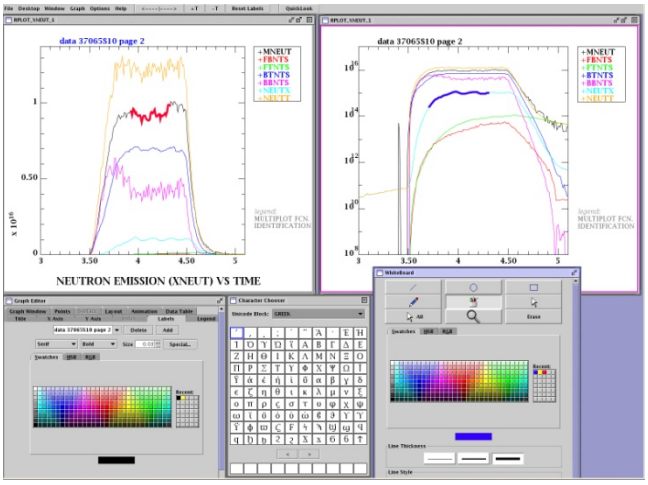

Figure 4: The collaborative whiteboard has specialized tools for scientific data, such as highlighting sections of curves, examining data on logarithmic axes, and editing labels with Greek letters.

Restricted and free operations are managed via an implicit floor control mechanism implemented by a server. To execute and post a restricted operation a user must have control of the floor, which he gains by attempting a restricted operation. At any given time, only one user may have control of the floor; this ensures that restricted operations do not conflict with one another. When users receive the floor control, they can send out an arbitrary number of restricted interactions until they either manually relinquish the floor control or the floor control is automatically taken back. Free operations can be executed at any time, regardless of the floor control holder. This behavior is configurable; the floor control can be made to restrict all interactions.

The behavior of the floor control mechanism was designed to be highly flexible in order to support different types of collaboration. The floor control mechanism can be controlled by two

parameters, an idle timeout (*t*) parameter, and a maximum single-user duration (*d*) parameter. The idle timeout parameter controls how long a user can hold the floor control without sending out an operation. The duration parameter sets a maximum time on how long the user can hold the floor control. With a large *t* and *d*, a user can give an interactive presentation using ElVis that streams to other computers while ensuring that only the presenter can modify the data. With a small *t*, many users can collaboratively manipulate the same data.

To avoid firewall issues, our mechanism for facilitating network transfer uses only standard HTTP requests. The clients register with the servlet and "park" a collaboration request. This establishes a connection between the client and servlet, allowing the servlet to respond to this request with new collaboration data, in the form of an Interaction object. After the client receives the object, the client sends a new request so it can receive subsequent objects. Clients continuously receive Interactions and notifications from the Servlet, and initially receive the floor control model from the Servlet.

RESULTS

A successful implementation was completed, and collaboration was demonstrated in both real-time and presentation mode with no noticeable lag. Legacy code modification was small, as predicted, and no aberrant behavior was noted in any tested features during runtime.

DISCUSSION/CONCLUSION

We conclude that collaboration which fulfills our requirements has been successfully implemented in ElVis. It allows users to collaboratively share and manipulate data while adding little additional overhead time to the execution of interactions. Furthermore, legacy ElVis code remains functional, demonstrating that our framework integrates seamlessly with all of ElVis'

existing diverse existing features. This should serve as an indication for implementation ease in other programs.

We have designed and implemented a collaborative graphics system that addresses the needs of scientists in the fusion research community. Any number of collaborators can work together in groups connected by a central server. The communication is handled through an HTTP interface that is compatible with firewalls. The framework is practical for use in scientific and government institutions. Our dynamic floor control model is convenient for users and effective at handling interactions in scientific graphics and visualization applications.

An interesting problem was theorized regarding network latency and program state with existing network-based ElVis services. When synchronizing web services, the initiating client opens a connection to the web service with which it interacts; these interactions and the server response is in turn synchronized with all collaborators. Similarly, a collaborator who did not initiate the web service will generating an interaction, then synchronize with the initiating client which sends to the web service. In practice, the latency caused by this change was observed to be minimal, and was preferable making services collaboration-aware.

ACKNOWLEDGMENTS

We would like to thank the U.S. Department of Energy for managing the Science Undergraduate Laboratory Internship program. We would also like to thank Princeton Plasma Physics Laboratory for organizing and sponsoring this project and Matt and Ben would like to thank their mentor, Eliot Feibush, for providing the vision, encouragement, and focus necessary to complete the project.

REFERENCES

1. Begole, J, Struble, C., Shaffer, C., Leveraging JAVA Applets: Toward Collaboration Transparency in JAVA, *IEEE Internet Computing*, Vol. 1(2), pp. 57-64, March 1997.

2. Brittain, J. and Darwin, I, *Tomcat: The Definitive Guide*, O'Reilly Media, June 2003.

3. Chengzheng, S. and Clarence, E., Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievement. *In Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pp. 59-68, 1998.

4. Ellis, C. and Gibbs, S., Concurrency Control in Groupware Systems. *ACM SIGMOD Record*, Vol. 18(2), June 1989, pp. 399-407.

5. Ki, Byeongsebob and Klasky, Scott, Scivis*, Concurrency: Practice and Experience,* Vol. 10(11-13), pp. 1107-1115, 1998. John Wiley & Sons, Ltd.

6. Rew, R. and Davis, G., The Unidata netCDF: Software for Scientific Data Access. In *Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology* (February, 1990, Anaheim, CA), pp. 33-40.

7. Richardson, T., Stafford-Fraser, Q., Wood, K., Hopper, A., Virtual Network Computing, *IEEE Internet Computing*, Vol. 2(1), January/February 1998.

8. Stefik, M., Bobrow, D., Foster, G., Lanning, S., and Tatar, D., WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Transactions on Office Information Systems*, Vol. 5(2), pp. 147-167, 1987.

9. Sun, C. and Chen, D., Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems, *In ACM Transactions on Computer-Human Interactions*, Vol. 9(1), pp. 1-41, March 2002.

10. van Holf. (2007, May). java.lang. Java language. [Online]. Available: http://download.oracle.com/javase/6/docs/api/java/lang/Runnable.html

11. Xia, S., Sun, D., Sun, C., Chen, D., Shen, H., Leveraging Single-User Applications for Multi-User Collaboration: the CoWord Approach. *In Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pp. 162-171, November 2004.