Rivet: A Flexible Environment for Computer Systems Visualization

Robert Bosch, Chris Stolte, Diane Tang, John Gerth, Mendel Rosenblum, and Pat Hanrahan
Computer Science Department
Stanford University

Abstract

Rivet is a visualization system for the study of complex computer systems. Since computer systems analysis and visualization is an unpredictable and iterative process, a key design goal of Rivet is to support the rapid development of interactive visualizations capable of visualizing large data sets. In this paper, we present Rivet's architecture, focusing on its support for varied data sources, interactivity, composition and user-defined data transformations. We also describe the challenges of implementing this architecture efficiently and flexibly. We conclude with several examples of computer systems visualizations generated within Rivet, including studies of parallel systems, superscalar processors, and mobile network usage.

1 INTRODUCTION

Computer systems are becoming increasingly complex due to both the growing number of users and their growing demand for functionality. Processors are more elaborate, memory systems are larger, operating systems provide more functionality, and networks are faster. This increasing complexity magnifies the already difficult task developers face in designing and using the new technology.

Computer systems visualization can be a powerful tool for addressing this problem, leveraging the immense power and bandwidth of the human visual system and its pattern recognition capabilities. Most computer systems visualization tools developed up to this point, however, have focused on very specific problems [12][7][4]. While some of these tools have been successful, they do not meet the demands of most developers. First, a developer may not even know about these very specialized tools. Second, these tools may not scale to the problem size. Even if they do scale, a steep learning curve is involved. Finally, even if the visualization helps a developer solve one problem, the next problem may be completely different. Computer systems analysis and visualization is a highly unpredictable and iterative process, with the demands varying greatly not only from task to task, but also iteration to iteration. What is required is a single, cohesive visualization environment that can be readily adapted to the users' needs, so they can learn the tool once and apply that knowledge to any problem.

Rivet is a visualization environment with the power and flexibility to be used in understanding a wide range of real-world computer systems problems. In designing Rivet, we encountered several challenges:

- **Supporting data transformations**. Providing data transformation capabilities within the visualization system greatly enhances its exploratory power. The environment must not only provide a robust set of standard operators, but also enable users to add their own transformations.
- **Interfacing with arbitrary data sources.** Since data collection tools for computer systems vary widely, from hardware monitors to software instrumentation to simulation, the visualization environment must be able to import large data sets from disparate sources.
- **Coordinating events, objects and data.** In order to support interactive exploration of large data sets, the system must provide coordination of multiple views and facilities for formulating visual queries.
- **Finding the right object model and interfaces**. Determining the right object granularity and the parameterizations of those objects is critical for easy configurability of the system, necessary for applicability to a broad range of problems.

In this paper, we provide a detailed description of the Rivet architecture and the challenges faced in its design and implementation. We also present several visualizations, developed within Rivet, for analyzing real computer systems problems.

2 ARCHITECTURE OVERVIEW

Figure 1 illustrates the three basic steps of the visualization process: modeling and managing data, providing visual representations of the data, and mapping the data to the visual. Visualizations may also provide some means for the user to interact with the data, its visual representation, and the mappings between them. In computer systems visualizations focused on solving specific problems, these steps can be tightly integrated into a monolithic application. However, for Rivet to be applicable to a wide range of problem domains, its architecture must be modularized, exposing the interfaces of each step of the visualization process.

In the remainder of this section, we present the architectural components and how they are combined to form visualizations. We first introduce the data and visual structures, followed by encodings, which map data to visual representations. We then describe coordination between objects in Rivet, and conclude with a brief discussion of our design choices.

2.1 Data Tuples, Tables, and Transforms

Data management in Rivet is done using a simplified relational model. The fundamental data element in Rivet is the *tuple*, an unordered collection of data attributes. Tuples with a common format may be grouped into *tables*, which store these tuples along with metadata describing the tuple contents. This homogeneous data model offers two benefits: its familiarity and the ability to easily visualize the same data set in many different ways.

Providing this data management model is not sufficient, however: users need some way to operate on the data tables. Otherwise, they would have to exit the visualization, change the data, and then import the transformed data back into the visualization environment.

As shown in Figure 1, Rivet supports data operations through *transforms,* which take one or more tables as input and produce one or more tables as output. These transforms are quite expressive, since they can be dynamically composed to form a *transformation network* expressing a more complex operation. They are also active: any changes in the data are automatically propagated.

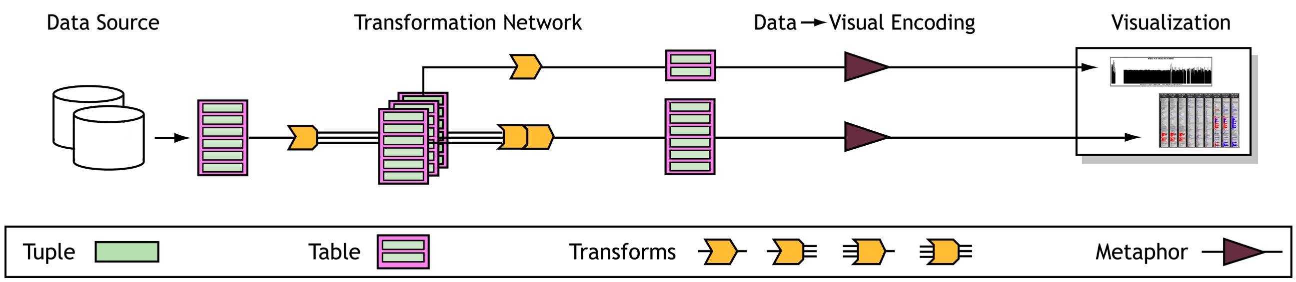


Figure 1. *A schematic depiction of the information flow in Rivet. Data is read from an external data source and then passed through a transformation network, which performs operations such as sorting, filtering and aggregation. The resulting tables are passed to visual metaphors, which map the data tuples to visual representations on the display. Interaction and coordination are not shown here.*

This property is especially useful in computer systems analysis where the data can change in real time. Rivet includes a set of standard transforms, including filtering, sorting, aggregation, grouping, merging multiple tables, and joining tables together. However, since we cannot hope to provide all operations users may need, they may write their own transforms and easily incorporate them into their visualizations.

Data can be imported from a variety of external sources. In particular, a commonly used data collection method for computer systems analysis is the generation of large *ad hoc* log files. To enable developers to easily visualize this data, we provide a regular expression parser for generating data tables from these files. To provide efficient access for multiple visualization sessions of a fixed data set, Rivet also includes the ability to directly load and save tables using a binary data format.

Related Work: Many visualization systems utilize a relational data model. The aspect of Rivet's data management distinguishing it from existing systems is its extensive support for data transformations *within* the visualization environment. Several different approaches have been taken by visualization systems to support data transformations. Some systems, such as IVEE [1], rely on external SQL databases to provide data query and manipulation capabilities. However, as has been discussed in Goldstein et al. [9] and Gray et al. [10], the SQL query mechanism is limited and does not easily support the full range of visualization tasks, especially summarization and aggregation. Visual programming and query-by-example systems such as Tioga-2 [2] and VQE [8] provide data transformations internal to the visualization environment; their transformation sets are not extensible by the user, and the existing transformations must be sufficiently simple to support the paradigm of visual programming. IDES [9] and DEVise [16] are both very flexible systems that provide extensive data manipulation and filtering capabilities through interaction with the visual representations; however, neither is easily extensible by the user. Data flow systems such as AVS [27], Data Explorer [17], Khoros [21] and VTK [23] closely match the flexibility and power offered by the data transformation components of Rivet, providing extensive prebuilt transformations and support for custom transformations. However, their focus is on three-dimensional scientific visualization, and thus they do not provide data models and visual metaphors appropriate for computer systems study.

2.2 Visual Metaphors and Primitives

Once the data to be studied has been imported and transformed into a collection of data tables, the tables are displayed using one or more *visual metaphors*. Metaphors create the visual representations for data tables by using *primitives*, which create the visual representations for individual data tuples.

Specifically, a metaphor is responsible for drawing attributes common to the table, such as axes and labels. It also defines the coordinate space for the table; thus, for every tuple in the table, it computes a position and size which are passed to the primitive along with the tuple. The primitive is then responsible for drawing the tuple within this *bounding box*.

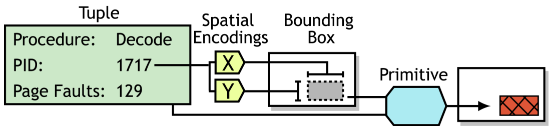
In the simplest case, a metaphor uses a single primitive to draw each tuple. However, users may wish to distinguish subsets of the data within a metaphor; for instance, they may want to highlight or elide some tuples. This task is accomplished using *selectors*, objects that identify data subsets. Metaphors may contain multiple selectors, each associated with a primitive to be used for displaying tuples in the specified subset.

Related Work: The explicit mapping of individual data tuples directly to visual primitives first appeared in the APT [18] system, and has been used in numerous systems since, including Visage [22], DEVise [16] and Tioga-2 [2]. However, the use of selectors to selectively map tuples to different visual primitives is unique to the Rivet visualization environment.

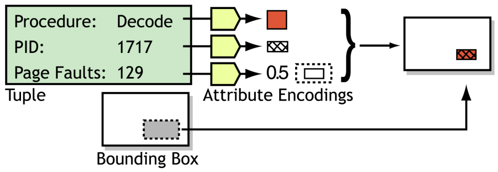
Rivet also provides mechanisms for allocating resources, such as drawing time and screen space, among metaphors. *Redraw managers* regulate the metaphor drawing process by allocating drawing time to each metaphor. Under the basic redraw manager, metaphors are given an unlimited amount of drawing time. However, more complex redraw managers may be used to restrict the metaphors' drawing times in order to provide interactivity or smooth animation. These managers actively monitor and distribute redraw time amongst metaphors. For instance, if the user is interacting with a particular metaphor, a redraw manager might allocate more drawing time to it. A metaphor can adapt to its allocation of time in a variety of ways, such as reducing the level of detail or omitting ornamentation.

Multiple metaphors may be displayed in a single window by using a *layout manager*. Layout managers utilize different techniques for allocating screen space to each metaphor, including the explicit specification of layout or the use of a regular layout such as a grid or a stack. In addition, layout managers enable the user to resize and reposition the metaphors through direct manipulation.

Finally, Rivet includes a set of *display manager*s, which handle interactions between Rivet and the underlying system. The display managers encapsulate the platform-specific display components, making Rivet easily portable to different systems. Rivet currently runs on X Windows, Microsoft Windows, and Stanford's Interactive Mural [14].



(a) Detailed view of a Metaphor



(b) Detailed view of a Primitive

Figure 2. *Diagrams depicting the creation of the visual representation of a tuple. (a) Metaphors use spatial encodings to compute the bounding box to be used by the primitive; here, the tuple's PID field determines its placement. (b) Primitives use attribute encodings to create the visual representation of the tuple within the bounding box. In this example, the color, fill pattern, and relative size of the rectangle encode three different fields of the tuple.*

2.3 Encodings From Data to Visuals

Rivet uses *encodings* to map data to visual representations. There are two classes of encodings. Metaphors use *spatial encodings* to map fields of a data tuple to a spatial extent or location; primitives use *attribute encodings* to map fields to retinal properties [5] such as color, fill pattern, and size. Examples of these encodings are illustrated in Figure 2.

Specifically, metaphors use one or more spatial encodings to determine the bounding box used by the primitive to render a given tuple. For example, a Gantt chart uses one encoding to determine the horizontal extent of a tuple, while a two-dimensional scatterplot has separate spatial encodings for horizontal and vertical axes. Because a spatial encoding can map any field or combination of fields in a tuple to a location, the metaphor itself is data independent.

A primitive uses several encodings to determine the retinal properties of a tuple's visual representation. For example, most primitives have a fill color encoding, which can be used to reflect some nominative field, such as process name, or some quantitative field, such as cache misses, of the tuple being displayed. Using encodings provides great flexibility in how a tuple can be mapped to a primitive: the user can selectively map any field or fields to any encoded retinal property of the primitive.

Related Work: The explicit use of encodings to parameterize visual metaphors and primitives is another innovation of the APT system. In APT and in subsequent systems such as Visage, encodings formalize the expressive capabilities of visual representations and are utilized by knowledge-based systems to automatically generate graphical displays of information. We find that encodings provide an ideal parameterization for visual representations within a programmable visualization environment.

2.4 Coordination

With the modular architecture of Rivet, we can achieve a lot of coordination simply by sharing objects. For example, metaphors can share a selector, enabling brushing across different displays. Metaphors can also share a spatial encoding, providing a common axis, or they can share a primitive, ensuring a consistent visual representation of data.

However, shared objects must stay consistent. All objects in Rivet subscribe to the *listener mechanism*: objects dependent on other objects 'listen' for changes. When an object is notified, it updates itself to reflect the change. For example, when a metaphor's spatial encoding is modified, the metaphor recomputes the bounding boxes for the tuples in its table. In addition to these simple examples, the listener model easily enables other features such as animation and active transformation networks.

While the listener mechanism is powerful, some situations require a more sophisticated coordination between objects. To handle these cases, Rivet provides two mechanisms: *bindings* and *selectors*.

Rivet objects raise events to indicate when some action occurs. Bindings allow users to execute an arbitrary sequence of actions whenever a specific object raises a particular event. For instance, a metaphor may raise an event when a mouse click occurs within its borders, reporting that a tuple is selected; a binding on this event could display the contents of the selected tuple in a separate view.

Selectors, introduced earlier, separate the selection process into two stages: the selection stage and the query stage. The first stage corresponds to the actions performed when selection occurs, such as raising an event or recording the tuple being selected. The second stage refers to querying the selector as to whether a tuple is selected. Metaphors use this second stage in deciding whether to elide or highlight a tuple, as described in Section 2.2.

Figure 3 provides an example showing how a coordinated multiple-view visualization can be developed using the techniques discussed in this section.

Related Work: North's taxonomy of multiple window coordination [19] identifies three major types of coordination: (1) coupling selection in one view with selection in another view, (2) coupling navigation in one view with navigation in another view, and (3) coupling selection in one view with navigation in another view. Whereas many visualization systems provide some form of coordination, the binding and selection mechanisms enable Rivet to support all three forms of coordination. Both the Visage and DEVise visualization environments provide extensive coordination support: Visage includes a well-architected direct manipulation environment for inter-view coordination, and DEVise uses *cursors* and *links* to implement inter-view navigation and selection. Whereas these implementations of coordination have highly refined user interface characteristics, Rivet's programmatic coordination architecture is more expressive and flexible. The Snap-Together Visualization [20] project presents a cohesive architecture for coordination, focusing only on the integration of numerous compiled components into a cohesive visualization. It does not, however, provide support for developing the visualizations themselves.

2.5 Architecture Discussion

Choosing interfaces to enable maximal object reuse was the main challenge underlying many design choices, including:

1. The separation of data objects from visual objects.
2. The homogeneous data model.
3. The use of encodings.
4. The separation of visual metaphors from primitives.
5. The abstraction of selectors into a separate object.

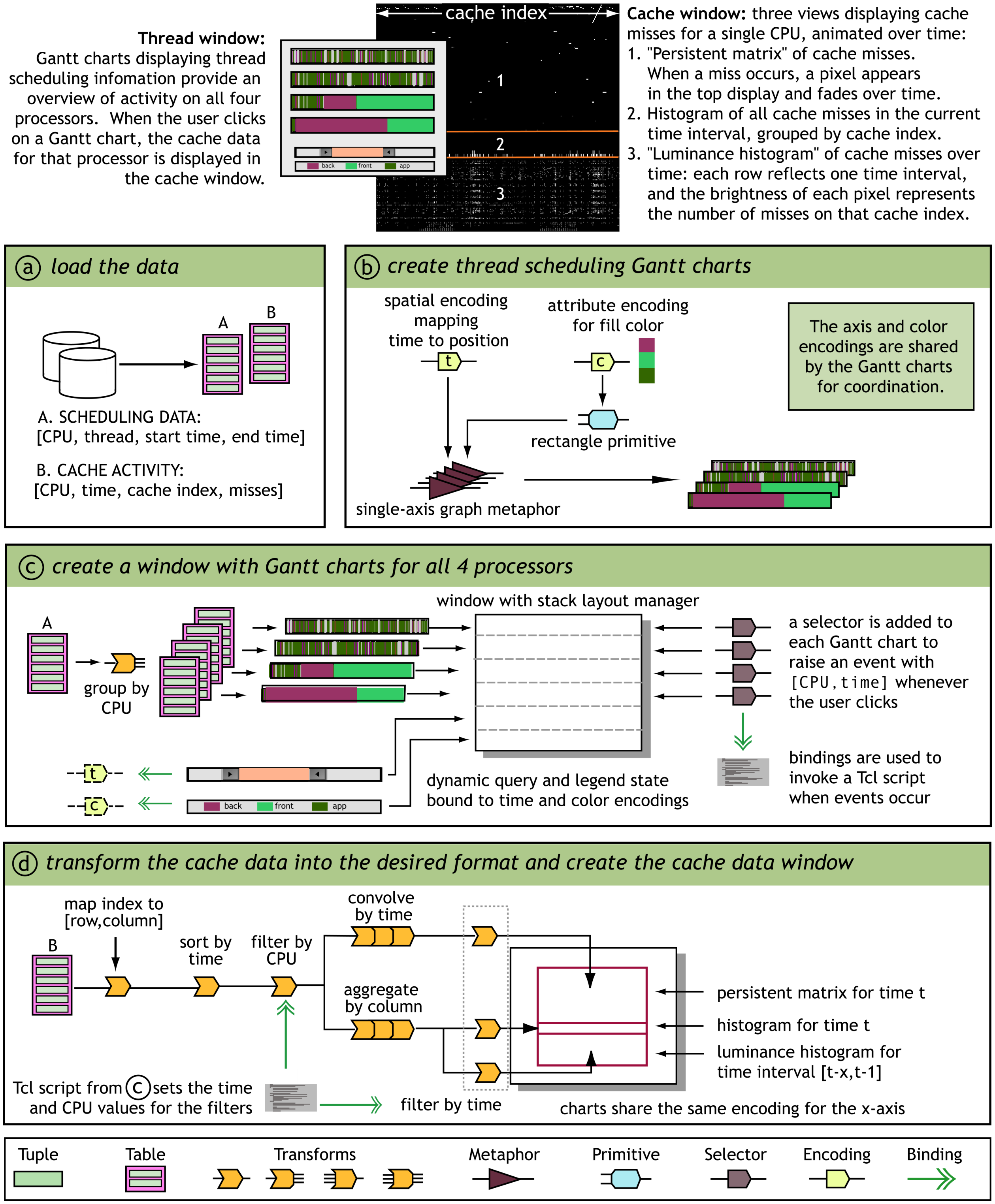


Figure 3. *An example of creating a visualization in Rivet, using data from the execution of a multiprocessing application. The visualization consists of coordinated views of thread scheduling behavior and cache miss data.*

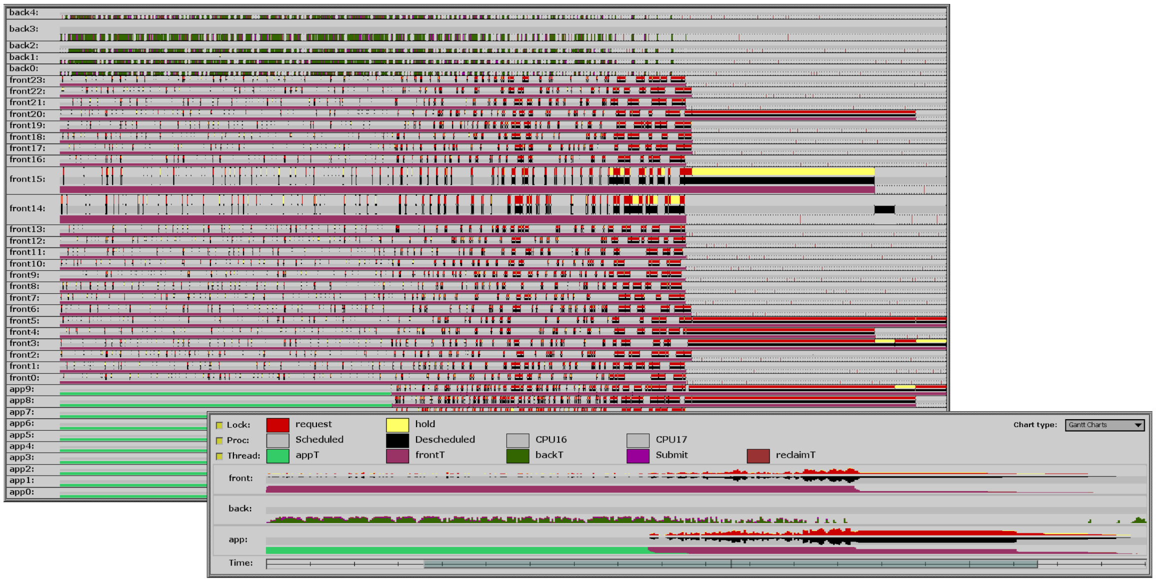


Figure 4. *A visualization used in an iterative performance analysis of the Argus parallel rendering library. The visualization is shown displaying kernel lock, processor utilization and thread scheduling information for a 39-processor run of the Argus library. This data is shown in the top view using a stack of resizable and moveable Gantt charts. The bottom view shows these application events aggregated according to process type. The legend's color scheme can be directly manipulated; any changes are propagated to the charts via the listener mechanism. The checkbuttons to the left of the legend control which event types are displayed in the top view. The time control in the bottom window acts as a dynamic query slider on the charts in the top window.*

These choices give rise to much of the functionality in Rivet. For example, the first two choices allow any data to be displayed using any visual metaphor: one visualization can have multiple views of the same data; conversely, the same metaphor can be used to display different data sets. The second choice also allows the user to build arbitrary transformation networks. The next two choices allow the user to explicitly define the mapping from data space to visual space: primitives use retinal encodings to display any data tuple, irrespective of dimensionality or type, and metaphors use spatial encodings to lay out any primitive. The last choice allows the user to have multiple views of different selected subsets of the same data; it also allows metaphors to be reused with a different interaction simply by changing which selector is used.

Several iterations were made during the evolution of the Rivet architecture. Previous Rivet implementations were more monolithic, resulting in an inability to easily change the imported data or visualizations. By choosing this modular architecture with a relatively small granularity and shareable objects, we have developed an easily configurable visualization environment applicable to a wide range of real-world computer systems problems.

3 IMPLEMENTATION CHALLENGES

The design goals of Rivet place two fundamental constraints on its implementation. First, visualizing the large, complex data sets typical of computer systems requires Rivet to be fast and efficient. Second, the desire for flexibility in the development and configuration of visualizations requires Rivet to export a readily accessible interface. In this section, we discuss these two implementation challenges.

3.1 Performance

In order to support interactive visualizations of computer systems data, a visualization system must be able to efficiently display very large data sets. This constraint requires us to use a compiled language and a high-powered graphics system. An early implementation of Rivet done entirely in Tcl/Tk was flexible but unable to scale beyond small data sets due to the performance limitations of the interpreter and the graphics library.

Consequently, the Rivet implementation now uses C++ and OpenGL. OpenGL is a widely used standard for the implementation of sophisticated graphics displays. It achieves high performance through hardware acceleration and is platform independent unlike most windowing systems, such as X11. Furthermore, using OpenGL enables Rivet to run on the Interactive Mural [14], which provides a large, contiguous screen space and support for collaborative interaction.

While OpenGL gives us the performance we need, it is not straightforward to incorporate into our modular design. Specifically, because context-switching in OpenGL is expensive, Rivet provides context management, allowing many metaphors to seamlessly share a single context.

3.2 Flexibility

While all objects in Rivet are implemented in C++ for performance, we also want to provide a more flexible mechanism for rapidly developing, modifying, and extending visualizations. Our implementation uses the Simplified Wrapper and Interface Generator (SWIG) [3] to automatically export the C++ object interfaces to standard scripting languages such as Tcl or Perl. SWIG greatly simplifies the tedious task of generating these interfaces and gives us a degree of scripting language independence. Since all Rivet object APIs are exported through SWIG, users create

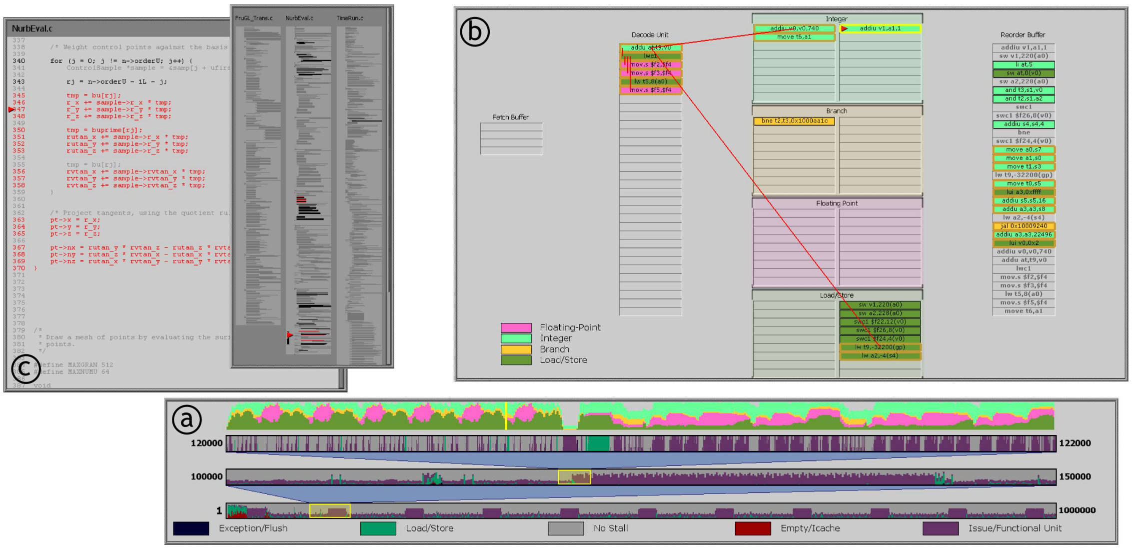


Figure 5. *A visualization designed for the study of application behavior on superscalar processors. The visualization includes three tightly coordinated components, which together provide a complete picture of the application's behavior. The timeline view (a) displays pipeline utilization and occupancy statistics for the entire period of study, and uses an interactive multi-tiered strip chart to provide rapid navigation and exploration capabilities. The pipeline view (b) animates instructions as they traverse the stages of the pipeline during regions of interest identified using the timeline view. The source code views (c) correlate the behavior being displayed in the animated pipeline view with the application source code.*

visualizations by writing scripts that instantiate objects, establish relationships between objects, and bind actions to object events.

One potential pitfall when using a scripting language is the performance cost, since the interpreter can quickly become a bottleneck if it is invoked too frequently, especially in the main event loop. However, in Rivet, high-frequency interactions are handled by the listener and selector mechanisms, which completely bypass the interpreter. While the binding mechanism relies on the interpreter to execute scripts bound to events, bindings are typically used to respond to user interactions, which are relatively infrequent (from the point of view of the computer). Thus, we are able to realize the benefits of flexibility without suffering a significant performance cost.

Related Work: Several other information visualization systems also use Tcl for describing visualizations [11][24]. VTK [23], like Rivet, integrates C++ with multiple scripting languages.

4 APPLICATIONS

The Rivet visualization environment has been successfully applied to studying several real-world computer systems problems. We discuss four applications of Rivet demonstrating its breadth of application within the computer systems domain.

4.1 Parallel Systems Performance

Achieving good application performance on scalable shared memory multiprocessors is a challenging task. We used Rivet to study the performance of Argus [15], a parallel rendering library for use in large real-time graphics applications such as scientific visualization systems. Specifically, the Argus developers encountered a scalability problem: Argus only scaled well to 26 processors before showing a sharp performance decline. They were unable to solve the problem using several traditional analysis tools, such as software profiling and hardware performance counters.

We used Rivet coupled with SimOS [13], a complete machine simulator, to perform multiple simulation and visualization iterations, each focusing on different aspects of the application and operating system. Several types of per-process events, such as thread scheduling and kernel traps, were displayed using two different metaphors according to the time scale: Gantt charts were used for detailed displays of individual events, and strip charts were used to display aggregates computed using data transforms. During the analysis process, we uncovered several surprising problems, including large amounts of contention for a kernel lock caused by a bug in the operating system [6]. Figure 4 shows one visualization used in this iterative analysis.

This study illustrates the importance of the design decisions made in Rivet. Because the data being visualized changed with each iteration, we needed to rapidly prototype different visualizations. Encodings enabled the Gantt charts and strip charts used to be easily applied to a wide range of data: processor utilization, kernel traps, lock activity and thread scheduling. Interactivity and efficient graphics were necessary to study millions of cycles of detailed activity across a large number of processes.

4.2 Superscalar Pipeline Analysis

Motivated by concerns about the increasing complexity of mainstream microprocessors and the inability of software to take full advantage of these processors, we developed a visualization for studying application behavior on superscalar processors [25]. This tool, shown in Figure 5, combines three separate views to provide an "overview-plus-detail" display of an application's execution. The application developer uses the timeline view to examine processor utilization statistics over the entire execution to locate problem areas, and then uses the animated pipeline view for a detailed study of its behavior in those areas. The source code views allow the developer to correlate the events in the other two windows with the application source code. This visualization can also be used for compiler design, hardware design and simulator debugging.

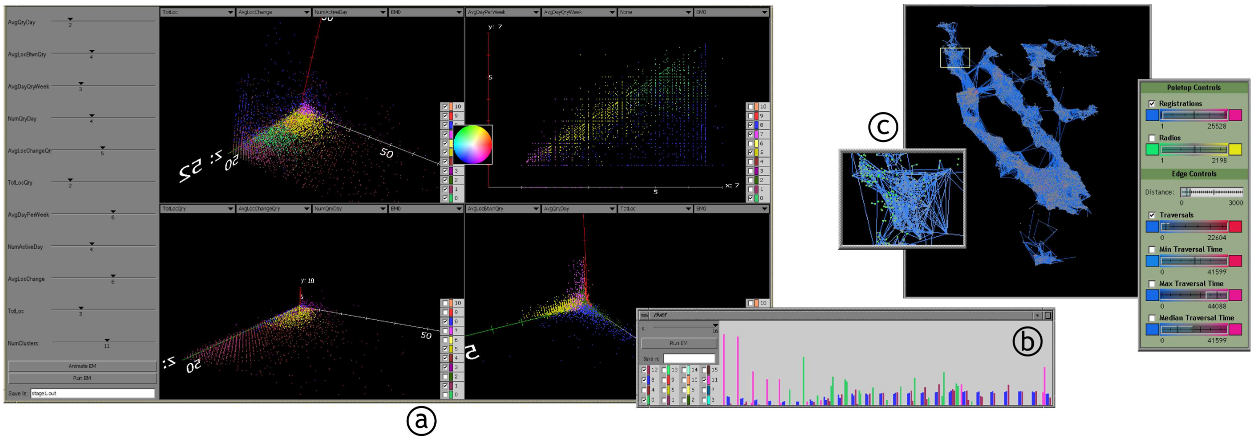


Figure 6: *Three visualizations developed to analyze mobile network usage: (a) is used to find user mobility patterns, (b) to find usage patterns in time rather than mobility, and (c) to allow the user to probe overall network statistics. (a) contains four scatterplots of the same data set, each with four pulldown menus controlling the spatial and color encodings and a legend describing the color encoding. Both (a) and (b) contain a control panel for configuring the current run of a custom clustering algorithm integrated into Rivet; the legends serve as selectors controlling which data subsets are displayed. In (a), any changes in one legend (made via a popup colorwheel) are propagated to the other views. Although the same algorithm is used in both visualizations, different visual metaphors are used, one focusing on mobility and the other on how usage varies by time of day. (c) shows a graph of the network in the San Francisco Bay Area, with an inset zoom. The control panel lets the user dynamically select which nodes and edges are displayed, as well as which parameters to use in encoding the colors of the nodes and edges.*

Coordination was essential to this visualization, since navigation and selection in all three views are linked. The modular architecture was also key, since the tool needed to be readily adaptable to different processors and processor configurations. Thus, the animated pipeline view used several simple pipe and container metaphors, rather than a single monolithic pipeline metaphor. Composing these simple metaphors together into an animated visualization of a superscalar processor was possible only by leveraging Rivet's coordination architecture.

4.3 Mobile Radio Network Usage Patterns

Studying mobile network usage patterns is important given the increasing number of people using wireless networked devices. Current research in mobile networking relies heavily on simulation; therefore, researchers need models of user movement based on actual observation. Rivet was used to help perform a detailed analysis [26] of a seven-week trace of the Metricom metropolitan-area packet radio wireless network; some of the visualizations are shown in Figure 6.

The researchers considered several other visualization tools before deciding on Rivet. However, applying several custom clustering algorithms from *within* the visualization environment was critical, since exiting the visualization to apply the clustering would have been time-consuming and frustrating. Also, the selection mechanisms in Rivet allowed the researchers to toggle between the full data set and specific subsets, necessary for understanding the data in its entirety.

4.4 Memory System Access Behavior

Most large-scale multiprocessors in use today are built using a non-uniform memory access (NUMA) model, in which knowledge of data placement and interconnection topology is necessary for achieving peak performance. Figure 7 shows a visualization of memory system behavior on a large-scale NUMA multiprocessor. In order to convey a complete picture of memory access patterns, this visualization consists of several linked views, such as an expanded version of the cache window presented in Figure 3. This visualization is heavily dependent on data transformation networks. Real-time or logged data is displayed directly in the primary view, and different transformation networks aggregate this data for display in the other four views.

5 CONCLUSION AND FUTURE WORK

We developed the Rivet information visualization environment to provide a cohesive platform for the analysis and visualization of modern computer systems. It uses a component-based architecture in which complex visualizations can be composed from simple data objects, visual objects and data transformations. Rivet additionally provides powerful coordination mechanisms, which can be used to add extensive interactivity to the resulting visualizations. The object interfaces chosen in the design of Rivet demonstrate how, with the proper parameterization, the design of a sophisticated and interactive visualization can be a relatively simple task.

Rivet has been successfully applied in focused studies of a wide range of computer systems: parallel applications, superscalar processors, memory systems, and wireless networks. In addition to continuing these focused studies, we plan to use Rivet to develop two new visualization frameworks: the Visible Computer and Visual Pivot Tables.

We have demonstrated several independent visualizations for portraying different components of computer systems, from the processor and caches to the memory system and networks. We would like to combine these components (as well as others) into a single *Visible Computer* interface. Starting with an overview, users will be able to interact with the display, allocating screen space to subsystems of interest while still providing context about the rest of the system. Such a system would be valuable both for pedagogical purposes and for detailed study of computer systems behavior.

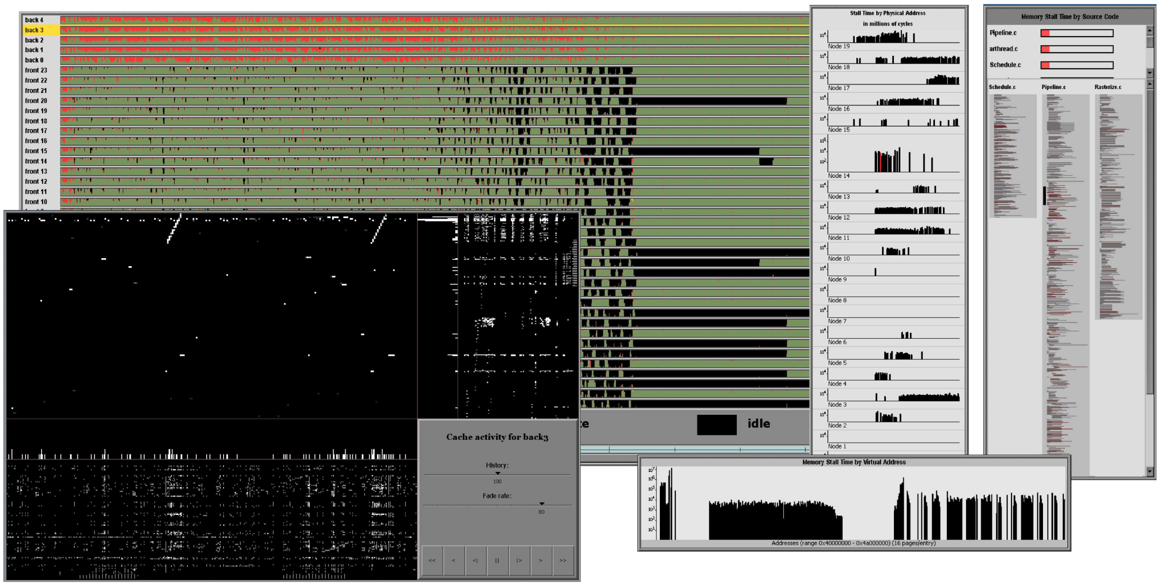


Figure 7. *A visualization of memory access patterns on a large-scale shared-memory multiprocessor. The collection of strip charts displays the total memory stall fraction over time for each processor. The two histogram windows show the stall time as a function of application virtual address and per-node physical address, and the source code overview shows the stall time incurred by each line of code. The lower left window, an extension of the cache window in Figure 3, presents multiple views of cache miss behavior on the selected processor, highlighted in the strip chart view.*

While Rivet was developed as an environment for computer systems visualization, this problem domain is sufficiently complex that the resulting environment is also appropriate for other information visualization tasks. For example, one interface we plan to explore is the pivot table, effective for navigating and exploring high-dimensional data. We plan to implement *Visual Pivot Tables* in Rivet, extending them to display tables of metaphors rather than just numbers. By encoding multiple data dimensions in both the tabular layout and the visual representations, Visual Pivot Tables will greatly simplify the task of analyzing complex data sets.

References

[1] C. Ahlberg, and E. Wistrand. "IVEE: An information visualization & exploration environment." In *Proceedings of IEEE Information Visualization 1995*, pages 66-73, 1995.

[2] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. "Tioga-2: A Direct Manipulation Database Visualization Environment." In *Proceedings of the IEEE Conference on Data Engineering 1996*, pages 208-217, 1996.

[3] D. Beazley. "SWIG: An easy to use tool for integrating scripting languages with C and C++." In *Proceedings of the Fourth Annual USENIX Tcl/Tk Workshop*, pages 129-139, 1996.

[4] R. Becker, S. Eick, and A. Wilks. "Visualizing Network Data." In *IEEE Transactions on Visualization and Computer Graphics*, 1(1), pages 16-28, 1995.

[5] J. Bertin. *Graphics and Graphic Information Processing*. Berlin: Walter de Gruyter & Co., 1981.

[6] R. Bosch, C. Stolte, M. Rosenblum, and P. Hanrahan. "Performance Analysis and Visualization of Parallel Systems using SimOS and Rivet: A Case Study." In *Proceedings of the Sixth International Symposium on High Performance Computer Architecture*, pages 360-371, January 2000.

[7] A. Couch. "Visualizing huge tracefiles with Xscal." In *Proceedings of the Systems Administration Conference 1996*, pages 51-58, 1996.

[8] M. Derthick, J. Kolojejchick, and S. Roth. "An Interactive Visual Query Environment for Exploring Data." In *Proceedings of ACM SIGGRAPH Symposium on User Interface Software & Technology*, pages 189-198, 1997.

[9] J. Goldstein, S. Roth, J. Kolojejchick and J. Mattis. "A Framework for Knowledge-Based, Interactive Data Exploration." In *Journal of Visual Languages and Computing*, pages 339-363, December 1994.

[10] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman and H. Pirahesh. "Data-cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total." In *Data Mining and Knowledge Discovery*, pages 29-53, 1997.

[11] T. He and S. Eick. "Constructing Interactive Visual Network Interfaces." *Bell Labs Technical Journal*, 3(2), pages 47-57, Lucent Technologies, April-June 1998.

[12] M. Heath and J. Etheridge. "Visualizing the Performance of Parallel Programs." In *IEEE Software*, 8(5), pages 29-39, September 1991.

[13] S. Herrod. "Using Complete Machine Simulation to Understand Computer System Behavior." Ph.D. Thesis, Stanford University, February 1998.

[14] G. Humphreys and P. Hanrahan. "A Distributed Graphics System for Large Tiled Displays." In *Proceedings of IEEE Visualization 1999*, pages 215-223, 1999.

[15] H. Igehy, G. Stoll and P. Hanrahan. "The Design of a Parallel Graphics Interface." In *Proceedings of SIGGRAPH 1998*, pages 141-150, August 1998.

[16] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki and K. Wenger. "DEVise: Integrated Querying and Visual Exploration of Large Datasets." In *Proceedings of ACM SIGMOD*, pages 301-312, May 1997.

[17] B. Lucas, G. Abram, N. Collins, D. Epstein, D. Greesh and K. McAuliffe. “An architecture for a scientific visualization system.” In *Proceedings of IEEE Visualization 1992*, pages 107-114, October 1992.

[18] J. Mackinlay. “Automating the Design of Graphical Presentations of Relational Information.” In *ACM Transactions on Graphics*, 5(2), pages 110-141, 1986.

[19] C. North and B. Shneiderman. “A Taxonomy of Multiple-Window Coordination.” *University of Maryland Computer Science Department Technical Repor*t #CS-TR-3854.

[20] C. North and B. Shneiderman. “Snap-Together Visualization: Coordinating Multiple Views to Explore Information.” *University of Maryland Computer Science Department Technical Report* #CS-TR-4020, 1999.

[21] J. Rasure and M. Young. “An open environment for image processing software development.” In *Proceedings of the SPIE Symposium on Electronic Image Processing*, pages 300-310, February 1992.

[22] S. Roth, P. Lucas, J. Senn, C. Gomberg, M. Burks, P. Stroffolino, J. Kolojejchick and C. Dunmire. “Visage: A User Interface Environment for Exploring Information.” In *Proceedings of the IEEE Information Visualization Symposium 1996*, pages 3-12, 1996.

[23] W. Schroeder, K. Martin and B. Lorenson. The Visualization Toolkit. An Object-Oriented Approach To 3D Graphics. 2nd Edition, Prentice Hall, 1997

[24] G. Sevitsky, J. Martin, M. Zhou, A. Goodarzi, H. Rabinowitz. “The NYNEX network exploratorium visualization tool: visualizing telephone network planning.” In *Proceedings of the SPIE - The International Society for Optical Engineering 1996*, vol. 2656, pages 170-180, 1996.

[25] C. Stolte, R. Bosch, P. Hanrahan and M. Rosenblum. “Visualizing Application Behavior on Superscalar Processors.” In *Proceedings of IEEE Information Visualization, 1999*, pages 10-17, 1999.

[26] D. Tang and M. Baker. “Analysis of a Metropolitan-Area Wireless Network.” To appear in *Wireless Networks.*

[27] C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. van Dam. “The application visualization system: A computational environment for scientific visualization.” In *IEEE Computer Graphics and Applications*, 9(4), pages 30-42, July 1989.