Universal Visualization Platform

Alexander G. Gee*, Hongli Li, Min Yu, Mary Beth Smrtic, Urska Cvek, Howie Goodell, Vivek Gupta, Christine Lawrence, Jianping Zhou, Chih-Hung Chiang, Georges G. Grinstein
Institute for Visualization and Perception Research, University of Massachusetts Lowell, Computer Science, 198 Riverside St., Olsen Hall, Room 301, Lowell, MA USA 01854

ABSTRACT

Although there are a number of visualization systems to choose from when analyzing data, only a few of these allow for the integration of other visualization and analysis techniques. There are even fewer visualization toolkits and frameworks from which one can develop ones own visualization applications. Even within the research community, scientists either use what they can from the available tools or start from scratch to define a program in which they are able to develop new or modified visualization techniques and analysis algorithms.

Presented here is a new general-purpose platform for constructing numerous visualization and analysis applications. The focus of this system is the design and experimentation of new techniques, and where the sharing of and integration with other tools becomes second nature. Moreover, this platform supports multiple large data sets, and the recording and visualizing of user sessions. Here we introduce the Universal Visualization Platform (UVP) as a modern data visualization and analysis system.

Keywords: data visualization, information visualization, system architecture, design, performance

1. INTRODUCTION

The visualization and analysis of data is the key to understanding complex interactions. Early data analysis involved the use of ledgers and calculators. Over time, these tables grew and migrated first to computer-based spreadsheets, and eventually stored within databases. Gone are the days of looking at a few values to make decisions, now we look at charts, graphs and other visual depictions of numerous returns. Consequently, within the last decade or so we have witnessed a sudden explosion in the research and development of techniques and applications providing access to our ever-growing data sets.

The common underpinning of past and present research in the area of data visualization is and has been the development of specific algorithms for solving particular problems; and in most cases a new application is created to exploit these new algorithms and their functionality. Researchers rarely extend existing applications to harness both new and old ideas since existing applications come with inherent personalities, their own underlying assumptions, restrictions and coding conventions. Today researchers have a choice of commercial and research platforms from which to choose from when selecting an existing system to extend; however, none provide a mechanism by which user-definable visualization and analysis algorithms can be easily developed and integrated. Moreover, almost all of these systems lack support for the recording of user actions and system events, and the ability to visualize, interact, and analyze these user sessions.

We propose the Universal Visualization Platform (UVP) as a general high-performance data visualization and analysis platform. The platform provides a common infrastructure from which to build knowledge discovery applications, while at the same time supporting experimental development.

In section 2, we review existing visualization and analysis systems and toolkits, pointing out known assumptions and important features. In section 3, we generalize the current state that led to the definition of our fundamental
We then describe our Java implementation to support research on multiple hardware architectures, in section 4. Section 5 presents an example application and tools built upon the UVP, and section 6 discusses ongoing and future work. Finally, we wrap up with our summary and conclusions in section 7.

2. RELATED WORK

Within the last few years, researchers and companies have started to define information visualization toolkits and platforms from which others can add functionality, design tools, or create applications. Christopher Ahlberg introduced Spotfire\(^1\) back in 1996 (DecisionSite from Spotfire, Inc.) as a visualization application that supports external integration and execution as a Microsoft Windows COM object. Stephen Eick in 1999 introduced Advizor\(^3\) from Advizor Solutions, Inc., a Microsoft Windows component based visualization tool set. That same year Chris North and others introduced the Snap-Together Visualization\(^4,5,6\) framework that provides a mechanism for linking together independent visualization programs using Sun’s Java. Out of the work by Leland Wilkinson and his ‘Grammar of Graphics’ came nViZn\(^7,8\), which provides a toolbox for defining statistical graphics. Finally, Jean Daniel Fekete in 2003 released his InfoVis Toolkit\(^9\), an information visualization infrastructure for building applications that support an existing set of techniques.

Spotfire’s DecisionSite\(^10\) is a data analysis application based on dynamic queries. DecisionSite is limited to working with a single data set; all open visualizations observe and link to this single data space, providing simple coordinations between visualizations. The latest version of DecisionSite includes web-service plug-in guides that provide preconfigured procedures for analyzing domain specific data sets. While it is possible to extend the latest version using the available API, there is no mechanism by which additional visualizations and tools can be added other than via application calls from external programs.

Visual Insights’ Advizor\(^3\) is a platform for building interactive visual query and analysis applications that consists of a set of visual components and an in-memory data pool. Advizor is built upon the Microsoft Windows ActiveX architecture and COM objects in which developers can define data exploration environments based on components provided by Advizor and custom programs\(^11\). Furthermore, Advizor also implements the standard linear user actions of undo and redo via the provided API; however, there is no global event stack, but rather each component maintains its own event stack.

Snap-Together Visualization\(^4\) (STV) is a framework by Chris North and others for supporting multiple coordinated views – the linking of visualizations for sharing state information. STV is a Java based centralized software system that defines the glue connecting multiple independent programs viewing a common underlying database. Coordinations within this framework are defined by the sharing of events between programs. Fusion\(^6\) is a web-based implementation of the STV system that provides a simplified approach for the specification of visualization workspaces.

nViZn\(^8,12\) is a Java based graphics API developed as a language for presenting statistical graphics on the World Wide Web. Unlike most other data visualization systems that provide visualization components, nViZn defines the building blocks for constructing visualizations using graphical elements, which are mapped to data. In order to support data exploration nViZn added a collection of controllers, objects that connect user gestures to graph functions\(^12,13\).

InfoVis Toolkit\(^9\) by Jean-Daniel Fekete is designed to support the creation, extension and integration of information visualization components into interactive Java applications. It provides a large collection of components from which to build advanced visualization applications. In order to support drawing demands required by visualization tools, the InfoVis Toolkit supports hardware acceleration through Agile2D, an implementation of Java2D based on the OpenGL API\(^14\).

Beyond the standard information visualization systems, there are also those systems supporting the visualization of scientific data. AVS (Advanced Visualization System)\(^24\) and Khoros\(^25\) are two example systems that provide a visual programming interface for constructing data-flow models using an assortment of functional nodes. These systems provide APIs for defining and including user-definable nodes.
3. GENERALIZED ARCHITECTURE

Existing applications and toolkits all solve specific system problems. Each one furthers our understanding of how users interact with data in a quest to obtain information and knowledge. By extracting out the fundamental features of each system and generalizing them into a unified framework, an attempt is made to evolve our understanding of data visualization applications into a more coherent knowledge discovery core.

A visualization and analysis system can be divided into three major parts: a framework, an application and numerous tools. The framework represents the functional core of the system providing the required objects and their relationships upon which applications and tools are built. Second, the containing application, a program that instantiates the framework, defines the system for analyzing data. Finally, the application provides a set of tools, subprograms that perform tasks via direct interactions from users or indirect events thrown by underlying objects.

![Diagram of a generalized architecture](image)

Figure 1 A generalized architecture defining the core components of a well-rounded data visualization and analysis system, and showing the primary managed objects and plug-in tool types.

A generalized system framework can be defined by three primary components: a data model, a linking mechanism and user sessions. The data model defines the data structure into which data is written to and read from by the various tools. All tools built using the framework all share a common data structure from which to analyze data. The linking mechanism defines dynamic interrelationships between tools and their associated data structures. New to most visualization and analysis systems is the concept of user sessions. User sessions capture the history of user actions and system events useful for generalized undo and redo, and branching history interactions. A new fourth component is introduced that facilitates sharing and development of tools, a plug-in mechanism by which tools can be automatically and interactively added to applications.

Applications and tools are then built on top of the framework. While the application instantiates, owns and initializes the framework, it is the tools that actually use the framework to perform their tasks. Applications come in two flavors, those created for domain specific analyses of known data sets with associated tools, and others written for more general purpose discoveries from assorted data sets using a variety of tools. Both are based upon a centralized framework, and proposed here to be of a single common architecture. Likewise, tools can be domain specific or generic; however, when all tools work with the same underlying data model then specialized tools can potentially be used within other domains that support similar data characteristics.
3.1 Data model

Of all the data visualization and analysis systems available, the one component that all these systems share is an underlying data model. The data model provides both the memory structure to store data and the interface methods to access this data. A data model then becomes a placeholder for data sets, collections of observations and values.

The data model of the UVP is based on two-dimensional data arrays consisting of records and dimensions (i.e. table rows and columns respectively), along with a standard set of interface methods to read and write values (see Figure 2). A requirement of this data model is its support for large data sets. To accommodate larger data sets the UVP includes a basic caching mechanism. Furthermore, the data model incorporates a data compression mechanism that takes advantage of the redundancy in common data sets.

![Figure 2](image)

Figure 2 The UVP includes an extended data model that supports the addition of metadata. A general two-dimension data table is defined of dimensions and records, and the new record called the descriptor. Descriptors are records with the functionality of dimensions.

The platform supports the primitive data types, including integers, longs, floats, and doubles – both 32-bit and 64-bit numbers are provided for data consistency and precision when needed. The UVP also includes dates, strings and URLs. Additionally, the data structure supports arbitrary objects to store complex structures, such as chemical encodings and graphic icons.

The platform includes the concept of dimension typing, the characterization of how values relate within a single dimension. The UVP includes the four well-known dimension types or scale measures, nominal, ordinal, interval, and ratio\(^2\). The nominal type refers to objects that have no natural ordering. The ordinal type implies a natural ordering, for example the days in the week. The interval type defines orderable objects with inherent measurable interrelationships. Finally, the ratio type includes the notion of zero and the concept of measurable differences between objects. These dimension types provide additional domain specific information about the data set under analysis.

The UVP also supports the additional of metadata, additional descriptions about records or dimensions. Record based metadata can be easily encoded using additional standard dimensions. In contrast, metadata about dimensions requires extending the standard data model. To accommodate this second type of metadata, the UVP introduces a new type of record with the functionality of a dimension, called a descriptor (see Figure 2). Descriptors describe the relationships between dimensions. Like a dimension, a descriptor includes a unique name, data type, and descriptor type.

Another structure commonly employed to encode information is the tree or graph that consists of nodes and edges. While graphs are commonly stored in memory using linked objects, the UVP maintains trees and graphs using the provided two-dimension array structure. This approach is an extension of InfoVis Toolkit’s\(^9\) implementation, and generalized to handle multiple associated dimensions. Furthermore, the UVP can also encode trees and graphs using two linked data models, a generalization of Advizor’s\(^11\) web plot. This second encoding scheme supports the addition of both node and edge description data.
The analysis of data generally involves the linking of multiple data sets together in order to provide a richer source of information to analyze. Whereas some existing systems only support the storage of a single data set per instantiation, the common trend in most modern systems is to support multiple data sets. Likewise, the UVP supports multiple data sets and the linking of the data models using standard database joins.

Another important aspect of the platform is its support for dynamic operations and the generation of data modification events. Actions performed on data arrays by direct manipulation from algorithms result in an automatic data event. Dynamic data arrays naturally lend themselves to supporting streaming data sources.

Finally, the UVP’s data model supports persistent data arrays for nondestructive data modifications using a simple versioning mechanism. Versioning of data write operations provides the feasibility for real-time undo and redo data operations, including computationally expensive actions.

### 3.2 Linked Collections

A second component of a visualization and analysis system is the support for interactive linked visualizations. Most visualization environments support the minimum of one coordination – the linking of multiple visualizations in such a way that events within one linked visualization are shared with the other linked visualizations\(^\text{30}\). The UVP maintains multiple linked collections supporting a generalized space of applications. Furthermore, the platform extends the idea of linked visualizations to include the linking of analysis algorithms.

![Linked Collection](image)

Figure 3 Within the UVP linked collections define groups of coordinated tools. Additionally, tools are indirectly associated with data sets via the linked collections the tools belong too.

Linked collections provide more than just the relationships between algorithms; they also define the associations between algorithms and data models. Algorithms applied to a data model belong to the associated linked collection, and visa versa. Furthermore, the platform’s linked collections specify the joining of data models. Unlike existing systems, the UVP supports linked collections as an object between data models and algorithms in order to capture the requirements necessary for defining coordinations.

### 3.3 User sessions

Modern visualization and analysis systems have begun to include support for the undo and redo of user actions. Most implementations use a standard command stack approach that provides a linear access to user actions. While it is convenient to use a command stack, it lacks the capacity to handle branched undo and redo sequences. Branches occur in a command history when a user performs one or more undo actions and then continues to interact with the system in a way not equivalent to the next stored redo command. Branching histories result in user session trees.

Although it is still possible for applications developed based on the UVP’s framework to include the standard undo and redo of user actions, the interactive capabilities of branching histories is only possible through more advanced user interfaces. To support the visualization of and interaction with session trees, user actions and system events are encoded within the common data model. This encoding enables the analysis of user sessions for learning, predicting, recommending, and evaluating action sequences and patterns.
Furthermore, the platform provides the capability to save the user’s current analysis session. Depending on user preferences, this may be a snapshot of the system state with all the various parameters and data sets, or entire user session, including the complete session history and associated data. Saved sessions can be reopened later to continue the discovery process from the last saved point, to replay the session for demonstration or educational purposes, or to analyze and compare user sessions.

3.4 Plug-in tools

Existing visualization and analysis applications include a collection of precompiled tools, visual and analytical subprograms designed to perform specific data analysis tasks. Most of the systems that support the integration of custom algorithms generally require compiling new code into the main application or registering the new program with the application, neither of which is typically a straightforward activity. The introduction of web services is a step towards sharable programs. However, web services also require some setup procedures. Research scientists need an easier way to develop and share their creations with their colleagues, a process that should not interfere with their discovery tasks or development work.

Figure 4 The Universal Visualization Platform defines the framework from which applications and tools are built. Applications instantiate the UVP and defines the location of tools within local machine directories and on remote web servers. The UVP automatically identifies the appropriate tools located within the specified tool locations. Finally, applications can request tools from the framework using the tool’s package name, or request the list of available tools. All plug-in tools extend the base tool, which provides all the internal functionality. The UVP defines four primary tools types: visualization, analysis, import and export, and external interface tools.

The UVP includes a new framework component that supports plug-in tools, subprograms that perform specific analysis tasks. The plug-in mechanism simplifies the integration of tools within an application and provides a method by which researchers can quickly develop and share custom tools. The UVP’s framework provides methods for specifying local and remote tool directories, and handles the listing and opening of desired tools at runtime. For the development of tools, the platform includes support for automatically updating running code based on modified plug-in tool versions.

Plug-in tools support one of four primary tool templates. These tool templates provide the required plug-in interfaces and specify the required functionality appropriate to each template type. These templates include data import and export tools, visualization tools, analysis tools, and external interface tools.

Data import and export tools read and write data models, providing the data to be analyzed. Data tools generally read and write flat files, connect to databases and return queries, and communicate with streaming data sources.

Visualization tools represent tools that provide visual views of data models. These include both the classic visualizations like scatterplots, line charts, and bar graphs, as well as more modern techniques such as heatmaps, parallel coordinates, and radviz to name a few. In addition to the standard visualization tool template they extend, these tools provide additional functionality as needed, such as layered displays, graphical annotations, selection
mechanisms, and other common components provided by the platform.

Analysis tools provide support for the inclusion of computational tools that manipulate and modify data values. These tools encode various statistical, machine learning and data mining algorithms, along with other numerous data manipulation procedures, including data normalization, filtering and searching techniques.

External interface tools provide connections between UVP based applications and other independent applications enabled via any number of data exchange mechanisms. These tools provide connectivity to other well used public and commercial software packages, including WEKA$^{26}$, Clementine$^{23}$, R-project$^{31}$, and many others.

3.5 Platform API

The UVP framework API provides the necessary methods for building applications and wraps the internal mechanisms of the UVP. The API includes methods for specifying tool locations, listing available tools, requesting individual tools, and obtaining linked collections. As depicted in Figure 4, the UVP is the underlying platform that supports the various tools.

4. IMPLEMENTATION

The Universal Visualization Platform (UVP) as described in the previous section is implemented in Java to provide the potential to run on multiple hardware architectures. Furthermore, Java was chosen as the language more likely to be used by research scientists in both commercial and academic environments, given its broad acceptance and popularity. While it is known that Java is not the best performing language, the UVP is able to handle and successfully work with one million records and with reasonable interactions, under one minute.

The UVP’s data model is optimized for data reads, given the nature of visualization algorithms and the requirement for rapidly accessing values. Consequently, the addition and modification of values incurs a performance hit due to type checking and permission verification.

The plug-in tool mechanism implements Java’s class loading functionality to dynamically load and reload tool code on the fly. The platform supports the incorporation of tools in compressed jar files and as a collection of original class files. The current plug-in implementation requires tools to be structured using a prescribed package format and associated directory hierarchy. Applications identify the location of available tools, UVP then loads the appropriate classes for use, and finally the application obtains actually tool instances from the framework.

5. EXAMPLE APPLICATION

Vizit is one example application designed using the Universal Visualization Platform’s API (shown in Figure 5). This application supports all the platform’s capabilities, including multiple linked visualization and analysis tools, support for multiple large linked data sets, and fully functional user session recording and browsing.

As with other visualization systems, Vizit includes a toolbox of tools from which to start using the platform. These tools include the standard tables as textural views of the data model. Also included are line graphs, bar charts, histograms, and heatmaps. Other standard two-dimensional visualizations are also provided, including scatterplots and pie charts. Additional visualizations include parallel coordinates and radviz plots.

Beyond the visualization tools, Vizit includes a variety of data analysis tools. These include tools to compute statistical summaries across both records and dimensions, providing additional metadata about the data set. Various data clustering tools are also included like hierarchical algorithms, k-means and self-organizing maps. Most analysis tools generate additional dimensions or descriptors that can be further analyzed.

Other tools that are currently provided with the first release includes data import and export tools to standard flat file formats such as comma separated and tab delimited, and xml formatted data. Additionally, a basic database SQL
query import tool is provided.

Figure 5 Screen snapshot of the Visit application demonstrating the inclusion of plug-in visualization tools. From the application’s menu bar is the submenu of available visualization tools, while Vizit maintains the menu objects the UVP framework stores the information about available tools. The various visualization tools displayed are all associated with a single linked collection; selection of objects within one visualization window will be highlighted within the other displays.

In an initial trial experiment, students of Dr. G. Grinstein’s Fall 2003 Visualization course, at the University of Massachusetts Lowell, were provided with Vizit and tasked to program specific visualization techniques as one of their projects. The Vizit application included both pie chart and scatterplot tools with accompanying code as a starting point for writing custom techniques. Students were tasked to create various standard visualizations defined in available publications. Working in pairs, his students successfully produced their assignments within an average of three hours using the provided example visualization tools code. In addition, students were able to share their tools with other groups to perform their final task of analyzing various data sets.

6. ONGOING AND FUTURE

Beyond the current state of the Universal Visualization Platform, there are a number of aspects still to be considered, tested and integrated. First, the available set of initial tools needs to be extended. Second, the enhancement of visualization tools to harness graphics hardware. Third, the simplification of visualization tool development by providing higher-level objects.

The current platform provides a standard set of visualization tools including scatterplots, bar charts, pie charts, radviz plots, and heatmaps. The next steps includes increasing the collection of tools with additional visualization, analysis procedures, mining algorithms, other data import and export tools, plus connections to other commonly used software packages.

In order to meet the demands of researchers working on large data sets, development work will continue to performance tune the framework. One area of research is the application of OpenGL hardware acceleration to visualization algorithms. Fekete demonstrated that using Agile2D sufficiently increases performance for information visualization techniques.

Finally, to simplify the customization and development of tools, a range of development options needs to be provided by the platform. Currently, tools are created using a minimal amount of common shared components. For some
developers of tools this is exactly how they like it, working from the ground up with access to all the internal workings of the tool. However, one of the goals of this platform is to provide support for researchers at every skill level to be able to define and share tools. Consequently, future versions of the platform will provide collections of common visualization widgets and tool components from which users can combine to define tools. Later, a scripting mechanism will be defined to simplify the actual programming of tools, similar to that defined by Processing.

7. CONCLUSION

Presented here is an introduction of the Universal Visualization Platform, which is being developed to provide research scientists with the capability to easily incorporate new tools with existing tools in an integrated data discovery system. The UVP aims to enable the sharing of tools and to assist in the development of such tools.

The coupling between visualization and analysis tools is an important aspect of this platform. The coordination of different kinds of tools along with a dynamic data model provides the potential to visualize data mining algorithms in progress. More importantly, with the use of persistent data structures and the recording of both user actions and system events for undo and redo, it enables the replay and interactive adjustment of data mining algorithms to test hypotheses and assumptions designed into such algorithms.

While there are a number of available visualization and analysis applications current available to choose from analyzing data it is generally difficult to incorporate ones own tools into such generally closed systems. Furthermore, each system includes underlying assumptions and limitations that potentially restrict and hinder the development of new ideas. By combining the various concepts mentioned above into a unified platform wrapped into a common API, the UVP provides a platform that will support research scientists and students of visualization and analysis techniques in their development and sharing of ideas and creations.

REFERENCES


