Published in final edited form as:
Int J Geogr Inf Sci. 2011 January 1; 25(2): 191–210. doi:10.1080/13658810903214203.

The GeoViz Toolkit: Using component-oriented coordination methods for geographic visualization and analysis

Frank Hardisty and **Anthony C. Robinson**
GeoVISTA Center, Department of Geography, The Pennsylvania State University, University Park, PA, USA

Dutton e-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University, University Park, PA, USA

Frank Hardisty: hardisty@psu.edu

Abstract

In this paper we present the GeoViz Toolkit, an open-source, internet-delivered program for geographic visualization and analysis that features a diverse set of software components which can be flexibly combined by users who do not have programming expertise. The design and architecture of the GeoViz Toolkit allows us to address three key research challenges in geovisualization: allowing end users to create their own geovisualization and analysis component set on-the-fly, integrating geovisualization methods with spatial analysis methods, and making geovisualization applications sharable between users. Each of these tasks necessitates a robust yet flexible approach to inter-tool coordination. The coordination strategy we developed for the GeoViz Toolkit, called *Introspective Observer Coordination*, leverages and combines key advances in software engineering from the last decade: automatic introspection of objects, software design patterns, and reflective invocation of methods.

Keywords

Software design; coordination; geographic visualization

Introduction

The GeoViz Toolkit (GVT) is a geovisualization environment that lets users easily create and modify custom palettes of coordinated exploratory and analytical tools. GVT users can build a custom geovisualization application by adding new views interactively, requiring no understanding of programming languages or technical details. Exploratory views such as scatter plots, choropleth maps, cartograms, and parallel coordinate plots (as illustrated in Figure 1), are augmented with spatial statistics tools like a spatial autocorrelation explorer, and cluster detection methods from Proclude (Conley *et al.* 2005) to help users develop and evaluate hypotheses. Behind the scenes, a sophisticated coordination architecture ensures that views can be added or taken away while retaining datasets, and visual characteristics like color and classification settings.

In the following sections we describe relevant prior work, the software architecture and design of the GVT with special attention to the *Introspective Observer Coordinator*, advantages that this coordination strategy enables, and an example real-world application using the GVT. We conclude with thoughts on our future plans for GVT development, dissemination, and evaluation.

Our interest in developing a new, user-friendly geovisualization environment is inspired by a wide array of previous work to develop geographic visualization and information visualization systems. Earlier examples of full-featured geovisualization application development environments include Descartes (Andrienko *et al.* 1999) and its successor, CommonGIS (Andrienko *et al.* 2002), which were designed to automatically suggest tools and representations based on user tasks and available data, and GeoVISTA *Studio* which implements a visual programming interface that allows sophisticated users to assemble applications using a data-flow paradigm. The GVT project is in large part derived from *Studio*. An extension to the GVT designed to support collaboration, called GeoJabber, is described in a recent paper (Hardisty 2009). A wide range of other geovisualization projects have included *Studio* components, including the VIT (Chen 2006) and ESTAT (Robinson 2007a) projects, which contain novel multivariate analysis techniques, as well as the VIS-STAMP (Guo *et al.* 2006) and ICEAGE (Guo 2003) projects, which are notable for allowing interaction with highly multivariate data spaces. These *Studio* tools were preceded by other GeoVISTA work on combining statistical and geographic views. (Edsall 1999) The Improvise system (Weaver 2004) also provides highly-coordinated visualization facilities, as well as meta-visualization views. The GAV Toolkit (Jern *et al.* 2007) provides a set of geovisual analysis tools. The GAV Toolkit's functions and appearance echo the earlier systems referenced above. Source code for the GAV Toolkit is currently unavailable for us to make direct comparisons to the GVT. These environments offer multiple, coordinated views that feature dynamically-linked interaction behaviors that allow users to highlight, select, and brush spatio-temporal data. GeoVISTA *Studio* is an open source toolkit that provides the means to construct custom geovisualization applications using views and coordination mechanisms implemented as discrete JavaBeans (Gahegan *et al.* 2002). In practice, *Studio* works best for expert users who have a solid understanding of Java programming (Gahegan *et al.* 2008).The GVT is in many ways an evolutionary step beyond the GeoVISTA *Studio* application construction environment. The design philosophy for the GVT was to take the components developed for GeoVISTA *Studio* and wrap them in a coordination mechanism that allows non-expert users to simply select the views they desire and create a custom application.

The aforementioned geovisualization application construction environments draw from information visualization application construction environments like like the InfoVis Toolkit (Fekete 2003), the Visualization Toolkit (VTK) (Azevedo *et al.* 2000), and prefuse (Heer *et al.* 2005) provide libraries of ready-to-use components that can be assembled and extended by software developers. Additional motivation for our work on the GVT is to couple coordinated, interactive geovisual representations with advanced spatial analysis techniques. In previous research (our own and completed by others) we have learned that many of the users who might want to work with geovisualization software would like to see visually-led exploratory approaches augmented with quantitative spatial analysis methods to confirm or reject hypotheses they generate (Rinner 2007, Robinson 2007b). We also draw upon related software development efforts that have successfully coupled interactive geovisual representations with spatial analysis methods like the Space-Time Analysis of Regional Systems (STARS) toolkit (Rey *et al.* 2006) and GeoDA (Anselin *et al.* 2006).

Common Software Architectures for Geographic and Information Visualization Environments

Information visualization and EDA strategies rely increasingly on use of two or more software applications to achieve desired functionality. Examples include the linking of *ArcView*® with *XGobi* (Cook *et al.* 1997), *Snap-together Visualization* (North *et al.* 2000), and *Orca* (Sutherland *et al.* 2000). To be fully effective, these systems need to support cross-application object selection with linking, and a consistent appearance, as well as

linking of statistical attributes (Unwin 1999). This means, for example, that it should be possible to examine data in a map, a parallel coordinate plot, and a spreadsheet, while linking important visualization behaviors across program components, even when these components were developed independently.

Linking and brushing (Joining different views on a set of information by visually highlighting the same objects on different views) can be traced back to (Fisherkeller *et al.* 1974) and (Newton 1978). Initial applications to geospatial data were proposed by (Carr *et al.* 1987) and (Monmonier 1989) and applied subsequently in many applications, e.g., (Cook *et al.* 1996, Dykes 1997, Haug *et al.* 1997, Andrienko *et al.* 1999, Andrienko *et al.* 2007). These studies provide evidence that coordinated views for information visualization and query (focused on standard linking and brushing) are effective tools for access and analysis of complex information (Edsall *et al.* 2001).

Strategies for visual coordination across applications (as well as views) beyond linking and brushing have been implemented using a *pipeline* metaphor in several scientific visualization packages (e.g. AVS (Advanced Visual Systems)(Homer *et al.* 1994), IBM Data Explorer (Abram *et al.* 1996)). Scientific visualization environments using the pipeline-based visual programming approach have been applied frequently to exploration of geospatial data (Treinish 1995, Wood *et al.* 1997, Gahegan 1998, Masters *et al.* 2000). Within scientific visualization, the focus of work on coordination has emphasized the development of multi-user environments for collaborative work (Brodlie *et al.* 1998, Watson 2001). A related project, which extends the notion of object-orientation into “actor-oriented” programming, is Ptolemy (Liu *et al.* 2003).

Overall, our work is motivated by the need for a system which can bring together novel geovisualization and spatial analysis methods in a usable manner. The GVT architecture presented below allows analysts to easily add and subtract components from their analysis environments, while allowing software developers to easily create or adapt new components.

The “Introspective Observer” Coordination Design of the GeoViz Toolkit

This section describes our approach for extending the potential for coordination among visual and computational components in a geovisualization environment. As noted above, the approach builds on earlier work implemented in the Java-based visual programming environment of GeoVISTA *Studio* (Gahegan *et al.* 2002). We begin by describing the basic features of Introspective Observer Coordination. Next, we describe the coordination mechanism that supports application construction in GeoVISTA *Studio* (the StudioCoordinator) as a basis for comparison against our current work to develop an improved coordination mechanism for the GeoViz Toolkit. Next, we describe how the *Introspective Observer Coordinator* was implemented in the GeoViz Toolkit (the GvtCoordinator). This is followed by an examination of how these two types of coordinators perform the crucial task of inter-component registration. Finally, we describe the advantages (and potential disadvantages) of the GvtCoordinator., compared with the StudioCoordinator.

Introspective Observer Coordination

Introspective Observer Coordination connects components using *introspection* to determine at runtime which interfaces and methods each component implements. This information is used to see if the objects could conform to a particular *design pattern* (the *Observer* pattern). Then, *reflective invocation* is used to connect the components appropriately. We explain these terms (introspection, design patterns, the Observer design pattern, and reflective invocation) below, since they may not be familiar to all readers, and since the concept of *Introspective Observer Coordination* is a natural outgrowth of their combination.

Introspection is the ability to discover information about objects at run-time. For example, we could ask of any class what its type is, and what the names and types of its properties and methods are. The ability to conduct introspection on all objects is a distinguishing feature of modern object-oriented languages like Java and C#. Introspection has been used in many contexts, such as grid computing (Baduel *et al.* 2007), automatic software testing (Simons 2007), and computer security (Kassab *et al.* 1998).

Design patterns describe commonly used combinations of objects, and form a common vocabulary of programming constructs in an object-oriented context. The most popular definition of design patterns is given in the seminal book by Gamma et. al: "A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems." (Gamma *et al.* 1995).Perhaps surprisingly, there are few descriptions in academic literature of using introspection with design patterns (for one exception, see (Neumann *et al.* 2002)).

The *Observer design pattern* defines a one-to-many dependency between objects so that when one object changes state, many objects receive notification and are updated automatically. It relies on *Subject* (the one) and *Observer* (the many) interfaces and concrete classes. The exact relationship and communication mechanism specified by the Observer pattern is described later in this paper.

Reflective invocation is an extension of the introspection process described above. It means that automatically discovered fields, methods, and constructors can be programmatically invoked. In the case of the Introspective Observer Coordination architecture, it means that the methods conforming to the Observer pattern which were discovered during introspection are invoked to connect the components.

The Introspective Observer Coordination architecture we have designed and implemented for the GeoViz Toolkit helps support the following user-centered goals for analytical geovisualization:

1. End-users should be able to make their own application component sets
2. Geovisualization applications should integrate tightly with spatial analysis methods
3. Geovisualization applications should be easily accessible and sharable between users

A description of how the Introspective Observer Coordination furthers these goals is provided in the "Advances and Advantages of the GeoViz Toolkit" section below. First, we provide a detailed description of what this coordination architecture is.

The GeoViz Toolkit Component-Based Coordinator – GvtCoordinator

The component-based coordinator we have designed for the GeoViz Toolkit coordinates arbitrary types of events between two or more components. It does so by requesting registration (and deregistration) of components based on their class information using *introspection*. In contrast to the object-oriented *Studio* coordinator (described below), the component-based coordinator (the main class for which is the GvtCoordinator) works by registering objects with each other. Events are then sent from object to object without the coordinator interceding in any way. The GvtCoordinator relies on introspection to discover which components should be coordinated and uses reflective invocation to automatically connect them. It relies on the *observer* software design pattern to know which components are listeners, and which are broadcasters. The Java programming language makes available extensive and automatically derived class metadata available for any object. This enables meta-programming based on class metadata. For example, we can ask any Java object to

report which class defines it. For any class, we can ask what methods, fields, and constructors that it has. In addition, classes can report the interfaces they implement. These automatic reporting mechanisms make powerful meta-programming approaches possible, such as the coordination explained here, the deployment of programs as web services, as well as incorporation of tools in many other programming contexts (Cazzola *et al.* 2002).

The Introspective Observer pattern described here brings the advantages of Inversion of Control (IoC) (Fowler 2004) to multi-component user interfaces. While in procedural computing, program flow is controlled by a central piece of code, with a central controller invoking operations on components, using Inversion of Control, control is dispersed to those components. IoC allows each part of the system can focus on its particular job. The observer pattern itself enables IoC, while the introspection and reflective invocation parts of the pattern enable the many couplings necessary for a large set of coordinated components.

Using class metadata queries, whenever an object (for example, geoMapOne) is registered to an instance of GvtCoordinator, the coordinator examines geoMapOne's class, GeoMap, for methods that can add and remove references to interfaces, for example, the SelectionListener. Then, all previously added components are examined to see if they implement that interface. If any previously added components do implement this interface, then Method.invoke() is called using the appropriate object identity reference and arguments. In this case, the method is addSelectionListener(SelectionListener l) declared by the class GeoMap, using geoMapOne and the previously added component which implements SelectionListener as arguments. The presence of these methods indicates that these components are following the *observer* software design pattern. After this is done, any selections created and passed on by geoMapOne will be passed on to the listening object. Similarly, GeoMap will be queried to see if it implements any interfaces for which previously-added components have listeners, and geoMapOne will be registered with them if any are found. Therefore, events such as changes in selections or color choices will be received by geoMapOne with no user intervention. This registration process is illustrated in Figure 2, using a generic "Java Bean" nomenclature.

The mechanism outlined above is more complicated than the process implemented in *Studio* where each component is simply registered with the coordinator. There is a payoff for this slightly more difficult process of registration, however. When events are sent, they are sent directly to the receiving component, with no intercession by the coordinator. In fact, the coordinator could be removed after registration has occurred, and the components would function just the same. In a typical analysis session, registration of each component occurs only once, while hundreds or thousands of events may be coordinated between components. It makes sense to have the coordinator do a little extra work on set-up to save effort while the program is running.

The Studio Object-Oriented Coordinator – StudioCoordinator

GeoVISTA *Studio* (Gahegan *et al.* 2002) allows fine-grained control and almost limitless flexibility in how components (implemented as discrete JavaBeans) can be connected, and therefore coordinated. This begs the question as to why there is the need for an improved coordinator for the GeoViz Toolkit. One reason is to reduce the burden on the application designer. If we have 15 components in a design, and each one can coordinate in 6 ways, then we will need 15 * 6 * 2 = 180 links for bi-directional association. Considering that in *Studio* one must use a GUI and make connection choices upon making each link, the process is tedious and time consuming. It is desirable to develop a coordinating component that does some of the wiring automatically. In general a coordinator simplifies an application design, even one with only a few components. Since the number of connections expands

exponentially with the number of coordinated components, this advantage can be substantial for designs with many components.

The basic design of the object-oriented coordinator used in *Studio* is that of a multi-caster; coordination events are fired by objects being coordinated, intercepted by the coordinator, and then sent by the coordinator to all listening objects. A new component is simply added to a list of components that are being coordinated. This process is illustrated in Figure 3. One negative consequence of using one master list for all coordination is that as this list grows, it needs to be traversed an increasing number of times as the number of components and types of coordinated events grows. For example, the sequence that occurs when a user initiates a coordinated interaction between two components, a GeoMap and a Scatterplot, is as follows. First, the two components are added to the coordinator. Next, the Scatterplot receives an interaction from the user. It passes this information to the coordinator, and the coordinator fires an event to the GeoMap. Note the “busy” nature of the coordinator during this sequence: all coordination events are sent by the coordinator. This process is illustrated in Figure 3, which contrasts the two coordination mechanisms operations during program use.

Comparing Registration Processes Between the StudioCoordinator and the GvtCoordinator

The advantage of the component-based approach is shown in Figure 3 which shows the sequence of a selection change propagated from the GeoMap to the Scatterplot. The user initiates the change, and the Scatterplot receives it, because it was previously registered with the GeoMap. Figure 3 also shows the more elaborate process that occurs when the same sequence happens using the object-oriented StudioCoordinator. The selection change is initiated by the user, and the GeoMap informs the StudioCoordinator that the selection has occurred. The StudioCoordinator then rebroadcasts the event to the Scatterplot.

We have placed a few restrictions on the component-based coordination process to improve its functionality. First, objects are not registered with themselves. Secondly, interfaces defined in Java *packages* (grouped classes in a common namespace) that primarily concern within-object communication, rather than between-object communication, are excluded. (*Packages* are groups of related classes such as geovista.geoviz.map or javax.swing.) The packages currently excluded for this reason are java.awt.event and javax.swing.event. If these packages are not excluded, then generic low-level mouse clicks and window resizing events, for example, would be duplicated across objects, which is (usually) undesirable behavior. Thirdly, the GvtCoordinator restricts itself to discovering public methods only. If this restriction is removed, then the GvtCoordinator violates Java’s security rules for applets (web-delivered applications running inside other programs). With this restriction, GvtCoordinator can operate in applets with no security violations.

Advantages of the Component-Based GvtCoordinator over the Object-Oriented StudioCoordinator

The component-based GvtCoordinator has four software architectural advantages over the object-oriented StudioCoordinator, listed here in order of importance.

1. Stable Coordination
2. Stable Clients
3. Runtime Control
4. Granular Control

Each of these four advantages is explained with the aid of UML diagrams. Each advantage is explained, and the general relationship to component-based coordination techniques is identified. We also describe how these software architecture advantages enable visual-analytical advances in the GeoViz Toolkit.

Stable Coordination

The source code of the GvtCoordinator does not require modification to be extended to a new type of event. In the object-oriented design of StudioCoordinator, if a new type of coordination is desired, the source code for the coordinator needs to be changed, and the class recompiled and redistributed. The GvtCoordinator, by contrast, can work with future types of coordination that have not been thought of yet, and the GvtCoordinator can work with them without modification or even re-compilation of the coordinator, as long as the future coordination types are presented to the coordinator using the same code-naming patterns. The GvtCoordinator can do this via the automatic introspection mechanisms mentioned earlier. We can call this the advantage of *stable coordination*.

The StudioCoordinator has eight event types, and eight corresponding listener lists. If a non-programmer end- user wanted to add an additional event type to this list, she/he would be unable to. A programmer would be able to add an event type by modifying the source code, which makes application extension a non-trivial task. The GvtCoordinator has only two member variables, a list of firing components, and a list of listening ones. Any new types of events are automatically discovered and coordinated as long as there are sending and receiving components for that event type – no modification to source code is required.

Stable Clients

The *stable client* advantage is that the coordinated objects do not need to be modified if events are modified or extended. In the object-oriented design, if a new type of event were used, clients would not be able to handle them without their source code being modified. Because events are passed directly from client to client using the component-based GvtCoordinator, the events themselves can be extended or otherwise modified without changing old clients or the coordinator. For example we could change the selection event to extend what a selection means from a single subset to multiple subsets with attached authors (i.e., to support multi-user interaction with a display). As another example, we could change the indication event, which is a transient selection of just one observation, to include all items in the class of the indicated observation. In both cases, neither the old clients nor the coordination manager would need to be modified. We call this the advantage of *stable clients*.

This advantage is a function of *stable coordination*, and stems from similar design considerations. Classes that are coordinated by the StudioCoordinator have a dependency on StudioCoordinatorClientListener, which specifies which events to listen for. Classes that are coordinated by GvtCoordinator also have dependencies, however these dependencies are not directly tied to the coordinator. The clients of StudioCoordinator, in addition to the coordinator itself, depend on the interface StudioCoordinatorClientListener. In order to add a new event type to those being coordinated this interface must be extended, and the source code of all clients changed. If we hypothesize that we would like our components to start coordinating conditioning, which enables filtering by a variable, then those clients would need to have their source code modified accordingly. However, the other coordination clients would remain unaffected. In our current toolset, we have sixteen coordinated components. It would negatively impact program stability to change all of them to add a single type of coordination to a single client, which would be required if we were using the StudioCoordinator.

Both advantages of stable clients and that of a stable coordination derive their stability from limiting dependencies. Limiting dependencies helps ensure application stability. Coordination must be performed in a way that does not result in errors that impact effective user interactions. The more classes the coordinator class depends on, the more likely it is that the coordinator will break unexpectedly if these classes change.

Limited dependencies make the GvtCoordinator more robust than the object-oriented StudioCoordinator. The GvtCoordinator has dependencies on only two other classes besides the core Java libraries. Both of these classes are in the same package (namespace) as the GvtCoordinator. The object-oriented StudioCoordinator design depends on nine custom classes and implements a custom interface.

Runtime control

The GvtCoordinator enables end users to modify how components coordinate when users interact with an application in the GeoViz Toolkit. The StudioCoordinator does not allow the user any such control without rebuilding the application in the design mode of GeoVISTA *Studio*. We can call this the advantage of *runtime control*.

This advantage reflects one of the primary goals we had in developing the component-based coordination design for the GVT. With *Studio*, application *end-users* do not use or see the visual programming environment that *Studio* provides for application *designers*. But this means that the application end-user has no way of controlling how components interact at runtime, because those mechanisms are only available on the visual programming side of the *Studio* environment. The GvtCoordinator overcomes this limitation and allows application end-users to redesign their toolkits on-the-fly.

Granular control

The GvtCoordinator in the GVT allows a fine level of control over how components interoperate. In the StudioCoordinator, a component connected to the StudioCoordinator would send and receive all the possible coordinated types of event. The GvtCoordinator allows individual components to send or not send any combination of potentially coordinated events. We call this the advantage of *granular control*.

A graphical user interface supports *granular control*. This interface allows users to to connect and disconnect listeners from each other using the GvtCoordinatorGUI. This allows a user to share some types of events, while excluding others. The StudioCoordinator does not have this capability, because the linkages between components are defined at compile time (object-oriented), not run-time (component-based). Granular control is offered by some other coordination architectures, but they do not as easily allow third-part components to participate, partially or wholly, in this advantage.

Figure 4 illustrates how granular control can be useful. In this design, there are two instances of the GeoMap. The two maps are coordinated in their color selection, but not in their data sources. This allows the user to determine whether patterns evident at one scale are evident at another. Here, using a bivariate color scheme, we can see that the general patterns of counties who voted proportionately more heavily for Obama also tended to be those states with high populations but low areas.

Limitations of Introspective Observer Coordination

There are some limitations of the component-based GvtCoordinator that deserve attention. Two particularly important limitations are discussed here. First, there is a tendency to pass references to objects as fields of events, which creates shared references to the objects,

which can break encapsulation if internal representations are exposed. Second, components may become “over-coordinated” if components are broadcasting events with similar effects.

The first limitation stems from the fact that the easiest way to include information in an event is to add a field that provides access to it. For example, our current SelectionEvent functions just this way, having a method getSelection() that returns an array of integers, notated int[] The potential problem is that multiple coordinator clients may acquire a reference to the same array. Arrays in Java are mutable objects, thus any of the coordinator clients can change the array, or assign it a null value, potentially breaking the other coordinator clients. A safer approach would be to impose a rule that events have a method that returns an Iterator, which is an interface that specifies a means for reading the values of an array.

Using an Iterator would allow for read-only access to an underlying array. The Iterator approach would be slower than the unsafe shared array approach, but probably faster than another safe alternative, which would be returning a “defensive copy” of the array (returning a copy of an object, to prevent the original from being modified) whenever the field is accessed.

Another potential limitation of the *Introspective Observer Coordination* approach is that it relies on each component sending and receiving the appropriate kinds of events. The default behavior is to coordinate in all possible ways that the components are capable. This can lead to an “over-wiring” situation, with more connections than are desired. Events that have similar meanings are especially problematic, because if a component subscribes to multiple events with similar meanings, later events will undo or change the effects of the first to arrive, without the user being aware of what has happened. For example, if a component is listening for color arrays as well as classes that contain color picking algorithms, one may interfere with the other. A solution to this is less obvious than to the first problem. One option may be to have types of coordination assigned to particular types of task, and have the coordination change to fit the user’s needs (Gahegan 2003).

Although the approach has the limitations outlined here, we believe the *Introspective Observer Coordination* based coordination strategy is a sound one and it has demonstrated to be fairly robust in practice. Next, we elaborate on its advantages are embodied in working software.

Advances and Advantages of the GeoViz Toolkit

The *Introspective Observer Coordination* strategy described above pays off in the following advantages in the GeoViz Toolkit: End users are able to make their own applications, interactive geovisualizations are tightly integrated with spatial analysis methods, and geovisualizations are sharable between users. In this section, we describe the above advantages of the Introspective Observer Coordination strategy along with some additional advantages of the GeoViz Toolkit that implements it.

The *Introspective Observer Coordination* strategy allows users to assemble a unique collection of visualization and spatial analysis tools in a single, tightly coordinated framework. The GVT advances the state of the art by providing a set of views that can be added to or taken away from the application using menus and mouseclicks. All data and representational coordination happens automatically in real-time, so that users can easily pursue real work. The component-based coordination of the GVT represents a substantial evolution beyond the visual programming environment of GeoVISTA *Studio* because allows application users to work as application designers without knowledge of programming

principles or limitations. Simply put, the software makes reasonable decisions about coordination without requiring user involvement.

The *Introspective Observer Coordination* strategy also allows for the integration of spatial analysis methods. Because the strategy is so general, software developers can work on separate parts of the GeoViz Toolkit environment, and their separate efforts do not conflict with each other. For example, we have integrated four clustering methods from the Proclude (Conley *et al.* 2005) suite of clustering tools into the GeoViz Toolkit. These tools work well together because they share the *Introspective Observer Coordination* design method. This coordination strategy also helps to integrate geovisualization and spatial analysis methods, because it allows long-running processes to remain connected to the graphical user interface without preventing interaction. It does this by relying on events for communication, which facilitates having multiple centers of control. Additionally, the GeoViz Toolkit has explicit support for spatial topology, spatial extents, and projection information built into the event communication framework.

Introspective Observer Coordination also allows end-users to create and work with geovisualizations in an open-source, web-distributable format. By leveraging Java to XML marshalling tools (XStream (Walnes *et al.* 2008) in particular), representations of software components and their relevant states can be turned into XML documents. These documents can then be stored or shared appropriately. Keeping the software pieces as separate components makes what could be an unmanageable problem into a tractable one.

The development methodology for the GeoViz Toolkit also incorporates a number of best practices from open source and commercial software development. These include unit testing, code coverage, and continuous integration (Cheon and Leavens 2001) (Roubtsov 2001). These techniques can be (and should be) used in concert to improve their effectiveness.

Finally, we have incorporated tools for importing data from SEERStat (Surveillance Research Program 2008), as well as other sources. These sources include popular data formats such as Excel spreadsheets and comma-delimited tables. We have also incorporated data export facilities, in the hopes that users can perform basic data handling tasks without having to use a commercial GIS. In the next section we describe how *Introspective Observer Coordination* helps enable geovisual analysis.

Exploring the 2008 U.S. Election With the GeoViz Toolkit

We have identified four advantages of the GvtCoordinator – stable coordination, stable clients, runtime control, and granular control. In the following example, we demonstrate how these advantages relate to a real world application example. A version of the GeoViz Toolkit with the election data included is accessible from http://www.geovista.psu.edu/gvt.

To begin, we compiled a dataset of county results for the 2008 U.S. Presidential Election. A user might wish to explore election results to compare recent results to previous elections and to evaluate possible relationships to socioeconomic and behavioral covariates. The initial task in this scenario could involve merging together multiple existing datasets – something that the GVT handles easily through having registered data sources with the GvtCoordinator. At the interface level, a user simply clicks on the “Load Data” file menu option and can choose datasets in common formats like CSV and XLS to merge with a boundary shapefile. The GVT identifies common keys and merges data sources automatically, using inner-join, left-hand join, and right-hand join semantics from relational databases.

Once the appropriate dataset has been assembled, the GVT lets users add components on the fly to create an application. The advantage of runtime control allows this capability. There is no need to select tools and then launch the application – the process occurs in real time and can involve as many modifications as the user desires. For example, the user may first choose to add a bivariate choropleth map, a scatter plot, and parallel coordinate plot tool (Figure 1). After working with the views for awhile, the user may then decide to remove the parallel coordinate plot and add a table viewer, a histogram tool, and a univariate map (Figure 5). The coordination design of the GVT ensures that there is no interruption to work due to changes in the number or types of components.

Let us assume the user would like to explore patterns in both the bivariate map and the univariate map. The advantage of granular control allows the user to set up different color and classification schemes for each map, while still maintaining highlight and selection coordination (Figure 6). If at some point the user decides to synchronize color and classification schemes between the two maps, they can link them via a context menu.

In this example, the advantages provided by stable coordination and stable clients are less visible, but remain quite important. These advantages ensure that the application is unlikely to crash or behave erratically while in use, assuming that the individual components are stable. Our testing with the default set of components shows that broadcasting selections remain stable over a period of days, with hundreds of thousands of events generated and processed. This architecture also allows for the addition of controls for highlighting behavior, letting the user set the degree of transparency for de-selected items.

Conclusions and Future Work

We suggest that future projects to develop geovisualization tools would benefit from adopting the approach outlined in this paper. This could be done by designing software in components that can be separately instantiated, while exposing their behavioral capabilities by using design patterns. Research and application areas that would be aided by this approach include geocomputation and data mining, recording analysis sessions, and sharing analysis sessions or analysis artifacts.

One research area where this architecture is already showing advantages is in the GeoJabber project, which supports geovisual collaboration (Hardisty 2009). GeoJabber relies on the GeoViz Toolkit for its user interface, and visualization and analytic functions. The coordination system described in this paper allows remote collaborators to participate in a user's analysis session as additional components, without any fundamental changes in the system.

The core advance presented here, of using introspection to search for and leverage commonly used software design patterns, could be extended by using other software design patterns besides the *Observer* pattern. Candidates include the *Strategy* pattern and the *Visitor* pattern. The Strategy pattern would allow for a GUI to be automatically generated to apply alternative algorithms to a common task, such as different clustering methods. The Visitor pattern would allow arbitrary operations to be performed on sets of objects, for example adjusting all color schemes in an application to change the saturation of those color schemes.

There is also a need for future research in methods for overcoming the limitations in the Introspective Observer pattern identified earlier. The two most serious limitations identified were breaking encapsulation and over-wiring. The problem with breaking encapsulation might be overcome with adding automatic indirection or to expose deep copies of the data structures. Over-wiring could be overcome by allowing for different topologies of coordination (Gahegan 2008). The current coordinator could be described as a complete

graph. Limiting the coordination topology to a tree structure might prove useful for tasks which benefit from a data-flow paradigm. A bipartite coordination graph could aid in analysis tasks that have distinct sets of tasks, such as sets of visualization tools on the one hand, and analysis tasks on the other.

A key future means of extending the GeoViz Toolkit will be through the G-EX Portal (Robinson *et al.* 2007). The G-EX Portal is a website intended to allow users to upload and collaborate on geographic analysis sessions. The GVT will leverage the G-EX project in two primary ways: first, GeoViz Toolkit projects will be exported to the G-EX Portal, allowing them to be retrieved and worked on later. Second, analysis artifacts such as screen captures will be published to the G-EX Portal. Conversely, comments created in the G-EX Portal will be integrated into the projects that the GeoViz Toolkit uses.

The GeoViz Toolkit features advantages for application designers and end-users of geovisualizations. For designers, the GVT embodies a robust yet flexible software architecture for coordination that allows novel types of coordination to be added at run-time. For visualization users, that flexibility and robustness translates into a software package that contains leading edge geographic visualizations and analysis tools, yet remains usable without programming expertise.

Acknowledgments

The GeoViz Toolkit relies on the work of numerous developers, designers, documentation creators, and funding sources. Please see http://code.google.com/p/geoviz/wiki/GeoVizToolkitContributors for a current listing.

This research is supported by the National Visualization and Analytics Center, a U.S. Department of Homeland Security program operated by the Pacific Northwest National Laboratory (PNNL). PNNL is a U.S. Department of Energy Office of Science laboratory.

This material is based upon work supported by the National Institutes of Health under Grant # R01 CA95949-01

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

Conley J, Gahegan M, Macgill J. A genetic approach to detecting clusters in point data sets. Geographical Analysis. 2005; 37:286–314.

Andrienko, G.; Andrienko, N. Knowledge-based visualization to support spatial data mining. Advances in Intelligent Data Analysis, Proceedings; Springer-Verlag Berlin; Berlin. 1999. p. 149-160.

Andrienko, G.; Andrienko, N. Intelligent support of visual data analysis in Descartes. Proceedings of the 2nd international symposium on Smart graphics; ACM Press; Hawthorne, New York. 2002. p. 21-26.

Hardisty F. GeoJabber: Enabling Geo-Collaborative Visual Analysis. Cartography and Geographic Information Science. 2009; 36

Chen, J. Visual Inquiry of Spatio-Temporal Multivariate Patterns; IEEE Symposium on Visual Analytics Science and Technology (VAST 2006); Baltimore, MD. 2006.

Robinson AC. A design framework exploratory geovisualization in epidemiology. Information Visualization. 2007a; 6:197–214. [PubMed: 20390052]

Guo D, Chen J, MacEachren AM, Liao K. A Visualization System for Space-Time and Multivariate Patterns (VIS-STAMP). Ieee Transactions on Visualization and Computer Graphics. 2006; 12:1461–1474. [PubMed: 17073369]

Guo D. Coordinating computational and visual approaches for interactive feature selection and multivariate clustering. Information Visualization. 2003; 2:232–246.

Edsall, R. The Dynamic Parallel Coordinate Plot: Visualizing Multivariate Geographic Data; 19th International Cartographic Conference; Ottowa, Cannada. 1999.

Weaver, C. Building Highly-Coordinated Visualizations in Improvise; IEEE Symposium on Information Visualization (InfoVis 2004); 2004. p. 159-166.

Jern, M.; Johansson, S.; Johansson, J.; Franzen, J. The GAV Toolkit for Multiple Linked Views. Proceedings of the Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization; IEEE Computer Society; 2007.

Gahegan M, Takatsuka M, Wheeler M, Hardisty F. Introducing GeoVISTA Studio: an integrated suite of visualization and computational methods for exploration and knowledge construction in geography. Computers, Environment and Urban Systems. 2002; 26:267–292.

Gahegan, M.; Hardisty, F.; Demšar, U.; Takatsuka, M. GeoVISTA *Studio*: Reusability by Design. In: Hall, GB.; Leahy, MG., editors. Open Source Approaches in Spatial Data Handling. Springer: 2008. p. 201-220.

Fekete, J. The InfoVis Toolkit. 2003 [accessed 20 May 2003 2003]. Available online at: http://www.lri.fr/~fekete/InfovisToolkit/

Azevedo J, Beires N, Charpentier F, Farrell M, Johnston D, LeFlour E, Micca G, Militello S, Schroeder K. Multilinguality in voice activated information services: The P502 EURESCOM project. Speech Communication. 2000; 31:369–379.

Heer, J.; Card, SK.; Landay, JA. prefuse: a toolkit for interactive information visualization. Proceedings of the SIGCHI conference on Human factors in computing systems; Portland, Oregon, USA; ACM. 2005.

Rinner C. A geographic visualization approach to multi-criteria evaluation of urban quality of life. International Journal of Geographical Information Science. 2007; 21:907–919.

Robinson, A. Synthesizing Geovisual Analytic Results; IEEE Visual Analytics, Science and Technology Conference Doctoral Colloquium; Sacramento, CA. 2007b.

Rey SJ, Janikas MV. STARS: Space-time analysis of regional systems. Geographical Analysis. 2006; 38:67–86.

Anselin L, Syabri I, Kho Y. GeoDa: An introduction to spatial data analysis. Geographical Analysis. 2006; 38:5–22.

Cook D, Symanzik J, Majure JJ, Cressie N. Dynamic graphics in a GIS: More examples using linked software. Computers & Geosciences. 1997; 23:371–385.

North C, Shneiderman B. Snap-together visualization: can users construct and operate coordinated visualizations? International Journal of Human-Computer Studies. 2000; 53:715–739.

Sutherland P, Rossini A, Lumley T, Lewin-Koh N, Dickerson J, Cox Z, Cook D. Orca: A visualization toolkit for high-dimensional data. Journal of Computational and Graphical Statistics. 2000; 9:509–529.

Unwin A. Requirements for interactive graphics software for exploratory data analysis. Computational Statistics. 1999; 14:7–22.

Fisherkeller, M.; Friedman, J.; Tukey, J. Prim-9: An Interactive Multidimensional Data Display And Analysis System. Stanford, California: Stanford Linear Accelerator Center; 1974.

Newton, C. Graphics: from alpha to omega in data analysis. In: Wang, editor. Proc. Symposium on Graphical Representation of Multivariate Data; Academic Press; 1978. p. 59-92.

Carr DB, Littlefield RJ, Nicholson WL, Littlefield JS. Scatterplot matrix techniques for large n. Journal of the American Statistical Association. 1987; 82:424–436.

Monmonier M. Geographic Brushing - Enhancing Exploratory Analysis of the Scatterplot Matrix. Geographical Analysis. 1989; 21:81–84.

Cook D, Majure JJ, Symanzik J, Cressie N. Dynamic graphics in a GIS: Exploring and analyzing multivariate spatial data using linked software. Computational Statistics. 1996; 11:467–480.

Dykes JA. Exploring spatial data representation with dynamic graphics. Computers & Geosciences. 1997; 23:345–370.

Haug, D.; MacEachren, AM.; Boscoe, FP.; Barnes, D.; Mararra, M.; Polsky, C.; Beedasy, J. GIS/LIS. Cincinnati, OH: 1997. Implementing exploratory spatial data analysis methods for multivariate health statistics; p. 205-213.

Andrienko G, Andrienko N, Jankowski P, Keim D, Kraak MJ, Maceachren A, Wrobel S. Geovisual analytics for spatial decision support: Setting the research agenda. International Journal of Geographical Information Science. 2007; 21:839–857.

Edsall, RM.; MacEachren, AM.; Pickle, L. Case study: design and assessment of an enhanced geographic information system for exploration of multivariate health statistics; Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on; 2001. p. 159-162.

Homer PT, Schlichting RD. A Software Platform for Constructing Scientific Applications from Heterogeneous Resources. Journal of Parallel and Distributed Computing. 1994; 21:301–315.

Abram, G.; Treinish, LA. An Extended Data-Flow Architecture for Data Analysis and Visualization; IBM Visualization Data Explorer Symposium; San Francisco, CA. 1996.

Treinish LA. Visualization of scattered meterological data. IEEE Computer Graphics & Applications. 1995; 15

Wood, J.; Wright, H.; Brodlie, K. IEEE Visualization'97. Phoenix, AZ, USA: 1997. Collaborative Visualization.

Gahegan M. Scatterplots and Scenes: Visualization Techniques for Exploratory Spatial Analysis. Comput. Environ. and Urban Systems. 1998; 22:43–56.

Masters, R.; Edsall, R. Visualization Development Environments. Princeton, New Jersey: 2000. Interaction Tools to Support Knowledge Discovery: A Case Study Using Data Explorer and Tcl/Tk.

Brodlie KW, Duce DA, Gallop JR, Wood JD. Distributed Coorperative Visualization. State of the Art Reports at Eurographics98. 1998:27–50.

Watson, VR. HICSS-34 Minitrack on Collaborative Problem Solving Environments. Maui, Hawaii: 2001. Supporting Scientific Analysis within Collabortive Problem Solving Environments.

Liu J, Eker J, Janneck JW, X L, Lee EA. Actor-Oriented Control System Design: A Responsible Framework Perspective. IEEE Transactions on Control System Technology. 2003

Baduel L, Baude F, Caromel D. Asynchronous typed object groups for grid programming. International Journal of Parallel Programming. 2007; 35:573–614.

Simons AJH. JWalk: a tool for lazy, systematic testing of java classes by design introspection and user interaction. Automated Software Engineering. 2007; 14:369–418.

Kassab LL, Greenwald SJ. Towards formalizing the Java security architecture of JDK 1.2. Computer Security - Esorics 98. 1998; 1485:191–207.

Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. 1995

Neumann, G.; Zdun, U. Pattern-based design and implementation of an XML and RDF parser and interpreter: A case study. In: Magnusson, B., editor. Ecoop 2002 - Object-Oriented Programming. 2002. p. 392-414.

Cazzola, W.; Ghoneim, A.; Saake, G. Reflective analysis and design for adapting object run-time behavior. Object-Oriented Information Systems, Proceedings; SPRINGER-VERLAG BERLIN; Berlin. 2002. p. 242-254.

Fowler, M. Inversion of Control Containers and the Dependency Injection pattern. 2004. Available online at: http://martinfowler.com/articles/injection.html2009

Gahegan M. Personal Communication. 2003

Walnes, J.; Schaible, J. XStream. 2008 [accessed June 10 2009]. Available online at: http://xstream.codehaus.org/

Surveillance Research Program N.C.I. SEER*Stat software. 2008. seer.cancer.gov/seerstat6.4.4

Gahegan, M. Coordination Typologies for Geographic Visualization and Analysis. Hardisty, F., editor. University Park, PA; 2008.

Robinson, A.; Koua, E.; Hardisty, F.; MacEachren, AM. ICA Commission on Visualization and Virtual Environments Workshop 'From Geovisualization Toward Geovisual Analytics'. Helsiki, Finland: 2007. The G-EX Portal: Web-based Dissemination of Geovisual Analytic Results.

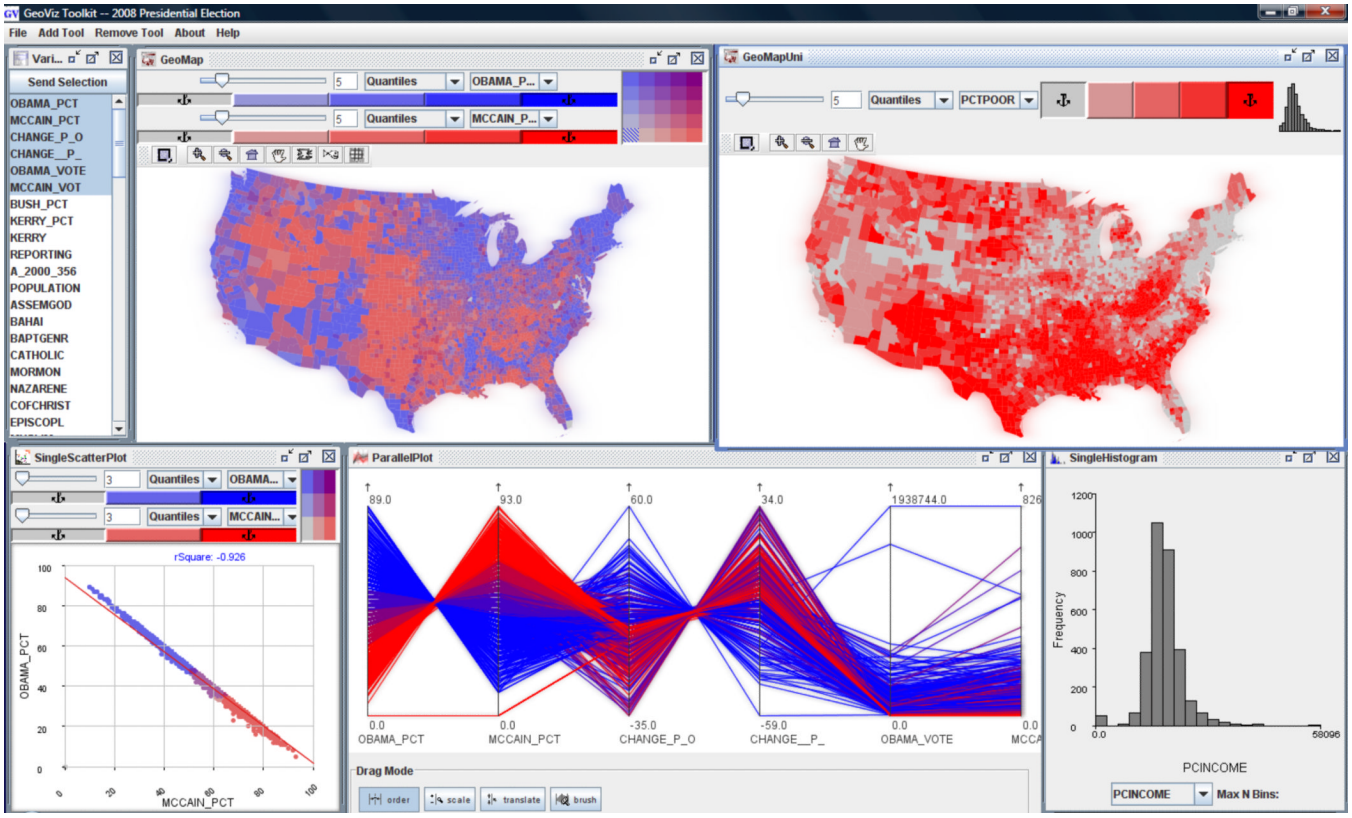

Figure 1.

A bivariate map, univariate map, scatterplot, parallel coordinate plot, and hisotgram showing 2008 Presidential Election Results in the GVT. The scatterplot and bivariate map show the percentage of votes for Obama and McCain (blue and red axes, respectively). Program available at http://www.geovista.psu.edu/gvt, source code available from http://code.google.com/p/geoviz/.

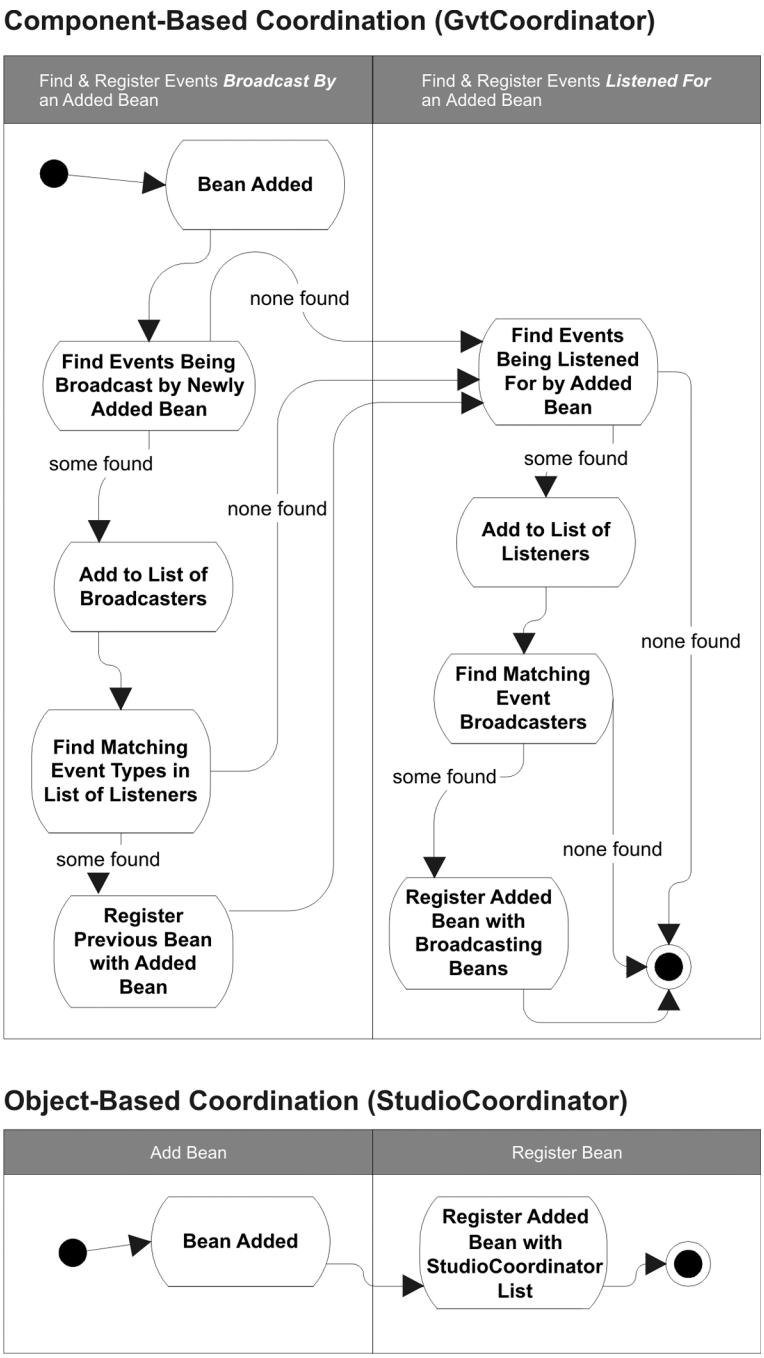

Figure 2.
UML activity diagram comparing the processes of adding a component to the GeoViz Toolkit component-based coordinator (GvtCoordinator, top) and the Studio object-based coordinator (StudioCoordinator, bottom)

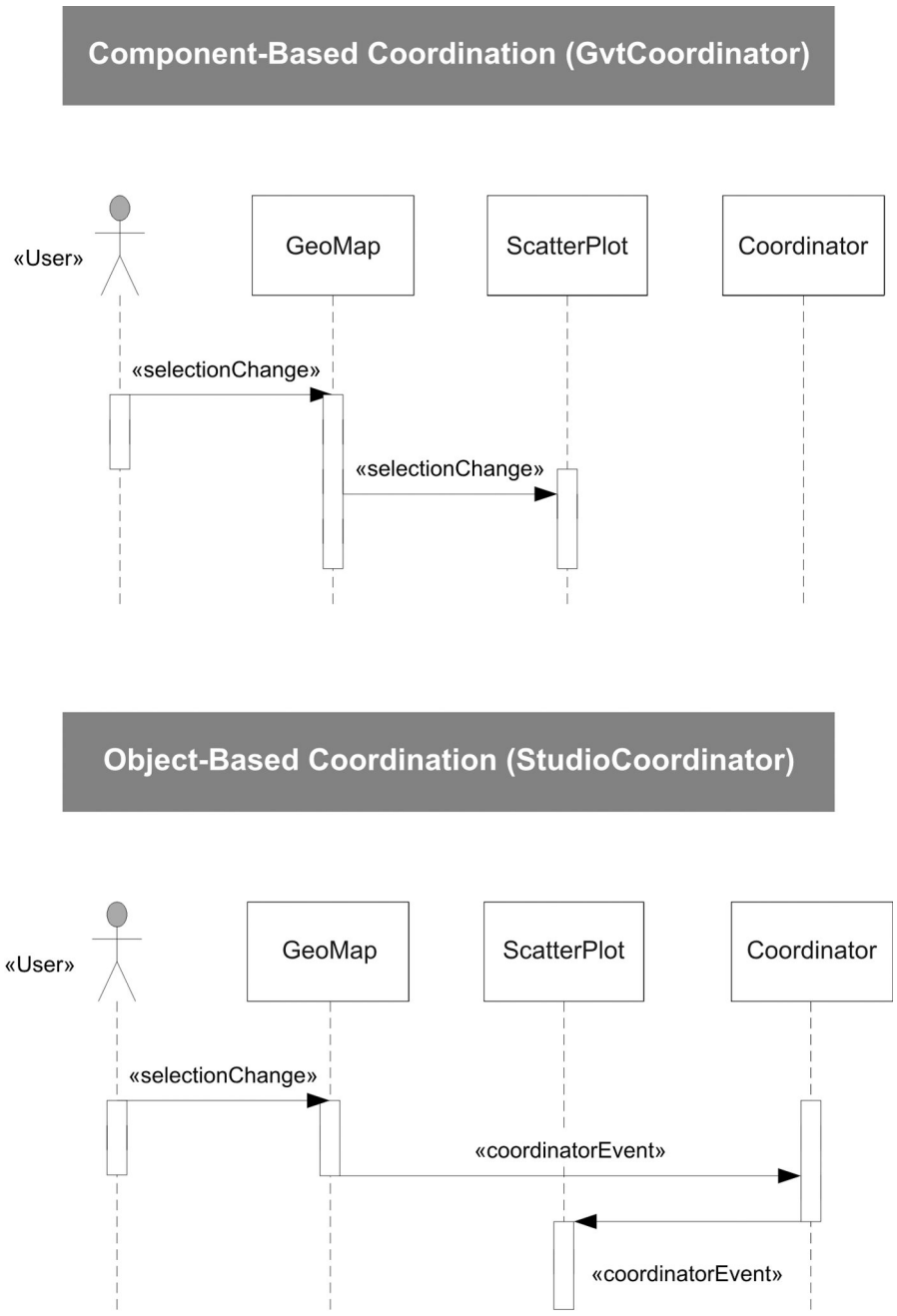

Figure 3.
UML Sequence Diagram showing event firing during application use, between the GeoViz Toolkit component-based coordinator (GvtCoordinator, top) and the Studio object-based coordinator (StudioCoordinator, bottom). This figure illustrates the run-time advantages of using introspection and reflective invocation.

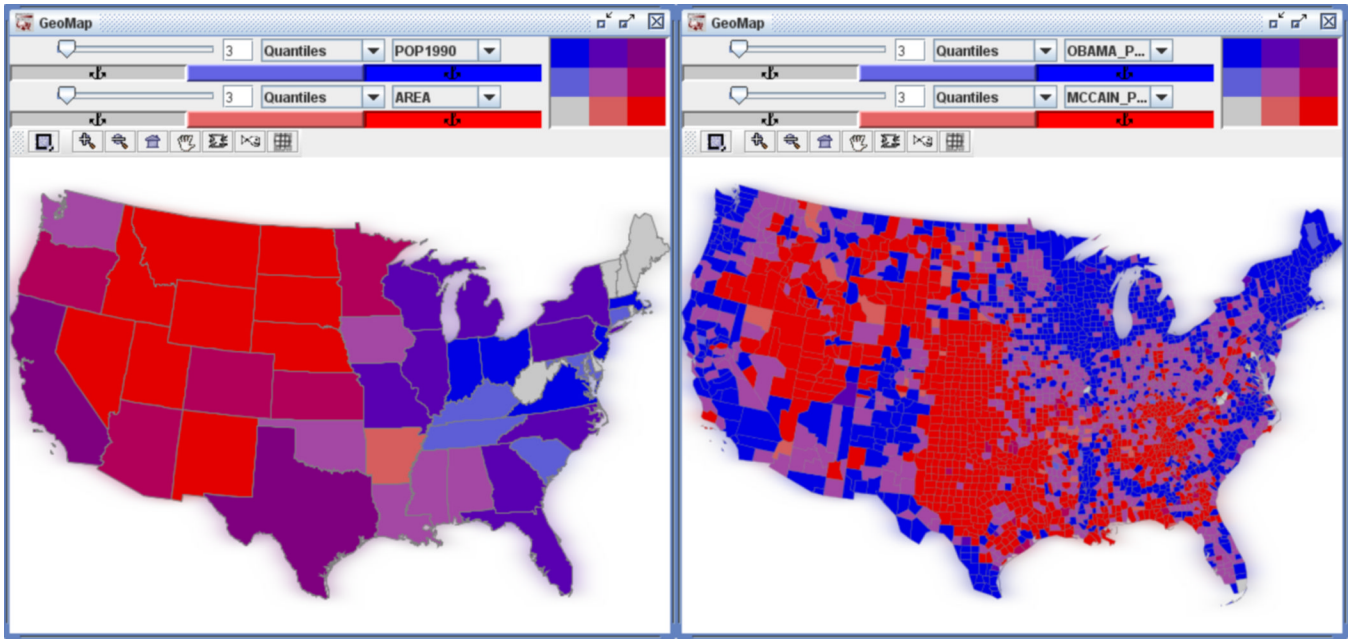

Figure 4.
Coordination between two GeoMap instances. The color scheme is coordinated between the two maps, but the data are not coordinated. This figure illustrates how viewing data at multiple scales can be coordinated and illustrate each scale.

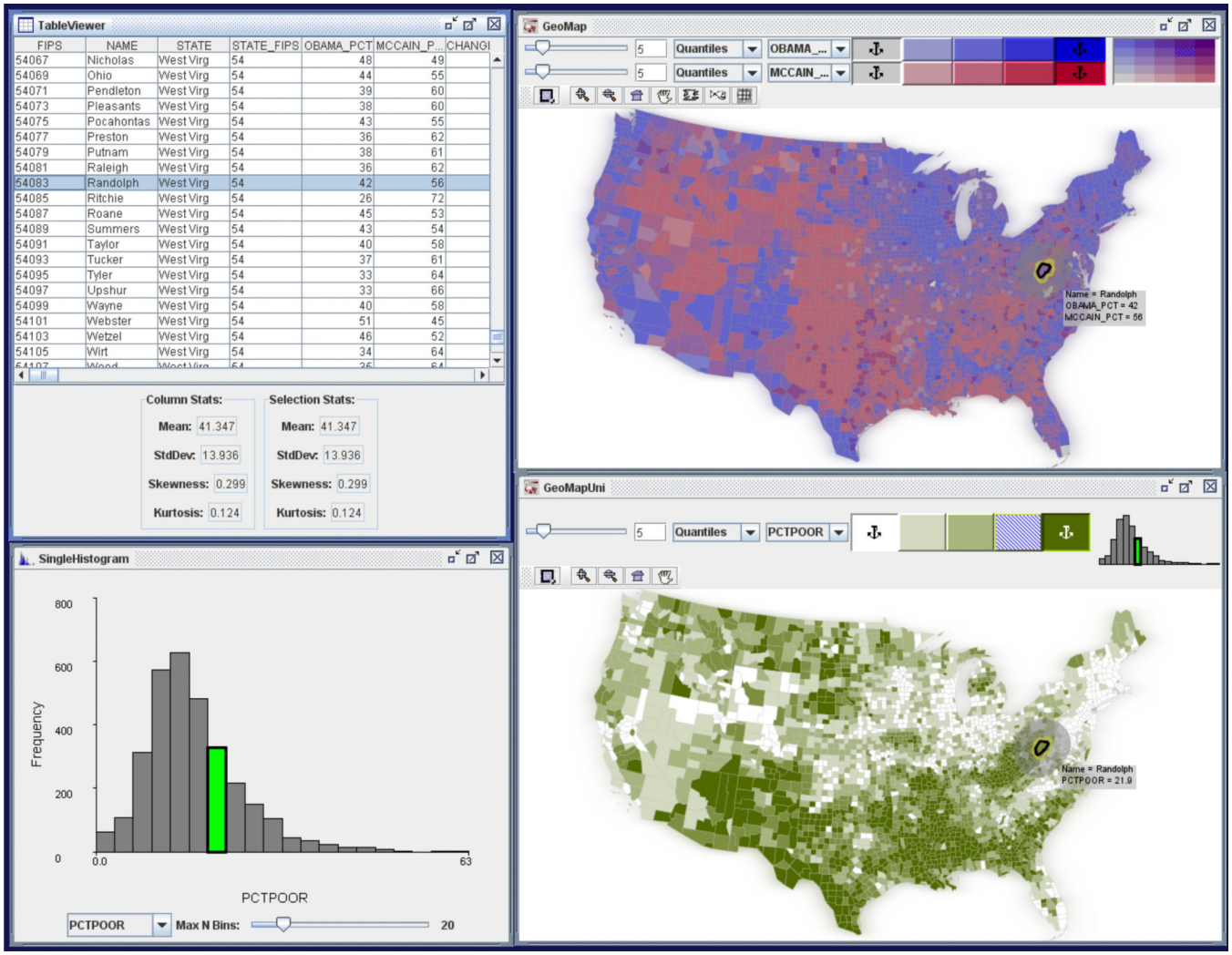

Figure 5.

The GVT has been reconfigured on-the-fly to remove the parallel coordinate plot and scatterplot, and add a table viewer, histogram tool, and univariate map. The table viewer shows the entire raw dataset. The histogram and univariate map are showing different representations of the percent of people currently living under the poverty line. Randolph County, West Virginia is highlighted in each of the views. It has a high percentage of impoverished residents and McCain won by 14 points.

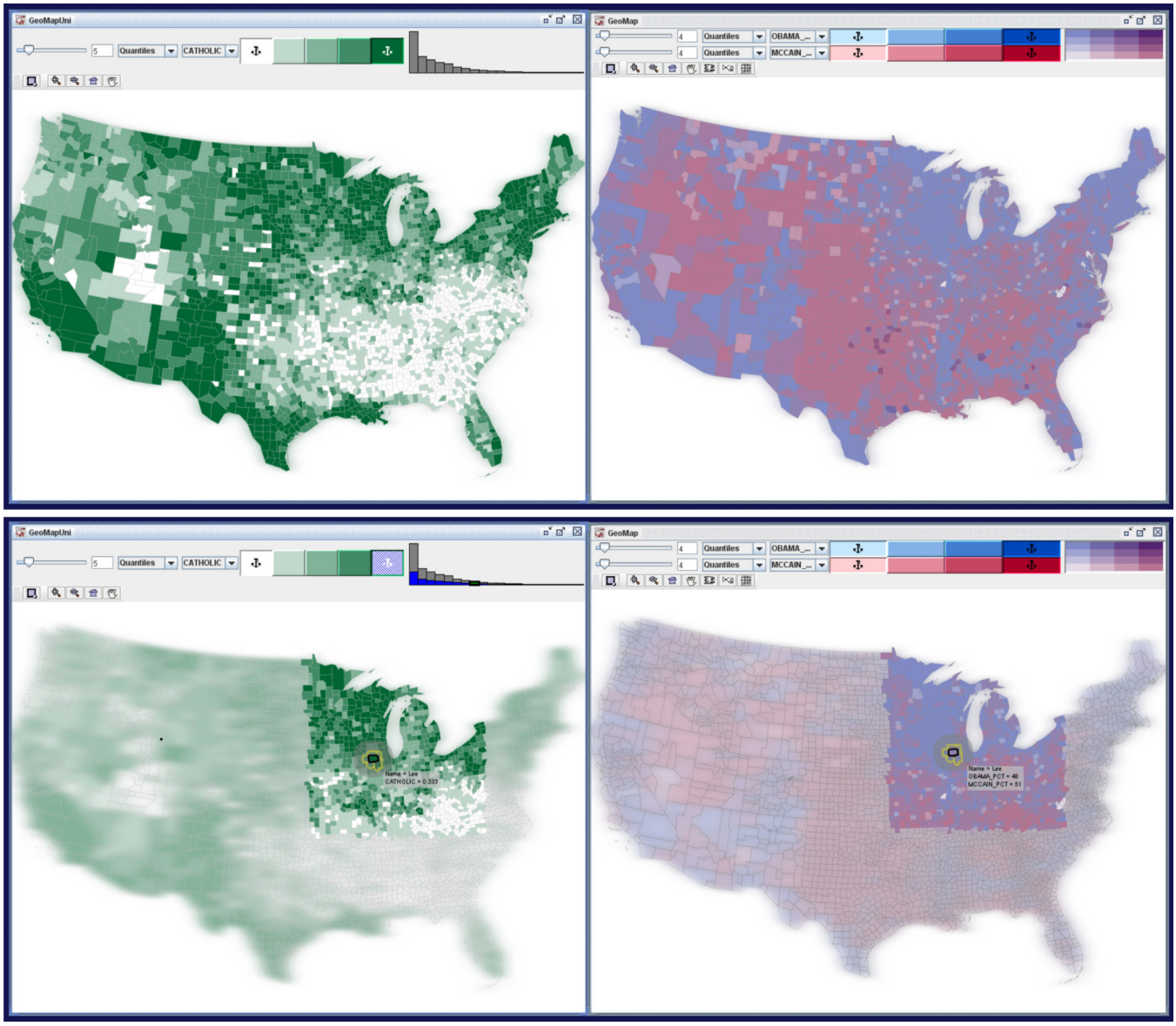

Figure 6.

At top left, a univariate map shows the percentage of Catholics by county. At top right a bivariate map shows the percentage of votes for Obama and McCain. The GVT allows granular control so that users can set up different representations of their data while still allowing coordination between views. At bottom, a selection and highlight has been performed and both maps reflect the change.