

# GGobi: XGobi Redesigned and Extended

Deborah F. Swayne AT&T Labs – Research Florham Park, NJ 07932 dfs@research.att.com	Duncan Temple Lang Bell Laboratories Lucent Technologies Murray Hill, NJ
---	---

Andreas Buja AT&T Labs – Research Florham Park, NJ	Dianne Cook Iowa State University Ames, Iowa
--	--

## Abstract

GGobi is a direct descendant of XGobi, with multiple plotting windows, a color lookup table manager, an XML (Extended Markup Language) file format for data, and other changes. Perhaps the biggest change is that GGobi can be embedded in other software and controlled using an API (Application Programming Interface). This design has been developed and tested in partnership with R. When GGobi is used with R, the result is a full marriage between GGobi's direct manipulation graphical environment and R's familiar extensible environment for statistical data analysis.

## Keywords

Visualization, interaction, graphics, R package, component software.

## 1 Introduction

This paper describes GGobi, an interactive and dynamic software system for data visualization. GGobi is the result of a significant redesign of the older XGobi [Swayne et al., 1992, Swayne et al., 1998] system, which was developed over the past decade or so. GGobi differs from XGobi in many ways, and it is those differences that explain best why we are undertaking a radical redesign. Therefore this paper is largely structured as a comparison of GGobi and XGobi. Here are some of the main points:

- GGobi's appearance: GGobi's appearance has changed in several ways, some of which can be seen in Figure 1: 1) It uses a different graphical toolkit with a more contemporary look and feel and a larger set of components. The new toolkit is called **GTK+**, which explains the initial **G** in GGobi. 2) With XGobi, there is in general a single visible plot per process; to look at multiple views of the same data, one launches multiple processes. In contrast, a single GGobi session can support multiple plots, and a single process can support multiple independent GGobi sessions. 3) While XGobi supported essentially a single scatterplot and a subordinate parallel coordinate plot, GGobi supports

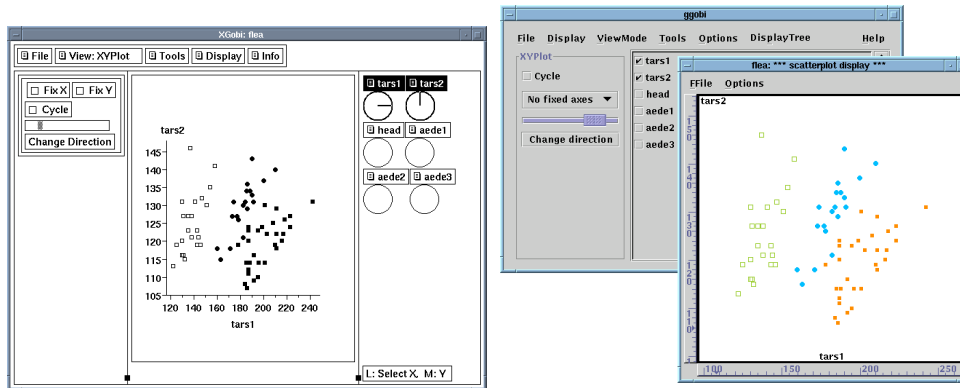


Figure 1: A comparison of XGobi (on the left) and GGobi. The most obvious differences are that GGobi now uses a multi-window design and a new toolkit. The basic elements are still similar.

several types of plots as first class citizens: scatterplots, parallel coordinate plots, scatterplot matrices, and time series plots.

Other changes in GGobi's appearance and repertoire of tools include an interactive color lookup table manager, the ability to add variables on the fly, a new interface for view scaling (panning and zooming), and support for categorical and graph data.

- **Portability:** A major advantage of using the new toolkit (GTK+) is portability. It originates in the Linux<sup>®</sup> community, but it has been ported to Microsoft Windows<sup>™</sup> and is being ported to the Macintosh<sup>®</sup>. It was complicated to run XGobi on a machine running Windows, because it required the installation of an X Window System server. GGobi, on the other hand, runs directly under Windows.
- **GGobi's data format:** GGobi's data format has been extended significantly from that of XGobi. To describe a set of data for the older XGobi, one creates a set of files with a common base name, with the data in one file, and other files for the labels, colors, glyphs, and so on. GGobi continues to support this scheme in a limited way, but its new format uses a single file in XML<sup>™</sup>, the Extensible Markup Language, which is emerging as a standard language for specifying structured documents and data formats.

The use of a single file aids consistency of the different elements of the input, making it easier to validate and maintain. An XML document looks similar to an HTML document, but it allows one to introduce new markup elements. The use of XML in GGobi allows complex characteristics and relationships in data to be specified. For example, multiple datasets can be entered in a single XML file, and specifications can be included for linking them. Using the XML software, GGobi can read compressed files and can read files over the network.

---

*Linux* is a registered trademark of Linus Torvalds.

*Microsoft Windows* is a trademark of Microsoft, Inc.

*Macintosh* is a registered trademark of Apple Computer, Inc.

- Embedding GGobi in other software: GGobi’s ability to interface with other software has become much richer than XGobi’s. The designers of XGobi had always intended that it be used in conjunction with some analytical software, but XGobi offers very limited support for such use. GGobi, however, can be treated as a *C* library and directly embedded in (or linked with) other software, then controlled using an API (Application Programming Interface). This allows GGobi’s functionality to be integrated into one’s own stand-alone application and also provided as an add-on to existing languages and scripting environments. For example, it can be controlled from the R command line or from a Perl or Python script.
- Extending GGobi with plugins: Another way to extend GGobi is by developing “plugins”. The plugin mechanism allows the authors and other developers to provide optional, add-on extensions. These might introduce new plot types, new ways to read data (such as databases and files with special formats), or auxiliary tools to view and manipulate data. This facility, in combination with the GGobi API, makes it feasible to create customized versions of GGobi for different audiences and data analysis contexts.

The rest of the paper is structured as follows. Section 2 introduces XML and describes GGobi’s data format. Section 3 briefly describes GGobi’s graphical user interface (GUI), and then discusses in more detail some of the differences between GGobi’s GUI and data management and those of XGobi. Section 4 describes the ability to use GGobi as a library, embedded in other software and controlled using an application programming interface (API).

## 2 GGobi’s data format

The XGobi input format uses a set of files with a common base name and different extensions to specify data, labels, point colors and glyphs, edges, etc. It has the advantage that each file is easy to describe and process, with line *i* in one file corresponding to line *i* in several other files. This simple scheme has its disadvantages, too: it’s difficult to extend or modify a set of files, and it’s difficult to specify any relationships among files and observations that do not use a simple line-to-line correspondence.

GGobi will continue to support this scheme for basic data input, but the new XML file format is preferred. It makes datasets easier to share with other applications and generally edit and maintain, and allows the specification of more complicated relationships within a single dataset and between datasets. It has the disadvantage that it’s no longer easy to manipulate with UNIX tools. However we expect this to change in the near future with the advent of tools such as XSL (the eXtensible Stylesheet Language), and also we consider this a minor concern when compared with the overall advantages of using XML.

### 2.1 What is XML?

XML, the Extensible Markup Language, is a way to annotate or mark up both content and data. XML looks something like HTML at first glance, but it allows the use of new tags or elements rather than providing only a fixed set of tags. XML allows one to include meta-information (such as a description of how or by whom

the data were recorded), parameters describing the structure of the data (such as the number of records, types of variables) and other auxiliary details along with the data itself. In this respect, XML is like many specialized markup languages. However, the power of XML is that it provides a common infrastructure or scheme within which one can easily define new data formats, simply by introducing new elements and relationships between these elements. In other words, XML is a meta-language since it allows one to define new markup languages. Each of these languages can be easily processed using the same high-level XML parsing tool, but each application can interpret the content in an application-specific manner. Additionally, an application can ignore content it doesn't understand. This makes it easy to extend a data format based on XML without interfering with existing software.

XML also offers the ability to validate the structure of a document, element by element, without having to process it with the software with which it will be read. This greatly minimizes one source of software and data error and separates preparation of inputs from the software itself.

XML is rapidly becoming a standard and already a large collection of tools for processing and generating XML documents are available for all of the common programming languages. This allows us to use the same XML data input within many applications and also significantly reduces the effort of creating software to read specialized file formats. It is becoming increasingly widespread on the Web, and is also used for more traditional data storage and exchange: XML is now used with genetic, geographic, graphical and business data, among others.

## 2.2 XML in GGobi

The use of XML has allowed us to design a system of mark-ups or tags that describe one or more datasets in great detail within a single file, even specifying the relationships between records in different datasets.

We based GGobi's XML format on a pre-existing XML format designed for the Omegahat project ([www.omegahat.org](http://www.omegahat.org)) and S languages (R and S-Plus). Some of the information that can be specified in the GGobi XML file includes:

- Variable types: We use the tags `<realvariable>` and `<categoricalvariable>` to specify a variable's type.
- Variable axis ranges: By providing **min** and **max** attributes within the `<realvariable>` element, a GGobi user can specify the limits to be used for creating variable axes, allowing related variables to be shown on the same scale. (We approximated this functionality using a `.vgroups` file in XGobi.)
- Multiple related datasets: Just as a database usually consists of more than one relational table, a visual data analysis project often consists of multiple datasets. It is convenient (but not required) to be able to store them in a single data file and examine them within a single GGobi session.
- Linking between datasets: In the simplest relationship, a record in one dataset represents the same subject as a record in another, and the datasets have the same number of records. In more complex situations, one dataset may contain data on only a subset of the data in another dataset, or a record in one dataset may correspond to a group of records in another. Linking in GGobi is based either on case identifiers or on the values of categorical variables. The tags for

specifying them in a GGobi XML file are *id* and *categoricalvariable*. Section 3.4 has more details on linking.

- **Graphs:** One of our reasons for writing GGobi is our need for new tools for exploratory visualization of networks and graphs in general. An elementary requirement for graph visualization is (roughly speaking) linking points to lines, where we have variables measured both on the points (people, ethernet addresses, ...) and on the edges (frequency of social contact, number of packets transmitted, ...). We use the *id* tags from one dataset to specify the *source* and *destination* tags in another.

We also use XML for specifying auxiliary information to the GGobi session, including color schemes, and initialization and configuration options.

### 3 GGobi’s graphical user interface

GGobi’s graphical user interface uses a toolkit called GTK+ (see [www.gtk.org](http://www.gtk.org)). It is extensively used by Linux programmers, which ensures a high level of quality and state of the art GUI technology. The toolkit runs not only under dialects of Linux and UNIX<sup>®</sup>, but also under Microsoft Windows. Thus GGobi will run on most popular platforms, which is a great relief to those users who found it difficult to run XGobi in an X Window System emulator under Microsoft Windows.

The new toolkit gives GGobi a cleaner look and feel than XGobi, since XGobi was written using a toolkit whose development was frozen in the early 1990s.

In the following sections we discuss a few GUI-related topics, such as the supported display types and their operations, and the linking model for multiple displays.

#### 3.1 Types of displays and their operations

XGobi’s display types and their operations have been extended in GGobi. XGobi’s two central display types, 1-D views in the form of dot plots and density plots, and 2-D views in the form of scatterplots, are both part of GGobi, but GGobi adds scatterplot matrices and time series plots. While XGobi has a limited parallel coordinate display without interactivity, GGobi has a parallel coordinate display with many of the same interactions that are available in scatterplots, such as color brushing and case identification (labeling).

XGobi’s more distinguishing features, the powerful projection facilities, are available in GGobi as view modes of scatterplot displays: 2-D grand tours, 2×1-D correlation tours. New is a 1-D tour for density plots. The tours feature some novel intuitive GUI controls.

Other data and plot manipulation tools found on XGobi’s “Tools” menu are available in GGobi too: variable transformation, jittering to separate ties as in categorical variables, missing value facilities for imputation and displays for exploring missing value patterns, subsetting by eliminating color and glyph groups, and subsetting by subsampling or selection of contiguous blocks of cases or selection of every *k*’th case. The actions of these tools apply to all display types equally.

---

*UNIX* is a registered trademark of The Open Group.

## 3.2 Multiple displays – controls

The organization of displays has changed drastically from XGobi to GGobi. Roughly speaking, XGobi has one display per process but GGobi permits multiple displays per process, in fact as many as a user desires. When multiple displays are needed, an XGobi user starts up multiple processes, but a GGobi user opens multiple displays from within the same process. This difference in design has broad implications in two areas: display controls and linking multiple displays.

XGobi’s way of controlling multiple displays is simple: every display has its own set of controls. This scheme is simple from a GUI perspective because there is never any confusion about which controls act on which displays; the scheme is expensive, though, in terms of screen real-estate.

GGobi, by comparison, controls multiple displays from a single detached control window or console. This console acts on only one data display at any given time. To make this work, two elements must be provided: 1) a way of visually indicating which data display is currently being controlled, and 2) a way of switching the data display that is being controlled. These issues are addressed as follows: a user clicks on a data display and thereby makes it the one being controlled; at the same time there appears a thick framing rectangle around the border of the selected plot. Part of GGobi training is to develop an awareness of the visual cue that singles out the currently controlled data display, but this is a small learning step.

## 3.3 Linking multiple displays of a single dataset

Linking displays is one of the most powerful paradigms in data visualization. In the simplest but generic example, brushing or labeling cases in one display is immediately reflected in other displays, effecting a form of immediate graphical lookup across multiple attributes, variables, dimensions.

In XGobi, linking requires inter-process communication and active trading of substantial amounts of data between processes. In GGobi, however, linking is implemented with a simpler and more efficient paradigm based on data sharing: all displays have access to the same underlying data, hence linking can be implemented by sharing data regarding color, labeling or other presentation qualities. This scheme covers linking of displays that show the same data.

## 3.4 Linking multiple displays of multiple datasets

XGobi, because of its simple data format, doesn’t know whether two displays are showing the same datasets or not: displays of different datasets are linked if they have the same number of cases, and XGobi assumes that the datasets store the cases in the same order. This initial simple rule for linking later evolved into a rather complicated hodge-podge with special handling of “row groups,” the “nlinkable” notion to exclude points from linking, and linking points to line segments (edges). In GGobi this has been replaced with a single set of rules that derive from a well-defined linking model for multiple datasets.

First, recall from section 2 that GGobi permits multiple datasets to be read from a single XML input file and to be viewed in the same process. It is also possible to add datasets sequentially, either interactively or through the API.

GGobi establishes linking with one of two possible mechanisms: case identifiers or categorical variables. Either can be supplied in the XML input file.

- Linking by case identifiers

The user can supply a case or record identifier for any observation within a dataset. Cases in different datasets will be linked if they share identifiers. This enables, for example, linking between displays of heterogeneous datasets that share only part of the cases, such as all cities of the Northeast in one dataset and the US cities with over 1 million inhabitants in another dataset.

- Linking by the values of a categorical variable

Linking takes place according to the values of the categorical variable: if a case is brushed or labeled in one display, all cases with the same value of the categorical variable will change accordingly, in this and all other displays. For example, a categorical variable may classify cities into two types, “coastal” and “inland”. If a coastal city is pointed at in one display, all coastal cities in this and all other displays will be highlighted. In this linking mode, displays of different datasets can be linked if they have a chosen categorical variable in common.

The major difference between case identifiers and categorical variables is that the former are unique within a dataset, whereas the values of a categorical variable are obviously not.

### 3.5 Some changes in view modes and tools

- The variable manipulation tool

The table in the variable manipulation tool displays summary statistics for each variable, and can be used to specify variable ranges and to add variables.

- Color manager for brushing

GGobi has a much richer notion of color selection than XGobi. Double-clicking on any element of the color palette in the “color & glyph chooser” brings up a color selection widget with access to the full interactive, editable color map.

- Panning and zooming interface for large data

The direct manipulation panning and zooming methods work as they did in XGobi, but we’ve added what we call “Click-style interaction” for more precise control. The new interaction style is used to pan or zoom the plot by a fixed amount, and is especially useful with big datasets, where the response of the plot to direct manipulation scaling can become sluggish.

- Generalized tours

The tour code in GGobi has been completely redesigned, based on the algorithm in Buja et al. (1997) and the object-oriented tour code in Orca [Sutherland et al., 2000]. The most obvious consequence is that there is now a 1D tour which generates a sequence of 1D projections and displays them as average shifted histograms. The two higher-dimensional modes, 2D tour and 2x1D (correlation) tour are still available, albeit with a smaller set of controls. The redesign brings the flexibility to experiment with other types of tours, for example, displaying 4D projections as parallel coordinate plots or scatterplot matrices. It also opens the components of the tour algorithm for access through the GGobi API.

## 4 Extending GGobi with an API

Too many statistical software projects are built from scratch and are not amenable to being used in existing environments. In many cases, much of the development effort is spent on creating incomplete or amateurish imitations of facilities in existing environments. At the other extreme, some projects are very tightly coupled to specific software such as R, S-Plus, Matlab, SAS, and can be used in only one of these environments. Ideally, one would like to create software that embodies concepts and ideas so that the software a) can be used as a stand-alone application, b) is accessible from several of the statistical environments, and c) is accessible from within existing or specialized software such as Excel, or from in-house applications. In general, this is easy to do but requires consideration and planning at the initial stages of the software design.

GGobi has been developed in a way that allows its functionality to be used as a C-level library. The same routines and high-level data structures that are used in the stand-alone GGobi application are available to others who want to include GGobi in their own applications, and are also available as add-ons to languages such as R, Python, Perl, and any others that provide an interface to C code.

While implemented in C, one can think of the GGobi library as providing an object-oriented class for creating GGobi instances, and methods for querying and manipulating each instance's state. The regular GGobi graphical interface uses these methods to implement the basic functionality. Different customized functionality and interfaces can be created by calling these same methods but differently, and creating GGobi instances within other contexts and applications. We have developed an extensive interface between GGobi and R which allows one to use a high level programming language (S) to query and manage GGobi instances. We have also created interfaces to Perl and Python as existence proofs, and plan to use the ability to embed GGobi to provide direct manipulation, high dimensional visualization to Gnumeric, the Gnome spreadsheet.

This approach to the design of GGobi offers many advantages.

- **Benefits for GGobi developers:** From the developers' point of view, one advantage of creating embeddable software components is that experts can focus on what they're best at doing. The graphics experts on the GGobi development team can continue to focus on graphical methods and the user interface, and they don't need to devote any effort to creating yet another specialized, ad hoc scripting language. Instead, GGobi can leverage existing and familiar languages. Additionally, GGobi can avoid many data management issues, random number generation details, etc. and exploit the expertise and well-tested code of others in these areas.
- **Benefits for developers and users of "host" applications:** This approach has other benefits for developers and users of the host applications. Specialized software created as embeddable components can be used in other applications and contexts, with its functionality encapsulated in an API so that details are hidden. Embedding the software in a variety of host applications makes it available to a wide audience via different and specialized interfaces. R users who prefer the command-line interface can exploit much of GGobi's functionality without using its direct manipulation interface. Devotees of the graphical user interface can continue to use GGobi without using any programming language at all.

- **Benefits for GGobi developers and users:** Another advantage (for GGobi developers *and* users) of providing GGobi as a library well as a stand-alone application is that some functions can be omitted from the stand-alone application (protecting the GUI from needless clutter) and still provided in the library. This is especially useful for tasks that are not well suited to a GUI. For example, one feature that is available through the API but not in the GUI is the ability to create mixed or composite displays. In other words, we can create a new display that may contain any number of any kind of plots, and control how those plots are laid out within the window. In R, the function `plotLayout()` allows one to specify a collection of plot descriptions along with the cells in a grid that each should occupy to create a new display. This provides a way to arrange the plots in different configurations that best display the features of interest, enhancing the visual presentation of linking, etc. This is too complex to provide in a simple GUI, and may be of more use to advanced users.

This style of development is reasonably standard in the software engineering and design communities, but has not been adopted widely in the statistical computing world. It has been explored and used to good effect in the Omegahat project and subsequently in R, allowing these environments to be used transparently in many varied settings such as within databases, Web browsers, and spreadsheets. More details can be found in [Chambers, 2000] and [Temple Lang, 2000].

The embeddable approach works very well when the two software projects are a) well established and have a reasonable following and activity community of users and developers, and b) are Open Source, allowing greater flexibility for users to explore different ways to use the software.

## 4.1 Evolution

The designers of XGobi had always intended that it be used in conjunction with some data analysis software. It is instructive to examine previous efforts towards this aim. There exists an S (i.e. R or S-Plus) function, distributed with the XGobi software, that allows an S user to launch an XGobi process given S objects as arguments. That function is embarrassingly simple: the S objects are written out as ASCII files, and a system call executes XGobi with those files as arguments. An XGobi process launched in this way has very limited ability to create S objects directly: after brushing, for example, the vector of point colors can be saved as a file in the S format. The S process has no ability to communicate further with XGobi.

The XGobi authors occasionally explored other approaches that would extend this unsatisfactory relationship [Swayne et al., 1991, Symanzik et al., 1999]. As early as 1991, we used interprocess communication to maintain a live connection between XGobi and S. One of the applications would draw a clustering tree in S, allow the user to click on it to cut the tree and immediately set the point colors in XGobi to show the result. It relied on a second program, also written in C, to gather input from the user and to manage the interprocess communication. It was necessary to assemble each S language command, ship it to S, read back the result and respond accordingly. To make this foolproof, it would have been necessary for XGobi to be fully able to parse S commands and handle errors. Because this was such a daunting task and one that would require continual updates to keep pace with changes to the S language, work on this model was discontinued after the first prototype.

Whenever interprocess communication is used, synchronizing communication between the two processes can be a non-trivial issue. Higher level synchronization – notifying one process of changes made to the data in the other – is outside of the capabilities of these models but is important for a general interface.

The new R-GGobi interface is implemented as a regular R package that one can choose to use at different points in an R session. The result is that there is a single R process and one or more GGobi instances that run within the process. The relationship doesn't need to use interprocess communication, because there's only one process. There's no need to write a GUI to assemble each command in the S language, because the user will supply it directly to R in the customary way (i.e. via the command line). Parsing of input and output between the two systems is also unnecessary as values are passed directly between them with full structural information.

The only complexity from the embedding approach relates to how user events in GGobi are handled by the host application, e.g. R. To handle input for R from the R graphics devices and console and simultaneously recognize events in GGobi, R needs to support multiple input sources. Work to generalize the R event loop has made this possible. However, more work is needed and research into the use of multiple threads within R is under way.

## 4.2 The GGobi API

The GGobi API allows the basic facilities of the stand-alone application to be incorporated into other applications, and to be made available in other programming languages such as R, Perl and Python, C. The routines in the API provide access to the entire GGobi computational model. It allows us to create new GGobi instances and manipulate both the data and the contents of the displays. The routines allow us to both query and set the state of multiple GGobi instances within the same process and also provide custom event handlers for certain high-level interactions.

The routines in the API allow us to create new GGobi instances at any time, specifying data either directly from memory (in the form of a data frame or matrix), or externally from a file or URL. We can access the data from the new GGobi instance and also replace individual cells. We can also add or remove variables from a dataset, edit the variable names, and generally retrieve and modify the data.

We can retrieve and set the visual characteristics of points in the display – the color, glyph type and size of each point – and choose whether a point is displayed or not. We can query the current status of the active brushing region, including its position, color and even the points it encloses. While brushing and point identification are usually done interactively, GGobi's brushing region can be controlled programmatically. This can be appealing when the number of points becomes large, because the brush response begins to lag behind the user's hand. It's also convenient for providing directed animations or “movies”, and also allows one to provide scripts which create a particular view that one wants a user to see.

As mentioned earlier, a GGobi process can have several plot displays open at the same time, and there are several types of displays, such as scatterplots, scatterplot matrices, and parallel coordinates plots. The API provides ways to manipulate the set of displays. For example, we can determine what displays are open, delete a particular display, open a new one, and control which one is active.

### 4.3 Embedding for tighter coupling

The facilities described above allow applications (e.g. R) to communicate with GGobi and both set and query its state. This, in itself, is a significant improvement over the previous approaches. However, it still preserves a separation between the GGobi and the host application, being a one directional interface from R to GGobi. A richer form of embedding allows the two systems to interact in a more dynamic and symbiotic manner. Two examples illustrate this. First, GGobi can use functionality in the internal R API. For example, the random sampling of records provided by GGobi can make use of the random number generation facilities in the R math library. In an experimental version of GGobi, smoothed values can be displayed on a plot. The interface between GGobi and R allows users to provide an S function to perform that smoothing, updating the points on the plot as the user moves the GGobi slider to change the smoothing parameter(s). This allows new smoothing functions to be quickly explored in the direct manipulation world.

More interestingly, we can allow user-level S functions to be specified at any time within the lifetime of a GGobi session to implement particular tasks. For instance, users can specify functions that will respond to a GGobi event. As an example, let's consider GGobi's *identify* mode for interactively labeling points. As the user moves the pointer close to a point on the plot, the label for that case is displayed. The identification of a point corresponds to a high-level GGobi event and users can specify an S function which is to be called when such an event occurs. In this case, the function is given the index of the observation and can use this, for example, to highlight the corresponding row in a data editor/spreadsheet or to display some characteristic of the observation in an R plot. This allows users to easily customize how GGobi responds to a growing number of user interactions using a high-level programming language, S.

Another form of event handling allows one to associate an R function with one of the number keys (i.e. 0, 1, . . . , 9). When that key is pressed while viewing a GGobi plot, the function is called. This can be used for tasks like computing summary statistics or updating a plot based on the current state of the GGobi display.

This type of event-driven, dynamic feedback to the high-level programming language (e.g. R) makes it significantly easier to experiment with and develop interactive graphics tools. It is relatively easy to implement customized linking algorithms and explore the characteristics of different statistical methods with simple user actions. Programming these in the low-level GGobi C code is prohibitively complex. But exposing these top-level events as programmable actions for high-level scripting languages significantly reduces the cost of experimentation.

## 5 Plugins

*Plugins* are a modular mechanism by which developers (both the original authors of GGobi and others) can add extensions and optional features to GGobi without interfering with the existing code base or adding to the complexity of the code and configuration process. In this way, developers can add to the GGobi base without having to understand all details of the the GGobi code itself.

Plugins allow different installations of GGobi to provide different features depending on the goals of the users and the availability of third-party libraries on the machine at that site. The example plugin provides a data grid or spreadsheet-like layout for displaying datasets and their values in a GGobi instance. It uses the `GTK+Extra` library which is not as commonly installed on machines as the standard

GTK+ libraries. If this library is available, one can compile and install the data grid plugin for GGobi. Users will then see an additional menu option in the GGobi control panel and can view and (eventually) edit the data.

Additional plugins that we might explore include: facilities to read data from different sources such as relational database servers, proprietary data formats such as SAS, S-Plus, Matlab, specialized-XML formats (e.g. genomic, geographic data); serializing data to different formats; generating plots in Postscript, PDF (Portable Document Format), SVG (Scalable Vector Graphics, an XML-based graphics representation).

The plugin facility could become “multi-lingual.” It is possible to implement the functionality of the plugin in languages other than C. For example, we might provide graphical interface plugins using Perl’s and Python’ GTK+ bindings, simplifying the development of such plugins. One can also embed the Java virtual machine inside GGobi and provide plugins via arbitrary Java classes.

The plugin system is similar to that used in Gnumeric, the Gnome spreadsheet application. It uses XML to describe and identify the plugin to the GGobi process. One can control which plugins are available by editing the XML file, and one can activate and deactivate the different plugins during the GGobi session via the GGobi GUI.

## 6 Conclusions

This smooth integration of R and GGobi is an example of a modern approach to component-based software inter-operability in statistical computing. No single piece of software has to do everything, but any piece of software can be designed to work well with others. GGobi’s design was greatly changed so that it could be compiled into a library and controlled via an API; R was changed so that it could accommodate GGobi’s event loop along with its own.

This software combination may hold special interest for the teachers of statistical computing courses. If R or S is already on the syllabus, this environment should make it easier to introduce interactive graphics, and it should also be a good platform for student projects.

We also hope that this combination will encourage other software developers to pursue a component-based strategy instead of “reinventing the wheel” whenever they need a bit of new functionality. By relying on existing software, and designing with that software in mind, new tools can be created that are more powerful, more reliable and more useful than if they are created in a vacuum.

The GGobi source code, sample data sets, and documentation can be found on [www.ggobi.org](http://www.ggobi.org).

## References

- [Buja et al., 1997] Buja, A., Cook, D., Asimov, D., and Hurley, C. (1997). Dynamic Projections in High-Dimensional Visualization: Theory and Computational Methods. Technical report, AT&T Labs, Florham Park, NJ.
- [Chambers, 2000] Chambers, J. M. (2000). Users, programmers, and statistical software. *Journal of Computational and Graphical Statistics*, 9(3):404–451.
- [Sutherland et al., 2000] Sutherland, P., Rossini, A., Lumley, T., Lewin-Koh, N., Dickerson, J., Cox, Z., and Cook, D. (2000). Orca: A Visualization Toolkit

for High-Dimensional Data. *Journal of Computational and Graphical Statistics*, 9(3):509–529.

- [Swayne et al., 1991] Swayne, D. F., Buja, A., and Hubbell, N. (1991). XGobi meets S: Integrating software for data analysis. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, pages 430–434, Fairfax Station, VA. Interface Foundation of North America, Inc.
- [Swayne et al., 1992] Swayne, D. F., Cook, D., and Buja, A. (1992). XGobi: Interactive dynamic graphics in the X Window System with a link to S. In *American Statistical Association 1991 Proceedings of the Section on Statistical Graphics*, pages 1–8. American Statistical Association.
- [Swayne et al., 1998] Swayne, D. F., Cook, D., and Buja, A. (1998). XGobi: Interactive dynamic data visualization in the X Window System. *Journal of Computational and Graphical Statistics*, 7(1):113–130.
- [Symanzik et al., 1999] Symanzik, J., Cook, D., Lewin-Koh, N., Majure, J. J., and Megretskaja, I. (1999). Linking ArcView 3.0 and XGobi: Insight Behind the Front End. *Journal of Computational and Graphical Statistics*, 9(3):470–490.
- [Temple Lang, 2000] Temple Lang, D. (2000). The omegahat environment: New possibilities for statistical computing. *Journal of Computational and Graphical Statistics*, 9(3):423–451.