Implementation of Graphing Tools by Direct GUI Composition

Article · April 1998

Source: CiteSeer

CITATIONS
0

READS
38

3 authors, including:

Daniel G. Aliaga
Purdue University
164 PUBLICATIONS **3,484** CITATIONS
SEE PROFILE

Matthias Schneider
Technische Universität München
56 PUBLICATIONS **554** CITATIONS
SEE PROFILE

Implementation of Graphing Tools by Direct GUI Composition

**Daniel G. Aliaga, Martin Brenner,
Matthias Schneider-Hufschmidt**

**Siemens Corporate Research and Development
Otto-Hahn-Ring 6
D-8000 Munchen 83**

**Phone: (+49) 89 636 49505
Fax: (+49) 89 636 48000**

Abstract: we describe an object oriented design and flexible implementation of a set of graphing tools implemented by using only the standard graphical user interface widgets provided by an interactive program composition system developed at Siemens Corporate R&D. The system, SX/Tools, allows for the addition of arbitrary tools (widgets) by direct composition of a set of initial objects. Using this mechanism and without any further system programming we were able to design a powerful set of graphing tools for applications ranging from static displays of data, to business graphs, to dynamically varying data and maintaining interactive rates.

1 Introduction

1.1 The general problem

1.1.1 User Interfaces

As a consequence of the computer boom of the last few decades, computers are being used by a continuously growing number of non-technical users. Specifically, computers have infiltrated and become an integral part of banking, international trade, air traffic control and even everyday home activities. In most cases, the users are well versed in the field for which they use computer technology but are not computer experts. Unfortunately, most software was originally designed for and by computer experts and thus was not easily accessible to bank clerks, financial counselors, etc.

As computer performance improved, it became possible to devote more CPU time to the user interface [1]. *Graphical User Interfaces* (GUI) became the most popular and intuitive, though not the easiest to design [2]. Thus, the field of *Computer-Human Interaction* (CHI) was born.

1.1.2 From CHI to Direct Composition

Among the difficulties found in CHI, was that every user is unique (and also quite stubborn). Somehow users have the magical ability to create situations that have never been thought of before. This made the quest to design simple but powerful user interfaces even more challenging. Graphical interfaces have the potential of being very intuitive to use. Unfortunately, it is not very clear how to bind functionality with a graphical representation.

Novelties such as object oriented programming, menu-based interfaces (the beginning of the Macintosh era) and many others soon appeared. This, together with the growing popularity of windowing systems and the trend to stan-

dardize their API, caused the advent of soon to become standard concepts like pull-down menus, sliders, buttons and, in general, *widgets*. Shortly afterwards, the difficulty of manually programming a graphical user interface for an application became clear. Thus, interface builders appeared which quickly evolved into more complete development systems where you could actually construct or link your application from within the interface builder. Many of these systems supported a C-like scripting language or produced C code that could later be compiled.

1.1.3 Direct Composition

One of the most powerful strategies used by several of the aforementioned systems is *direct composition* [3]. In essence, an interface designer is given an initial set of widgets with which the user interface is to be built. Once the layout is determined, the application designer or programmer links the user interface to the application. The outcome is a fully functional system with a pleasing, though not perfect, user interface. In most cases, the user interface can be modified without having to change the back-end application.

The ease with which the user interface itself can be altered has caused areas like *Adaptive User Interfaces* (AUI) to flourish. The difficulty lies in determining an interface suitable for all users, AUI tries to solve this through self-adaptation, user-controlled adaptation, computer-aided adaptation, among others [4].

Naturally, the interface designers, namely non-computer experts, soon also wanted to take part in the application construction and linkage. This caused the systems to provide more powerful composition methods. The previous interface designers now also had sets of black-box functionality they could manipulate. A simple task, for example requesting information from a database, could be black-boxed into a single widget. Later, it could be dragged into a new interface without having to reprogram. A succession of these operations is all that is needed to create a new application.

1.1.4 Platforms

For computer systems running Unix and the X Windowing System, many widget packages exist (X Toolkit, Athena Toolkit, MOTIF, etc.). To name only a few interface builders, some of which obey a direct composition-like strategy: UIMX (MOTIF) [5], DevGuide (OpenLook) [6], NeXT Interface Builder [7].

We implemented our graphing tools using a platform developed at Siemens Corporate R&D, namely *SX/Tools*, [8] which follows the principle of direct composition.

1.2 Graphing Tools

1.2.1 Graphs in General

It is well known that humans understand facts better when it is presented to them in a visual or graphical manner. From financial activities to scientific explanations, pictures and graphs are a powerful medium to express ideas.

In terms of computer technology, the development of high resolution displays, large colormaps (instead of the previous green and black computer screen) has allowed for very complex computer generated graphs. Indeed graphics has become a very important part of computer software today. The field of computer graphics has grown tremendously; but we will limit our graphs to two dimensional non-photorealism graphs. Namely, we wish to display information (e.g. from a database) or the evolution of parameters overtime and not achieve a photo-realistic representation. We can group these graphs into three categories. This subdivision is not the only one possible, but it seems to be a good categorization:

- *Static Presentation Graphs*: the graphs in this category are very elaborate and are especially tailored for the specific data or concept to be expressed.
- *Partially Dynamic Presentation Graphs*: these graphs may be elaborate, but also have some degree of variability; for example, predictions of the number of company shares sold in the next trimester given a set of initial conditions.
- *Fully Dynamic Graphs*: this category contains the graphs which are constantly undergoing real-time changes. Such dynamic qualities may be used to display the current weather conditions, for engine tuning purposes or to display the status of a complex experiment.

1.2.2 Our Goal

We wish to design and implement a powerful set of graphing tools using only direct composition. Our tools will

hopefully be able to generate graphs for each of the previous categories, though we will emphasize more on the second and third category, since the graphs can always be embellished afterwards. The graphs should also not be limited to one or a few specific application areas and should easily be incorporable into more complex visualizations limited, in fact, only by the imagination of the designer.

Furthermore, we wish to design the tools in such a way as to minimize the amount of work for the application designer; though if we were to strictly follow the minimum-work paradigm, it would lead us to constructing graphs with a very rigid layout despite being totally automated. Hence, we will pursue a design in between both extremes.

Another issue to consider is extensibility. We will not claim our graphing tools (or the platform on which they are built) as being capable of performing all the tasks we will ever imagine. As you will read later, our direct composition platform allows for communication with external partners (or clients). We wish to design our graphs in such a fashion as to allow for easy interaction with such partners; thus, applications can easily access resources external to the platform on which the graphs are built. At the same time, we would like to maintain a clear separation between the partner and the graph configuration, so editing of the graph configuration (i.e. change the graph type) does not require modification of the partner.

Finally, we do not wish to duplicate the functionality already available with the underlying platform. If the direct composition platform already provides extensive modifiable properties for ellipses, providing a method to alter each property of the ellipses of a pie graph would be an instance of unnecessary duplication of functionality. Additionally, if the platform were ever to provide new editable ellipse properties, the graphs would not be able to take advantage of them.

1.2.3 Graph Types

We determined the following three types of graphs to be a good base set:

- *Pie Graph*
- *Bar Graph*
- *Line Graph*

2.0 Implementation

2.1 SX/Tools

As mentioned before, the graphing tools are implemented using the SX/Tools direct composition platform developed in our lab. The first author of this paper was able to construct on his second day using SX/Tools a functional pie graph with a variable number of slices. Soon afterwards, it became clear that a full implementation of a set of graphing tools was possible. After a few weeks, the initial version was complete as well as a set of interesting applications.

SX/Tools provides the user with a *workarea* (initially a blank window) and a selection of initial graphical objects (toolbox). The workarea, once filled with objects, can be saved into a *scene*. All objects possess a list of methods which can be called from a C-like scripting language provided by the platform. Furthermore, event-triggered and user-defined methods can be added to any object. With a click of the mouse, a configurable pop-up menu appears for the currently selected object. Through this menu the object's methods and properties[1] can be altered as well as other standard editing functions.

2.2 Toolboxes

2.2.1 What are Toolboxes?

In SX/Tools, any scene or part of a scene can be compounded into a single object (called a *container* object). The details of how the compounded object is constructed are hidden. A collection of such objects is put into a *toolbox* and essentially added to the set of initial objects available. Thus, through the toolbox mechanism, any number of tools can be added to the platform. In our case, we added graphing tools, but there is no reason why not to add audio and video tools, for example. With a direct composition platform, like SX/Tools, such expansions are trivial.

2.2.2 Graph Toolbox

Figure 4 is how the graph toolbox appears to the designer. Each graph tool has a symbolic icon. The designer selects

1. See Section 2.3 and 2.4 for more information on methods and properties.

the desired tool and drags it onto a workarea. This creates a new instance of the corresponding graph tool.

FIGURE 1. The Graph Toolbox

The tools were constructed by composing objects from the initial set provided by SX/Tools. The following diagram (figure 5) outlines the general structure of each graph tool.

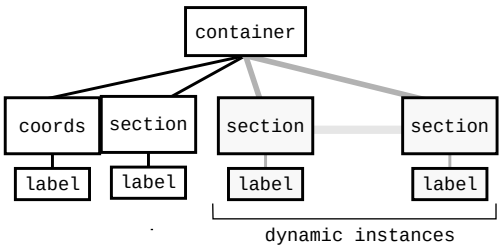

FIGURE 2. Outline of a graph tool object hierarchy

The container object in figure 5 encapsulates and sends messages to the section objects. For pie graphs the section object is an ellipse; for bar graphs it is a rectangle and for line graphs it is a polyline.

2.3 Methods

Once the tool object hierarchies were constructed (a process which is accomplished by a sequence of dragging operations), it was necessary to overload and expand the functionality of each section object. For example, in the case of the bar graphs, rather than having a method for setting rectangle width and height by specifying the number of pixels, a method was needed to set the width according to the number of bars and the height according to the section value, in other words, according to the percentage the bar is to represent. Similarly, when a rectangle belonging to a bar graph is selected with the mouse, the container object should be informed in order to permit operations on the currently selected section to function properly.

The script interface to the graphs, namely the methods, is consistent among the three graph types. With a few minor exceptions, the actual type of a graph does not matter. The number of methods is also not as large as one would expect, namely because many of the esthetical qualities of the graphs can be altered using already existing accessors. The function *getSection* is provided to obtain the section object whose properties can be directly accessed using standard SX/Tools features. In order to make editing of the most commonly changed properties (color and fill style) even simpler, shortcut methods are provided.

Table 1 contains the most important methods common to all graph types. Applications using these methods can interchange graph types at will. Table 2 contains the essential methods for features relevant only to specific graph types[2].

TABLE 1. Essential Common Graph Methods

Method Name	Description
setNumberSections	Create the desired number of sections.
getSection	Accessor for a section object.
getSelectedSection	Accessor for the currently selected object.
getType	Return the graph type.
setSectionValue	Set the value of a specified section.
setSectionLabel	Set the label of a specified section.
setSectionColor	Set the color of a specified section.
setSectionStyle	Set the fill style of a specified section.
setShowValues	Select if section value should be displayed in a text object.
setShowLabels	Select if section label should be displayed in a text object.
setMinMax	Set range of values for each section (a floating point range).

2. See [9] for a complete listing of methods.

TABLE 2. Essential Graph-specific Methods

Method Name	Description
For bar and line graphs:	
setCoordinateStyle	Select type of coordinate axes to use.
setCoordinateDivs	Select the number of divisions per axis.
For pie graphs:	
setFullGraph	Automatically complete the pie circle.
For bar graphs:	
setOrientation	Select the orientation of the bars.
For line graphs:	
setSectionLWidth:	Specify the line width.
setSectionLStyle:	Specify the line style (solid or dashed).

2.4 Links & Property Sheets

The next problem encountered was how would the designer interactively modify the characteristics of a graph. All objects in SX/Tools have a list of properties which can be edited through a generic property editor. Some example properties are: fillForegroundColor, lineWidth, fillStyle, width, height, etc. Some objects may also have more specific properties. The slider object has a minValue, maxValue and actValue property; a radio button column object has a numberItems and value property, etc. The following sections outline the two approaches we pursued.

2.4.1 Link Mechanism

Our first solution was to have additional tools for altering graph properties. The tools were created in a generic fashion such that they could be linked to any graph in order to edit a predetermined property.

For example, to edit the number of sections, we constructed a tool which had a slider linked to the *setNumberSections* method of the graph being edited. Thus, when an instance of a graph is created, an instance of a section number tool can also be created and interactively linked to the graph. Similarly, if the designer wishes to change a section color, an instance of a section color tool can be created and interactively linked to the graph. We implemented three such tools: number of sections tool, color tool and value tool.

2.4.2 Graph Specific Property Editors

After gaining experience and designing a few sample applications, the idea to create an object dependent property editor arose. Furthermore, the property editor itself could be designed with SX/Tools thus giving the object creator total freedom in the design of the property editor and allowing it to be truly object specific.

This is the implementation we decided to follow for our final version. We currently have a property editor for each graph type. The three editors are similar in style, but each one is specially tailored for the graph type it corresponds to. The property editor may be used at any time, either during the design phase or, if so desired, during the execution phase[3].

The property editors provide an intuitive mechanism to create and modify graph properties. The general structure of the property editors is as follows: the upper portion of the property editors is used to specify the number of sections to have in a graph (the number can change at any time). The middle portion is for editing of section properties. A slider is given to easily switch between sections. The values displayed are automatically updated according to the current section number. Additionally, the designer can graphically select (via the mouse) the section to edit and the property editor will be notified. The lower portion of the property editors is for global properties; for example, coordinate systems, minimum and maximum section values, etc.

3. See Section 3 for more information on design and execution phase.

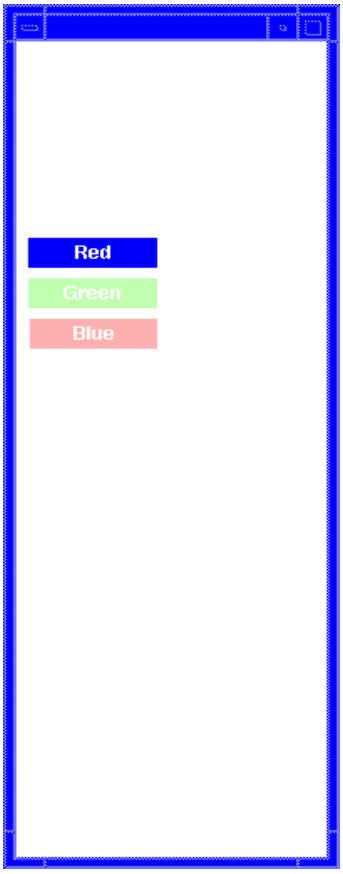

FIGURE 3. Pie property editor

2.5 Dynamic Usage

SX/Tools provides a mechanism by which events can be sent to an external partner (or client) and viceversa. Consequently, the graph methods can also be invoked from a separate client. This allows for applications requiring external resources or for already existing applications to easily use graph configurations designed under SX/Tools. Hence, with a combination of the graph specific property editors and the remote invocation of graph methods a dynamically changing graph can easily be constructed.

The general process by which a dynamic graph can be created starts with the specification of the interface to use. The designer, through the SX/Tools direct composition mechanism, constructs the user interface. Graphs can be instantiated and placed anywhere in the interface, as well as any other additional tools. Through the property editors, the designer configures the graph according to the application's needs. If the configuration of the graph is to change dynamically, the application need only invoke the graph methods.

3.0 The idea behind SX/Tools

- Envisaged application areas.
- The principle of direct composition (basically a ref to 1.1.3).
- Describe SX as a class library and a design environment.
- The major features of SX: aggregation (needs some details here), uniformity of design and runtime environment, end-user adaptability, design of the SX/Tools design environment using SX/Tools features.
- Properties and scripts.
- The structure of an SX-interface and an SX-application.
- The server-client structure and the application interface (don't forget this, I have already made 2 indirect refs to it).
- Extensibility, openness, multimedia environments.Implementation details, hard- and software requirements.
- Also have a ref here about design and execution phase.

4.0 Example Applications

In this section, we will briefly describe some of the applications we have built using SX/Tools and the graph tools addition.

- Business Applications

The construction of a static pie graph, bar graph or line graph is a very easy process. The designer requires only to create a new workarea, drag in the desired graph object. Then pop-up the property editor and simply edit the properties. No programming, no compiling, all interactive!

FIGURE 4. Pie graph created using the property editor

FIGURE 5. Bar graph created using the property editor

FIGURE 6. Line graph created using the property editor

■ Dynamic Applications

The X Windowing System comes with a set of utility programs. *xload* is one of such utility programs which displays in a window the CPU load over time. We found it trivial to design a similar application with our package. First we designed the graph configuration, then we wrote a small partner to obtain the CPU load (a series of system calls) and we had our version of xload. Furthermore, with minimal effort we could change the graph type.

FIGURE 7. SX/Load utility (bar graph version)

FIGURE 8. SX/Load utility (line graph version)

- Interactive Applications

Another program we implemented was a two dimensional function plotting program. The user inputs an expression (of one variable, using basic arithmetic operations, trigonometric functions and a variety of other standard functions). The parser, implemented as a partner, evaluates the expression and returns the points to plot. Again, the graph configuration and the functionality of the interface was constructed using interactive direct composition.

FIGURE 9. 2D Plotting Program

5.0 Future Work

We can divide the future work into improvements of the direct composition platform and improvements of the graph tools themselves.

5.1 Direct Composition Platform

The basic functionality of a direct composition platform is relatively easy to recognize. What presents great difficulty is determining the interface for the composition and which tasks should be easy to accomplish. Further work with Adaptive User Interfaces and Intelligent User Interfaces will surely contribute to improving the user-system relationship. Such improvements will dramatically improve the production speed of new tools and, in our case, more powerful graphs.

More specific enhancements are listed below:

- State saving: our current platform does not yet allow the user to save the dynamic state of user interface objects. Consequently, all interactive editing of a graph must occur in the original design phase. The platform does have a save option, but it does not include the dynamically generated information of objects, rather only saves the predefined properties.
- Link capability: we would like to allow for an elegant interactive specification of a link between 2 objects. For our initial property editing facility, we utilized a global object. A truly dynamic link capability (for example, with the introduction of a link event) would allow for a very intuitive though complex operation to be trivially accomplished.
- Multimedia, additional tools: in addition to graphing tools, we would also like to construct (with direct composition) a set of audio and video control tools, for example. A combination of these tools can easily produce a very impressive multimedia platform. A tangent group to ours is already pursuing this goal.

5.2 Graph Tools

The most obvious improvement for the graph tools is to further embellish their presentation. This can be accomplished by the addition of more methods and parameters.

A more interesting improvement is to provide mechanisms by which the designer can quickly compose his own graphs. Namely, rather than only having a toolbox containing three finished graphs, also offer a toolbox of graph components. This improvement is not only a change of the granularity of the end-user's tools, but rather a totally different concept whereby the designer, in fact, can "program" the actual graph.

One can envisage a graph with an initial set of methods available to a remote partner. The graph could be interactively composed for a specific application and thus offer yet another set of methods that control functionality before non-existent.

6.0 Conclusions

We accomplished a successful implementation of a powerful set of graphing tools using only direct composition, no additional system programming, thus proving the effectivity of a direct composition platform. The graphs operate at interactive rates, have a flexible interface and are easily expandable.

There are surely variations of graphs that our tools cannot currently handle, but with a system like SX/Tools adapting our graphs to the specific needs would require minimal effort. Furthermore, providing a low-level (object oriented in our case) scripting language within the platform has shown to be very useful. It allows for programmers to create whatever additional functionality is needed. Parallely, for the average non-programmer property editors and dragging and clicking is also available. Thus, programmers can compound objects that have specific functions and the non-programmers or designers need merely to click and drag in order to add the compounded objects to their application.

Additionally, the definition of a proper initial set of minimal widgets is crucial and as well as the flexibility of the implementation of additional tools. With SX/Tools, every tool is a scene, hence can always be edited. Ideally, of course, Adaptive User Interfaces would take care of the user-system relationship during the design phase and execution phase of an application.

7 References

[1] B. Shneiderman: *Designing the User Interface*, Addison-Wesley Publishing Company, 1987.

[2] Pergamon Infotech Limited, Maidenhead Berkshire, England: *Designing end-user interfaces*, 1988.

[3] T. Kuhme, M. Schneider-Hufschmidt: *SX/Tools - An Open Design Environment for Adaptable Multimedia User Interfaces*, Proceedings of Eurographics '92, Cambridge, UK, 7-11 September 1992.

[4] T. Kuhme, H. Dieterich, U. Malinowski, M. Schneider-Hufschmidt: *Approaches to Adaptivity in User Interface Technology: Survey and Taxonomy*, Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, Ellivuori, Finland, 10-14 August 1992.

[5] Visual Edge Software Ltd.,3870 Cote Vertu, Montreal, Quebec H4R 1V4: *UIMX*, 1990.

[6] SunSoft: *DevGuide*. ($$$$$$$)

[7] NeXT Inc., 900 Chesapeake Drive, Redwood City, CA 94063: *NeXTStep and the NeXT Interface Builder*, 1991.

[8] SX/Tools doc ($$$$$$$)

[9] D. Aliaga: *SX/Graphs: Implementation Overview*, July 1992.

[further ideas:

Add an example to the last paragraph of the graph tools future work section. I couldn't think of one (and its already Friday 24th!).

to-do:

Section 3.0 needs to be filled. 2 refs are still missing, if cannot be completed, omit them. Once final text is all inserted, arrange so that figures do not cause large empty sections in the paper.]