SID: A Graphical Browser

Tami L. Crawford
Computer Science Department
Canisius College
Buffalo, NY 14208
tami@canisius.edu

Abstract

Humans have powerful image processing systems which can assist in managing large amount of information. Program visualization can be used to exploit this capability to make complex systems manageable. In addition to being a tool for managing complexity, program visualization can support the maintenance of conceptual integrity, by providing a framework for viewing the program as a whole.

This paper presents SID, a program visualization tool. The program view is determined by a control panel, which is accessed by the user and a knowledge based assistant called the Gue. SID is a prototype system for viweing C++ programs.

1. INTRODUCTION

As hardware becomes increasingly more powerful and simultaneously less expensive, the types of tasks we choose to solve become increasingly larger and more complex. Unfortunately, advances in software technology have not kept pace with the gains in hardware. The reason for this, according to Brooks, is that software has several inherent properties, such as complexity, the need for constant change, and the inability of visualizing software[4].

Another of Brooks' observations is that "conceptual integrity is *the* most important consideration in system design" [3]. Conceptual integrity occurs when all parts of the system follow from the same set of underlying principles. Because conceptual integrity refers to the system as a whole, it cannot be achieved by considering parts of the system in isolation. A sense of the system as a whole is required.

Program visualization is a way of addressing both of these issues: that of the inherent complexity of software and the issues of conceptual integrity. Program visualization is concerned with using graphical techniques to display programs. Unlike visual programming, which deals with the specification and creation of programs using graphical techniques, program visualization is concerned with the using graphical techniques to display programs written in traditional languages (such as Pascal or C).[10] The tool proposed in this paper is concerned with program visualization.

Program visualization can be used as a way of managing complexity. Software is complex because there are so many different types of interconnections to manage. Humans have very limited processing capabilities, i.e., Miller's "magic number seven" representing the number of items an individual can remember at any given time has been confirmed over and over again[9].

One possibility for overcoming this limitation lies in the area of visual processing. Linguistic memory differs from pictorial memory[7,13]; humans can remember more from a picture than they might be able to remember from strictly verbal memory. It is not uncommon to hear "I never forget a face", but it is rare indeed to find someone who never forgets a name. By exploiting our powerful image processing capabilities, the amount of complexity which can be mastered by an individual may be increased.

Program visualization can support conceptual integrity by making it possible for the programmer to view the system as a whole. Abstraction is typically used whenever there are too many details to be considered all at once; the problem with software is that it is difficult to know which details are important and what are not. SID is a program visualization tool which copes with this problem by allowing the user to adjust a control panel which determines which features should be displayed. A knowledge based assistant, the Guru, also is used to determine which features should be displayed and which should be abstracted away.

SID is designed to enable the programmer to get a sense of the system as whole, and to enable the programmer to view the different types of relationships present in software systems within a single framework.

2. OVERVIEW OF THE SYSTEM

The system consists of several components, as shown in Figure 1. Program specific information is gathered by the compiler and stored in a database (Program Element Database). This includes information about program objects such as functions and variables and relationships between these objects. Program independent information is stored in the Program Independent Knowledge Base. This includes knowledge about programs and program components which is independent of any particular program. This consists primarily of knowledge about the semantics of particular objects which may be used by many different programs. This allows more

© 1991 ACM 089791-396-5/91/0006/0062 $1.50

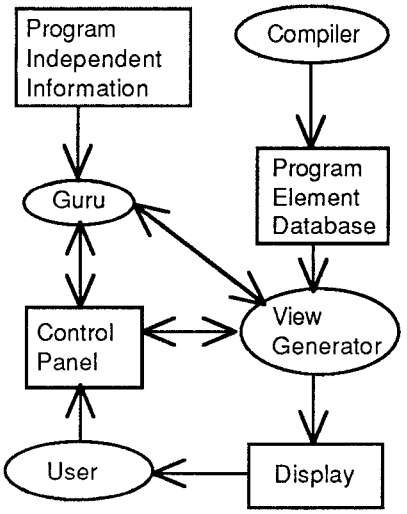

Figure 1. System Overview

detailed information to be stored about the effects of common objects, such as system calls or those found in frequently used libraries. The View Generator produces the image seen by the user.

There are four active entities (processes) : the Compiler, the Guru, the View Generator and the User. Each of these are described below:

- *Compiler*: The Compiler gathers surface level program information and deposits it in the Program Element Database. This guarantees that the view will be as up-to-date as the current executable and eliminates the overhead associated with generating the information each time the viewer is invoked.

- *Guru*: The Guru has access to program independent knowledge. The guru uses this knowledge when interacting with the view generator. The viewing controls and current location of the view are used to guide the guru in determining additional display parameters.

- *View Generator*: The View Generator produces the display using both syntactic and semantic information. The type and amount of information displayed is determined by the control settings (Control Panel). These settings determine which items to display and what attributes should be used to display the items. The View Generator uses the information contained in the Program Element Database and determines which subset to display depending upon user-specified controls. It also may consult with the Guru about additional control information.

- *User*: The User sets the controls and adjusts the viewing area by scrolling to select the viewing window, and zooming to select the magnification level.

There are three sources of information: the Program Element Database, the Program Independent Knowledge Base, and the Control Panel.

- *Program Element Database*: This contains information on variables, types, functions, and files. Interdependencies between the different components are stored. For instance, each function is defined in a file; it defines zero or more local variables; has zero or more parameters; uses zero or more global variables; calls other functions; and is called by other functions. The Program Element Database contains only global information.

- *Program Independent Knowledge Base:* The Program Information Knowledge Base captures the aspects of the system which are general and program independent.

  This knowledge repository is intended to contain knowledge an experienced programmer has about frequently used programming elements. For example, any seasoned C programmer has knowledge of standard system and library calls that goes beyond the number and type of arguments. An experienced C programmer would know, for instance, that files are accessed through a system file table which has a limited size, and that functions such as *open* and *creat* allocate a file descriptor, whereas the *close* function frees a file descriptor. Likewise, a X Windows programmer using the Xt Toolkit [1] knows quite a bit about Widgets which cannot be extracted automatically by the parser. Nevertheless, such information is crucial to the successful development of applications which use the Xt toolkit.

  A knowledge base of information about every object in the system would not be practical because of the effort of entering the information would be too great. However, it makes sense to encode this knowledge for parts of the system which are used over and over again, such as system calls and standard libraries.

- *Control Panel*: The control panel allows the user to adjust the view by specifying items to remove from consideration(filter), items which should stand out (enhance), and areas of interest. Controls can be adjusted by the user. The guru may inspect the controls and set additional controls based upon the settings and the contents of the Program Independent Knowledge Base.

3. THE DISPLAY

The display contains a program view, status information, and user controls. The program view consists of a group of objects with arcs representing relationships between objects. The objects consist of files, types, variables, and functions. There are a number of relationships between objects which can be displayed, such as *uses* and *defines*. The particular objects and relationships which are displayed are determined by the user's actions.

Displaying the program as a graph is only the first step. For any but the smallest programs, a graph contains so much information that is becomes difficult to isolate the information of interest. To address this problem, SID allows viewing parameters to be set to control which information is displayed and how that information is displayed.

Shape is used to indicate the type of object displayed. Functions are rectangles; variables are ovals; files are ellipses; and types are rounded rectangles. Color is used to distinguish arcs which represent different types of relationships.

Relationships between objects can be displayed in graph form or in tree form. In tree-form, hierarchical structure is enforced. Nodes are duplicated if necesary to keep the structure hierarchical. Conceptually the graph displays the interconnections between the components more compactly, but visually this results in a cluttered display.

4. CONTROL PANEL

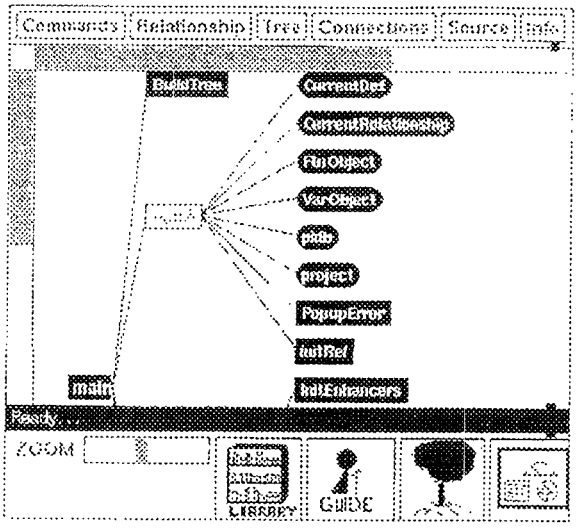

Figure 2. SID Display

4.1 Types of Controls

The control panel provides a mechanism to adjust the display. Viewing parameters may be set by the user and the guru. The controls determine what information to display and the attributes which should be used to display the information. Controls are associated with objects.

A group of objects can be specified as

- *Regular Expression:* A regular expression can be specified which represents the names of the objects to be affected. For example, all functions which begin with *Xt*.

- *Named Group:* A name which has been associated with a set of objects previously can be specified. For example, the *SystemCalls*, which represents all system calls. SID knows about the contents of standard libraries such as *X* and *Xt*, and has the ability to create name lists for any library.

- *Interest Area:* An interest area can be specified to indicate an interest tothe Guru. For example, an interest area might be *Terminal I/O*. The Guru consults the Program Independent Knowledge Base to determine which objects are affected by this interest.

- *Program Relationships.* The user can define a view over the Program Element Database, such as all functions which modify a particular global variable or all functions which have a McCabe's complexity measure [8] greater than 20.

Once a set of objects has been selected, the set may be filtered, enhanced, or used to apply a fisheye view.

4.2 Filters

One type of viewing control is the filter. Filters eliminate from the view all objects which meet a certain criteria. For instance, the user might view the call graph, which displays the calling relationships between functions. This will give the user an overall view of the structure of the program. For a small program, the call graph may provide meaningful data, but for a large program the call graph is more likely to resemble a ball of twine than anything else. One problem with the call graph is that not all functions contribute equally to program structure. For instance, a call of the function *strcmp*, which compares two strings and returns a result, contributes little to the overall structure of the program. Including such a function in the call graph obscures relationships between other functions which are more relevant to the program structure.

For this reason, the user could specify a filter which eliminates standard library functions (such as *strcmp*) from the display by selecting the *StandardLibraryCalls* list as a filter.

4.3 Enhancers

Another type of viewing control is the enhancer, which augments the display of items which meet certain criteria. For example, the user may specify that all functions which access a particular global variable are displayed in light blue; or that all functions which have a complexity greater than 20 be displayed in purple. The user may select the attributes used to enhance the display. Typical attributes are background and foreground colors.

4.4 Fisheye View

The fisheye view is a technique for showing local information in detail and displaying related information in a level of detail depending upon its importance and distance from the current focal point[5]. Other details are shown depending upon their *a priori* importance and distance from the focal point. The system defines a threshold degree of interest for displaying items. The degree of interest is defined as:

y = current focal point
x = item
DegreeOfInterest(x)= A Priori Important(x) - Distance(x,y)

Items further away will not be shown unless the items has some *a priori* importance. Areas of interest can be specified from the control panel. The areas of interest are used to compute the *a priori* importance of items.

4.5 User Control Adjustments

The user has several different ways of controlling the display. First, the user is responsible for deciding which parts of the program to view. The user can selectively expand different nodes of the graph or expand all nodes of the graph. The user can zoom in or out and scroll the graph if it is too large to fit on the screen all at once.

Second, the user determines the type of information represented by the graph. When nodes in the graph are expanded, the relationship displayed depends on the current relationship. For a function, the current relationship can be one of: calls, is called by, global variables used, and defined in.

The user decides whether the display will be a graph or hierarchical display. If the user chooses a hierarchical display, nodes which are referenced more than one place are duplicated.

The display initially contains a single object. The user specifies the relationship of interest and expands nodes by clicking on nodes of interest. All nodes in the display may be selected to be expanded or the user may select nodes one by one. When the user clicks on a node, the node is expanded to contain information on the current node using the current relationship.

4.6 Guru Control Adjustments

The Guru can adjust the controls based upon the interest areas the user has specified. If, for instance, the user is using the Xt Toolkit and is interested in color settings, the Guru can use this to highlight any functions which may set the color. Color can be set using resource files, by modifying widgets with the *XtSetValues* functions, or by specifying the color on the command line. A resource file is stored in one of several different locations and a fallback list of resources may be specified inside the program if the resource file cannot be located. Using this knowledge, the Guru examines the program and the environment to determine which of this applies to the current situation, and then displays the results accordingly.

5. IMPLEMENTATION

A prototype of SID is implemented in g++ on a Sun Sparc-Station 1+, using X Windows (X11R4), and the Athena widget set. The g++ parser was modified to output structural information to a database during compilation. The Guru is being implemented using the CLIPS system[14].

6. RELATED WORK

A number of systems have some goals common to SID. CIA++ is a system based upon a relational database[6]. Browsers and other tools can be easily implemented which use the cia++ database. Another system which uses a program database is the FIELD environment, which includes a C++ class browser (*cbrowse*)[12]. Like SID, both of these system store program information in a database. The difference between these systems and SID is that SID is concerned with the interactive display and how to modify the display using user-settable controls and a knowledge based assistant which contains knowledge about common programming elements.

Some work has centered on using typographic attributes such as fonts and page layout principles to increase the comprehensibility of programs[2,11]. SID is concerned more with the "big-picture" view of the program and is designed to work interactively with a color display.

7. CONCLUSION

In conlusion, SID is a system for program visualization. The distinguishing characteristics are that it uses a knowledge based software assistant to assist in determining which components of the program should be visible and what attributes should be used to display these components. The user also can adjust the viewing controls.

An important component of the system is the Program Independent Knowledge Base, which contains knowledge about components which are used in many different systems, such as libraries of code.

Although much work remains to be done, a prototype which demonstrates the feasibility of the concept has been implemented.

REFERENCES

1. Asente, Paul J. and Ralph R. Swick, *X Window System Toolkit: The Complete Programmer's Guide and Specification*, Digital Press, 1990.
2. Baecker, Ronald M. and Aaron Marcus, *Human Factors and Typography for More Readable Programs*, Addison-Wesley, 1990.
3. Brooks, Fred P., *The Mythical Man Month*, Addison-Wesley, 1975.
4. Brooks, Fred P. "No Silver Bullet: Essense and Accidents of Software Engineering", *IEEE Computer*, April, 1987.
5. Furnas, George W. "Generalized Fisheye Views", *CHI '86 Proceedings*, 1986.
6. Grass, Judith E., and Yih-Farn Chen, "The C++ Information Abstractor", *USENIX C++ Conference Proceedings*, 1990.
7. Haber, Ralph Norman, "How We Remember What We See", *Scientific American*, 1970, 222,104-112.
8. McCabe, T. J. "A Complexity Measure", *IEEE Transactions on Software Engineering*, SE-2,4, December 1976,p308-320.
9. Miller, George, "The Magic Number Seven Plus or Minus Two: Some Limits on Our Capacity for Processing Information", *Psychological Review*, 63,81-97.
10. Myers, Brad, "Visual Programming, Programming by Example, and Program Visualization: A Taxonomy", *CHI '86 Proceedings*, 1986.
11. Oman, Paul W. and Curtis R. Cook. "Typographic Style is More than Cosmetic", *Communications of the ACM*, Volume 33, Number 5, May 1990.
12. Reiss, Steven P. and Scott Meyers, "FIELD Support for C++", *Usenix C++ Conference Proceedings*, 1990.
13. Shepard, R. N., "Recognition memory for words, sentences, and pictures.", *Journal of Verbal Learning and Verbal Behavior*, 1967, p.156-163.
14. CLIPS, COSMIC, Athena, Ga.