An Interactive Visualization Tool for Understanding Complex Programs

J. Cherry, M. Arrieta, E. Brown and S. Ramaswamy
Software Automation and Intelligence Laboratory, Department of Computer Science
Tennessee Technological University, Cookeville TN 38505. Phone: (931)-372-3691
Email: srini@acm.org / srini@ieee.org

Abstract

In this paper, we present the development of a program parser coupled with an off-the-shelf interactive program visualization tool that assists in the understanding of complex computer programs. The major objective of this work is to quickly help newcomers become familiar with existing software code and become productive members of the software development team. These software systems that often incorporate several interacting pieces will be produced and maintained by several individuals and/or vendors. Hence, it will be difficult for one individual or group to be fully cognizant of the intricacies of the overall software system. In work-related training, interactive visual systems will be required to provide support for teaching organizations’ newer workforce to quickly learn and effectively contribute to the development and maintenance of these complex and critical software systems. An environment that provides an in-depth view of the software system will be necessary to fulfill this requirement. Additionally, an interactive visual system may provide a means to dynamically access and analyze the skills and capabilities of trainees for skill-based job placement.

I. Introduction

Becoming familiar with a complex software program is an intricate and time-consuming task for individuals and group members new to a particular project. The task can be somewhat simplified with the use of a graphical visualization tool that displays a structural view of the software system. The hierarchical structure of programming code and its components can be represented in numerous ways through the use of various visualization environments.

This paper is organized as follows: Section II presents an overview of the various visualization tools commonly found in the literature. Section III presents our general framework for program visualization tools. Section IV presents an example visualization parser built for C++ programs. Section V concludes the paper.

II. Background

In the process of development of a tool for visual presentation of software programs, it is necessary to select a visualization tool that is capable of presenting large programs in a form that is easily visualized and traversed. In this section, we will first present the various visualization tools found in the literature and the major drawback of these tools, and then present several commercial hyperbolic tree based visualization tools. In our approach we chose the hyperbolic visualization mechanism because of its widespread commercial adoption.

II.A. Overview of Visualization Tools

Several researchers have worked on visualization environments for software systems. In [1-2], the authors provide a Java based visualization tool with a zoom-able interface. This tool provides context-based visualization, which is a semantic presentation that displays a particular view of a node based on a specific task. It presents a very flexible visualizer that is capable of three different types of zooming – geometric, semantic and fisheye. In geometric zooming the user is able to scale a nested view around a central point. In semantic viewing the user is able to display specific semantics about the selected node in the visualization – for example, visualizing children nodes, documentation, etc. However, it is developed for visualizing Java programs. In [3] the authors present Tree maps, a tool for quick overview of hierarchically structured data. While Tree maps effectively utilize limited space, they offer no navigational capabilities, which are essential for a tool whose purpose is to provide visualization of software code. Other tree-based visualization tools include the Spence layout [6], fractal approaches [7], cone trees [8], and multi trees [9]. Issues addressed in these various mechanisms
include: efficient usage of screen space, presentation elegance, and clarity to the user.

II.B. Hyperbolic Trees

Visualization applications using a hyperbolic tree structure were proposed in [4-5]. In this approach, the hierarchy is mapped onto a circular region using a hyperbolic mapping, wherein parallel lines, originating from a central node, diverge from one another subsequently increasing the available amount of space for lower tree levels. This structure is especially effective in organizing and displaying huge volumes of related data. The user can choose part of the tree as a region of interest, which then becomes the display’s new, graphical center. This system of visualization permits the user to view, and navigate, a hierarchical representation of a complex software system through a series of interconnected nodes. As a result, related program components are graphically represented close to one another, which allows the user to easily traverse the visualization.

Some commercial products have been developed using these techniques. For example, Inxights’s VizServer uses Star Tree (a hyperbolic tree) for navigating information collections and another application (Table Lens) analyzing large data tables. Although hyperbolic visualization tools provide features necessary for general visualization of hierarchical information, some researchers have reported problems with this type of visualization [10]. One such problem is the circular shift around the center of the visualization. This shift, or focal change, takes place when the user interacts with the system. Due to the nature of the shift, the re-positioning of the graphical elements may create some confusion for the user. Also, at lower levels in the tree, the hyperbolic structure is said to suffer from an information overload, and that curved tree links and non-regularity of hyperbolic space may hamper effective tree visualization.

However, in our experience the above-mentioned problems

![Figure 1. Framework for Program Visualization](image1)

![Figure 2. Perl-based Parser for C++ Programs](image2)
do not seem to surface in the context of program representation, and the hyperbolic visualization systems appear to be ideal for program understanding; wherein program dependency structures are often unbalanced. It has been noted that by trying to provide a rotating structure such as a hyperbolic tree, users have been able to gain useful insights into the information being visualized [8]. This is due to the way in which programmers perceive the structure and interconnections of the components of a modular software system. Often the users, usually newcomers to a programming team, are people who already have a good sense of the programming language, which helps them in understanding the context of the information being presented.

III. General Framework

One of the main objectives of our work is the development of a full-fledged visualizing environment that is not specific to a particular language or environment. Thus we started off with a literature survey to study the various visualization tools and their language and/or environment

```cpp
class C_no : public B_no {
public:
    int i;
};
class B_no {
private:
    int j;
};
class Point {
private:
    Point() {}
};
class Circle : Point {
public:
    int i;
};
class C_yes : public B_no {
public:
    int value;
};
class b_no {
public:
    int stuff;
};
```

Figure 3. Sample Program Fragment

Figure 4. Basic View

Figure 5. Visualizing Class Derivation Information
dependencies. Many of the visualization tools presented in the previous section (see II.A.) were indeed tied to a particular language / environment. For example, the ShriMP visualization tool is specific to Java. Our approach to developing a generic program visualizer is represented in Figure 1. As shown in the figure, we decided to use a commercially available visualizer – InXight’s hyperbolic tree visualizer. The major reason for this is that this visualizer is developed as a Java applet, thereby making it platform and browser independent. With this framework in place, it should be easy to develop a parser for any language that outputs a suitable text file for visualization by the hyperbolic visualizer applet. A C++ Parser written in Perl is presented in Figure 2. Figure 3 - Figure 5 illustrate the application of the parser and the visualizer for a simple C++ code fragment. Figure 5 also illustrates the parser retrieving super-class sub-class derivation relationship.

IV. Example Application

The framework described above was used to help students “transitioning” through the lab get acclimated with our projects. The use of this parser to generate an input file for visualizing GNU’s C++ header files is shown in Figure 6. The visualizer for a C++ software project developed for one of our clients is shown in Figure 8. This project had several students working on it over a period of 3 years.

V. Conclusions

In this paper, we have presented a general framework for visualizing software programs. While several visualization tools are language and platform dependent, we have proposed a mechanism for the general visualization of large programs. The parser presented is for C++ and in the future we propose to develop a parser for other frequently used languages, such as C and Java. One restriction we have noticed with the use of hyperbolic tree based visualizer for program visualization is the basic tree structure itself. Visualizing common software modules that occur as leaf nodes in different branches of the tree hence becomes a problem. So we are in the process of developing a hyperbolic graph structure that may be more amenable to such representations.

VI. References

Figure 7. Visualization of an Industrial C++ Project

Figure 8. Visualization of an Industrial C++ Project