An Overview of BLOOM

Steven P. Reiss
Department of Computer Science
Brown University
Providence, RI 02912-1910
401-863-7641, FAX: 401-863-7657
spr@cs.brown.edu

ABSTRACT

BLOOM is a system for doing software understanding through visualization. It provides facilities for static and dynamic data collection. It offers a wide range of data analyses. It includes a visual query language for specifying what information should be visualized. All these are used in conjunction with a back end that supports a variety of 2D and 3D visualization strategies.

1. INTRODUCTION

Software understanding is the task of answering questions and gaining insights about software systems. In some cases it involves gathering broad-stroke information about what a system is doing; in other cases it involves understanding a particular aspect of the system; in still other cases it involves addressing very specific issues such as why was this routine called or what will happen if I change this input.

Providing tools to enhance and facilitate software understanding has always been difficult. While there have been a wide range of efforts, both in the reengineering community and in the software environments community, few of these efforts have been really successful or led to tools that are in active use today. In particular, tools that used visualization as a means to software understanding have been proposed and demonstrated, but have rarely been incorporated into successful programming environments, and, when they have been, have not been used.

For example, a lot of work went into developing a variety of different visual browsers as part of programming environments [8]. These provided the user with visual, often hierarchical views of the call graph, the module structure, the dependency structure, or the class hierarchy. These were included in a variety of programming environments including FIELD, HP Tooltalk, DEC Fuse, and Sun SparcWorks. However, they were not widely used in these environments and have not been duplicated in current environments except in a rudimentary form as a simple tree view.

Several years ago, we analyzed the reason why these systems, which intuitively seemed so obviously useful, were not being used. Our analysis suggested that the primary reason was that they failed to address the actual issues that arise in software understanding. In particular, they provided fixed views of a fixed set of information and were not flexible enough to let the user address the specific questions or varying issues that actually arise. Secondary concerns were the difficulty in using such systems, both in terms of setting up the data for them and in understanding how to get the desired information out of them, and the overwhelming amount of information inherent to a real software system.

To address these problems, we started designing and developing a new visualization system that would let the user easily address specific problems and would handle large software systems. This system, BLOOM, offers several capabilities that should allow it to succeed where previous systems have not. In particular it:

- Collects a variety of program information including program traces, structural information, and semantic information, all unobtrusively.
- Provides a range of different analyses of this information that both summarize and highlight its relevant aspects.
- Offers the ability to combine the different analyses in new and potentially interesting ways.
- Presents a graphical front end for defining what analysis or data is interesting to the problem at hand and for describing how that data should be combined.
- Uses a powerful information visualization framework to provide multiple high-density views of the resultant information along with the necessary browsing facilities for making use of the information.

In the remainder of this paper we consider the current state of BLOOM in each of these areas.

2. INFORMATION SOURCES

Because the problems of software understanding are varied and cut across all aspects of the software, it is important for a software understanding system to provide information from a variety of different sources. Moreover, because the specific problems that will arise are not known in advance, it is important that the relevant data be either available a priori

PASTE'01, June 18-19, 2001, Snowbird, Utah USA.

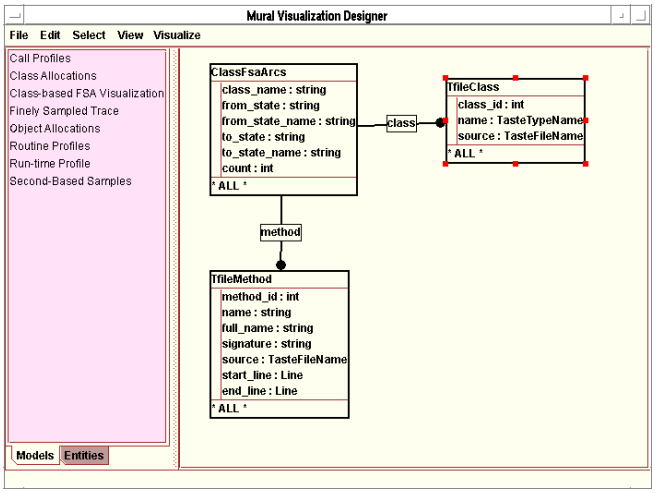

FIGURE 1. The visual query interface for defining what to visualize. Here it shows three entities relating information about class usage via finite state machines with associated static data.

or that it be generated relatively rapidly. The data also should be available without the programmer having to go to such extremes as recompiling the whole system with different options or using a different compiler.

We have developed several information sources that meet these requirements. We use a program database from the Desert [5] project for C and C++, and a similar program database for Java that is part of the experimental Tea environment at Brown. We also use a tracing facility, described in [6], that provides information on calls, memory usage, and synchronization for both C/C++ and Java programs.

3. INFORMATION ANALYSIS

Raw trace data, and even raw structural or semantic data for a realistic system, is too large to be understood directly. Instead, one must analyze this data to summarize it, to find patterns, or to highlight potential problems or areas of interest. To address this, BLOOM uses a facility described in [7] that provides a variety of different data analyses.

4. COMBINING ANALYSES

While the above analyses provide lots of useful information, real insight into the behavior or structure of a program often requires that the various analyses be combined in new and different ways. Realizing this, BLOOM tries to provide the facilities necessary for creating new information through combinations of the ones provided.

BLOOM does this through a data manager that provides access to data in a uniform way and that lets new objects be created from existing ones. The data manager is based on an object-oriented view with a entity-relationship based query language. The base data comprise the original set of entities. Relationships among this data can either be explicit through links (or IDREFs in the XML database), implicit through fields in one entity that have domains that match keys in another entity, or user-defined by associating values in one entity with values in the second. New entities can be formed by combining existing entities using any of these relationships or restricting the entities with constraints. New relationships can be formed by combining sequences of relationships, either directly or using various forms of transitive closure, and by restricting the relationships using constraints. The result is a fully-functional query language.

5. ASKING THE QUESTION

While it is important to provide a wide range of different analyses and to permit these to be combined in order to do realistic software understanding, it is equally important to offer users an intuitive and easily used interface that lets them select and specify what information is relevant to their particular problem.

Using experiences from our previous work in this area [3,4], BLOOM includes a front end for specifying the desired information using a visual query language. This interface provides several features designed to let the user select and refine what information is relevant to the problem at hand. It attempts to make it easy for the user to select and relate the different trace analyses and the static data. This is all done without having to understand the exact structure of the underlying databases or the data. Moreover, the system provides facilities for saving useful queries so they can be reused.

While our experience to date with the this interface has been limited, it has proven to be as powerful as our previous interface and a lot easier to use. A screen shot of the interface can be seen in Figure 1.

6. SEEING THE RESULT

Once users have defined what information is relevant to their immediate software understanding task, they need to see the information in an understandable and meaningful manner. BLOOM provides for this in a three-step process.

The first step is to let the user select an appropriate visualization for the given data. The visual front end determines which visualization methods can be used with the user-specified data and assigns each a score based on how well the data definitions match what the visualization expects. In doing this, the system automatically looks at all possible ways of combining entities and relationships to get a better match. In the end, the user is presented with a sorted list of those methods that seem to be the best.

The second step is a visualization framework that supports a wide range of visualizations along with an initial set of visualizations geared toward the display of high-dimensional, large data sets as well as the types of structured data that frequently occurs as the result of analysis. The current set of visualizations includes:

- *Box trees*. This is a tree representation where the nodes are drawn adjacent to their parents without having to display the arcs. Color, height, depth, and size can be used to represent various dimensions of data. We also support a variant of 3D tree maps [8].
- *File maps*. This is a representation that is roughly equivalent to SeeSoft [2]. It can be used to display information relevant to a single file or a set of files. Color, height, and texture can be used to represent information that is associated with lines in the files.
- *Layouts*. We include the ability to display 2D or 3D graphs consisting of nodes and arcs where this is appropriate.
- *Point maps*. These are generalized 2D or 3D scatter or dot plots [1]. Hue, saturation, size, and rotation can be used to represent information. If appropriate, consecutive points can be connected by parameterized lines.
- *Spirals*. We use spirals to show time series data in a compact and space-efficient manner. Hue, saturation, height, and width can be used to show information relating to time.

A set of visualizations illustrating some of the capabilities of the system is seen in Figure 2. The framework is designed to be easy to extend. New visualization methods can be written as separately compiled entities and dynamically loaded. Most of the underlying support for drawing, picking, layout, mapping values, and retrieving information from user objects is built-in. We have been able to create and add new visualization methods in one to two days depending on the complexity of the graphics and the number of data dimensions the method is designed to support.

While the basic visualizations that are provided with this facility are helpful and provide a useful overview of the selected data, most software understanding problems require some sort of drill-down to get at the details of the data. The third step in BLOOM's visualization pipeline provides for this. Each visualization is provided with a control panel. This panel lets the user define and change the associations between user data fields and visualization properties, and to change the various parameters that control the visualization. The panel also lets the user restrict the data being displayed by specifying ranges (for value fields) and regular expression patterns (for text fields) that either describe what data is relevant or what data should be ignored. In addition, the system will display full details about whatever object is under the mouse. Together these features provide a powerful facility for obtaining the specific information that is needed. We are currently in the process of expanding this to allow the user to dynamically define groups of objects and to provide automatic semantics-based correlations among multiple views.

7. CURRENT AND FUTURE WORK

BLOOM is currently in its alpha state. All the components are there, although several are missing features that we feel are or will be needed. Some of these features include:

- Data collection is still too inefficient and can be speeded up by at least a factor or 2. Moreover, we need to have the ability to collect both finer-grain data (such as basic block tracing) as well as higher-level data such as input-output processing or transaction and event processing.
- More data analyses will be needed for understanding more complex systems. In particular we are considering various analyses of thread interactions as well as event-based modeling.
- We are looking into various XML-based query languages and XML-based databases as potential replacements for our specialized data manager.
- We need to do user testing on the visual query front end to determine its usability and appropriateness. We also need to provide additional facilities that let the user obtain information about what data is available.
- We need to provide additional visualization approaches to address other problems such as describing the various encodings or showing interval graphs.
- We need to finish the additional browsing facilities we are currently working as well as the interface and mechanisms to support multiple connected views.

However, even with our limited experience, it appears that BLOOM is a practical and useful system for software understanding. As the system matures we hope to collect user data to either verify or disprove this hypothesis.

8. REFERENCES

1. Kenneth W. Church and Jonathan I. Helfman, "Dotplot: a program for exploring self-similarity in millions of lines for text and code," *Journal of Computational and Graphical Statistics* Vol. **2** pp. 153-174 (1993).

2. Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner, Jr., "Seesoft - a tool for visualizing software," AT&T Bell Laboratories (1991).

3. Steven P. Reiss, "Cacti: a front end for program visualization," *IEEE Symp. on Information Visualization*, pp. 46-50 (October 1997).

4. Steven P. Reiss, "Software visualization in the Desert environment," *Proc. PASTE '98*, pp. 59-66 (June 1998).

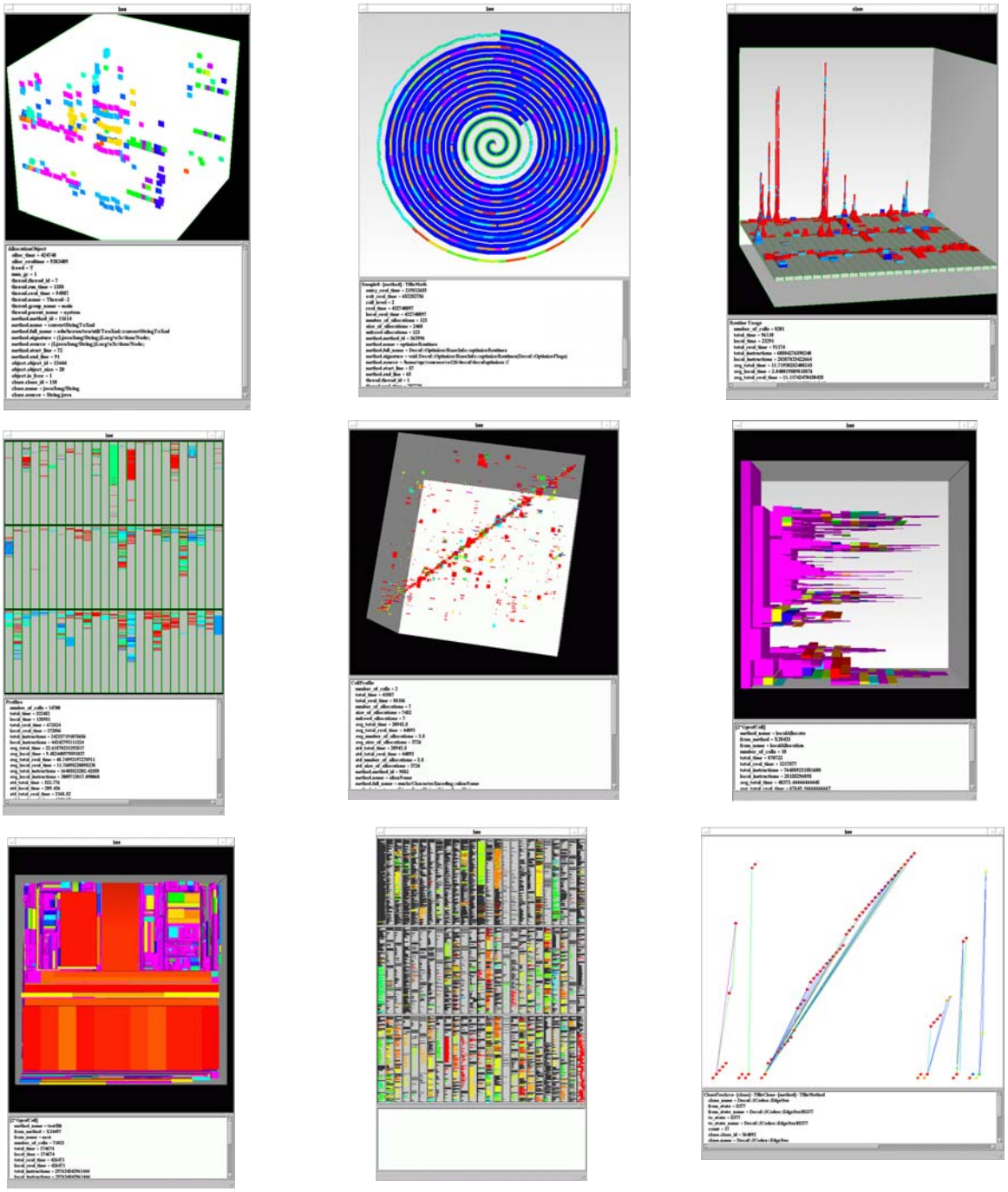

FIGURE 2. Sample visualizations. From left to right, top to bottom, they represent a point map of a restricted set of object allocations located by time and colored by class, a spiral showing a restricted and sampled call graph of an optimizing compiler, a file view showing performance information for that compiler, a simpler file view showing real time performance, a dot-plot showing similar performance information by calls, a box tree view of gprof-like performance information, a nested tree-map view of the same data, a SeeSoft like view of routine performance, and a dot plot illustrating a subset of class usage automata.

5. Steven P. Reiss, "The Desert environment," *ACM TOSEM* Vol. **8**(4) pp. 297-342 (October 1999).

6. Steven P. Reiss and Manos Renieris, "Generating Java trace data," *Proc Java Grande*, (June 2000).

7. Steven P. Reiss and Manos Renieris, "Encoding program executions," *Proc ICSE 2001*, (May 2001).

8. Ben Schneiderman, "Tree visualization with tree-maps: a 2-D space-filling approach," *ACM Transactions on Graphics* Vol. **11**(1) pp. 92-99 (January 1992).