FEAT
A Tool for Locating, Describing, and Analyzing Concerns in Source Code

Martin P. Robillard and Gail C. Murphy
University of British Columbia
Department of Computer Science
201-2366 Main Mall, Vancouver, BC
Canada, V6T 1Z4
{mrobilla,murphy}@cs.ubc.ca

1 Introduction

Developers working on existing programs repeatedly have to address concerns, or aspects, that are not well modularized in the source code comprising a system. In such cases, a developer has to first locate the implementation of the concern in the source code comprising the system, and then document the concern sufficiently to be able to understand it and perform the actual change task.

Several approaches are available to help software developers locate and manage scattered concern code. Lexical searching tools, such as grep [1], code browsers, such as the Smalltalk integrated development environment [3], cross-reference databases, such as CIA [2], and slicers [9], can each help a developer identify relevant points in the code and can help elicit the relationships between the different parts of a program. Alternatively, a developer may be able to leverage the identification of the change from a previous modification using version differencing in a source code repository. All of these tools produce a similar result: the developer is presented with the lines of source code contributing to the concern in the system. This ad hoc, source code-intensive representation of concerns is difficult to use as the basis for reasoning about and analyzing concerns for the purpose of software evolution.

To address the problem of effectively capturing knowledge about the implementation of a concern in source code, we have investigated the possibility of describing concerns as a graphs of relations between program elements. We have named this representation Concern Graph [7, 8]. We have developed a tool, called FEAT [6], to support building and analyzing Concern Graphs. Specifically, FEAT supports locating, describing, and analyzing the code implementing a one or more concerns in a Java system.

2 The FEAT Tool

FEAT is developed as a plugin for the Eclipse Platform [5], an integrated software development environment with a plugin architecture supporting the addition of functionality (Figure 1, last page).

With the FEAT plugin activated, users of Eclipse can use the integrated development environment as usual, to browse and modify code, perform searches, etc. However, if a user desires to create a concern representation, the FEAT Perspective can be activated and a concern representation created. At this point, a *concern* in FEAT is simply a named container for a fragment of a program which is of interest to a user. Any class, method, or field in a project can then be moved to the FEAT Perspective.

Elements in the FEAT Perspective can be analyzed for their dependencies to other elements in the source code. Any element or relations of interest to a user can be stored as part of a concern. The source code corresponding to any element or relation can also be viewed in code viewer (bottom window).

Concerns descriptions can be used to systematically analyze the code for a concern, or to compare two different concern descriptions. In both cases, FEAT automatically detects relations between elements in concerns, and presents the relations visually to the users.

The architecture of FEAT consists in three components: a *model* component providing operation on concerns and ensuring the consistency of the concern description, an Java bytecode *analyzer* component, built on top of IBM's Jikes Bytecode Toolkit [4], which provides relations between different elements in a program, and a *GUI* component which ties the tool into the Eclipse Platform.

We have successfully used an earlier prototype of the FEAT tool to locate, describe, and analyze concerns for the purpose of software evolution in 4 academic case stud-

0-7695-1877-X/03 $17.00 © 2003 IEEE

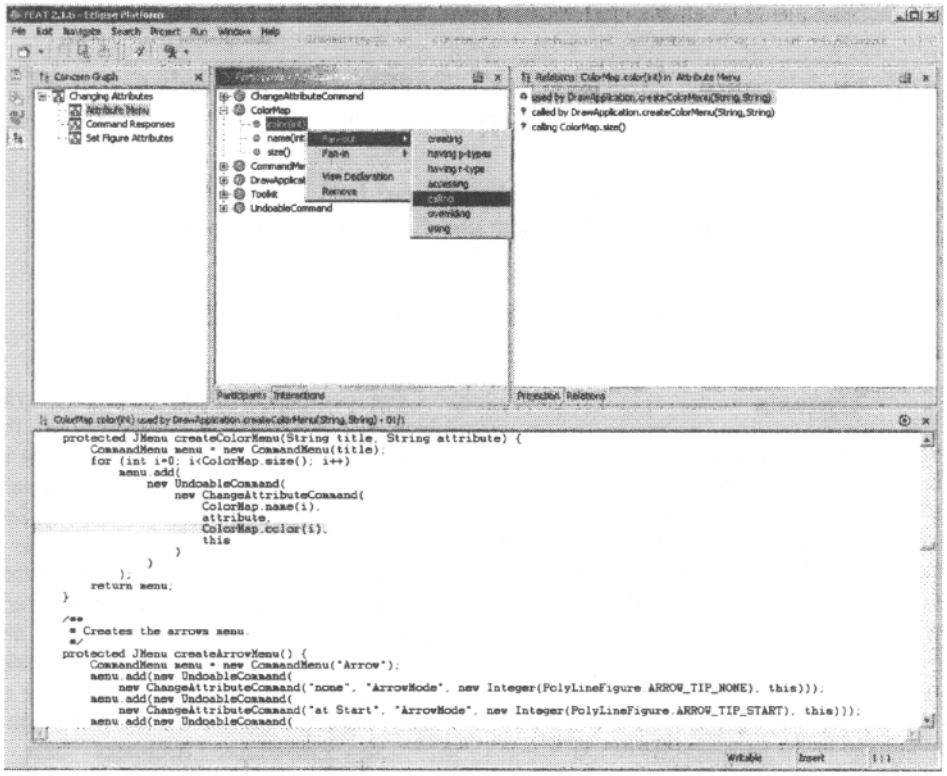

Figure 1. The FEAT tool.

ies [8], including a scalability study involving a large industrial code base. The current version of the tool is presently being used in a new series of user studies.

3 Description of the Demonstration

The demonstration will consist in performing part of a change task on a Java program with the help of FEAT. First, we will use FEAT to locate and analyze a set of concerns relevant to the change task. Specifically, we will show how, by visually navigating structural program dependencies through the tool's graphical interface, we can rapidly locate the code implementing a concern, and store the result as an abstract representation consisting of building blocks that are easy to manipulate and query. We will then show how the representation of the sample concern supported by FEAT can be used to investigate the relationships between the captured concern and the base code, between the different parts of the concern itself, and between different concern representations. Finally, we will show how FEAT can be used to keep track of the actual source code implementing the concern.

References

[1] A. V. Aho. Pattern matching in strings. In R. V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 325–347, New York, 1980. Academic Press.

[2] Y.-F. Chen, M. Y. Nishimoto, and C. Ramamoorthy. The C information abstraction system. *IEEE Transactions on Software Engineering*, 16(3):325–334, March 1990.

[3] A. Goldberg. *Smalltalk-80: The Interactive Programming Environment*. Addison-Wesley, 1984.

[4] The Jikes bytecode toolkit. IBM, March 2000. http://www.alphaworks.ibm.com/tech/jikesbt.

[5] Object Technology International, Inc. Eclipse platform technical overview. White Paper, July 2001.

[6] M. P. Robillard. The FEAT Eclipse Plugin: A tool for locating, describing, and analyzing concerns in source code. http://www.cs.ubc.ca/~mrobilla/feat2.

[7] M. P. Robillard and G. C. Murphy. Capturing concern descriptions during program navigation. Position paper for the OOPSLA 2002 Workshop on Tool Support for Aspect-oriented Software Development, November 2002.

[8] M. P. Robillard and G. C. Murphy. Concern Graphs: Finding and describing concerns using structural program dependencies. In *Proceedings of the 24th International Conference on Software Engineering*, May 2002.

[9] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.