AutoCode: Using Memex-like Trails to Improve Program Comprehension

Richard Wheeldon, Steve Counsell and Kevin Keenoy
Department of Computer Science
Birkbeck College, University of London
London WC1E 7HX, U.K.
{richard,steve,kevin}@dcs.bbk.ac.uk

1 Introduction

In his seminal paper "As We May Think" [1], Vannevar Bush suggested a future machine called a "memex". In doing so, he introduced the world to the concept of linked documents and of the *trail* - a sequence of linked pages. The concept of trails is well established in the hypertext community and many systems have been built which support their construction [2].

Previous work has described a *navigation engine* for automatically constructing trails as a means of assisting users browsing Web sites [4]. This navigation engine was further used to provide search and navigation facilities for Javadoc program documentation. If JavaDoc-style program documention, which is derived from source code, can be indexed, it seems logical that the source code itself can be indexed.

We have developed a new tool called AutoCode based upon the navigation engine design. AutoCode provides full-text indexing of the java source code and uses a probabilistic best-first algorithm to identify trails in graphs of coupling-type relationships.

2 Trails on Java Code

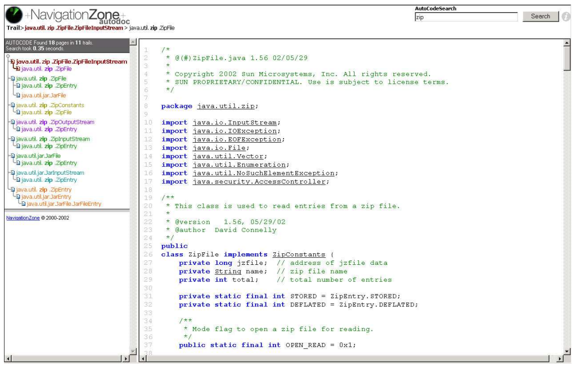
Classes and objects in OO systems to not work in isolation. The classes are connected to each other by various dependencies. The Java language connects classes together via five coupling relationships - Aggregation, Inheritance, Interface, Parameter and Return Type [3]. Each of these coupling relations can be used to construct a graph of dependencies. AutoCode constructs trails on each of these five graphs and presents them in a Web-based interface.

The *NavSearch* user interface used to present the trails (figure 1) has three main elements. At the top is a *navigation tool bar* comprising of a trail of classes considered most relevant (the "best trail"). On the left is a *navigation tree window* showing all the trails. Whenever the mouse pointer moves over these trails, a small pop-up appears which shows metadata and an extract. The rest of the display is dedicated to showing the source code of the selected class. A demonstration of this interface showing the 6000 classes of the JDK libraries is available at http://nzone.dcs.bbk.ac.uk/.

Each trail is colour-coded according to the type of coupling involved. This coupling type is also shown in the pop-up for each class. green trails denote parameter type references, cyan trails denote return-type references, gold trails show interface extensions, purple trails shows chains of aggregation links and orange trails show inheritance relationships from subclass to superclass.

Figure 1 shows how the trails are presented for the results to the query "zip" on the JDK 1.4 source code. Figure 2 shows the trails more clearly. It can be easily seen from the first trail that there is a member variable of type `ZipFile` in the class `ZipFileInputStream`. The second and third trails start with the common root, `ZipFile`. These show that one or more methods in the `ZipFile` class must take `ZipEntry` as a parameter and that `ZipFile` has a subclass called `JarFile`. The fourth trail shows that `ZipFile` implements the interface `ZipConstants`. The fifth shows that `ZipOutputStream` has a member variable of type `ZipEntry`. The sixth and seventh trails show that both `ZipInputStream` and `JarFile` have methods which take `ZipEntrys` as parameters. The eighth trail shows that `JarInputStream` has at least one method which returns a `ZipEntry` and the ninth shows that `ZipEntry` is the superclass of `JarEntry` which is, in turn, the superclass for `JarFile.JarFileEntry`.

AutoCode indexes the Java code using a custom doclet.



Trail > java.util. zip .ZipFile.ZipFileInputStream > java.util. zip .ZipFile

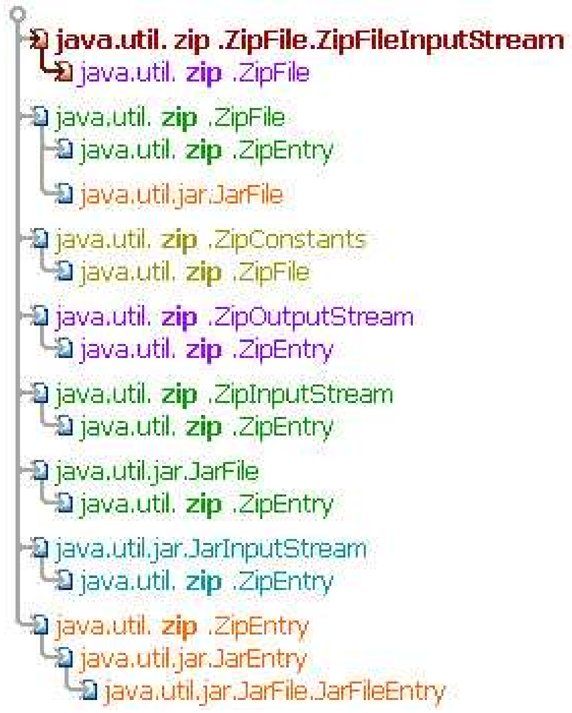
AUTOCODE Found 18 pages in 11 trails.
Search took 0.35 seconds.

```
java.util. zip .ZipFile.ZipFileInputStream
  java.util. zip .ZipFile
java.util. zip .ZipFile
  java.util. zip .ZipEntry
  java.util.jar.JarFile
java.util. zip .ZipConstants
  java.util. zip .ZipFile
java.util. zip .ZipOutputStream
  java.util. zip .ZipEntry
java.util. zip .ZipInputStream
  java.util. zip .ZipEntry
java.util.jar.JarFile
  java.util. zip .ZipEntry
java.util.jar.JarInputStream
  java.util. zip .ZipEntry
java.util. zip .ZipEntry
  java.util.jar.JarEntry
    java.util.jar.JarFile.JarFileEntry
```

NavigationZone © 2000-2002

```java
/*
 * @(#)ZipFile.java 1.56 02/05/29
 *
 * Copyright 2002 Sun Microsystems, Inc. All rights reserved.
 * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 */

package java.util.zip;

import java.io.InputStream;
import java.io.IOException;
import java.io.EOFException;
import java.io.File;
import java.util.Vector;
import java.util.Enumeration;
import java.util.NoSuchElementException;
import java.security.AccessController;

/**
 * This class is used to read entries from a zip file.
 *
 * @version   1.56, 05/29/02
 * @author  David Connelly
 */
public
class ZipFile implements ZipConstants {
    private long jzfile;  // address of jzfile data
    private String name;  // zip file name
    private int total;    // total number of entries

    private static final int STORED = ZipEntry.STORED;
    private static final int DEFLATED = ZipEntry.DEFLATED;

    /**
     * Mode flag to open a zip file for reading.
     */
    public static final int OPEN_READ = 0x1;
```

Figure 1. Results for the query "zip" on the JDK 1.4 source code.



```
java.util. zip .ZipFile.ZipFileInputStream
  java.util. zip .ZipFile
java.util. zip .ZipFile
  java.util. zip .ZipEntry
  java.util.jar.JarFile
java.util. zip .ZipConstants
  java.util. zip .ZipFile
java.util. zip .ZipOutputStream
  java.util. zip .ZipEntry
java.util. zip .ZipInputStream
  java.util. zip .ZipEntry
java.util.jar.JarFile
  java.util. zip .ZipEntry
java.util.jar.JarInputStream
  java.util. zip .ZipEntry
java.util. zip .ZipEntry
  java.util.jar.JarEntry
    java.util.jar.JarFile.JarFileEntry
```

Figure 2. Trails returned for the query "zip" on the JDK 1.4 source code.

This communicates with an external parser and constructs the five coupling graphs. Given the graphs of related classes, the navigation engine can be used to construct trails. This works in 4 stages. The first stage is to calculate scores (using $tf.idf$) for each of the classes matching one or more of the keywords in the query, and isolate a small number of these for future expansion, by combining these score with a metric called *potential gain* [4, 3]. The second stage is to construct the trails using the *Best Trail* algorithm [4]. This builds trails using a probabilistic best-first traversal. The third stage involves filtering the trails to remove redundant information. In the fourth and final stage, the navigation engine computes small summaries of each class and formats the results for display in a web browser. Jason Shattu's Java2HTML[1] is used to present the source code, as it provides effective syntax highlighting, has a public API and makes links to both Javadocs and between classes in source code.

3 Future Work

Object Oriented languages gain particular benefit from the mapping between classes and Web pages. It is intended that AutoCode be extended to support both C++ and C#. It is also hoped that the system can be extended to allow personalized results so that programmers working on a particular field have query results tailored to their needs.

References

[1] Vannevar Bush. As we may think. *Atlantic Monthly*, 76:101–108, 1945.

[2] Siegfried Reich, Leslie Carr, David De Roure, and Wendy Hall. Where have you been from here? : Trails in hypertext systems. *ACM Computing Surveys*, 31(4), December 1999.

[3] Richard Wheeldon and Steve Counsell. Making refactoring decisions in large-scale java systems: an empirical stance. *Computing Research Repository*, cs.SE/0306098, June 2003.

[4] Richard Wheeldon and Mark Levene. The best trail algorithm for adaptive navigation in the world-wide-web. *Computing Research Repository*, cs.DS/0306122, June 2003.

[1] http://java2html.com/