SAND REPORT
SAND2001-3094
Unlimited Release
Printed November 2001

Source Code Assurance Tool: An Implementation

Philip L. Campbell, Juan Espinoza Jr.

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

,

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2001-3094
Unlimited Release
Printed November 2001

Source Code Assurance Tool: An Implementation

Philip L. Campbell
Networked Systems Survivability & Assurance Department

Juan Espinoza Jr.
Cryptography & Information Systems Surety Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87175-0785
{plcampb, jespino}@sandia.gov

Abstract

We present the tool we built as part of a Laboratory Directed Research and Development (LDRD) project. This tool consists of a commercially-available, graphical editor front-end, combined with a back end “slicer.”

The significance of the tool is that it shows how to slice across system components. This is an advance from slicing across program components.

Keywords: slicing, program understanding, data flow, control flow, program dependence.

Table of Contents

List of Figures

1 Introduction

The Source Code Assurance Tool is a Laboratory Directed Research and Development (LDRD) effort. This report presents material on the tool that we built. The larger context of the research is available elsewhere. [2]

The purpose of the research is to develop a tool that can "slice" across system components. [1] A slice, based on a statement, S, of a computer program, P, is the set of statements in P that are the transitive closure of statements that are data- or control-flow dependent on S. When slicing was first introduced, the focus was on slicing within functions. A commercial product, CodeSurfer, has expanded that ability to be able to slice on entire programs. Our research is to take that one step further and slice on independent programs that constitute a system. For example, if program A generates a file that is read by program B, and if neither A nor B are given any indication of the other's identity, then it is not possible to slice across them because the semantics do not exist in the programs. The system, as such, exists in some other program or human operator. The intent of our tool is to enable slicing across systems.

We built a tool that combines a commercially-available graphical editor, Tomahawk, from Tom Sawyer Software [4] and a slicing engine. The intent of the tool is to enable slicing across system components, as opposed to within a single executable image.

2 Model

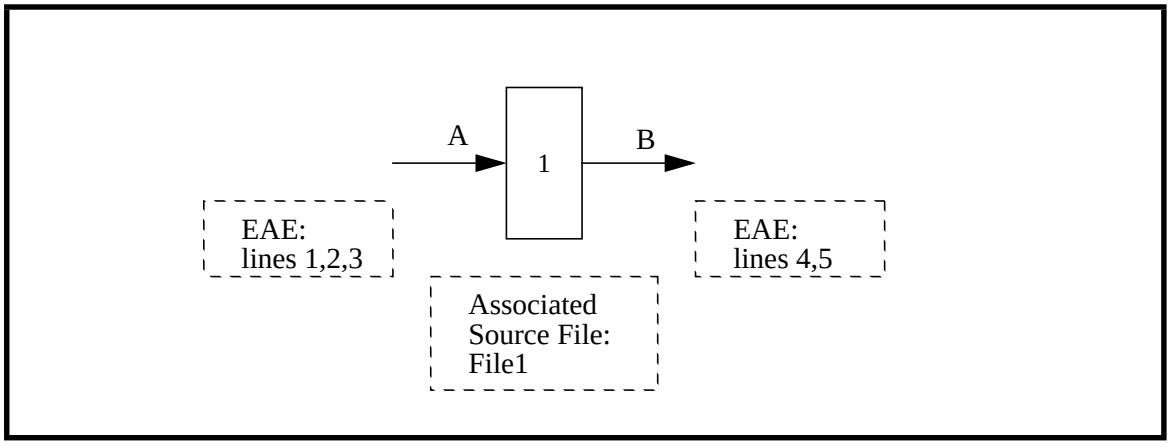
The tool is based on the following model. A system is assumed to consist of nodes (executable elements) and arcs (communication elements).

For each node there is associated exactly one file consisting of source code, divided into lines. For each arc there is exactly one source node and exactly one target node. Each arc has an "edge association element" (EAE) for the arc's source node and a second and distinct EAE for the arc's target node. The EAE for the arc's source node identifies the source code lines which, if "active," will cause output to be generated on the arc. Similarly, the EAE for the arc's target node identifies the source code lines which will become activated when output is generated on the arc. In this way the pair of EAE's associated with an arc associate lines of source code in one file with lines of source code in a second file.

Figure 1 shows the system elements discussed above. The Node, labeled "1" of the Figure has one incoming arc, labelled "A," and one outgoing arc, labelled "B." The source code file associated with Node 1 is named "File1." There is also dependence information associated with File1. This dependence information answers the following question for any given line, X, of the file: if line X is given control, what other lines are subsequently given control? The source code lines of File1 associated with arc A are lines 1, 2, and 3. The source code lines of File1

associated with arc B are lines 4 and 5.

Figure 1. System Elements.

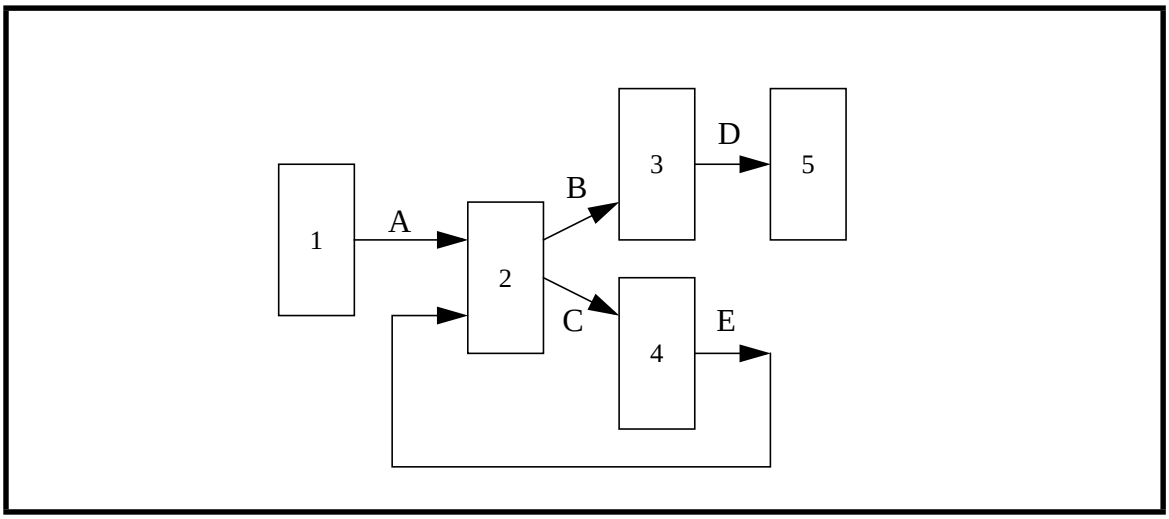


When input is received on arc A, lines 1, 2, and 3 of File1 are considered “active.” The slicing engine then uses the dependence information for File1 and performs a transitive closure, starting with lines 1, 2, and 3. In other words, the slice engine determines all of the lines of File1 that will be given control if lines 1, 2, and 3 are given control. For explanatory purposes let the transitive closure be the set S of source code lines. The slicing engine then determines, for each outgoing arc, if the source code lines in the EAE for that arc intersect set S. If they do, then that arc is considered active and output is generated on it.

Figure 2 shows a simple system, with nodes 1 through 5 and arcs A through E. The user first marks nodes as “active,” then the user starts the slice. Let us assume that the user marks only node 1 as active and that every active node generates output on all of its arcs. When the slice begins, output is generated on arc A, making node 2 active. Node 2, in turn, makes nodes 3 and 4 active. Node 3 makes node 5 active. Node 5 generates no output, so this branch of the slice ends here. Node 4 sends output to node 2 but since node 2 is already active, node 2 does not generate another output, ending this branch of the slice. Since this was the last branch of the slice, the

slice terminates.

Figure 2. A Simple System.



The system shown in Figure 2 is simple primarily because of the low degree of each node. As the degree increases, the possibilities increase for the complexity of the function performed by a given node.

Tomahawk is a sample graph editor provided by Tom Sawyer Software, Inc., the makers of the Graph Editor Toolkit and Graph Layout Toolkit. Using Tomahawk, the user can create nodes and arcs, label them and re-arrange them on the screen. To accommodate the slicing engine we added menu items to Tomahawk that allow the user to specify the following:

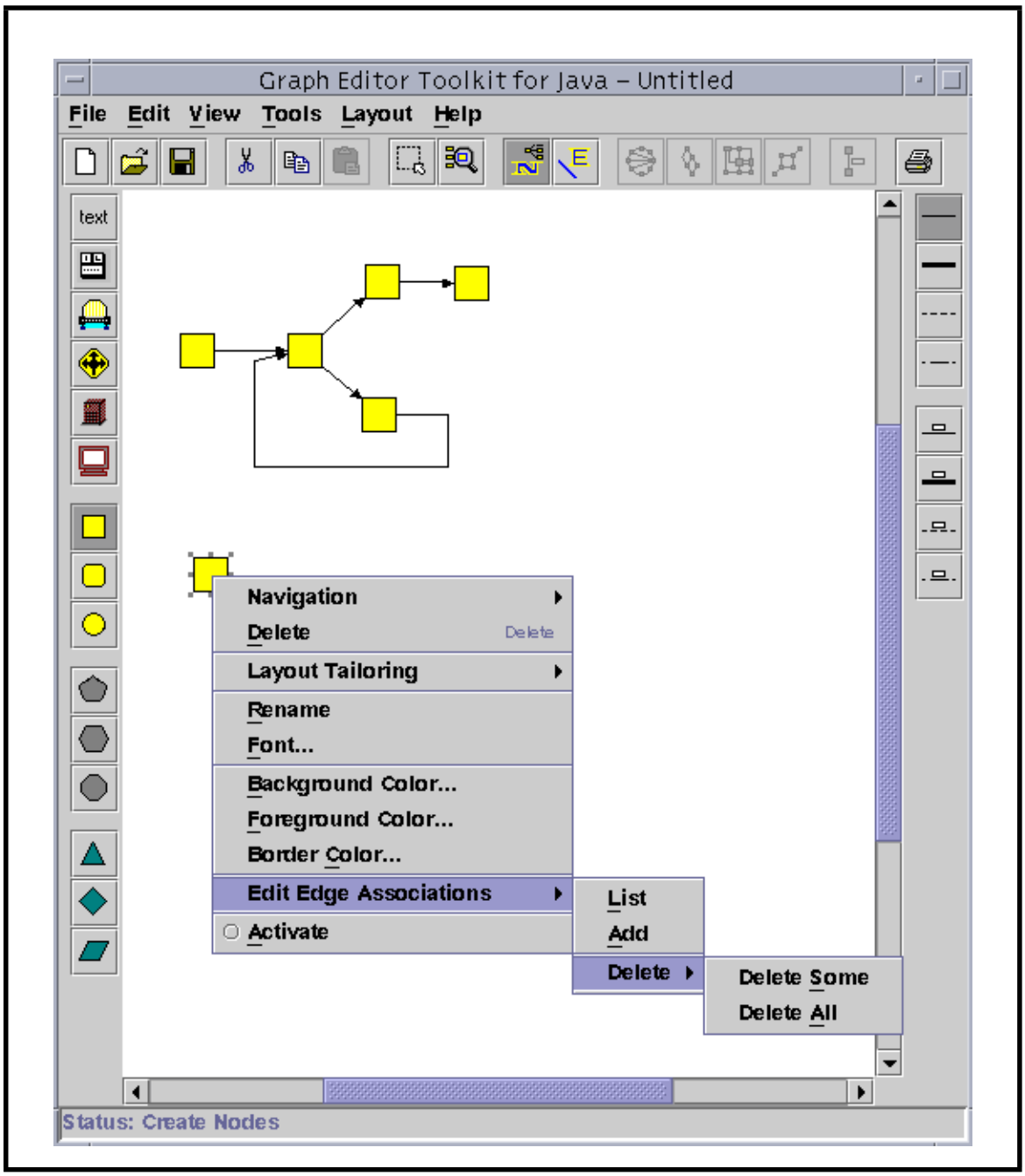
- the source file associated with each node,
- the EAE for each arc (one EAE for the node that is the source of the arc and a second EAE for the node that is the target of the arc),
- the active nodes,
- that a slice is to begin.

When a slice begins, the slicing engine computes the slice, following the procedure outlined above, changing the appearance of each arc and node that is activated.

The connection between the front-end graph editor and back-end slice engine required only a few additions to the graph editor menus. To the background menu we added two items: “delete all eae’s,” and “slice.” To the menu for each node we added two items: “edit eae’s,” and “activate.” The former, “edit eae’s,” has submenus that enable the user to add, delete, and list the

eae's for the current node. Figure 3 shows a screenshot of some of the menu items.

Figure 3. A Screenshot.



3 Dependence Graphs and Slicing

The slice engine is written in Java and is shown in Appendix A. In this section we explain two of

the fundamental parts of the engine: how the slice engine uses dependence graphs and how it slices.

3.1 Dependence Graphs

For the slice engine, dependence graphs are ASCII files with the grammar, described here in BNF:

dependence_graph :: input_line | input_line dependence_graph
input_line :: head_number : linenumber_set
head_number :: linenumber
linenumber_set :: linenumber | linenumber linenumber_set
linenumber :: integer
integer :: non_zero_digit | non_zero_digit any_integer
non_zero_digit :: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
any_integer :: any_digit | any_digit any_integer
any_digit :: 0 | non_zero_digit

So, for example, the file

3 : 2 5
5 : 6 3 9

Indicates that line numbers 2 and 5 are dependent on line 3. That is, if line 3 is active, then lines 2 and 5 will also be active. Similarly, lines 6, 3, and 9 are dependent on line 5. So if line 5 is active, then lines 6, 3, and 9 will also be active.

Note that this structure is based on line numbers, not statements.

The slice engine requires a dependence graph for each source code file (and the engine requires exactly one source code file for each node).

It is up to the user to construct the dependence graph information. This could be automated by the use of a tools such as CodeSurfer from GrammaTech. [3] However, the generality of the slice engine we have constructed enables slicing on regular text files—the files do not have to be written in a computer language such as C or Java. This enables the user to specify the functionality of a given node in a natural language.

References

[1] Valdis Berzins, ed., “Sotfware Merging and Slicing.” IEEE Computer Society Press. Los Alamitos, CA. 1995. (This collection includes the seminal 1984 paper by Mark Weiser entitled “Program Slicing.”)

[2] Juan Espinoza, Philip L. Campbell, “Source Code Assurance Tool: LDRD Final Report.” SAND2001-3091. Sandia National Laboratories, Albuquerque, NM. Printed October 2001.

[3] GrammaTech, Inc. home page: http://www.grammatech.com/grammatech.html.

[4] Tom Sawyer Software, Inc., home page: http://www.tomsawyer.com/.

Appendix A Slice Engine

This Appendix shows the Java source code for the slice engine.

Included here are the files that constitute the slice engine:

- lc.java --the list controller
- nl.java --the node list
- el.java --the edge list
- nle.java --a node list element
- ele.java --an edge list element
- eae.java --an edge association element
- dgr.java --dependence graph reader
- availNumber.java --provides unique integers

The slice engine consists of a single object of type list controller class, and many objects of the following types: node list, edge list, node list element, edge list element, and edge association element.

The object of type list controller class contains a list of nodes and a list of edges. The node list class contains a list of node list elements. Each node list element has a nodeNumber, for example. The edge list class contains a list of edge list elements. Each edge list element has an edgeNumber, for example. Each edge list element also has a list of edge association elements (EAEs).

All method invocation comes through the list controller. The list controller passes the message on to the appropriate node or edge list, which in turn may pass it on to a particular node or edge.

The list controller class (file “lc.java”) is the top level class for the slice engine. The list controller contains a list of nodes and a list of edges.

The node list class (file “nl.java”) contains a list of nodes and methods that operate on them. For example, addNode, removeNode, doesNodeExist, loadDg (i.e., load a dependence graph) for a given node, and slice. The slice method gives every node an opportunity to slice at most once.

The edge list class (file “el.java”) contains a list of edges and methods that operate on them. For example, addEdge, removeEdge, removeAdjacentEdges, isDuplicateEdge, doesEdgeExist, isEdgeAdjacentToNode, isEdgeActive, and setEdge (i.e., set an edge to active or inactive).

The node list element class (file “nle.java”) contains the data for a particular node. That data includes a node number, a list of incoming EAEs, a list of outgoing EAEs, and a dependence graph for the node. This class contains the performSlice method that performs the slice function on the given node.

The edge list element class (file “ele.java”) contains the data and methods for a given edge. The data includes edgeNumber, sourceNodeNumber, and targetNodeNumber, and finally isActive. The methods include getEdgeNumber, getSourceNodeNumber, getTargetNodeNumber, isActive, and setEdge (i.e., set isActive).

The edge association element (file “eae.java”) contains the data for an edge association element. The data includes an eaeNumber, a nodeNumber, an edgeNumber, and a startLineNumber and an endLineNumber. The last two fields indicate a set of contiguous source code lines of the file associated with nodeNumber.

A.1 lc.java

```
package slice_engine;
import java.util.*;
// list controller;
// creates and controls the node list and edge list;
public class lc
{
      private final nl nodeList;
      private final el edgeList;
      static lc instance = null;
      // constructor
      public lc ()
      {
            if ( instance != null )
                  System.exit(1);
            instance = this;
            nodeList = new nl ();
            edgeList = new el ();
      }
      public void addNode( int nodeNumber )
            { nodeList.addNode ( nodeNumber, edgeList ); }
      public void removeNode( int nodeNumber )
      {
            edgeList.removeAdjacentEdges ( nodeNumber, nodeList );
            nodeList.removeNode ( nodeNumber, edgeList );
      }
      public void printNodeList()
            { nodeList.printNodeList( edgeList ); }
      public void activateLine ( int nodeNumber, int line )
            { nodeList.activateLine ( nodeNumber, line ); }
      public void slice()
            { nodeList.slice( edgeList ); }
      public boolean doesNodeExist( int nodeNumber )
            { return ( nodeList.doesNodeExist ( nodeNumber )); }
      public void loadDg ( int nodeNumber, String filename )
            { nodeList.loadDg ( nodeNumber, filename ); }
      public void addEae(
            int nn, int en, boolean isIn, int sln, int eln )
```

```
            { nodeList.addEae ( edgeList, nn, en, isIn, sln, eln); }
      public void removeEae ( int nn, int en, boolean isIncoming )
            { nodeList.removeEae( nn, en, isIncoming ); }
      public void addEdge( int en, int snn, int tnn )
            { edgeList.addEdge ( nodeList, en, snn, tnn); }
      public void removeEdge( int edgeNumber )
            { edgeList.removeEdge ( edgeNumber, nodeList ); }
      public boolean doesEdgeExist( int edgeNumber )
            { return ( edgeList.doesEdgeExist ( edgeNumber )); }
      public boolean isDuplicateEdge( int snn, int tnn )
            { return ( edgeList.isDuplicateEdge (snn, tnn)); }
      public boolean isEdgeActive ( int edgeNumber )
            { return ( edgeList.isEdgeActive ( edgeNumber )); }
      public int getSourceNodeNumber( int edgeNumber )
            { return ( edgeList.getSourceNodeNumber ( edgeNumber )); }
      public int getTargetNodeNumber( int edgeNumber )
            { return ( edgeList.getTargetNodeNumber ( edgeNumber )); }
}
```

A.2 nl.java

```
package plc_engine;
import java.util.*;
// node list, containing node list elements (nle);
// each nle consists of two lists of edge association elements,
// (eae), one for incoming edges and one for outgoing edges;
// each eae contains
//    a unique identifier (an integer),
//    whether or not the edge is incoming to the node,
//    the edge number,
//    and the start & end line numbers associated with the edge;
// the code permits the addition of nodes in any order;
// the code permits the removal of nodes whenever;
//---------------------------------------------------------------------
class nl
{
private final TreeMap nodeList = new TreeMap();
static nl instance = null;
//---------------------------------------------------------------------
// constructor
nl ()
{
      if ( instance != null )
      {
            System.out.println (“error: cannot create new object of type nl”);
            System.out.println (“\tan object of that type already exists”);
            System.exit(1);
      }
}
//---------------------------------------------------------------------
public void addNode( int nodeNumber, el edgeList )
```

```
{
      if ( nodeNumber < 0 )
      {
            System.out.println (“nl: addNode: fatal error”);
            System.out.println (“\tnode “ + nodeNumber + “ < 0”);
            System.exit(1);
      }
      if ( doesNodeExist ( nodeNumber ))
      {
            System.out.println (“nl: addNode: fatal error”);
            System.out.println (“\tnode “ + nodeNumber + “ already exists”);
            System.exit(1);
      }
      // create new node association list
      nle n = new nle (nodeNumber);
      Integer a = new Integer(nodeNumber);
      nodeList.put(a, n);
}
//---------------------------------------------------------------------
public void removeNode( int nodeNumber, el edgeList )
{
      if ( nodeList.isEmpty() )
      {
            System.out.println (“nl: removeNode: fatal error”);
            System.out.println (“\tattempting to remove node “
            + nodeNumber + “, but node list is empty”);
            System.exit(1);
      }
      fatalIfNotInList ( nodeNumber, “removeNode” );
      // we do not need to recycle the eaes because when we
      // remove a node, we first remove all of the adjacent
      // edges and their eaes, so nodeNumber should, at
      // this point, have no eaes;
      // halt if any eaes are present;
      nle a = getnle ( nodeNumber, “removeNode” );
      a.haltIfEaesPresent();
      Integer b = new Integer(nodeNumber);
      nodeList.remove(b);
}
//---------------------------------------------------------------------
public boolean doesNodeExist ( int nodeNumber )
{
      // cannot use getnle here: it would cause a loop!
      Integer a = new Integer(nodeNumber);
      nle b = (nle) nodeList.get(a);
      boolean c;
      if ( b == null )
            c = false;
      else
            c = true;
      return ( c );
```

```
}
//---------------------------------------------------------------------
private void fatalIfNotInList( int nodeNumber, String msg )
{
      if ( ! doesNodeExist (nodeNumber) )
      {
            System.out.println (“nl: “ + msg + “: fatal error”);
            System.out.println (“\tnode “ + nodeNumber + “ not in list”);
            System.exit(1);
      }
}
//---------------------------------------------------------------------
// this is how you set a line to active and get everything rolling;
public void activateLine ( int nodeNumber, int line )
{
      nle a = getnle ( nodeNumber, “activateLine”);
      a.activateLine ( line );
}
//---------------------------------------------------------------------
HashSet slicedNodesTreeSet = new HashSet ();
public void slice( el edgeList )
{
      if ( nodeList.isEmpty() )
      {
            System.out.println (“\t(node list is empty)”);
            return;
      }
      // give each node the opportunity to slice at most once;
      // this may take multiple passes through the list of nodes;
      // on each pass, skip the nodes that have already sliced;
      // the last pass is the one in which no node sliced;
      slicedNodesTreeSet.clear();
      boolean again = true;
      while ( again )
      {
            again = false;
            Set s = nodeList.keySet();
            Iterator a = s.iterator();
            while ( a.hasNext() )
            {
                  nle n = (nle) nodeList.get(a.next());
                  int nn = n.getNodeNumber();
                  Integer c = new Integer (nn);
                  if ( slicedNodesTreeSet.contains(c) )
                        continue;
                  if ( n.performSlice( edgeList ) )
                  {
                        again = true;
                        slicedNodesTreeSet.add(c);
                  }
            }
```

```
      }
}
//---------------------------------------------------------------------
private nle getnle ( int nn, String msg )
{
      fatalIfNotInList ( nn, msg );
      Integer a = new Integer(nn);
      return ( (nle) nodeList.get(a) );
}
//---------------------------------------------------------------------
// load in a new dependence graph, using filename;
// pass in eaeAvailList so that the current eaes for nodeNumber,
// if there are any, can be recycled;
public void loadDg ( int nodeNumber, String filename )
{
      nle a = getnle ( nodeNumber, “loadDg”);
      a.loadDg ( filename, eaeAvailList );
}
//---------------------------------------------------------------------
static availNumber eaeAvailList = new availNumber( “eae”);
public void addEae(
      el edgeList,
      int nn, int en, boolean isIn, int sln, int eln )
{
      int neae = eaeAvailList.getNumber();
      nle a = getnle ( nn, “addEae” );
      a.addEae ( eaeAvailList, neae, en, isIn, sln, eln);
}
//---------------------------------------------------------------------
// nodeNumber and some other node, Z, share edge edgeNumber;
// edge edgeNumber is about to be removed, in anticipation of the
// removal of node Z;
// we now need to recycle the eae associated with the
// incoming or outgoing edge edgeNumber in node nodeNumber;
public void removeEae (
      int nodeNumber, int edgeNumber, boolean isIncoming )
{
      fatalIfNotInList ( nodeNumber, “removeEae” );
      nle a = getnle ( nodeNumber, “removeEae” );
      a.removeEae ( eaeAvailList, edgeNumber, isIncoming );
}
//---------------------------------------------------------------------
public int getStartLineNumber( int nn, boolean isIncoming, int en )
{
      nle a = getnle ( nn, “getStartLineNumber” );
      eae b = a.getEae ( en, isIncoming );
      int c = b.getStartLineNumber();
      return ( c );
}
//---------------------------------------------------------------------
public int getEndLineNumber( int nn, boolean isIncoming, int en )
```

```
{
      nle a = getnle ( nn, “getStartLineNumber” );
      eae b = a.getEae ( en, isIncoming );
      int c = b.getEndLineNumber();
      return ( c );
}
//---------------------------------------------------------------------
}
```

A.3 el.java

```
package slice_engine;
import java.util.*;
// maintain the edge list;
// eleList is a list of edge list elements (ele)
// we require that source and target node numbers be
// given when the edge is created;
// the source and target numbers may not be changed;
//---------------------------------------------------------------------
class el
{
private final TreeMap eleList = new TreeMap();
static el instance = null;
//---------------------------------------------------------------------
// constructor
el ()
{
      if ( instance != null )
      {
            System.out.println (“error: cannot create new object of type el”);
            System.out.println (“\tan object of that type alread exists”);
            System.exit(1);
      }
      instance = this;
}
//---------------------------------------------------------------------
// we require that both the source node number (snn) and the
// target node number (tnn) exist at the time of the edge’s
// creation; that is, nodes are created before edges;
// we also check to make sure that there is not already
// an edge between snn and tnn;
public void addEdge( nl nodeList, int en, int snn, int tnn )
{
      haltOnBadEdgeNumber (en);
      haltOnBadNodeNumbers ( nodeList, snn, tnn );
      haltOnDuplicateEdge( snn, tnn );
      if ( isInList ( en ))
      {
            System.out.println (“el: addEdge: fatal error”);
            System.out.println (“\tedge number “ + en + “ already exists”);
            System.exit(1);
```

```
	}
	// create new edge element
	ele e = new ele (en, snn, tnn);
	Integer a = new Integer(en);
	eleList.put(a, e);
}
//---------------------------------------------------------------------
// removing an edge also involves removing
// the associated eaes, if any;
public void removeEdge( int edgeNumber, nl nodeList )
{
	if ( eleList.isEmpty() )
	{
		System.out.println (“el: removeEdge: fatal error”);
		System.out.println (“\tedge list is empty”);
		System.out.println (“\t(attempting to remove edge “ + edgeNumber );
		System.exit(1);
	}
	haltOnBadEdgeNumber (edgeNumber);
	haltIfNotInList ( edgeNumber, “removeEdge” );
	Integer a = new Integer(edgeNumber);
	ele e = (ele) eleList.get (a);
	int snn = e.getSourceNodeNumber();
	int tnn = e.getTargetNodeNumber();
	// “false” implies the edge is outgoing from node snn;
	// remove eae on edge outgoing from node snn;
	nodeList.removeEae (snn, edgeNumber, false);
	// remove eae on edge incoming to node snn;
	nodeList.removeEae (tnn, edgeNumber, true);
	// now remove the edge;
	eleList.remove(a);
}
//---------------------------------------------------------------------
// we are about to remove node nn;
// remove all edges adjacent to nn;
public void removeAdjacentEdges ( int nn, nl nodeList )
{
	HashSet edgesToBeRemoved = new HashSet();
	Set s = eleList.keySet();
	Iterator a = (Iterator) s.iterator();
	while ( a.hasNext() )
	{
		Integer b = (Integer) a.next();
		int c = b.intValue();
		ele e = (ele) eleList.get (b);
		int snn = e.getSourceNodeNumber();
		int tnn = e.getTargetNodeNumber();
		if ( (snn == nn) || (tnn == nn ))
			edgesToBeRemoved.add(b);
	}
	// we have collected the edge numbers of
```

```
      // each edge that needs to be removed;
      // now we remove them;
      a = (Iterator) edgesToBeRemoved.iterator();
      while ( a.hasNext() )
      {
            Integer b = (Integer) a.next();
            int c = b.intValue();
            removeEdge(c, nodeList);
      }
}
//---------------------------------------------------------------------
private void haltOnBadEdgeNumber( int edgeNumber )
{
      if ( edgeNumber < 0 )
      {
            bad = true;
            System.out.println (“el: haltOnBadEdgeNumber: fatal error”);
            System.out.println (“\tedge “ + edgeNumber + “ < 0”);
            System.exit(1);
      }
}
//---------------------------------------------------------------------
private void haltOnBadNodeNumbers( nl nodeList, int nn0, int nn1 )
{
      boolean bad = false;
      if ( ! nodeList.doesNodeExist ( nn0 ))
      {
            bad = true;
            System.out.println (“(el: haltOnBadNodeNumbers)”);
            System.out.println (“fatal error:”);
            System.out.println (“\ttrying to create edge from node “
            + nn0 + “ to node “ + nn1);
            System.out.println (“\tbut node “ + nn0 + “ does not exist”);
      }
      if ( ! nodeList.doesNodeExist ( nn1 ))
      {
            bad = true;
            System.out.println (“(el: haltOnBadNodeNumbers)”);
            System.out.println (“fatal error:”);
            System.out.println (“\ttrying to create edge from node “
            + nn0 + “ to node “ + nn1);
            System.out.println (“\tbut node “ + nn1 + “ does not exist”);
            System.out.println (“\tnode “ + nn1 + “ does not exist either”);
      }
      if ( bad )
            System.exit(1);
}
//---------------------------------------------------------------------
// halt if duplicate edge
// (one with same source and destination nodes)
// is attempting to be created;
```

```
private void haltOnDuplicateEdge( int xsnn, int xtnn )
{
      Set s = eleList.keySet();
      Iterator a = (Iterator) s.iterator();
      while ( a.hasNext() )
      {
            Integer b = (Integer) a.next();
            int c = b.intValue();
            ele e = (ele) eleList.get (b);
            int snn = e.getSourceNodeNumber();
            int tnn = e.getTargetNodeNumber();
            if ( (snn == xsnn) && (tnn == xtnn ))
            {
                  System.out.println (“el: haltOnDuplicateEdge: fatal error”);
                  System.out.println (“\tattempting to create new edge”);
                  System.out.println
                  (“\tbut existing edge “ + e.getEdgeNumber() +
                  “ has same source & destination nodes”);
                  System.out.println (“\t(source node “ + snn + “)”);
                  System.out.println (“\t(target node “ + tnn + “)”);
                  System.exit(1);
            }
      }
}
//---------------------------------------------------------------------
// return true if duplicate edge exists, false otherwise;
// (a duplicate edge is one with matching source and destination
// node numbers);
public boolean isDuplicateEdge( int xsnn, int xtnn )
{
      Set s = eleList.keySet();
      Iterator a = (Iterator) s.iterator();
      while ( a.hasNext() )
      {
            Integer b = (Integer) a.next();
            int c = b.intValue();
            ele e = (ele) eleList.get (b);
            int snn = e.getSourceNodeNumber();
            int tnn = e.getTargetNodeNumber();
            if ( (snn == xsnn) && (tnn == xtnn ))
            {
                  return ( true );
            }
      }
      return ( false );
}
//---------------------------------------------------------------------
private boolean isInList( int edgeNumber )
{
      // cannot use getnle here: would result in loop!
      Integer a = new Integer(edgeNumber);
```

```
      ele b = (ele) eleList.get(a);
      boolean c = (b != null);
      return ( c );
}
//---------------------------------------------------------------------
private void haltIfNotInList( int edgeNumber, String msg )
{
      if ( ! isInList (edgeNumber) )
      {
            System.out.println (“el: “ + msg + “: fatal error”);
            System.out.println (“\tedge “ + edgeNumber + “ not in list”);
            System.exit(1);
      }
}
//---------------------------------------------------------------------
private ele getEleHard ( int en, String msg )
{
      haltIfNotInList ( en, msg );
      Integer a = new Integer(en);
      return ( (ele) eleList.get(a) );
}
//---------------------------------------------------------------------
public boolean doesEdgeExist( int edgeNumber )
{
      return ( isInList ( edgeNumber ));
}
//---------------------------------------------------------------------
public boolean isEdgeActive( int edgeNumber )
{
      ele e = getEleHard( edgeNumber, “isEdgeActive”);
      return ( e.getIsActive () );
}
//---------------------------------------------------------------------
public void setEdge( int edgeNumber, boolean setting )
{
      ele e = getEleHard ( edgeNumber, “setEdgeInactive” );
      e.setEdge ( setting );
}
//---------------------------------------------------------------------
public int getSourceNodeNumber( int edgeNumber )
{
      ele a = getEleHard ( edgeNumber, “getSourceNodeNumber” );
      return ( a.getSourceNodeNumber());
}
//---------------------------------------------------------------------

public int getTargetNodeNumber( int edgeNumber )
{
      ele a = getEleHard ( edgeNumber, “getTargetNodeNumber” );
      return ( a.getTargetNodeNumber());
}
```

```
//---------------------------------------------------------------------
}
```

A.4 nle.java

```
package slice_engine;
import java.util.*;
// node list, containing node list elements (nle);
// each nle consists of two lists of edge association elements,
// (eae), one for incoming edges and one for outgoing edges;
// each eae contains
//    a unique identifier (an integer),
//    whether or not the edge is incoming to the node,
//    the edge number,
//    and the start & end line numbers associated with the edge;
//---------------------------------------------------------------------
class nl
{
private final TreeMap nodeList = new TreeMap();
static nl instance = null;
//---------------------------------------------------------------------
// constructor
nl ()
{
      if ( instance != null )
      {
            System.out.println (“error: cannot create new object of type nl”);
            System.out.println (“\tan object of that type already exists”);
            System.exit(1);
      }
}
//---------------------------------------------------------------------
public void addNode( int nodeNumber, el edgeList )
{
      if ( nodeNumber < 0 )
      {
            System.out.println (“nl: addNode: fatal error”);
            System.out.println (“\tnode “ + nodeNumber + “ < 0”);
            System.exit(1);
      }
      if ( doesNodeExist ( nodeNumber ))
      {
            System.out.println (“nl: addNode: fatal error”);
            System.out.println (“\tnode “ + nodeNumber + “ already exists”);
            System.exit(1);
      }
      // create new node association list
      nle n = new nle (nodeNumber);
      Integer a = new Integer(nodeNumber);
      nodeList.put(a, n);
}
```

```
//---------------------------------------------------------------------
public void removeNode( int nodeNumber, el edgeList )
{
      if ( nodeList.isEmpty() )
      {
            System.out.println ("nl: removeNode: fatal error");
            System.out.println ("\tattempting to remove node "
            + nodeNumber + ", but node list is empty");
            System.exit(1);
      }
      fatalIfNotInList ( nodeNumber, "removeNode" );
      // we do not need to recycle the eaes because when we
      // remove a node, we first remove all of the adjacent
      // edges and their eaes, so nodeNumber should, at
      // this point, have no eaes;
      // halt if any eaes are present;
      nle a = getnle ( nodeNumber, "removeNode" );
      a.haltIfEaesPresent();
      Integer b = new Integer(nodeNumber);
      nodeList.remove(b);
}
//---------------------------------------------------------------------
public boolean doesNodeExist ( int nodeNumber )
{
      // cannot use getnle here: it would cause a loop!
      Integer a = new Integer(nodeNumber);
      nle b = (nle) nodeList.get(a);
      boolean c;
      if ( b == null )
            c = false;
      else
            c = true;
      return ( c );
}
//---------------------------------------------------------------------
private void fatalIfNotInList( int nodeNumber, String msg )
{
      if ( ! doesNodeExist (nodeNumber) )
      {
            System.out.println ("nl: " + msg + ": fatal error");
            System.out.println ("\tnode " + nodeNumber + " not in list");
            System.exit(1);
      }
}
//---------------------------------------------------------------------
// this is how you set a line to active and get everything rolling;
public void activateLine ( int nodeNumber, int line )
{
      nle a = getnle ( nodeNumber, "activateLine");
      a.activateLine ( line );
}
```

```
//---------------------------------------------------------------------
HashSet slicedNodesTreeSet = new HashSet ();
public void slice( el edgeList )
{
      if ( nodeList.isEmpty() )
      {
            System.out.println (“\t(node list is empty)”);
            return;
      }
      // give each node the opportunity to slice at most once;
      // this may take multiple passes through the list of nodes;
      // on each pass, skip the nodes that have already sliced;
      // the last pass is the one in which no node sliced;
      slicedNodesTreeSet.clear();
      boolean again = true;
      while ( again )
      {
            again = false;
            Set s = nodeList.keySet();
            Iterator a = s.iterator();
            while ( a.hasNext() )
            {
                  nle n = (nle) nodeList.get(a.next());
                  int nn = n.getNodeNumber();
                  Integer c = new Integer (nn);
                  if ( slicedNodesTreeSet.contains(c) )
                        continue;
                  if ( n.performSlice( edgeList ) )
                  {
                        again = true;
                        slicedNodesTreeSet.add(c);
                  }
            }
      }
}
//---------------------------------------------------------------------
private nle getnle ( int nn, String msg )
{
      fatalIfNotInList ( nn, msg );
      Integer a = new Integer(nn);
      return ( (nle) nodeList.get(a) );
}
//---------------------------------------------------------------------
// load in a new dependence graph, using filename;
// pass in eaeAvailList so that the current eaes for nodeNumber,
// if there are any, can be recycled;
public void loadDg ( int nodeNumber, String filename )
{
      nle a = getnle ( nodeNumber, “loadDg”);
      a.loadDg ( filename, eaeAvailList );
}
```

```
//---------------------------------------------------------------------
static availNumber eaeAvailList = new availNumber( “eae”);
public void addEae( el edgeList, int nn, int en, boolean isIn, int sln, int eln )
{
      int neae = eaeAvailList.getNumber();
      nle a = getnle ( nn, “addEae” );
      a.addEae ( eaeAvailList, neae, en, isIn, sln, eln);
}
//---------------------------------------------------------------------
// nodeNumber and some other node, Z, share edge edgeNumber;
// edge edgeNumber is about to be removed, in anticipation of the
// removal of node Z;
// we now need to recycle the eae associated with the
// incoming or outgoing edge edgeNumber in node nodeNumber;

public void removeEae ( int nodeNumber, int edgeNumber, boolean isIncoming )
{
      fatalIfNotInList ( nodeNumber, “removeEae” );
      nle a = getnle ( nodeNumber, “removeEae” );
      a.removeEae ( eaeAvailList, edgeNumber, isIncoming );
}
//---------------------------------------------------------------------
public int getStartLineNumber( int nn, boolean isIncoming, int en )
{
      nle a = getnle ( nn, “getStartLineNumber” );
      eae b = a.getEae ( en, isIncoming );
      int c = b.getStartLineNumber();
      return ( c );
}
//---------------------------------------------------------------------
public int getEndLineNumber( int nn, boolean isIncoming, int en )
{
      nle a = getnle ( nn, “getStartLineNumber” );
      eae b = a.getEae ( en, isIncoming );
      int c = b.getEndLineNumber();
      return ( c );
}
//---------------------------------------------------------------------
}
```

A.5 ele.java

```
package slice_engine;
import java.util.*;
// edge list element;
public class ele
{
      private boolean isActive = false;
      private final int
            edgeNumber,
            sourceNodeNumber,
```

```
            targetNodeNumber;
      // constructor
      ele( int en, int snn, int tnn )
      {
            edgeNumber  = en;
            sourceNodeNumber = snn;
            targetNodeNumber = tnn;
      }
      public void setEdge( boolean s ) { isActive = s; }
      public boolean getIsActive() { return isActive; }
      public int getEdgeNumber() { return edgeNumber; }
      public int getSourceNodeNumber() { return sourceNodeNumber; }
      public int getTargetNodeNumber() { return targetNodeNumber; }
}
```

A.6 eae.java

```
package slice_engine;
import java.util.*;

// edge association element

class eae
{
      private final boolean isIncoming;
      private final int eaeNumber,
                        nodeNumber,
                        edgeNumber,
                        startLineNumber,
                        endLineNumber;
      eae( int eaen, int nn, int en, boolean isIn, int sln, int eln )
      {
            eaeNumber= eaen;
            nodeNumber      = nn;
            edgeNumber      = en;
            isIncoming      = isIn;
            startLineNumber = sln;
            endLineNumber   = eln;
      }
      public int getEaeNumber() { return eaeNumber; }
      public boolean getIsIncoming() { return isIncoming; }
      public int getEdgeNumber() { return edgeNumber; }
      public int getStartLineNumber() { return startLineNumber; }
      public int getEndLineNumber  () { return endLineNumber;   }
}
```

A.7 dgr.java

```
package slice_engine;
import java.util.*;
```

```
// data graph reader
class dgr
{
      private TreeMap dgTreeMap;
      private String filename;
      private int lineNumber = 0;
//---------------------------------------------------------------
      // constructor
      dgr() { }
//---------------------------------------------------------------
      public TreeMap createDg ( String xfilename )
      {
            filename = xfilename;
            dgTreeMap = new TreeMap();
            try
            {
                  BufferedReader in = new BufferedReader
                        (new FileReader (filename));
                  String line;
                  while ((line = in.readLine()) != null )
                  {
                        lineNumber++;
                        parseLine ( line );
                  }
                  // add final item
                  finishParsing ( line );
                  in.close();
            }
            catch (IOException ioe)
            {
                  System.out.println("dgr: fatal error:");
                  System.out.println("\tIO error: ");
                  System.out.println("\t" + ioe);
                  System.exit(1);
            }
            return ( dgTreeMap );
      }
//---------------------------------------------------------------
      // syntax of input file (whitespace delimited):
      //
      // input_line :: head_number ":" linenumber_set
      // head_number :: linenumber
      // linenumber_set :: linenumber | linenumber_set
      //
      // for example:
      // 3 : 2 5
      // 5 : 6 3 9
      // note that newline characters can appear anywhere
      // there is whitespace;
      // it is illegal to have duplicate head_numbers;
      // the variable "state" below persists across input lines
```

```
      //
      // explanation of states (“#” denotes a linenumber):
      //
      // state    input       action            next state
      // -----    -----       ------            ----------
      //
      // 0        :           error             -
      // 0        #           nextDg            1
      //
      // 1        :                             2
      // 1        #           error             -
      //
      // 2        :           error             -
      // 2        #           save              3
      //
      // 3        :           nextDg            4
      // 3        #           addToDg, save     3
      //
      // 4        :           error             -
      // 4        #           save #            3
      // state transitions
      //    (syntax: <state>/<input>)
      //    (all transitions not shown are errors):
      //    0/# ---> 1/: ---> 2/# -+--+-> 3/: ---> 4/# --+
      //                           |  |                  |
      //                           ^  V                  V
      //                           |  |                  |
      //                           |  +-> 3/# ---+       |
      //                           |             |       |
      //                           +-------<-----+-------+
//-------------------------------------------------------------
      private int state = 0;
      private final String colonString = “:”;
      private String previousToken;
      private int elements = 0;
      private void parseLine( String line )
      {
            StringTokenizer s = new StringTokenizer( line );
            int i = 0;
            while ( s.hasMoreTokens() )
            {
                  String currentToken = s.nextToken();
                  i++;
                  int k = currentToken.compareTo(colonString);
// switch statement moved to left margin to give space at right;
switch ( state )
{
      // k == 0 when token == colon;
      case 0:
            if ( k == 0 )
            {
```

```
            System.out.println( “dgr: error: “);
            System.out.println (“\tmissing initial head number”);
            System.exit(1);
      }
      else
      {
            // we have initial head number;
            nextDg (currentToken, line);
            state = 1;
      }
      break;
case 1:
      if ( k == 0 )
      {
            // we have initial colon;
            state = 2;
      }
      else
      {
            System.out.println( “dgr: error: “);
            System.out.println (“\tmissing initial colon”);
            System.exit(1);
      }
      break;
case 2:
      if ( k == 0 )
      {
            System.out.println( “dgr: error: “);
            System.out.println(“\tduplicate colon”);
            System.exit(1);
      }
      else
      {
            // we have initial linenumber;
            previousToken = currentToken;
            state = 3;
      }
      break;
case 3:
      if ( k == 0 )
      {
            // we have colon;
            if ( elements <= 0 )
            {
                  System.out.println( “dgr: error: “);
                  System.out.println (“\thead number but no elements”);
                  System.exit(1);
            }
            elements = 0;
            nextDg (previousToken, line);
            state = 4;
```

```
            }
            else
            {
                  elements++;
                  addToDg (previousToken, line);
                  previousToken = currentToken;
            }
            break;
      case 4:
            // this state catches multiple adjacent colons,
            // and it re-loads previousToken after state
            // 3 used its contents for the new head number;
            if ( k == 0 )
            {
                  System.out.println( “dgr: error: “);
                  System.out.println(“\tduplicate colon”);
                  System.exit(1);
            }
            else
            {
                  // and now to load previousToken again;
                  previousToken = currentToken;
                  state = 3;
            }
            break;
      default:
            System.out.println( “dgr: internal error: “);
            System.out.println(“\tstate out of range: “ + state);
            System.exit(1);
}
            }   //while
      }
//-------------------------------------------------------------
      // to get the last line number into the dg;
      private void finishParsing( String line)
            { addToDg (previousToken, line); }
//-------------------------------------------------------------
      // return the integer corresponding to the input string
      private int getIntForString ( String a, String line )
      {
            int j = -1;
            try
            {
                  j = java.lang.Integer.parseInt ( a );
            }
            catch (NumberFormatException nfe)
            {
                  System.out.println(“dgr: error: “);
                  System.out.println(“\tbad number: \”” + a + “\””);
                  System.exit(1);
            }
```

```
            if ( j < 1 )
            {
                  System.out.println("dgr: error: ");
                  System.out.println("\tnumber out of range: \"" + a + "\"");
                  System.exit(1);
            }
            return ( j );
      }
//-------------------------------------------------------------
      // each dg element consists of a TreeMap;
      // the key object is an Integer, whose value
      // is the number before the colon;
      // the value object is a Vector, whose elements
      // are Integers, whose values are the values on
      // the rest of the line;
      //
      // example:
      // so if this is the input line:
      // 3 : 2 5
      // then the key object is the Integer 3,
      // and the Vector object consists of two
      // Integer objects, one for 2 and one for 5;
      Vector v;
      Integer keyobj;
      private void addToDg( String a, String line )
      {
            int b = getIntForString ( a, line );
            Integer c = new Integer ( b );
            if ( v == null )
            {
                  System.out.println("dgr: internal error: ");
                  System.out.println("\tVector v is null");
                  System.exit(1);
            }
            v.add ( c );
      }
//-------------------------------------------------------------
      private void nextDg( String a, String line )
      {
            int b = getIntForString ( a, line );
            keyobj = new Integer ( b );
            if ( dgTreeMap.containsKey (keyobj))
            {
                  System.out.println("dgr: error: ");
                  System.out.println("\tduplicate entries for line number "
                        + b + ":");
                  System.exit(1);
            }
            v = new Vector();
            dgTreeMap.put ( keyobj, v);
      }
```

```
//-------------------------------------------------------------
}
```

A.8 availNumber.java

```
// this class creates an available list for integers;
// the list is initially empty and the class returns
// numbers starting with 0, 1, ...;
// if the list is not empty, the class will return
// numbers from the list, in ascending sequence,
// until the list is empty, whereupon it will continue
// to return numbers continuing from the last number
// outside the list;
package slice_engine;
import java.util.*;
//-----------------------------------------------------------------------
class availNumber
{
private final String name;
private int nextAvailNumber = 0;
private final TreeSet availNumberSet = new TreeSet();
//-----------------------------------------------------------------------
// constructor
availNumber( String xname ) { name = xname; }
//-----------------------------------------------------------------------
public int getNumber()
{
      int return_value = -1;
      if ( availNumberSet.size() <= 0 )
            return_value = nextAvailNumber++;
      else
      {
            Integer a = (Integer) availNumberSet.first();
            availNumberSet.remove(a);
            int i = a.intValue();
            return_value = i;
      }
      return ( return_value );
}
//-----------------------------------------------------------------------
public void putNumber( int i )
{
      if ( i < 0 )
      {
            System.out.println
            (“available number list for \”” + name + “\”:”);
            System.out.println
            (“\terror: trying to put negative: “ + i);
            System.exit(1);
      }
      Integer a = new Integer(i);
```

```
      if ( availNumberSet.contains(a) )
      {
            System.out.println
            (“available number list for ‘” + name + “‘:”);
            System.out.println
            (“\terror: trying to put duplicate: “ + i);
            System.exit(1);
      }
      availNumberSet.add(a);
}
//-----------------------------------------------------------------------
}
```

Distribution

1	MS	0188	LDRD Office, 1030
2		0785	J. Espinoza, 6514
1		0785	R. E. Trellue, 6514
2		0785	P. L. Campbell, 6516
1		0785	R. L. Hutchinson, 6516
1		0839	R. L. Craft, 16000
1		0899	Central Technical Files, 8945-1
2		0899	Technical Library, 9616
2		0612	Review & Approval Desk, 9612
			For DOE/OSTI