Smalltalk Class Hierarchy Graph Browser

Neil W. Van Dyke

December 8, 1992 (Minor corrections on December 18, 1992)

CSE509 Object-Oriented Programming Final Project

Oregon Graduate Institute of Science & Technology

1 Introduction

The goal of this final project was to design and implement a graphical class inheritance hierarchy browser, which could be integrated with the Objectworks\Smalltalk programming environment.

The product of this effort is called "CHG,"[1] (as in "Class Hierarchy Graph.") CHG provides a graphical view of the Smalltalk class hierarchy, and allows Class Browsers to be spawned from the represented classes.

There is also the concept of a *focus class*, which allows the view to be restricted to the subclass tree and superclass chain of a given class. The focus class can be changed interactively, and there are a few other features which allow flexible and efficient browsing.

2 Classes

CHG is implemented using five classes, described below.

2.1 CHGNode

CHGNode is a node in the graph, representing a class in the inheritance hierarchy. It has the instance variables:

children — a CHGNodeCollection of child nodes, representing subclasses.

className — the class name as a String.

classRef — the Class being represented.

extent — a Point for the extent of the rectangular shape that the CHGNode is displayed with.

labelText — the Text to use for displaying the class name, stored so that it does not need to be constantly recalculated.

origin — a Point for the origin of the display shape for the node.

It has the public instance method protocol:

addChild: *aCHGNode*
bounds
childConnectionPoint
children
children: *aCHGNodeCollection*
className
classRef
classRef: *aClass*
containsPoint: *aPoint*
displayConnectionOn: *aGraphicsContext*
  to: *aCHGNode*
displayOn: *aGraphicsContext*
displayOn: *aGraphicsContext*
  focused: *aBoolean*
extent
isLeaf
origin
origin: *aPoint*
parentConnectionPoint

an initialize private instance method, and a public class protocol new method. (Method descriptions are provided as comments in the source code.)

1. Affectionately pronounced "chug."

2.2 CHGNodeCollection

CHGNodeCollection is a variable subclass of SortedCollection. The public class method, new, invokes the private class method, initialize, which sets the sortBlock to sort in ascending alphabetical order of class name. The bounds instance method returns a Rectangle bounding box for the bounds of all CHGNodes in the collection.

2.3 CHGModel

CHGModel is the data model for CHG, and has the instance variables:

focusNode — the CHGNode being focused on.

layoutHorizontalSpacing — an Integer for the spacing between the origins of horizontally-adjacent CHGNodes, computed and used in different phases of the graph layout algorithm.

layoutVerticalSpacing — an Integer for the vertical spacing between nodes, and remains constant in the current implementation.

rootNode — the CHGNode at the actual root of the inheritance tree. In Smalltalk, this will always represent the Object class.

showSuperclasses — a Boolean to indicate whether or not superclasses should be included in the graph. If not, the visible root of the graph is the focus node.

CHGNode's public instance protocol is:

focusNode
nodeContainingPoint: *aPoint*
rootNode
showSuperclasses
showSuperclasses: *aBoolean*
updateFromSmalltalk

The private instance protocol for CHGNode is:

adjustSubtreeLayout: *aCHGNode*
build
buildChildNodes: *aCHGNode*
buildParentNodes: *aCHGNode*
centerNodeRelativeToChildren: *aCHGNode*
focusOn: *aClass*
initialize
largestNodeWithinSubtree: *aCHGNode*
layout
layoutSubtreeSimply: *aCHGNode* at: *aPoint*
nodeContainingPoint: *aPoint*
  inSubtree: *aCHGNode*
rootNode: *aCHGNode*
visibleRootNode: *aCHGNode*

Again, the new public class method is overridden to invoke the initialize instance method.

2.4 CHGView

CHGView, a subclass of View, is the view for CHGModel, with the public instance protocol:

defaultControllerClass
displayOn: *aGraphicsContext*

and the private instance protocol:

displayFocusNodeOn: *aGraphicsContext*
displayStatusOn: *aGraphicsContext*
displaySubtree: *aCHGNode*
  on: *aGraphicsContext*
pickAClass: *aString*

The public class method protocol is:

openFocusedOn: *aClass*
openPickFocusClass

and the private class protocol is:

open
open: *aCHGView*
openOnModel: *aCHGModel*

2.5 CHGController

CHGController, a subclass of ControllerWithMenu, is the controller for the CHGView. Its instance variables are:

lastYellowButtonPoint — a Point for the location of the mouse cursor when the yellow mouse button was last pressed.

selectedNode — the CHGNode under the cursor when the yellow mouse button was last pressed; nil if none.

CHGController has a menu public instance method, and the following private instance protocol:

changeFocusNode
inspectModel
inspectNode
redButtonActivity
showAboutInfo
spawnCHGOnPicked
spawnCHGOnSelected
spawnClassBrowser
toggleSuperclasses
updateModel
yellowButtonActitivity

3 Graph Layout

CHG uses a three-step graph layout algorithm that is easy to implement, yet fairly efficient, and yielding of adequate graphs. The three steps are:

1. Determine the horizontal spacing to be used, based on the width of the largest node in the tree—found by invoking CHGModel's largestNodeWithinSubtree: recursive method for the root of the tree.
2. Traverse the tree breadth-first, positioning children to the right of their parent, and vertically-aligned by their siblings. CHGModel's layoutSubtreeSimply:at: method performs this step.
3. Traverse the tree depth-first, centering each parent node vertically, relative to its children. This is done by CHGModel's centerNodeRelativeToChildren: method.

CHGModel's layout method invokes each of the three steps mentioned above.

4 Installation

CHG is installed by using the file-in feature of Smalltalk to load the ClassHierarchyGraphBrowser.st file-out into the image.

In order to be able to invoke CHG from the Launcher and other browsers, the class definitions of those must be modified to send the message

CHGView openFocusedOn: *aClass*

or

CHGView openPickFocusClass.

For example, LauncherView's browseMenu class method can be modified by adding an item to the keys variable, incrementing the specified constant for the size of the menu values array, and adding a line such as

```
subMenu valueAt: 6 put:
  [CHGView openPickFocusClass].
```
[1]

5 Using CHG

When installed as suggested above, a CHG browser can be invoked by selecting the appropriate menu option and entering the name of the desired class. For example, if Collection is entered, a CHG browser window which looks similar to that in Fig. 1 will open.

In the browser, each class is represented by a rectangular graph node, labeled with the name of the class. The focus node is drawn with a thicker rectangle. Connection lines from the right ends of nodes to the left ends of other nodes indicate the relationship of a superclass to its subclasses. A status line in the upper-left corner of the window indicates the name of the focus class, and whether or not the superclass chain of the focus class is shown.

Two menus are supported—a Class menu and a Hierarchy menu. The Class menu, available when the cursor is over a node (considered the *selected* node/class), has the options:

Spawn Class — Spawn a Class Browser on the selected class.

Spawn CHG — Spawn a new CHG, focused on the selected class.

1. Valid for ParcPlace Objectworks\Smalltalk, version 4.1. Source code for modified method is not reproduced here due to copyright restrictions.

Focus On Class — Change the focus to the selected class.

Inspect Node — Open an inspector on the selected node. (Used for debugging purposes in this prototype.)

The Hierarchy menu, available when the cursor is not over a class node, has the options:

Update CHG — Update from the current class hierarchy in the Smalltalk system.

Spawn CHG... — Spawn a new CHG browser, prompting for a focus class.

Toggle Superclasses — Toggle whether or not the superclass chain of the focus node is shown.

About — Show information about this program.

Inspect Model — Open an Inspector on the browser model. (Used for debugging.)

Clicking the Red mouse button performs the same operation as selecting Focus On Class from the Class menu, but is quicker and easier to operate.

6 Enhancements

Some potential enhancements which were not possible to implement within the time frame allotted for this project are listed below.

6.1 Window Scrolling

One useful feature would be the capability to scroll the window. Presently, the user cannot navigate the entire class hierarchy by clicking on nodes, because a window large enough to accommodate the full graph would not fit on a normal screen.

6.2 Graph Layout Improvements

The nodes in the superclass chain of the focus node should probably be separated by a fixed-width gap, instead of equally spacing the origins of the nodes. I felt it desirable to have the *subclass* nodes spaced horizontally by origin; but spacing of all nodes by gap could be accomplished with the elimination of the first step in the layout algorithm.

7 Comments

While becoming familiar with programming in Smalltalk, some comments were formed on the subjects of type checking, control of visibility of class methods, and productivity.

7.1 Type Checking

A well-known characteristic of Smalltalk is its lack of type-checking available in languages such as C++. An instance variable, for example, will reference an object of any type.

While this allows the programmer increased flexibility, it also permits unintended type errors to sneak in—of which the first indication is a run-time exception for a message not understood by the erroneous object.

An extension to the language might prevent this by allowing the programmer to express constraints on the types of object a variable or message parameter may take. Some programmers already include this information in comments or separate documentation, but it is not known to the Smalltalk system.

7.2 Visibility of Class Methods

Smalltalk lacks support for controlling visibility of class methods, (e.g., *private* to the class, or *public*), such as is provided in other object-oriented languages such as C++. I found this to be more freedom than desired, and for documentation purposes, grouped the methods into protocols named private and public.

It is a Smalltalk convention to group methods into protocols and name private-only protocols private or private-*something*. This approach seemed cumbersome when there are both public and private methods for each protocol (e.g., for menus in CHGController), but the class has few methods in total.

7.3 Smalltalk Productivity

The combination of garbage collection, the programming environment, (i.e., browsers, inspector, debugger, interpretive nature, etc.), class library, and language features, (e.g., blocks), made the implementation of project proceed much quicker than it would seem a comparable project implemented in C++ would be.

8 Bibliography

Krasner, Glenn A., Stephen T. Pope, *A Description of the Model-View-Controller Interface Paradigm in the Smalltalk-80 System*, ParcPlace Systems, 1988.

LaLonde, Wilf R., John R. Pugh, *Inside Smalltalk*, Prentice-Hall, 1990.

Lippman, Stanley B., *C++ Primer*, Second Edition, Addison-Wesley, 1991.

9 Availability

This package is available via Internet anonymous-ftp from cse.ogi.edu in the file /pub/smalltalk/chg_0_3.tar.Z. It may also be submitted to the Manchester Smalltalk Archive.

This project was done in a short amount of time as an educational programming exercise and is not a polished product. It is distributed for the sake of those who will get some benefit from using it as it is, or wish to enhance it in interesting ways.

The author may be contacted via E-mail to neil@cse.ogi.edu or telephone at +1 503 690 1300.

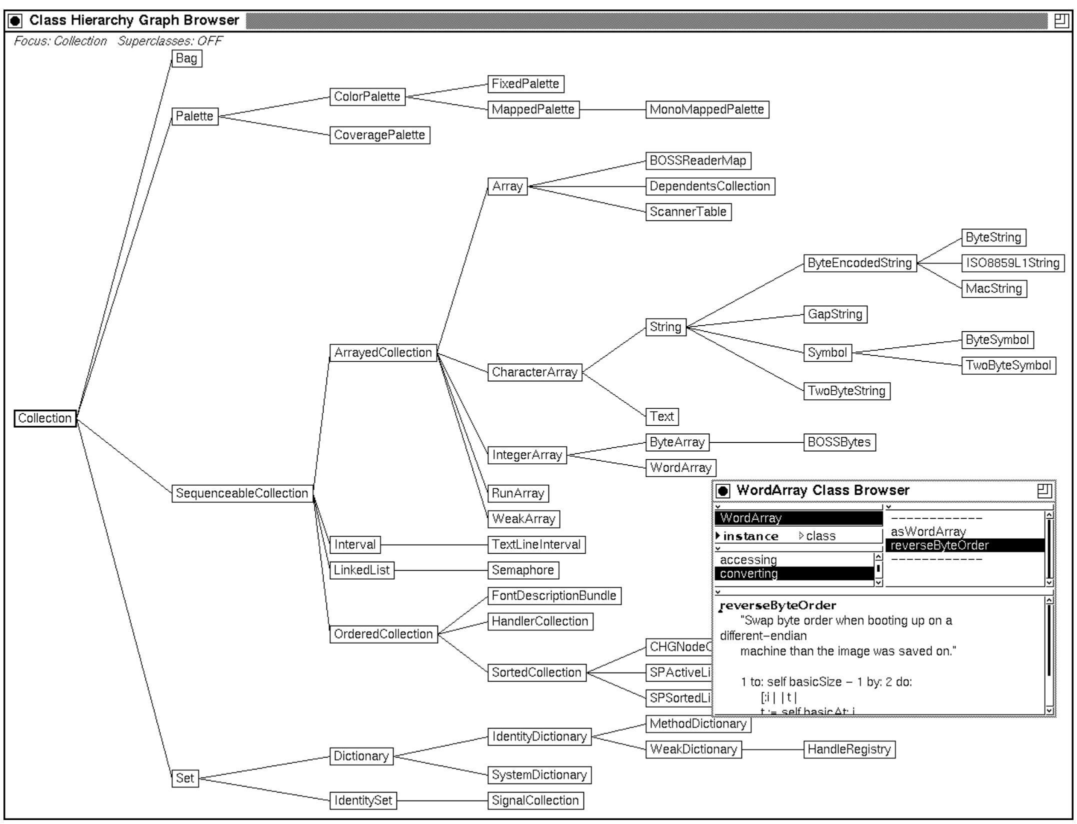

Fig. 1. Sample view focused on the Collection class, with spawned Class Browser.