OREGON STATE UNIVERSITY COMPUTER SCIENCE DEPARTMENT

OSU v 3.0 Browser: Window into GUI Applications

Tong Li
Dr. T. G. Lewis
Dr. T. Budd
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

92-60-1

OSU v 3.0 Browser :
Window into GUI Applications

By Tong Li

A research paper submitted in partial fulfillment of the
requirements for the degree of Master of Science

Major Professor : Dr. T. G. Lewis
Minor Professor : Dr. T. Budd

Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

Dec. 13, 1991

Abstract

Graphical user interface(GUI) applications based on object-oriented design are difficult to build without a supportive tool to graphically visualize the structure of the entire application. As an application becomes larger and more complex, it becomes harder to visualize its class hierarchy. Several systems, such as Smalltalk, MacApp, THINK C 5.0 and THINK Pascal, have provided powerful tools for this visualization. However, none of them can be invoked internally by the Oregon Speedcode Universe version 3.0 (OSU v3.0) supporting tools such as the Petri Net Editor [Keh 91]. The Petri Net Editor needs to view the class hierarchy of an application and obtain information to specify a transition arc, such as the method that sends a message to an object, the class that defines this method, and the path to find the definition of this class.

The solution is straight forward: to build our own OSU 3.0 Browser to meet the OSU v3.0 supporting tools' needs. A browser provides a graphical view of the class hierarchy of an entire application and gives a better idea of how the system or the application is structured and how the classes relate to each other.

The functionalities of the OSU 3.0 Browser is to parse the C++ source code of the OSU Application Framework, save the necessary information in an internal data structure, display the class hierarchy in a tree chart, and return the path name of the definition of the selected methods to the Petri Net Editor. The Browser is built on the OSU Application Framework [Wittel 91] and integrated with the Petri Net Editor.

Table of Contents

1. Introduction

One of the most complex and time-consuming programming areas is the development of graphical direct-manipulation user interface applications [Myers 89] [Myers 90] [Urlocker 89]. In this paper, we explore some of the reasons why GUI applications are difficult to program without a browser, and discuss various browsers, in particular, our OSU 3.0 Browser, that have been incorporated into our approach to these problems.

Oregon Speedcode Universe version 3.0 or OSU v3.0 consists of an MVC-based (Model-View-Control-based) object-oriented application framework called OSU Application Framework and a set of integrated high level tools for specification, visualization, modeling, simulation, validation, and rapid prototyping of GUI applications. The OSU v3.0 architecture is pictured in Figure 1: the RezDez allows the designer to create and edit icons, menus, windows, panes, palettes, pictures, cursors, dialog boxes, and alerts; the Petri Net Editor provides a graphical front end to most of the underlying application framework features and produces an executable specification (model) which is the design representation of the modeled system and can be easily translated into a C++ program; the Graphical Application Builder allows the designer to do "reverse specification", interactively manipulating user interface objects, and to generate the formal specification from there; the Simulator sets the initial state of the modeled system according to the initial marking of the net, and then executes the system by using the user's inputs; the Analysis Tools based on previously developed reachability graph analysis techniques can be implemented for analyzing the Petri net design representation of a system to determine system properties; and the Code Generator takes an annotated Petri net as input and produces an OSU

Application Framework-based C++ program as output. Interested readers of this framework and its tools are referred to the related reports [Keh 91], [Wittel 91], [Lai 91], [Luo 91].

The Browser that I implemented allows the designer to navigate through the application framework class hierarchy, retrieve desired features if necessary, and visualize the connection between the sequence and the class hierarchy. It can parse C++ code (either the application specific C++ code, or the C++ source program generated by the Code Generator, or the source code in our Framework) and display the class hierarchy in a tree chart. Inheriting the features from the Framework, it can save the tree chart in documents and can read the saved Browser documents and display them in a window. Currently the Browser is directly used by the Petri Net Editor for obtaining the information about a chosen method of a chosen class to specify transition arcs.

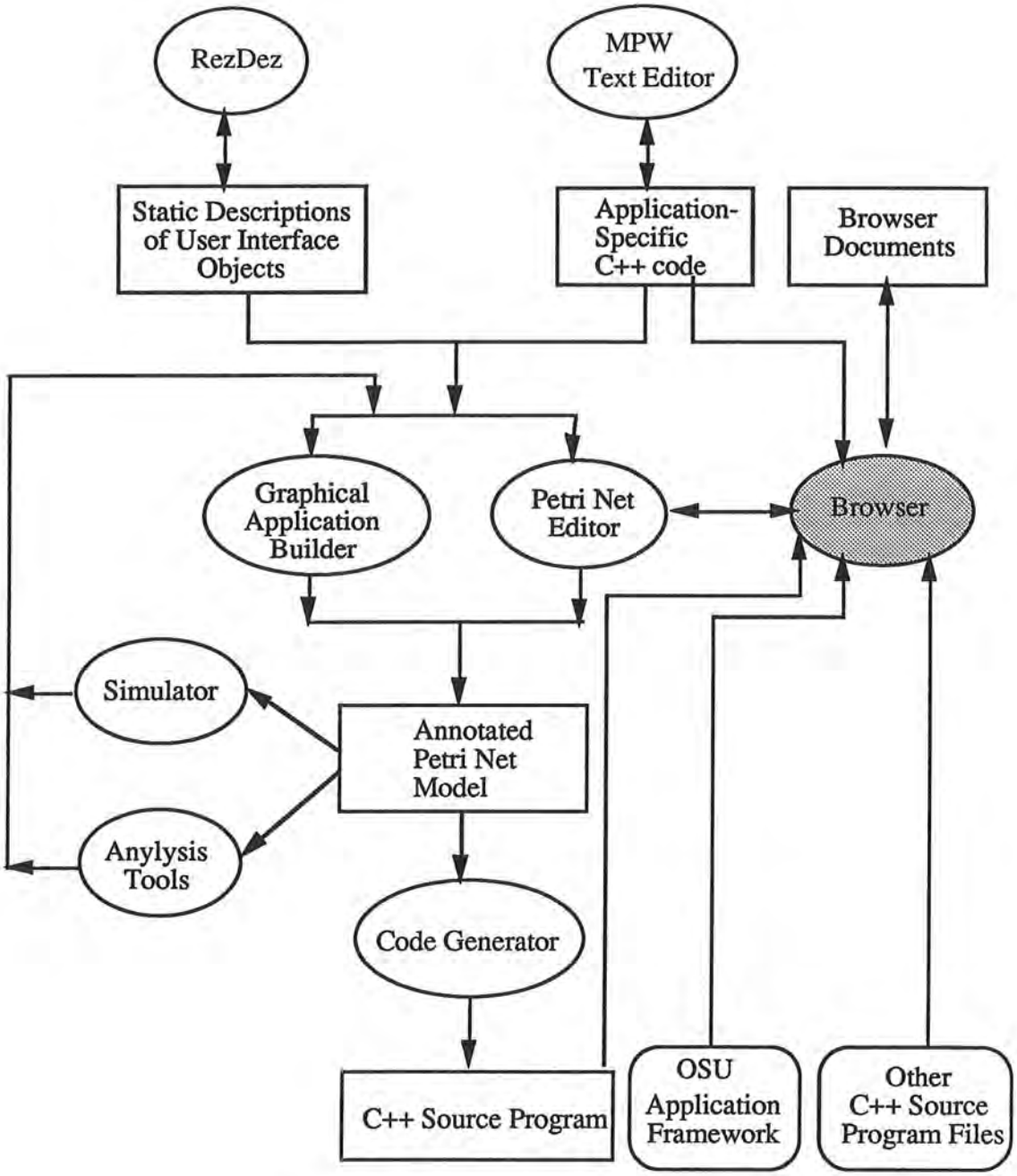

Figure 1. Architecture of OSU v3.0

2. The Problem

The introduction of the Lisa in 1983, followed by the Macintosh in 1984 exposed a wide audience to Apple's implementation of the GUI developed in the 1970's at the Xerox Palo Alto Research Center (PARC) [Allen 1990]. Since then, the desktop metaphor with bit mapped graphical windows, icons, and a mouse for input has become an accepted standard for user-friendly applications. However, Graphical User Interfaces (GUI's) are still difficult to program [Weinand 89] because of asynchronous input management, lack of high level abstraction in GUI toolkits, and lack of a standardized model for generic GUI functionality. Although GUI toolkits provide good abstractions for the lowest levels of a GUI, such as line and shape drawing, mouse and keyboard input, and display of standard graphical items such as windows, buttons, and menus, the programmer must constantly reinvent the wheel when integrating these features into an application.

The problem of too little functionality is addressed in our framework by using an object-oriented approach that encourages both the reuse of design and the reuse of code. But several cognitive problems prevent users from successfully exploiting their function-rich systems [Fischer 87]. Users do not know

- that reusable components exit,
- how to access these reusable components,
- when to use these reusable components,
- what the reusable components do, or
- how to combine, adapt, and modify these reusable components to their specific needs.

To build new applications as much as possible with existing parts, the designer must understand how these existing parts organize.

3. The Approach

A browser is a window into GUI application. It supports the object-oriented approach by visualizing the hierarchical architecture of an application and providing easier access to the system. The OSU 3.0 Browser displays the inheritance structure of our Framework and lets the programmer look around a system and search for reusable components. The connections between the Framework and the classes it inherits can be analyzed in more detail by selecting a class and looking at its methods. Specifically, the user of the Petri Net Editor would create a GUI object such as menu, window, dialog, alert, or so forth, and specify a transition arc to this object. The OSU 3.0 Browser provides a graphical view of the reusable framework and lets the user to choose the desired methods to specify the transition arc.

Before we discuss our OSU 3.0 Browser in detail, let's elaborate on the other browsers from four different systems, THINK Pascal [Borenstein 88], THINK C [Borenstein 89], MacApp [Wilson 90], and Smalltalk-80 [Goldberg 83], for comparison.

3.1. The THINK Pascal Browser

From the perspective of a software production staff, profitability depends largely on component availability and on easily locating components, accessing their applicability, and incorporating them into the software system being developed [Woodfield 87]. The convenience of getting a program serviced easily is very appealing. The less effort the users have to expend in finding candidates for reuse, the more likely they are to use them. The reusable components are useless unless the designer knows that they are available and how the right one

can be found. A browser addresses this retrievability problem by letting the user locate resources without knowing and remembering names. And the THINK Pascal Browser is a good example of solving this problem.

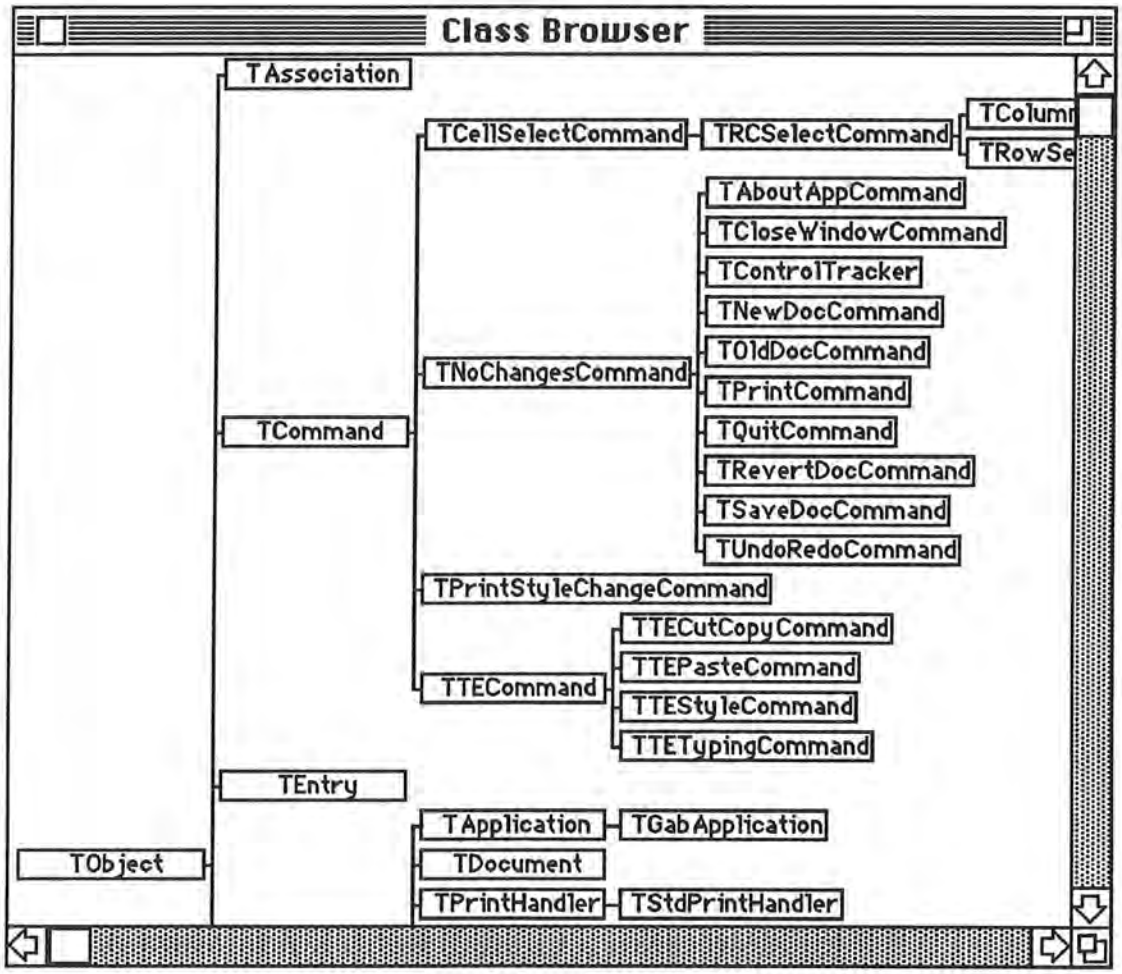

Figure 2. The THINK Pascal Browser

The interface of this Browser is shown in Figure 2. It displays all the classes defined in a project as a tree chart. Each box represents one class. Subclasses are connected to their superclasses like an organizational chart. To find the declaration of a class, double-click on the box that represents the class. The editor opens the file that contains the class declaration and scrolls to the declaration of the class. To find all the methods that a class defines or overrides, hold the mouse down on the class

name. A pop-up menu appears next to the class. If the user chooses a method from the pop-up menu, the editor opens the file that defines the method. Here, the pop-up menu shows only the methods that a class defines or overrides. It does not include the methods that the class inherits from its superclass, since several classes may define or override the same method, it's not clear which file the editor should open.

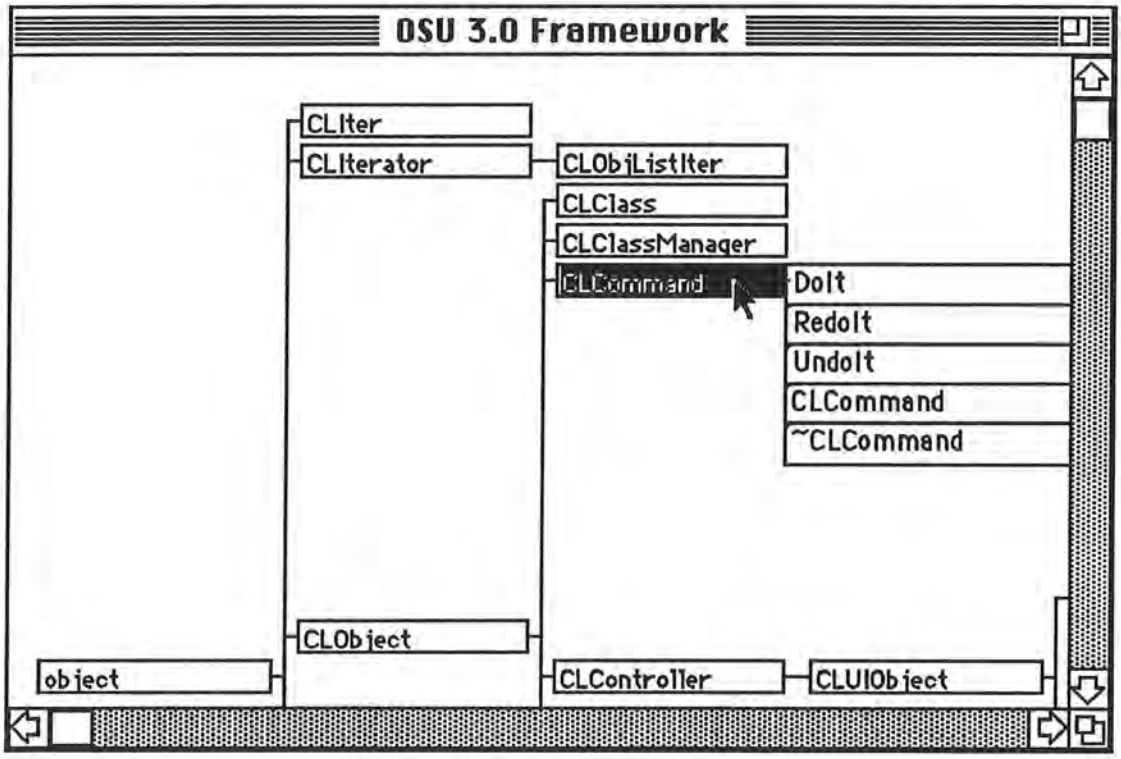

Figure 3. The OSU Browser

This kind of interface shows clearly the entire class hierarchy of the application and the inheritance relationship between classes (superclass and subclass). As an exercise in graphical user interface design and implementation, I chose this style of interface for my Browser, as shown in Figure 3. The differences between these two interfaces are: since the OSU Application Framework can not support pop-up menus, I decided to have two windows, one for displaying the tree chart of the class hierarchy, another one for displaying the method list of the selected class; also,

in the THINK Pascal Browser, the sizes of the boxes which contain the class names are varied with the length of the class names; but the sizes of the boxes in the OSU Browser are fixed. The advantages of having these fixed size boxes and class names to represent a class in the class hierarchy view is that, the tree structure seems to be more even and the depth of a node (class) is shown more clearly. Also, it simplified the layout algorithm. The disadvantage of this design is, since the long class names (more than 15 characters) are truncated, it may happen to have identical class names in the tree chart representing several different classes. This case doesn't happen in the OSU 3.0 Framework.

3.2. The THINK C 5.0 Browser

The browser in THINK C 5.0 is called Class Browser. The user can use this Browser to look at the declaration of a class or to find the definitions of methods. The interface design of this Browser is the same as the THINK Pascal Browser that was discussed above except for the keyboard shortcut functionality: the user can use the keyboard to navigate through the classes in the Class Browser window. For example, if the user types the name of a class, the Class Browser highlights the classes whose name matches what the user has typed so far.

3.3. The MacApp Mouser

Instead of showing the class hierarchy in a tree chart, the browser in MacApp (called Mouser) displays a list of class names in the top left view, and displays message and variable names for the selected class in the next two views, as shown in Figure 4. The large view on the bottom displays the interface information of any selected method. The users can use Mouser as a substitute for the MPW text editor and modify the method code directly. The differences between Mouser and the OSU

Browser are: the Mouser has a subview to display the member fields but the OSU Browser doesn't support this functionality since the Petri Net Editor doesn't need this information; and the OSU Browser doesn't have a substitute for the MPW text editor.

The MacApp Mouser also allows many types of searching and cross-referencing through both the user's code and the original MacApp source code. For example, the user can find all methods that send the message DoIt to any object, or flatten the hierarchy and see all the methods inherited from a superclass in one scrollable list. The Mouser makes it much easier to navigate through the large class library. It is written in MacApp and can handle source code in Object Pascal or C++.

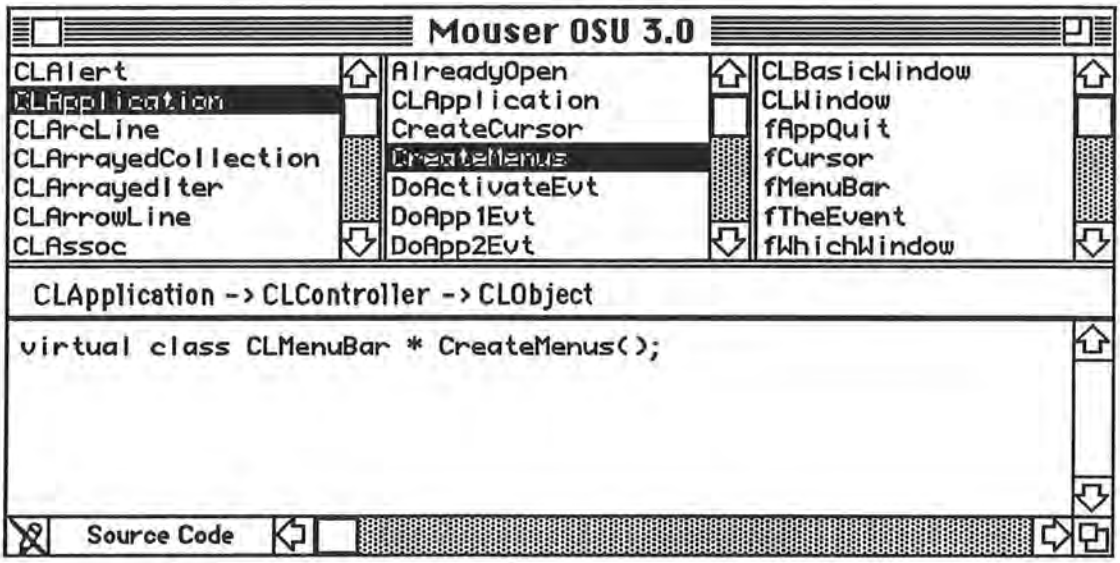

Figure 4. The MacApp Mouser code browser

3.4. The Smalltalk Browser

The Smalltalk Browsers were the first and are the most powerful browsers in the computer world. There are many browsers in Smalltalk: class browser, Class Hierarchy Browser, Project Browser, Protocol Browser, message-set browser, Message Browser, Message Category Browser, Change-Set Browser, Change-

Management Browser, System Browser, System Category Browser, and file list browser.

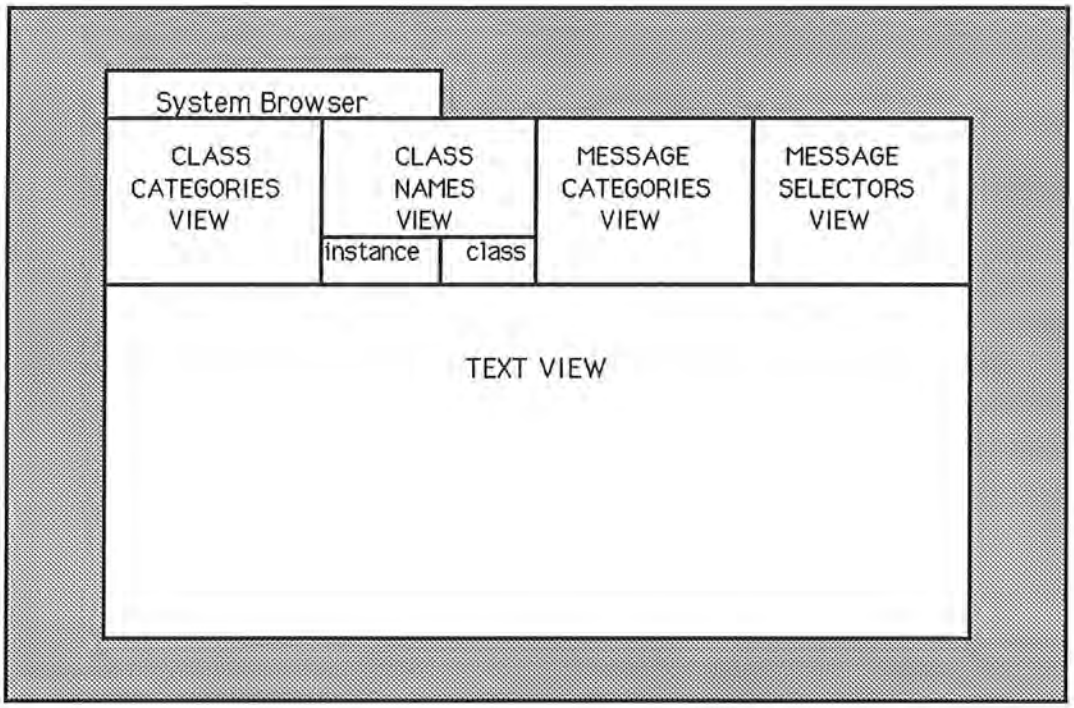

Figure 5. The Structure of a System Browser

An example of these browsers is the System Browser as shown in Figure 5. It is made up of five subviews and two menu items labeled "class" and "instance". The various subviews of the browser are referred to by their labels in Figure 5. The bottom one is a view in which methods can be defined and modified using the text editor. The information about a class that the user can retrieve using this browser includes:

- a comment about the role of the class in the system
- a description of the part of the system class hierarchy in which the class is found
- a description of the variables of a class

- a description of the message and methods of the class, including comments about the use of the message and the design of the method
- a classification of the class with respect to other classes
- a classification of the message of the class
- access to all methods in the system that send a particular message
- access to all methods in the system that implement a particular message
- a list of all message sent in a particular method

Among this list of functions, my Browser only can show the class hierarchy and the list of the methods of a class. If the OSU tools need more information and functionalities in the future, my Browser should be enhanced. The current design is ready for the enhancements, which will be explained in the implementation section of this report.

The Smalltalk system browser also provides access to templates for defining new classes and templates for defining new messages. This Browser gives the user access to all the class descriptions available in the system, including comments about the classes, comments about the methods, and examples of how to use many of the classes. Other ways to find out about messages and methods involve creating system views called Message-Set Browser. These views are created in response to queries to determine which methods send a particular message, which classes implement a particular message, or which methods reference a particular variable or literal. All these help the users to understand the system and determine whether they would reuse it and how to reuse it.

3.5. The OSU 3.0 Browser

The OSU 3.0 Browser was designed as a support tool for the OSU 3.0 Framework. Currently the Petri Net Editor is using it to browse the class hierarchy of the OSU Framework and choose methods to specify an arc [keh 91]. When the user chooses the "Browse Hierarchy" in the "Tools" menu (Figure 6.), the Browser will be invoked and will bring up a dialog (Figure 7.) asking the user to choose a header file. The user then selects a file in the "includes" folder of the OSU 3.0 Framework.

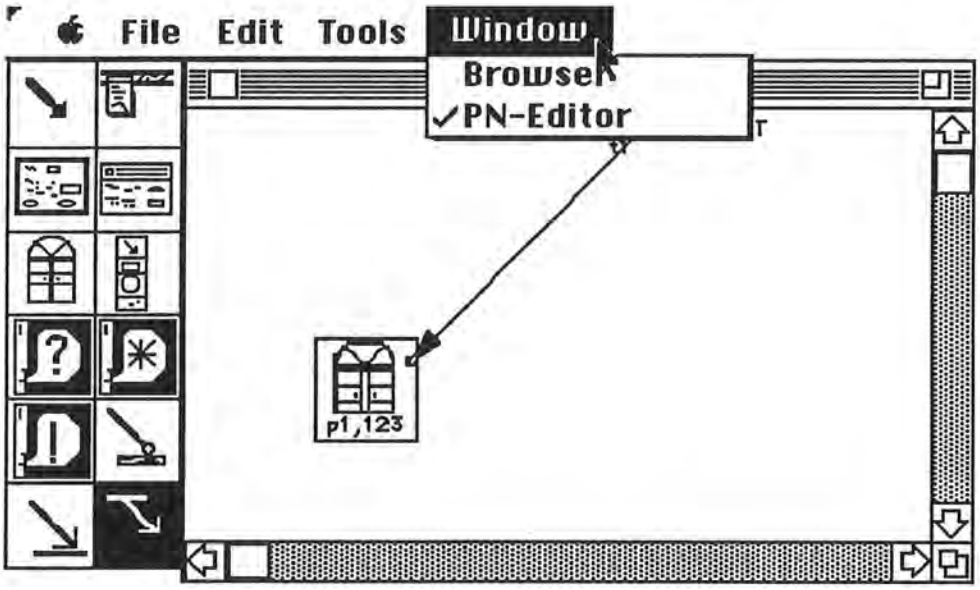

Figure 6. Petri Net Editor Environment

Then the Browser automatically parses all the header files (consisting of the class definitions) in the current folder. At the same time, it shows a modeless dialog (Figure 8.) telling the user which file it is parsing. When it parses all the files in the "includes" folder, it will check whether all the class definitions have been parsed, in other words, it will check if all the necessary header files are in the "includes" folder.

If there are any header files missing, the Browser will ask for the files where the specific classes are defined (Figure 9.).

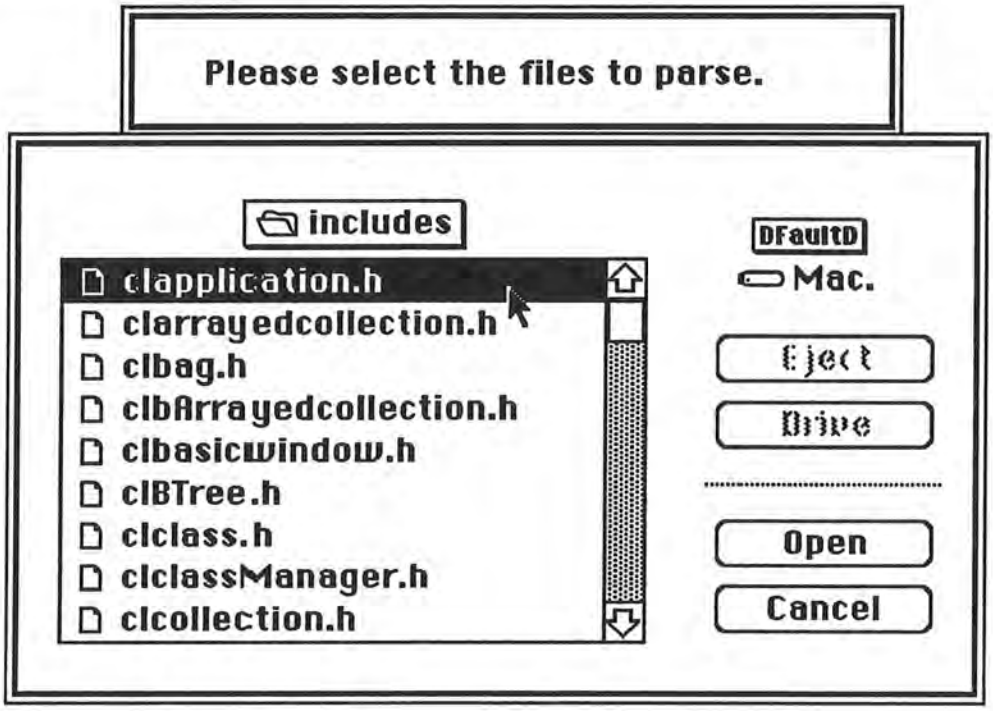

Figure 7. Select a File in the "includes" Folder

Figure 8. A Dialog Showing which File is Being Parsed

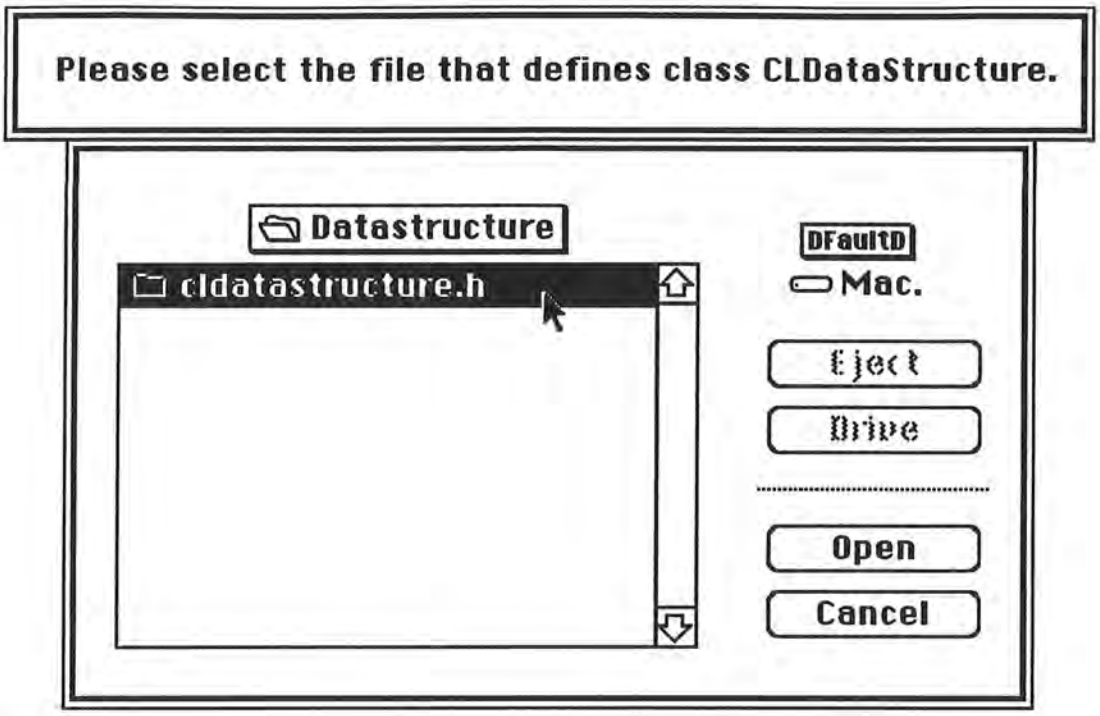

Figure 9. Find the File that Defines Class CLDataStructure

After all the parsing and checking, the Browser will display a window containing the tree chart that represents the class hierarchy (Figure 10.). The user can click on a box that represents a class to see its protected and public function members, and double click on a method name or select a few methods and choose the "copy" item in the "Edit" menu to select methods. After the user chooses all the methods he wants, the list of messages of the method definitions can be pasted to the high-lighted arc.

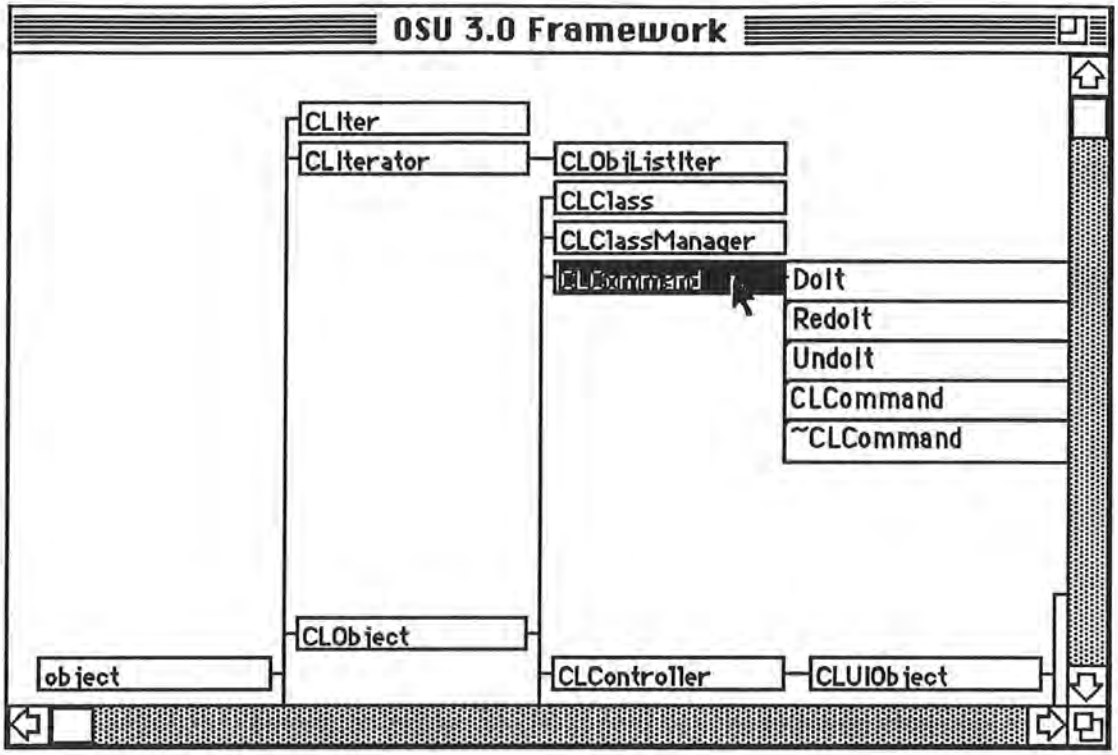

Figure 10. OSU Browser showing the OSU Framework Class Hierarchy

Whenever the user wants to choose messages to specify an arc, he/she can choose the "Browser" in the "Windows" menu and the Browser window will be brought up to the front. The user also can double click on an arc to see the messages specifying it. The method definitions will be displayed in a dialog as shown in Figure 11. And the user can edit these methods.

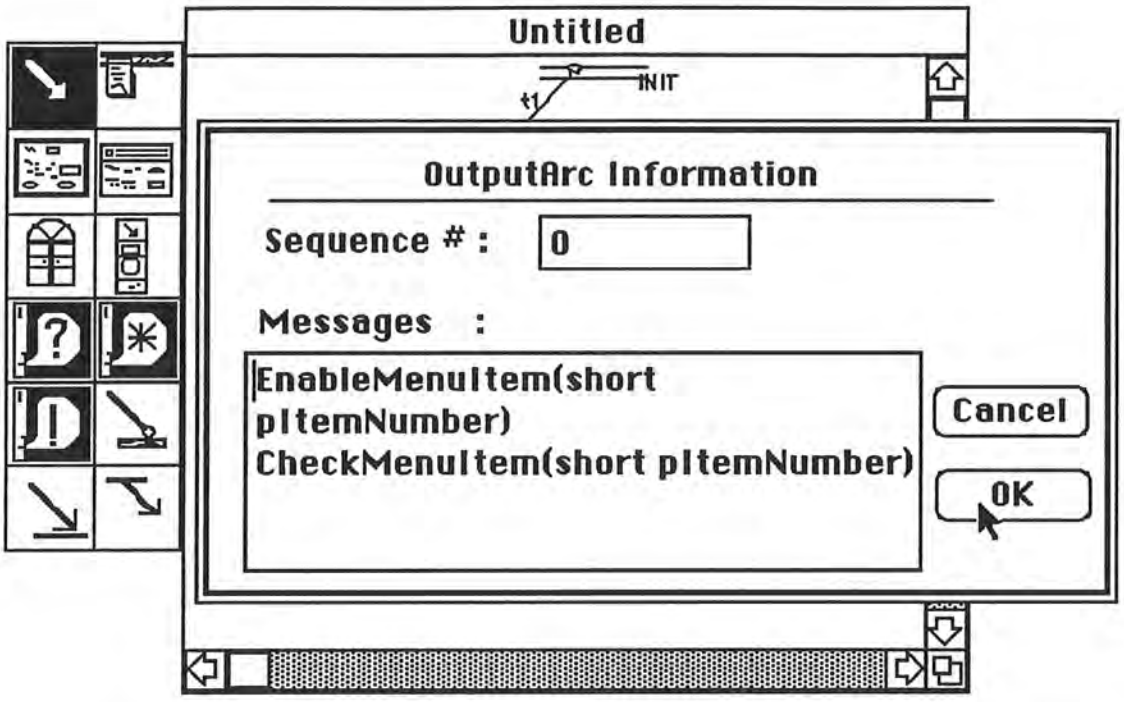

Figure 11. Returning the Selected Method Names to the Petri Net Editor

4. The Design of OSU Browser

In this section, I first discuss how my Browser inherited the object-oriented design of the OSU Application Framework. Then I will address the specific layout problem of displaying the tree chart. Basic knowledges of Macintosh applications, Object-Oriented design, and MVC model are referred to [Lewis 90], [Budd 90], [Krasner 88].

4.1. Subclassing the OSU 3.0 Application Framework

OSU 3.0 Browser was constructed by subclassing and instantiating classes from the OSU 3.0 Framework. The OSU 3.0 Framework is used by deriving new concrete classes from existing classes and configuring a set of objects by providing parameters to each object and connecting them (Figure 12). OSU 3.0 has a large class library of concrete subclasses of each abstract class, so that most of the time an application can be plugged together from existing components.

The model class holds the domain specific data that is to be represented and manipulated by the GUI application. The domain specific data of this Browser application are the BRClass and BRMethod classes which contain the information about an application program's classes or methods. In this application, the BRClassView renders parts of the data, the names of the classes and the names of the methods, on the screen. The BRClassView decides the layout of the tree chart which will be discussed in detail in the following section. In this particular application, a subclass of the Window class, BRWindow, is responsible for accepting asynchronous input from the mouse and keyboard and passing appropriate messages to the model

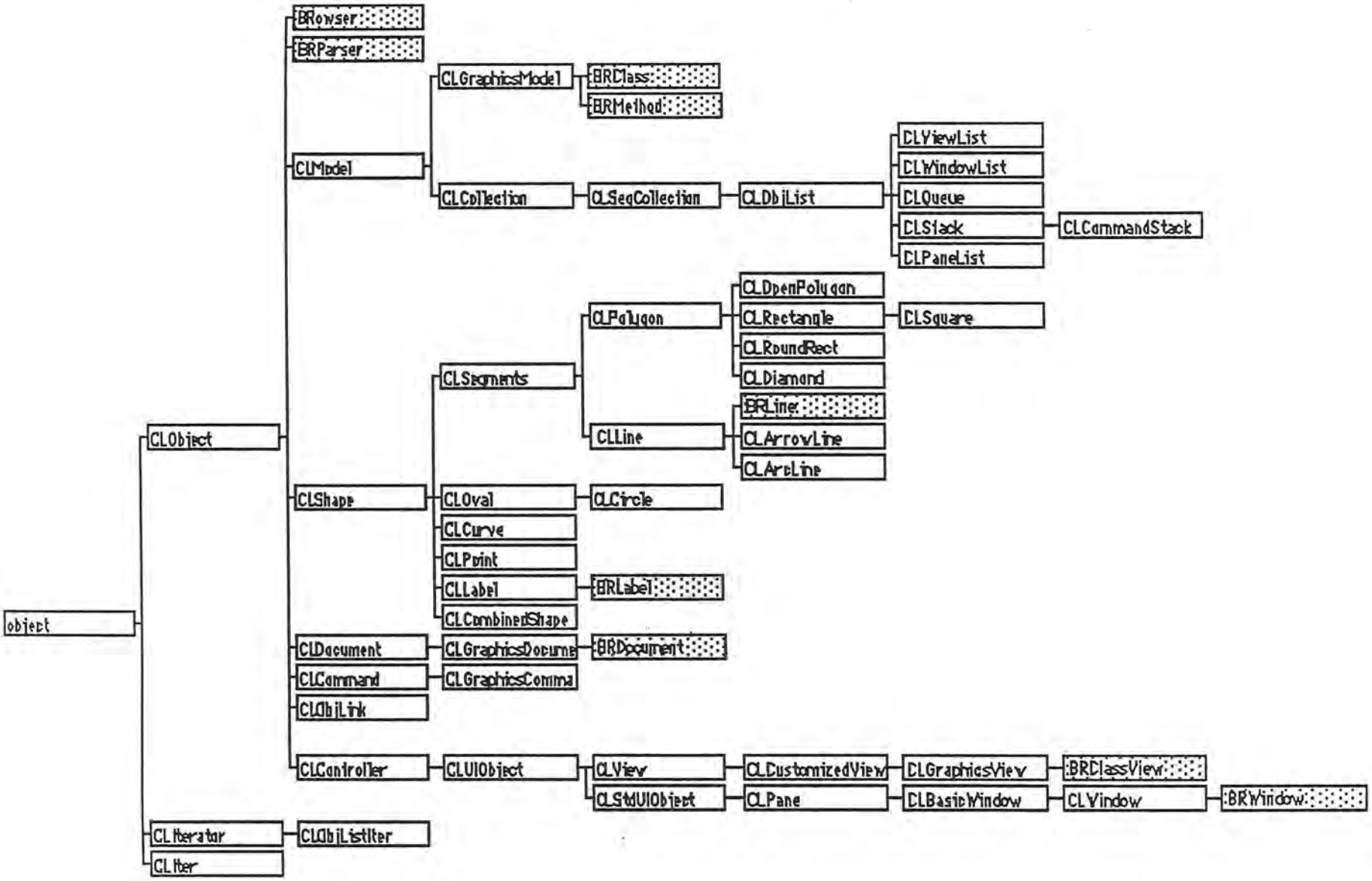

Figure 12. OSU 3.0 Browser Class Hierarchy
(The shaded classes are created for the Browser.)

and view classes. The BRDocument class which is subclass of the GraphicsDocument is created for file I/O.

BRLabel class and BRLine class are subclasses of the CLLabel class and CLLine class, which are subclasses of the CLShape class.

The BRParser class and the BRowser class are the most important classes created for this Browser: the BRParser class is responsible for parsing files, meanwhile building up a binary tree structure to store the class hierarchy; the BRowser class is responsible for invoking the Browser tool and returning the message list to the Petri Net Editor.

All these subclasses will be discussed in more detail in the next section.

Some classes in the Framework are so powerful that I could simply instantiate them. These classes are mostly the subclass of data structure classes [Luo 91], CLObjList, and the subclass of StdUIObject classes [Luo 91], CLModelessDialog. The way to use these classes is fairly easy. For example, to use a modeless dialog to ask for the files where the specific class is defined (Figure 9.), what I need to do is just the following four lines of code:

```
aDialog = new CLModelessDialog(128, classA -> getSuperclassName() );
aDialog -> Draw();
SFGetFile( where, (Str255) "", (FileFilterProcPtr) 0, -1, (SFTypeList) 0,
        (DlgHookProcPtr) 0, &theReply );
delete aDialog;
```

4.2. The Layout Algorithm

The key idea of my layout algorithm for drawing the class hierarchy tree chart is that I have two member fields in the BRLabel class: the coordinate numbers (the

vertical number "yNum" and the horizontal number "xNum"). After the BRParser reads all the files and builds the binary tree containing all the information of the class hierarchy, the BRClassView does a depth-first-search [Aho 74] to compute all the coordinate numbers as following:

```
void BRClassView :: setClassPositions(BRClass *theClass) {
    // at first pass the root of the binary tree
    BRLabel     *labelA, *labelB;
    BRClass     *classA, *classB, *classC;
    int         tempYNum, i;
    theClass -> setTag ( 0 );
    CLObjList *aSubclassList = theClass -> getSubclassList();
    if ( aSubclassList != 0 ){
        CLIter nextClass ( aSubclassList );
        classC = ( BRClass * ) nextClass();
        labelA = findClassLabel(theClass -> getClassName());
        labelB = findClassLabel(classC -> getClassName());
        labelB -> setxNum (labelA -> getxNum() + 1);
            // set the x number of the subclass' label equal to the x number
            // of the superclass' label plus one.
        while ( classC != 0 ) {
            if ( classC -> getTag() == 1)
                setClassPositions ( classC );
            classB = ( BRClass * ) nextClass();
            labelA = findClassLabel(classC -> getClassName());
            if ( classB != 0){
                labelB = findClassLabel(classB -> getClassName());
                labelB -> setxNum( labelA -> getxNum() );
                    // set all the subclasses with the same superclass
                    // the same x number.
            }
            classC = classB;
        }
    }
    if ( theClass -> getSubclassCount() == 0 ) {
        // if theClass doesn't have any subclasses.
        aYNum++;
        findClassLabel(theClass -> getClassName()) -> setyNum(aYNum);
    }
    else {
```

```
                classA = ( BRClass * ) aSubclassList -> First();
                labelA = findClassLabel(classA -> getClassName());
                labelB = findClassLabel(theClass -> getClassName());
                i = labelA -> getyNum();
                if (theClass -> getSubclassCount() == 1)
                        labelB -> setyNum( i );
                else {
                        tempYNum = (( aYNum - i ) / 2) + i;
                        labelB -> setyNum(tempYNum);
                }
        }
        setEachClassPosition( theClass );
};
```

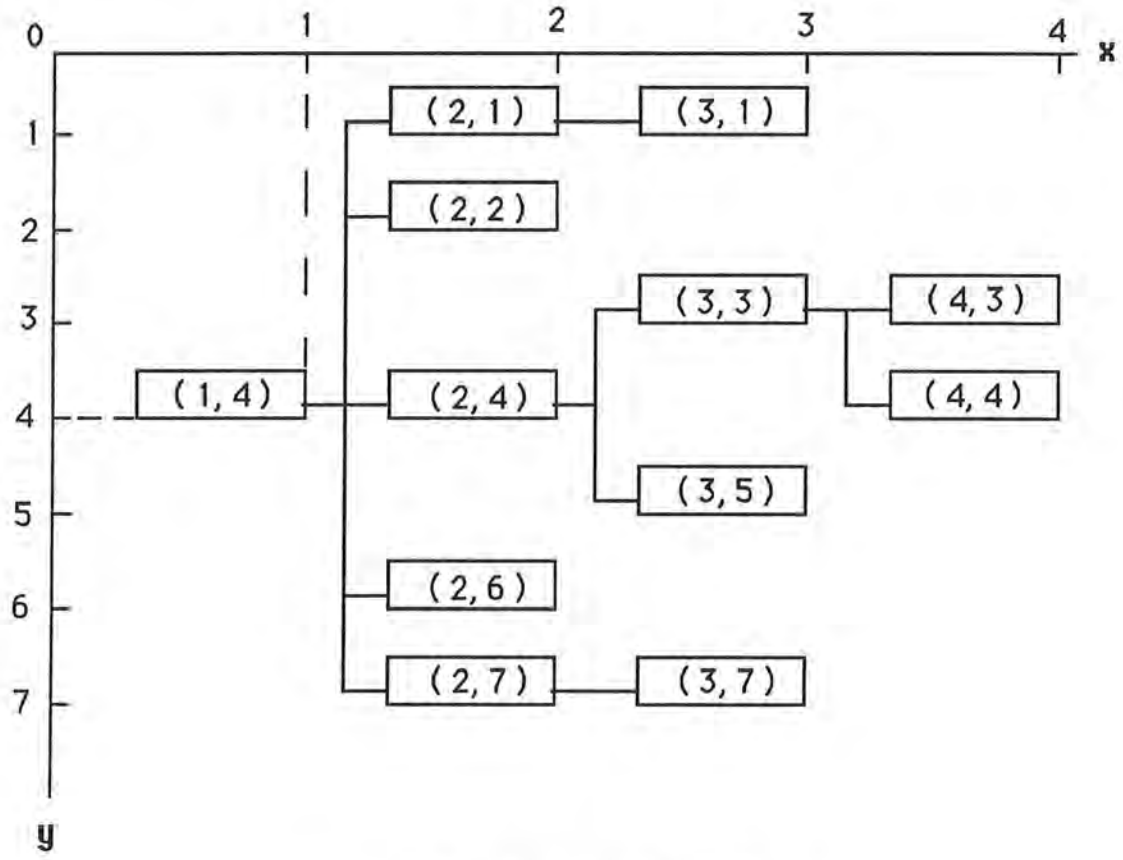

Figure 12. The Layout of a Tree Chart

As the result of this search, the layout of the tree will be as even as the ones in Figure 12. The numbers in the parentheses are the "xNum" and "yNum" of each node.

5. Implementation of the OSU 3.0 Browser

The descriptions below are introductory and describe the current state of the Browser. Instance variables and member functions of special interest to the implementation are also explained.

5.1. The BRClass Class

The BRClass class is a subclass of the CLGraphicsModel class. It contains the information we need about a class object, such as the name of the class, the name of its super class, the name of the file in which the class is defined, and the number of subclasses of it; and also some information for internal use, such as the pointer to its method list, and the pointer to the subclass list, also a tag for the depth-first-search representing whether a node has been searched to. If more information is needed in the future, more instance variables should be added in this class.

The methods in this BRClass class are to hold information about the instance variables and to get the information from the instance variables, also to read these information from the header files, as explained in detail below.

Instance Variables:

- className — the name of the class
- superclassName — the name of its super class
- fileName — the name of the file in which the class is defined
- subclassCount — the number of subclasses of this class
- methodList — the pointer to the method list
- subclassList — the pointer to the Subclass list
- tag — the tag for depth-first-search

Member Functions:

• setClassName	set the class name
• setSuperclassName	set the super class name
• setFileName	set the file name
• setMethodList	set the pointer to the method list
• setSubclassList	set the pointer to the subclass list
• setSubclassCount	set the number of subclasses
• setTag	set the tag value ('0' means that this class object node has not been searched and '1' otherwise.)
• addSubclassCount	add one to the number of subclasses
• getClassName	get the class object name
• getSuperclassName	get the super class name
• getFileName	get the file name
• getMethodList	get the pointer to the method list
• getSubclassList	get the pointer to the subclass list
• getSubclassCount	get the number of subclasses
• getTag	get the tag value
• readSelf	read the header files to set the information
• readMethods	read all the methods of the class

5.2. The BRMethod Class

The BRMethod class is a subclass of the CLGraphicsModel class. It just contains the name of the method and the line that defines the method and is responsible for setting and getting the name and the line.

Instance Variables:

• methodName	the name of the method

- methodDefStr — the line that defines the method

Member Functions:

- setMethodName — set the name of the method
- setMethodDefStr — set the line that defines the method
- getMethodName — get the name of the method
- getMethodDefStr — get the line that defines the method
- readSelf — read the line sent from the BRClass to set the information
- getMethodName — parse the line containing the method definition for the method name
- getMethodDef — parse the line containing the method definition for the method definition without return type

5.3. The BRLabel Class

The BRLabel class is a subclass of the CLLabel class in the Shape Library [Luo 91]. Because when the user double-clicks on a class in the tree chart, the Browser should display the list of methods of the selected class on the method window (see Figure 13.); when the user double-clicks on a method in the method window, the Browser should save the definition of the selected method in a list for the Petri Net Editor. So I need a member field as a pointer to the selected object in the BRLabel class. The two dimensional coordinate numbers (xNum, yNum) represent the related position of the class box in the graphical view. The CLHighlight method is to show a class is selected by inverting the region instead of putting four knobs at the corners of the region when the user clicks on it, the CLLabel class doesn't have this choose. And the CLDrag method was overrode for not doing anything. I use this BRLabel class for displaying labels in both the class window and the method window.

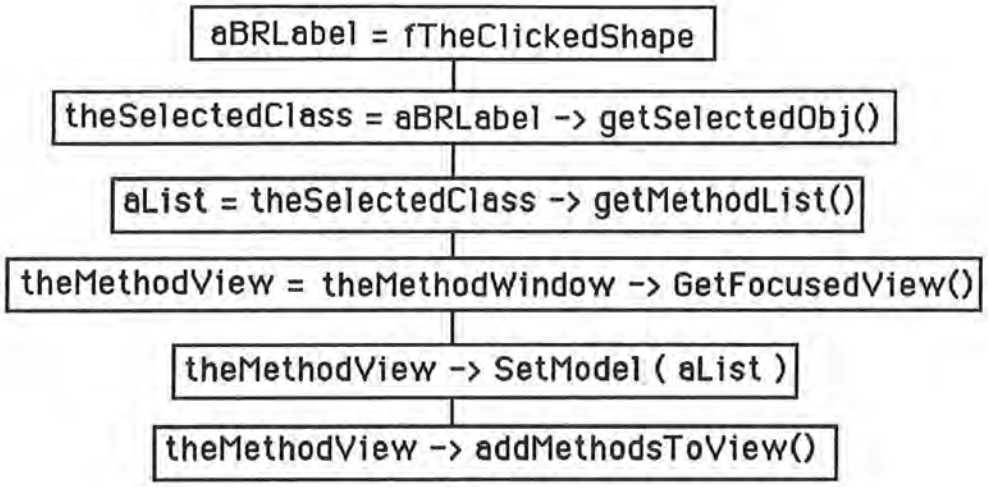

Figure 13. Double-click on a BRLabel in the BRClassView

Instance Variables:

• selectedObj	the point to the method list
• xNum	the coordinate number (horizontal) in the graphical view
• yNum	the coordinate number (vertical) in the graphical view

Member Functions:

• setSelectedObj	set point to the object list
• setxNum	set the horizontal coordinate number
• setyNum	set the vertical coordinate number
• getSelectedObj	get point to the object list
• getxNum	get the horizontal coordinate number
• getyNum	get the vertical coordinate number
• CLHighlight	highlight the label region by inverting it (override method)
• CLDrag	do nothing (override method)

5.4. The BRLine Class

The BRLine class was created for overriding the CLHilight method of the CLLine class. The lines in the tree chart should not be able to high-lighted, as a result they can not be dragged.

5.5. The BRClassView Class

The BRClassView class is a subclass of the CLGraphicsView class. It is responsible for drawing the graphical tree chart on the screen and detect the mouse actions.

Member Functions:

• xyToRect	convert the internal coordinate values of a class into a position (a rectangle) where the class name will be located on the screen
• addLabels	add one label to the view for each class object
• setEachClassPosition	decide the position for each label and add the lines to the view for each label
• setClassPositions	decide the positions for all the labels
• findClassLabel	knowing the class name, get the label of this class in the fShapeList of the view. Since I keep the xNum and yNum in the BRLabel class and decide the position of each label after I create all the labels, so I need to find the labels and set the position.
• DoubleClick	handle double-clicks for the view (this is an override method.)
• ReleaseMouse	handle release-mouse for the view (this is an override method.)
• addMethodsToView	add the methods of the selected class to the view
• addMethodsToStrList	add the selected methods to the list for the Petri Net Editor

5.6. The BRParser Class

The BRParser class is a subclass of the CLObject class. It is a major class of the Browser. The only member field is a pointer to a CLObjList object. Since each element in this list will be an object of the BRClass class which has a member field "subclassList" as also a CLObjList, this constructs a binary tree structure.

The BRParser class is responsible for parsing all the header files to build up the class hierarchy tree and setting the layout of the tree chart.

The parsing method that was used to parse the files is LL(1) parsing [Aho 86]. That means left-to-right, left-most, and a single input symbol used to resolve the production choice. This parsing process has a time bound that is linear with the length of the input string.

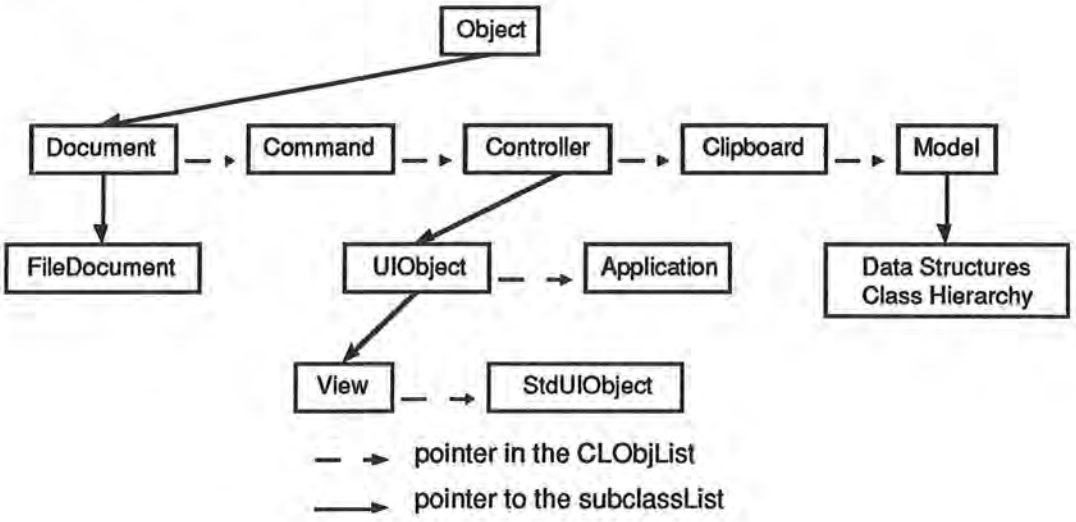

Figure 14. Binary tree structure to save Class Hierarchy

The instance variables and member functions of this class are:

Instance Variables:

- treeRoot — the root of the class hierarchy tree

Member Functions:

- getTreeRoot — get the root of the class hierarchy tree
- setTreeRoot — set the root of the class hierarchy tree
- initTreeRoot — initialize the root of the tree
- deleteTreeRoot — delete the root of the tree
- parse — do the parsing
- parseOneFolder — parse all the header files to get the class information for building the class hierarchy tree
- parseOneFile — parse one header file to get all the class information for building the class hierarchy tree (LL(1) parsing)
- getMissedFiles — ask for the files that define the classes whose subclasses have been parsed
- addClassToTree — add the new class to the tree
- setSuperclassPtr — look for the BRClass object that contains the same class name as the 'superClassName' (a global variable) using depth-first search method
- newOrNot — check if the class is already saved in the binary tree
- setTagZero — set all the tag of the class object link to '0', it should be called every time before the depth-first searching

5.7. The BRowser Class

The BRowser class is a subclass of the CLObject class. It is an interface class to the Petri Net Editor. Each member function is responsible for a function required by the Petri Net Editor:

- newBrowser — initialize the Browser
- getSelectMethods — get the list of selected methods (by double click.)
- addMethodsToList — add the selected methods to the list
- closeWindow — close the browser window

- emptyList — empty the list of selected methods
- getMethodName — get the method name from the method definition

5.8. The BRWindow Class

The BRWindow class is a subclass of the CLWindow class. I just override the "CreateDocument" method to create my own document.

5.9. The BRDocument Class

The BRDocument class is a subclass of the CLDocument class. I override the "CreateView" and "CreatModel" methods to create my own view (BRClassView) and model (BRClass).

6. Conclusion

The Browser used the reusable design of the OSU 3.0 Application Framework. There are 9 classes and 82 methods in the Browser and it reused about 146 methods in the Framework (Table 1.). This is a report of satisfaction and success because software reuse has been encouraged and practiced. And the promise of increased productivity by reusing software is realized. It reduced application development time from overnight to over lunch.

As I said earlier in this report, if more functionalities are needed by the OSU 3.0 tools, enhancements should be made according to these needs. For examples, if a tool needs information about the member fields of a class, the "parse()" method of the BRParser class should be able to get the information and a number of member fields should be added to the BRClass class; if a tool needs to see the text of a class definition, the Browser should be able to open the file that has the definition. Since there is a field in the BRClass class as "fileName" storing the name of the file where the class is defined, it should be possible to accomplish this requirement if it's needed.

Another significant future work on this Browser is to generalize the binary tree structure and depth-first-search in the Browser implementation into a new data structure in the Framework because currently there is no binary tree data structure in the Framework that supports the depth-first-searching. This generalization can basically be done by changing the names of classes, member fields and member functions and by adding more general methods such as to replace a node in the tree, to make a copy of a subtree, and so forth.

Classes	My Code			Reused Code		
	Number of Methods	Lines of Code		Number of Methods	Lines of Code	
		.h	.cp		.h	.cp
BRClass	19	74	201	2	16	25
BRMethod	9	43	136	2	16	25
BRLabel	10	47	70	17	90	300
BRLine	2	4	10	16	44	104
BRParser	14	67	507	5	116	52
BRowser	9	46	173	5	115	52
BRClass View	11	61	448	41	125	770
BRWindow	4	35	60	28	272	713
BRDocment	4	40	61	2	10	15
CLCursor	0	0	3	3	42	28
CLDialog	0	0	4	5	32	51
CLObjList	0	0	6	20	57	180
Total	82	413	1669	146	935	2315
		2082			3250	

Table 1. Comparison of code size

7. References

[Aho 86] Aho Alfred V., Sethi Ravi., Ullman Jeffrey D., *Compilers, Principles, Techniques, and Tools,* Addison-Wesley, Reading, MA, 1986.

[Aho 74] Aho Alfred V., Hopcroft John E., Ullman Jeffrey D., *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[Allen 1990] Allen Daniel K., On Macintosh Programming: Advanced Techniques, Addison-Wesley, Reading, MA, 1990.

[Borenstein 88] Borenstein Philip, *Think's Lightspeed Pascal™ User's Manual,* Symantec Corporation, Cupertino, CA, 1988.

[Borenstein 89] Borenstein Philip, Mattson Jeff, *Think's C™ Object-Oriented Programming Manual,* Symantec Corporation, Cupertino, CA, 1989.

[Budd 90] Budd Timothy, *An Introduction to Object-Oriented Programming,* Addison Wesley, Reading, MA, 1990.

[Fischer 87] Fischer Gerhard, "Cognitive View of Reuse and Redesign" IEEE Software, July 1987, pp 60-72.

[Goldberg 83] Goldberg Adele, *Smalltalk-80: The Interactive Programming Environment,* Addison-Wesley Publishers, Menlo Park, 1983.

[Keh 91] Keh Huan Chao and Lewis T.G., "Direct-Manipulation User Interface Modeling with High-Level Petri Nets," *Proceedings of 19th ACM Computer Science Conference*, March 1991, San Antonio, TX, pp. 487-495.

[Krasner 88] Krasner Glenn and Pope Stephen, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, Vol. 1, No. 3, Aug./Sep. 1988, pp. 26-49.

[Lai 91] Lai Chi, "Adding Object-Oriented Structured Graphics and Graphics Building Block to the MVC Paradigm Application Framework", Project Report, Dept. of Computer Science, Oregon State University, Corvallis, OR, 1991.

[Lewis 90] Lewis T.G., *CASE, Computer-Aided Software Engineering,* Van Nostran Reinhold, New York, NY, 1990.

[Luo 91] Luo Chung Cheng, Lewis Ted, "Oregon Speedcode Universe 3.0 Programming Manual", Project Report, Dept. of Computer Science, Oregon State University, Corvallis, OR, 1991.

[Myers 89] Myers Brad, "User Interface Tools: Introduction and Survey," *IEEE Software*, Vol. 6, No. 1, Jan. 1989, pp. 15-23.

[Myers 90] Myers Brad, "Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces," *IEEE Computer*, Vol. 23, No. 11, Nov. 1989, pp. 71-85.

[Urlocker 89] Urlocker Zack, "Abstracting the User Interface," *Journal of Object-Oriented Programming*, Vol. 2, No. 4, Nov./Dec. 1989.

[Weinand 89] Weinand André, Gamma Erich and Marty Rudolf, "Design and Implementation of ET++, a Seamless Object-Oriented Application Framework," in *Structured Programming*, Vol. 10, No. 2, 1989

[Wilson 90] Wilson David, Rosenstein Larry, and Shafer Dan, *Programming with MacApp*, Addison-Wesley, Reading, MA, 1990.

[Wittel 91] Wittel, Walter I., "Integrating the MVC Paradigm into an Object-Oriented Framework to Accelerate GUI Application Development," Project Report, Dept. of Computer Science, Oregon State University, Corvallis, OR, 1991.

[Woodfield 87] Woodfield Scott N., Embley David W., and Scott Del T., "Can Programmers Reuse Software?" IEEE Software, July 1987, pp 52-59.

Acknowledgements

I would like to thank my Major Professor, Dr. Ted G. Lewis, for the opportunity to work on the testing and documentation of Oregon Speedcode Universe v2.0 and the development of OSU v3.0 Browser. His guidance, support, and encouragement throughout this project has been instrumental to my progress. His Software Engineering series did much to prepare me for work on this project. I would also like to thank my Minor Professor, Dr. Timothy Budd, for his support, concern and caring. His classes did much to develop my understanding concerning computer science and enlarged my interest in this field. I am grateful to Dr. Lawrence Crowl for his willingness to participate on my Graduate Committee.

My special thanks go out to the other members of the OSU v3.0 development team, Dr. Huan Chao Keh, Mr. Walter Wittel, Mr. Chung-Cheng Luo, Ms. Fangchen Lin, Mr. Chih Lai, Mr. Kangho Lee, Mr. Kee-Yun Chan, and Ms. Huei-i Huang, for the excellent framework upon which my Browser was built and for all the helpful explanations. I could not have done this project without their support and friendship. I appreciate this cooperative environment and enjoyed our team work together. Ms. Sherry Yang and Mr. Dan Boerner, two Macintosh specialists in our laboratory, have always been willing to answer my questions.

I would also like to thank my American host family, Professor and Mrs. Walter Kraft, who have been playing the role of parents for me since I came to the United States, and also my family in China and Hong Kong, who gave me the opportunity to come to America, and, to Burnette Or, for his special love to me.