IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661 Volume 3, Issue 3 (July-Aug. 2012), PP 50-55
www.iosrjournals.org

Modeling Object Oriented Applications by Using Dynamic Information for the Iterative Recovery of Collaborations and Roles (Collaboration Browser Model)

[1]Smt Vedavati R Tiwari, [2]Shri M.D Panchamukhi
[1, 2] Dept of Cse Khkie Vidyagiri Dharwad-4

***Abstract**: The paper presents the Collaboration-based or role-based design that decompose an application into tasks performed by a subset of applications' classes. Collaborations provide a larger understanding and reuse than classes. They form an important aid in the maintenance and evolution of software. The extraction of collaborations is therefore an important issue in the design recovery. The paper presents an iterative approach that uses the dynamic information to support recovery and understanding of collaborations.*
The paper presents the details of prototype "Collaboration Browser" modeled for querying of dynamic information using collaborations. The prototype modeled intends to eliminate visual representations that are laborious and expensive to understand and hence represents dynamic information on the collaboration browser in the form of collaboration patterns through which the developer can perform querying to extract and understand the collaborations between instances of classes in Java programming application. The collaboration browser is developed in Java language on Net beans 6.0 platform using Swings feature of Java.
***Keywords:** collaboration-based design, design recovery, program understanding, object-oriented reverse engineering, dynamic analysis.*

I. Introduction

In contrast to procedural applications, where a specific functionality is often identified with a subsystem or module, the functionality in object-oriented systems comes from the cooperation of interacting objects and methods.
In designing object-oriented applications, the importance of modeling how objects cooperate to achieve a specific task is well recognized. Collaboration-based or role-based design decomposes an object-oriented application into a set of collaborations between classes playing certain roles. Each collaboration encapsulates an aspect of the application and describes how instances of class interact to achieve a specific task. The recovery of collaborations from the code is an important aid for understanding and maintaining object-oriented applications. However, detecting and deciphering interactions of objects in the source code is not easy: polymorphism makes it difficult to determine which method is actually executed at runtime, and inheritance means that each object in a running system exhibits behavior which is defined not only in its class, but also in each of its super classes. This difficulty is further aggravated in case of the languages like Java where many instances of different class types collaborate interactively and also repeatedly and methods are never statically bound. To get a better understanding of the dynamic interactions between instances, developers often turn to tools which display the run-time information as interaction diagrams. Designers of such tools are confronted with the challenge of dealing with a huge amount of trace information and presenting it in an understandable form to the developer. Several visualization techniques, such as information murals, program animation and execution pattern views have been proposed to reduce the amount of trace information presented and to facilitate its navigation.
This paper proposes an approach to the recovery and study of collaborations by using dynamic information, but does not rely heavily on visualization techniques. The most visualization tools display an entire trace and give the user a feel for the overall behavior of an application, and paper proposes focusing on understanding much smaller chunks of interactions and the roles that classes play in these. The paper proposes developing a tool prototype, the *Collaboration Browser*, to demonstrate the validity of approach. Collaboration Browser is used to query run-time information iteratively to answer concrete questions about collaborations and interactions between instances of classes of Java programs.
Collaboration-based or role-based designs decompose an application into tasks performed by a subset of the applications' classes. Collaborations provide a larger unit of understanding and reuse than classes, and are an important aid in the maintenance and evolution of the software. The extraction of collaborations is therefore an important issue in design recovery
This paper presentation is not oriented primarily towards program visualization. It simply presents the use of a sequence diagram (visualization) to generate and display the collaboration pattern. The paper concentrates on the prototype tool that supports the recovery and understanding of collaborations.

The paper is organized as follows.
It starts with the design of prototype, its architecture and then implementation details as to how to iteratively recover collaborations and roles. It is a reverse engineering process. It starts from program of object-oriented concept, then gets the run-time execution trace or dynamic information of the program and then designs the model or prototype i.e, collaboration browser used to recover collaborations and roles. This mainly helps the developer who can perform querying to extract and understand the collaborations between instances of classes in Java programming application. The extraction of collaborations is an important issue in the design recovery.

II. The Model Collaboration Browser Design

The design of collaboration browser is preceded by the task that parses the dynamic information obtained from execution trace of sample java application. The tool or the prototype Collaboration browser is developed using JDK 1.6 on netbeans platform with usage of Profile for execution trace of Object Oriented Application. The tools generates collaboration patterns and allows the developer to re query and thus understand and recover the collaborations.

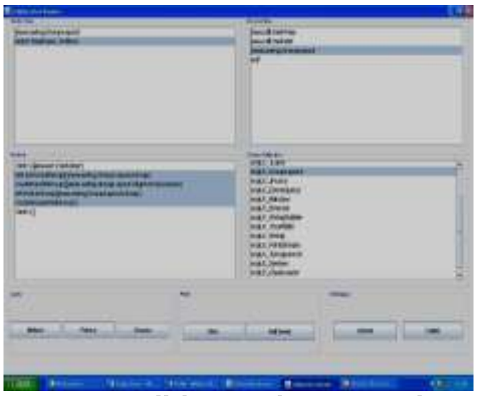


Figure 1. Collaboration browser window

Panels a, b and c list the sender classes, the receiver classes and the invoked methods respectively. Panel d lists collaboration patterns.
It presents the dynamic information to the user through four basic elements of information: sender classes, receiver classes, invoked methods and collaboration patterns. Each of these four elements is displayed on the screen in a separate panel as seen in Figure 1 Panels a, b and c list the sender classes, the receiver classes and the invoked methods respectively. Panels d list collaboration patterns. Collaboration patterns are designed using sequence diagrams. Sequence diagrams are drawn using UML feature of netbeans 6.0. The different buttons are defined for iterative querying of the information displayed on the panels a,b,c and d.

Various functionalities of the command buttons of the browser
Method: After selecting the sender class, receiver class clicking on method button sets focus to all the methods those are interfaces for the selected classes.
Class: After selecting method, clicking on class sets focus to methods sender and receiver class.
Pattern: After selecting method, clicking on pattern sets focus to all collaboration patterns within which the selected method is invoked.
Filter: After selecting the sender class, receiver class clicking on filter filters or deletes interfaces or the methods of the selected two classes from the methods panel.
Self sends: After clicking on sender class, clicking on self send deletes the method from the methods panel for the selected sender class.
Current: Clicking on current displays current execution trace of the program.
Colab: Clicking on Colab sets the display to the base of of the collaboration pattern and also stes focus on sender class and receiver class and also focuses collaboration instances of the two highlighted classes.

III. The prototype architecture:

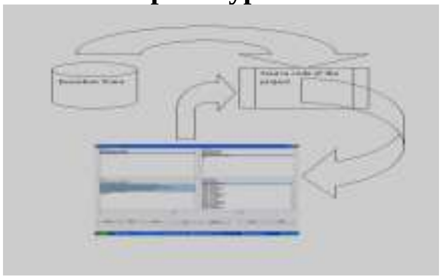


Figure 2. Prototype Architecture

The architecture consists of three layers
Presentation layer: It consists of Graphical User Interface and classification browser that presents execution trace classified as panels for sender class, receiver class and interface and methods. It also consists of panel for collaboration pattern for which values are to be generated depending on execution trace repository.
Logic layer: It consists of java implementation classes and functions for event-driven programming.
Database layer: It consists of schemas, tables of the execution trace.

Architecture Components:
The Collaboration Browser supports three basic kinds of operations:
- querying the current base of dynamic information
- editing the base of dynamic information through filtering out information or loading a collaboration instance,
- querying the dynamic information of the selected collaboration pattern

The Queries are numbered as (Q1,Q2,Q3,Q4 and Q5), editing (E1, E2, E3, E4) and displaying functions (D1, D2, D3).
Query about the interface of a class. These queries are of the form
Q1: what methods of class A are invoked by class B?
Selecting sender and receiver classes, the user requests a list of the methods displayed in panel c.
Q2: which classes are senders and receivers of the following method?
Conversely, selecting a method in panel c, the developer can request the list of senders
and receivers of the selected method.
Query about a collaboration.
By selecting the method of interest (panel c) the following query can be answered:
Q3: what collaboration patterns result from this method invocation?
Panel d lists the collaboration patterns that have selected method in their invoked method list
Q4: For the selected collaboration pattern which are the sender class, receiver class and the methods.
Panel a highlights sender class, panel b highlights receiver class and panel c lists the methods and also highlights the methods those are interfaces between the sender class
Query about a role. These queries are of the form:
Q5: given a collaboration pattern what are all the methods that collaborate the senders and receivers ?
The developer can request a list of receivers which play this role in the same collaboration pattern and the corresponding senders.
Editing the dynamic information. To focus the investigation on the events of interest the developer can filter out method invocation events which are not relevant by specifying sender classes, receiver classes and methods to be filtered out. This reduces the amount of dynamic information to be analyzed and presented.
Another option for focusing on events of interest is to load an instance of one collaboration pattern as the current base of information. This allows the developer to focus on analyzing one collaboration pattern.
The browser queries operate on a current execution trace. When the tool starts up, the current trace corresponds to original execution trace obtained through instrumenting and executing the application. In the course of the iterative process this trace can be edited by the user (using the buttons in panel f) to:
E1: remove method invocation events from the trace for selected senders, receivers and methods,
E2: remove method invocation events which are self-sends,
E3: set the current trace to an instance of a selected collaboration pattern, or
E4: reset the current trace to the original execution trace.

Implementation and testing

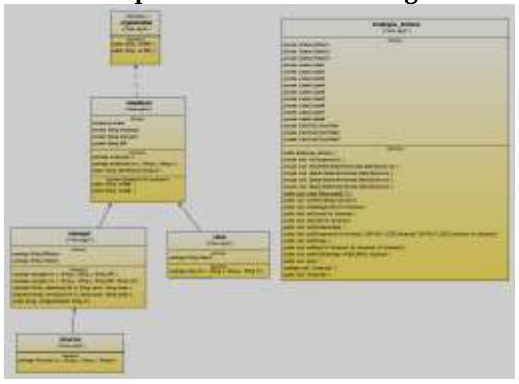


Figure 3 class diagram of the example

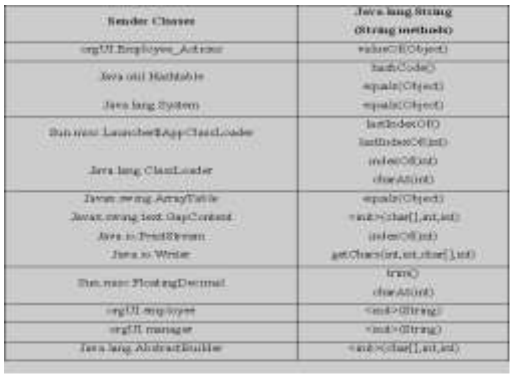
1. The approach uses dynamic information recorded from program execution. For each method invocation event sender class, receiver class and name of invoked method is recorded.
2. Pattern matching is used to eliminate similar execution sequences in the execution trace. This makes the developer not to confront the entire trace of execution but rather with the collaboration patterns.
3. The tool Collaboration browser lets the developer query the dynamic information in terms of classes and interactions of interest. Thus allows the developer to refine the investigation to focus on collaborations of interest

IV. Types of Tests carried

a)Recovery of collaborations of java.lang.String

Start by querying about the interface that class java.lang.String presents to other classes in Java. Table 1 shows the methods of class java.lang.String which are invoked by other Java classes

The Table1 depicts that there is overlap in the table cells. That is, some methods of java.lang.String are invoked by instances of two different classes.



Sender Classes	Java.lang.String (String methods)
orgUI.Employee_Actions	valueOf(Object)

Java.util.Hashtable	hashCode()
	equals(Object)

Java.lang.System	equals(Object)

Sun.misc.Launcher$AppClassLoader	lastIndexOf()
	lastIndexOf(int)

Java.lang.ClassLoader	indexOf(int)
	charAt(int)

Javax.swing.ArrayTable
Javax.swing.text.GapContent
Java.io.PrintStream
Java.io.Writer
	equals(Object)
	<init>(char[],int,int)
	indexOf(int)
	getChars(int,int,char[],int)

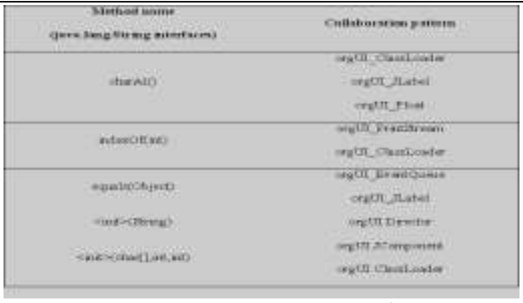
Sun.misc.FloatingDecimal	trim()
	charAt(int)

orgUI.employee	<init>(String)
orgUI.manager	<init>(String)
Java.lang.AbstractBuilder	<init>(char[],int,int)

Table1 java.lang.String interface matrix

For example , both Java.lang.ClassLoader and Sun. misc. Floating Decimal invoke charAt(int). equals(Object) method is invoked by instances of three different classesJava.util.Hashtable, Java.lang.System and Javax.swing.ArrayTable. Both Javax.swing.text.GapContent and Java.lang.AbstractBuilder invoke <init>(char[],int,int) and both orgUI.employee and orgUI.manager invoke <init>(String).

Using the collaboration browser we group methods in the interface String class presents to the ClassLoader and look at the shortest collaboration patterns in which these methods occur. This is accomplished by interactively selecting methods and posing queries of the form Q3:"what are the collaboration patterns in which instance of ClassLoader invokes the methods(indexOf(int) and charAt(int)) on an instance of String class. These queries and the corresponding response are shown schematically in Table 2 and Table 3.

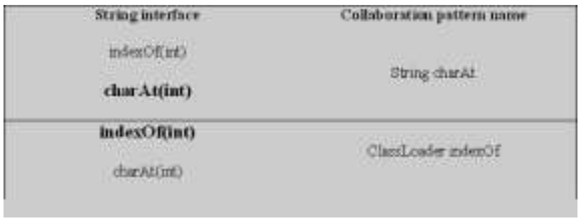


Method name (java.lang.String interfaces)	Collaboration pattern
charAt()	orgUI_ClassLoader orgUI_JLabel orgUI_Float
indexOf(int)	orgUI_EventStream orgUI_ClassLoader
equals(Object) <init>(String) <init>(char[],int,int)	orgUI_EventQueue orgUI_JLabel orgUI Director orgUI JComponent orgUI ClassLoader

Table 2 the contexts i.e. collaboration patterns for the selected method.

The interface indexOf(int)) occurs in two different contexts as shown in Table 2 by loading an instance of the patterns Java.lang.ClassLoader .

Table 3 shows two interfaces .The interfaces of the collaboration pattern are shown in bold face font. The name of the collaboration pattern is named after the method of a class which is the root of a call tree.



String interface	Collaboration pattern name

indexOf(int)
charAt(int)
	String charAt

indexOf(int)
charAt(int)
	ClassLoader indexOf

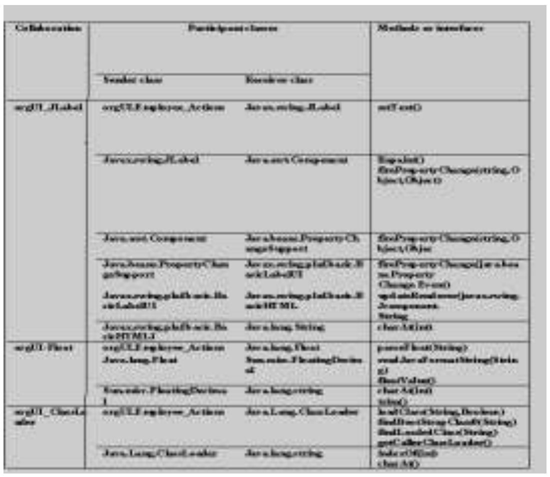
Table 3 collaborations involving the ClassLoader-string interface

You can concentrate on String charAt , to learn about the how String class invokes charAt() method. By loading an instance of the collaboration patterns orgUI_ClassLoader and orgUI_JLabel (E3 function) and querying about interfaces (Q1 and Q2 queries) we can arrive at charAt(int) collaboration which describes the predictable participants as shown in the Table 4. The variation in the call tree resulting from invocation of charAt() is due to the different kinds of user events. The role of classes is represented simply by means of the interface they represent in the collaboration.

The test can be continued for discovering other collaboration patterns in which the instances of java.lang.String participate. By using Q4 ,it is investigated that java.lang.String participates in nine main collaborations. It required 18 queries. The method E1 and E2 are not demonstrated for the example.

V. Conclusions:

The application begins with an execution trace and condenses this information by representing program behavior in terms of collaboration patterns. It presents this information to developers in terms of sender classes, receiver classes, invoked methods and collaboration patterns and allows developers to query each of these items in terms of the others. In this way it lets a developer focus on the aspect of the application of interest without wading through a lot of trace information and the output is checked for 19 collaboration patterns resulted after instrumenting sample code.



Collaboration	Participant classes		Methods or interfaces
	Sender class	Receiver class	
orgUI_JLabel	orgUI.Employee_Action	Javax.swing.JLabel	setText()

	Javax.swing.JLabel	Java.awt.Component	Repaint()
			firePropertyChange(string,Object,Object)

	Java.awt.Component	Java.beans.PropertyChangeSupport	firePropertyChange(string,Object,Object)

	Java.beans.PropertyChangeSupport
	Javax.swing.plaf.basic.BasicLabelUI
		Javax.swing.plaf.basic.BasicLabelUI
		Javax.swing.plaf.basic.BasicHTML
			firePropertyChange(java.beans.PropertyChangeEvent)
			updateRenderer(javax.swing.Jcomponent,String)

	Javax.swing.plaf.basic.BasicHTML	Java.lang.String	charAt(int)

orgUI_Float	orgUI.Employee_Action
	Java.lang.Float
		Java.lang.Float
		Sun.misc.FloatingDecimal
			parseFloat(String)
			readJavaFormatString(String)
			floatValue()

	Sun.misc.FloatingDecimal	Java.lang.string	charAt(int)
			trim()

orgUI_ClassLoader	orgUI.Employee_Action	Java.Lang.ClassLoader	loadClass(String,Boolean)
			findBootstrapClass0(String)
			findLoadedClass(String)
			getCallerClassLoader()

	Java.Lang.ClassLoader	Java.lang.string	indexOf(int)
			charAt()

Table 4. Participant classes for interface charAt(int)

This is how Collaboration Browser is used to discover important collaborations in an application and to understand the roles that classes play in these collaborations. The approach is promising and is used to discover important collaborations in an application and to understand the roles that classes play in these collaborations.

References:

[1] Visualising the behavior of Object Oriented Systems Proceedings of 9th Annual Conference on OOP Systems, languages and Applications
[2] Execution patterns in Object Oriented Visualization IBM Research center, Wim DePauw, David Lorenz
[3] Roles and classes in OOP- Reenskuag
[4] white papers for Toolbox for IT: it.toolbox.com
[5] Getting Dynamic: Java & UML Interaction diagrams by Stephen Palmer
[6] Netbeans 6.0 UML tutorials
[7] MySQL Reference manual
[8] The JFC Swing tutorial by Kathy Walruth