Finding Classes by Execution from an Object-Oriented Class Library

Shaochun Xu               Young Park

Introduction

• Software Reuse? Why Reuse?
• Component Retrieval Approaches
  – Information Retrieval Methods
  – Descriptive Methods
  – Operational Semantic Methods

Introduction

• Steps to organize and retrieve components
  – Organizing the component library
  – Formulating the user query
  – Retrieving components from library
  – Browsing the candidates

Introduction

• Execution-based retrieval?
  – Podgurski & Pierce (1992) proposed behavior retrieval for function components
  – Atkinson & Duke (1995) proposed for OO classes, but without implementation
  – Extended and implemented by Niu (1999)

Introduction

• The class library in previous method

```java
public class ObjectEquals {
    public boolean equals (Object arg1, Object arg2) {
        return arg1.equals (arg2);
    }
    public boolean notEquals (Object arg1, Object arg2) {
        return !arg1.equals (arg2);
    }
    public boolean identical(Object arg1, Object arg2) {
        return arg1==arg2;
    }
    public boolean identical(Object arg1, Object arg2) {
        return arg1==arg2;
    }
}
```
Introduction

- Class only perform math calculations, string manipulations, object comparisons
- Similar to the functions of procedure languages
- Different from the “real world” class

- Person
  - age : String
  - salary : int
  - Address : String
  - name : String
  - Person()
  - Person(String)
  - setAge(String) : void
  - getAge() : String
  - calculateSal(int) : int

- Uncaptureable behavior
  - only method calculateSal(int) can be executed

- Default constructor
  - can not retrieve those classes with user-defined constructors

- Order of arguments
  - require user to input the correct argument order
  - This is not practical
  - reduce retrieval recall

Organize the input program

- Fully consider the “real world” classes
  - Class definition
    - fields
    - constructors
    - observer methods
    - modifier methods
  - Class N={(F), {C}, {O}, {M}}
Organize the input program

• Class behavior
  – A set of responses from the class methods based on a test program
  – A complete class behavior is all the responses from all the methods and constructors

Organize the input program

• In order to disclose the complete behavior of a class
  – create an instance (object)
  – update the states (by calling modifier)
  – inspect each state to determine whether it is in the state as expected (by calling observer)

Organize the input program

• Program executing order
  – execute one or more constructors to create some objects with initial states (if no, a default constructor is called)
  – execute zero/more modifier methods to modify the states of the object
  – apply observer methods to the objects.
  – the result returned by the observer are compared with the user’s input

CM*O+

Organize the input program

• Message
  – a set of input values and types
  – desired return type and value
  – the type is determined by the message (constructor, observer, modifier)

message1{(int, 5), (none, none)}
message2{(int, 2), (boolean, true), (void, none)}
message3{(none, none), (int, 2)}

Organize the input program

• Test program
  – A set of messages to be sent to the system
  – expected behavior

[(int, 5), (int, 2), (boolean, true), (none, none)]
with expected behavior {none, none, 2}

Retrieve classes from library

• Type Checking & Run-time storage
  – Store the info during first time

For each constructor
  if (it is the first constructor)
    load class from the library
    stored constructor info
  if (arglist==parameterlist)
    execute constructor
  else skip
Retrieve classes from library

- Type Checking & Run-time storage
  
  For each method
  
  if (it is the first method)
    load class from the library
    stored method info
  
  if (arglist == parameterlist && return types matches)
    execute method
  
  else skip

Retrieve classes from library

- Retrieval regardless of argument order
  
  - message {(int, 5), (String, “li”), (float, 6.0), (none, none)}
    
    - (int, String, float)
    - (int, float, String)
    - (String, float, int)
    - (String, int, float)
    - (float, String, int)
    - (float, int, String)
  
  - Recursively execution on these compositions

Retrieve classes from library

- Matching process and candidate classes
  
  - A match is found, if the message and the method have:
    
    - the same type (constructor, observer, and modifier)
    - the same number of arguments and convertible types
    - the same or convertible return types
    - the same or tolerable return values

Retrieve classes from library

- Matching process and candidate classes
  
  - Matching algorithm
    
    for each test message
    
    if (it is constructor)
      if (argument list matches parameterlist in each constructor)
        construct the instance
        return true
      else return false

Else //it is method

if (the return types match)
  
  if (check each case of argument list
    with parameter list of each method)
    
    if (there is no instance)
      call default constructor
      executes on the method
    
    if (result same as the expected)
      return true
  
  return false

else return false

Retrieve classes from library

- Matching process and candidate classes
  
  - Candidates classes
    
    - matches info stored in a vector associated with the class name
    - classes with at least 1 match
    - display the candidates (the most match class comes first)
Retrieve classes from library

- Inheritance, Overloading and Overriding
  - Inheritance
    - subclass and superclass
    - inherited methods should be executed and counted if the match is found

- Overloading
  - Selection of the actual method is based on the type matching between arguments of messages and parameter of methods
  - Do not need to take care specially

- Overriding
  - overriding method and overridden method
  - overriding method does not cause side effect
  - overridden method is hidden from the subclass

Browse the Candidates

- Directly browse the code (implementation) is practical and useful
  - Class library is small scale
  - Candidate classes are not too many

Implementation

Main Interface

Input interface for observer method

Retrieval Interface
An Example of Library

- A class library contains 7 classes
  - Geometry related classes: Shape, Circle, Triangle, Rectangle, Equilateral, Isosceles and Square
  - They are related.

Conclusions

- An execution-based class retrieval method is proposed for OO class library
  - Capturing the complete behavior of a class
  - Considering the OO characteristics (inheritance, overloading and overriding)

- Techniques
  - Providing user-defined constructors
  - Neglecting argument-order
  - Storing the class info during execution

Future work

- It can be extended to include the user-defined types
- Access modifier matching can be added to before execution

Conclusions

- Precision
  - High precision because it actually compares the behavior of class with expected behavior
  - Improve precision by allowing input constructor, observer and modifier

- Recall
  - Improve recall by allowing defined constructor to create instance & by neglect argument order

- Efficiency
  - Type checking & run time storage