Zsolt Balanyi and Rudolf Ferenc
University of Szeged
Hungary

Motivation

- Design patterns
  - represent a high level of abstraction in OO designs
- Recognizing them in the source can help
  - program comprehension
  - code documentation
  - correct pattern implementation inspection

Design Pattern Categories

- Creational
  - concerned with creation of objects
  - function body analysis is needed (object creations)
  - difficult to recognize
- Structural
  - deal with composition of classes and objects
  - class structure is enough
  - easiest to recognize
- Behavioral
  - describe class interactions and responsibility distributions
  - deeper function body analysis is needed
  - hardest to identify

Mining Process

- Analyze the C++ source code with Columbus tool
  - build ASG
- Extract from ASG
  - class diagram
  - call graph
  - object creation graph
- Load design pattern descriptions
  - from DPML – Design Pattern Markup Language
  - into XML DOM tree
- Run matching algorithm

Design Pattern Markup Language

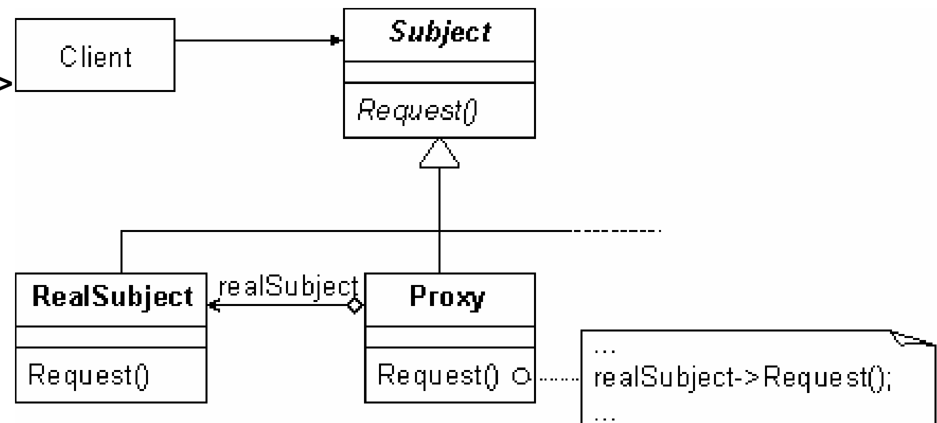
- XML-based format
  - grammar described with DTD
  - clear syntax
- Pattern library
  - contains most of the well known Gamma patterns
- Pattern library can be easily modified
  - adapt them to own needs
  - create new pattern descriptions or simply classes in certain relations with each other
- Algorithm is independent from the pattern descriptions

DPML Example

```
<DesignPattern name='Proxy'>

<Class id='id10' name='Subject' isAbstract='true'>
  <Operation id='id11' name='Request'
                    isVirtual='true' isPureVirtual='true'>
    <hasTypeRep ref='id50'/>
  </Operation>
</Class>

<Class id='id20' name='Proxy'>
  <Base ref='id10'/>
  <Aggregation ref='id30'/>
  <Operation id='id21' name='Request'
                    isVirtual='true‘ isPureVirtual='false'>
    <defines ref='id11'/>
    <calls ref='id31'/>
    <hasTypeRep ref='id50'/>
  </Operation>
  <Attribute id='id22' name='realSubject'>
    <hasTypeRep ref='id52'/>
  </Attribute>
</Class>
```




```
<Class id='id30' name='RealSubject'>
  <Base ref='id10'/>
  <Operation id='id31' name='Request'
           isVirtual='true' isPureVirtual='false'>
    <defines ref='id11'/>
    <hasTypeRep ref='id50'/>
  </Operation>
</Class>
…
```

Pattern Miner Algorithm

- Preparation
  - analyze source code, prepare class diagram, call graph, object creation graph, load DPML
- Search for one design pattern at a time
- Gathering candidate source classes for each pattern class which have
  - appropriate number of operations and attributes with the right properties (virtuality, accessibility, storage class, etc., but no type checking in this step)
  - appropriate number of relations (inheritance, composition, aggregation, association)

Pattern Miner Algorithm (cont.)

- Filtering the candidate classes
  - if two pattern classes are related in some way then all candidates of the first one have to be related in the same way to at least one candidate of the second one
  - iterate on all candidates of all pattern classes until no further candidate is removed
    - remove the candidate if it does not satisfy the above condition
- Binding classes
  - all combinations of the candidate classes have to be tested
  - if a combination of candidates has all the required relations then class element binding is performed

Pattern Miner Algorithm (cont.)

- Binding candidate class elements
  - all combinations of attributes and operations have to be tested
  - type checking is also done
  - if a right combination is found then low-level check is performed
- Low-level check
  - check whether the bound operations perform the needed
    - call delegations
    - object creations
    - operation redefinitions
  - if it passed then we found a design pattern instance

Optimizations

- The outlined algorithm gives only the main idea on how it works
- It is precise but far not optimal in performance
- In the implementation there are two optimizations which speed up the search
  - candidates list reduction
  - cut in operation and attribute matching

Experimental Results

- Four real-life, open-source C++ projects
  - Jikes (ca. 75 KLOC, 329 classes)
  - Leda (ca. 85KLOC, 1617 classes)
  - StarOffice Calc (ca. 1.2 MLOC, 5051 classes)
  - StarOffice Writer (ca. 1.5 MLOC, 6729 classes)
- 26 different patterns searched
  - 15 strict + 11 soft
- 15 different kinds found
  - 6 strict + 11 soft
  - 978 different instances
- Speed (P4-2000): Jikes – 4 min, Writer – 9 hours

Experimental Results (cont.)

- Precision
  - the structure of the found instances was always correct
  - but it was not always the intent of the programmer
  - all checked manually
  - except the Adapter Object and its soft version (350 instances…)

Statistics	Jikes	Leda	Calc	Writer
Builder	-	-	0% (2)	14% (7)
Builder soft	-	-	0% (17)	11% (9)
Factory Method soft	-	-	100% (1)	100% (9)
Prototype	100% (1)	-	-	100% (1)
Prototype soft	100% (1)	-	-	100% (1)
Adapter Class	-	-	-	0% (16)
Adapter Class soft	-	-	0% (13)	0% (16)
Bridge soft	-	-	100% (73)	100% (80)
Proxy	0% (36)	-	-	50% (4)
Proxy soft	18% (44)	-	-	60% (5)
Iterator soft	-	-	0% (1)	-
Strategy	100% (4)	100% (1)	100% (10)	100% (5)
Strategy soft	100% (12)	100% (2)	100% (20)	100% (32)
Template Method	100% (5)	-	100% (94)	100% (101)
Visitor soft	-	-	-	0% (5)

Future Work

- Take into account STL containers
  - would allow us to find patterns which use lists (Flyweight, Mediator and Observer)
- Experiment with pattern descriptions and make them more sophisticated
- Prepare a benchmark
  - testing pattern miner tools
  - manually check
  - deficiencies could be easily discovered
- Make it part of the official Columbus distribution

Summary

- We presented a method for discovering design patterns in C++ source code
- New approach which takes into account
  - call delegations
  - object creations
  - operation redefinitions (overriding)
- We introduced a new language for pattern description
  - DPML – Design Pattern Markup Language
- We tested the system on four real-life C++ projects