Reverse Engineering of Design Patterns

Jinwoo Lee

Information Security Lab

January 11, 2012

Contents

Introduction (1/6)

- Reverse Engineering
  - Analyzing a subject system to create representations of the system at a higher level[1]
  - In practice, reverse engineering may divided into two types
    - Case 1 source code is available for the software (but not documented well)
    - Case 2 no source code is available for the software
  - Case 2 of the term is well known

[1] Chikofsky, E.J.: Reverse Engineering and Design Recovery, IEEE Computer Society 1990

Introduction (2/6)

- Why do we need Reverse Engineering?
  - Recovery of lost information
  - Assisting with maintenance
  - Migration to another HW/SW platform
  - Facilitating software reuse
  - Analysis for software assurance
  - Analysis for investigation of cyber crime (malicious code)
  - Analysis for copyright invasion

Introduction (2/6)

- Why do we need Reverse Engineering?
  - Recovery of lost information
  - Assisting with maintenance
  - Migration to another HW/SW platform
  - Facilitating software reuse
  - Analysis for software assurance
  - Analysis for investigation of cyber crime (malicious code)
  - Analysis for copyright invasion

Introdunction (3/6)

- Analysis for Copyright Invasion
  - Main technique:
    - Detecting similar code fragments in software
    - Called simple clones detection
    - Cannot detect structural clones
  - Structural clones:
    - Higher-level clone
    - Architecture clone, Tool clone, Framework clone, etc.

Introduction (4/6)

- Design Patterns
  - Are a general reusable solution to a commonly occurring problems
  - Are a description or template for how to solve a problem that can be used in many different situations
  - Gained popularity after the book Design Patterns: Elements of Reusable Object-Oriented Software[2]

[2] Gamma et al.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley 1994

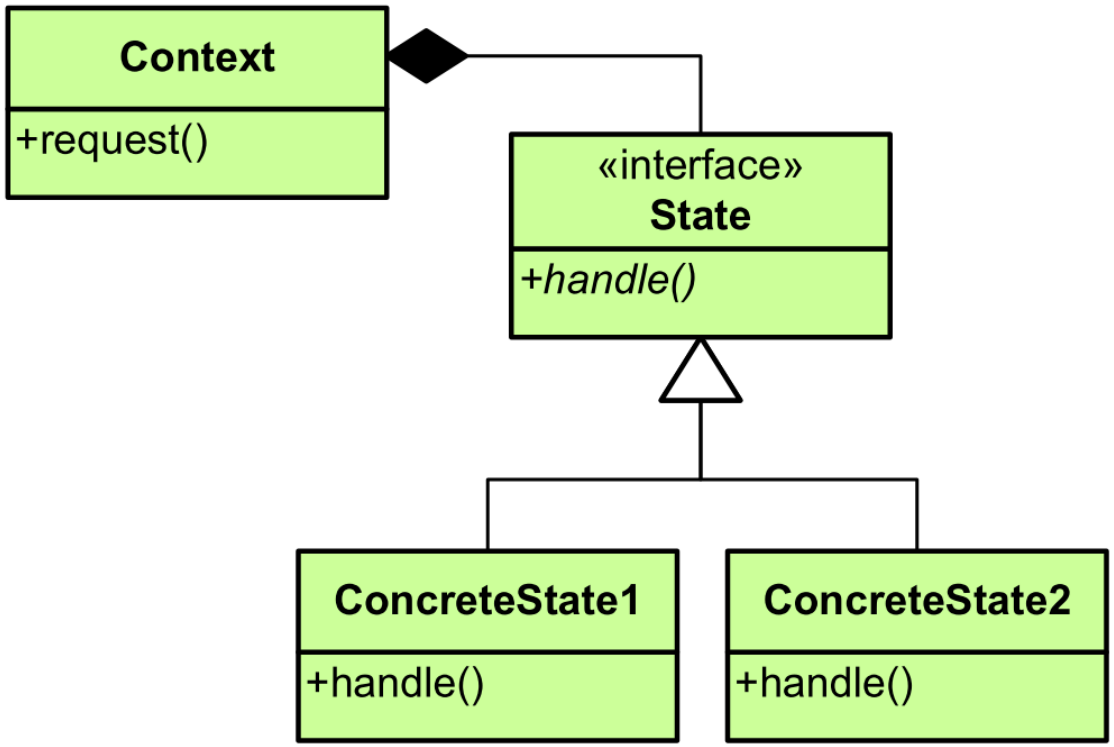
Introduction (5/6)

- Design Patterns
  - C: creational, S: structural, B: behavioral

C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	CoR	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

Introduction (6/6)

- Example: State Pattern
  - Type: Behavioral
  - Allow an object to alter its behavior when its internal state changes



- Other Patterns: Facade, Prototype, etc.

Reverse Engineering of Design Patterns (1/4)

- Motivation
  - Design patterns help to reuse expert experience in system design
    - have been extensively applied in industry
  - After implementation, pattern-related knowledge is generally no longer available from source code
  - "It takes a professional programmer about 6-9 months to become really proficient with a larger framework"[3]
  - Mining the instances of design patterns from the source code can assist the architect
  - Comparing results from pattern-based reverse engineering helps detecting higher-level clones in software

[3] Booch, G.: Managing the Object-Oriented Project, Addison-Wesley 1996

Reverse Engineering of Design Patterns (2/4)

- Difficulties
  - Design patterns are just a description — implementation differs
    - No standard, no rigorous definition for the design patterns
    - Many variations for each design pattern
  - "The main arguments are that patterns can be implemented in many different ways without ever being the same twice"[4]
  - Open source, or in-house systems are used for testing
    - Most of them do not provide design documents that identify the design patterns

[4] Rudolf, K.: Pattern-Based Reverse-Engineering of Design Components, ICSE 1999

Reverse Engineering of Design Patterns (3/4)

- Difficulties Single-thread singleton implementation versus multi-thread singleton implementation

```
public sealed class Singleton
{
   static Singleton instance=null;

   Singleton()
   {
   }

   public static Singleton Instance
   {
      get
      {
         if (instance==null)
         {
            instance = new Singleton();
         }
         return instance;
      }
   }
}
```

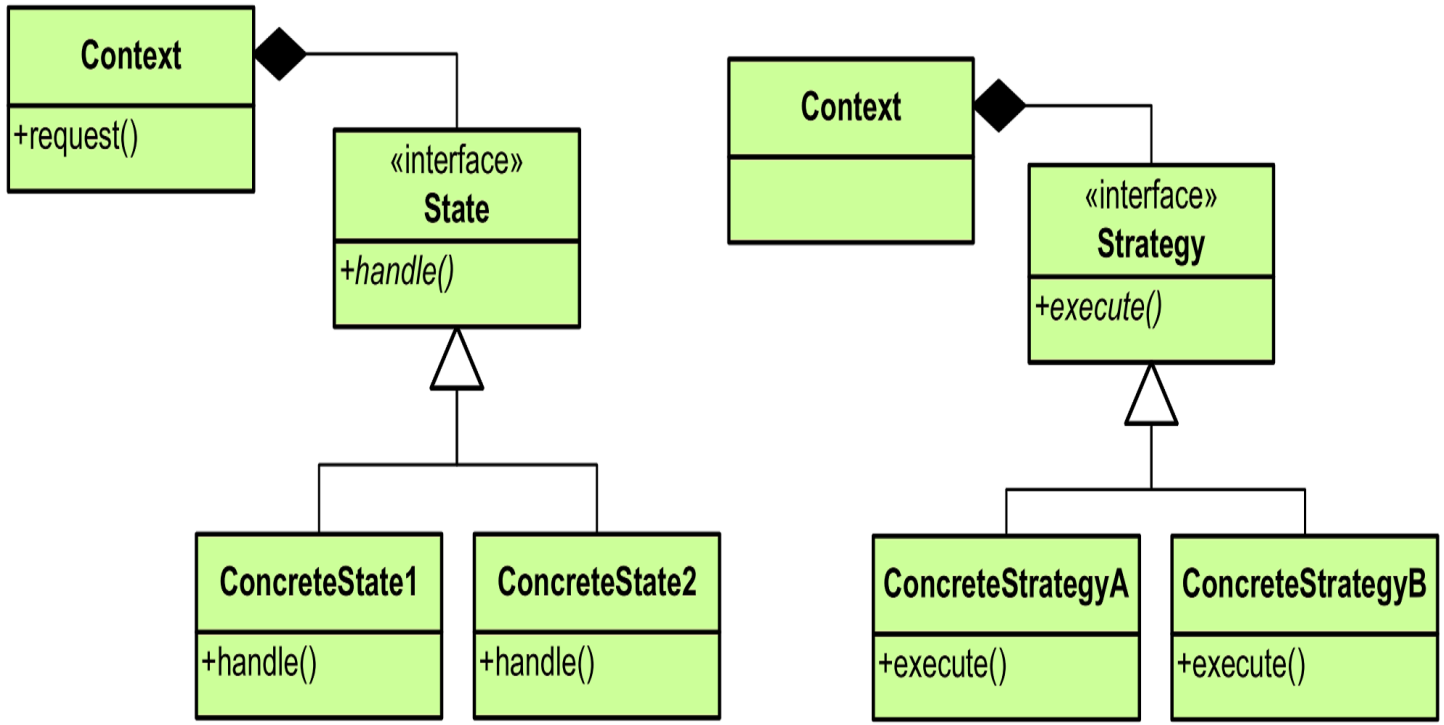
```
public sealed class Singleton
{
   static Singleton instance=null;
   static readonly object padlock = new object();

   Singleton()
   {
   }

   public static Singleton Instance
   {
      get
      {
         lock (padlock)
         {
            if (instance==null)
            {
               instance = new Singleton();
            }
            return instance;
         }
      }
   }
}
```

Reverse Engineering of Design Patterns (4/4)

- Difficulties Two different patterns with same structure



Detecting Steps

Original Source Code

Pre-Processing

Main Processing

Human Analysis
(Supported by tools)

Analyzed Patterns

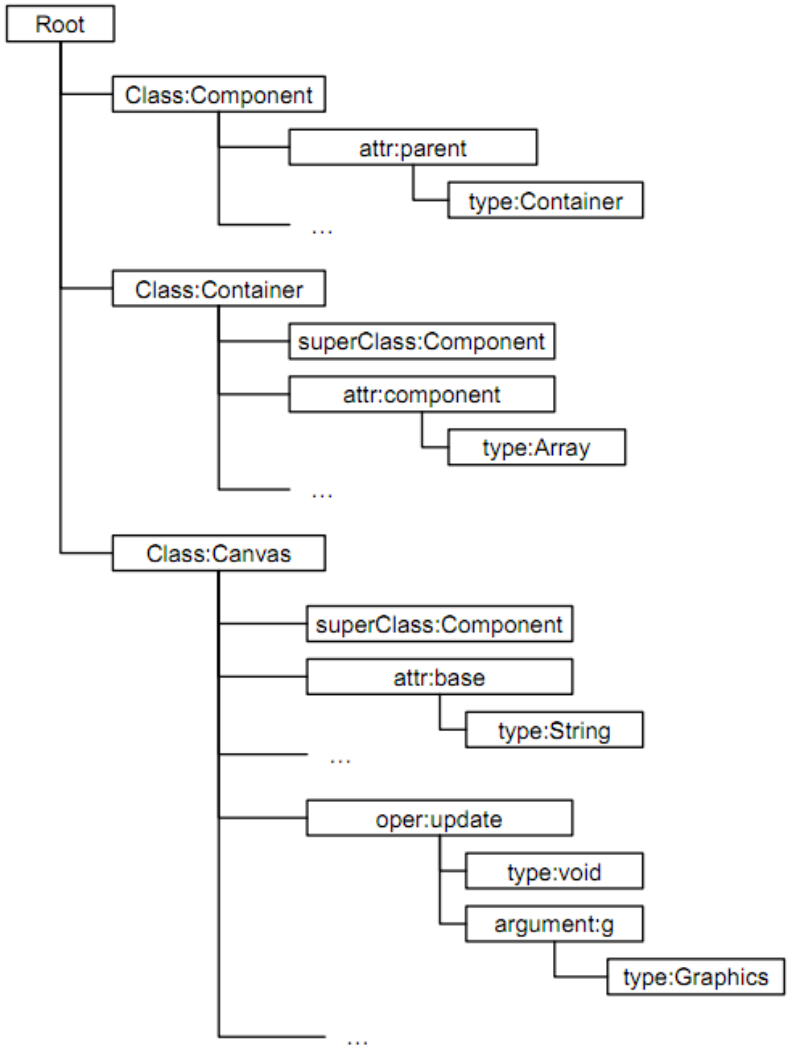
Adapter	18
Composite	1
Decorator	3
Factory Method	3
Observer	5
Prototype	1
Singleton	2
State	23
Template Method	5
Visitor	1

Intermediate Representations (1/3)

- Many class information gathering tools exist
  - There is no point discovering design patterns from scratch
- Current approaches usually use some existing tools to
  - Transform the source code into some intermediate form
  - Thus can reduce search complexity

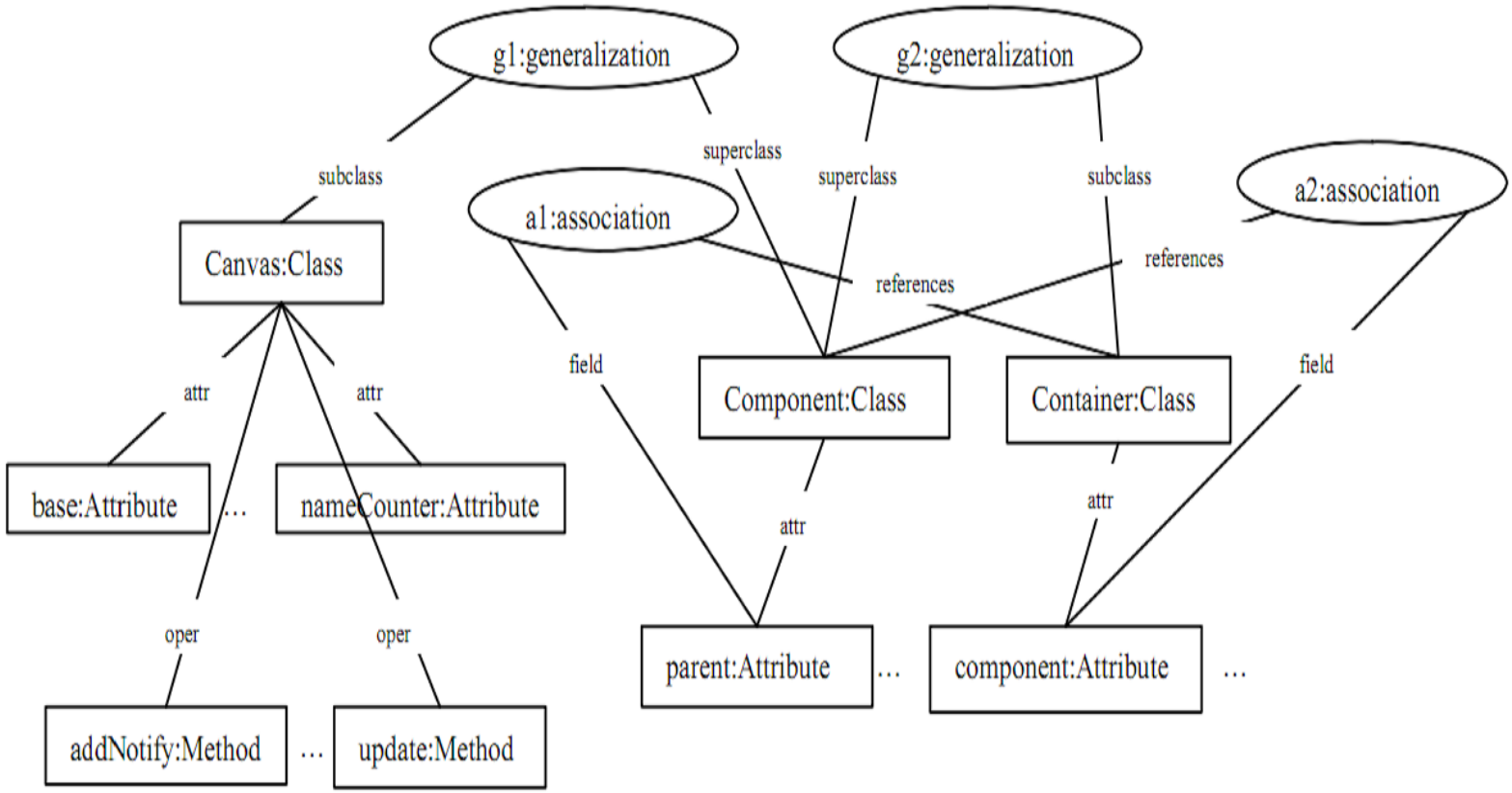
Intermediate Representations (2/3)

- Abstract Syntacx Tree (AST)



Intermediate Representations (3/3)

- Abstract Semantic Graph (ASG)



Three Pattern Aspects

- Structural
  - Relatively easy to detect from code
  - Find relationships between classes
  - Also inspect class attributes and methods
- Behavioral
  - Structural approaches are unable to identify patterns that are structurally identical
    - ex) state vs. strategy and chain of responsibility vs. decorator
    - But they are differ in behavior
  - These can be identified with
    - Dynamic analysis using runtime data
    - Machine learning
- Semantic
  - Take advantage from common naming conventions
    - ex) word “instance” usually stands for the singleton
    - ex) word “factory” usually stands for the factory
    - ex) word “strategy” usually stands for the strategy

Visualization and Human judgement

- Visualization is important in reverse engineering
  - User can easily understand the system
- Some tools provide visualization support
- ex) SPOOL[5], Visualizing Design Patterns[6]

- People know what they are looking for
- Can take advantage from human knowledge
  - Semi-automatic process
  - ex) FUJABA[7]
- However, human interaction slows down the process

[5] Rudolf, K.: Pattern-Based Reverse-Engineering of Design Components, ICSE 1999

[6] Jing, D. et al.: Visualizing Design Patterns in Their Applications and Compositions, IEEE Transactions on Software Engineering 2007

[7] Niere, J et al.: Towards pattern-based design recovery, ICSE 2002

Discovered Patterns

- Architecture and design patterns detected by different approaches[8]

Authors \ Design Patterns	Abstract Factory	Adapter /Command	Builder	Bridge	Chain of Responsibilities	Command	Composite	Decorator	Facade	Factory Method	Flyweight	Mediator	Observer	Prototype	Proxy	Singleton	Strategy/State	Template Method	Visitor
Kramer 1996		×		×			×	×							×				
Seemann 1998				×			×										×		
Antoniol 1998		×		×			×	×							×				
Keller 1999				×						×								×	
Blewitt 2001				×		×				×					×	×			
Asencio 2002	×			×				×		×					×	×	×		
Niere 2002				×			×										×		
Heuzeroth 2003					×		×	×				×	×						×
Balanyi 2003	×	×		×	×			×		×				×	×	×	×	×	×
Gueheneuc 2004	×	×	×			×	×	×		×			×	×		×	×	×	×
Costagliola 2005		×		×			×	×							×				
Ferenc 2005		×															×		
Hericko 2005							×						×				×		
Kaczor 2006	×						×												
Shi 2006	×	×		×	×		×	×	×	×	×	×	×		×	×	×	×	×
Tsantalis 2006		×		×			×	×		×			×	×		×	×	×	×
Dong 2007		×		×			×										×		

[8] Jing, D. et al.: A Review of Design Pattern Mining Techniques, IJSE 2009

High error rate

- Different results from the same system of the same version[9]

Systems	JHotDraw5.1		JRefactory2.6.24		JUnit3.7	
Authors / Patterns	Tsantalis *et al.*	Guéhéneuc *et al.*	Tsantalis *et al.*	Guéhéneuc *et al.*	Tsantalis *et al.*	Guéhéneuc *et al.*
Adapter	18	1	7	7	1	0
Composite	1	1	0	0	1	1
Decorator	3	1	1	0	1	1
Factory Method	3	3	4	1	0	0
Observer	5	2	0	0	4	3
Prototype	1	2	0	0	0	0
Singleton	2	2	12	2	0	2
State	23	2	12	2	3	0
Template Method	5	2	17	0	1	0
Visitor	1	0	2	2	0	0

[9] Jing, D. et al.: A Review of Design Pattern Mining Techniques, IJSE 2009

Possible Future Works

- Reducing error rate
- Benchmark making
  - Previous approaches present different results for the same pattern
  - No benchmark system available so far
  - Bechmark making can be a possible future work[10]
- Reverse engineering environment for modified design patterns
  - All previous works have its focus on GoF's design patterns
  - Many other design patterns are in use
  - ex) Multiton, Object pool, Front controller, etc.

[10] Jing, D. et al.: Architecture and Design Pattern Discovery Techniques – A Review, ICSE 2007