Design Patterns (cont)
Softwaretechnik

Matthias Keil
Institute for Computer Science
Faculty of Engineering
University of Freiburg

16. Mai 2013
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample Code
- Known Uses
- Related Patterns
### Intent

- Control access to object

### Motivation

- Multi-media editor loads images, audio clips, videos etc on demand
- Represented by proxy in document
- Proxy loads the “real object” on demand
Structural Pattern: Proxy

Motivation

Albert-Ludwigs-Universität Freiburg

```
if (image == NULL)
    image = loadImage(fileName);
if (image == NULL)
    image.draw();
else return image.getExtent();
```

```
if (image == NULL)
    return extent;
else return image.getExtent();
```
Structural Pattern: Proxy

Structure

Albert-Ludwigs-Universität Freiburg

Client

Subject
request()
...

Proxy
request()
realSubject
realSubject.request(); ...
1 *Remote Proxy* Communication with object on server (CORBA)

2 *Virtual Proxy*
   - Creates expensive objects on demand
   - Delays cost of creation and initialization

3 *Protection Proxy* controls access permission to original object

4 *Smart Reference* additional operations: reference counting, locking, copy-on-write
Intent

- Extend object’s functionality dynamically
- More flexible than inheritance
- Graphical object can be equipped with border and/or scroll bar
- Decorator object has same interface as the decorated object
- Decorated forwards requests
- Recursive decoration

```
:BorderDecorator
  component

:ScrollDecorator
  component

:TextView
```

Matthias Keil
Design Patterns (cont)
16. Mai 2013
Structural Pattern: Decorator

Motivation (cont)

Albert-Ludwigs-Universität Freiburg
Structural Pattern: Decorator

Structure

Albert-Ludwigs-Universität Freiburg

Component
operation()

ConcreteComponent
operation()

Decorator
operation()

ConcreteDecoratorA
addedState
operation()

ConcreteDecoratorB
operation()
addedBehavior()

super.operation();
addedBehavior();
component.operation()

component.operation();

super.operation();
addedBehavior();

Matthias Keil
Design Patterns (cont)
Structural Pattern: Decorator

Consequences

- More flexible than inheritance
- Avoids feature-laden classes high up in the hierarchy
- Decorator ≠ component
- Lots of little objects → hard to learn and debug

Applicability

- Dynamically add responsibilities to individual objects
- For withdrawable responsibilities
- When extension by inheritance is impractical
Intent

- Recursive object structures
- Uniform treatment of leaf components and containers
- arithmetic expression consists of subexpressions evaluation follows tree structure

```java
return e1.eval()+e2.eval();
```
Structural Pattern: Composite

Structure

Albert-Ludwigs-Universität Freiburg

Client

Component

operation()
add(Component)
remove(Component)
getChild(int)

Leaf

operation()

Composite

operation()
add(Component)
remove(Component)
getChild(int)

forall g in children
g.operation();
Structural Pattern: Composite

Consequences

Albert-Ludwigs-Universität Freiburg

- Uniform client code
- Easy to add new composite classes as well as leaf classes

Applicability

- Recursive object structures

Related Patterns

- Decorator
Comparison of Structural Patterns
Albert-Ludwigs-Universität Freiburg

similar underlying concepts:
- class-based $\rightarrow$ inheritance
- object-based $\rightarrow$ object composition

different goals

**Adapter vs. Bridge vs. Facade**

- all: flexibility through indirection

- differences
  - **Adapter**: reconciling differences between existing interfaces
  - **Facade**: bundling of interfaces
  - **Bridge**: interface with multiple, dynamically exchangeable implementations
- both: recursive composition to organize open-ended number of objects
- Decorator adds responsibilities without subclassing
- Composite enables uniform processing of object graphs
- complementary → often used in concert
Decorator vs. Proxy

Albert-Ludwigs-Universität Freiburg

- both: indirection and forwarding

- Proxy:
  - controls access to particular object
  - not recursive

- Decorator:
  - stepwise addition of responsibilities
  - recursive