Proxy Pattern

CSCI-4448 - Boese

University of Colorado **Boulder**

Objectives

- Problem
- Definition
- Why
- How
- Proxy Types
  - Remote Proxy
  - Virtual Proxy
  - Protection proxy
  - Smart Reference
- Design Considerations

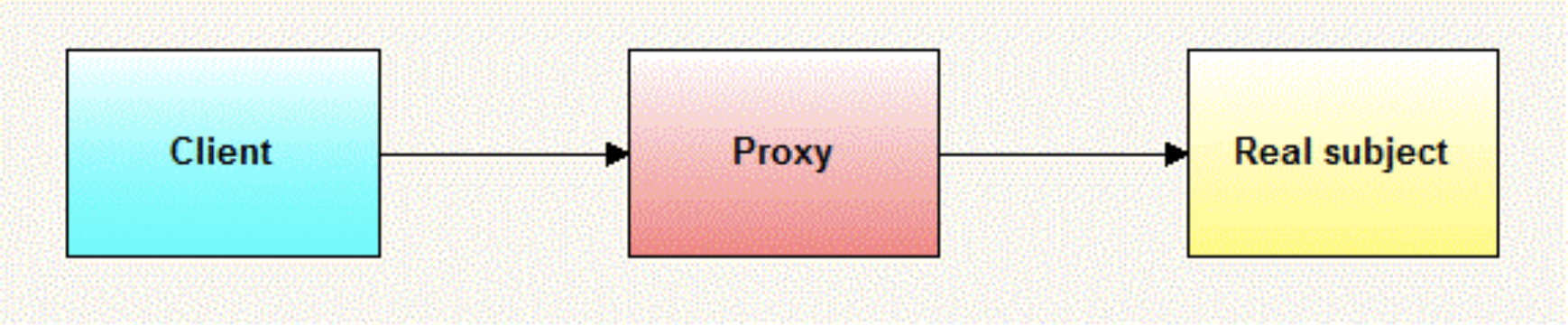
Problem

Why

- Object creation can be expensive
  - Loading images from disk
  - Loading information from a database
  - Accessing information from a server
- May want to avoid expensive, up front object creation
  - Loading objects on demand would be preferable
- May want to deal with complex object access
  - Locking for concurrently accessed data

Proxy

- create a wrapper class over real object



Proxy

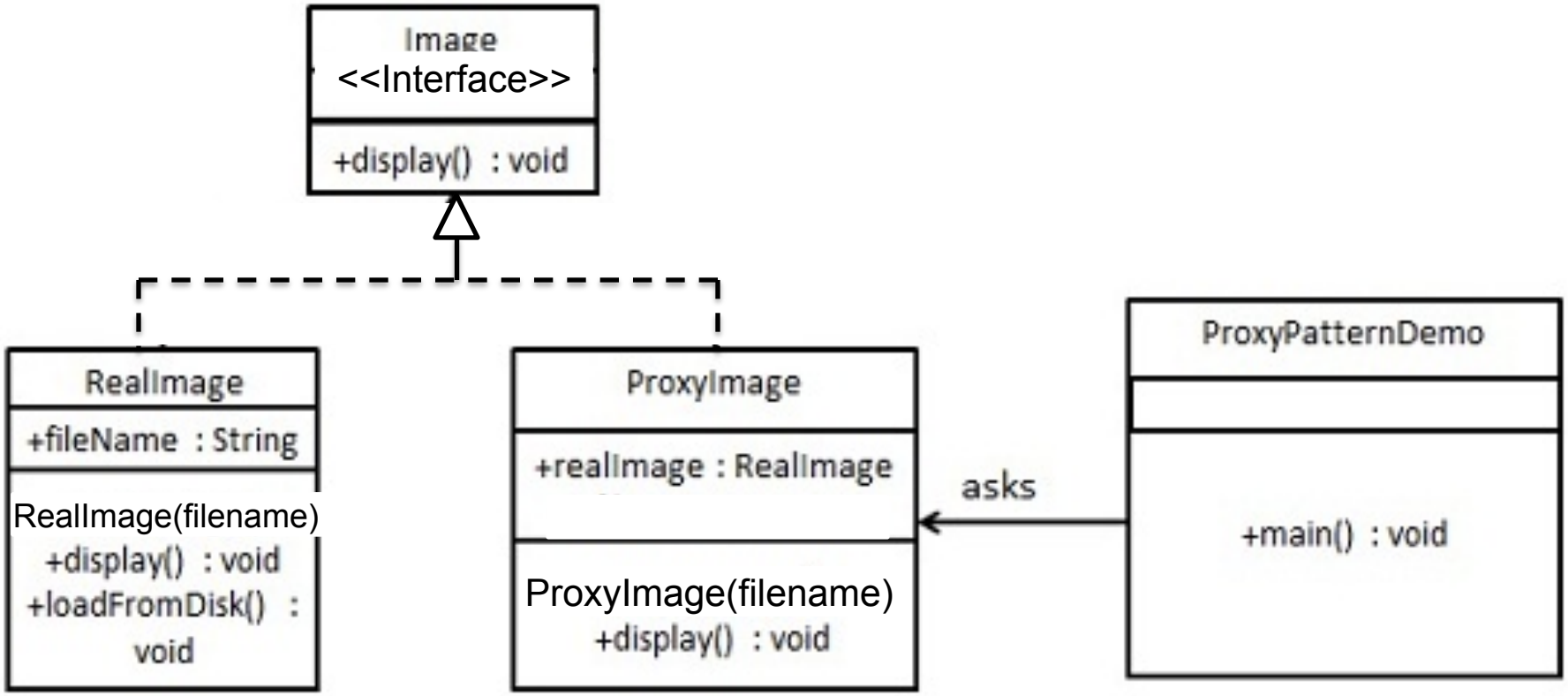


Image
<<Interface>>
+display() : void
RealImage
+fileName : String
RealImage(filename)
+display() : void
+loadFromDisk() : void
ProxyImage
+realImage : RealImage
ProxyImage(filename)
+display() : void
asks
ProxyPatternDemo
+main() : void

Definition

"Provide a surrogate or placeholder for another object to control access to it."

-Gang of Four

Definition

- **Name** “Proxy”
  - A substitute or stand-in for some function or object
  - An object which has the authority or power to act for another
- **Intent**
  - Object creation / access may be expensive
    - *Still need a placeholder to interact with*
    - *Proxy decides when to perform an expensive operation*
  - Control access to real object so in turn you can add extra functionalities to real object without changing real object's code

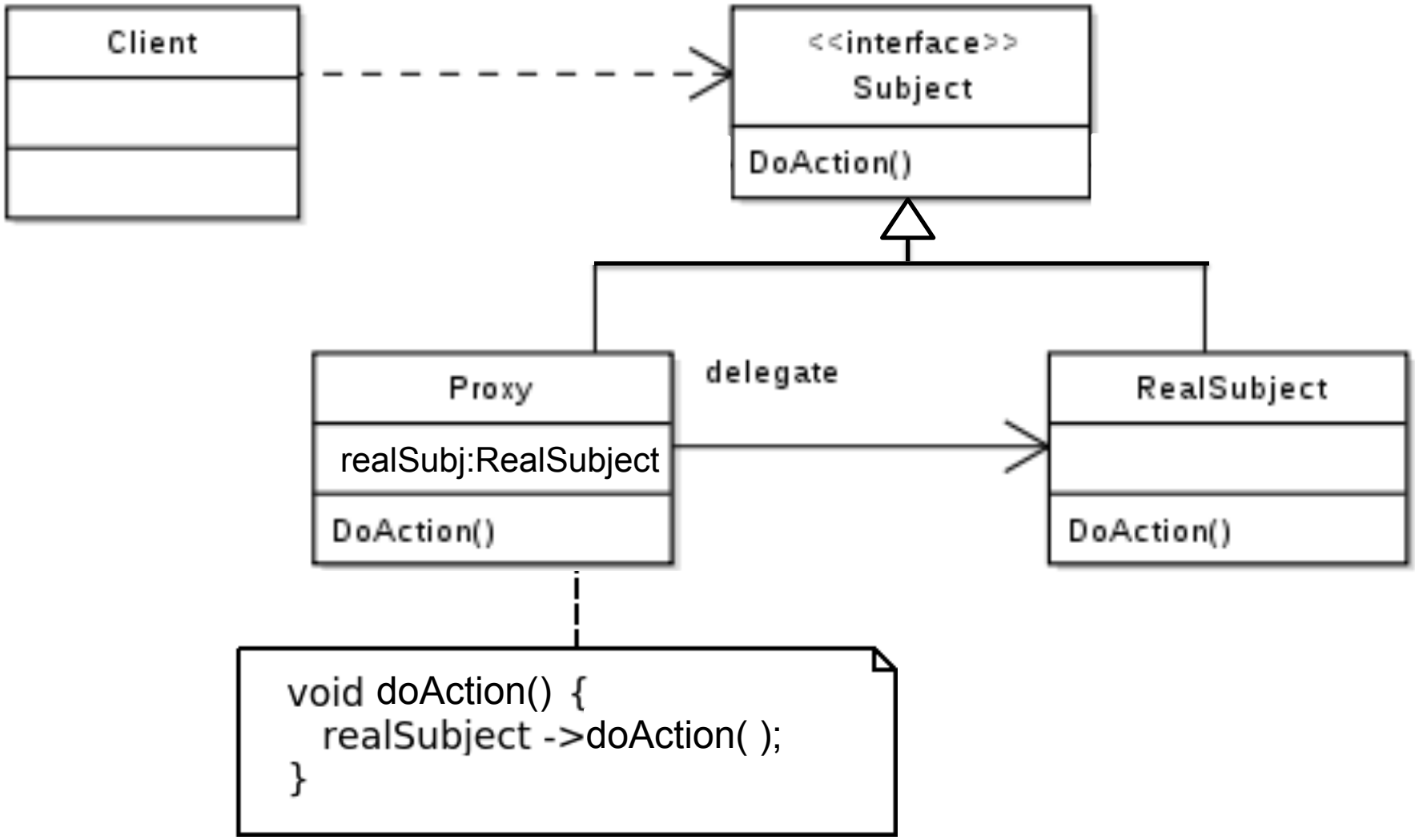
How

Composite Pattern - Participants

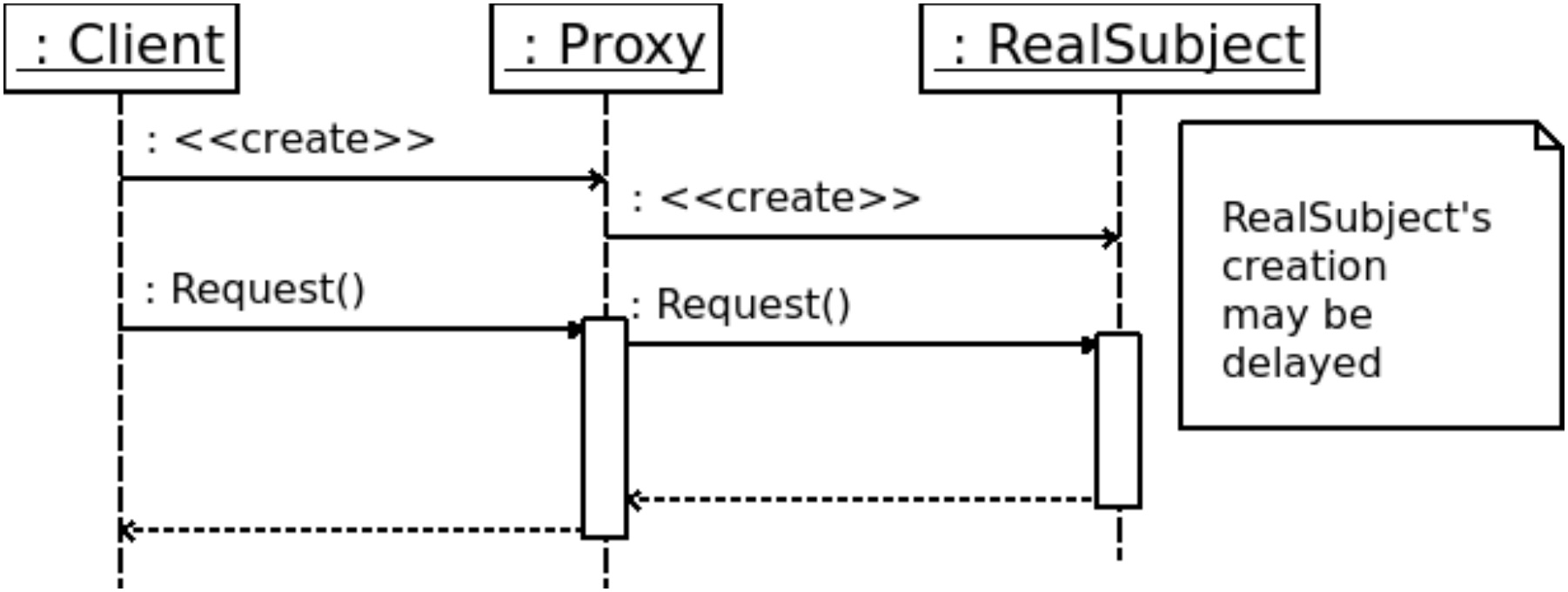
Participants

- **Proxy**
  - Maintains a reference that lets the proxy access the real Subject
  - Provide an *identical* interface to the Subject's so the proxy can be substituted for the real subject
  - Responsible for access to the real subject
- **Subject**
  - Defines the common interface for RealSubject and Proxy so that Proxy can be used anywhere RealSubject is expected
- **RealSubject**
  - Defines the real object that the proxy represents

Structure


Client
<<interface>>
Subject
DoAction()
Proxy
realSubj:RealSubject
DoAction()
delegate
RealSubject
DoAction()
void doAction() {
realSubject ->doAction( );
}

Proxy Pattern – Behavior Example

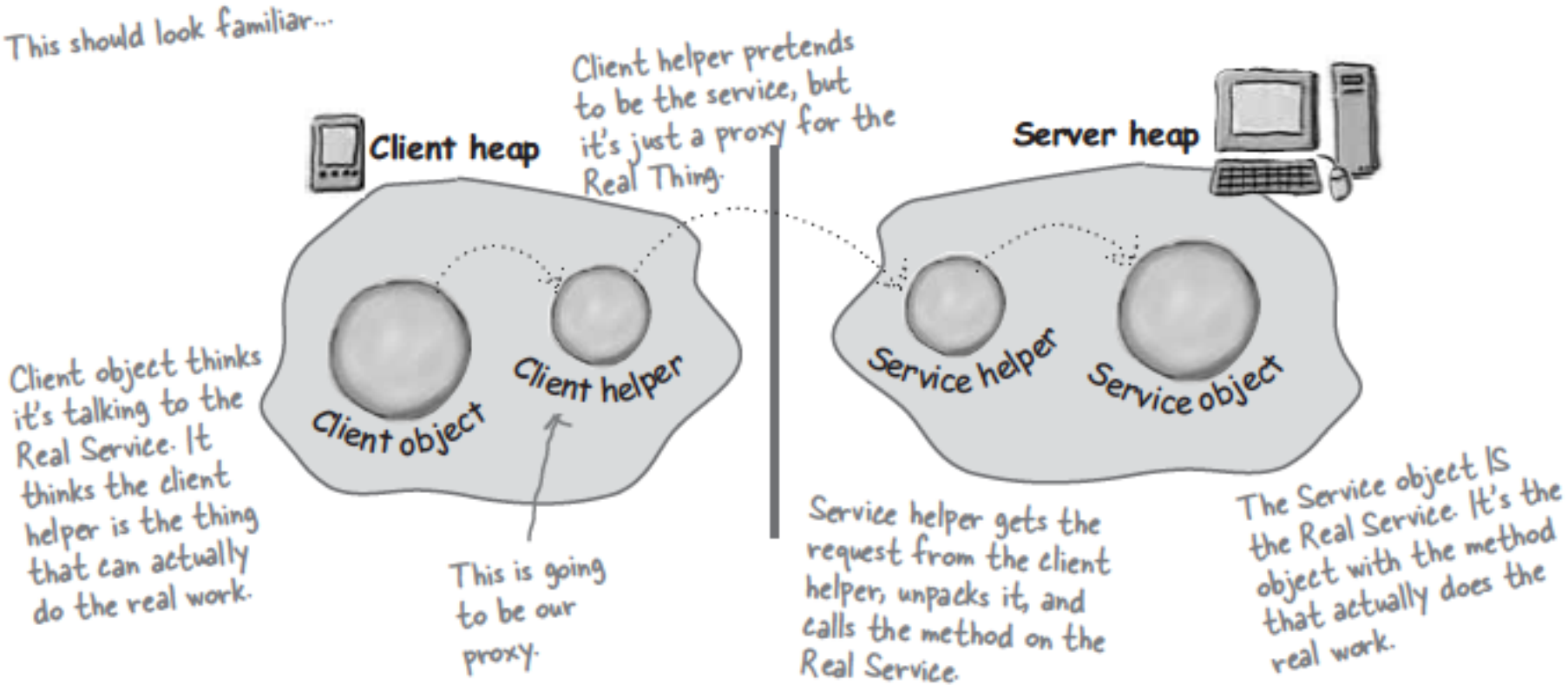


Proxy Types

Proxy Types

- Proxies may be employed in several situations
  - **Remote Proxy:** a proxy can provide a *local* representation of a remote object
  - **Virtual Proxy:** a proxy may act as a stand in for an object which are expensive to create
  - **Protection Proxy:** a proxy provides control access to the original object
  - **Smart Reference:** a proxy performs additional actions when an object is accessed

Remote Proxy



Remote Proxy

- Object may not *actually* exist in the scope of the local process
  - Object exists as data in a database
  - Object exists on some server in a client-server system
- Still need to access the object locally
  - Accessing data directly from a database could become needlessly expensive
    - *SQL query for every access or mutation*
  - Access to the persistent object may not always be available
    - *Internet connection may fail, still want to update an object until an internet connection can be re-established*

Remote Proxy

Subject

```
class Server {
  public:
    virtual int getData(string key);
    virtual void add(string key, int val);
};
```

RealSubject

```
class RemoteServer : public Server {
  private:
    map <string, int> serverData;
  public:
    int getData(string key)
    {
      return serverData[key];
    }
    void add(string key, int val)
    {
      serverData[key] = val;
    }
    int isOpen()
    { … }
};
```

Proxy

```
class ServerProxy : public Server {
  private:
    map <string, int> localData;
    RemoteServer server;
  public:
    int getData(string key) {
      // can I access the server?
      if (server.isOpen())
      {
        localData[key] =
server.getData(key);
      }
      return localData[key];
    }
    void add(string key, int val)
    {
      localData[key] = val;
      if(server.isOpen())
      {
        server.add(key, val);
      }
    }
};
```

Virtual Proxy

- Object may be expensive to create
  - Loading images in a document
  - Preloading *every* level in a game
- May not necessarily need to create the object immediately
  - Only need to actually load images when *drawImage()* is called
  - Only need to load individual levels when a player enters
  - Proxy object that can answer simple questions about the full object and only instantiates the full object when required.
    - *E.g., Image – initially get size but not full image*
    - *E.g., DB proxy*

Virtual Proxy

Subject

```
class Graphic {
   public:
      virtual void draw();
      virtual void getWidth();
      virtual void getHeight();
      virtual void load();
};
```

RealSubject

```
class Image : public Graphic {
  private:
    int width;
    int height;
  public:
    void draw() { … }
    void getWidth() { return width; }
    void getHeight() { return height; }
    void load() { // very expensive }
};
```

Proxy

```
class ImageProxy : public Graphic {
  private:
    string filename;
    Image* image = null;
    int width;
    int height;
  public:
    void draw() {
      if (image == null) {
        image = loadImg(filename);
      }
      image->draw();
    }
    void getWidth() {
      if (image == null) {
        return width;
      } else {
        return image->getWidth();
      }
    }
    …
};
```

Virtual Proxy

- Example Client Usage

```
Graphic image1 = new ImageProxy("big_image.png");
Graphic image2 = new ImageProxy("real_big_image.tiff");

// layout the text first around the images
int image1_width = image1.getWidth();  // proxy allows for
…                                      // rapid text layout

// When the user scrolls down to where the image is, call draw
void draw() {
  image1.draw()           // Now, image1 is loaded
}
```

Protection Proxy

- Some objects may only allow access in a given context
  - Database table may be read by anyone, but only written to by administrator
  - File can be decrypted *only* after proper validation

Smart Reference

- Smart references replace bare pointers (references) to other objects
  - Allows for additional actions to be performed
    - *How many references to the real object exist?*
    - *Load a persistent object into memory when first referenced*
    - *Check for lock on the real object before accessing*
  - Hides many of the implementation details
    - *Garbage collection – is reference count 0?*
    - *Persistence – has this object been loaded from the database?*
    - *Locking – is another object accessing this object?*

Design Considerations

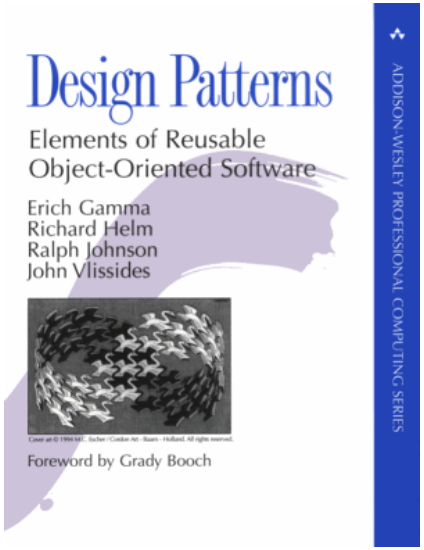
Consequences

- Introduces a level of indirection when accessing an object
  - Hides details like where an object *actually* resides
  - Create objects on demand
  - Allow additional housekeeping to be performed
  - Copy-on-write: Proxy allows complex object copying to be performed *only* when an object is written to

Implementation Considerations

- Exploiting language features
  - C++ allows operator overloading
    - *Overloading dereference (*) and member access (->) operators allows the proxy to behave as if it were a pointer*
    - *Don't need to implement the method calls in the Proxy*
  - Using proxy as an exception handler
  - Proxy does not need to be aware of the type of the real subject
    - *Assuming some common interface…*

Further Reading



- **Design Patterns**
  pp. 207 - 217

Chapter 11