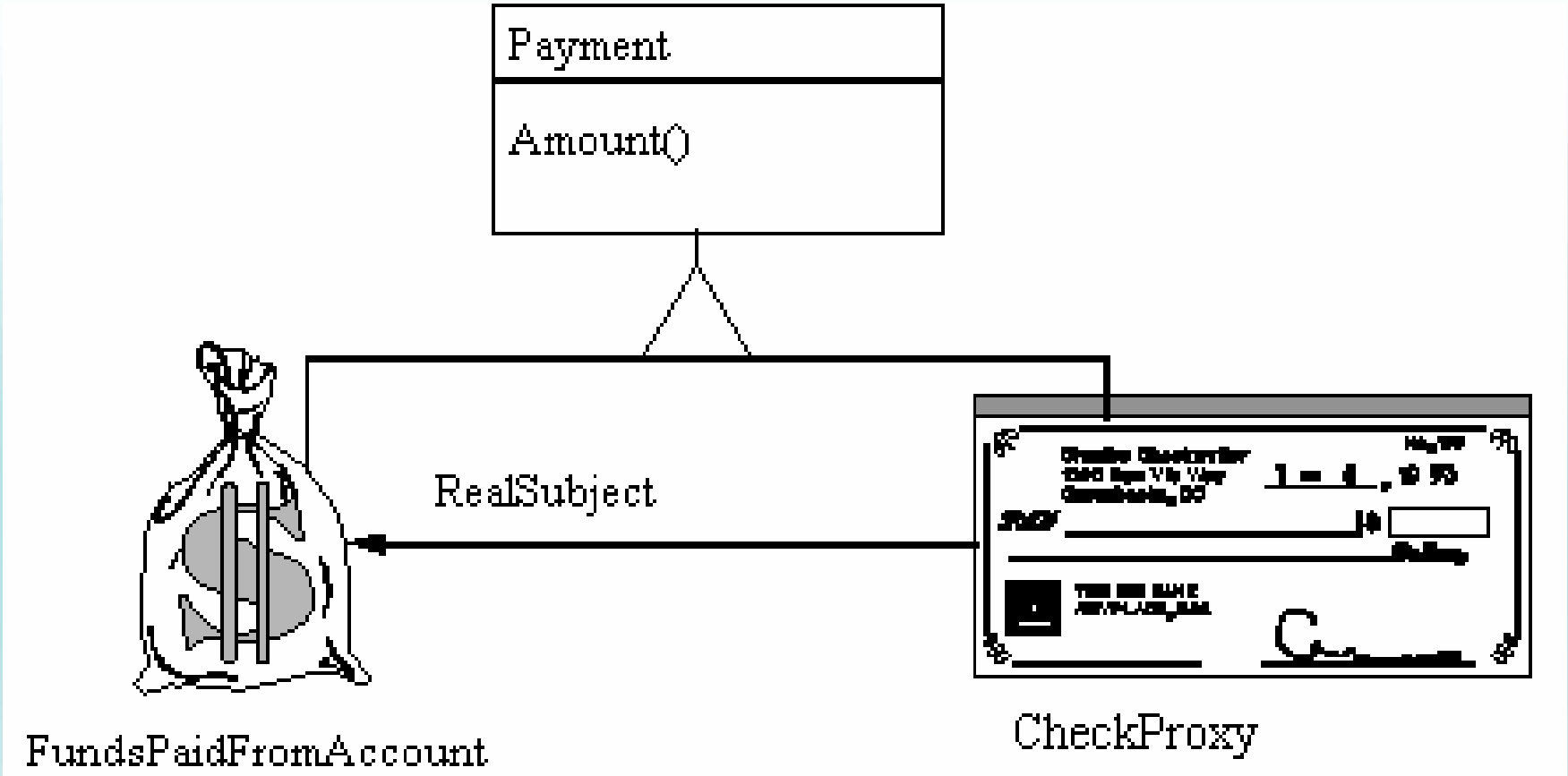
Proxy Design Pattern

Anjan Kumar Bollam
Srikanth Gorlla

Example – Financial proxy



Questions on Proxy design pattern

- What is the intent of Proxy Design Pattern?
- What is the motivation of Proxy Design Pattern?
- If a Proxy is used to instantiate an object only when it is absolutely needed, does the Proxy simplify code?
- How to handle multiple copies of the Complex Object using Proxy?
- What are the advantages of using Proxy?
- What are the other related patterns for Proxy?
- When do we use Proxy Design Pattern?

Intent

- Provide a surrogate or placeholder for another object to control access to it.
  - Provides extra level of indirection
  - Protects real component from undue complexity

- Also Known as :
  - SURROGATE

Motivation

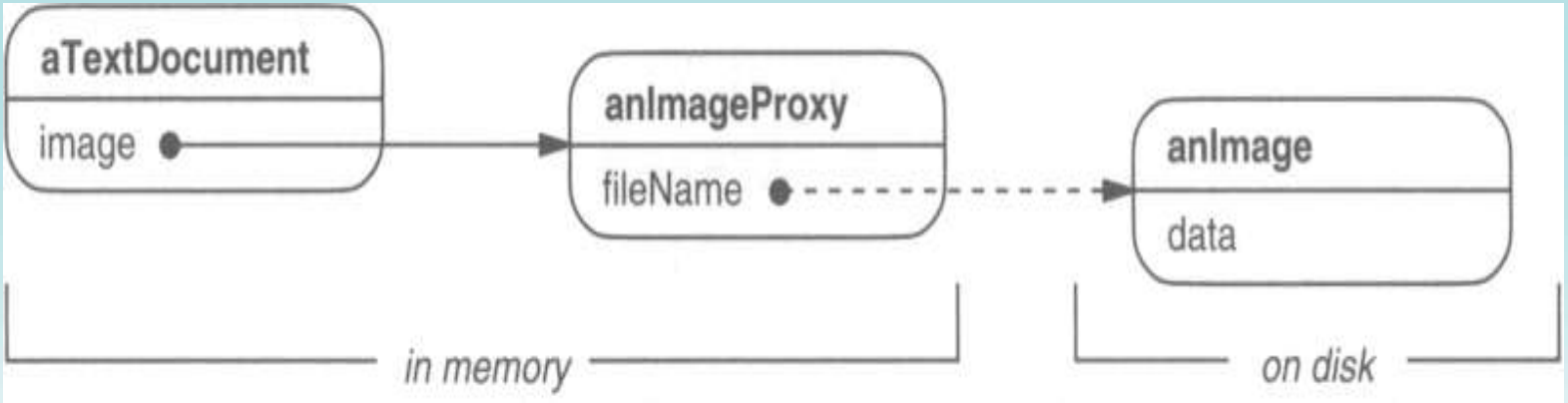
- Creating objects even though they are not required may be expensive in terms of time or computer resource.
  - Proxy allows us to postpone this creation until we need the actual object.
  - Proxy also allows us to vary how and where an object is accessed.
  - Proxy creates each expensive object only when required.

- When to postpone object creation?
  - During initializations or creations.
  - When you need to represent an object that is complex or time consuming to create with a simpler one.

Motivation - Example

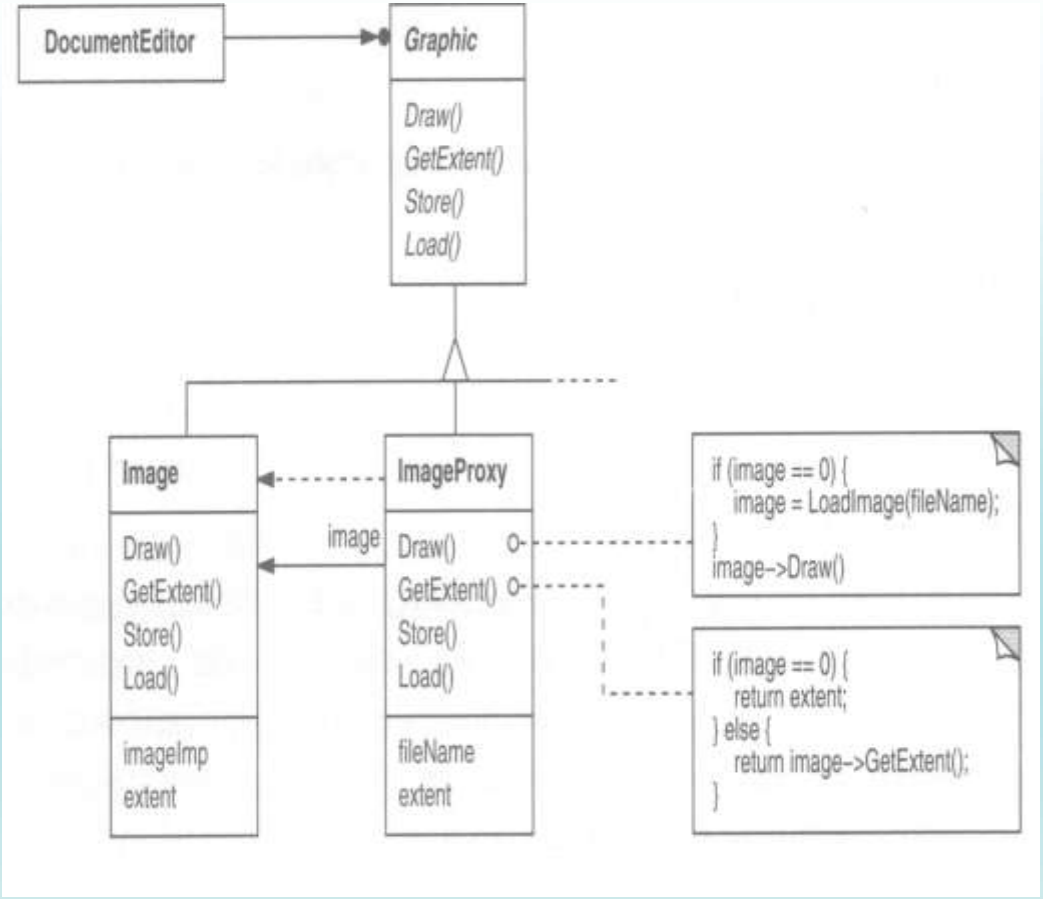
- Document Editor
  - Need to Embed Graphical images
  - Loading of Graphical images takes time.
  - This may slow down the opening of the document.
  - But what to put instead of the actual object while initializing?

- We can use a PROXY



Motivation - Example

- Image Proxy
  - Acts as stand in for real image
  - Takes care of instantiating the real object when required by invoking the draw() function.
  - Forwards the subsequent requests directly to the image
  - Creates the real image only when the document editor asks it to



Applicability

- Proxy is applicable whenever there is a need for more versatile or sophisticated reference to an object than simple pointer.

Key Words: Versatile or sophisticated

- Behavior of an object depends on its state, and it must change its behavior depending on that state.

Applicable Situations

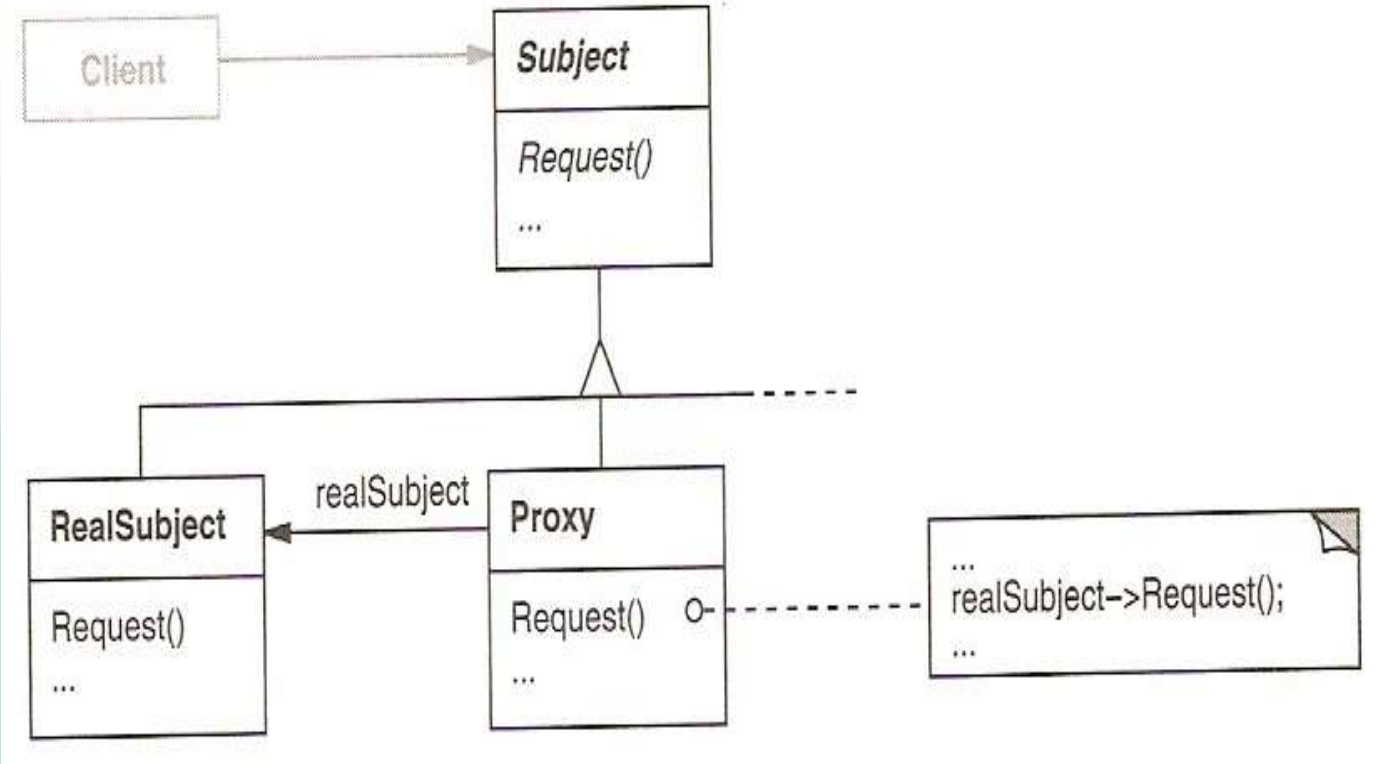
- A **remote proxy** provides a local representative for an object in a different address space
- A **virtual proxy** creates expensive objects on demand.
- A **protection proxy** controls access to the original object.
- A **smart reference** is a replacement for a bare pointer that performs additional actions when an object is accessed.

Smart References

Typical uses include

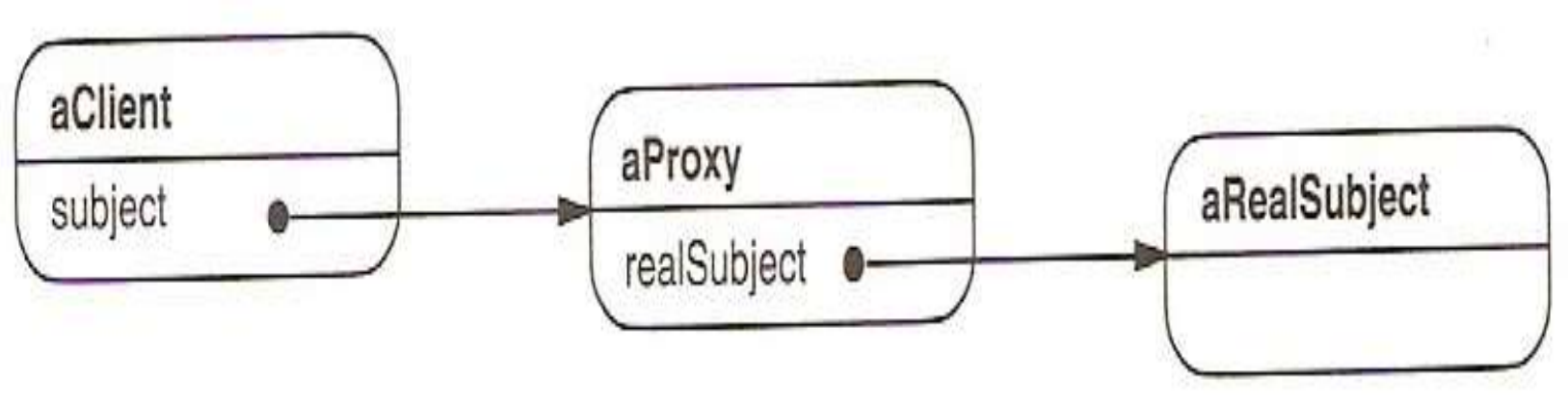
- counting the number of references to the real object so that it can be freed automatically when there are no more references (also called **smart pointers)**
- loading a persistent object into memory when it's first referenced.
- checking that the real object is locked before it's accessed to ensure that no other object can change it.

Structure – Class Diagram



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

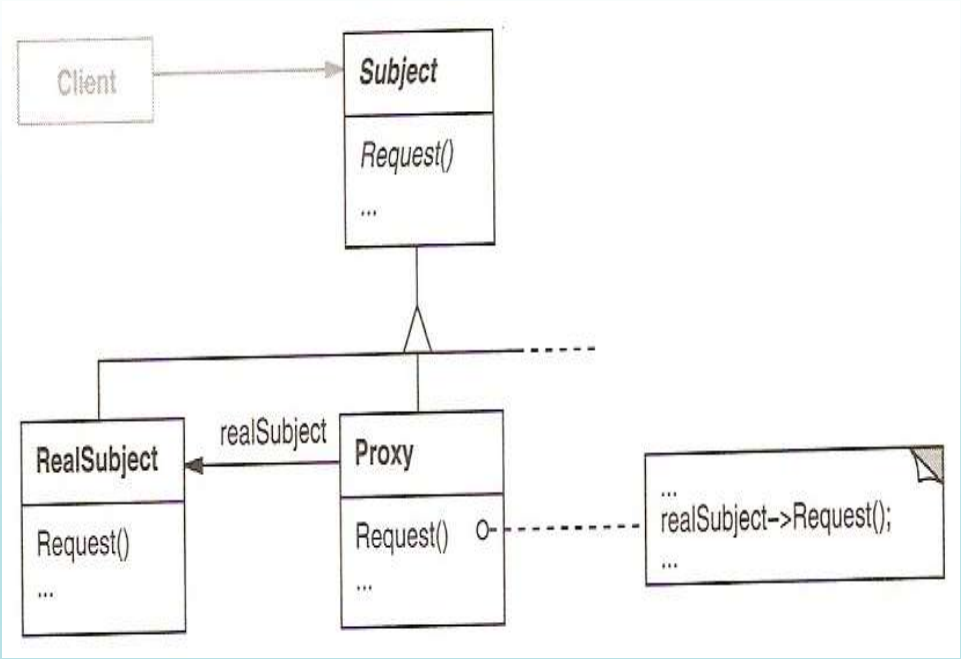
Structure – Object Diagram



Participants

The three participants are:

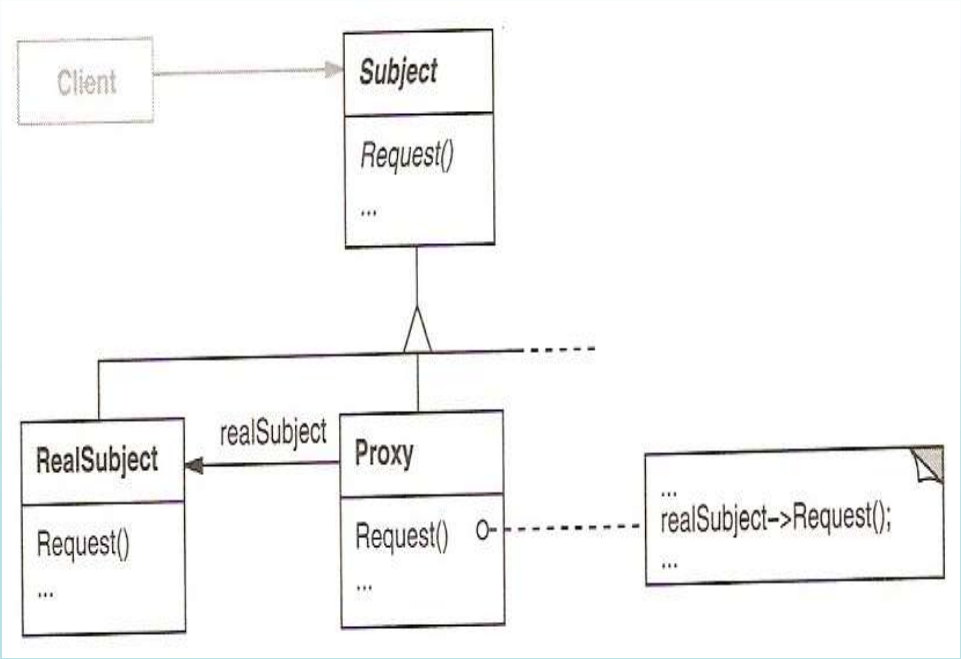
- **Proxy**
- **Subject**
- **Real Subject**



Participants

Proxy

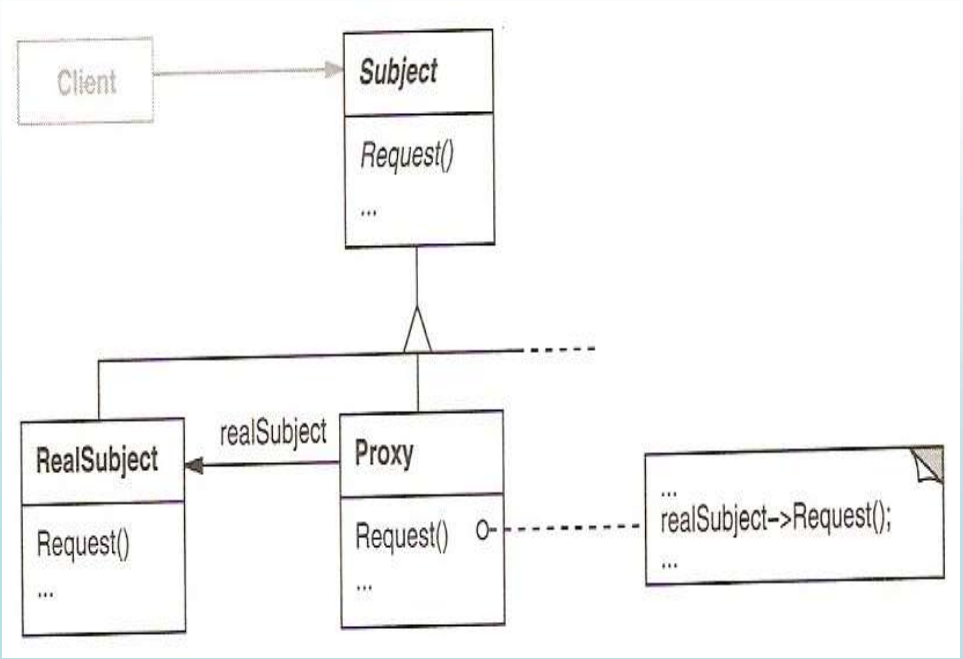
- Maintains a reference that lets the proxy access the real subject.
- Provides an interface identical to subject's so that a proxy can by substituted for the real subject.
- Controls access to the real subject and may be responsible for creating and deleting it.



Participants

Subject

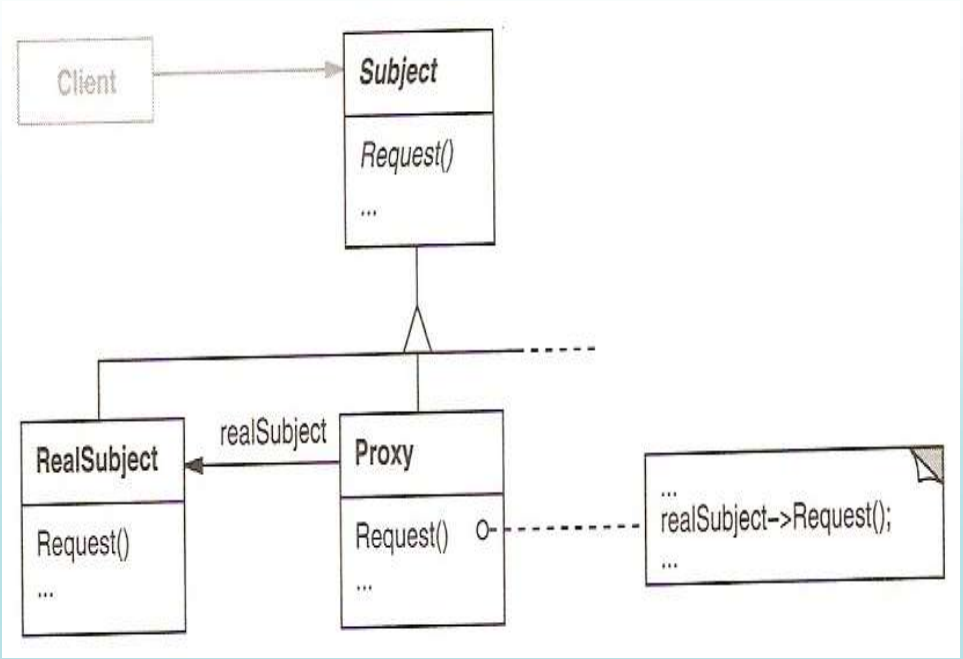
- Defines the common interface for Real Subject and Proxy so that a Proxy can be used anywhere a Real Subject is expected.



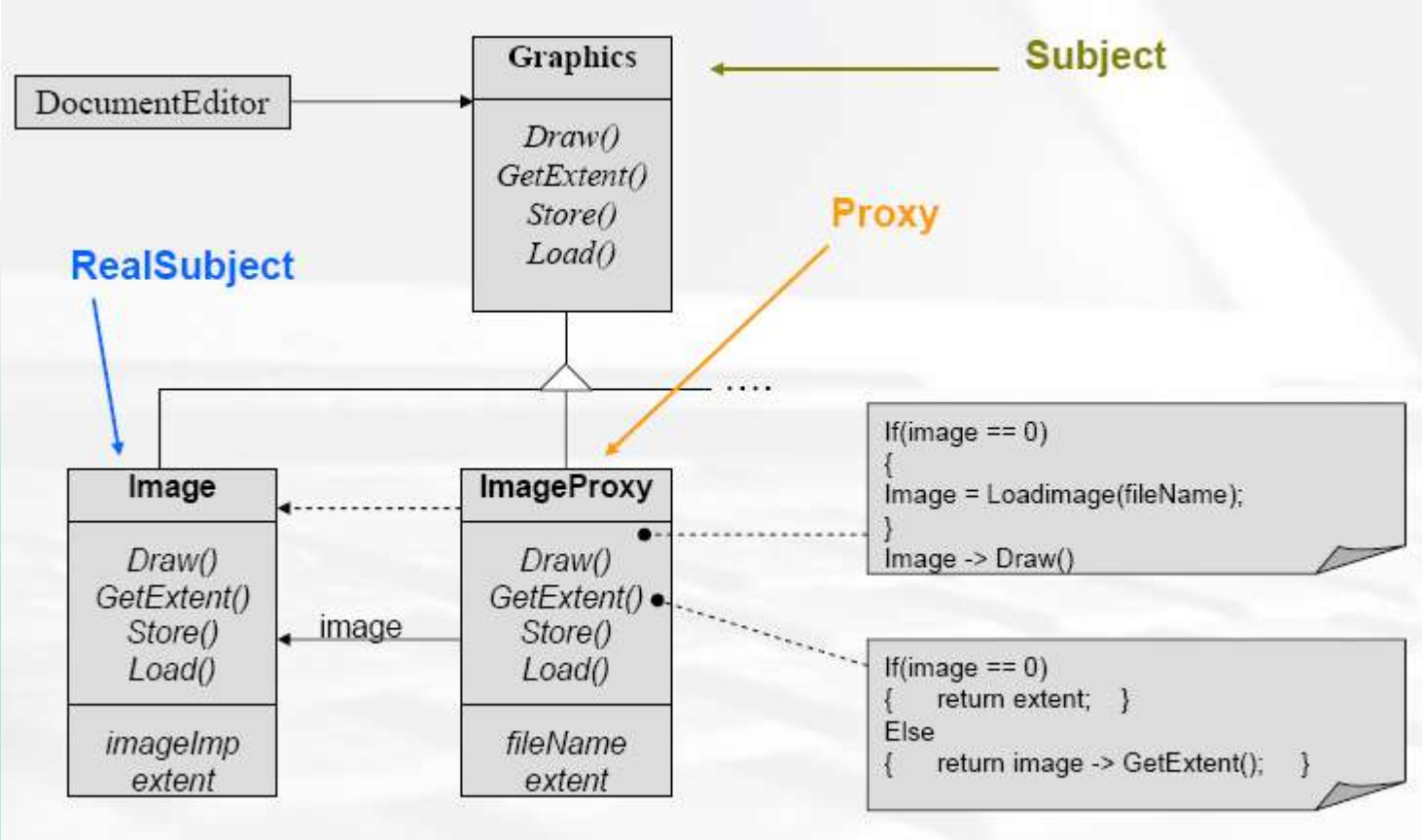
Participants

Real Subject

- Defines the real object that the proxy represents.



Participants - Example



Collaborations

- Proxy forwards requests to Real Subject when appropriate, depending on the kind of proxy.

Consequences

- The Proxy pattern introduces a level of indirection when accessing an object. This additional indirection has many uses.
  - A remote proxy can hide the fact that an object resides in a different address space.
  - A virtual proxy can perform optimizations such as creating an object on demand.
  - Both protection proxies and smart references allow additional housekeeping tasks when an object is accessed.

Consequences contd.

- Another optimization that is hidden from the client is Copy-on-Write

  - Copying a large and complicated object can be an expensive operation. If the copy is never modified, then there's no need to incur this cost. By using a proxy to postpone the copying process, we ensure that we pay the price of copying the object only if it's modified.

Implementation

- Overloading the member access operator in C++.
  - This lets us to perform additional work whenever an object is de-referenced.
  - Helps in implementing some kinds of proxy just like a pointer.
- Using “doesNotUnderstand” in Smalltalk.
  - Proxy class (with no super class) can redefine doesNotUnderstand so that the message forwarded to the subject.
- Proxy doesn’t always have to know the type of real subject.

Overloading the member access pointer

```
class Image;
    // external function
   extern Image* LoadAnImageFile(const char*);

   class ImagePtr {
   public:
        ImagePtr(const char* imageFile);
        virtual ~ImagePtr();
        virtual Image* operator->();
        virtual Image& operator*();
   private:
        Image* LoadImage();
   private:
        Image* _image;
        const char* _imageFile;
   };
```

Overloading the member access pointer

- The impementation of the methods of class ImagePtr (virtual proxy)

```
ImagePtr::ImagePtr (const char* theImageFile) {
        _imageFile = theImageFile;
        _image = 0;
}

Image* ImagePtr::LoadImage ()   {
        if (_image == 0)
                _image = LoadAnImageFile(_imageFile);
        return _image;
}
```

Overloading the member access pointer

- The overloaded -> and * methods call the LoadImage() to return the _image object.

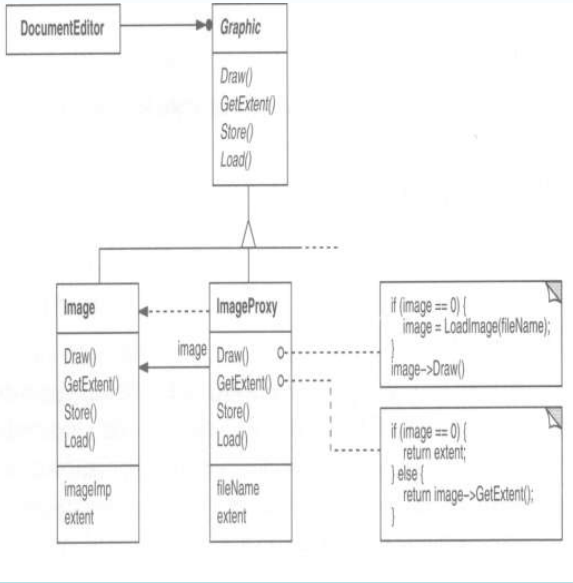
```
Image* ImagePtr::operator-> ()      {
     return LoadImage();
}

Image& ImagePtr::operator* ()       {
      return *LoadImage();
}
```

Overloading the member access pointer

- This approach allows us to call Image class operations through ImagePtr objects without going to the trouble of making the Image class operations part of the ImagePtr interface:

```
ImagePtr image = ImagePtr("anImageFileName");
image->Draw(Point(50, 100));
//(image.operator->())->Draw(Point(50,100))
```



- This isn't a good solution to every kind of proxy, where the proxy need to know precisely which operation is called. For example, in the virtual proxy example presented in Motivation, image should be referenced when the Draw operation is called.

Using DoesNotUnderstand in Smalltalk

- Smalltalk provides a hook that provides automatic forwarding of messages from client to receiver.

- It calls DoesNotUnderstand : aMessage when the client sends a message to a receiver that has no corresponding methods.

  How do we redirect the message to the subject in this case???

  - We can use a Proxy class (no superclass) that can redefine DoesNotUnderstand so that the message is forwarded to its subject.

Proxy doesn't always have to know the type of real subject

- If a proxy class can deal with the its subject solely through an abstract interface, then there is no need to make Proxy class for each RealSubject class.
  - ➢ Proxy can deal with all RealSubjects uniformly.
- If Proxy class is going to instantiate the RealSubjects (like inVirtual Proxy), then proxy should know about the concrete class.

  How to refer to subject before it is instantiated???
  - ➢ We need address space-independent object identifiers

Sample Code

- Remote Proxy
  - Client – Server Example
- Virtual Proxy
  - ImageLoad Example
- Protection Proxy
  - Database Login Example

Known Uses

- NEXTSTEP [Add94] uses proxies (instances of class NXProxy) as local representatives for objects that may be distributed.

- World Wide Web Proxy generally runs on a firewall machine and allows people inside the firewall concurrent access to the outside world.

- In Microsoft's OLE DB, proxies hide whether a specific server is local or remote from a client.

Related patterns

- Adapter

  Adapter Pattern provides a different interface to its subject. Proxy provides the same interface.

- Decorator

  Proxy provides the same interface. Decorator Pattern provides an enhanced interface

References

- Design Patterns, Elements of Reusable Object-Oriented Software, Erich Gamma, et. al., Addison-Wesley, 1994
- Head First Design Patterns by Eric Freeman & Elisabeth Freeman
- http://www.dofactory.com/Patterns/PatternProxy.aspx
- http://home.earthlink.net/~huston2/dp/proxy.html
- http://en.wikipedia.org/wiki/Proxy_pattern
- http://dotnetperls.com/protection-proxy

Questions on Proxy design pattern

- What is the intent of Proxy Design Pattern?
- What is the motivation of Proxy Design Pattern?
- If a Proxy is used to instantiate an object only when it is absolutely needed, does the Proxy simplify code?
- How to handle multiple copies of the Complex Object using Proxy?
- What are the advantages of using Proxy?
- What are the other related patterns for Proxy?
- When do we use Proxy Design Pattern?

Questions?