

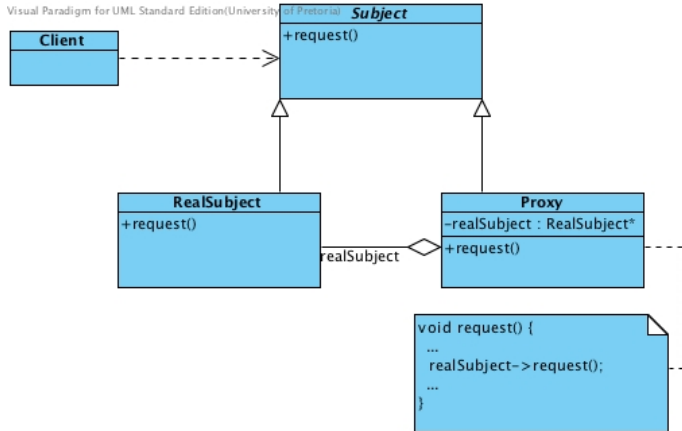
Proxy Design Pattern

Linda Marshall and Vreda Pieterse

Department of Computer Science
University of Pretoria

27 & 28 October 2014

Name and Classification: Proxy
(Object Structural) **Intent:** “Provide a surrogate or placeholder for another object to control access to it.” GoF(207)



Subject

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

Real Subject

- Defines the real object that is represented by the proxy.

Proxy

- serve as substitute for the real subject.
- maintains a reference to the real subject.
- controls access to the real subject
- may be responsible for creating and deleting the real subject.
- more responsibilities specific to its kind

There are at least 4 kinds of proxy

- Remote proxy
- Virtual proxy
- Protection proxy
- Smart reference

- provides a local representative for an object in a different address space
- can hide the fact that an object resides in a different address space

Responsible for

- encoding a request and its arguments
- sending the encoded request to the real subject in a different address space

- provides a representative for an expensive object
- can postpone access to an object until it is really needed

Responsible for

- creating expensive objects on demand
- caching information about the real subject so that access to it can be avoided if possible

- controls access to the real object
- useful when objects/users have different access rights

- check that the caller has the access permissions required to perform a request
- may perform additional housekeeping tasks when an object is accessed

A replacement for a bare pointer that performs additional actions when an object is accessed

Typical uses

- **Memory management** - count the number of references to the real object
- **Load on demand** - load a persistent object into memory on first reference
- **Safe updating** - lock the real object before it's accessed

Smart Pointers: They're Smart, but They're not Pointers

author: Daniel R. Edelson

Proceedings: Usenix C++ Conference

year: 1992

An abstract data type that simulates a pointer while providing additional features intended to reduce bugs caused by the misuse of pointers while retaining efficiency.

Keep track of the objects they point to for the purpose of memory management

- bounds checking
- automatic garbage collection

STL C++ (03) auto pointers std :: auto_ptr included in <memory>

- it doesn't do all the useful things that smart pointers might
- can be a dangerous snake that turns and bites you

(Herb Sutter, 2002)

Boost smart pointers

<http://www.boost.org/doc/libs/>

Smart Ptr Smart pointer class templates.

Authors: Greg Colvin, Beman Dawes,
Peter Dimov, and Darin Adler

First Release 1.23.0 (July 2001)

STL C++ (11) smart pointers

- **unique_ptr** - unique ownership, move constructible and move assignable
- **shared_ptr** - counted pointer, object is deleted when the use count goes to zero
- **weak_ptr** - robust unowned pointer.

Adapter

Adapter and Proxy both provide an interface to access another object. However, the reasons for doing this are different.

Adapter

Provides a different interface to the object it adapts

Proxy

Provides the same (or diminished) interface as its subject

Decorator

Decorator and Proxy both describe how to provide a level of indirection to an object and forward requests to it. However, they have a different purpose

Decorator

- 1) dynamic attach or detach
- 2) recursive composition

Proxy

Provide a stand-in when it is inconvenient or undesirable to access the subject directly

Decorator

The component provides only part of the functionality, and one or more decorators furnish the rest.

Proxy

Subject defines the key functionality, and the proxy provides (or refuses) access to it

Prototype

Prototype and Proxy both offer a solution to a problem related to an object that is expensive to create. However, the solutions are different.

Prototype

Keep a copy of the object handy and clone it on demand

Proxy

Create a stub for the object and create the object on demand

Flyweight

Flyweight and Proxy both apply a smart reference to manage access to an object. However, the purpose of the reference is quite different.

Flyweight

Control multiple pointers
to a shared instance

shared_ptr

Proxy

Control single access to
a specific real object

unique_ptr

```
class ProxyOnlineBankAccount : public OnlineBankAccount {
public :
    ProxyOnlineBankAccount();
    virtual ~ProxyOnlineBankAccount();
    virtual bool login( string name, string pass );
    virtual bool pay( string to , string from , int amount );
private :
    RealOnlineBankAccount* realSubject ;
    long maxTime;
    long startTime ;
    bool loggedIn ;
};
```

An example of a protection proxy
Subject - OnlineBankAccount
Proxy - ProxyOnlineBankAccount
RealSubject - RealOnlineBankAccount

```
ProxyOnlineBankAccount ::ProxyOnlineBankAccount() {  
    // initialization of maxTime = 10000, loggedIn = false , etc  
}  
ProxyOnlineBankAccount ::~ProxyOnlineBankAccount() {  
    //deletions  
}
```



```
bool ProxyOnlineBankAccount::login( string name, string pass ) {  
    //Already logged in  
    if (loggedIn) {  
        delete realSubject ;  
    }  
    //get start time of session  
    startTime = clock () ; bool valid = false ;  
    //validates password with database.  
    //Create real bank account  
    if (valid) {  
        realSubject = new RealOnlineBankAccount () ;  
        loggedIn = realSubject->login ( name, pass ) ;  
    } else { cout << Invalid login details << endl; }  
    return loggedIn ;  
}
```

```
bool ProxyOnlineBankAccount ::pay( string to , string from , int amount ) {  
    long duration = clock () - startTime ;  
    if ( duration < maxTime && loggedIn ) {  
        return realSubject->pay(to, from, amount ) ;  
    } else {  
        cout << Your session has expired << endl ; loggedIn = false ;  
        delete realSubject ;  
        return false ;  
    } }  
}
```