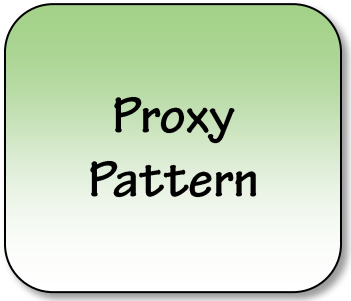
Proxy Pattern

Design Patterns

Intent



- Provide a surrogate or placeholder for another object to control access to it
- Also known as: *Surrogate*

Motivating Example

Proxy Pattern

- You need a placeholder for an actual object that is expensive to create

Example – A placeholder image to show and allow for screen to lay out while actual image is being rendered

- You need to provide a local object that stands in place of a remote object and acts as an intermediary

Example – You need to interact with a remote service over the network, but want to keep your code decoupled from networking details

Applicability

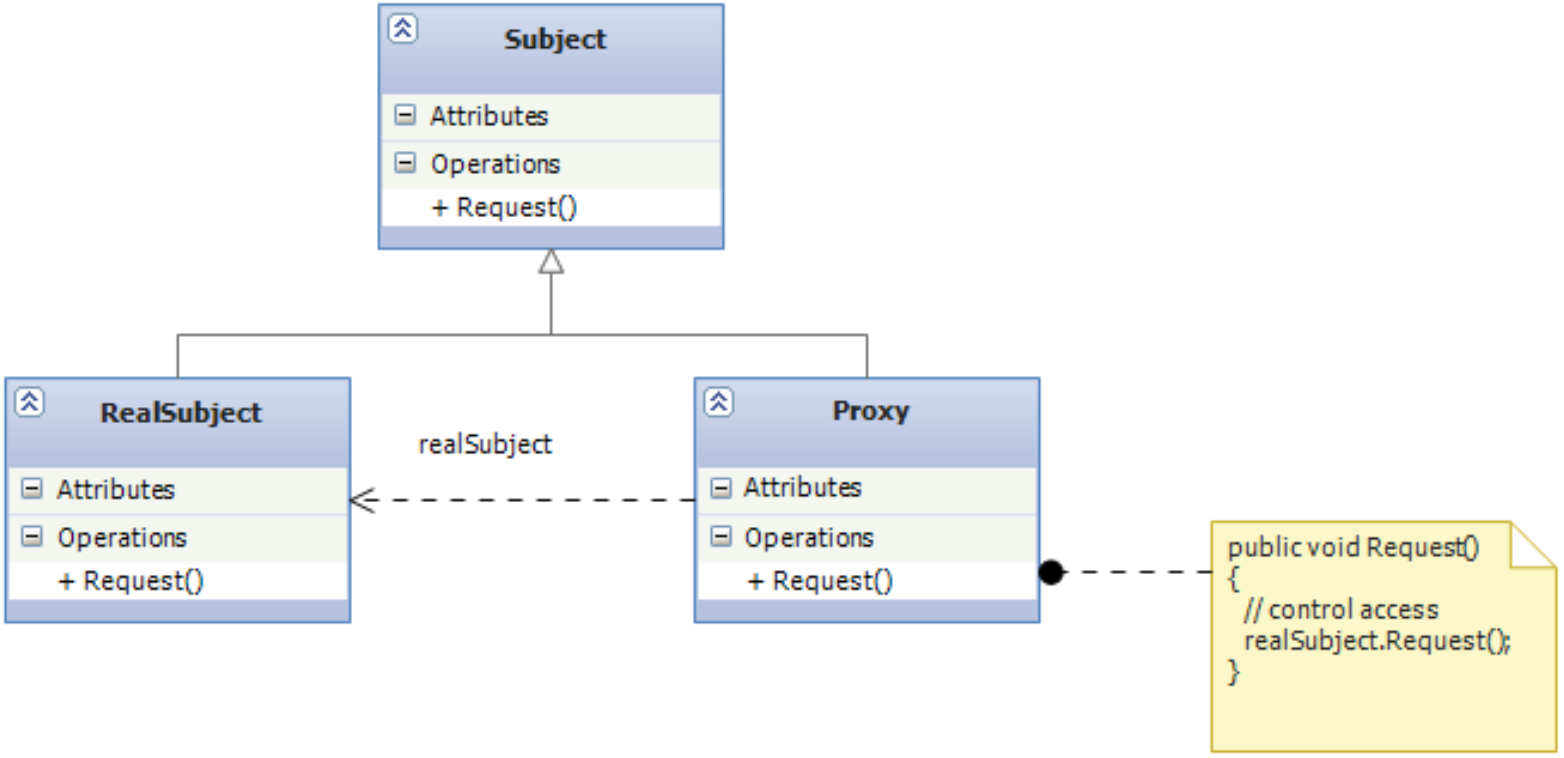
Proxy Pattern

The Proxy pattern is applicable in several common scenarios:

- A *remote proxy* acts as a local representative of a remote object, and abstracts away the details of communicating with the remote object
- A *virtual proxy* creates expensive objects on demand. A common example is an Image Proxy that shows a placeholder while the image is being loaded or rendered.
- A *protection proxy* can be used to control access to an object, based on authorization rules.

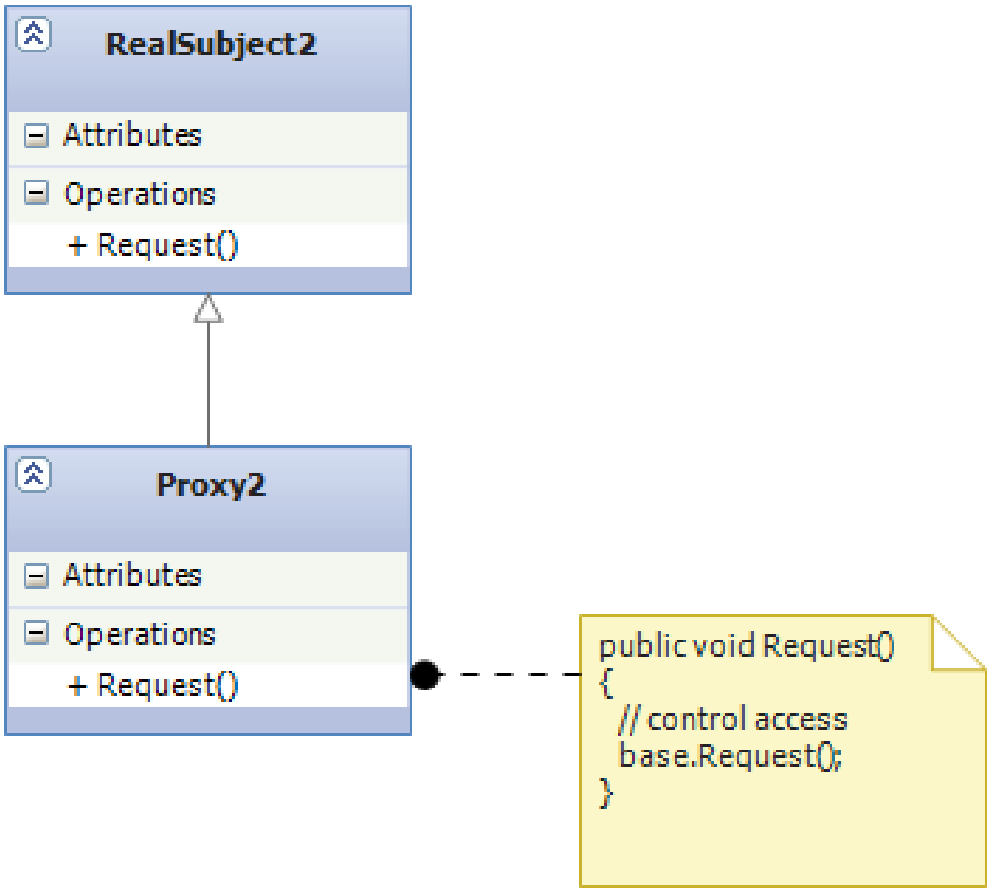
Structure

Proxy Pattern

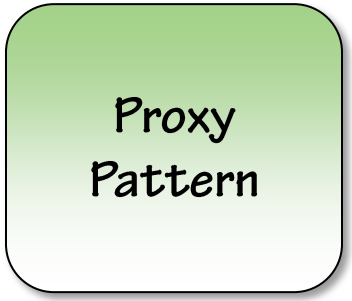


Structure (alternate)

Proxy Pattern



How It Gets Used



- **Clients work with Proxy as if it were the actual object**
- **Proxy controls access to actual object, delegating calls to it as needed**
- **Used to:**
  - Improve performance and load time of an application
  - Simplify interactions with remote objects
  - Defer expensive calls until needed – implement *lazy loading* of objects from persistence

Lazy Loading

Proxy Pattern

Objects stored in database may span many tables and rows

Example:

An Order object has a collection of OrderDetail items and a reference to a Customer record

Each OrderDetail item refers to a Product

Each Customer record refers to a collection of Orders and other objects

Creating an Order object such that all of its fields are fully instantiated may result in loading thousands of records from the database. If we only need to display the Order Number and Total Price, for instance, there's no value in populating the Customer or OrderDetail records.

Lazy loading refers to the practice of fetching object state from persistence only as it is requested by the object's client(s).

Consequences

Proxy Pattern

- **Client code does not need to be changed to work with a Proxy**
  - However, Proxy must be kept synchronized with Real Object
- **Proxy may be used to optimize performance in an existing system without touching existing class behavior**
  - Follows Open/Closed Principle
- **Client code may make incorrect assumptions about behavior of real object**
  - Interface may use many small calls rather than few, large calls
  - Chatty versus Chunky interface
  - Client access to lazy-loaded object may result in more database calls than necessary

Implementation Example

Proxy Pattern

Real Object

```
public Order(int id)
{
    _id = id;
    GetEntity();
    this.OrderDate = this.OrderEntity.OrderDate;
    this.Customer = GetCustomer();
    this.Items = GetItems();
}
```

Replaces

Virtual Proxy Object

```
public class OrderProxy : Order
{
    public OrderProxy(int id)
    {
        _id = id;
    }
```

```
public override Customer Customer
{
    get
    {
        return GetCustomer();
    }
}
```

```
protected virtual Customer GetCustomer()
{
    return new Customer(this.OrderEntity.CustomerId);
}
```

```
protected override Customer GetCustomer()
{
    if (!_customerLoaded)
    {
        GetEntity();
        _customer = base.GetCustomer();
        _customerLoaded = true;
    }
    return _customer;
}
```

```
public Customer(int id)
{
    _id = id;
    CustomerEntity customerEntity = new CustomerRepository().GetById(id);
    this.Name = customerEntity.Name;
}
```

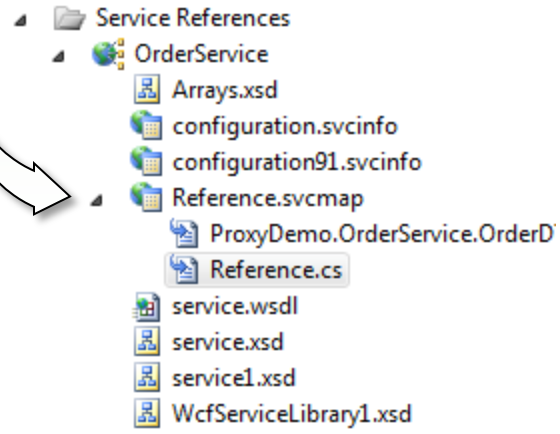
Implementation Example

Proxy Pattern

Client

```
public class RemoteOrderRepository : Repository<OrderEntity>
{
    public override OrderEntity GetById(int id)
    {
        Console.WriteLine("--> Using WCF to Fetch Order " + id);
        var dto =
            new OrderService.OrderServiceClient().GetDataUsingDataContract(id);
        return new OrderEntity()
                    {
                        Id = dto.Id,
                        CustomerId = dto.CustomerId,
                        OrderDetails = dto.OrderItems,
                        OrderDate = dto.OrderDate
                    };
    }
}
```

Remote Proxy



Actual Remote Object

```
public class OrderService : IOrderService
{
    public OrderDTO GetDataUsingDataContract(int orderId)
    {
        return new OrderDTO
                    {
                        Id = orderId,
                        CustomerId = 123,
                        OrderDate = DateTime.Now,
                        OrderItems = new[] {1, 2, 3, 4, 5}
                    };
    }
}
```

Implementation Example

Proxy Pattern

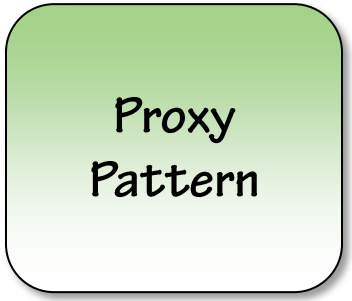
Repository

```
public class OrderRepository : Repository<OrderEntity>
{
    public override OrderEntity GetById(int id)
    {
        Console.WriteLine("--> Fetching Order " + id);
        return new OrderEntity()
                    {
                        Id = id,
                        CustomerId = 1,
                        OrderDetails = new int[] {1, 2, 3},
                        OrderDate = DateTime.Now
                    };
    }
}
```

Cached Repository

```
public class CachedOrderRepository : OrderRepository
{
    public override OrderEntity GetById(int id)
    {
        string cacheKey = "OrderEntity-" + id;
        var entity = MemoryCache.Default[cacheKey] as OrderEntity;
        if(entity == null)
        {
            entity = base.GetById(id);
            var cacheItem = new CacheItem(cacheKey, entity);
            var policy = new CacheItemPolicy();
            MemoryCache.Default.Add(cacheItem, policy);
        }
        return entity;
    }
}
```

Related Patterns

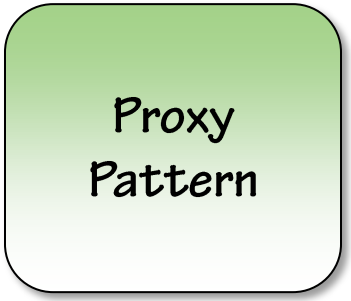


- **Adapter**
  - The Adapter pattern changes an object's interface; the Proxy pattern retains the same interface but controls access to the underlying behavior

- **Decorator**
  - The Decorator has a similar implementation to the Proxy, but its intent is different. A Decorator adds responsibility to an object, while the Proxy controls access to an object.

You can learn more about these patterns in the Pattern Library at PluralSight On Demand.

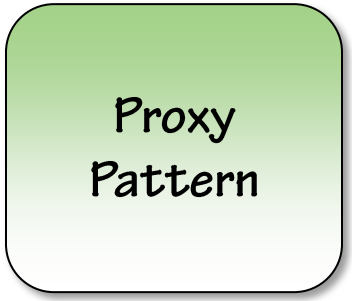
References



- **Books**
  - Design Patterns, http://amzn.to/95q9ux
  - Design Patterns in C#, http://amzn.to/bqJgdU
  - Head First Design Patterns, http://amzn.to/aA4RS6

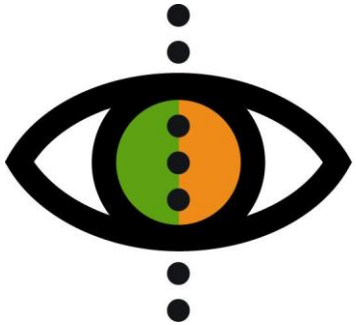
- **Online**
  - http://en.wikipedia.org/wiki/Proxy_pattern

Summary



- **The Proxy pattern is used to control access to an object by channeling access through a proxy object**
- **A virtual proxy can be used to defer expensive operations and improve application performance**
  - Virtual proxies may be generated automatically as part of an O/RM tool
- **A remote proxy can be used to encapsulate access to an object that lives in a remote address space**
  - Remote proxies are commonly generated automatically for WCF and other service calls

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust

Blog: SteveSmithBlog.com

Twitter: @ardalis