Proxy Design Pattern

Purpose and Motivation

Proxy means “a figure that can be used to represent something else”. The proxy design pattern is used to create a wrapper to cover the main object’s complexity from the client.
The main purpose of the proxy design pattern is to abstract and encapsulate code segments. It solves many different problems, but all of them involve some object taking the place of and/or acting like another object. They are widely used for security purposes, to hide networking logic and to load large objects on demand.

Intended Use Cases

There are 4 main intended use cases for the proxy design pattern. Because of this, there are 4 main realizations of the proxy pattern:

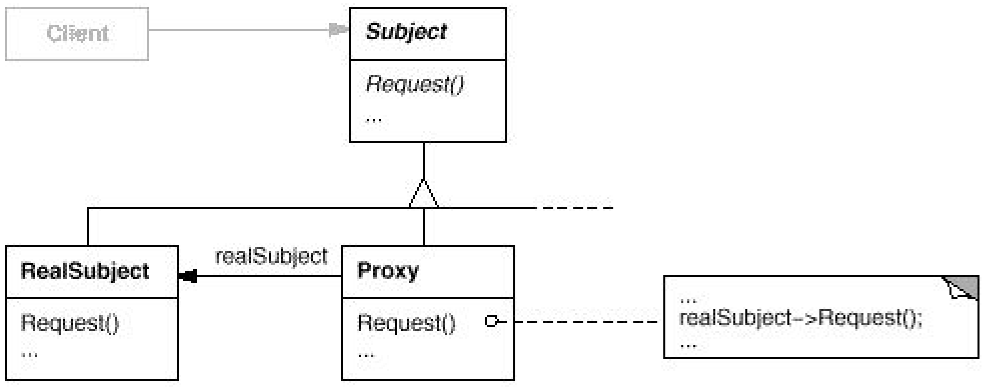
- Protection proxy: Manipulates the arguments of the real object’s functions for security reasons, such as filtering, or masking inputs and outputs to and from the real object. It uses access rights to facilitate access control of resources. It effectively abstracts the real object away from the users.
- Remote proxy: There are many frameworks and libraries that allow remote code execution 1 using objects. These objects are created on the client-side but are executed on the server-side where they are able to access information unavailable to the clients. A widely used example is object-relational mappers, these will map database rows to objects for blending the two technologies together.
- Virtual proxy: Provides placeholders and default values while invoking the real object to perform the actual task. Upon completion of the real object’s operation, the virtual proxy will send the real data to the clients who received the mocked values. The most common use-case is to provide the user information on a need-to-know basis. A good example of this is file explorer not returning detailed file information until file properties are pressed.
- Smart proxy: is used to add additional actions such as assertions when accessing the real object. A use case is when accessing a resource, the proxy can check if the resource lock is available to hold. The smart proxy can be used to implement the smart pointer, it can keep track of the number of references to the real object, and free the object automatically once the number of references falls to zero.

Vocabulary

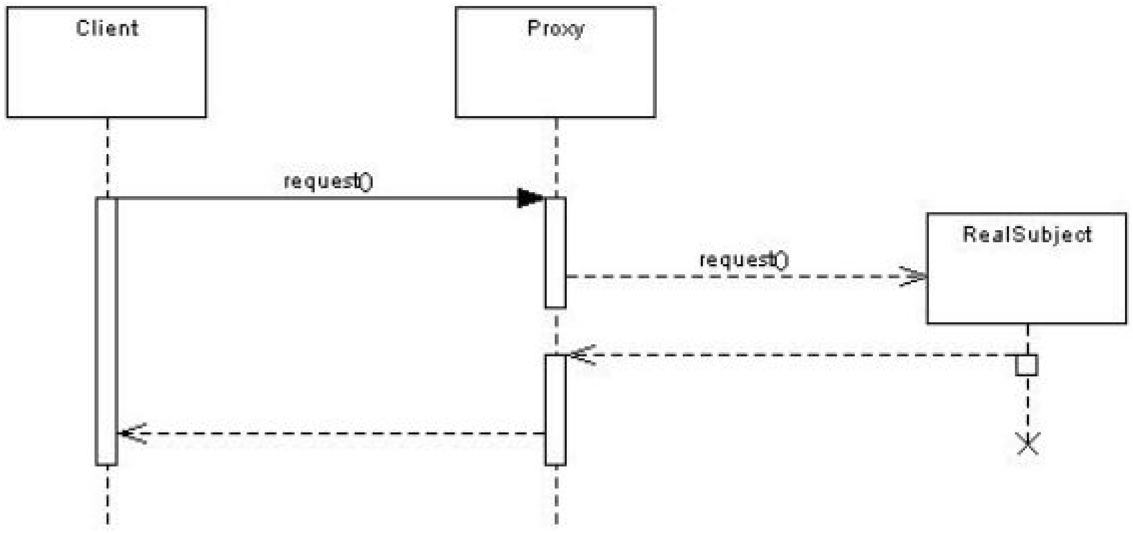
- **Subject Interface**: Relays requests made to the proxy. The client does not know whether they are calling a proxy or the real subject as the subject interface (a form of API layer) will handle the request.
- **Real Subject**: The real subject is where the core logic of the design pattern resides. The proxy can abstract a variety of fundamental security checks or cache commonly used requests, however, the subject must be in charge of most of the real functionality.
- **Proxy**: Design pattern; wrapper that encapsulates access to the resource
- **Request**: The function call made by the client, expecting some form of action in return from the subject. This request is sent to the interface which forwards it to the proxy. The proxy can then decide how the request should be completed.
- **Client**: Component that requests data or action to be performed

Structure and Runtime Behaviour

Structure (Class Diagram):



Behaviour (Sequence Diagram):



The client makes a request to what they believe to be the real subject. The request is parsed in the subject proxy and delegated accordingly to the logic that resides in the real subject. Before the request is delegated, the proxy is able to filter and handle edge cases according to the specific needs of the system. The proxy object should implement the same fields/methods as the real subject, as defined by the subject interface. The proxy must then delegate workload for each method to the real subject according to its desired use-case

Known Consequences

PositiveConsequences

- Increases modularity and encapsulation
- Protection proxy: adds security
- Smart proxy: adds assertions, memory management
- Virtual proxy: Improves performance with lazy loading
- Remote proxy: Facilitates remote code execution

Negative Consequences

- Developers may misuse the remote proxy thinking the logic runs locally not knowing the overhead they create with network requests
- Extra developer bandwidth is required as new changes must be made in the proxy concurrently with the real object.
- The additional level of indirection introduces complexity

NFPs

Improved NFPs

- **Performance:** Computation and memory expensive objects are instantiated by the proxy so those objects are only created when they are needed. The proxy implements the same interface as the expensive object so the methods are exposed by the light proxy object without the actual object being instantiated. So, the object is not created if the user does not have access to the method.
- **Security:** Only authorized users are allowed to access specific resources and methods of an object which increases confidentiality and integrity. Proxies can be chained to provide varying levels of authorization. Method usage can also be logged for increased monitoring and analysis.
- **Availability:** Returns a resource even if one is not currently found i.e. it takes some time to load an image so the proxy returns a loading image as opposed to an empty view until the image is fully loaded. The proxy will cache the image for future use.

Inhibited NFPs

Complexity: Having a proxy increases the size of the system, adding extra interdependency between client and subject.
Reliability: Difficult to assess as proxy can add another point of failure that must reflect any changes done to the object.
Maintainability: Someone may accidentally update one but not the other.

Credits

Students:

Chore Story (Cristian David, Mark Keller, Isabelle Leeson, Jim Wang)
Morpheme (Arthur Leung, Michael Li, Mark Liang, Victor Yan)

Instructor and TAs:

Mei Nagappan, Arman Naeimian, Ivens Portugal