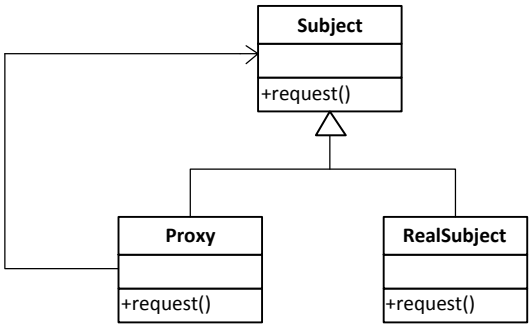
## 10 Proxy Pattern [Gamma et al]

Proxy pattern is used in scenarios when it is required to use avoid heavy-weight objects. So light-weight objects that are actually replica of the original objects exposing the same interface but with minimum functionality are created. Proxies support client server communication in numerous contexts such as a distributed environment. Since proxy pattern resemble with Decorator pattern in structure but yet not with intent. [38] Quoted an example of the proxy pattern using client server communication. Following are the variants of proxy pattern.

### 10.1 Proxy associated with Subject [38]

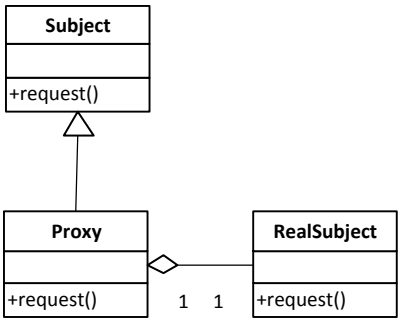
This variant is structurally close to Decorator pattern with a basic difference is decorator role is abstract whereas proxy role is concrete. Proxy initializes its association to Subject role with a reference to an object of RealSubject which is instantiated inside the constructor of Proxy.




10.1 UML Diagram of proxy associated with subject class [37].

### 10.2 Aggregation relation of Proxy and Subject [3]

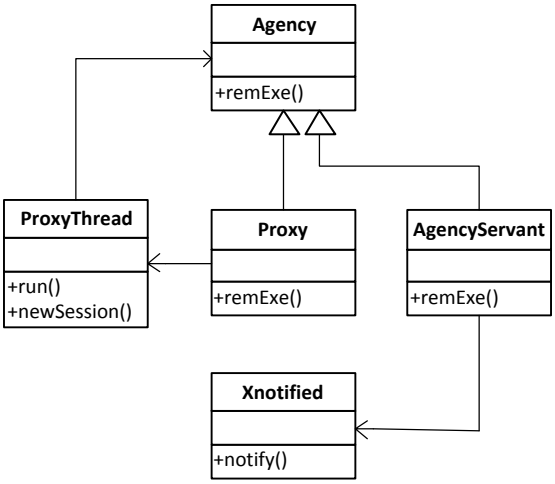
In this variant Proxy holds an aggregate relation of the Real Subject class. This implementation uses idea of parameterized instantiation of the RealSubjet Class. There could be different remote objects that channelize the creation of certain specific type object which is evaluated based on certain condition. The RealSubject is initialized at either the constructor level or at method level.



10.2 UML Diagram of proxy associated with subject class [37].

## 10.3 Proxy in heterogeneous environment [39]

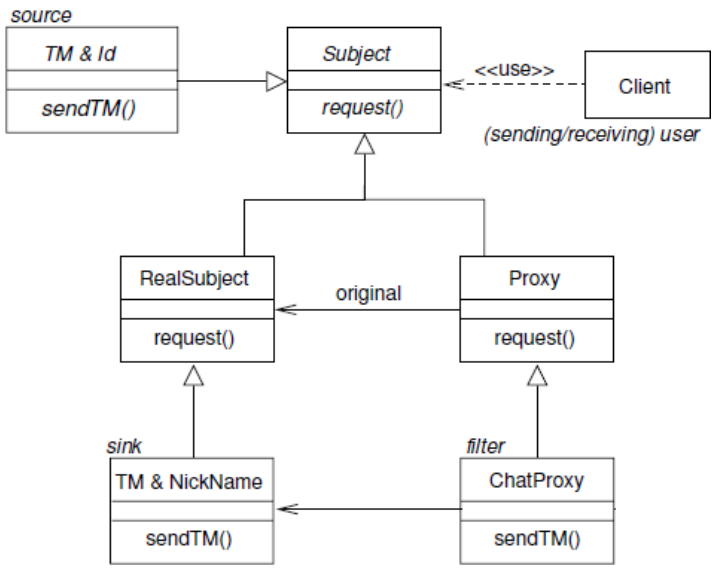
A proxy object appears in this variation is to delegate the method call to the Real Subject which resides in the database based on the authentication (Proxy Thread).




10.3 UML Diagram of proxy in heterogeneous environment [39].

## 10.4 Pipe and filter implementation [40]

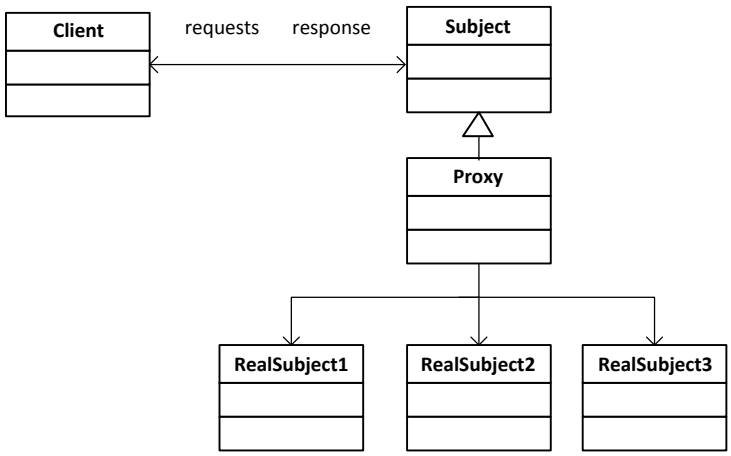
This variant uses proxy pattern with combination an architectural style called pipe and filter. A number of transformations are applied on the data to reach from source to destination via pipes. In this variant the Proxy and RealSubject Classes are subclassed to extend the types of objects that call original subject belonging to a different domain. [41] implemented a chat application, there could be other examples as well like to channelize the input data to multiple distributed sources.




10.4 UML Diagram of pipe and filter style proxy [40].

## 10.5 Reverse Proxy [41]

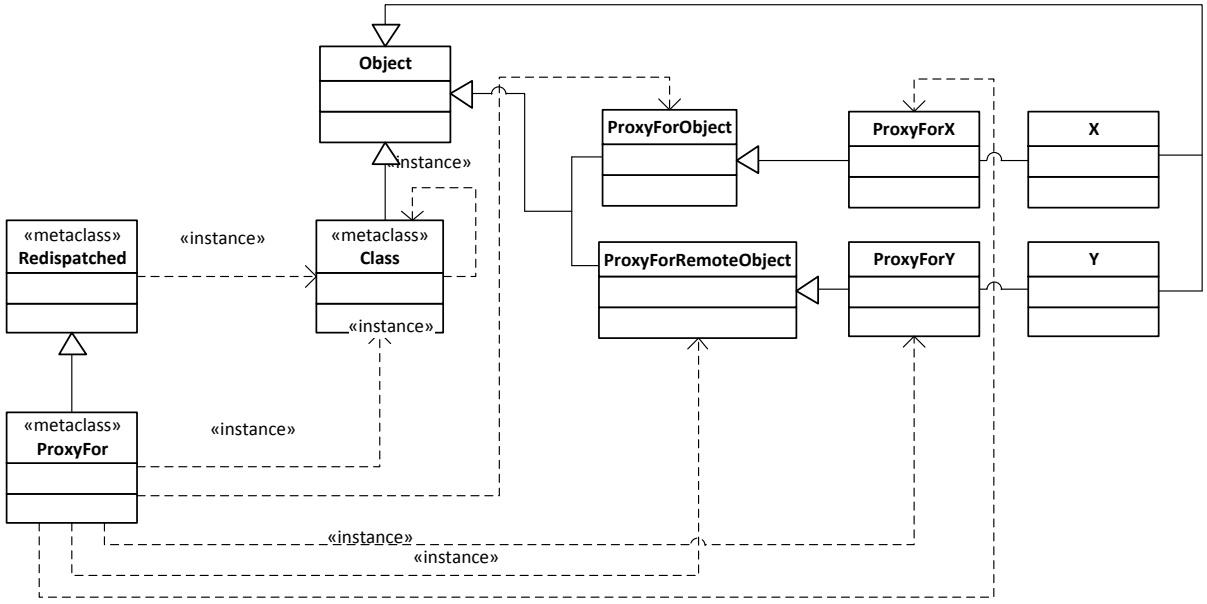
A proxy object is surrogate for client objects whereas reverse proxy object serves as placeholder for servers. This variant redirects a request and forwards its response to the client. There are number of replicated servers which act as listeners for the request to maintain transparency. A reverse proxy object inquires for any availability and delegate the request to a replica if a certain server does not respond in time.




10.5 UML Diagram of proxy associated with subject class [41].

## 10.6 Generic Proxy Implementation using reflective architecture [42]

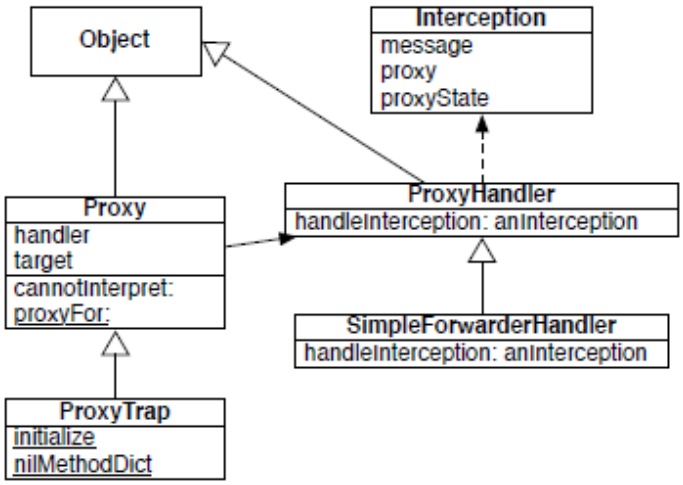
This variant is a generic implementation to incorporate the functionalities of different kinds of proxies like virtual, remote, protection, synchronization, counting, and firewall proxy. Local and remote proxies are defined as ProxyForX and ProxyForY which represent X and Y objects.

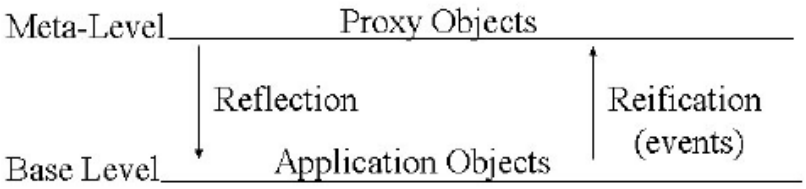



10.6 UML Diagram of proxy associated with subject class [42].

## 10.7 Dynamic Proxies [43]

In Existing work, proxies were limited to regular objects only. But today proxy can be a target class or a method. In all canonical representations there were no clear divisions between interception of message and handling [44]. Due to certain limitations the community advocates the use of Dynamic Proxies which are termed as Meta objects [45] that contain the Meta representation of other objects at base level. Meta objects are like normal objects [45] and they expose certain interfaces which are used in Meta Object Protocols [47]. [44] Describes an implementation of Ghost Model which supports proxies for regular objects as well as classes, methods. Meta objects are discussed in detail by [46].

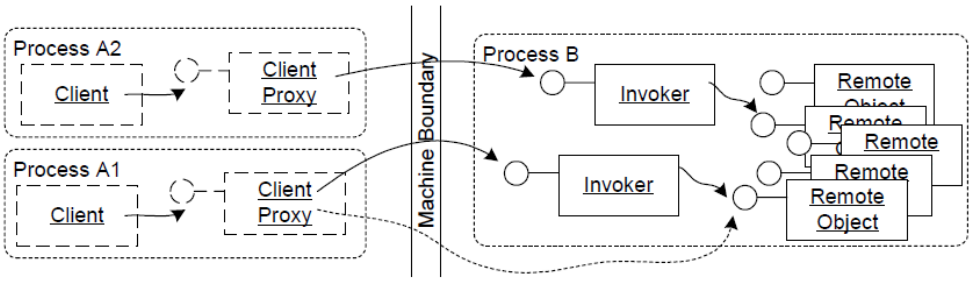



10.7 Diagram of basic architecture separating base and Meta level code [47] and UML Diagram of dynamic proxies [43]

## 10.8 Client Proxy [48]

This variant of proxy pattern deals with objects that lies on a distribute network. Such an object which is instantiated and maintained remotely can be accessed via distributed object frameworks. Programming across networks is not always an easy task. Remote invocation of objects involves bunch of error conditions like network latency, unavailability, unnecessary delays, failures, crashes due to which it is not optimistic to think for a winwin situation. Target object is accessed through the use of invokers which are a part of server application and invokers dispatches the request through which clients feel transparency of a remote invocation call as local invocation call. The invokers are provided with remoting information i.e. operation name.




10.8 Diagram of client proxy [48]