Proxy (Surrogate) Desing Pattern

Explanation, structure and examples.

What is a proxy?

The word “proxy” means or can be used for :

- A figure that can be used to represent something else
- Placeholder for another object, a substitude
- A person (authority) authorized to act for another person

Motivation and intention

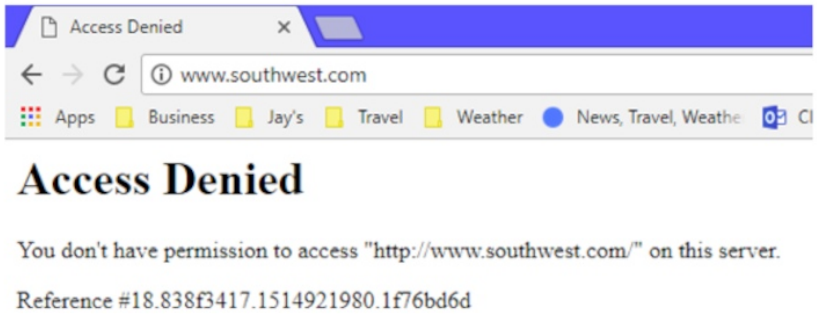
The proxy design pattern is used to create a wrapper to cover the main object's complexity or confidentiality from the client part. This abstraction and encapsulation on real subject solves many different problems, yet all of them involve some object taking the place of another object. By saying "taking the place", we mean that both acting like it and/or representing it as its state.

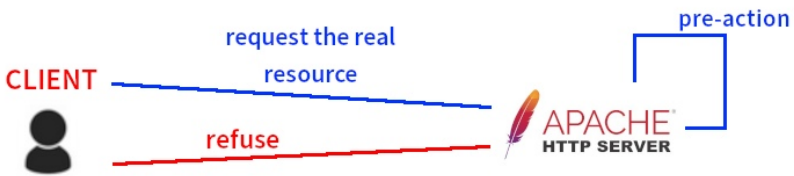
Motivation and intention

This design pattern may be used for:

- Security purposes such as setting restrictions to the clients according to their states, roles, limitations, to protect the original object in short.
- To hide the underlying networking logic.
- To load large objects abroad as a representation on demand in order to reduce costly operations or some other purposes, to remotely access a local object so to speak.
- To make the original object smarter.

Example : Access restriction



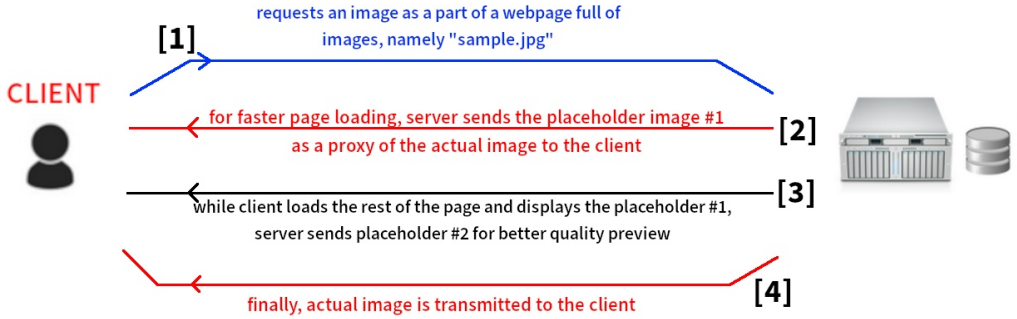


Hasn't got authenticated or does not have priviledges for accessing the resource.

A web server acts as a proxy on behalf of the real subject.

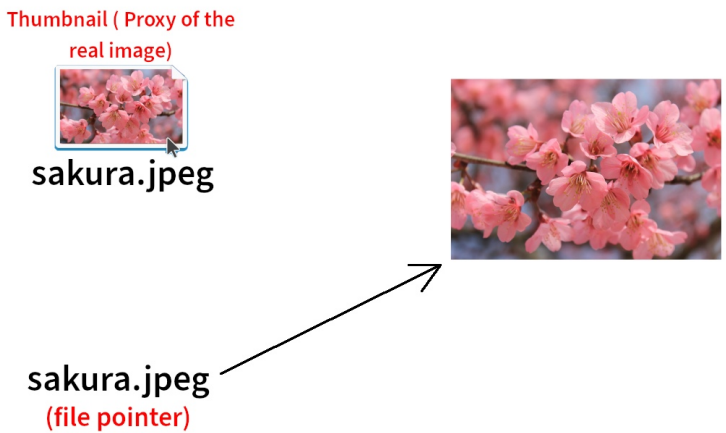
Backend server that serves the real resource which is the index page

Example : Loading objects abroad, Lazy Loading.



Applicability

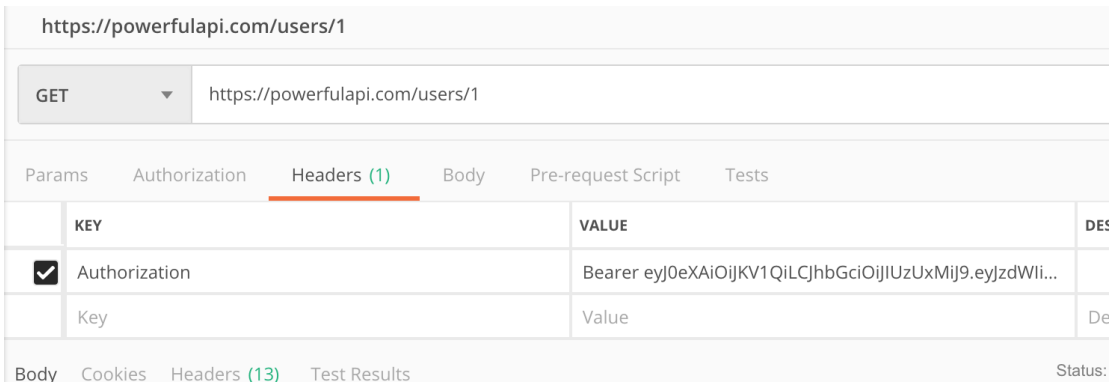
To sum up and categorize its types, the proxy design pattern is suitable in situations where there is a need for a more versatile or sophisticated -for an intention- reference to an object rather than a reference in simplest form, like pointer.



Types of Proxies

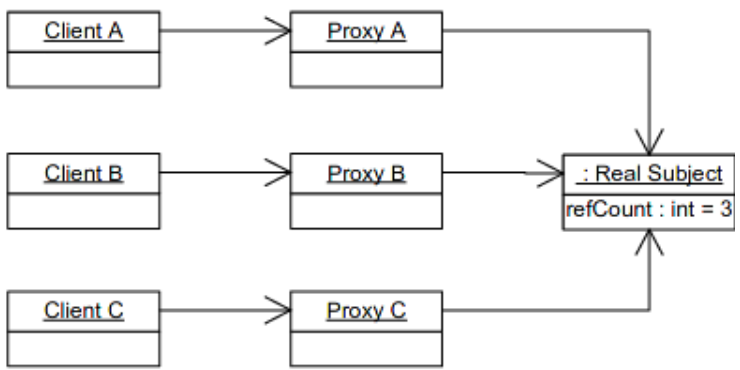
There are four main realizations of Proxy pattern we'll mention:

· Protection (Access) Proxy : It controls and manipulates access to the real object based on priviledges. It effectively sets a abstraction layer between the real object and the client that is trying to access to it. It is useful when objects or clients have different access rights like in the examples of the role-based access control systems.



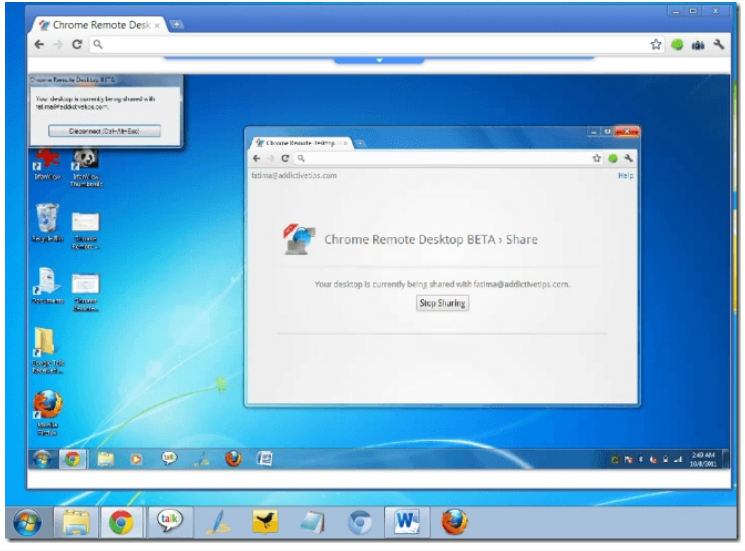
Types of Proxies

· Smart Proxy ： This proxy type is used to add additional actions and assertions to make the real object smarter. For example, in a programming language that implements a garbage collector, by Proxies, we can maintain the reference count inside the object to track the number of references to itself by other objects. When the reference count to that object falls to the zero, that object can be freed. In this situation, references held are the specified proxies (placeholders) of the real object. Smart pointers made our life easier if you think C programming language and its need to manual management of memory.



Types of Proxies

· Remote Proxy : Its main purpose is to provide a local representative for an object in a different address space abroad. If it's implemented accordingly, even can hide the fact that object resides in a different space. Remote code execution can be also used to communicate with the real object. The logic and interface of it will be encapsulated and client application need not to worry about it.



Types of Proxies

· Virtual Proxy : Generally, used for creating an optimized delegate of space or time-consuming objects on demand. The example provided before on lazy loading of images is an example of such a proxy. Virtual Proxy often defers the creation of the big “expensive to create” object until it is needed. It also acts as a surrogate by handling requests for the real object before and while it is being created, after creation is done, the proxy delegates requests directly to the real object.

Skeleton (Virtual) Image [70KB]

(Act as a vassal until real object is generated and transmitted)

Original Image [5MB]

Participants - Vocabulary

· Real Subject : Defines the real object that the proxy represents.

· Proxy : Design pattern, wraps and encapsulates the access to the resource. Has different responsibilities according to their kind mentioned before. But in general maintans a reference to provide proxy access to the Real Subject, provides an interface, controls the access.

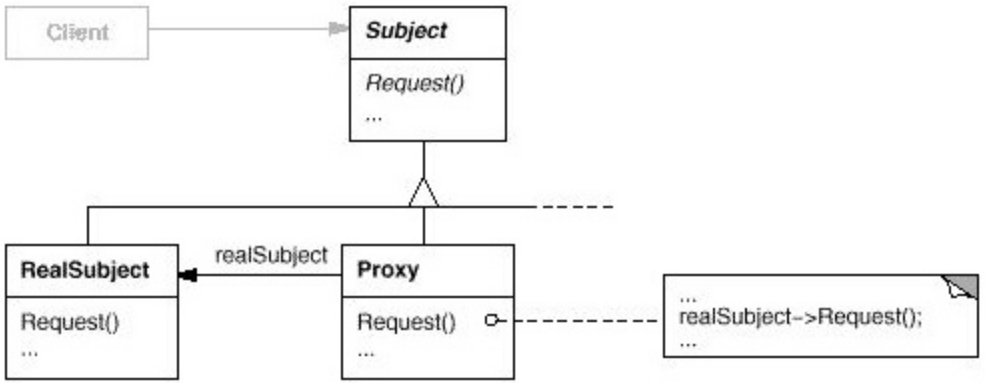
· Client : Component that requests data or action to be performed.

· Request : A function call made by the client, expecting a form of action/response.

· Subject (Interface) : Relays or handles the requests, defines a common interface for Real Subject and Proxy so that a Proxy can be used anywhere a Real Subject is expected.

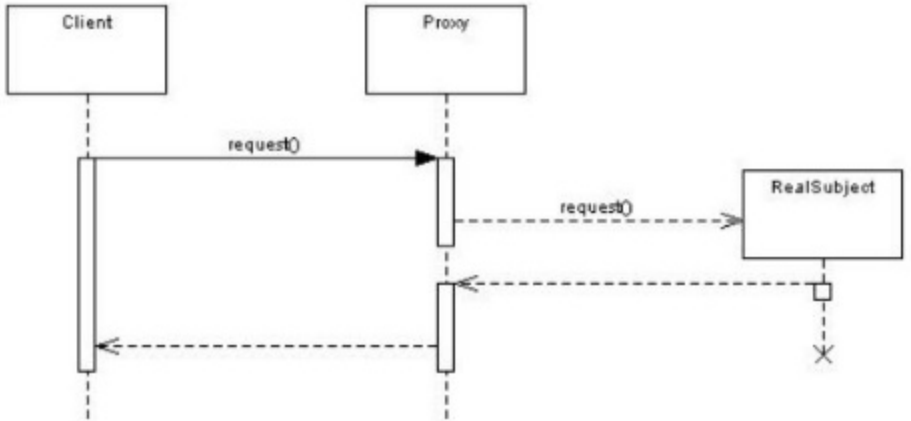
General Structure and Runtime Behaviour

Class Diagram :



General Structure and Runtime Behaviour

Sequence Diagram:



Toy Project: Source Code

```
7   ################################################
8   # ApiInterface class is SUBJECT INTERFACE
9   ################################################
10  class ApiInterface:
11      # This class is the common interface that both proxy and the real object
12      # will implement.
13
14      def fetchResource(self, user: str) -> str:
15          raise NotImplementedError()
16
17  ################################################
18  # RealApi class is REAL SUBJECT
19  ################################################
20  class RealApi(ApiInterface):
21      # This class is the real object that has not protection on its resources
22      # whatsoever.
23      def __init__(self) -> None:
24          pass
25
26      def fetchResource(self, user: str) -> str:
27          return (f'I am serving a resource to the client: "{user}"\n')
28
```

Toy Project: Source Code

```
29  ##############################################
30  # Proxy class is PROTECTION PROXY
31  ##############################################
32  class Proxy(ApiInterface):
33
34      def __init__(self) -> None:
35          self.real_subject = RealApi()
36
37      def fetchResource(self, user: str) -> str:
38          print(f'Client "{user}" has requested to access a resource...\n')
39
40          if user == "Authorized User":
41              return self.real_subject.fetchResource(user)
42          else:
43              return f"Sorry, only \"Authorized User\" can access my resource.\nYou called me as \"{user}\"\n"
44
45  ##########################################################################################
46
47  def clientAction(server: Union[RealApi, Proxy], user: str) -> str:
48      return server.fetchResource(user=user)
49
50  ##########################################################################################
```

Toy Project: Source Code

```
class Win(QMainWindow):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Protected API application.")
        self.setGeometry(10, 10, 400, 300)
        self._proxy = Proxy()
        self._real_api = RealApi()


        self.formGroupBox = QGroupBox("Api Fetch:")
        self.serverTypeComboBox = QComboBox()
        self.serverTypeComboBox.addItems(["Real Api", "Proxy Server"])
        self.userNameLineEdit = QLineEdit()
        self.createForm()
        self.buttonBox = QPushButton('Fetch Resource')
        self.buttonBox.clicked.connect(self.doClientAction)
        self.outputBox = QTextBrowser()

        layout = QVBoxLayout()
        layout.addWidget(self.formGroupBox)
        layout.addWidget(self.buttonBox)
        layout.addWidget(self.outputBox)
        widget = QtWidgets.QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
        self.show()


    def createForm(self):
        layout = QFormLayout()
        layout.addRow(QLabel("User Credentials:"), self.userNameLineEdit)
        layout.addRow(QLabel("Server to fetch:"), self.serverTypeComboBox)
        self.formGroupBox.setLayout(layout)

    def doClientAction(self):
        #self.outputBox.insertPlainText(self.userNameLineEdit.text())
        #self.outputBox.insertPlainText(self.serverTypeComboBox.currentText())
        username = self.userNameLineEdit.text()
        if self.serverTypeComboBox.currentText() == "Real Api":
            self.outputBox.insertPlainText(clientAction(self._real_api, user=username))
        else:
            self.outputBox.insertPlainText(clientAction(self._proxy, user=username))
        self.outputBox.insertPlainText("~~~~~~ END OF TRANSACTION ~~~~~~\n")
```

References

Proxy Design Pattern [Accessed 20 December 2021]

Gamma, E., Helm, R. and Johnson, R., 1998. Design patterns. Reading [etc.]: Addison Wesley.