Chapter 25

Proxy

25.1 The Problem

Consider a large document with a lot of images in it. Images are expensive objects in terms of memory size and thus time to load and decode. This means the document will be slow to load as it takes a long time to read from disk. The result is that the user will perceive the word processor as slow and perhaps buy another product. So, the challenge is to make the system load the document fast even if it contains a large number of objects that are slow to load.

25.2 A Solution

The key insight to solve the problem is to realize that most of the images will not be visible once the document has been loaded. Usually a word processor starts by showing page one. If the document contains 50 images but only one appears on page one, it suffices to load the text and the single image on the first page and defer loading the other images until they become visible. If the user only looks at the first couple of pages before closing the document, the system has even avoided the cost of loading most of the images all together. So, I need a mechanism to delay loading images until they need to be shown. In design language, I can state this as defer loading until the `show()` method is called on image objects. A classic variability challenge that calls for the ③-①-② process.

③ *Encapsulate what varies.* Seen from the client (the word processor) I want the image objects to have variable behavior; those that become visible will fetch image data and show themselves whereas those that have not yet been visible simply do not spend time loading the image data.

① *Program to an interface.* By insisting that images are only accessed from the client via an interface I can provide it with an intermediate object that will defer the loading until the `show()` method is called but in all other respects acts just like a real image object.

② *Object composition.* Just like DECORATOR I can compose the real image behavior by putting an "object-in-front", the proxy, that will only fetch the real image data once it needs to be shown.

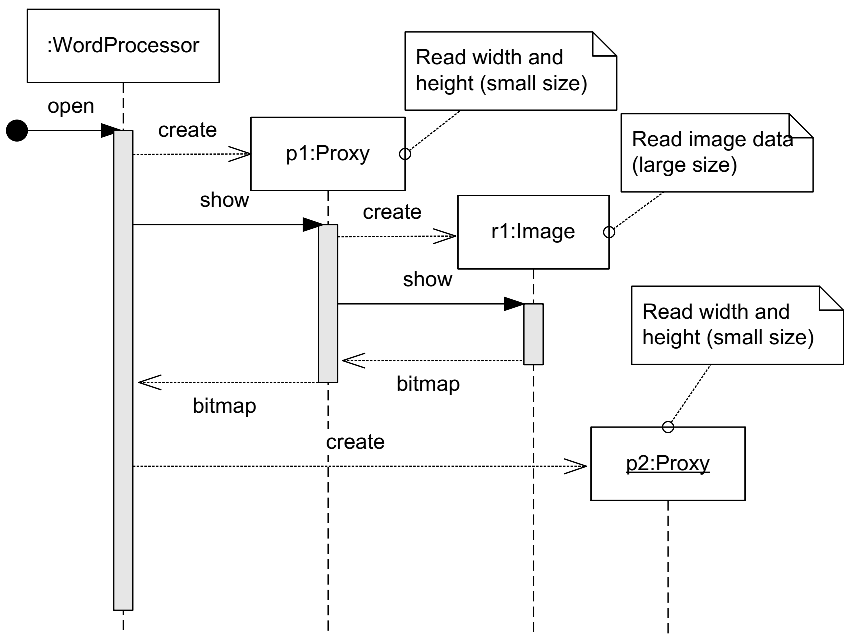

:WordProcessor
open
create
p1:Proxy
Read width and height (small size)
show
create
r1:Image
Read image data (large size)
show
Read width and height (small size)
bitmap
bitmap
create
p2:Proxy

Figure 25.1: Proxy defers the loading of image data.

Figure 25.1 shows a sequence diagram of a potential load of a document with two images, one that is shown and one that is not. Typically, the proxies read essential but small sized data, like the width and height of the image, thus its getWidth() and getHeight() methods return proper values so the word processor can layout pages correctly. When the bulky data is required for the show() method, the proxy creates the real image object which then goes on to load the raster image. Note that to the client the first invocation of show() is much slower than subsequent invocations.

☞ Review and execute the demonstration of the Proxy pattern. The source code is in folder *chapter/proxy* at the web site. Try to let WordProcessor declare JPGImage directly and see the difference in behavior.

25.3 The Proxy Pattern

This is the PROXY pattern (design pattern box 25.1, page 47). Its intent is

Provide a surrogate or placeholder for another object to control access to it.

The central roles are the **Subject** that defines the interface for the objects that can be proxied, **RealSubject** is the real implementation of the object, while **Proxy** is the intermediate object that represents the real subject.

If the developer has adhered to the ① *program to an interface* principle it is easy to introduce PROXY at a later stage in the development as both **proxy** and **real subject** roles have the same protocol and interface.

Proxies are very versatile in their use. Typical uses are:

- *Protection proxies / Access control.* Protection proxies may check that the caller has the correct permissions to interact with the real subject. If not, requests are not granted. You may also use it to handle concurrent access to it by locking.
- *Virtual proxies / Performance control.* Proxies that cache information or results that are expensive in terms of computation time or memory consumption. The image proxy above is an example of a virtual proxy.
- *Remote proxies / Remote access.* Proxies that act as local representatives of objects that are really located on a remote computer. Remote proxies primary responsibility is to encode method calls and parameters into a binary string that can be sent over the network to the real object on the remote computer, accept the return result, decode it and return it to the caller.

The proxy must of course maintain a reference to the real subject to delegate requests to it. However, this reference may take various forms depending on the type of proxy. In the case of remote proxies, the reference is indirect, such as host address and local address at host. For virtual proxies, the reference will first be resolved when it is needed: for instance in case of the image proxy the real subject is not created until `show()` is called and until then it is probably an image file name or an offset into the document's data on disk.

Proxy has the benefit that it strengthens reuse potential. As "housekeeping" tasks (like access control, caching, etc.) are the responsibility of the proxy I avoid putting them in the real subject implementations increasing the likelihood they can be reused somewhere else. This is the classic benefit of compositional designs: abstractions tend to be smaller and with a few clear responsibilities leading to higher cohesion and easier code to read. The liabilities are the extra level of indirection inherent to compositional designs; and if legacy code has not been *programmed to an interface* then there will be an overhead in introducing it.

Exercise 25.1: If you look at the structure of PROXY and DECORATOR they are very similar. Outline their similarities. Explain the differences between the two, and argue why proxy is not just a decorator.

If a proxy needs to create or delete real subjects the coupling between them becomes tight. Thus if you have several different implementations of the real subject you will have to have multiple proxy implementations as well. If not you may use a single proxy implementation for all types of real subjects.

Typically, it is the client that must create instances of the subject interface which creates a hard coupling between the client and the proxy class, as you saw in Figure 25.1. You may use ABSTRACT FACTORY to reduce coupling.

The proxy pattern is the key pattern used in Java Remote Method Invocation (RMI) and C# Remoting that allows distributed object programming. These techniques mask distribution by providing a pseudo object model for interacting with objects located on remote machines in a network. The client interacts with a proxy of the remote object; the proxy then forwards methods calls to the remote object, waits for the response, and translates it back to a normal method return value. Standard or special compilers can generate the proxies classes based upon the interfaces that define remotely accessible objects.

☞ Study the Java RMI system. You can find several good resources on the internet. The online "Java Tutorial" provides small examples of RMI programs and guidelines for compiling and executing them.

25.4 Selected Solutions

Discussion of Exercise 25.1:

Indeed they are very similar in that both the decorator and the proxy contain references to their decorated/proxied object. However, their intents are different: decorators add responsibilities to an object whereas proxies control access to it. Also at the implementation level there are differences. Proxies are typically tightly bound to the object type they control; for instance virtual proxies are responsible for creating the real objects. Decorators are loosely coupled to their decorated object as they only know them by interface, and they are not responsible for configuration as it is the client that sets up the chain of decorated objects.

25.5 Review Questions

Describe the PROXY pattern. What problem does it solve? What is its structure and what is the protocol? What roles and responsibilities are defined? What are the benefits and liabilities?

25.6 Further Exercises

Exercise 25.2. Source code directory:
`chapter/proxy`

Consider that some images in a document are allowed to be viewed by persons with a special security clearance. In order to view images in the document they have to type a password the first time they are about to view a protected image. Once the password has been typed, all images in the document can be viewed.

1. Enhance the ProxyDemo with such an image protection proxy.
2. Add an access proxy that counts the number of times each image has been viewed.

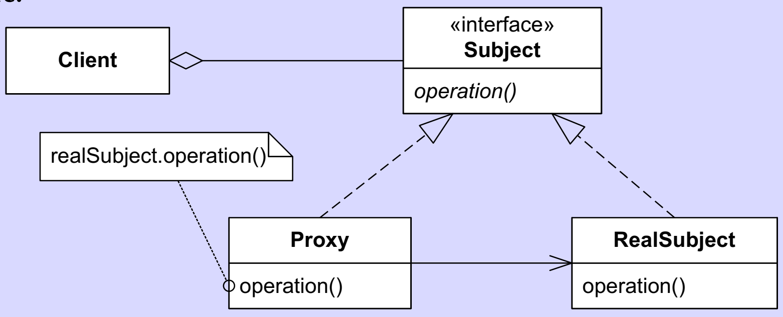
[25.1] Design Pattern: Proxy

Intent Provide a surrogate or placeholder for another object to control access to it.

Problem An object is highly resource demanding and will negatively affect the client's resource requirements even if the object is not used at all; or we need different types of housekeeping when clients access the object, like access logging (audit trail), access control, or pay-by-access.

Solution Define a placeholder object, the Proxy, that acts on behalf of the real object. The proxy can defer loading the real object, control access to it, or in other ways lower resource demands or implement housekeeping tasks.

Structure:



Roles A **Client** only interacts via a **Subject** interface. The **RealSubject** is the true object implementing resource-demanding operations (bandwidth, computation, memory usage, etc.) or operations that need access control (security, pay-by-access, logging, etc.). A **Proxy** implements the **Subject** interface and provides the relevant access control by holding a reference to the real subject and delegating operations to it.

Cost - Benefit It *strengthens reuse* as the housekeeping tasks are separated from the real subject operations. Thus the subject does not need to implement the housekeeping itself; and the proxy can act as proxy for several different types of real subjects.