Software Engineering

Lesson Design Pattern 07
Proxy
v1.0

Uwe Gühl

Fall 2007/ 2008

Proxy

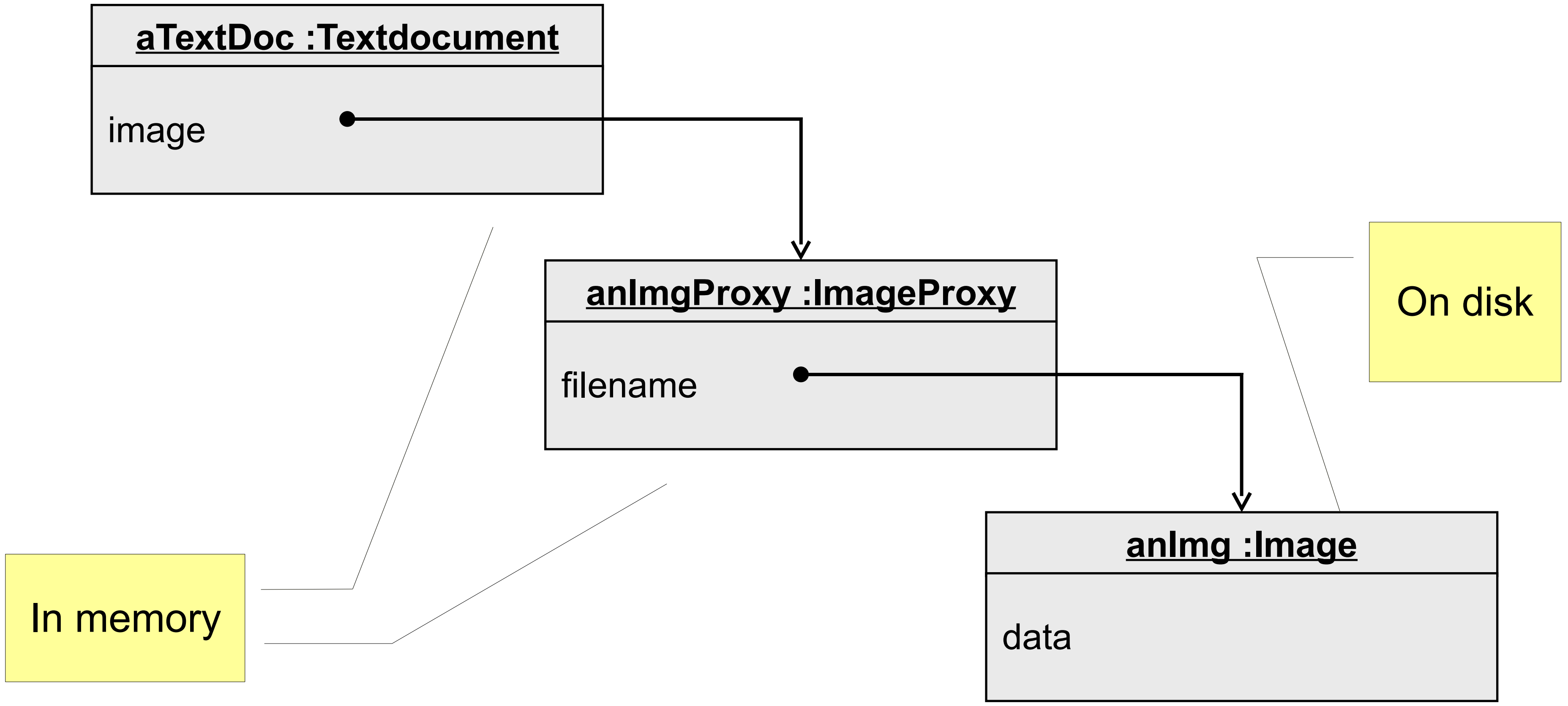
- Intent:
  - Provide a placeholder for another object to control access to it
  - Use a wrapper and delegation to enable distributed, controlled, or intelligent access
  - ... also known as Surrogate
  - ... is a Structural Pattern

Proxy

- Motivation
  - If the creation and initialization of objects are expensive, e. g. resource-hungry, it makes sense to postpone it unless and until they are really needed
  - Example: Document with graphical objects in it
    - Big images could be expansive to create
    - Opening a document should be fast!
    - Not every image is necessary in the beginning, typically they won't be visible at the same time
    - Idea: Use another object instead: An image **proxy** – acting for the real image

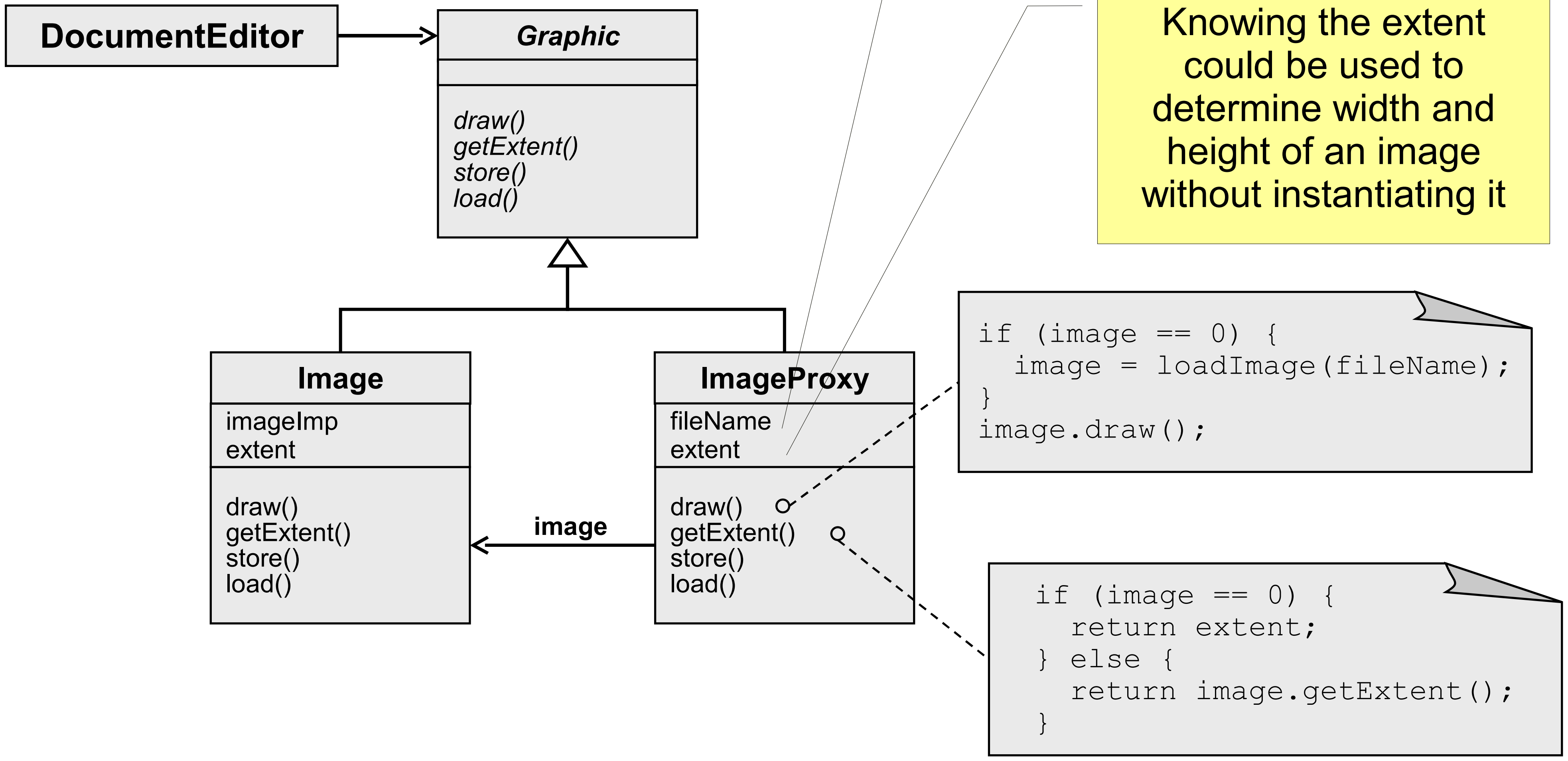
Proxy

- Motivation
  - The image **proxy** –calls the real object only after the request of the editor, invoking `draw()`



Proxy

- Motivation

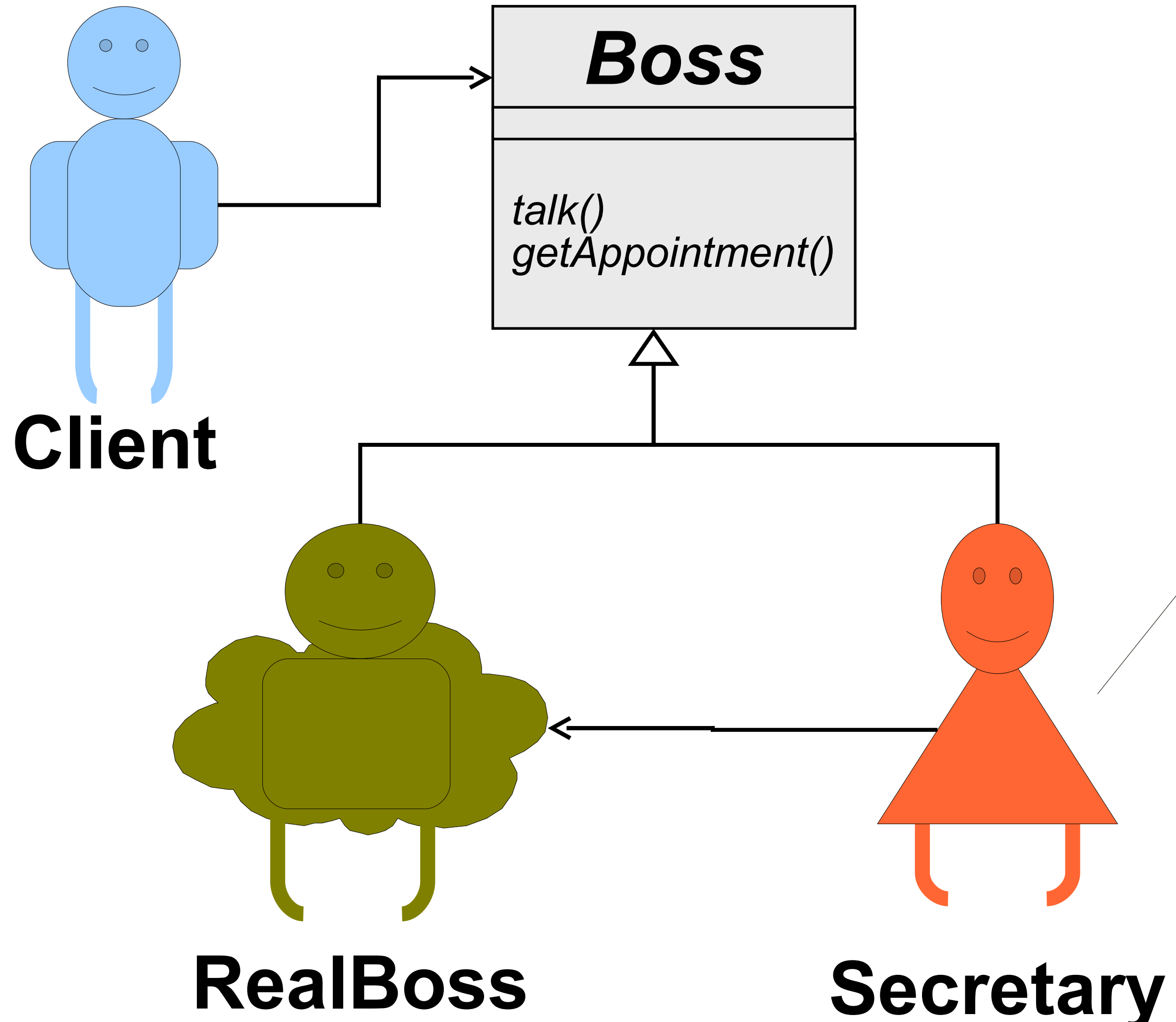


```
if (image == 0) {
   image = loadImage(fileName);
}
image.draw();
```

```
if (image == 0) {
   return extent;
} else {
   return image.getExtent();
}
```

Proxy

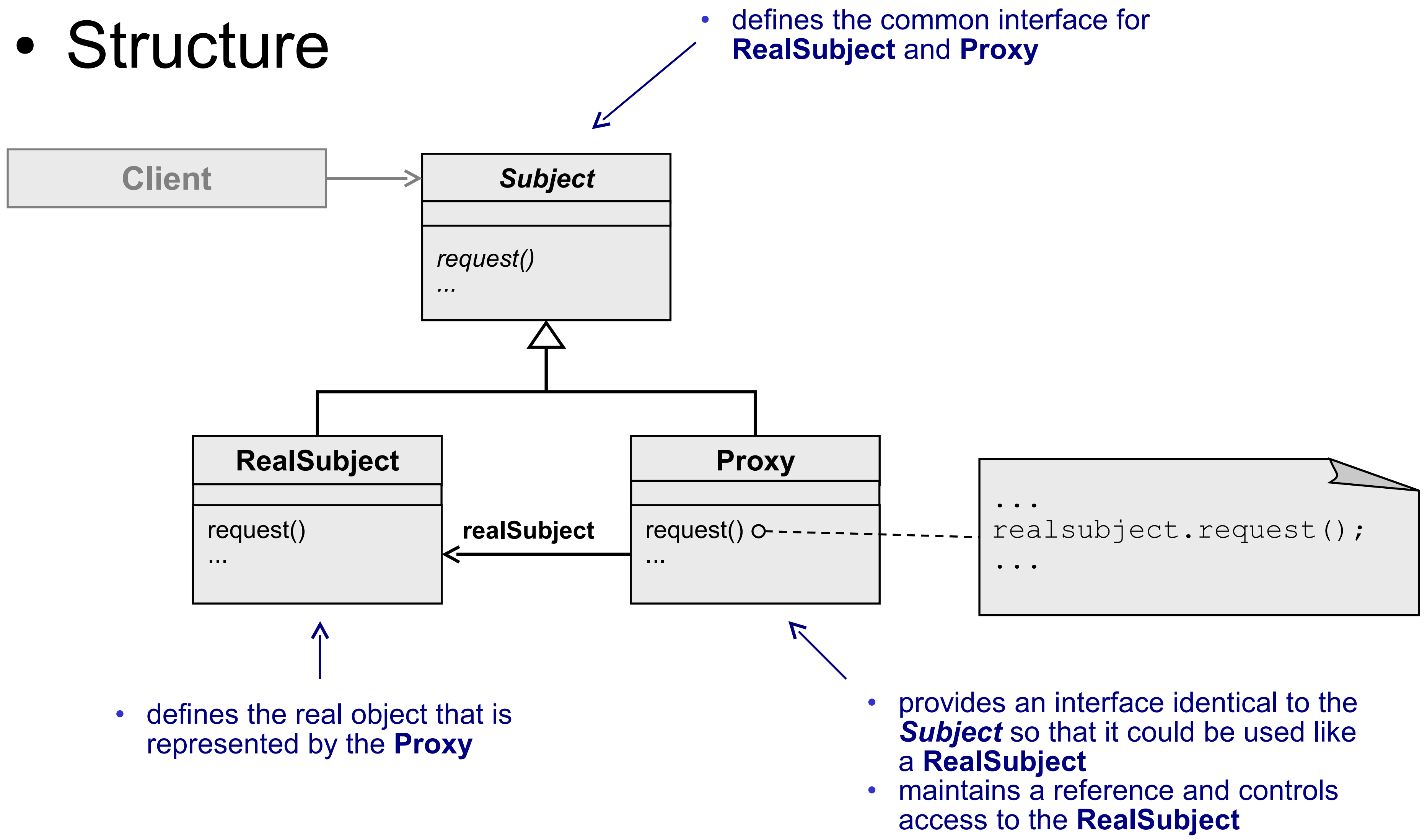
- Non Software Example



- getAppointment() and talk() with RealBoss only after talking to Secretary
- Secretary moderates between Client and Boss

Proxy

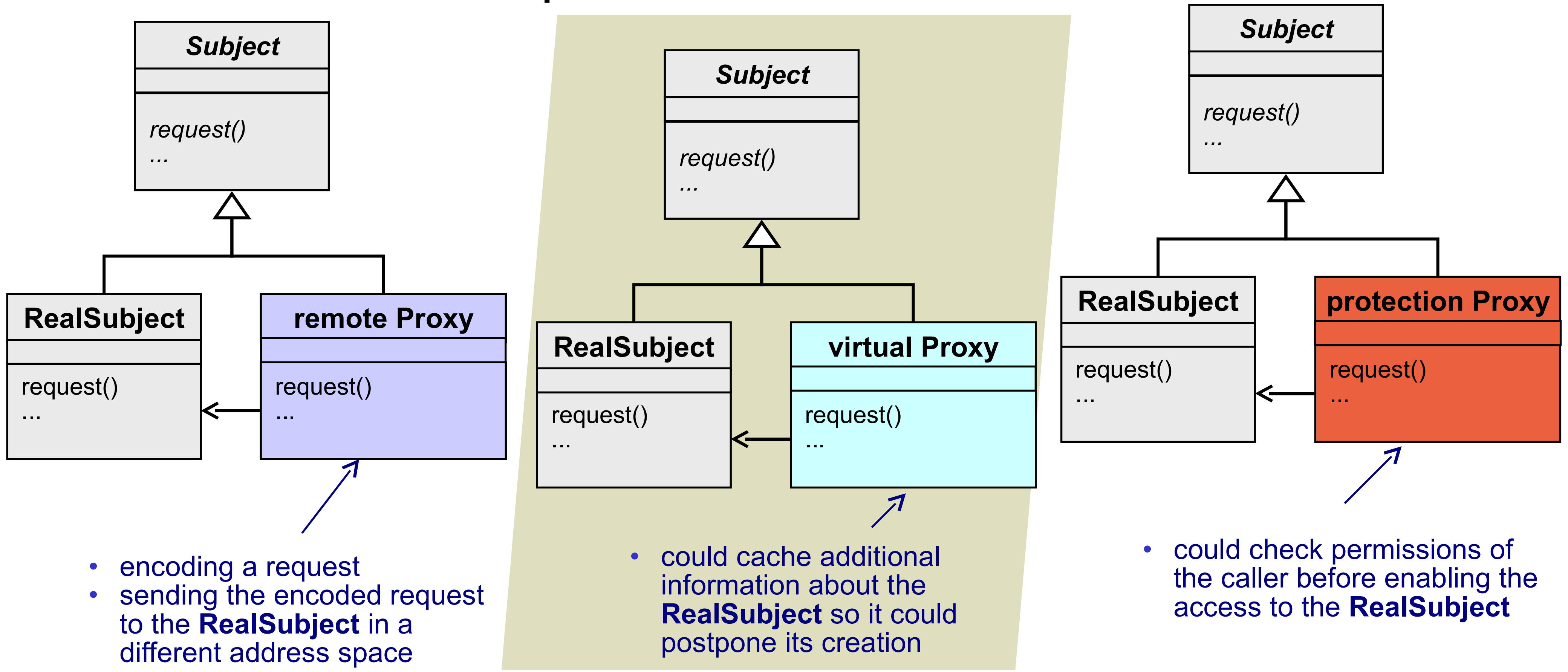
- Structure



defines the common interface for RealSubject and Proxy
Client
Subject
request()
...
RealSubject
request()
...
realSubject
Proxy
request()
...
...
realsubject.request();
...
defines the real object that is represented by the Proxy
provides an interface identical to the Subject so that it could be used like a RealSubject
maintains a reference and controls access to the RealSubject

Proxy

- Structure
  - additional responsibilities of different **Proxies**



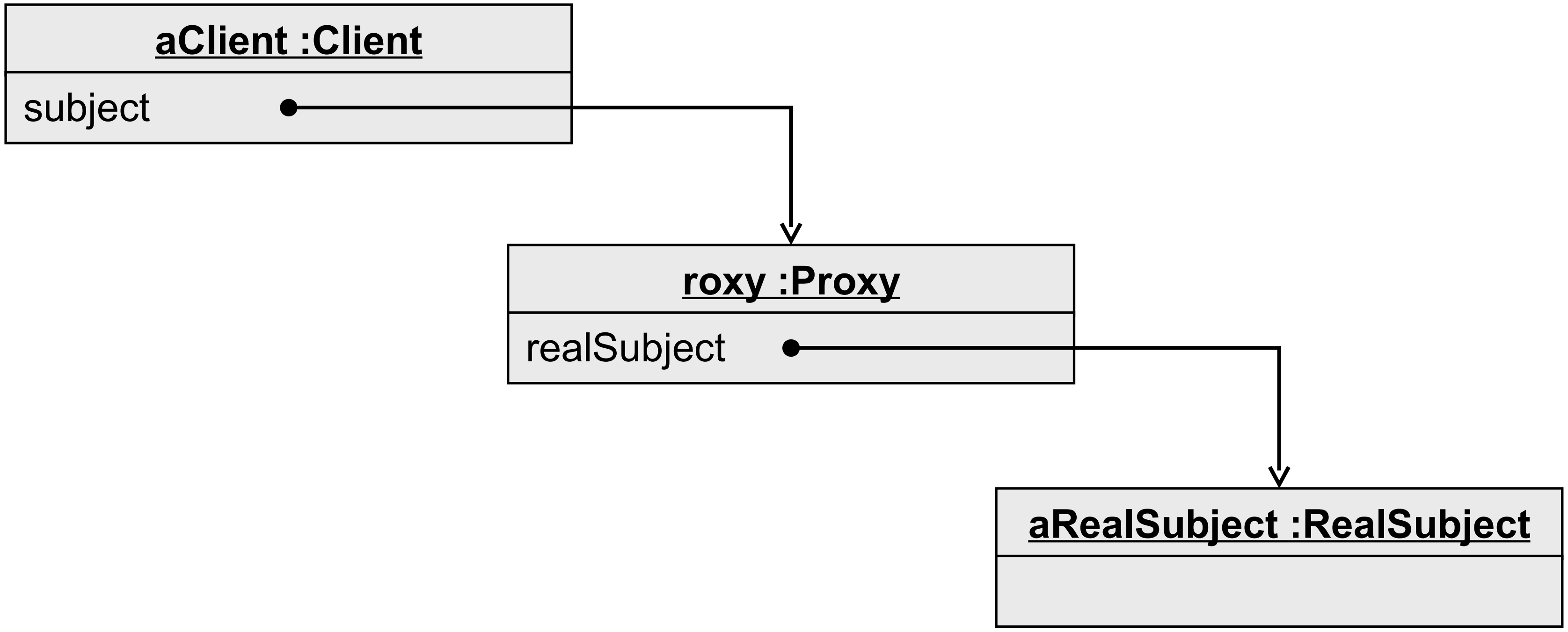
- encoding a request
- sending the encoded request to the **RealSubject** in a different address space

- could cache additional information about the **RealSubject** so it could postpone its creation

- could check permissions of the caller before enabling the access to the **RealSubject**

Proxy

- Structure
  - Possible object diagram of a proxy structure at run- time



Proxy

- Collaboration
  - **Proxy** forwards requests to **RealSubject** when appropriate, depending on the kind of proxy
  - The **RealSubject** provides the key functionality, the **Proxy** provides or refuses access to it

Proxy

- Applicability
  Use the Proxy Pattern if
  - you need to provide a substitute for an object because it's inconvenient or not wanted to access it directly. Possible reasons, if an object
    - is located in a different address space
      - you would use it as Remote Proxy
    - has restricted access with different access rights
      - you would use it as Protection Proxy
    - is expensive to create, and should be created only on demand
      - you would use it as Virtual Proxy

Proxy

- Applicability
  Use the Proxy Pattern if
  - you need a smart reference performing additional actions instead of a bare pointer; requirements could be
    - counting the number of references to a real object
      - could be helpful if an object should / could be deleted or destructed if there are no more references (*smart pointer*)
    - loading a persistent object into memory after first referencing
    - locking of an object so that it could be changed only by one other object

Proxy

- Consequences

+ The additional indirections could be used so that a proxy acts as a

  - remote proxy
    to hide the fact that an object resides in a different address space
  - virtual proxy
    to perform optimizations
  - protection proxy
    to control access to objects
  - smart pointer
    to do additional meaningful jobs, e. g. for garbage collection

Proxy

- Consequences

+ Allows optimization like **copy-on-write**

  - Instead of really copying a large object this process is postponed until there are changes
  - Subject must be reference counted – copy then means increasing the reference count
  - If an operation modifies the subject, it gets copied
  - If the reference count is zero, the subject gets deleted

Proxy

- Implementation
  - Knowledge about the real subject
    - A communication through an abstract interface is possible – so all **RealSubject** classes could be treated uniformly
  - Special language issues
    - C++: Overloading "`->`" - the member access operator
    - Smalltalk: Using "`doesNotUnderstand`"
      - supporting a hook to support automatic forwarding of requests
  - Reference to real subject before it is instnatiated
    - address space-independent object identifier (e. g. file name)

Proxy

- Implementation
  - Checklist [from Vince Huston, vincehuston.org)
    1. Identify what has to be done and implemented as proxy
    2. Define the ***Subject*** as an interface so that the **Proxy** and the **RealSubject** as original component are interchangeable
    3. Consider to define a Factory that can encapsulate the decision of whether a proxy or original object is desirable.
    4. **Proxy** points to **RealSubject** and implements the interface.
    5. The pointer may be initialized at construction, or on first use.
    6. Each wrapper method contributes its leverage, and delegates to the **RealSubject** object.

Proxy

- Known Uses (see [GHJ+95])
  - ET++ text building block classes
  - NEXTSTEP uses proxies as local representations for objects that may be distributed
  - "stub" code in RPC and CORBA provides a local representative as a remote proxy

RPC = Remote Procedure Call
CORBA = Common Object Request Broker Architecture

Proxy

- Related Patterns
  - Adapter
    - An adapter offers a different interface to the adaptees
    - A proxy offers the same interface as its subject

Proxy

- Related Patterns
  - Decorator
    - Decorators may have the same implementations as Proxies but they have another intent
    - Both, Decorator and Proxy, compose an object and provide an identical interface to clients
    - A Decorator
      - adds more responsibilities to an object without subclassing
      - uses recursive composition to add flexible additional behavior
    - A Proxy
      - controls access to an object
      - not designed for recursion
      - focuses on one relationship – between the proxy and its subject