AARHUS UNIVERSITET

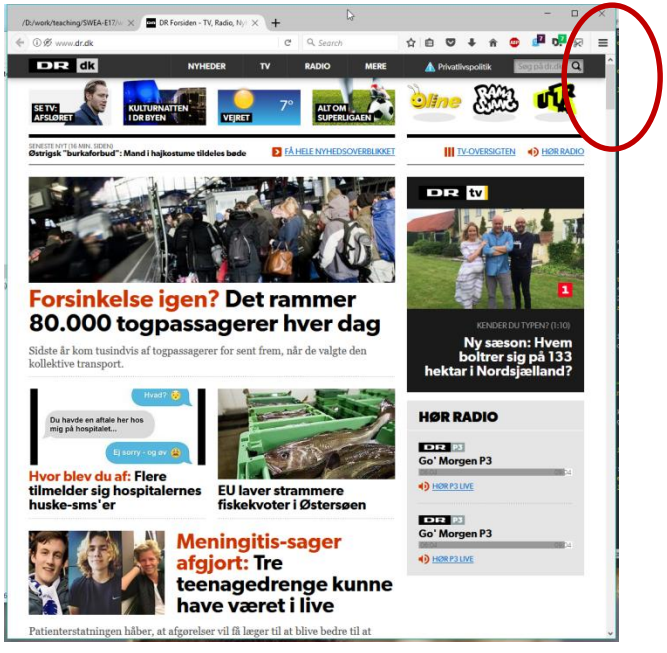
Software Engineering and Architecture

Pattern Catalog: Proxy

AARHUS UNIVERSITET

Proxy

- A web page contains a lot of images, many of which will never be displayed as they are ‘at the end of the scroll’ where the average user do not see at all. Can we avoid downloading them?
  - Same for a Word document etc.
    - Loading from disk also takes time…
- *Download on demand* – i.e. only when they become visible



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.

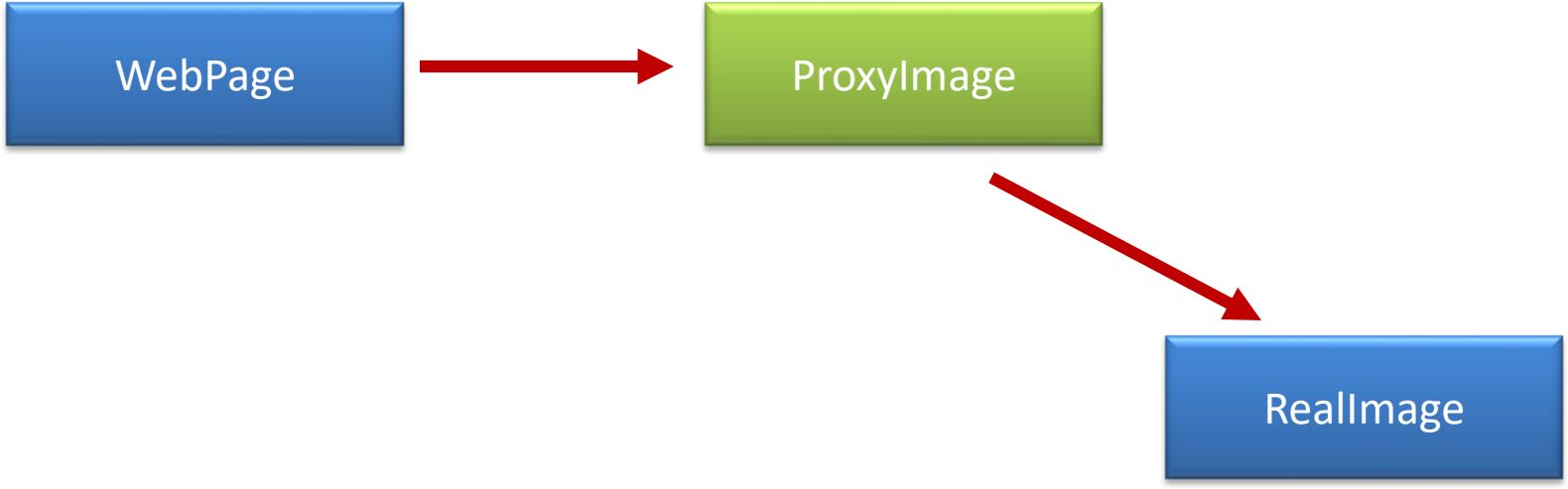
① *Program to an interface.* By insisting that images are only accessed from the client via an interface I can provide it with an intermediate object that will defer the loading until the `show()` method is called but in all other respects acts just like a real image object.

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.

AARHUS UNIVERSITET

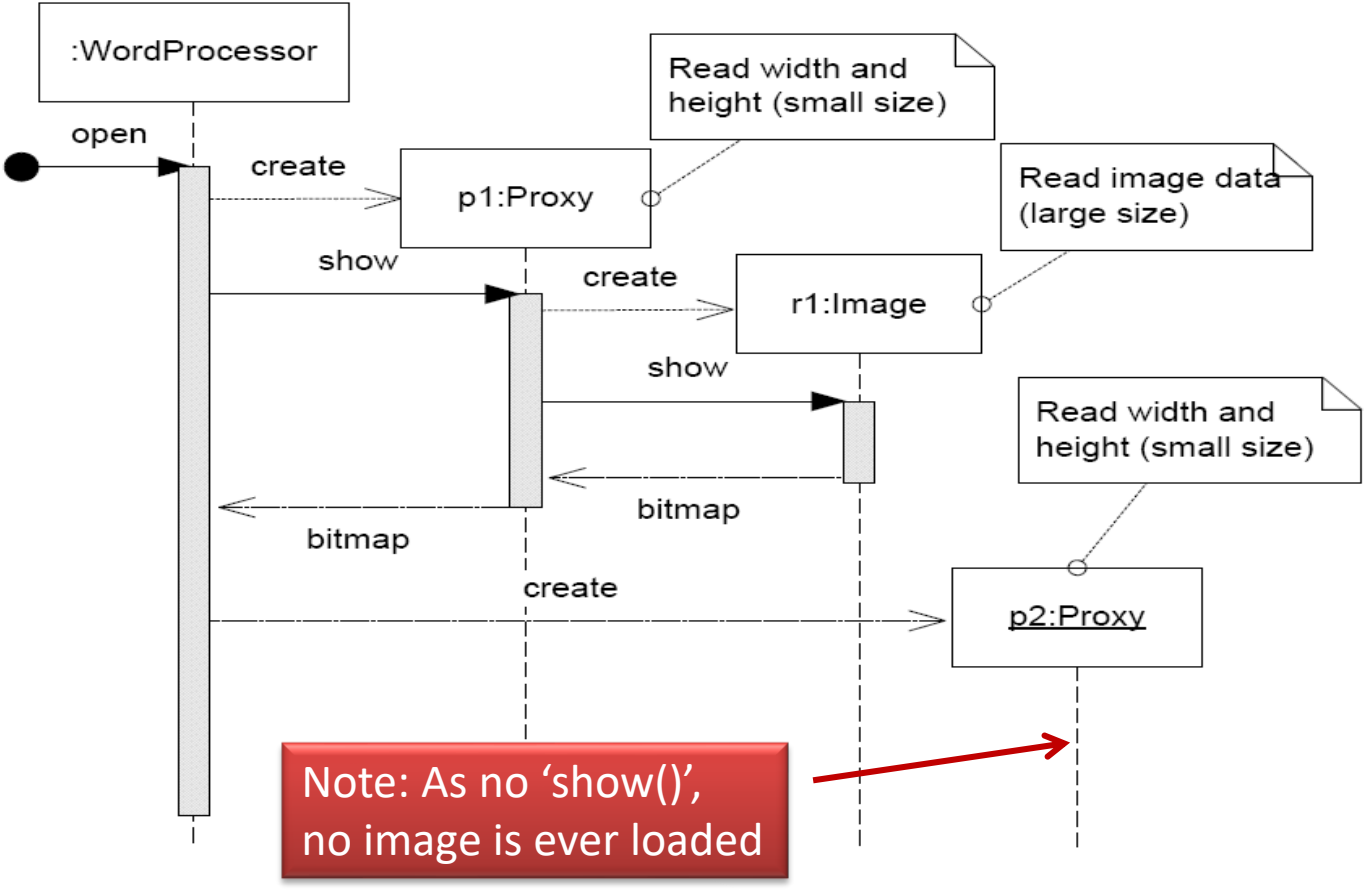
In-Front-Object

- WebPage loads a *proxy* with the filename of the image
  - Same interface as RealImage, but load method *does not load anything, just remembers the name of the file*
  - When show() method is invoked *then the image is loaded*



AARHUS UNIVERSITET

Dynamics



AARHUS UNIVERSITET

The load() method is slow

```
interface Image {
  public void load(String filename);
  /** show the image (here return a description string) */
  public String show();
}

/** The real subject: a jpg image */
class JPGImage implements Image {
  private String filename;
  public void load(String filename) {
    System.out.println( "*** JPGImage reading from file:"+filename+" ***");
    this.filename = filename;
  }
  public String show() {
    return "Showing JPG image from file "+filename;
  }
}

/** The proxy */
class ProxyImage implements Image {
  private String filename;
  private Image realSubject;
  public void load(String filename) {
    // just remember the filename
    this.filename = filename;
    realSubject = null;
  }
  public String show() {
    if ( realSubject == null ) {
      realSubject = new JPGImage();
      realSubject.load(filename);
    }
    return realSubject.show();
  }
```

Simulating the 'slow loading' algorithm

Webpage calls load() for *all images* so wait time is *long!*

AARHUS UNIVERSITET

The Proxy defers the load!

```
interface Image {
  public void load(String filename);
  /** show the image (here return a description string) */
  public String show();
}

/** The real subject: a jpg image */
class JPGImage implements Image {
  private String filename;
  public void load(String filename) {
    System.out.println( "*** JPGImage reading from file:"+filename+" ***");
    this.filename = filename;
  }
  public String show() {
    return "Showing JPG image from file "+filename;
  }
}

/** The proxy */
class ProxyImage implements Image {
  private String filename;
  private Image realSubject;
  public void load(String filename) {
    // just remember the filename
    this.filename = filename;
    realSubject = null;
  }
  public String show() {
    if ( realSubject == null ) {
      realSubject = new JPGImage();
      realSubject.load(filename);
    }
    return realSubject.show();
  }
```

Now load() is not called until 'show()' is called! Images that are never shown are never loaded!

AARHUS UNIVERSITET

Demo

```
D:\proj\Book\src\chapter\proxy>java ProxyDemo
======= Demonstration of Proxy =======
--- Loading document (no image load) ---
--- Showing page 1 ---
This is the text of page 1, that has no images.
--- Showing page 2 (image load) ---
This is another text, on page 2, with a single image.
*** JPGImage reading from file:flower.jpg ***
Showing JPG image from file flower.jpg
--- Back to page 1 ---
This is the text of page 1, that has no images.
--- And again showing page 2 (no load)---
This is another text, on page 2, with a single image.
Showing JPG image from file flower.jpg
```

```
public class ProxyDemo {
  public static void main(String[] args) {
    System.out.println( "======= Demonstration of Proxy =======");
    WordProcessor wp = new WordProcessor();

    System.out.println( "--- Loading document (no image load) ---");
    wp.load("mydocument.doc");

    System.out.println( "--- Showing page 1 ---");
    wp.showPage(1);

    System.out.println( "--- Showing page 2 (image load) ---");
    wp.showPage(2);

    System.out.println( "--- Back to page 1 ---");
    wp.showPage(1);

    System.out.println( "--- And again showing page 2 (no load)---");
    wp.showPage(2);
  }
}

/** This class plays the client role for the subject */
class WordProcessor {
  private String textPage1, textPage2;
  private Image imagePage2;

  public void load(String filename) {
    // I ignore the file name :)
    textPage1 = "This is the text of page 1, that has no images.";
    textPage2 = "This is another text, on page 2, with a single image.";
    imagePage2 = new ProxyImage();
    imagePage2.load("flower.jpg");
  }

  public void showPage(int number) {
    if ( number == 1 ) {
      System.out.println( textPage1 );
    } else if ( number == 2 ) {
      System.out.println( textPage2 );
      System.out.println( imagePage2.show() );

    }
  }
}
```

AARHUS UNIVERSITET

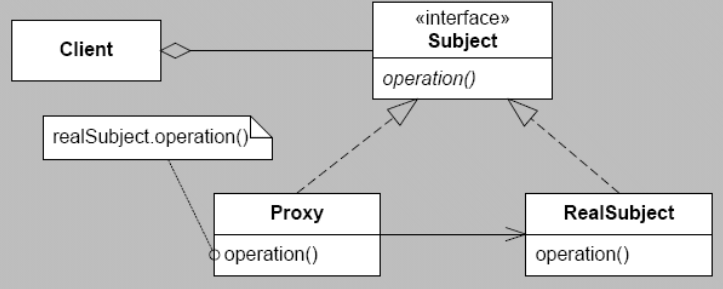
[25.1] Design Pattern: Proxy

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 diffent types of housekeeping when clients access the object, like logging, 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:



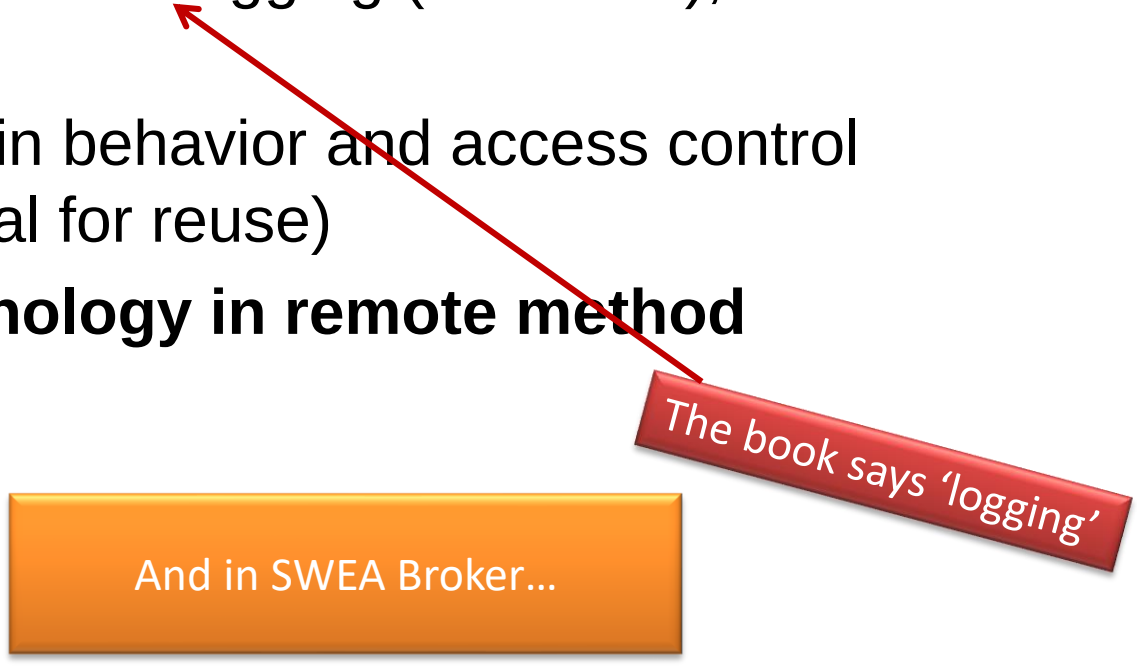
Roles A **Client** only interacts via a **Subject** interface. The **RealSubject** is the true object implementing resource-demanding operations (bandwidth, computation, memory usage, etc.) or operations that need access control (security, pay-by-access, logging, etc.). A **Proxy** implements the **Subject**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

AARHUS UNIVERSITET

Consequences

- Benefits
  - Supports all types of access control: deferred access, pay-by-access, access logging (audit trail), access control, …
  - Decouples domain behavior and access control behavior (potential for reuse)
  - **Is the core technology in remote method invocation**
    - Java RMI
    - .Net Remoting



AARHUS UNIVERSITET

Decorator Versus Proxy

- The look very alike!
  - Indeed I almost always code Decorators without the abstract class shown in the UML for Decorator
    - Thus it looks structurally much more like proxy

- *But they look at different things*
  - *Decorator looks at the object itself (looking **inside**)*
    - Adding behaviour to the object itself
  - *Proxy looks at the user of the object (looking **outside**)*
    - Monitoring/controlling who is *using* the object