Laboratorul 9

Şablonul de proiectare Proxy

1. Obiectiv

Obiectivul laboratorului 9 este implementarea unui program după şablonul de proiectare structural *Proxy* (o traducere posibilă ar fi „Intermediar” sau „Delegare”). Vom considera varianta *proxy*-ului de protecţie pentru stabilirea drepturilor de acces la nişte documente protejate. Tema pentru acasă completează şablonul cu varianta *proxy*-ului la distanţă, pentru accesarea distribuită a documentelor.

2. Scop şi motivaţie

Şablonul *Proxy* asigură un înlocuitor pentru alt obiect pentru a controla accesul la acesta.

Unul din motivele pentru controlarea accesului la un obiect este întârzierea creării şi iniţializării lui până în momentul în care acesta trebuie utilizat. De exemplu, să considerăm un program de editare de documente care poate conţine obiecte grafice. Crearea unor obiecte-imagine raster de mari dimensiuni poate fi foarte costisitoare din punct de vedere al timpului necesar citirii fişierelor de pe harddisk, mai ales dacă presupunem că sunt stocate în format *bmp*. Deschiderea unui document trebuie să fie însă o operaţie rapidă, deci crearea tuturor acestor obiecte costisitoare odată cu deschiderea documentului trebuie evitată. Acest lucru nici nu este necesar, deoarece nu toate imaginile vor fi vizibile în acelaşi timp.

O soluţie este crearea *la cerere* a fiecărui obiect costisitor, în cazul nostru acest lucru având loc în momentul în care o imagine devine vizibilă.

Se pune problema ce vom plasa în document în locul imaginii reale şi cum putem ascunde faptul că imaginea este creată la cerere, astfel încât să nu complicăm implementarea editorului iar optimizarea să nu afecteze codul de formatare şi redare a imaginilor.

Se poate utiliza un alt obiect, un *proxy* de imagine, care să acţioneze ca înlocuitor al imaginii reale. Obiectul *proxy* acţionează la fel ca imaginea şi o instanţiază când este nevoie, adică doar atunci când programul de editare îi cere să o afişeze. Cererile ulterioare sunt transmise direct către imaginea reală şi deci *proxy*-ul trebuie să aibă o referinţă către aceasta (figura 1).

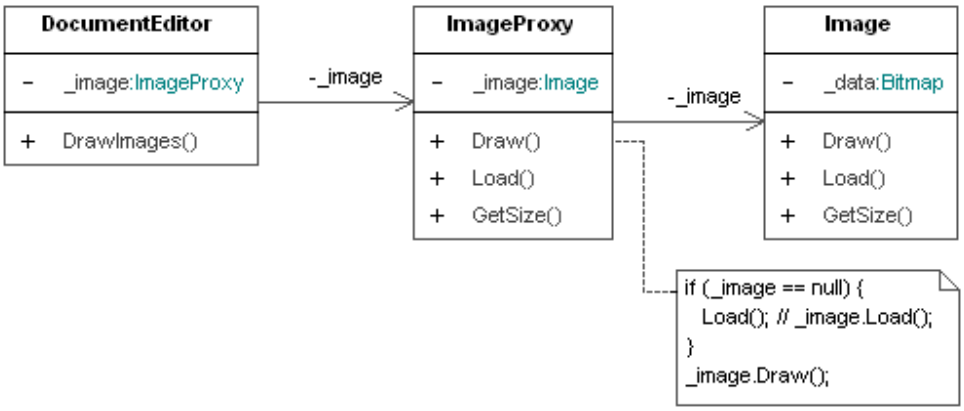

Figura 1. *Diagrama de clase a unui proxy de imagine*

Referința la obiectul _image din ImageProxy va fi nulă până în momentul în care editorul apelează metoda Draw și obiectul *proxy* instanțiază imaginea reală. Operația Draw asigură faptul că imaginea este instanțiată înainte de a fi apelat obiectul Image.

Până atunci, dacă mai există o metodă GetSize care returnează dimensiunile imaginii, obiectul ImageProxy poate returna o valoare proprie implicită. După instanțierea imaginii raster, obiectul ImageProxy va returna dimensiunile reale ale imaginii apelând metoda GetSize din Image.

3. Aplicabilitate

Șablonul *Proxy* poate fi aplicat oriunde este nevoie de o referință mai sofisticată la un obiect. Sunt prezentate în continuare câteva situații în care se poate aplica acest șablon:

- ***Proxy*-ul la distanță** (engl. "remote proxy") reprezintă un obiect local care intermediază accesul la un obiect de pe o altă mașină. Este util în general pentru aplicațiile distribuite;
- ***Proxy*-ul virtual** (engl. "virtual proxy") creează la cerere obiecte costisitoare. Un exemplu este clasa ImageProxy din secțiunea anterioară. Imaginea reală poate fi însă și o imagine descărcată de pe internet; în acest caz *proxy*-ul poate afișa o imagine temporară până când este descărcată imaginea reală. Șablonul nu se referă doar la imagini, ci la orice obiect a cărui creare necesită timp sau resurse importante;
- ***Proxy*-ul de protecție** (engl. "protection proxy") controlează accesul la obiectul real. Este util atunci când obiectul reprezentat presupune drepturi diferite de acces;
- **Referința inteligentă** (engl. "smart reference") este un înlocuitor pentru o referință, care efectuează acțiuni suplimentare în momentul accesării unui obiect. De exemplu, se poate folosi pentru păstrarea cópiilor unor obiecte mari care pot sau nu să se modifice. Dacă se dorește crearea unei alte instanțe de acest tip, *proxy*-ul poate decide să nu facă efectiv copia, folosind în continuare primul obiect. Dacă clientul încearcă să facă modificări în al doilea obiect, *proxy*-ul face abia atunci copia și modificările. De asemenea, se poate stabili astfel numărul de referințe către obiectul real, astfel încât acesta să poată fi automat eliberat atunci când nu mai există referințe.

4. Analiza şablonului

Diagrama generică de clase a şablonului *Proxy* este prezentată în figura 2.

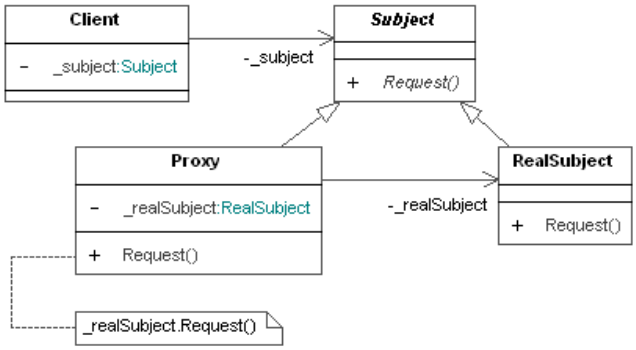

Figura 2. *Diagrama de clase a şablonului Proxy*

Trebuie precizat că atât clasa Proxy cât şi clasa RealSubject au o interfaţă identică, definită de clasa Subject, astfel încât un obiect *proxy* să poată fi înlocuit de obiectul „real”. În funcţie de tipul său, obiectul Proxy trimite apelurile către obiectul RealSubject.

5. Exemplu de implementare

Codul C# corespunzător structurii şablonului *Proxy* este prezentat mai jos.

Subiectul abstract

```
abstract class Subject
{
  // Metode
  abstract public void Request();
}
```

Subiectul real

```
class RealSubject : Subject
{
  // Metode
  override public void Request()
  {
    Console.WriteLine("Apelul din subiectul real");
  }
}
```

Proxy

```
class Proxy : Subject
{
   RealSubject _realSubject;

   override public void Request()
   {
      // Se poate folosi inițializarea întârziată
      if (_realSubject == null)
         _realSubject = new RealSubject();

      _realSubject.Request();
   }
}
```

Clientul

```
public class Client
{
   public static void Main(string[] args)
   {
      // Se creează un proxy și se apelează metoda dorită
      Proxy p = new Proxy();
      p.Request();
   }
}
```

6. Aplicații

6.1. Realizați un program pentru accesul la o serie de documente potrivit nivelului de acces al utilizatorului. Un schelet al aplicației pentru construirea interfeței grafice cu utilizatorul este prezentată în figura 3.

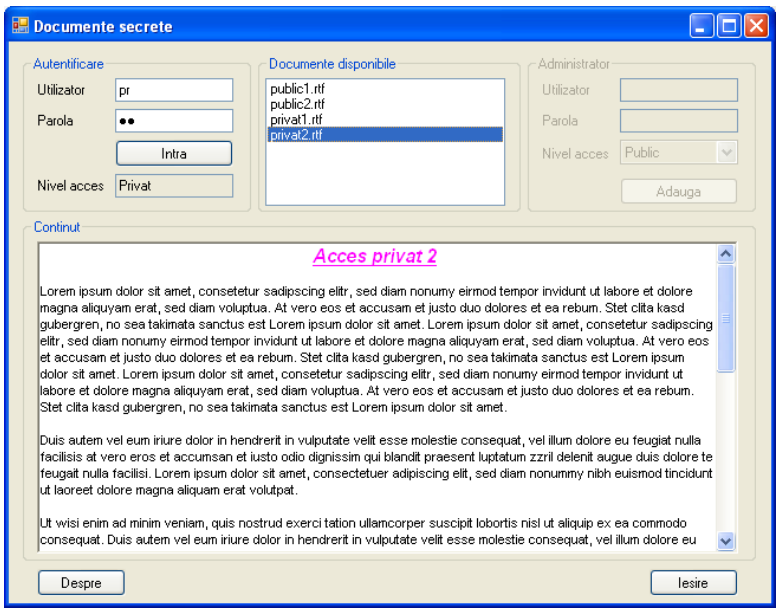

Figura 3. *Exemplu de rezolvare: interfața cu utilizatorul*

Structura propusă de fișiere este prezentată în figura 4.

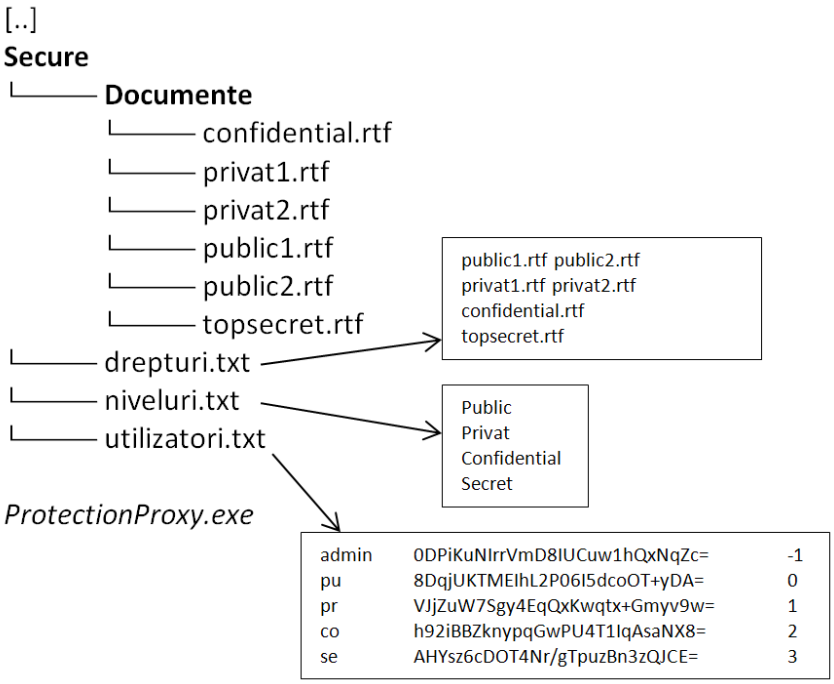

Figura 4. *Structura de fișiere a aplicației*

Nivelurile de acces se dau în fișierul *niveluri.txt*:

```
Public
Privat
Confidential
Secret
```

Lista de utilizatori se găsește în fișierul *utilizatori.txt*, care este de forma specificată în tabelul 1.

Tabelul 1. *Exemplu de listă de utilizatori*

Numele utilizatorului	Hash-ul parolei	Nivelul de acces
admin	0DPiKuNIrrVmD8IUCuw1hQxNqZc=	-1
pu	8DqjUKTMEIhL2P06I5dcoOT+yDA=	0
pr	VJjZuW7Sgy4EqQxKwqtx+Gmyv9w=	1
se	AHYsz6cDOT4Nr/gTpuzBn3zQJCE=	3
co	h92iBBZknypqGwPU4T1IqAsaNX8=	2

Pe prima coloană este numele utilizatorului. Pe coloana a doua este *hash*-ul parolei, pentru evitarea stocării în clar a acesteia. La autentificare, se va calcula *hash*-ul parolei și se va compara cu *hash*-ul din fișier. Pe a treia coloană se notează nivelul de acces. Administratorul este un tip special de utilizator. Acesta nu poate consulta documente, dar poate adăuga utilizatori. Inițial, numele administratorului este *admin* și parola este tot *admin*. În situația de mai sus, parolele utilizatorilor sunt egale cu numele lor. În program, la autentificarea unui utilizator se verifică existența utilizatorului și corectitudinea parolei.

Documentele din directorul *Documente* au asociate și ele drepturi de acces, setate în fișierul *drepturi.txt*:

```
public1.rtf public2.rtf
privat1.rtf privat2.rtf
confidential.rtf
topsecret.rtf
```

Pe prima linie sunt documentele corespunzătoare primului nivelul de acces (*Public*), pe linia a doua documentele pentru nivelul *Privat* ș.a.m.d. Când un utilizator se autentifică, el primește lista documentelor de pe nivelul său de acces, dar și documentele de pe toate nivelurile inferioare. Se presupune că serverul de documente ar fi pe altă mașină. La selectarea unui document, acesta este criptat și trimis clientului care îl decriptează și îl afișează. Accesul la documente se va face printr-un *proxy* de protecție. Acesta la rândul său accesează managerul real de documente. Clasele ProxyDocumentManager și RealDocumentManager vor implementa ambele interfața IDocumentManager.

Diagrama de clase a aplicației este prezentată în figura 5.

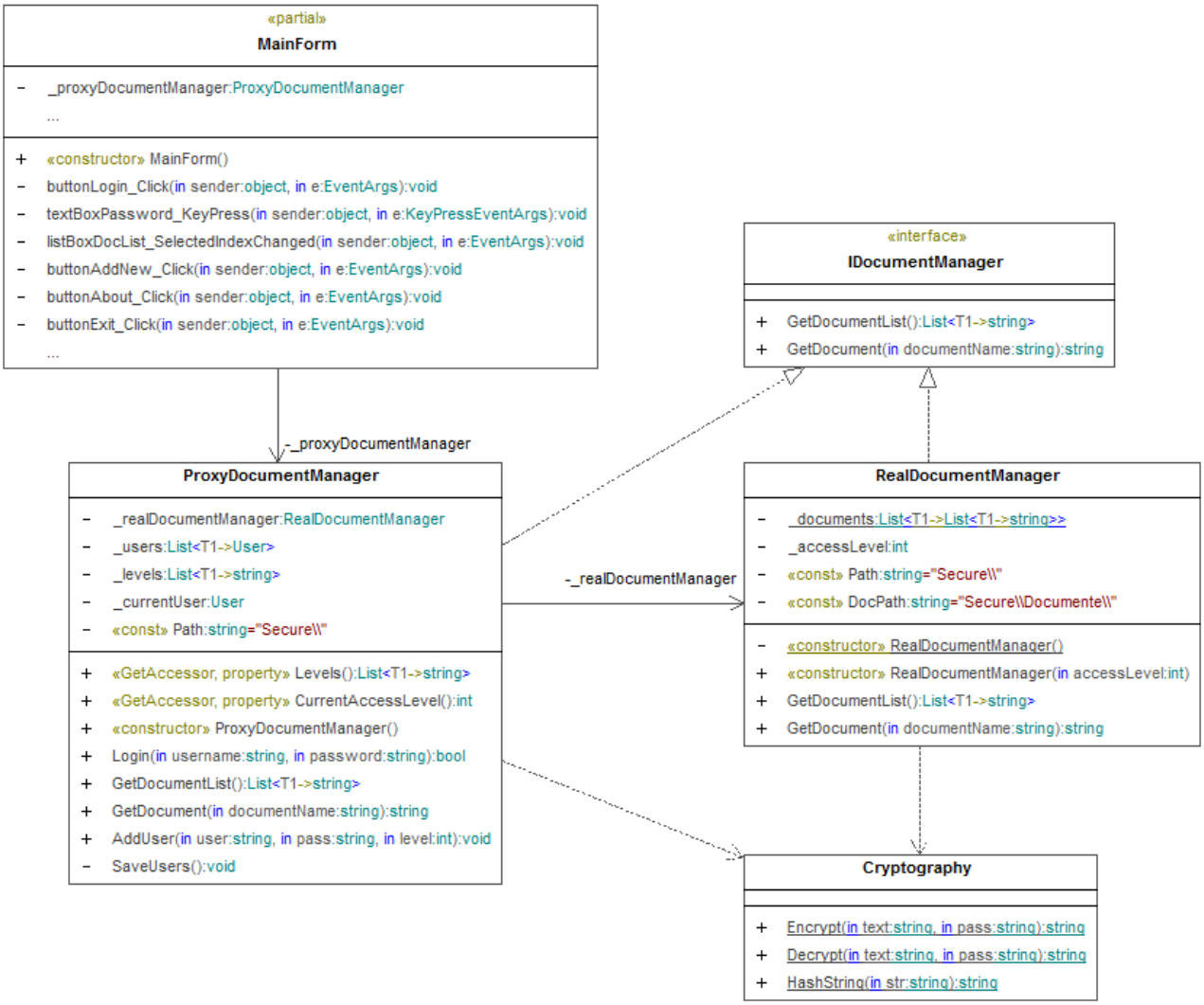

Figura 5. *Exemplu de rezolvare: diagrama de clase*

ProxyDocumentManager se ocupă de autentificare și gestionarea drepturilor de acces. Metodele GetDocumentList și GetDocument apelează metodele corespunzătoare din RealDocumentManager. RealDocumentManager criptează documentul și îl trimite către ProxyDocumentManager. Această operație este importantă mai ales atunci când ProxyDocumentManager și RealDocumentManager se află pe mașini diferite (situația de la aplicația opțională 6.2).

Parola pentru criptarea unui documentului transmis se poate considera cunoscută de către ProxyDocumentManager și RealDocumentManager. Stabilirea în mod dinamic a unei parole comune între două entități care comunică printr-un canal public este o problemă în sine și nu poate fi tratată aici. De asemenea, parola folosită pentru criptarea documentelor este diferită de parola utilizatorului și nici nu trebuie să fie cunoscută de către acesta.

Constructorul static al clasei RealDocumentManager asigură faptul că listele cu drepturile de acces vor fi inițializate o singură dată înainte de utilizarea efectivă a clasei de către ProxyDocumentManager. Constructorul static este apelat automat înainte de a fi creată prima instanță a clasei sau înainte de referențierea membrilor statici.

6.2. (opțional) Extindeți programul astfel încât documentele și managerul de documente să poată fi pe un alt calculator iar *proxy*-ul să înglobeze și funcționalități de delegare la distanță. Pentru eficientizarea comunicațiilor, arhivați documentul trimis *înainte* de criptare.