Problema liftului – Studiu de caz

Scopul laboratorului – utilizarea **UML (Unified Modeling Language)** drept mediu de dezvoltare software.

UML este un limbaj de modelare bazat pe notații grafice folosit pentru a *specifica*, *vizualiza*, *construi* şi *documenta* componentele unui program.

Formularea problemei:

Se cere dezvoltarea unui produs software care să controleze funcționarea a ***n*** lifturi într-o clădire de ***m*** etaje. Produsul trebuie să asigure deplasarea lifturilor între etaje, ținând cont de următoarele restricții:

- Fiecare lift are un set de ***m*** butoane, corespunzătoare fiecărui etaj. Fiecare buton se „aprinde" când este apăsat şi determină deplasarea liftului la etajul corespunzător. Butonul se „stinge" când liftul ajunge la etajul respectiv.
- Fiecare etaj, cu excepția parterului şi a ultimului etaj, are 2 butoane: unul care determină urcarea liftului şi unul care determină coborârea acestuia. Aceste butoane se „aprind" când sunt apăsate. Butoanele se „sting" când un lift ajunge la etajul de unde se face cererea.
- Atunci când un lift nu este solicitat, el rămâne la etajul curent cu uşile închise

Etape

- Analiza orientată obiect (OOA)
  - modelarea cazurilor de utilizare (Use case)
  - modelarea claselor
  - modelarea dinamică
- Proiectarea orientată obiect (OOD)
  - Diagrame de interacțiune
  - Diagrame detaliate de clase

Paradigma orientată obiect

În acest moment putem da o definiție a unui obiect, care constă, acesta constând în:

- *Date* – atribute, variabile de stare, variabile instanțe, câmpuri, membri;
- *Acțiuni* – metode sau funcții membre.

Încapsulând datele şi acțiunile, obiectele pot fi privite ca unități independente, având atât o independență conceptuală cât şi fizică.

OOA constă în următorii paşi:

1. Modelarea Use Case (cazuri de utilizare)

- Determină cum se pot obține anumite rezultate folosind produsul ce trebuie creat, fără a interesa ordinea.
- Prezintă informațiile sub forma diagramelor Use Case şi a scenariilor asociate.
- Use Case-ul descrie ce face un program sau subprogram dar nu precizează nimic despre cum este realizată o funcționalitate.

2. Modelarea claselor

- Determină clasele şi atributele lor
- Apoi se determină relaţiile şi interacţiunile între clase
- Prezintă informaţiile sub forma unei diagrame

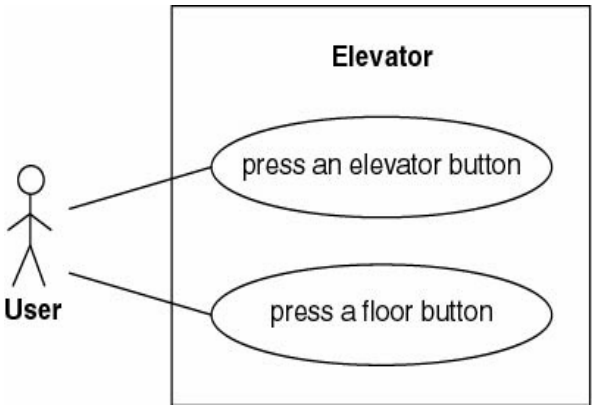
3. Modelarea dinamică

- Determină acţiunile făcute de fiecare clasă sau subclasă
- Prezintă informaţiile sub forma unei diagrame

În practică, cei trei paşi nu se fac exclusiv secvenţial. O schimbare într-o diagramă determină revizia celorlalte două. Astfel, cei trei paşi ai OOA se fac efectiv în paralel. Acest fapt are sens deoarece în paradigma OO, nici o dată sau acţiune nu are prioritate în faţa alteia. În cursul analizei, cunoştinţele acumulate despre produs se reprezintă în diferite moduri, fiecare reflectând un aspect diferit al produsului ţintă. Diagramele se actualizează continuu, pe măsură ce se obţin mai multe percepţii asupra modelării sistemului La sfârşitul fazei OOA, modurile diferite de abordare vor oferi o înţelegere generală a produsului care ar fi greu de obţinut dacă se foloseşte doar o tehnică de modelare

1. Modelarea Use Case

Un Use Case descrie funcţionalitatea produsului ce trebuie construit. Se furnizează astfel o descriere generică a funcţionării generale.



Scenariile sunt instanţe ale Use Case, la fel cum obiectele sunt instanţe ale claselor.

Un Use Case descrie interacţiunile generale dintre clasele produsului software şi utilizatorii produsului

Un *Scenariu* este un set particular de interacţiuni între obiecte specifice şi utilizatori. Nu vom discuta acum despre scenarii deoarece acestea sunt utilizate în special pentru OOD. Vom menţiona doar faptul că în UML există două tipuri de scenarii: *diagramele de secvenţe* şi *diagramele de colaborare*.

Exemplu de scenariu normal

1. Utilizatorul (user) A apasă butonul de urcare de la etajul 3 pentru a chema un lift. Utilizatorul A doreşte să meargă la etajul 7.
2. Butonul de urcare de pe etaj se aprinde.
3. Un lift soseşte la etajul 3. În lift se află utilizatorul B, care a urcat în lift la etajul 1 şi a apăsat butonul din lift corespunzător etajului 9.
4. Butonul de urcare de pe etaj se stinge.
5. Se deschid uşile liftului.
6. Se porneşte cronometrul.
   Utilizatorul A intră în lift.
7. Utilizatorul A apasă butonul din lift corespunzător etajului 7.
8. Butonul liftului corespunzător etajului 7 se aprinde.
9. Uşile liftului se închid după expirarea timpului de staționare pe etaj.
10. Liftul se mişcă spre etajul 7.
11. Butonul liftului corespunzător etajului 7 se stinge.
12. Uşile liftului se deschid pentru a permite utilizatorului A să iasă din lift.
13. Se porneşte cronometrul.
    Utilizatorul A iese din lift.
14. Uşile liftului se închid după expirarea timpului de staționare pe etaj.
15. Liftul urcă spre etajul 9 cu utilizatorul B.

Scenariul de mai sus reprezintă un set de interacțiuni între utilizatori şi lifturi şi corespunde modului în care înțelegem că ar trebui folosite lifturile.

Cele 15 evenimente descriu în detaliu cele două interacțiuni dintre Utilizatorul A şi butoanele sistemului liftului (evenimentele 1 şi 7) şi acțiunile componentelor sistemului liftului (evenimentele de la 2 la 6 şi de la 8 la 15). Două acțiuni nu sunt numerotate, deoarece Utilizatorul A nu interacționează cu componentele liftului când intră sau iese din lift.

În contrast cu scenariul normal se poate descrie şi un ***scenariu anormal (excepție):***

1. Utilizatorul A apasă butonul de urcare de la etajul 3 pentru a chema un lift. Utilizatorul A vrea să meargă la etajul 1.
2. Butonul de urcare de pe etaj se aprinde.
3. Un lift soseşte la etajul 3. În el se găseşte Utilizatorul B, care a intrat în lift la etajul 1 şi a apăsat butonul corespunzător etajului 9.
4. Butonul de urcare de pe etaj se stinge.
5. Se deschid uşile liftului.
6. Porneşte cronometrul.
   Utilizatorul A intră în lift
7. Utilizatorul A apasă butonul din lift corespunzător etajului 1.
8. Se aprinde butonul din lift corespunzător butonului 1.
9. Uşile liftului se închid după expirarea timpului de staționare pe etaj.
10. Liftul urcă la etajul 9.
11. Butonul liftului corespunzător etajului 9 se închide.
12. Uşile se deschid pentru a permite utilizatorului B să iasă din lift.
13. Cronometrul porneşte.
    Utilizatorul B iese din lift.
14. Uşile liftului se închid după expirarea timpului de staționare pe etaj.
15. Liftul coboară la etajul 1 cu Utilizatorul A.

Utilizatorul A apasă butonul de pe etaj greşit.

Observații:

În domenii mai puțin familiare decât lifturile, scenariile derivă din documentele cerințelor.

Trebuie studiate scenarii suficiente pentru a da echipei OOA o privire cuprinzătoare asupra comportării sistemului ce trebuie modelat

Aceste informații sunt utilizate în faza următoare, modelarea claselor, pentru a determina clasele (obiectele). De asemenea sunt folosite în OOD.

2. Modelarea claselor

În acest pas, clasele şi atributele lor se extrag şi se reprezintă folosind o diagramă entitate-relație.

Se determină doar atributele clasei, nu şi metodele (acestea se vor determina în faza OOD).

Este dificil de a reuşi extragerea claselor şi a atributelor din primul moment.

O metodă de a determina clasele este de a le deduce din Use Case-uri şi din scenariile asociate acestora, prin identificarea componentelor care joacă un rol în Use Case-uri. Adesea există multe scenarii şi există pericolul posibil de a considera prea multe clase candidate.

Este mai simplu să se adauge o nouă clasă la model decât de a şterge o clasă care nu trebuia inclusă.

Există două abordări pentru modelarea claselor:

1. **Extragerea substantivelor** – dacă dezvoltatorul are puțină experiență sau nu are deloc în domeniul aplicației
2. **Cardurile CRC** - dacă dezvoltatorul are experiență în domeniu.
   (CRC = Class Responsibility Collaboration)

Extragerea substantivelor

Pentru dezvoltatorii fără experiență în domeniul aplicației, o metodă bună este de a folosi un proces cu trei etape pentru a extrage clasele candidate şi apoi pentru a rafina soluția.

Et. 1 Definirea concisă a problemei

Se defineşte produsul cât mai concis posibil, de preferat într-o singură propoziție.

Ex. *Butoanele din lifturi şi de pe etaje controlează mişcarea a n lifturi într-o clădire cu m etaje.*

Et. 2 Strategia informală

Se introduc constrângerile, exprimând rezultatele într-un singur paragraf (de preferat).

Ex. ***Butoanele din lifturi şi de pe etaje controlează mişcarea a n lifturi într-o clădire cu m etaje****. Butoanele se aprind când sunt apăsate pentru a cere un lift la un anumit etaj. Butoanele se sting când cererea este satisfăcută. Când un lift nu are nici o cerere, rămâne la etajul curent cu uşile închise.*

Et. 3 Formalizarea strategiei

Se identifică substantivele din strategia informală, excluzându-le pe acelea care depăşesc domeniul problemei.

Se folosesc substantivele drept clase candidate.

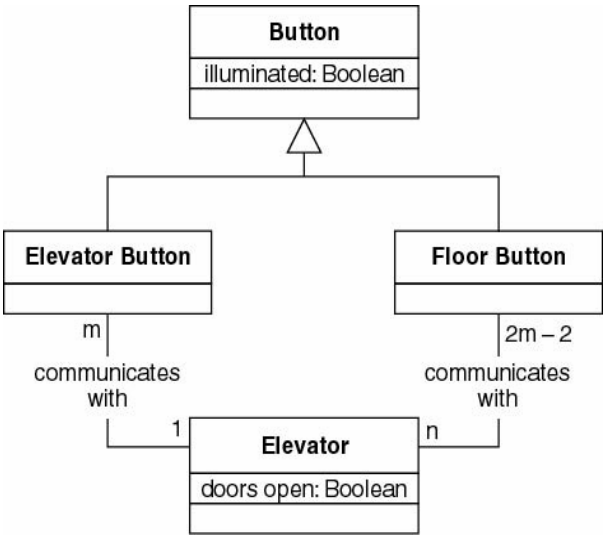
O regulă uzuală este de a utiliza substantivele rare pentru a defini clasele în timp ce cele frecvente sunt atribute ale claselor (de ex. iluminare este atribut al butonului)

Substantive: buton, lift, etaj, mişcare, clădire, iluminare, cerere, uşă

- Etaj, clădire, uşă sunt în afara problemei, deci se exclud.
- Mişcare, iluminare, cerere sunt substantive abstracte, deci se exclud (pot deveni atribute)

Clase candidate: (Lift) **Elevator** şi (Buton) **Button**.
Subclase: (Buton Lift) **Elevator Button** şi (Buton Etaj) **Floor Button**.

Prima iteraţie a *Diagramei de Clase*



Atributul **illuminated** modelează evenimentele 2, 4, 8 şi 11.

Problema specifică două tipuri de butoane, deci se definesc două subclase ale clasei **Button** şi anume **Elevator Button** şi **Floor Button** => moştenire.

Atributul **doors open** modelează evenimentele 5, 9, 12, 14 din scenarii.

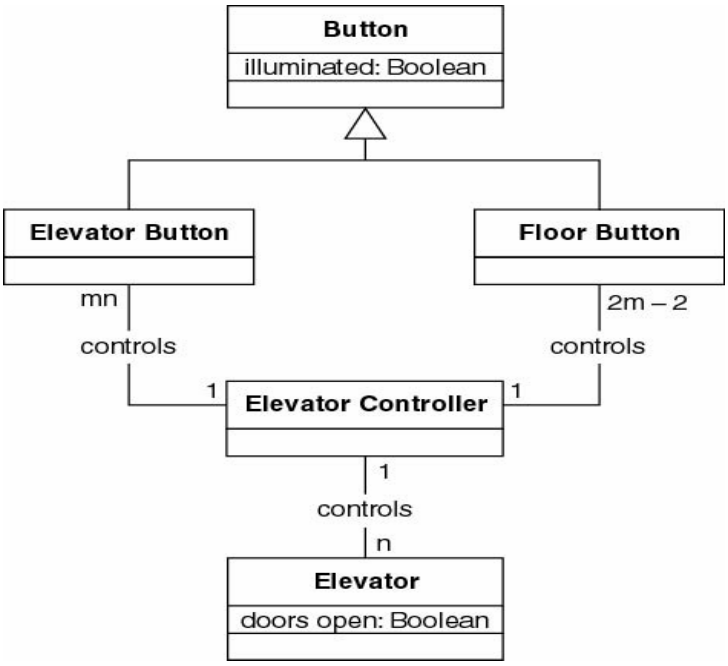
Fiecare din **Elevator Button** şi **Floor Button** comunică cu **Elevator.**

Din păcate începutul nu este bun. Într-un lift real, butoanele nu au o comunicare directă cu lifturile. Este nevoie de o nouă clasă **Elevator Controller**.

Totuşi, expunerea problemei nu menţionează controlerul, deci nu poate fi selectat drept clasă în procesul de extragere a substantivelor.

Tehnica de găsire a claselor candidate oferă un punct de start dar cu siguranţă nu face mai mult decât atât.

A doua iterație pentru *Diagrama de clase*



Dacă folosim doar relații 1-la-n proiectarea şi implementarea sunt mai uşoare. De aceea este indicată reducerea, pe cât este posibil, a relațiilor "mai multe" – la - „mai multe" la relații 1-la-n.

Cardurile CRC

Pentru fiecare clasă, echipa de dezvoltare software completează în card numele clasei, funcționalitatea sa (responsabilitatea) şi o listă a celorlalte clase ce permit obținerea acestei funcționalități (colaborarea).

În acest moment, întregul proces poate fi automatizat; Uneltele CASE, cum ar fi "System Architect" include module pentru crearea şi actualizarea cardurilor CRC.

Puterea cardurilor CRC este aceea că, atunci când sunt utilizate de o echipă, interacțiunea între membrii echipei pune în evidență lipsa sau completarea incorectă a câmpurilor clasei, cât şi a atributelor şi metodelor.

De asemenea se clarifică relațiile între clase când se folosesc cardurile CRC..

O slăbiciune a cardurilor CRC este aceea că această abordare generală nu este cea mai bună cale de a identifica clasele decât dacă membrii echipei au o experiență considerabilă în domeniul aplicației.

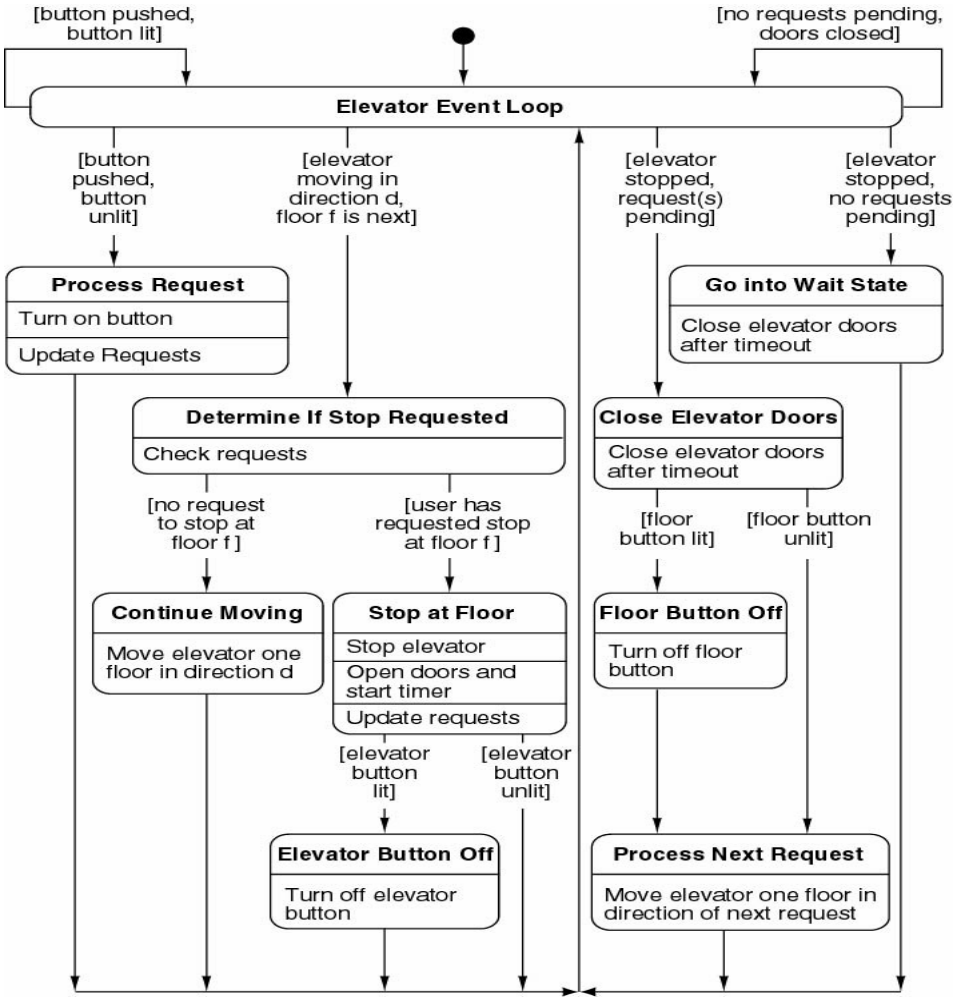
Pe de altă parte, odată ce dezvoltatorii determină majoritatea claselor şi au o idee despre responsabilități şi colaborări, cardurile CRC pot fi un mod excelent de a completa procesul şi de a fi siguri că totul este corect.

3. Modelarea dinamică

Scopul modelării dinamice este de a realiza *Diagrama de stări,* care este o descriere a produsului țintă, similară cu un automat cu stări finite, pentru fiecare clasă.

Reprezentarea diagramei de stări UML este mai puțin formală. Cele trei aspecte ale unui automat finit (stări, evenimente, predicate) sunt distribuite peste diagrama UML.

Predicate sau "gărzile" UML apar între paranteze



Testarea in timpul fazei OOA

Următorul pas este verificarea OOA.

O componentă a acestei verificări este utilizarea cardurilor CRC.

Fiecare card CRC este completat pentru fiecare clasă.

Cardul CRC pentru **Elevator Controller** se deduce din diagrama de clase şi din diagrama de stări.

Mai precis:

- RESPONSABILITATEA pentru **Elevator Controller** se obține listând toate acțiunile din diagrama de stări pentru **Elevator Controller**.
- COLABORAREA pentru **Elevator Controller** se determină examinând diagrama de clase şi notând care din clasele **Elevator Button**, **Floor Button** şi **Elevator** interacționează cu clasa **Elevator Controller**.

CLASS **Elevator Controller**
RESPONSIBILITY 1. Turn on elevator button 2. Turn off elevator button 3. Turn on floor button 4. Turn off floor button 5. Move elevator up one floor 6. Move elevator down one floor 7. Open elevator doors and start timer 8. Close elevator doors after timeout 9. Check requests 10. Update requests
COLLABORATION 1. Class **Elevator Button** 2. Class **Floor Button** 3. Class **Elevator**

Acest card CRC pune în evidență două probleme majore în prima iterație a OOA.

Să considerăm responsabilitatea:
1. *Turn on elevator button* (aprinde butonul liftului)

Această comandă este inacceptabilă din punctual de vedere al paradigmei orientate obiect (OOP). Din punct de vedere al polimorfismului, obiectele clasei **Elevator button** ar trebui să fie responsabile pentru aprinderea sau stingerea butoanelor. De asemenea se ignoră ascunderea informațiilor

Responsabilitatea corectă este:
1. Trimite mesaj către **Elevator Button** pentru a se aprinde singur.

A doua problemă este că o clasă a fost „supraîncărcată”.

Să considerăm responsabilitatea:

7. Deschide uşile liftului şi porneşte cronometrul.

Conceptul cheie aici este noțiunea de stare. Atributele unei clase sunt considerate adesea ca variabile de stare. Conceptul de stare joacă un rol important în OOP. Acest concept poate fi utilizat pentru a ne ajuta să determinăm dacă o componentă poate fi modelată ca o clasă. Dacă respectiva componentă are o stare care va fi schimbată în timpul implementării, atunci ea poate fi probabil modelată ca o clasă.

Deci **Elevator Doors** ar putea fi o clasă..

Un alt motiv este protejarea în fața schimbărilor neautorizate ale stării = considerente de siguranță.

Observație: După ce se face această schimbare, trebuie să reconsiderăm modelul dinamic şi modelul Use Case.

A doua iterație a cardului CRC

CLASS **Elevator Controller**
RESPONSIBILITY 1. Send message to **Elevator Button** to turn on button 2. Send message to **Elevator Button** to turn off button 3. Send message to **Floor Button** to turn on button 4. Send message to **Floor Button** to turn off button 5. Send message to **Elevator** to move up one floor 6. Send message to **Elevator** to move down one floor 7. Send message to **Elevator Doors** to open 8. Start timer 9. Send message to **Elevator Doors** to close after timeout 10. Check requests 11. Update requests
COLLABORATION 1. Subclass **Elevator Button** 2. Subclass **Floor Button** 3. Class **Elevator Doors** 4. Class **Elevator**

În completarea celor două probleme majore puse în evidență de cardul CRC, responsabilitățile **Check requests** şi **Update requests** necesită ca atributul **requests** să fie adăugat clasei **Elevator Controller.**

A treia iterație a Diagramei de clase

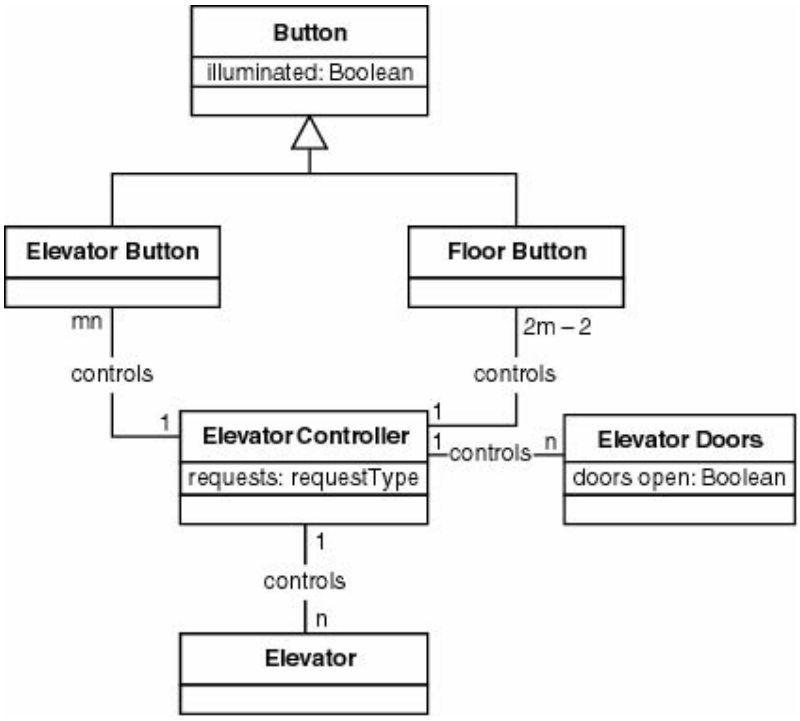
La acest pas, cererile sunt definite ca fiind de tipul **requestType** ; o structură de date pentru cereri va fi aleasă în timpul pasului de proiectare detaliată.

Modificând diagrama de clase trebuie să reexaminăm diagramele Use Case şi de stări pentru a vedea dacă este nevoie de viitoare rafinări.

Use-Case => Ok

Diagrama de stare => trebuie actualizată cu acțiunile care reflectă noile responsabilități şi extinsă pentru a include clasele adiționale.

.



A doua iterație a scenariului normal

1. Utilizatorul A apasă butonul de urcare de la etajul 3 pentru a chema un lift. Utilizatorul A doreşte să meargă la etajul 7.
2. Butonul etaj informează controlerul liftului că a fost apăsat
3. Controlerul liftului trimite un mesaj către butonul de urcare de pe etaj să se aprindă.
4. Controlerul liftului trimite o serie de mesaje către lift să se mişte în sus spre etajul 3. În lift se află utilizatorul B, care a luat liftul de la etajul 1 şi a apăsat butonul din lift corespunzător etajului 9.
5. Controlerul liftului trimite un mesaj către butonul de urcare de pe etaj să se stingă.
6. Controlerul liftului trimite un mesaj către uşile liftului să se deschidă.
7. Controlerul liftului porneşte cronometrul.
   Utilizatorul A intră în lift.
8. Utilizatorul A apasă butonul din lift corespunzător etajului 7.
9. Butonul liftului informează controlerul liftului că a fost apăsat.
10. Controlerul liftului trimite un mesaj către butonul liftului corespunzător etajului 7 să se aprindă.
11. Controlerul liftului trimite un mesaj către uşile liftului să se închidă după expirarea timpului de staționare pe etaj
12. Controlerul liftului trimite o serie de mesaje către lift să urce la etajul 7.
13. Controlerul liftului trimite un mesaj către butonul liftului corespunzător etajului 7 să se stingă
14. Controlerul liftului trimite un mesaj către uşile liftului să se deschide pentru a permite utilizatorului A să iasă din lift
15. Controlerul liftului porneşte cronometrul.
    Utilizatorul A iese din lift.
16. Controlerul liftului trimite un mesaj către uşile liftului să se închidă după expirarea timpului de staționare pe etaj.
17. Controlerul liftului trimite o serie de mesaje către lift să urce la etajul 9 cu utilizatorul B

În acest moment toate cele trei modele sunt gata. De fapt, ar fi mai bine să spunem că toate cele trei modele sunt gata pentru moment. S-ar putea să fie nevoie să revenim la analiză orientată obiect în timpul fazei de proiectare orientată obiect

Etapele proiectării orientate obiect (OOD)

OOD constă în 4 paşi:

1. Construirea diagramei de interacțiuni pentru fiecare scenariu
2. Construirea diagramei detaliate de clase
3. Proiectarea produsului în termenii clienților obiectelor
4. Trecerea la proiectarea detaliată

Etapa 1. Construirea diagramei de interacțiuni pentru fiecare scenariu

Se construieşte ***diagrama de secvențe***, care pune accentul pe aspectul temporal (ordonarea în timp a mesajelor).

Notația grafică este un tabel care are pe axa X obiecte, iar pe axa Z mesaje ordonate crescător în timp. Axa Z arată pentru fiecare obiect linia vieții (linie punctată verticală) şi perioada în care obiectul preia controlul execuției (reprezentată printr-un dreptunghi subțire pe linia vieții). În această perioadă obiectul efectuează o acțiune, direct sau prin intermediul procedurilor subordonate.

Se construieşte ***diagrama de colaborare***, care pune accentul pe organizarea structurală a obiectelor care participă la interacțiune. Ea conține obiecte, legături care se stabilesc între acestea precum şi mesajele prin care obiectele comunică. Mesajele sunt prefixate de un număr care indică ordonarea în timp a acestora şi sunt ataşate legăturilor dintre obiecte. Unei legături îi pot fi ataşate mai multe mesaje.

Notația grafică este un graf în care vârfurile reprezintă obiecte, iar arcele reprezintă legăturile dintre ele.

Se consideră scenariul normal dat mai sus, pentru care se construiesc diagramele de secvențe şi de colaborare următoare:

Diagrama de secvențe

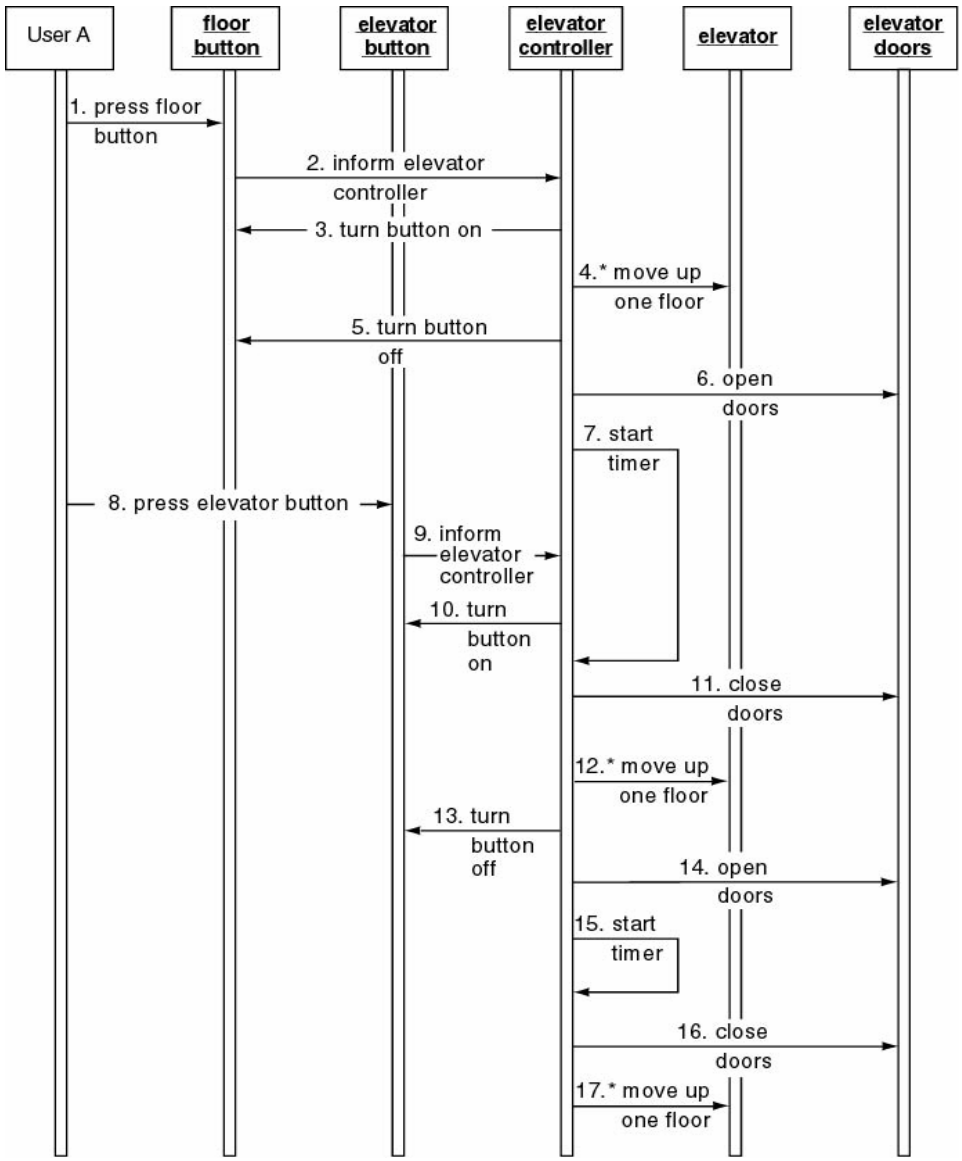
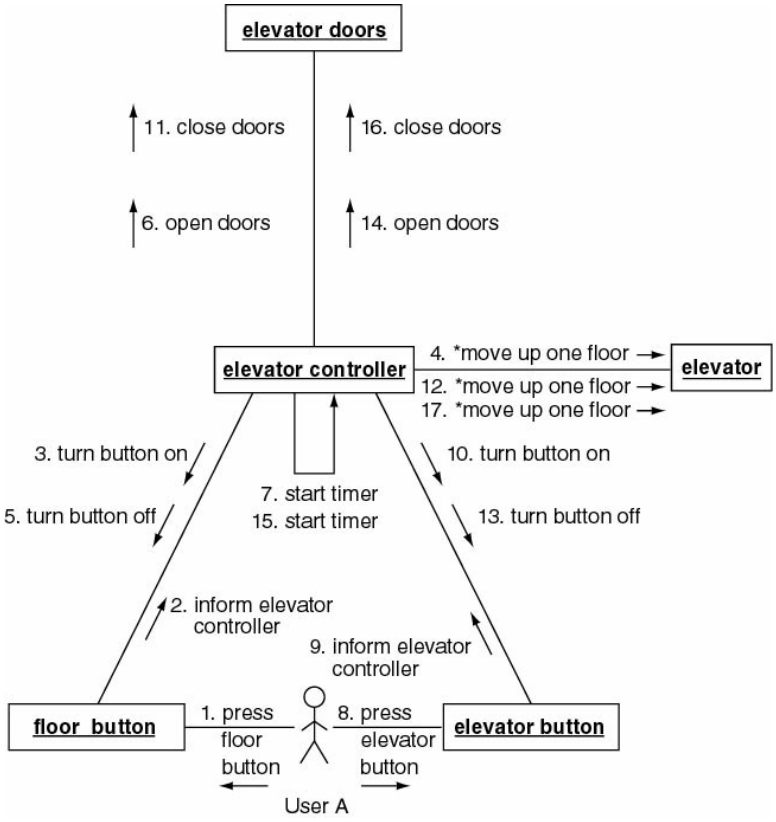


Diagrama de colaborare



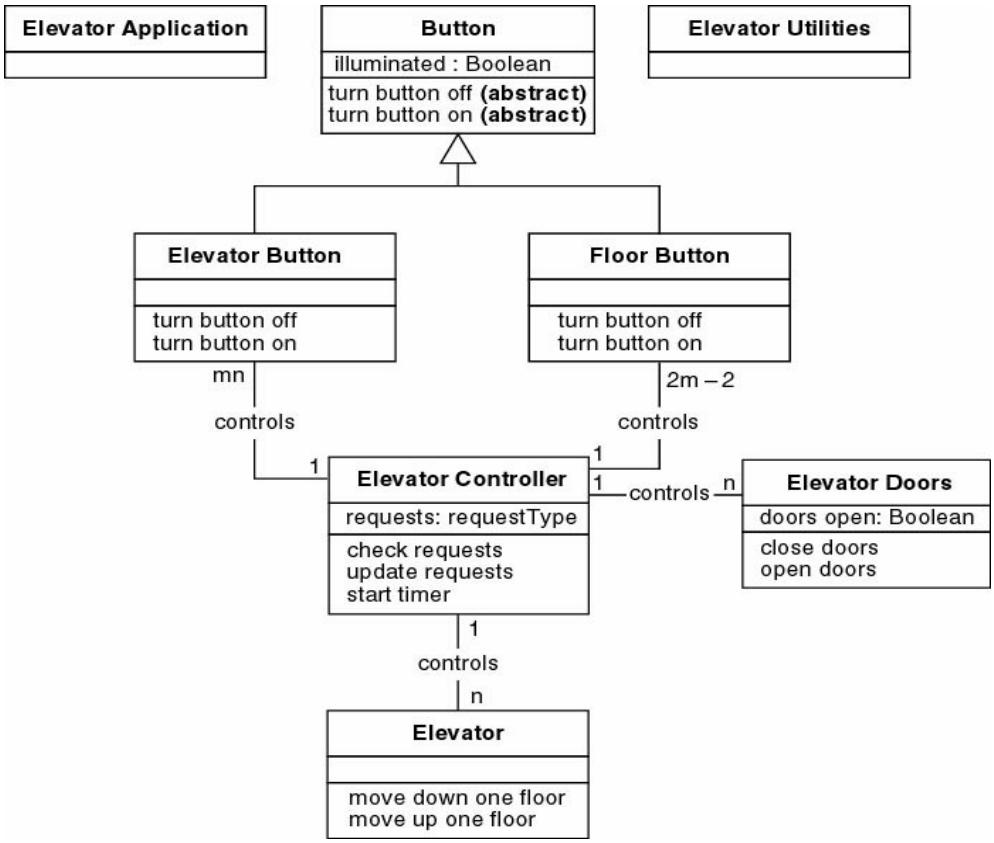
Etapa 2. Construirea diagramei detaliate de clase

Se pune problema atribuirii unei acţiuni unei clase sau unui client al acelei clase. Această atribuire se face în funcţie de următoarele criterii:

- ascunderea informaţiei
- reducerea numărului de copii ale acţiunii
- polimorfism

Ex.: (închide uşi) **close doors** este atribuită clasei **Elevator Doors**.
(coboară un etaj) **Move one floor down** este atribuită clasei **Elevator**.

Diagrama detaliată de clase

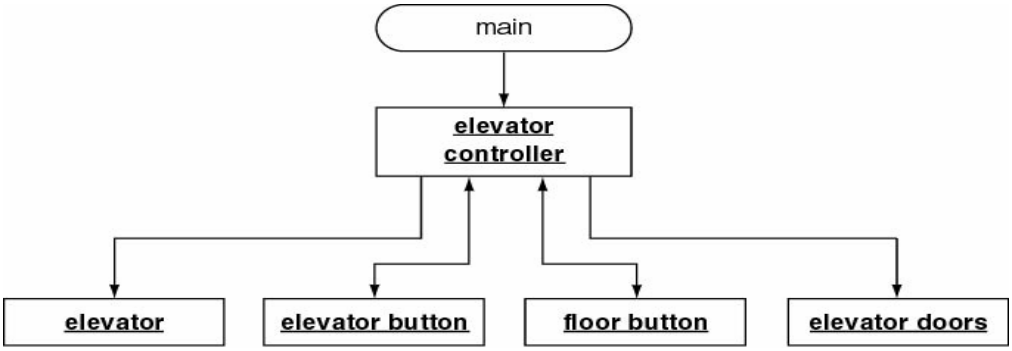


Etapa 3. Proiectarea produsului în termenii clienților obiectelor

Se desenează o săgeată de la un obiect la un client al acelui obiect. Obiectele care nu sunt clienți ale unor alte obiecte trebuie să fie instanțiate, probabil de metoda (funcția) **main**.

Sunt necesare metode adiționale: **Elevator Controler** are nevoie de metoda **elevator event loop** astfel încât **elevator application** să o poată apela.

Relațiile Client – Obiect (C++)



Etapa 4. Proiectarea detaliată

Se prezintă ca exemplu proiectarea detaliată a metodei

```
void elevator event loop (void)
{
  while (TRUE)
  {
    if (a button has been pressed)
      if (button is not on)
      {
        update requests;
        button::turn button on;
      }
    else if (elevator is moving up)
    {
      if (there is no request to stop at floor f)
        elevator::move one floor up;
      else
      {
        stop elevator by not sending a message to move;
        elevator doors::open doors;
        start timer;
        if (elevator button is on)
          elevator button::turn button off;
        update requests;
      }
    }
    else if (elevator is moving down)
      [similar to up case]
    else if (elevator is stopped and request is pending)
    {
      elevator doors::close doors;
      if (floor button is on)
        floor button::turn button off;
      determine direction of next request;
      elevator::move one floor up/down;
    }
    else if (elevator is at rest and not (request is pending))
      elevator doors::close doors;
    else
      there are no requests, elevator is stopped with elevator doors closed, so do nothing;
  }
}
```