SwRTc – Proiect

Procesul dezvoltarii unui sistem software orientat spre obiecte (I)

P.1. Utilizarea diagramelor UML ca suport pentru procesul de dezvoltare a unui sistem software (I)

Programarea sau **implementarea** (*scrierea codului, compilarea, executia si depanarea lui*) este activitatea centrala a oricarui proces de dezvoltare a unui sistem *software*. Fara aceasta activitate sistemul nu poate fi realizat. Insa alte etape, premergatoare sau ulterioare, sunt necesare pentru succesul proiectelor *software*, cu atat mai multe si mai ample cu cat complexitatea sistemului si riscul de esec al proiectului sunt mai mari.

Stabilirea arhitecturii, structurii, comportamentului si a algoritmilor utilizati este o etapa anterioara programarii, numita conceptie sau **proiectare**. In functie de complexitatea sistemului, aceasta poate merge de la o simpla organigrama sau un simplu pseudocod, pana la descrieri formale complexe (ce pot permite, daca exista instrumentele *software* adecvate, *generarea automata a codului*, totala sau partiala).

O sub-activitate initiala de **proiectare arhitecturala** (de ansamblu, de nivel inalt) poate fi necesara in cazul sistemelor *software* complexe, pentru a stabili elementele esentiale, structurale, ale viitoarei solutii. Ea va fi urmata de o sub-activitate de **proiectare a detaliilor**, care va tine seama mai mult de constrangerile de implementare.

In cazul sistemelor *software* mari, devin importante activitati anterioare proiectarii, cum ar fi **specificarea si analiza cerintelor** si **analiza domeniului problemei a posibilelor solutii**. Dezvoltarea sistemelor software complexe porneste in general de la un **caiet de sarcini initial**, care poate fi apoi dezvoltat (corectat, actualizat, etc.) odata cu sistemul.

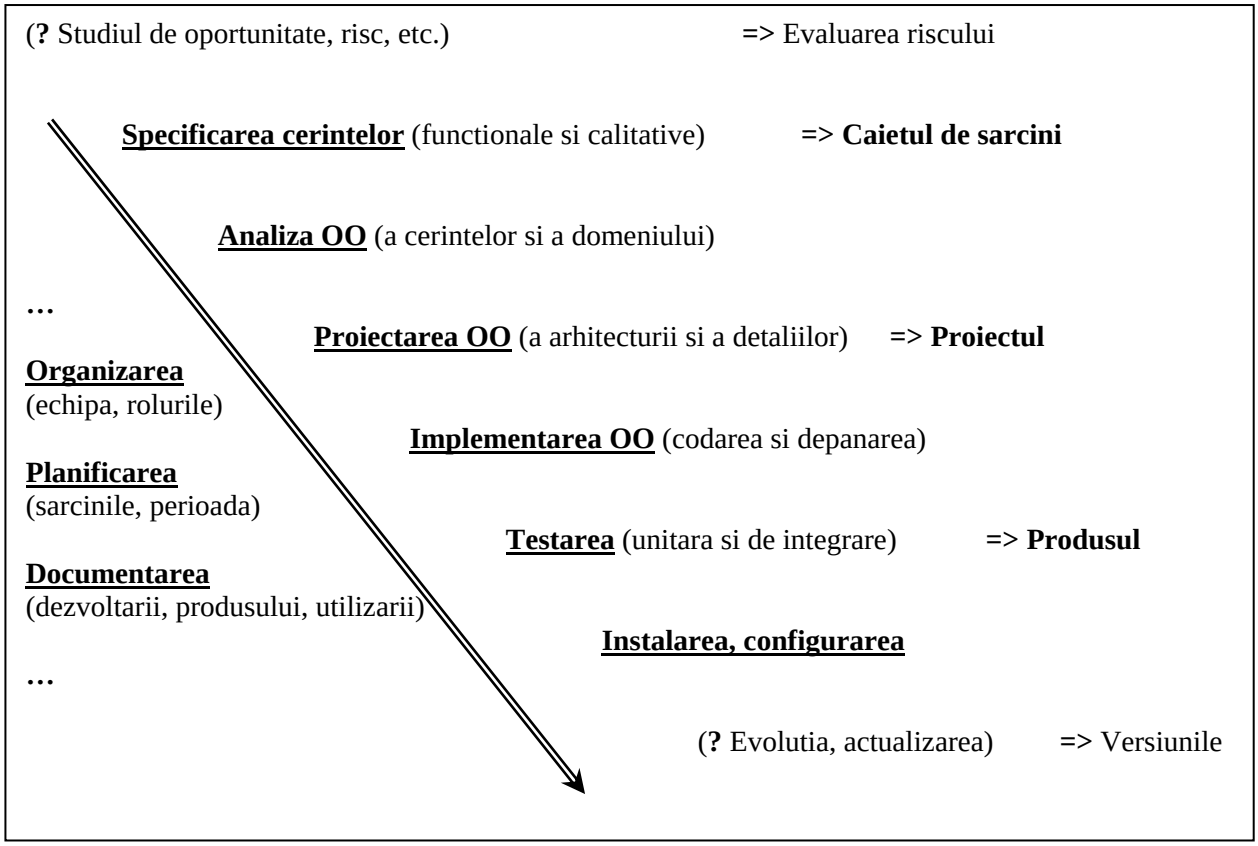
O activitatea necesara uneori stabilirii caietelor de sarcini este **studiul de oportunitate** (studiul de piata, financiar, de risc, etc.) al produsului pe care si-l propune drept tinta procesul de dezvoltare.

In cazul sistemelor software complexe, programarea nu este ultima activitate a procesului de dezvoltare, ea fiind urmata in general de **testarea sistemului software** creat. Si aceasta activitate poate fi divizata intr-o prima faza de **testare unitara**, a fiecarui element (bloc, modul, obiect, componenta, subsistem) nou creat, urmata de testarea de ansamblu (**de integrare**) a sistemului.

In plus, in cazul sistemelor software complexe, produsul trebuie sa fie insotit de o documentatie de instalare, utilizare, etc., ceea ce poate impune o activitate de **documentare** desfasurata (de preferinta) in paralel cu celelalte activitati de dezvoltare).

Dupa finalizarea produsului poate fi necesara de asemenea **instalarea si configurarea** lui la beneficiar, si asigurarea **intretinerii** lui, si eventual a generatiilor sau versiunilor succesive, intr-o forma evolutiva.

Mai jos sunt prezentate **principalele activitati ale unui proces de dezvoltare a unui sistem *software* complex**, si **rezultatele acestor activitati** (*artefactele*).



P.1.1. Specificarea initiala a cerintelor (anterioara utilizarii diagramelor UML)

sus

Specificarea initiala a cerintelor (functionale si calitative) se face in general printr-un **caiet de sarcini** initial (uneori rezultat in urma unui studiu de oportunitate).

Iata un set de cerinte initial, pentru un **sistem de comunicatie distribuit, client-server, bazat pe conexiune** (socketuri TCP)**, suport pentru conversatie textuala** (***chat***).

Sistemul *software* pentru conversatie textuala (*chat*) va avea urmatoarele caracteristici: implementat in limbajul de programare **Java**, structura de tip **client-server**, bazat pe **socketuri flux** (orientate spre **conexiune**, folosind protocolul **TCP**), oferind utilizatorilor o **interfata grafica Swing**.

Modul de lucru este urmatorul: **Utilizatorul *lanseaza*** componenta **client** a sistemului. **Clientul** se ***conecteaza*** la server, ***ofera*** o interfata grafica **utilizatorului**, apoi ***trimite*** mesaje catre server, ***preluate*** de la **utilizator** prin interfata grafica. **Serverul *accepta*** conexiunile si ***creaza* fire de executie pentru tratarea** clientilor. Fiecare **fir de tratare** a unui client va ***primi*** mesaje de la **clientul** tratat si va ***difuza*** aceste mesaje catre toti **clientii**. Fiecare **client *preia*** mesajele de la server si le ***prezinta* utilizatorului** in interfata grafica.

P.1.2. Rolul diagramelor UML in faza analizei OO (orientate spre obiecte)

sus

Analiza OO intentioneaza sa captureze si sa descrie toate cerintele domeniului si sa creeze un model care defineste clasele cheie ale domeniului in sistem (ce se intampla in sistem). **Scopul** este ***sa ofere o intelegere*** si sa asigure o comunicatie despre sistem intre dezvoltatori si oameni stabilind cerintele (utilizatori/client). Din aceste motive analiza este indrumata de obicei in cooperare cu utilizatorul sau clientul. Analiza nu este restrictionata de o solutie tehnica sau detalii. Dezvoltatorul nu trebuie sa gandeasca in termeni de cod ori de programe in timpul acestei faze; este primul pas inspre adevarata intelegere a cerintelor si a realitatii sistemului in proiectare.

P.1.2.1. Analiza cerintelor

Analiza cerintelor incearca sa identifice cerintele, sa clasifice cerintele, sa stabileasca prioritatile in satisfacerea cerintelor, etc.

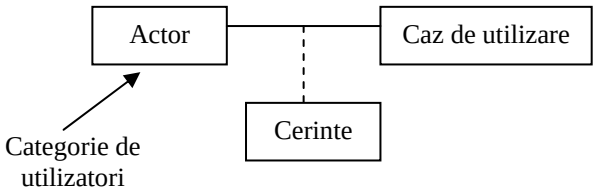
Primul pas al analizei cerintelor este ***definirea*** **cazurilor de utilizare** care descriu ce ofera sistemul biblioteca in termeni de functionalitate – cerintele functionale ale sistemului. Analiza unui caz de utilizare implica citirea si analizarea specificatiilor ca si discutia sistemului cu potentialii utilizatori (clienti) ai sistemului.

Analiza incepe prin ***cautarea*** **actorilor** (**categoriilor de utilizatori** ai) sistemului. Un **actor** reprezinta **rolul jucat de catre o persoana sau de catre un lucru care interactioneaza cu sistemul**. Nu este intotdeauna usoara determinarea limitei sistemului. Prin definitie, actorii sunt exteriori sistemului.

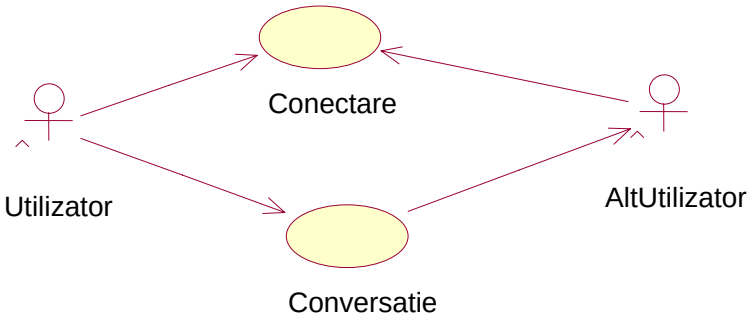
Actorii se gasesc, de exemplu, printre utilizatorii sistemului si printre cei responsabili cu configurarea si intretinerea sa. **Actorii** identificati **pentru sistemul** ***chat*** sunt **utilizatorii** lui.

Utilizator

Un caz de utilizare este o abstractie a unei parti a comportamentului sistemului. Cazul de utilizare este instantiat la fiecare utilizare a sistemului de catre o instanta a unui actor. Dupa intervievarea utilizatorilor, se obtine descompunerea cerintelor functionale ale categoriilor de actori.



Cazurile de utilizare identificate **pentru sistemul** ***chat*** sunt conectarea utilizatorului si conversatia intre utilizatori.



Cazurile de utilizare sunt implementate pe tot parcursul dezvoltarii sistemului, pentru a oferi descrieri ale cerintelor functionale ale sistemului. Ele sunt **folosite in analiza ca sa verifice daca clasele potrivite ale domeniului au fost definite**, si ele pot **fi folosite in timpul proiectarii pentru a confirma ca solutia tehnica este suficienta** pentru asigurarea functionalitatilor cerute. Cazurile de utilizare pot fi vizualizate in diagrame de secventa, care detaliaza realizarea lor.

Cazul de utilizare **Conectare** corespunde specificatiei primei faze de lucru a sistemului:

> **Utilizatorul** ***lanseaza*** componenta **client** a sistemului. **Clientul** se ***conecteaza*** la **server**, ***ofera*** o interfata grafica **utilizatorului**, ... **Serverul** ***accepta*** conexiunile si ***creaza*** **fire de executie pentru tratarea** clientilor ….

Cazul de utilizare **Conversatie** corespunde specificatiei celei de-a doua faze de lucru a sistemului:

> **Clientul** … ***trimite*** mesaje catre server, ***preluate*** de la **utilizator** prin interfata grafica. **…** Fiecare **fir de tratare** a unui client va ***primi*** mesaje de la **clientul** tratat si va ***difuza*** aceste mesaje catre toti **clientii**. Fiecare **client** ***preia*** mesajele de la server si le ***prezinta*** **utilizatorului** in interfata grafica.

Un caz de utilizare trebuie caracterizat prin:

1. Nume (cat mai sugestiv, pentru a sintetiza cazul de utilizare).

Ex. ***Conectare***

2. Scurta descriere.

Ex. ***Clientul*** *se* ***conecteaza*** *la* ***server****, care* ***accepta*** *conexiunea si se pregateste pentru tratarea* ***clientului****.*

3. Actori (entitati exterioare sistemului modelat, implicate in cazul de utilizare).

Ex. ***Utilizator****.*

4. Preconditii (conditiile necesare pentru declansarea cazului de utilizare).

Ex. ***Serverul*** *e aflat in executie.* ***Clientul*** *cunoaste adresa si numarul de port pe care asculta* ***serverul****.*

5. Evenimentul care declanseaza cazul de utilizare.

Ex. ***Utilizatorul lanseaza*** *componenta* ***client*** *a sistemului.*

6. Descriere a interactiunii dintre actori si fiecare caz de utilizare.

Ex. **I.** ***Utilizatorul lanseaza*** *componenta* ***client*** *a sistemului, care se* ***conecteaza*** *la* ***serverul*** *cunoscut.* ***Serverul accepta*** *conexiunea si* ***se pregateste*** *pentru tratarea clientului.*
II. ***Clientul ofera*** *o interfata grafica* ***utilizatorului***

7. Alternative la cazul de utilizare principal.

Ex. *Daca* ***serverul nu este lansat****, conectarea* ***esueaza****, iar* ***clientul anunta*** *acest lucru* ***utilizatorului****.*

8. Evenimentul care produce oprirea cazului de utilizare.

Ex. ***Clientul*** *prezinta* ***utilizatorului*** *o interfata grafica.*

9. Postconditii (efectele incheierii cazului de utilizare)

Ex. *Sistemul este pregatit pentru ca* ***utilizatorul*** *sa poata conversa cu alti utilizatori prin intermediul interfetei grafice.*

E important sa fie clar prezentate:

- **schimburile de informatii** (parametrii interactiunilor)

Ex. ***Utilizatorul*** *se conecteaza la sistem si isi da numele si parola.*

- **originea informatiilor** si **ordinea schimbarii** lor

- **repetitiile comportamentului**

Ex. Prin constructii pseudocod de genul:

loop		***bucla***
…	sau	**…**
end loop		***sfarsit bucla***

sau

while (condition)		***cat timp (conditie)***
…	sau	**…**
end while		***sfarsit cat timp***

- **situatiile optionale**

Ex. ***Utilizatorul*** *alege unul dintre elementele urmatoare (eventual de mai multe ori)*
a) optiunea X
b) optiunea Y
apoi continua cu …

Cazul de utilizare **Conversatie** ar putea avea urmatoarea descriere:

1. Nume
Conversatie

2. Scurta descriere.
Clientii *preiau mesaje de la* ***utilizatori,*** *le* ***trimit*** *catre* ***server****, care le* ***difuzeaza*** *catre toti* ***clientii****, iar acestia le* ***prezinta*** *catre* ***utilizatori****.*

3. Actori
Utilizatori

4. Preconditii
Cazul de utilizare ***Conversatie*** *s-a incheiat cu succes.*

5. Evenimentul care declanseaza cazul de utilizare.
Un ***utilizator*** *scrie un mesaj in interfata grafica a* ***clientului*** *sau.*

6. Descriere a interactiunii dintre actori si cazul de utilizare (Transmisie si receptie).
I. ***Utilizatorul scrie*** *mesajul in interfata grafica a* ***clientului*** *sau,* ***clientul trimite*** *mesajul la* ***server****.*
II. ***Clientul*** *primeste inapoi mesajul* ***difuzat*** *de* ***server****, si* ***prezinta*** *mesajul in interfata grafica a* ***utilizatorului***

7. Alternativa la cazul de utilizare principal (Doar receptie).
Clientul *primeste mesajul trims de* ***alt utilizator****, mesaj* ***difuzat*** *de* ***server****, si* ***prezinta*** *mesajul in interfata grafica a* ***utilizatorului*** *sau.*

8. Evenimentul care produce oprirea cazului de utilizare.
Clientul *a prezentat* ***utilizatorului*** *mesajul primit.*

9. Postconditii
Sistemul este pregatit pentru ca ***utilizatorul*** *sa poata* ***continua conversatia*** *cu alti utilizatori prin intermediul interfetei grafice, sau pentru a* ***incheia conversatia*** *prin inchiderea interfetei grafice.*

P.1.2.2. Analiza domeniului

sus

Analiza domeniului detaliaza domeniul in care isi desfasoara activitatea sistemul, ***stabilind*** **clasele cheie in sistem**. Pentru a realiza o analiza a domeniului, cititi specificatiile si cazurile de utilizare si uitati-va care "concepte" trebuie tratate de catre sistem. Sau, organizati o dezbatere cu utilizatorii si cu expertii in domeniu pentru a incerca sa identificati toate conceptele cheie care trebuie tratate, impreuna cu relatiile dintre ele.

Clasele domeniului sunt doar "schitate" in acest stadiu. Operatiile si atributele definite nu sunt cele finale. **Ele sunt acelea care par potrivite pentru aceste clase in acest moment**.

In cazul sistemului *chat*, urmatoarele **clase** pot fi considerate necesare:

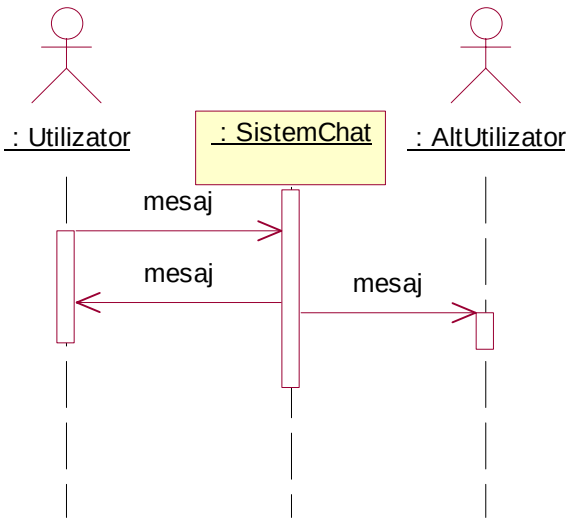
Clasele identificate, **Client** si **Server** pot deveni, in fazele de proiectare si implementare, **prin detaliere, subsisteme**.

Unele din operatiuni sunt definite prin **schitarea diagramelor de secventa** peste cazurile de utilizare. Pentru a descrie **comportamentul dinamic al claselor domeniu**, oricare din **diagramele UML dinamice** poate fi folosita: de secventa, de colaborare, sau de activitati.

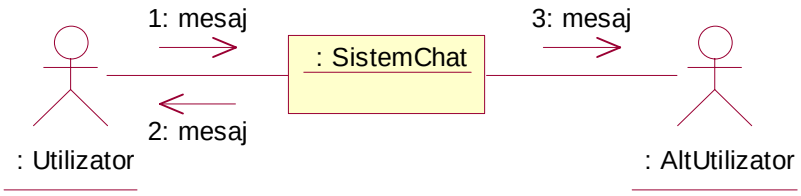
Bazele diagramelor de secventa sunt cazurile de utilizare, unde fiecare caz de utilizare este descris cu impactul sau asupra claselor domeniu, pentru a ilustra cum clasele domeniu colaboreaza pentru a realiza cazul de utilizare in interiorul sistemului.

Functionalitatile descrise prin cazuri de utilizare si **detaliate prin diagrame de secventa** sunt **dezvoltate in continuare prin** intermediul **colaborarilor intre obiectele domeniului**. Prin simplificarea diagramelor de colaborare, acestea sunt transformate in diagrame de obiecte. Obiectele fiind instante ale claselor, diagramele de obiecte conduc in continuare la diagrame de clase. Anumite clase au diagrame UML de stari care sa arate diferitele stari pe care obiectele acelor clase le pot avea, impreuna cu evenimentele care le fac sa-si schimbe starea.

In cazul sistemului *chat*, urmatoarea **diagrama de secventa poate modela comportamentul intregului sistem**, in relatia lui cu ***Utilizatorii***, in cazul de utilizare ***Conversatie***:



<u>Diagrama de colaborare</u> echivalenta diagramei de secventa anterioara este urmatoarea:



Daca utilizam cele doua clase identificate, **Client** si **Server**, **<u>se poate detalia</u> diagrama de secventa care modeleaza comportamentul sistemului**, in relatia lui cu ***Utilizatorii***, in cazul de utilizare ***Conversatie***:

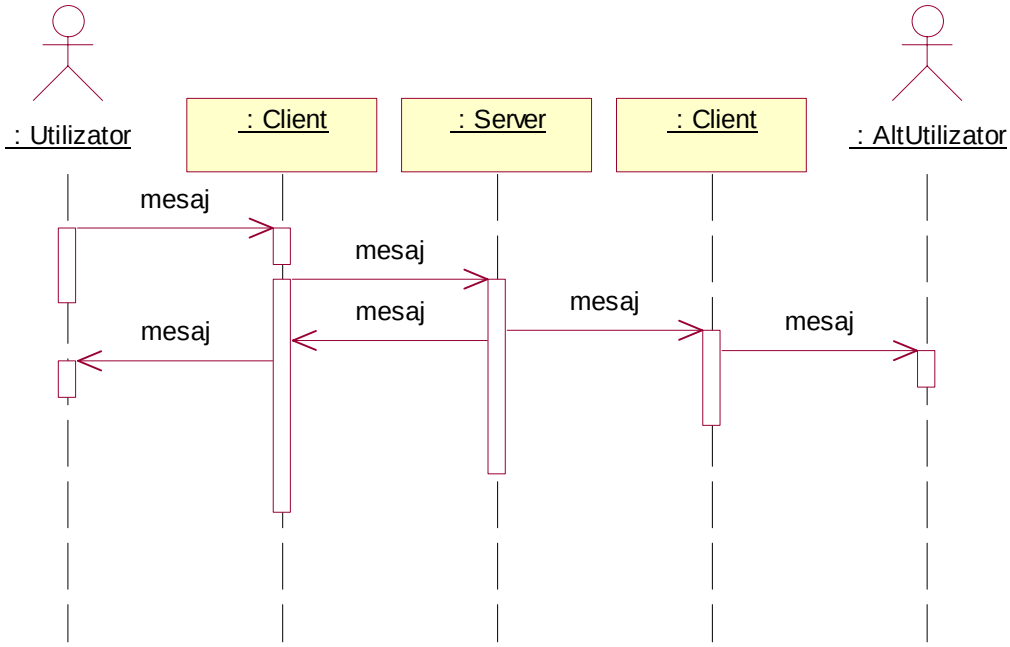
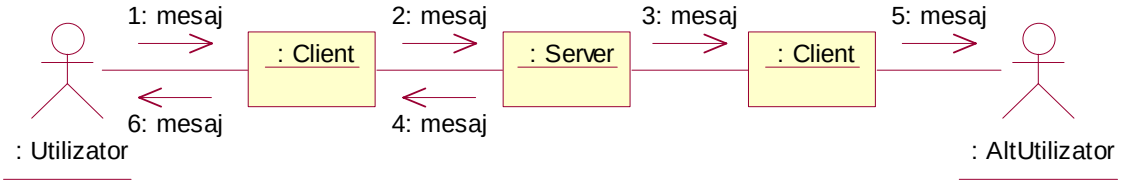
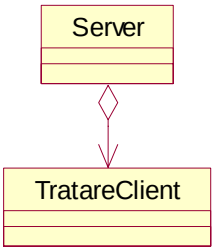


Diagrama de colaborare echivalenta diagramei de secventa anterioara este urmatoarea:



In aceasta faza, putem observa **necesitatea existentei unor obiecte ale subsistemului Server** care **sa se ocupe cu tratarea fiecarui client in parte**, sub forma clasei **TratareClient**, aflata **in relatie de subordonare** fata de clasa **Server**.



Se pot acum <u>detalia suplimentar</u> diagramele de secventa care modeleaza cazurile de utilizare ***Conectare*** si ***Conversatie***:

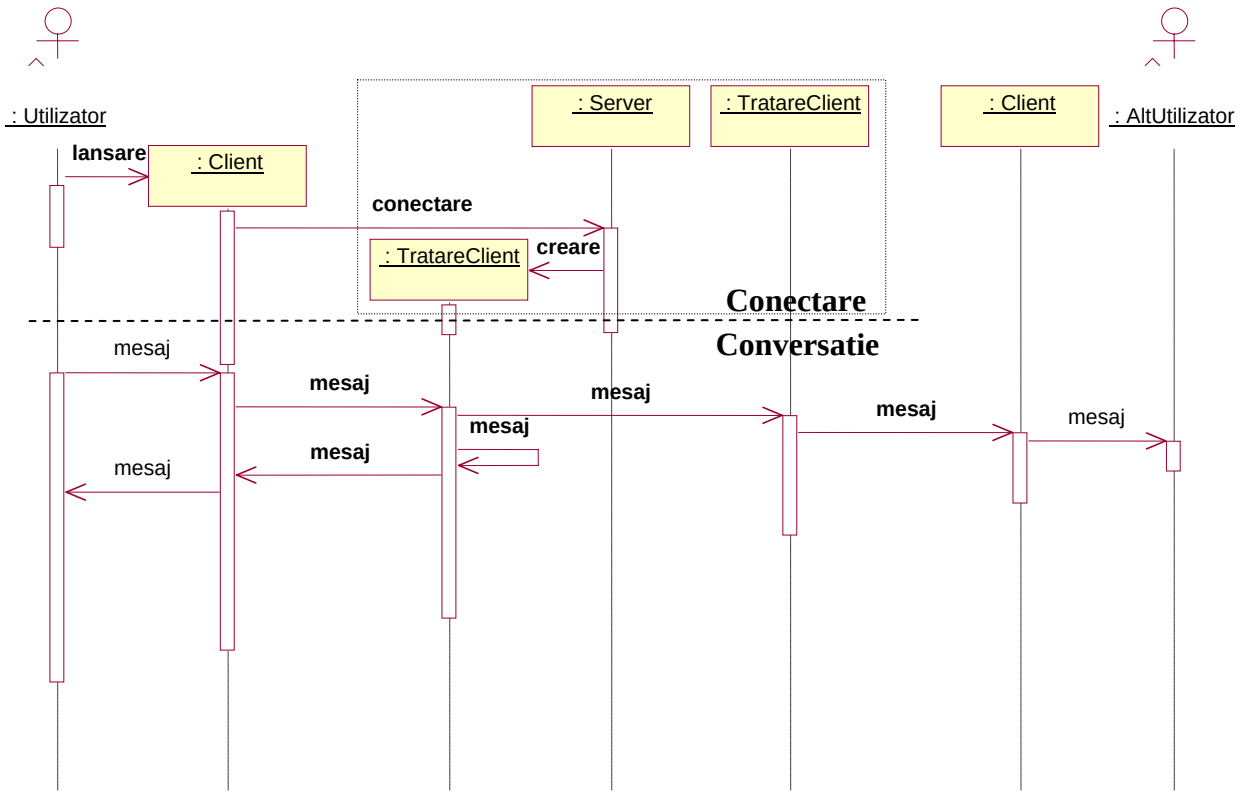
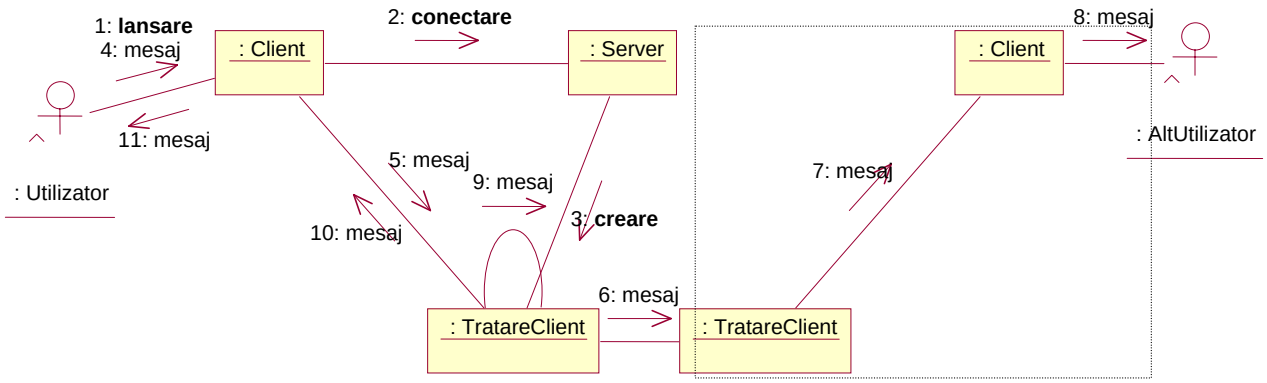


Diagrama de colaborare echivalenta diagramei de secventa anterioara este urmatoarea:



Cand modelam cu diagrame de secventa, **<u>devine evident ce ferestre sau dialoguri sunt necesare</u> pentru a asigura o interfata actorilor**. In analiza este suficient sa fie constientizate ferestrele interfata care sunt necesare si identificarea interfetelor de baza. **Interfata de utilizator detaliata nu este specificata la acest moment**; din nou, **aceasta este doar <u>o schita</u> a ceea ce interfata de utilizator include**. Pentru a separa in analiza clasele fereastra de clasele domeniu, clasele fereastra sunt grupate de obicei intr-un pachet distinct.

In cazul sistemului *chat*, se poate decide existenta unei **interfete grafice formata dintr-o intrare de text pentru editarea mesajelor de trimis, si o zona grafica de text pentru prezentarea mesajelor receptionate** de la server.

La acest moment, **aplicatiei i se poate da**, de asemenea, **un nume**.

P.1.3. Rolul diagramelor UML in faza proiectarii OO (orientate spre obiecte)

sus

Proiectarea OO extinde si detaliaza modelul obtinut prin analiza tinand cont de toate implicatiile tehnice si restrictiile. Scopul proiectarii este sa specifice o solutie care functioneaza, care poate fi usor trecuta in cod de programare. Clasele definite in analiza sunt detaliate si clase noi sunt adaugate pentru a rezolva ariile tehnice, cum ar fi bazele de date, interfata cu utilizatorul, comunicatia, dispozitivele si altele.

P.1.3.1. Proiectarea arhitecturii

Proiectarea arhitecturii este o proiectare **de nivel inalt**, unde sunt definite pachetele (subsistemele), incluzand dependentele si mecanismele primare de comunicatie intre pachete. Desigur, scopul este o arhitectura limpede si simpla, unde sunt dependente putine si dependentele bidirectionale sunt evitate pe cat posibil.

O **arhitectura bine proiectata** este **baza unui sistem usor extensibil si modificabil**. Pachetele pot sa aiba fie preocuparea manevrarii unui domeniu functional specific, fie a unui domeniu tehnic specific. Este vital **sa separam logica aplicatiei (clasele domeniului) de logica tehnica** astfel incat schimbarile din oricare din aceste segmente sa poata fi realizate fara prea mare impact in nici una din parti.

Scopurile cheie, cand definim arhitectura, sunt **identificarea si stabilirea unor reguli pentru dependentele intre pachete**, in asa fel incat sa nu fie creata nici o dependenta bidirectionala intre pachete (se evita ca pachetele sa devina prea strans integrate intre ele), si ***identificarea nevoii de biblioteci standard*** si **gasirea bibliotecilor utilizabile**. Bibliotecile disponibile pe piata de azi a domeniilor tehnice sunt: **interfata cu utilizatorul**, **bazele de date**, sau **comunicatiile**, dar, mai multe **biblioteci specifice aplicatiei**, sunt asteptate, de asemenea, sa apara.

In cazul sistemului *chat*, urmatoarele biblioteci pot fi considerate necesare:

– biblioteca standard Java pentru comunicatii la nivel de *socket-uri* (`java.net`)
– biblioteca extensie standard Java pentru interfete grafice avansate (`javax.swing`) si bibliotecile grafice asociate (`java.awt` si `java.awt.event`)
– biblioteca standard Java pentru fluxuri de intrare-iesire (`java.io`)
– biblioteca standard Java pentru clase utilitare (`java.util`)

P.1.3.2. Proiectarea detaliata

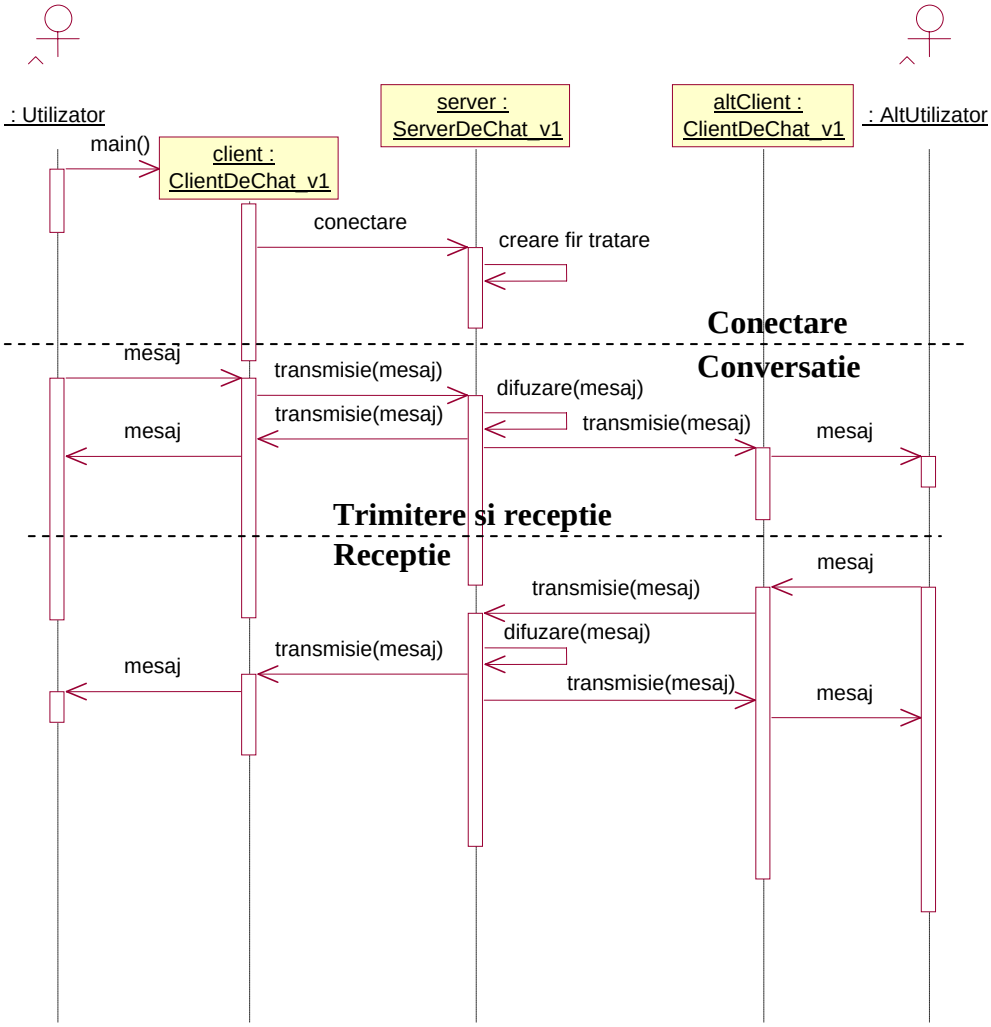
Proiectarea detaliata detaliaza continutul pachetelor, in asa fel incat clasele sunt descrise destul de amanuntit pentru a da specificatii clare programatorului care va crea codul claselor. Modelele dinamice din UML sunt utilizate pentru a demonstra cum obiectele claselor se comporta in situatii specifice.

Scopul proiectarii detaliate este **sa descrie noile clase tehnice** – clase din **interfata cu utilizatorul** si din **pachetele bazei de date** – si **sa extinda si sa detalieze descrierea claselor domeniului**, care au fost deja schitate in analiza. Aceasta se face creand noi diagrame de clase, diagrame de stare si diagrame dinamice (cum ar fi cele de secventa, de colaborare si de activitate).

Sunt **aceleasi diagrame ca cele folosite in analiza**, dar aici ele sunt **definite la un nivel tehnic si de detaliere mai ridicat**. Descrierile cazului de utilizare din analiza sunt folosite pentru verificarea fiabilitatii cazurilor de utilizare in proiectare; iar diagramele de secventa sunt folosite pentru a ilustra cum fiecare din cazurile de utilizare este realizat tehnic in sistem.

Mai intai pot fi create <u>diagrame UML de secventa</u> (a mesajelor) si **<u>de colaborare</u>** (a obiectelor) **<u>de nivel inalt</u>**, care sa reflecte cazurile de utilizare sis a detalieze aspecte tehnice legate de implementare.

Varianta de cel mai inalt nivel al diagramei de secventa prezinta **<u>serverul in ansamblu</u>**, **fara a detalia interactiunile dintre el si firele de tratare a clientilor**.



Prima parte a diagramei de secventa a mesajelor, scenariul **Conectare,** corespunde cazului de utilizare **Conectare**, a doua parte a diagramei, scenariul **Conversatie,** corespunde cazului de utilizare **Conversatie**.

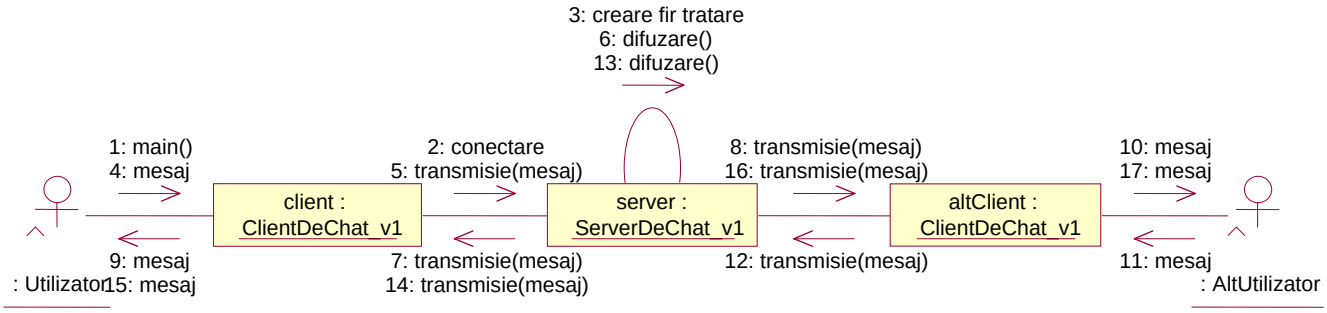
Scenariul **Trimitere si Receptie** corespunde situatiei in care clientul curent preia un mesaj de la utilizatorul sau, trimite mesajul catre server, firul de executie al serverului care trateaza acest client difuzeaza mesajul, care e preluat inclusiv de clientul utilizatorului curent si prezentat acestuia.

Scenariul **Receptie** corespunde situatiei in care clientul curent preia un mesaj de la server, mesaj difuzat de un alt client, si prezinta mesajul utilizatorului sau.

Diagramele de secventa au ca echivalent (biunivoc) **diagrame de colaborare**. Diagramele de secventa a mesajelor prezinta mai degraba aspectul temporal, comportamental, al obiectelor implicate in colaborare, pe cand diagramele de colaborare a obiectelor prezinta mai degraba aspectul static, structural.

Altfel spus, este complicat sa se inteleaga structura sistemului privind o diagrama de secventa complexa (cum este cazul nostru). De asemenea, este complicat sa se inteleaga comportamentul sistemului privind o diagrama de colaborare, dar numerotarea mesajelor poate simplifica intelegerea.

Diagrama de colaborare de cel mai inalt nivel, care prezinta **serverul in ansamblu**, **fara a detalia interactiunile dintre el si firele de tratare a clientilor**, este urmatoarea.



<u>Pentru a avansa cu proiectarea, catre implementare</u>, se poate trece la **<u>detalierea diagramelor</u> de secventa si colaborare**. Varianta diagramei de secventa care prezinta **serverul care accepta conexiunile separat de firele de tratare a clientilor** detaliaza interactiunile interne serverului ca ansamblu.

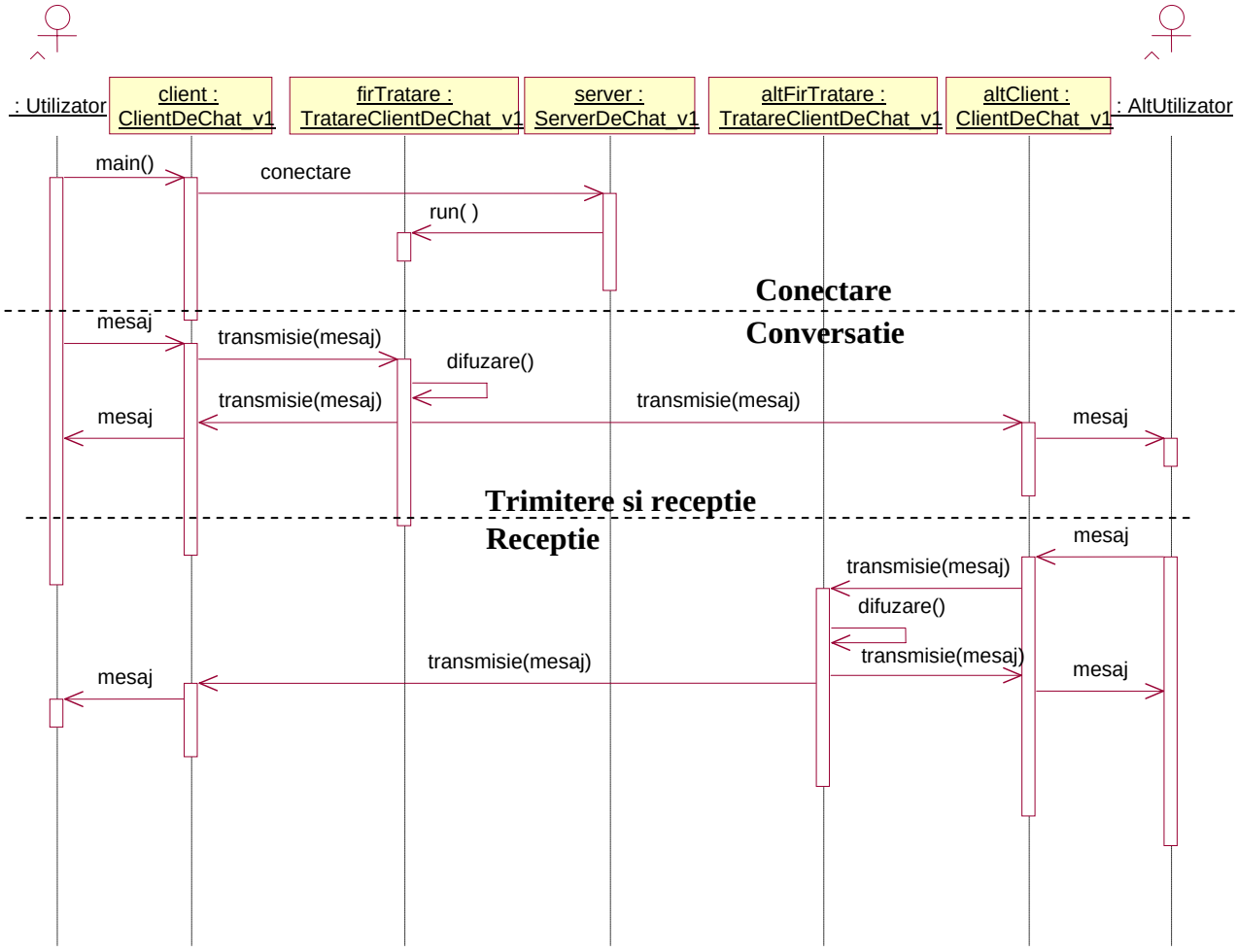
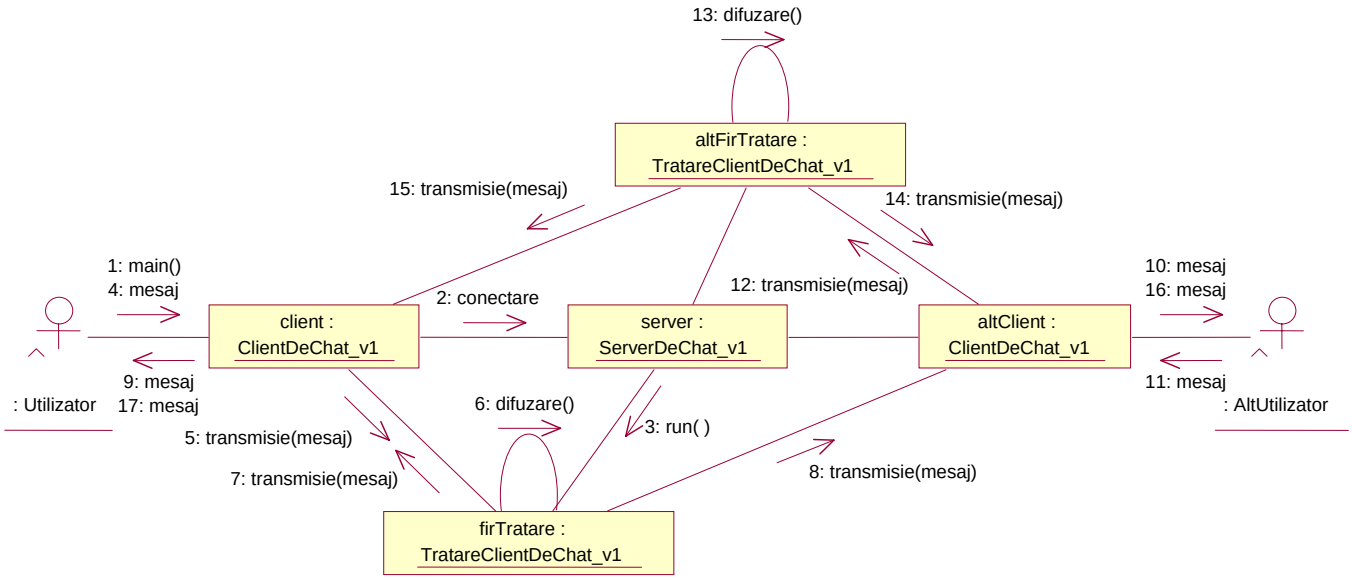
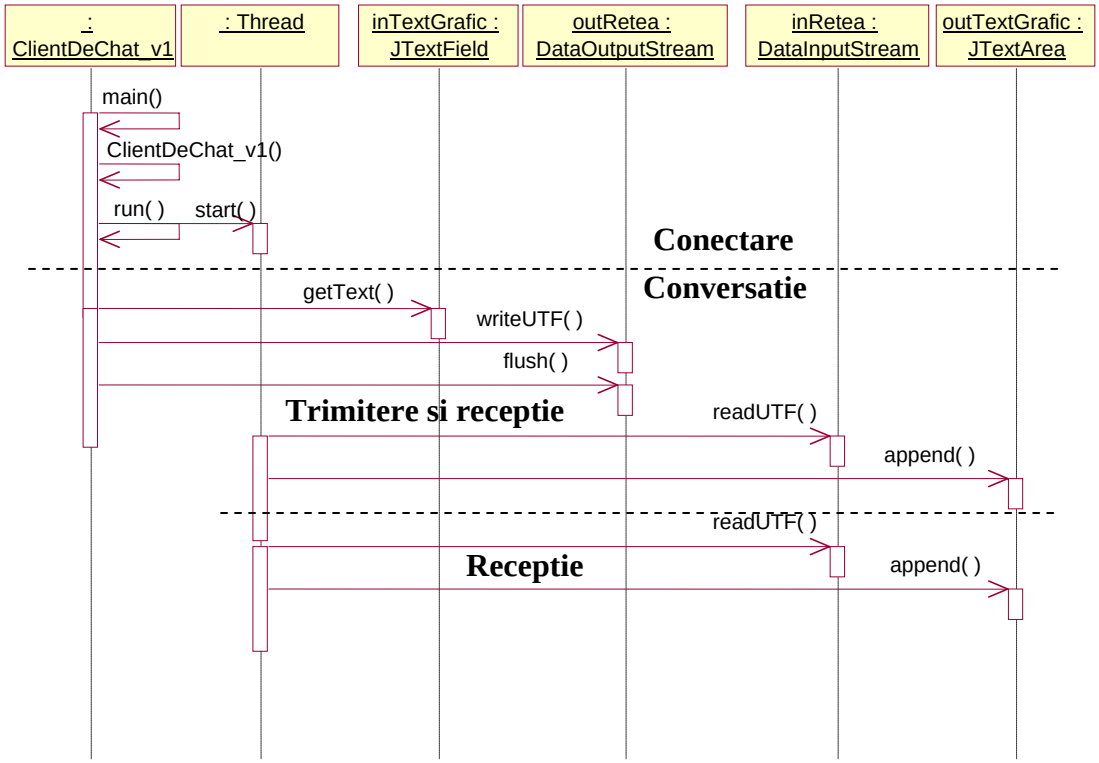


Diagrama de colaborare care prezinta **serverul care accepta conexiunile separat de firele de tratare a clientilor** (echivalenta diagramei de secventa anterioare) este urmatoarea.



Se poate merge <u>mai departe cu detalierea</u> diagramelor UML de secventa si **de colaborare** pentru sistemul chat, pe masura ce sunt decise detalii de implementare (numele metodelor Java).

Diagrama de secventa pentru **client** se poate **detalia**, la nivel de apeluri de metode Java, astfel:



Prima parte a diagramei de secventa a mesajelor pentru **client** corespunde scenariului **Conectare**, a doua parte a diagramei corespunde scenariului **Conversatie**.

Diagrama de colaborare pentru **client**, echivalenta diagramei de secventa anterioara, este urmatoarea.

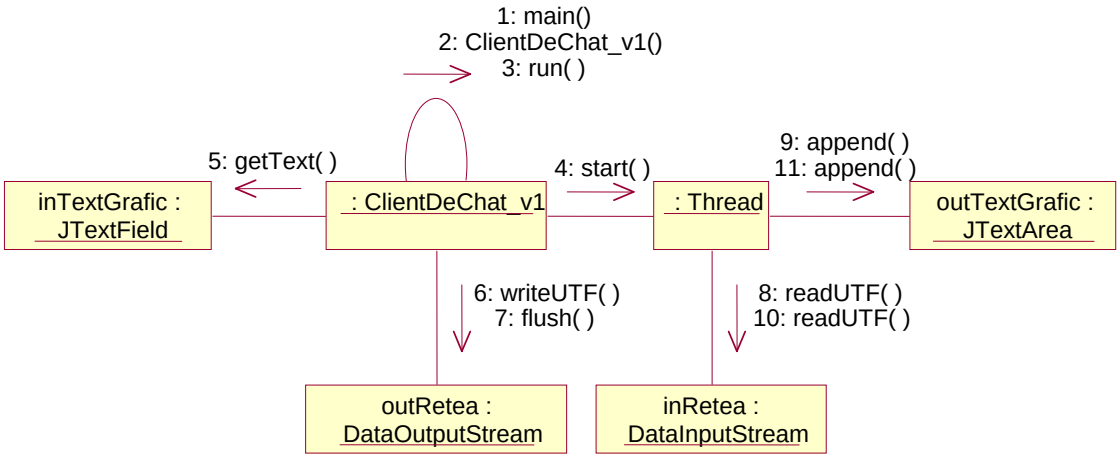
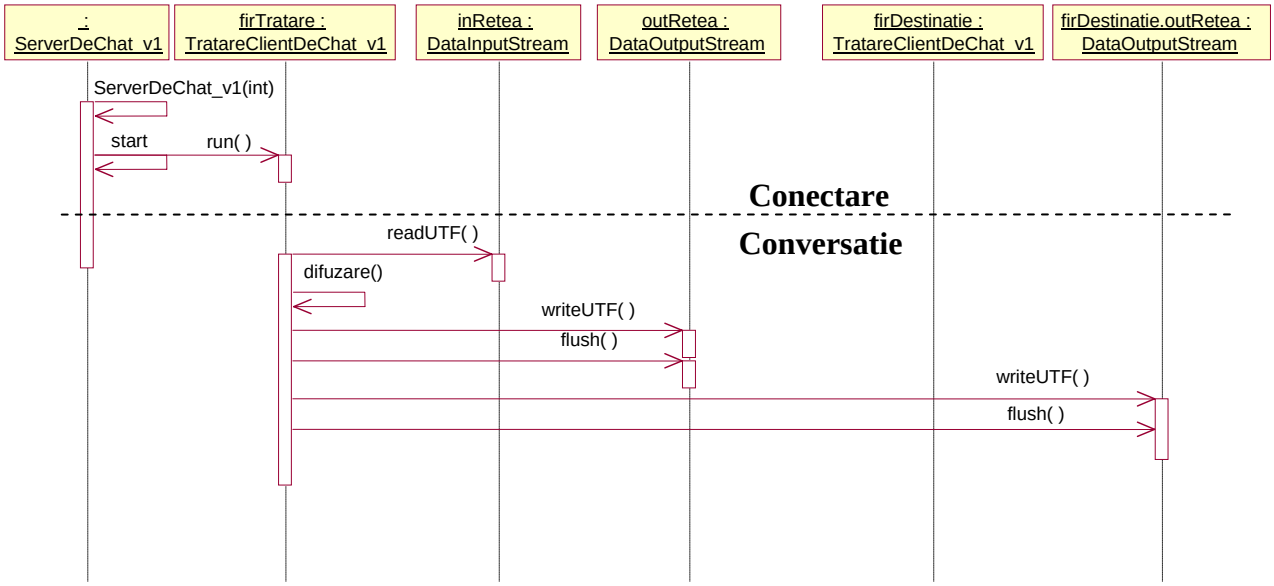
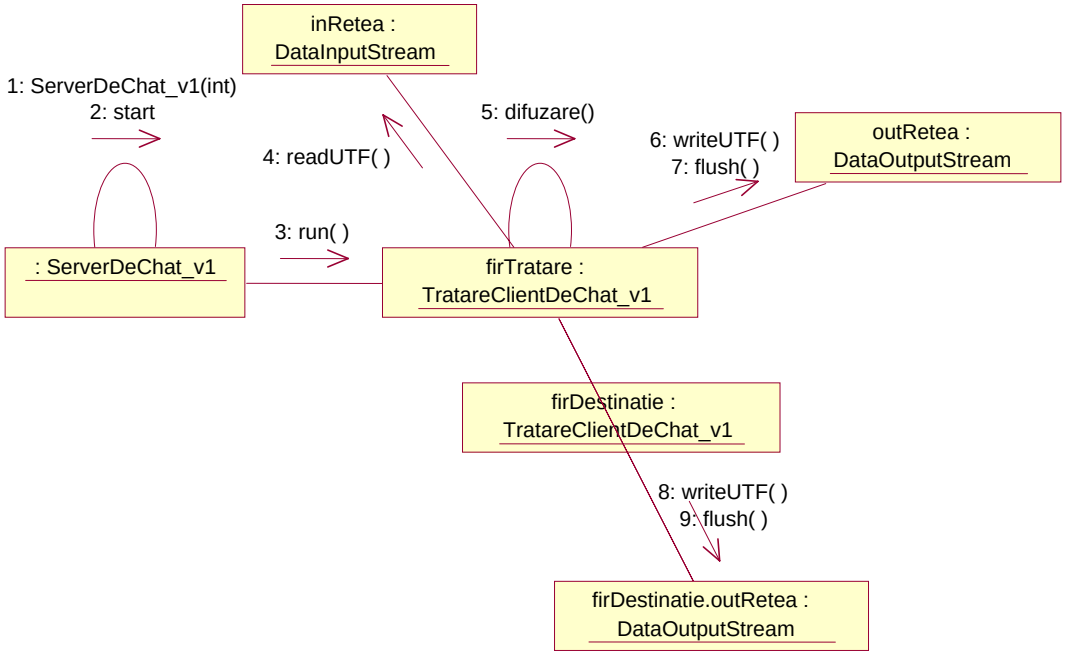


Diagrama de secventa pentru **server** se poate **detalia**, la nivel de apeluri de metode Java, astfel:



Prima parte a diagramei de secventa a mesajelor pentru **server** si **firele de tratare a clientilor** corespunde cazului de utilizare **Conectare**, a doua parte a diagramei corespunde cazului de utilizare **Conversatie**.

Diagrama de colaborare pentru si **firele de tratare a clientilor**, echivalenta diagramei de secventa anterioara este urmatoarea.



Se poate trece acum la <u>crearea diagramelor UML de clase</u> pentru sistemul *chat* in forma cea mai detaliata.

Diagrama de clase a **clientului** este urmatoarea:

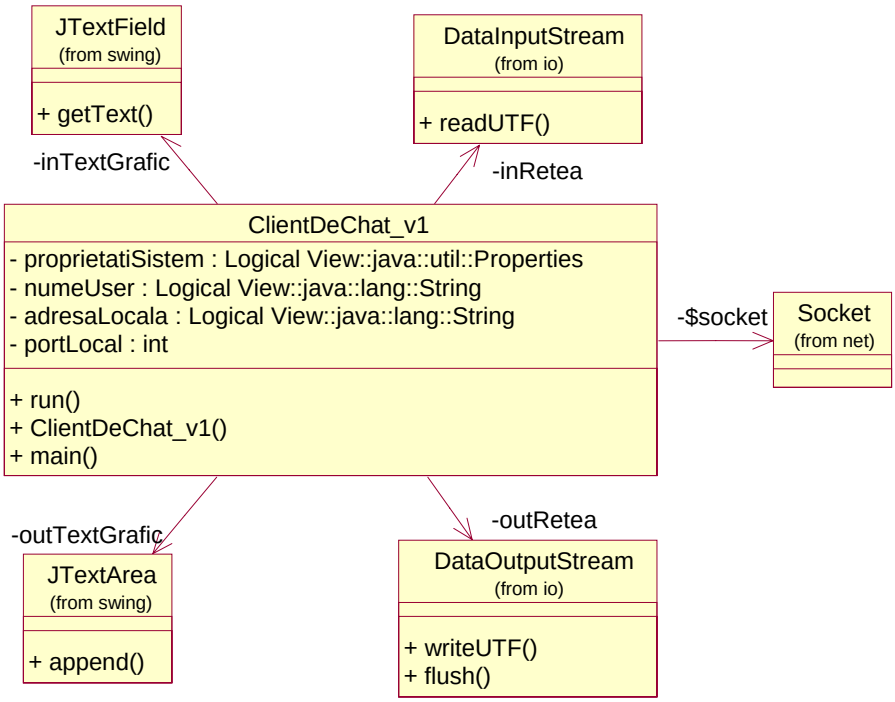
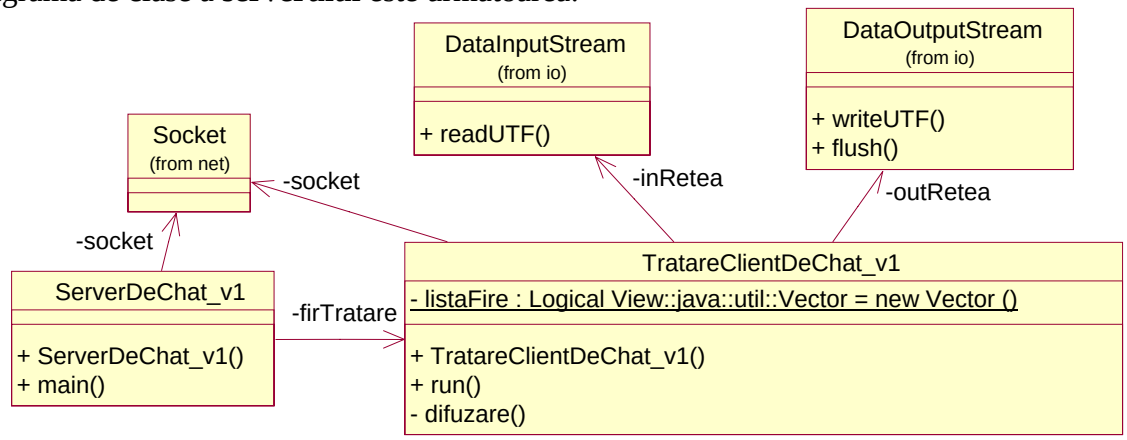
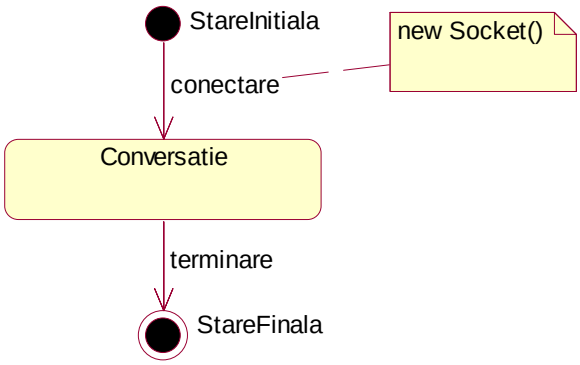


Diagrama de clase a **serverului** este urmatoarea:

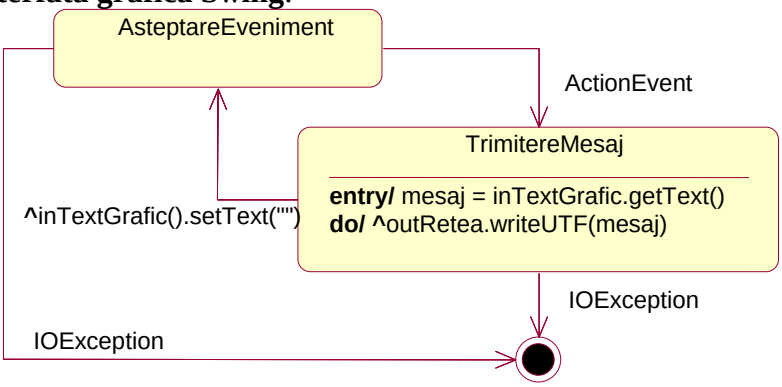


<u>Comportamentul principalelor clase</u> din sistem poate fi **<u>detaliat prin diagrame UML de stari</u>** (si tranzitii).

Diagrama de stari a **clientului** este urmatoarea:



Starea **Conversatie** a **clientului** poate fi detaliata pentru **firul de executie** care **trateaza evenimentele din interfata grafica Swing**:



Starea **Conversatie** a **clientului** poate fi detaliata pentru **firul de executie** care **trateaza receptia mesajelor de la server**:

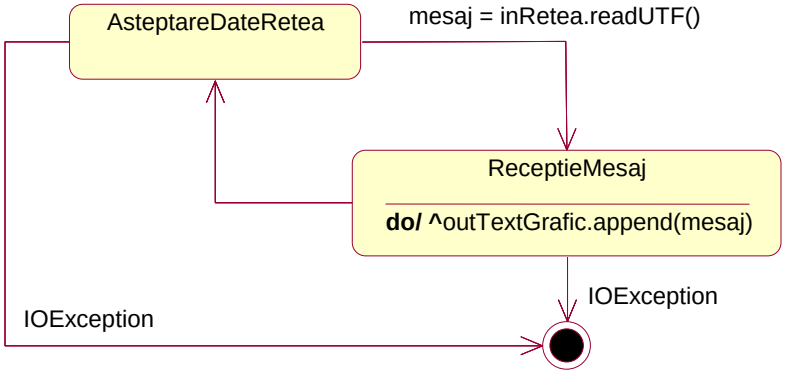


Diagrama de stari a **serverului** este urmatoarea:

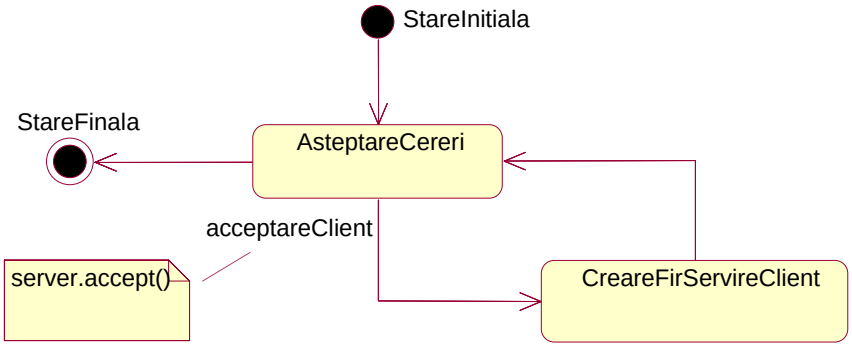
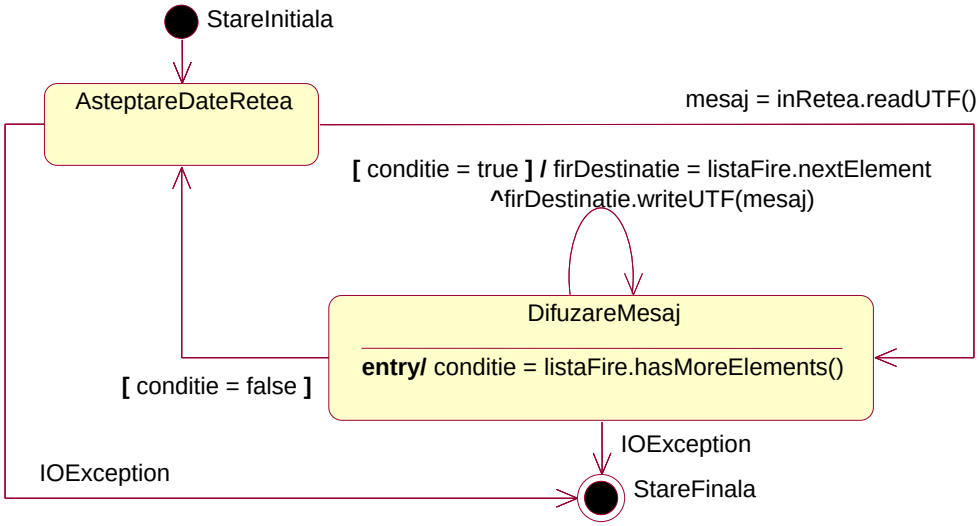
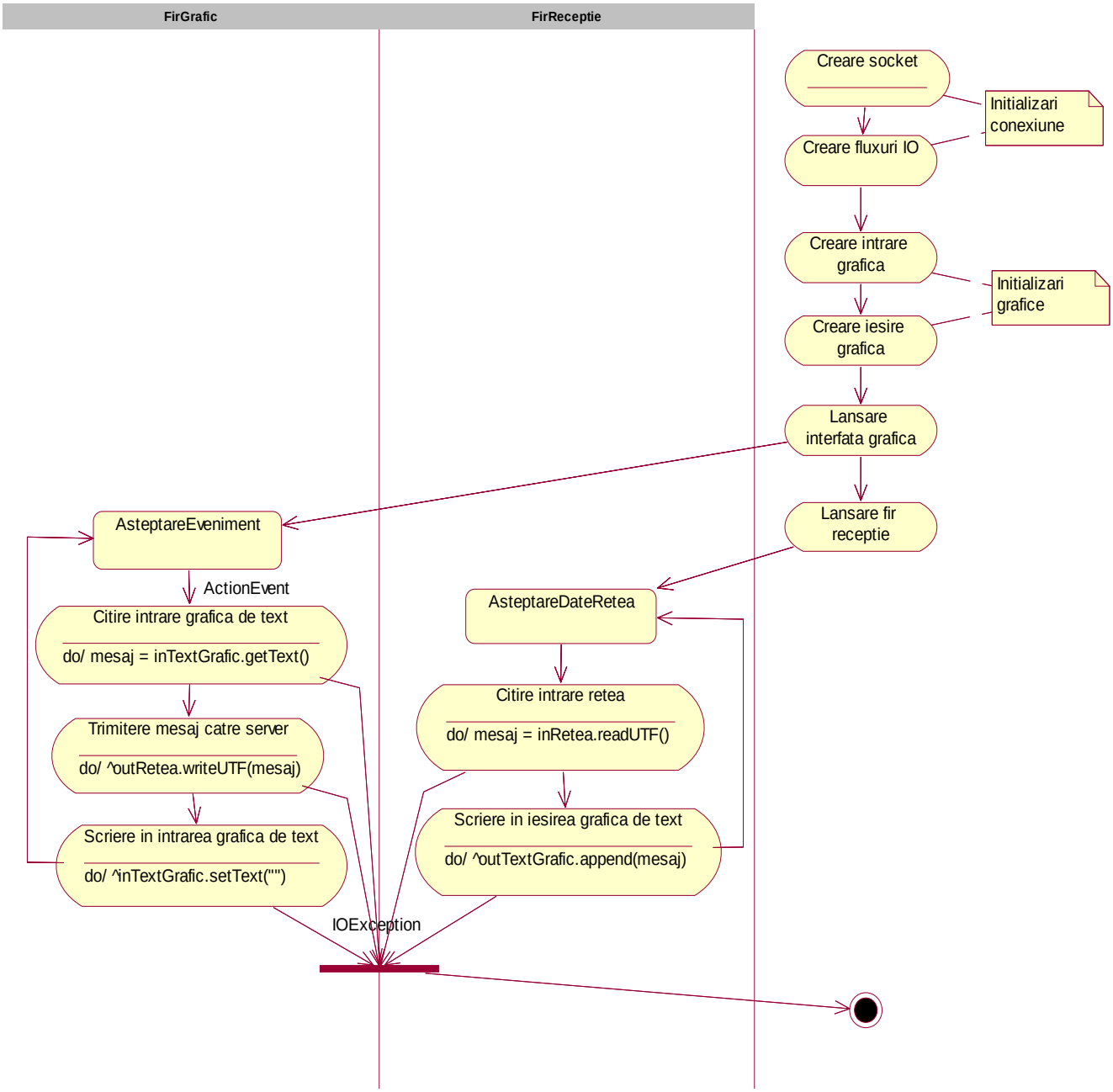


Diagrama de stari a **firului de tratare a clientilor** este urmatoarea:



O **alternativa la modelarea comportamentului prin diagrame de stari**, este modelarea lui prin **diagrame UML de activitati**.

Diagrama de activitati a **clientului** este urmatoarea:



In mod asemanator poate fi creata si o diagrama de activitati a serverului.

Se observa ca **diagramele de activitati se aseamana cu organigramele**, mijloace clasice de modelare a algoritmilor.

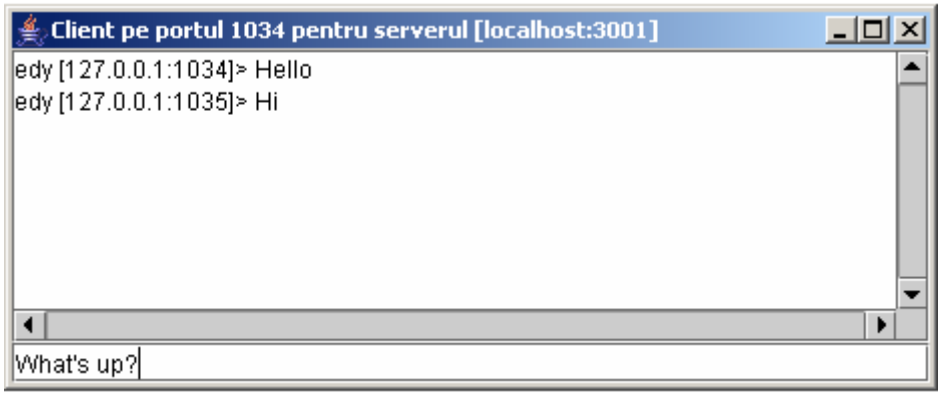
P.1.3.3. Proiectarea interfetei cu utilizatorul

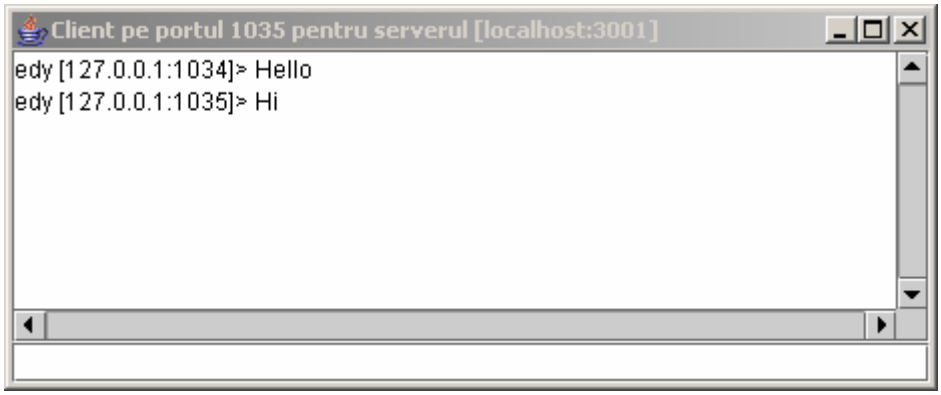
sus

Crearea proiectului interfetei cu utilizatorul ("felul cum arata ea") este o activitate speciala executata in timpul fazei de proiectare. Initiata in timpul fazei de analiza, aceasta este facuta **separat, dar in paralel cu alte activitati de proiectare**.

In cazul sistemului *chat*, se poate decide existenta unei **interfete grafice formata dintr-o intrare de text** (sub forma unei linii) **pentru editarea mesajelor de trimis, si o zona grafica de text** (multilinie si cu posibilitati de defilare) **pentru prezentarea mesajelor receptionate** de la server.

Iata cum ar putea arata o astfel de **interfata grafica, pentru doi utilizatori aflati in conversatie**:





P.1.4. Rolul diagramelor UML in faza implementarii OO (orientate spre obiecte)

sus

Implementarea OO bazata pe generare automata a codului pornind de la modele UML obtinute in faza de proiectare (**ingineria *software* directa** – *forward engineering*) a facut obiectul lucrarii de laborator anterioare.

Aplicarea unui **proces de dezvoltare iterativ** este usurat de posibilitatea aplicarii **ingineriei *software* inverse** (*reverse engineering*), care consta in **generarea modelului UML din cod sursa**. Acest proces iterativ ia forma **ingineriei *software* iterative** (*round-trip engineering*).

P.1.5. Rolul diagramelor UML in faza testarii sistemului

Descrierile cazurilor de utilizare pot fi folosite pentru **verificarea fiabilitatii cazurilor de utilizare** in proiectare, iar **diagramele de secventa** din faza de proiectare **pot fi folosite pentru a testa felul in care fiecare din cazurile de utilizare este realizat tehnic in sistem**.

P.1.6. Codul sursa pentru sistemul chat

sus

In continuare va fi prezentat codul sursa pentru clasele componente ale sistemului *chat* (versiunea 1), mai intai in **forma finala,** apoi doar **partea de cod care poate fi obtinuta prin generare automata a codului din diagramele de clasa ale modelului UML (versiunea 1).**

O varianta evoluata de cod sursa (versiunea 1+).

Codul sursa al clientului de *chat* (versiunea 1):

(rulat local cu **fisierul *batch***)

```
 1  import java.net.*;                // ClientDeChat_v1.java
 2  import java.io.*;
 3  import java.util.*;
 4  import java.awt.*;
 5  import java.awt.event.*;
 6  import javax.swing.*;
 7  import java.util.Properties;
 8
 9  /**
10   * Client de chat simplu - aplicatie de sine statatoare
11   * Aplicatie grafica Swing (extinde JFrame)
12   * care poate lansa in executie un fir nou (implementeaza Runnable)
13   */
14  public class ClientDeChat_v1 extends JFrame implements Runnable {
15    private Properties proprietatiSistem;
16    private String numeUser;
17    private String adresaLocala;
18    private int portLocal;
19
20    /**  Flux de intrare dinspre retea */
21    private DataInputStream inRetea;
22
23    /**
24     * Zona de text (configurata ca non-editabila)
25     */
26    private JTextArea outTextGrafic;
27
28    /**
29     * Intrare de text (editabila)
30     */
31    private JTextField inTextGrafic;
32
33    /**
34     * Fir de executie
35     */
36    private Thread firReceptie;
37
38    /**
39     * Socket flux (TCP)
40     */
41    private static Socket socket;
42
43    /**
44     * Flux de iesire catre retea
45     */
46    private DataOutputStream outRetea;
47    private InetAddress localHost;
48
49    /**
50     * Initializeaza obiectul de tip ClientDeChat_v1
51     * @param title      Titlul ferestrei
52     * @param inRetea    Flux de intrare dinspre retea
53     * @param outRetea   Flux de iesire catre retea
54     */
55    public ClientDeChat_v1(String title, InputStream inRetea,
56                                         OutputStream outRetea) {
57      // Stabilire titlu fereastra (JFrame)
58      super (title);
59
60      this.inRetea = new DataInputStream (new BufferedInputStream (inRetea));
61      this.outRetea = new DataOutputStream (new BufferedOutputStream (outRetea));
62
63      Container containerCurent = this.getContentPane();
64
65      containerCurent.setLayout(new BorderLayout());
66
```

```
67      // Zona de text non-editabila de iesire (cu posibilitati de defilare)
68      outTextGrafic = new JTextArea(8, 40);
69      JScrollPane scrollPane = new JScrollPane(outTextGrafic,
70                                  JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
71                                  JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
72      containerCurent.add("Center", scrollPane);
73      outTextGrafic.setEditable (false);
74      // Camp de text editabil de intrare
75      inTextGrafic = new JTextField(40);
76      containerCurent.add("South", inTextGrafic);
77
78      // Variabila locala finala (folosita in clasa interna anonima de tip
79      // ActionListener)
80      final DataOutputStream outR = this.outRetea;
81
82      // Crearea unui "ascultator" de "evenimente actionare"
83      ActionListener ascultatorInText = new ActionListener() {
84
85        // Tratarea actionarii intrarii de text (apasarea tastei "Enter")
86        public void actionPerformed(ActionEvent ev) {
87
88          // Citire mesajului din intrarea de text
89          String intrare = inTextGrafic.getText();
90
91          try {
92            // Scrierea mesajului pe fluxul de iesire spre retea
93            outR.writeUTF(numeUser + " [" + adresaLocala + ":" + portLocal +
94                          "]> " + intrare);
95            // Fortarea trimiterii mesajului (fortarea golirii bufferului)
96            outR.flush ();
97          }
98          // In cazul unei erori legata de conexiune
99          catch (IOException ex) {
00            // Afisarea exceptiei
01            ex.printStackTrace();
02            // Inchiderea firului de executie care efectueaza receptia
03            firReceptie.stop ();
04          }
05          // Pregatirea intrarii de text pentru noul mesaj (golirea intrarii)
06          inTextGrafic.setText ("");
07        }
08      };
09
10      // Inregistrarea "ascultatorului" de "evenimente actionare" la
11      // "obiectul sursa" intrare text
12      inTextGrafic.addActionListener(ascultatorInText);
13
14      // Crearea unui "adaptor pentru ascultator" de "evenimente fereastra"
15      WindowAdapter ascultatorInchidere = new WindowAdapter() {
16
17        // Tratarea inchiderii ferestrei curente
18        public void windowClosing(WindowEvent ev) {
19
20          // Daca mai exista firul de executie de receptie
21          if (firReceptie != null) {
22            // Inchiderea firului de receptie
23            firReceptie.stop ();
24          }
25          // Terminarea programului
26          System.exit(0);
27        }
28      };
29
30      // Inregistrarea "ascultatorului" de "evenimente fereastra" la "sursa"
31      // (fereastra curenta)
32      this.addWindowListener(ascultatorInchidere);
33
34      // Impachetarea (compactarea) componentelor in container
35      pack();
36      // Fereastra devine vizibila - echivalent cu frame.setVisible(true)
37      show();
38      // Cerere focus pe intrarea de text din fereastra curenta
39      inTextGrafic.requestFocus();
```

```
40
41      // Fir de executie pentru receptia mesajelor de la server
42      firReceptie = new Thread (this);
43      // Lansarea firului de executie - se executa run()
44      firReceptie.start ();
45    }
46
47    /**
48     * Metoda principala a firului care receptioneaza mesaje de la server
49     */
50    public void run() {
51
52      try {
53        // Obtinerea obiectului care incapsuleaza proprietatile sistemului de
54        // operare (obtinerea variabilelor mediului de executie)
55        proprietatiSistem = System.getProperties();
56
57        // Obtinerea valorii variabilei de mediu "USER"
58        numeUser = proprietatiSistem.getProperty("user.name");
59
60        // Obtinerea adresei locale ca obiect InetAddress
61        localHost = InetAddress.getLocalHost();
62
63        // Obtinerea formei String a adresei locale
64        adresaLocala = localHost.getHostAddress();
65
66        // Obtinerea numarului de port local
67        portLocal = socket.getLocalPort();
68
69        // Tratarea mesajelor serverului (citirea si interpretarea lor)
70        while (true) {
71
72          // Citirea mesajului din fluxul de intrare dinspre server
73          String line = inRetea.readUTF ();
74
75          // Adaugarea textului primit in iesirea de text
76          outTextGrafic.append (line + "\n");
77        }
78      }
79
80      // In cazul unei erori legata de conexiune
81      catch (IOException ex) {
82        // Afisarea exceptiei
83        ex.printStackTrace ();
84      }
85
86      // Curatenie finala
87      finally {
88
89        // Inchiderea firului de receptie curent
90        firReceptie = null;
91
92        // Ascunderea intrarii de text
93        inTextGrafic.setVisible(false);
94
95        // Reasezarea interfetei grafice
96        validate ();
97        try {
98          // Inchiderea fluxului de iesire spre retea
99          outRetea.close ();
00          }
01        // In cazul unei erori legata de conexiune
02        catch (IOException ex) {
03          // Afisarea exceptiei
04          ex.printStackTrace ();
05        }
06      }
07    }
08
09
10
11
12
```

```
13    /**
14     * Metoda principala - creaza socketul, fluxurile si lanseaza clientul
15     * @param args[]
16     * @param args
17     * @throws java.io.IOException
18     */
19    public static void main(java.lang.String[] args) throws IOException {
20
21      if (args.length != 2)
22        throw new RuntimeException ("Sintaxa: ClientDeChat_v1 <host> <port>");
23
24      // Adresa serverului - primul parametru primit din linia de comanda
25      String adresaServer = args[0];
26
27      // Portul serverului - al doilea parametru primit din linia de comanda
28      int portServer = Integer.parseInt (args[1]);
29
30      // Crearea socketului catre server
31      socket = new Socket (adresaServer, portServer);
32
33      System.out.println ("Client TCP lansat catre server [" +
34                      socket.getInetAddress() + ":" + socket.getPort() + "]...");
35      System.out.println ("pe portul local: " + socket.getLocalPort());
36
37      // Crearea fluxurilor, crearea si lansarea clientului grafic
38      new ClientDeChat_v1 ("Client pe portul " + socket.getLocalPort() +
39                           " pentru serverul [" + args[0] + ":" + args[1] + "]",
40                       socket.getInputStream (), socket.getOutputStream ());
41    }
42  }
```

Codul sursa al clientului de *chat* care poate **obtinut prin generare automata din diagrama de clase**:

```
 1  //Source file: ClientDeChat_v1.java (cod generat din diagrama UML)
 2
 3  import java.net.*;
 4  import java.io.*;
 5  import java.util.*;
 6  import java.awt.*;
 7  import java.awt.event.*;
 8  import javax.swing.*;
 9  import java.util.Properties;
10
11  /**
12   * Client de chat simplu - aplicatie de sine statatoare
13   * Aplicatie grafica Swing (extinde JFrame)
14   * care poate lansa in executie un fir nou (implementeaza Runnable)
15   */
16  public class ClientDeChat_v1 extends JFrame implements Runnable {
17    private Properties proprietatiSistem;
18    private String numeUser;
19    private String adresaLocala;
20    private int portLocal;
21
22    /**
23     * Flux de intrare dinspre retea
24     */
25    private DataInputStream inRetea;
26
27    /**
28     * Zona de text (configurata ca non-editabila)
29     */
30    private JTextArea outTextGrafic;
31
32    /**
33     * Intrare de text (editabila)
34     */
35    private JTextField inTextGrafic;
36
```

```
37    /**
38     * Fir de executie
39     */
40    private Thread firReceptie;
41    /**
42     * Socket flux (TCP)
43     */
44    private static Socket socket;
45
46    /**
47     * Flux de iesire catre retea
48     */
49    private DataOutputStream outRetea;
50    private InetAddress localHost;
51
52    /**
53     * Initializeaza obiectul de tip ClientDeChat_v1
54     * @param title      Titlul ferestrei
55     * @param inRetea    Flux de intrare dinspre retea
56     * @param outRetea   Flux de iesire catre retea
57     */
58    public ClientDeChat_v1(String title,InputStream inRetea,OutputStream outRetea) {
59
60    }
61
62    /**
63     * Metoda principala a firului care receptioneaza mesaje de la server
64     */
65    public void run() {
66
67    }
68
69    /**
70     * Metoda principala - creaza socketul, fluxurile si lanseaza clientul
71     * @param args[]
72     * @param args
73     * @throws java.io.IOException
74     */
75    public static void main(java.lang.String[] args) throws IOException {
76
77    }
78  }
```

---

sus

Codul sursa al serverului de *chat* **(versiunea 1)**:

(rulat local cu **fisierul *batch***)

```
 1  import java.net.*;          // ServerDeChat_v1.java
 2  import java.io.*;
 3  import java.util.*;
 4
 5  /**
 6   * Server chat simplu - componenta server pentru noi conexiuni
 7   */
 8  public class ServerDeChat_v1 {
 9    private TratareClientDeChat_v1 firTratare;
10    private Socket socket;
11
12    /**
13     * @param port
14     * @throws java.io.IOException
15     */
16    public ServerDeChat_v1(int port) throws IOException {
17
18      // Server pentru asteptarea cererilor de conectare
19      ServerSocket server = new ServerSocket (port);
```

```
20      System.out.println ("Server TCP lansat pe port " + port  + "...");
21
22      // Bucla infinita
23      while (true) {
24
25        // Asteptarea cererilor de conectare si returnarea unui nou socket
26        Socket client = server.accept ();
27        System.out.println ("Acceptata conexiunea de la: [" +
28                    client.getInetAddress() + ":" + client.getPort() + "]...");
29        System.out.println ("pe portul local: " + client.getLocalPort());
30
31        // Crearea unui fir de executie pentru tratare client nou
32        firTratare = new TratareClientDeChat_v1 (client);
33
34        // Lansarea firului de executie - se va executa: firTratare.run()
35        firTratare.start ();
36       }
37    }
38
39    /**
40     * @param args
41     * @throws java.io.IOException
42     */
43    public static void main(java.lang.String[] args) throws IOException {
44
45      if (args.length != 1)
46        throw new RuntimeException ("Sintaxa: ServerDeChat_v1 <numarPort>");
47
48      new ServerDeChat_v1 (Integer.parseInt (args[0]));
49    }
50  }
```

Codul sursa al serverului de *chat* care poate **obtinut prin generare automata din diagrama de clase**:

```
 1  //Source file:  ServerDeChat_v1.java
 2
 3  /**
 4   * Server chat simplu - componenta server pentru noi conexiuni
 5   */
 6  import java.net.*;
 7  import java.io.*;
 8  import java.util.*;
 9
10  public class ServerDeChat_v1 {
11    private TratareClientDeChat_v1 firTratare;
12    private Socket socket;
13
14    /**
15     * @param port
16     * @throws java.io.IOException
17     */
18    public ServerDeChat_v1(int port) throws IOException {
19
20    }
21
22    /**
23     * @param args
24     * @throws java.io.IOException
25     */
26    public static void main(java.lang.String[] args) throws IOException {
27    }
28  }
```

---

sus

Codul sursa al firului de tratare al clientului de *chat* (versiunea 1):

```
 1  import java.net.*; // TratareClientTextChat_v1.java
 2  import java.io.*;
 3  import java.util.*;
 4  import java.util.Vector;
 5
 6  /**
 7   * Server chat simplu - componenta de tratare a unei conexiuni
 8   */
 9  public class TratareClientTextChat_v1 extends Thread {
10
11    /**
12     * Vector de referinte la obiecte care trateaza clienti (ptr. inregistrare)
13     */
14    private static Vector listaFire = new Vector ();
15
16    /**
17     * Socket flux (TCP)
18     */
19    private Socket socket;
20
21    /**
22     * Flux de intrare dinspre retea
23     */
24    private DataInputStream inRetea;
25
26    /**
27     * Flux de iesire catre retea
28     */
29    private DataOutputStream outRetea;
30
31    /**
32     * Initializeaza obiectul (firul) care trateaza un nou client
33     * @param socket
34     * @throws java.io.IOException
35     */
36    public TratareClientTextChat_v1(Socket socket) throws IOException {
37      this.socket = socket;
38      inRetea = new DataInputStream (new
39                          BufferedInputStream (socket.getInputStream ()));
40      outRetea = new DataOutputStream (new
41                          BufferedOutputStream (socket.getOutputStream ()));
42    }
43
44    /**
45     * Metoda principala a firului de executie.
46     * Primeste mesajele si apeleaza difuzarea lor.
47     */
48    public void run() {
49      try {
50        // Inregistrarea firului curent in lista (Vector)
51        listaFire.addElement (this);
52
53        System.out.print("\nNou fir de executie ...");
54        System.out.println(this.toString());
55        System.out.println(listaFire.toString()+"\n");
56
57        // Tratarea mesajelor clientului (citirea si difuzarea mesajelor)
58        while (true) {
59
60          // Citirea mesajului din fluxul de intrare de la client
61          String mesaj = inRetea.readUTF ();
62
63          // Difuzarea catre toti clientii curent inregistrati
64          difuzare (mesaj);
65        }
66      }
67
68      // In cazul unei erori legata de conexiune
69      catch (IOException ex) {
70        // Afisare exceptie
```

```
71        ex.printStackTrace ();
72      }
73
74      // Curatenie finala
75      finally {
76        // Eliminarea firului curent din lista (Vector)
77        listaFire.removeElement (this);
78
79        System.out.print("\nFir de executie eliminat...");
80        System.out.println(this.toString());
81        System.out.println(listaFire.toString()+"\n");
82
83        try {
84          // Inchiderea socketului
85          socket.close ();
86        }
87        // In cazul unei erori legata de conexiune
88        catch (IOException ex) {
89          // Afisarea exceptiei
90          ex.printStackTrace ();
91        }
92      }
93    }
94
95    /**
96     * Difuzeaza mesajul primit catre clienti
97     * @param mesaj
98     */
99    private static void difuzare(String mesaj) {
00
01      // Enumerare creata pornind de la lista firelor de executie
02      Enumeration enum = listaFire.elements ();
03
04      // Cat timp mai sunt elemente in enumerare
05      while (enum.hasMoreElements ()) {
06
07        // Referinta catre firul curent initializata cu null
08        TratareClientTextChat_v1 firDestinatie = null;
09
10        // Protectie la acces concurent la Vectorul firelor
11        synchronized (listaFire) {
12
13          // Obtinerea referintei catre firul curent
14          firDestinatie = (TratareClientTextChat_v1) enum.nextElement ();
15        }
16
17        // Daca referinta e valida
18        if (firDestinatie  != null) {
19
20          try {
21           // Protectie la acces concurent la fluxul de iesire
22           synchronized (firDestinatie.outRetea) {
23
24              // Scrierea mesajului in fluxul de iesire al firului curent
25              firDestinatie.outRetea.writeUTF (mesaj);
26            }
27
28            // Fortarea trimiterii mesajului
29            firDestinatie.outRetea.flush ();
30          }
31
32          // In cazul unei erori legata de conexiune
33          catch (IOException ex) {
34            // Inchiderea firului curent
35            firDestinatie.stop ();
36          }
37        }
38      }
39    }
40  }
```

Codul sursa al firului de tratare al clientului de *chat* care poate **obtinut prin generare automata din diagrama de clase**:

```
1  //Source file: TratareClientDeChat_v1.java
2
3  /**
4   * Server chat simplu - componenta de tratare a unei conexiuni
5   */
6  import java.net.*;
7  import java.io.*;
8  import java.util.*;
9  import java.util.Vector;
10
11 public class TratareClientDeChat_v1 extends Thread {
12
13   /**
14    * Vector de referinte la obiecte care trateaza clienti (ptr. inregistrare)
15    */
16   private static Vector listaFire = new Vector ();
17
18   /**
19    * Socket flux (TCP)
20    */
21   private Socket socket;
22
23   /**
24    * Flux de intrare dinspre retea
25    */
26   private DataInputStream inRetea;
27
28   /**
29    * Flux de iesire catre retea
30    */
31   private DataOutputStream outRetea;
32
33   /**
34    * Initializeaza obiectul (firul) care trateaza un nou client
35    * @param socket
36    * @throws java.io.IOException
37    */
38   public TratareClientDeChat_v1(Socket socket) throws IOException {
39
40   }
41
42   /**
43    * Metoda principala a firului de executie.
44    * Primeste mesajele si apeleaza difuzarea lor.
45    */
46   public void run() {
47
48   }
49
50   /**
51    * Difuzeaza mesajul primit catre clienti
52    * @param mesaj
53    */
54   private static void difuzare(String mesaj) {
55
56   }
57 }
```