7. Implementarea sistemelor distribuite (SD)

7.1. Modele SD

Modificarea de la distanță a parametrilor și algoritmilor

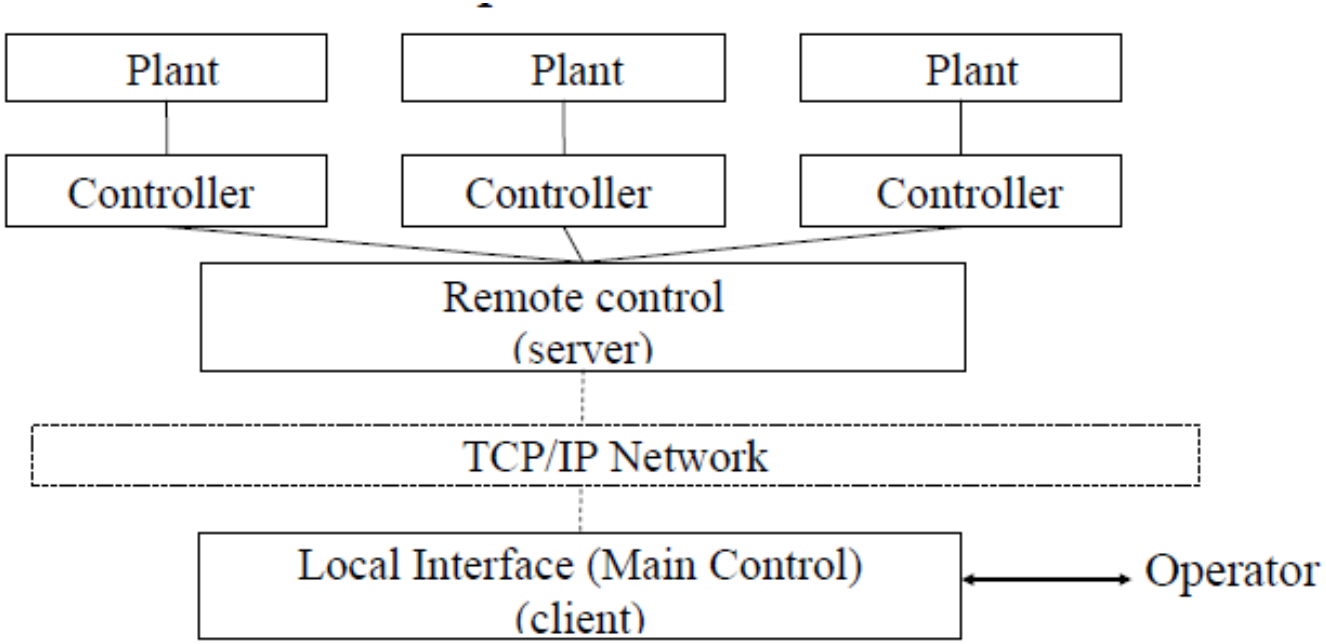


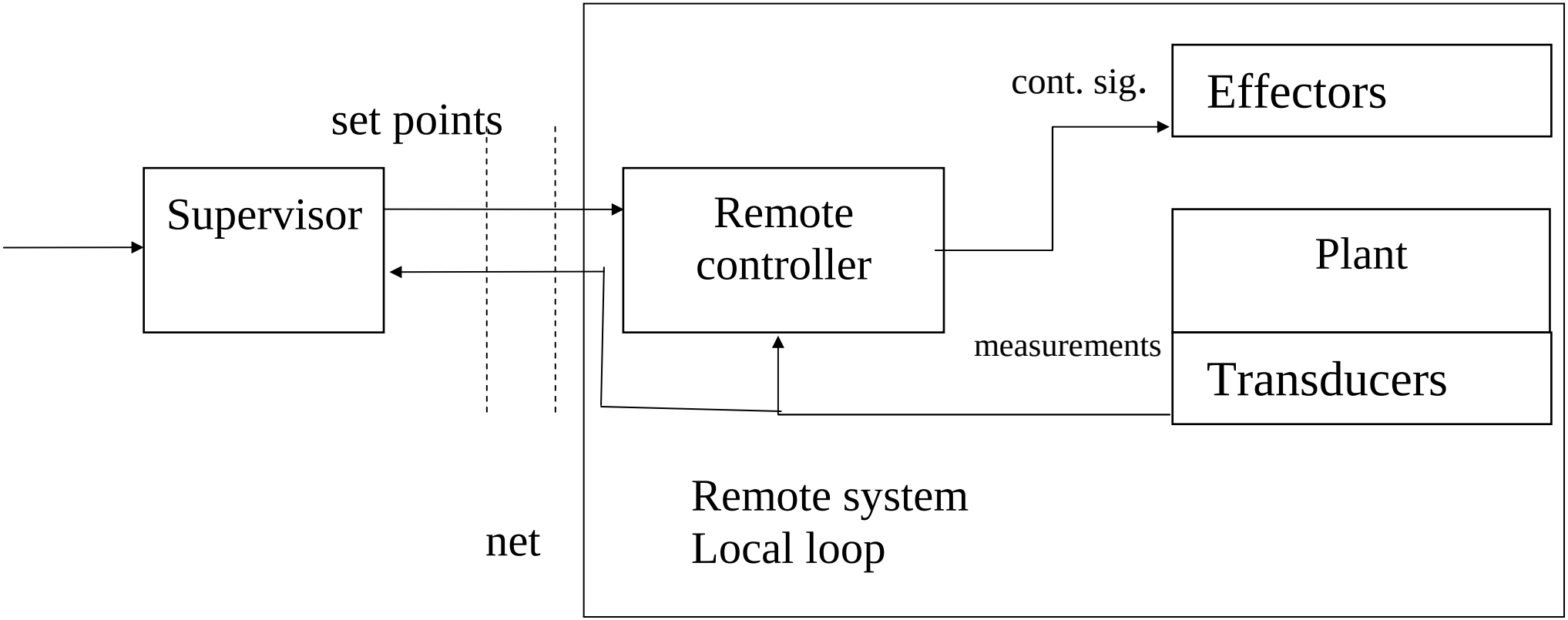
Fig. 7.17. Arhitectura generală a aplicaţiei RMI

Temă:

1. Construiți diagrama componentelor. Adăugați toate metodele necesare.

2. Construiti diagrama componentelor aferentă FLETPN pentru aplicația RMI.

Fig. 10.1. Sisteme de control direct.



Arhitecturi concurente

Arhitecturi Client-server

Peer-to-peer

Alte architecturi:

- Filter Pipeline = Conducte pentru filtrare
- Supervisor-Worker = Supervizor-muncitor
- Announcer-Listener = Crainic-ascultător

1) Filter Pipeline = Conducte cu filtrare

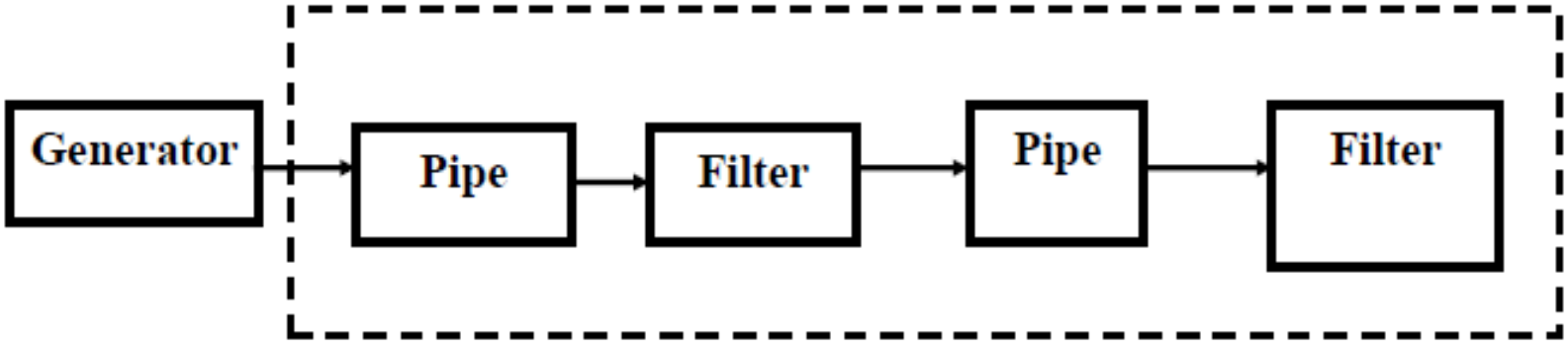
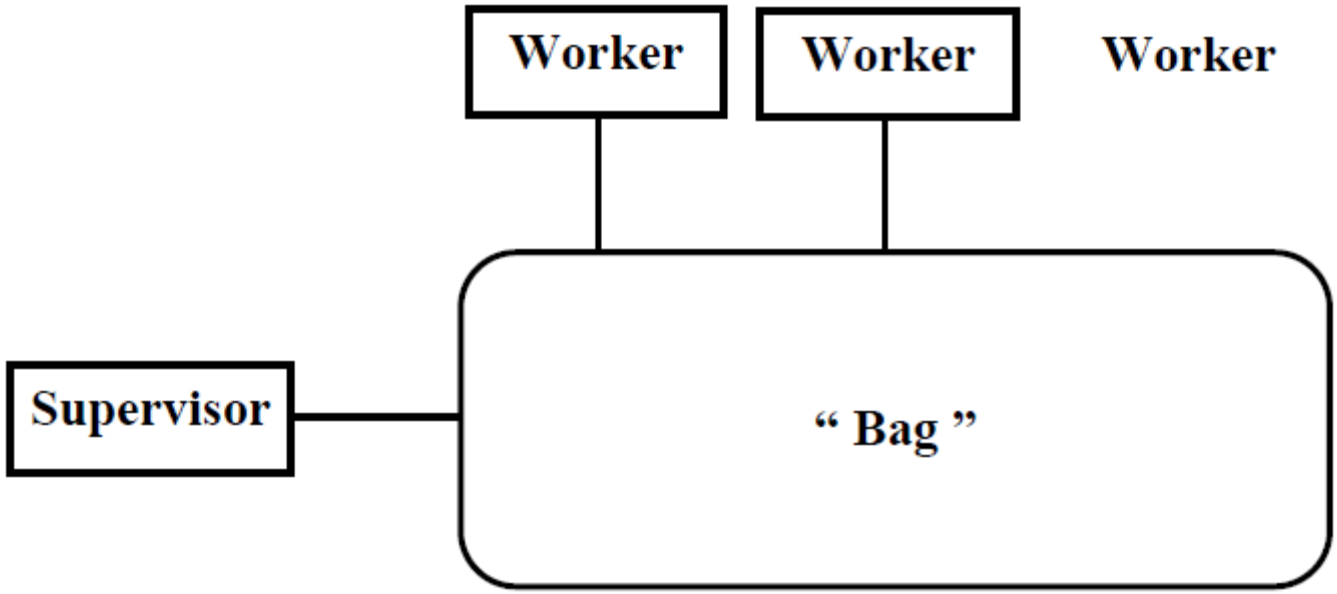
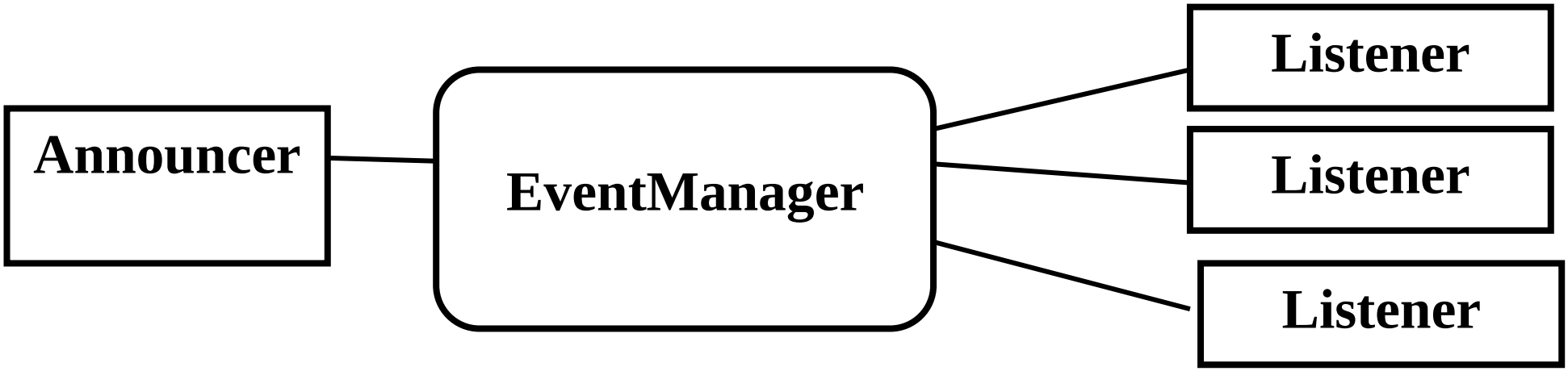


Image processing

2) Supervisor-Worker = Supervizor-muncitor



3) Announcer-Listener = Crainic-ascultător



Structură bazată pe evenimente

Crainicul notifică un eveniment.

Ascultătorul alege să reacționeze la unul sau mai multe evenimente pe baza unui registru.

Poate fi implementat ***recursiv***. Construiți o structură recursivă a acestei metode.

Crainicul nu știe numărul de ascultători, el este complet izolat.

Numele mecanismului este **invocare implicită**.

Modele de tip mainframe (Mainframe models)

Toată logica aplicației într-un singur calculator.

Modele de tip file sharing (File sharing models)

- Rețele de calculatoare cu servere
- Fișierele shared sunt actualizate pe servere de
- Calculatoarele client (client workstations) – clienții execută toate operațiunile

Acest model funcționează pentru trafic redus.

Cloud computing

Balansarea procesării (Processing balance)

Modelul Client – Server

- presupune existenţa unui server care realizează prelucrări de date în numele clientului returnând către acesta doar rezultatele.

Ex.- implementarea la nivelului serverului a sistemelor de gestiune a bazelor de date în cadrul cărora se realizează operaţiile de interogare/actualizare a datelor transmiţând clientului doar rezultatele acestora.

Comunicația Client-Server:

- Protocoale SQL (pentru interogări şi actualizări ale bazelor de date)
- RMI = Remote Method Invocation ← Java
- RPC = Remote Procedure Call ← C; remote function call.

- Server secvențial (monofir)
- Server multifir

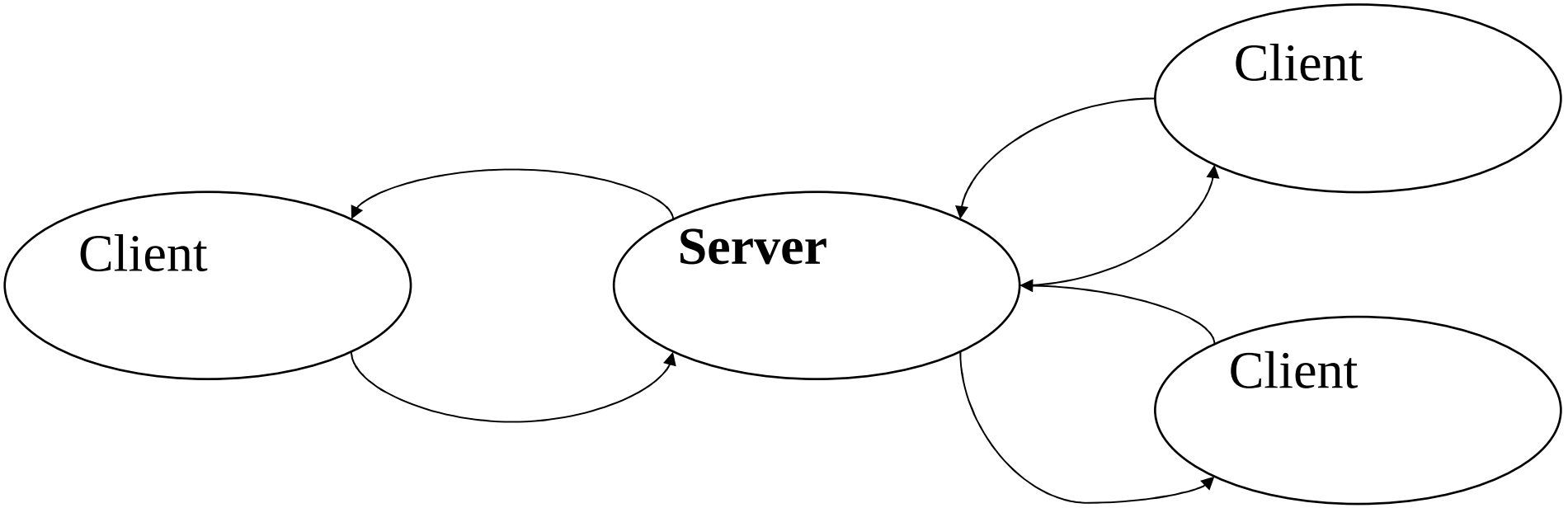


Fig. 7.1. **Modelul Client Server**

Probleme:

- securitate,
- acces simultan,
- localizare,
- sincronizare și
- comunicația dintre module.

Modelul *peer-to-peer*

Fiecare peer joacă simultan rolul clientului și al serverului.
Toate nodurile sunt egale.

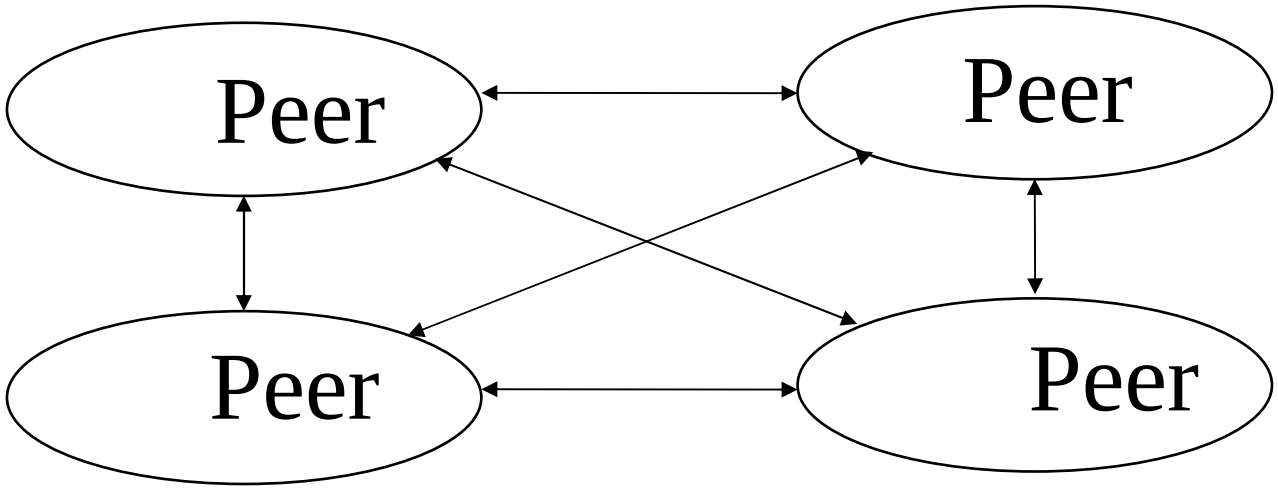


Fig. 7.2. Arhitectura peer to peer

Sistemele multiagent folosesc această arhitectură.

Modelul Mobile code– cod executabil transmis de client

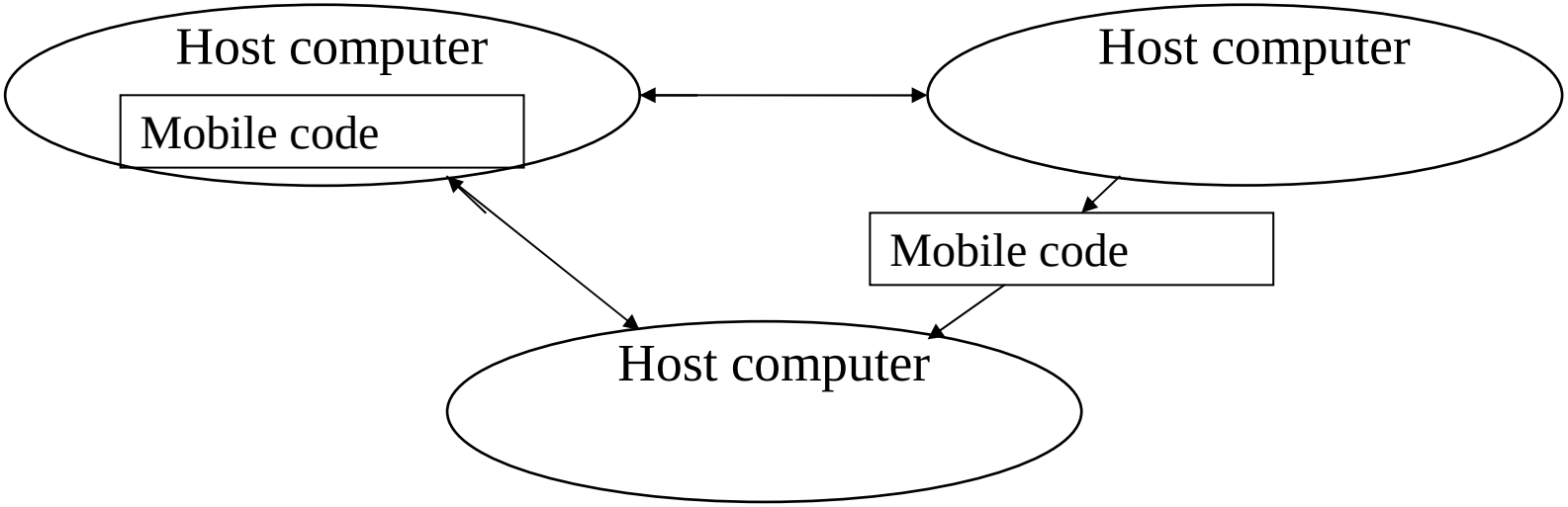
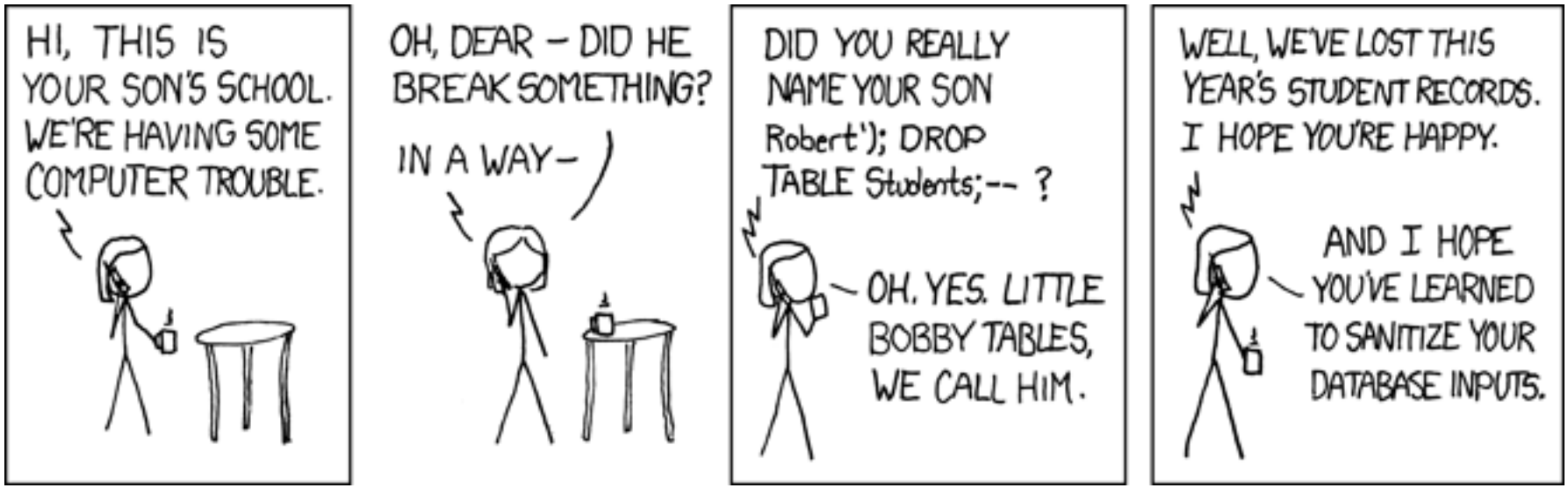


Fig. 7.3. Migrarea codului mobil

Se folosește în sistemele multiagent software ➔ AGENT software MOBIL
Ex: biblioteca - interogare
Justificare Java

Taxonomia codului mobil

- applets - downloadable applications - aplicaţii descărcabile
- servlets – up-loadable services - servicii încărcabile
- extlets – up-loadable & downloadable - trăsături încărcabile şi descărcabile
- deglets – delegated tasks - taskuri delegate
- netlets – autonomous tasks - taskuri autonome
- piglets – malevolent mobile code - coduri mobile răuvoitoare



Principalele avantaje ale agenţilor software mobili sunt:

- reducerea încărcării reţelei prin deplasarea codului executabil în locaţiile unde se găsesc datele
- toleranţa la defecte, agenţii având capacitatea de a rula fără a exista o conexiune activă permanentă cu reţeaua
- flexibilitatea întreţinerii şi dezvoltării

Tehnologia agenţilor software îşi găseşte aplicaţie în: managementul reţelelor, monitorizarea resurselor, culegerea de date, controlul proceselor tehnice distribuite etc.

7.2. Internet

Internet-ul este o colecţie de reţele interconectate între ele prin intermediul unor dispozitive de reţea (rutere, switch-uri, hub-uri) care funcţionează ca o singură reţea.

Reţelele de tip LAN (acronim de la Local Area Network) s-au dezvoltat odată cu apariţia calculatoarelor personale (PC).

Principalele tehnologii folosite în cadrul reţelelor de tip LAN sunt: Ethernet, Token Ring şi FDDI.

Specificată în standardul IEEE 802.3, tehnologia Ethernet este cea mai des folosită în cadrul reţelelor LAN. Dispozitivele sunt interconectate între ele prin echipamente de reţea şi sunt în competiţie pentru obţinerea resurselor de reţea.

Se foloseşte protocolul CSMA/CD (acronim de la Carrier Sense Multiple Access/Collision Detection).

La reţelele de tip Token Ring, calculatoarele sunt legate într-o topologie de tip stea sau inel. Este folosită o schemă de transmisie a datelor bazată pe un jeton pentru a evita apariţia coliziunilor. Funcţionarea acestui tip de reţea este descrisă prin protocoalele IBM Token Ring şi IEEE 802.5.

FDDI (acronim de la Fiber Distributed Data Interface) este un set de standarde pentru transmisia de date pe fibră optică. Acest protocol are la bază protocolul Token Ring. În mod uzual FDDI este folosit ca şi mijloc de transport al datelor între reţele interconectate la distanţă (WAN).

Reţelele de tip WAN (acronim de la Wide Area Network) interconectează în cadrul lor mai multe reţele LAN separate geografic. Câteva dintre tehnologiile folosite pentru interconectarea reţelelor de tip LAN sunt: T1, T3, ATM, ISDN şi ADSL.

Modelul OSI\ISO

Principiile de comunicaţie în cadrul Internet-ului au la bază modelul *Open System Interconnection* (cu acronimul OSI) dezvoltat de către International Organisation for Standardization (ISO) în 1984. ➔ din punct de vedere funcţional are 7 niveluri.

Nivelul fizic
Nivelul legăturilor de date
Nivelul reţea.
Nivelul de transport
Nivelul sesiune
Nivelul prezentare
Nivelul aplicaţie

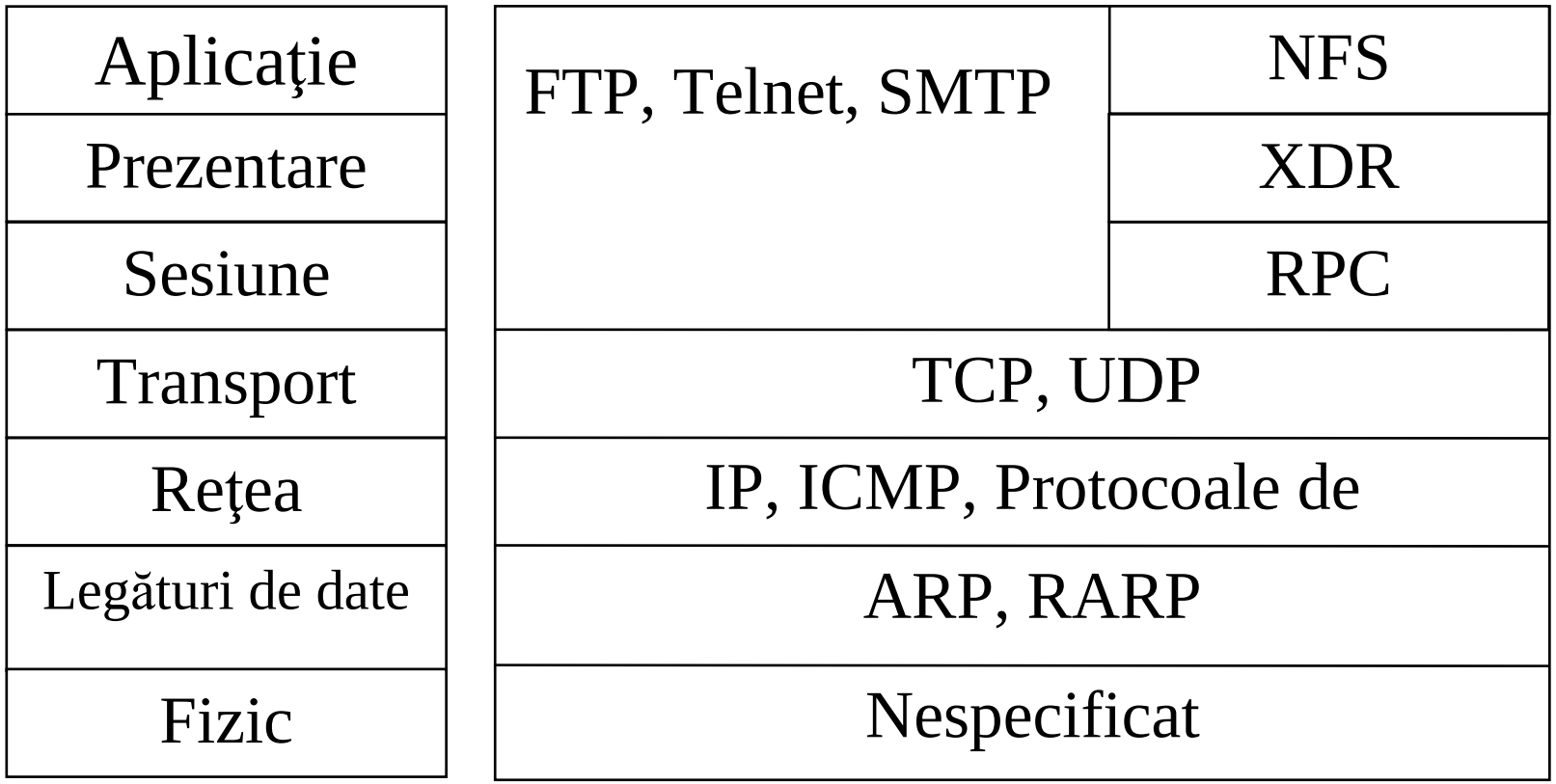


Fig. 7.4. Asocierea dintre nivelurile OSI şi protocoale Internet

3. Protocoale de comunicaţie

Protocolul IP – are informaţii despre adresele calculatoarelor şi pentru rutare

- asigură livrarea datagramelor la destinaţie şi
- asigură fragmentarea şi defragmentarea datagramelor pentru a susţine transmisia pe medii cu diferite capacităţi de transfer

Protocolul TCP - acronim de la Transmission Control Protocol

Calculatoarele se sincronizează între ele – este similar cu o **convorbire telefonică** – orientat pe conexiune directă

Nu se pierd mesaje – se garantează livrarea mesajelor – dacă trebuie se retransmite mesajul

Se păstrează ordinea mesajelor

Durează mai mult decât cu UDP

Protocolul UDP - acronim de la User Datagram Protocol
Nu este orientat pe conexiune – este ca la **trimiterea scrisorilor**
Se pot pierde mesaje – nu se garantează livrarea lor
Mesajele pot sosi într-o ordine diferită de cea în care au fost transmise

Acest protocol este folosit de aplicaţii de nivel înalt cum ar fi:

- Network File System (NFS),
- Simple Network Management Protocol (SNMP),
- Domain Name System (DNS) şi
- Trivial File Transfer Protocol (TFTP).

Socluri (engl.: sockets).
Soclurile sunt asociate porturilor care sunt capetele canalelor de comunicaţie. Se poate folosi o interfaţă de reţea de către mai multe aplicaţii, fiecare dintre ele având asociat câte un **port**.

Port este o adresă într-un calculator care este asociată cu un protocol de aplicaţie particular. Există unele porturi asociate cu unele protocoale:

- 21 - pentru File Transfer Protocol (FTP)
- 23 - pentru Telnet Protocol
- 25 - pentru Simple Mail Transfer Protocol (SMTP)
- 80 - pentru HyperText Transfer Protocol (HTTP)

Comunicarea prin TCP

Conexiunile sunt legături de comunicaţie create pe baza protocolului TCP.
TCP/IP este unul dintre cele mai utilizate protocoale de comunicaţie în reţea. Caracteristicile principale ale protocolului TCP sunt:

- Transferul de date se face în ***flux continuu***.
- Se asigură ***corectitudinea transmisiei***, adică recuperarea pachetelor transmise cu erori, eliminarea duplicatelor sau a pachetelor cu numere de secvenţă eronate.
- Se permite ***multiplexarea***, adică mai multe procese care sunt rulate simultan într-un nod pot să-i folosească serviciile virtual în acelaşi timp.
- Se realizează un circuit virtual prin controlul fluxului conexiunii.

Pentru realizarea comunicaţiei există definite în pachetele java mai multe clase care servesc acestui scop.

Clasa InetAddress

Clasa InetAddress încapsulează adrese InternetIP şi permite conversia adresei zecimale cu punct sau nume de gazdă (hostname) în adresă IP
Clasa InetAddress are metodele:

- *public static InetAddress getLocalHost() throws UnknownHostException –*
- *public static synchronized InetAddress getByName(String host) throws UnknownHostException*
- *public static synchronized InetAddress[] getAllByName(String host) –*
- *public String getHostName()*
- *public byte[] getAddress()*
- *public String getHostAddress()*

Clasa ServerSocket

Constructorii clasei sunt:

- *public ServerSocket(int port) throws IOException* – creează un soclu de server pe portul specificat (un număr între 0 şi 65535) cu lungimea cozii de 50.
- *public ServerSocket(int port, int backlog) throws IOException* – idem mai sus cu backlog specificând lungimea cozii.
- *public ServerSocket(int port, int backlog, InetAddress bindAddress) throws IOException* – idem mai sus cu bindAddress o adresă locală (de Internet) la care serverul va aştepta să fie conectat.

Printre metodele clasei ServerSocket sunt:

- *public Socket* ***accept()*** *throws IOException* – realizează şi acceptă o conexiune pe soclu
- *public void close() throws IOException* – închide un soclu
- *public InetAddress getInetAddress() throws IOException* – returnează adresa la care este conectat soclul, sau null dacă soclul încă nu este conectat
- *public int getLocalPort()* – returnează numărul portului la care ascultă soclul
- *public synchronized void setSoTimeout(int timeout) throws SocketException* – permite/elimină timpul limită de aşteptare a legăturii (engl.: timeout) specificat în milisecunde
- *public synchronized int getSoTimeout(int timeout) throws SocketException* –furnizează timpul limită de aşteptare al legăturii (engl.: timeout) specificat în milisecunde – dacă este setat la zero aceasta semnifică aşteptarea la infinit pentru comunicare.

Crearea unui soclu de server se face astfel:

ServerSocket serverSocket=new ServerSocket(12345,7);

Primul parametru reprezintă portul, iar al doilea (opţional) specifică lungimea stivei de ascultare. Stiva de ascultare este o coadă a cererilor care nu au fost onorate până la un moment dat.

Clasa Socket

Printre constructorii clasei sunt:

- *protected Socket()* – creează un soclu neconectat.
- *public Socket(String host, int port) throws UnknownHostException, IOException* –
- *public Socket(InetAddress address, int port) throws IOException* – cu *adresă*,

Metode din clasa Socket:

- *public synchronized void close() throws IOException* – închide soclul.
- *public InetAddress* ***getInetAddress()*** – întoarce adresa IP a calculatorului aflat la distanţă şi la care este conectat soclul.
- *public InputStream* ***getInputStream()*** *throws IOException* – întoarce un flux de intrare pentru citire de octeţi din soclu.
- *public OutputStream* ***getOutputStream()*** *throws IOException* – întoarce un flux de ieşire pentru scriere de octeţi în soclu.
- *public synchronized void setSoTimeout(int timeout) throws SocketException* – permite/anulează un timp de aşteptare specificat în milisecunde. O operaţie de citire (read()) din fluxul de intrare asociat soclului va pune firul în aşteptare pentru realizarea acesteia maximum *timeout* milisecunde. Dacă citirea nu se poate realiza în acea perioadă de timp se semnalează o excepţie java.io.InterruptedException (timeout=0 este interpretat ca un timp de aşteptare infinit)
- *public String toString()* – întoarce un şir de caractere reprezentând acest soclu.

Pentru transmiterea şi recepţionarea de informaţii se utilizează metodele *getInputStream()* şi *getOutputStream()* care returnează fluxurile necesare.

Schema de implementare client-server

Există mai multe tipuri de programe client:

- *programe de E-mail* - furnizează o interfaţă cu care se creează, trimit, recepţionează sau afişează mesaje (mail-uri)
- *browser-e de Web* - furnizează o fereastră pentru WWW care permite afişarea de pagini de Web, miniaplicaţii Java etc.
- *programe FTP* - transmit fişiere de la serveri de fişiere din Internat sau din directori de fişiere proprii.

Un server efectuează următoarele:

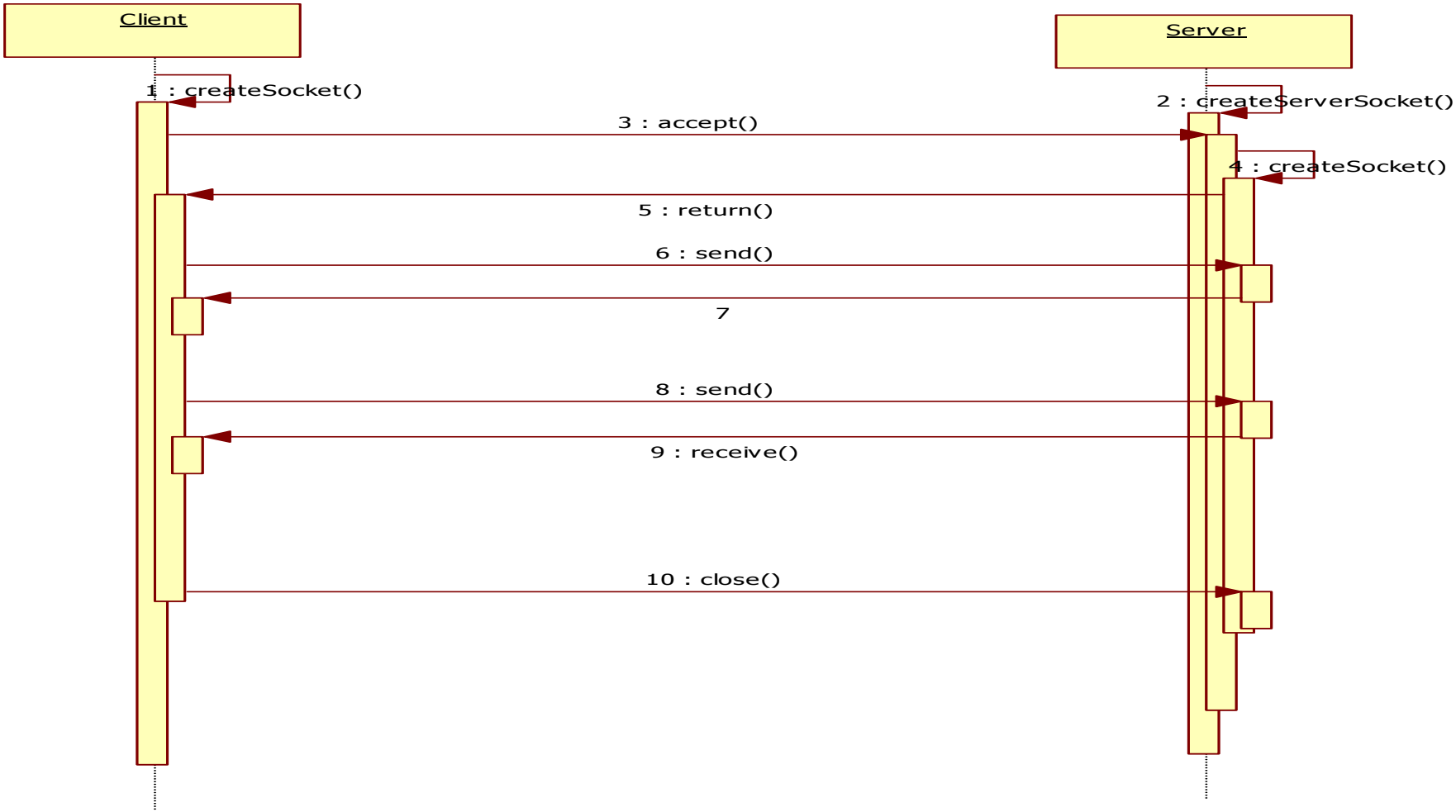
- declară un *socket* – prin care cere nucleului să creeze un soclu cu care poate să comunice cu clienţii – se fixează
  - setul de adrese
  - tipul de soclu folosit (stream sau datagram)
- transmite nucleului (sistemului de operare) numărul de port la care (serverul) aşteaptă să fie contactat
- şi cere nucleului să dimensioneze lungimea cozii de aşteptare la soclu.

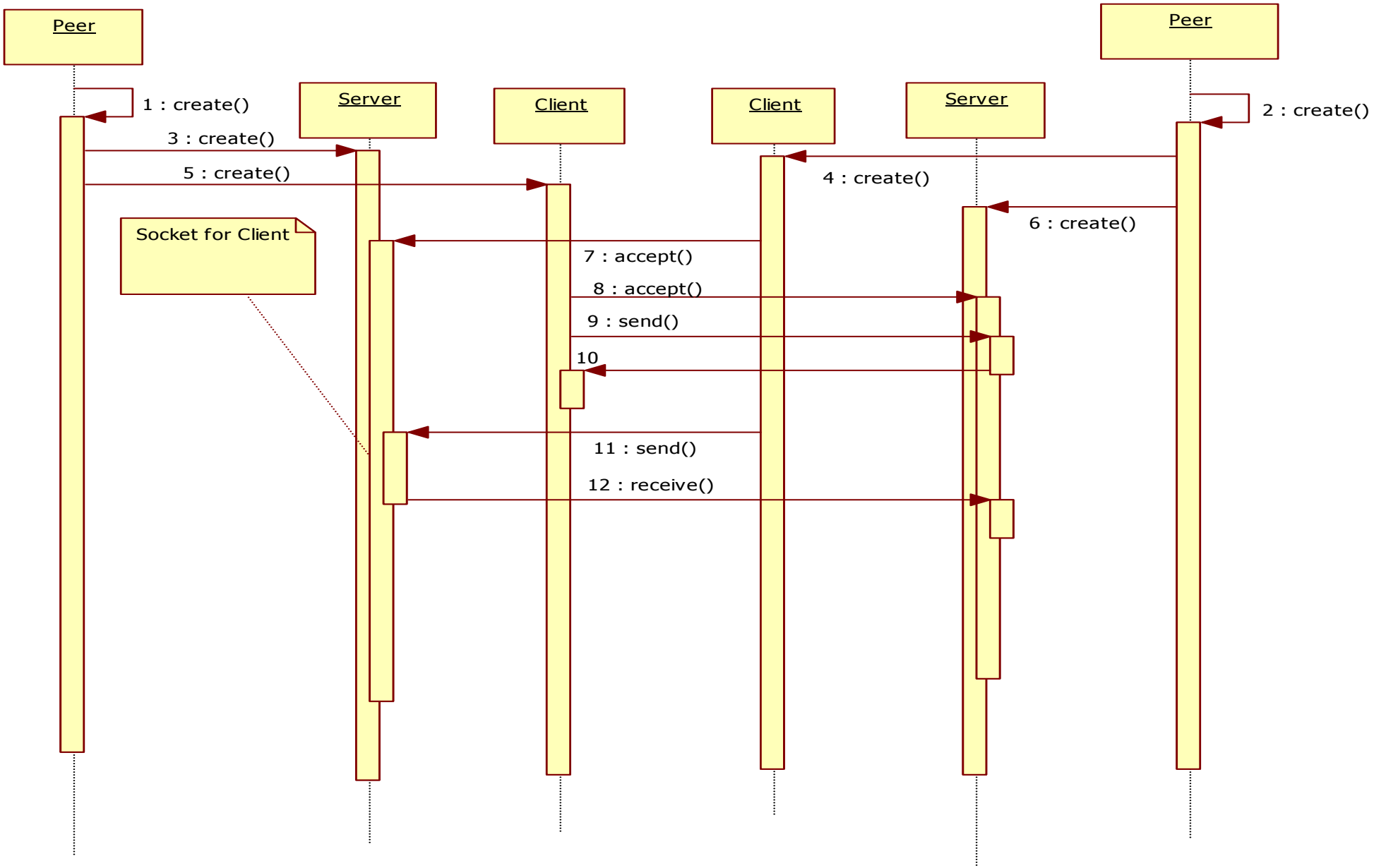
Un client efectuează următoarele:

- declară un *socket* – prin care cere nucleului să creeze un soclu la care poate fi contactat – se fixează setul de adrese şi tipul de soclu folosit (stream sau datagram)
- abordează serverul prin mesaje, iar dacă este cazul aşteaptă răspunsurile corespunzătoare.

Relaţia client-server:

1) *Clientul* execută o funcţie prin care transmite nucleului (lui) adresa IP şi numărul de port ale serverului căruia vrea să-i ceară un serviciu.
2) *Nucleul* contactează serverul solicitat prin mecanismul de soclu.
3) *Serverul* execută metoda *accept* şi rămâne în aşteptarea clientului ca să-l contacteze. La contactare primeşte adresa IP a clientului şi numărul de port al clientului care l-a contactat.
4) *Clientul* execută o transmise (sau mai multe) prin *write* sau *print* cu care comunică serverului un mesaj şi prin care-i cere un serviciu.
5) *Serverul* efectuează o recepţie prin *read* preluând mesajul transmis de (un) client.
6) *Serverul* execută acţiunile transmise (cerute) de client.
7) Dacă este cazul, *serverul* transmite clientului un mesaj executând *write* sau *print* (pe canalul în cauză).
8) Dacă este cazul, *clientul* recepţionează mesajul transmis de server prin *read*.
9) *Clientul* închide legătura (prin *close*) dacă nu mai are nimic de transmis serverului. În caz contrar se reia punctul 4).
10) *Serverul* închide legătura (prin *close*).


Client
Server
1 : createSocket()
2 : createServerSocket()
3 : accept()
4 : createSocket()
5 : return()
6 : send()
7
8 : send()
9 : receive()
10 : close()


Peer
Server
Client
Client
Server
Peer
1 : create()
2 : create()
3 : create()
4 : create()
5 : create()
6 : create()
Socket for Client
7 : accept()
8 : accept()
9 : send()
10
11 : send()
12 : receive()

Aplicaţii client-server

Serverii pot fi monofir sau multifir după cum pot comunica cu un singur client sau cu mai mulţi clienţi.

Serverul monofir

Pentru exemplificarea schemei client-server descrisă mai sus se propune construirea a două programe: server şi client. Serverul poate servi numai un singur client ceea ce permite implementarea lui printr-un singur fir, de unde şi numele lui.

Specificaţia serverului este:

- Deschide o legătură prin soclu pe un port dat ca argument la startarea programului şi aşteaptă să fie contactat de către un (singur) client.
- Deschide fluxuri de intrare şi de ieşire a datelor.
- Până când recepţionează un mesaj "end" face următoarele:
  - recepţionează mesajul
  - scrie mesajul recepţionat pe ecranul monitorului într-o fereastră proprie (a consolei)
  - trimite înapoi mesajul cu un şir de caractere în faţă conţinând numele lui
- Comunică clientului că închide legătura.
- Scrie pe ecran că închide legătura.
- Închide legătura (soclul şi fluxurile de intrare şi de ieşire).

Specificaţia clientului este:

- Deschide o legătură cu un calculator (server) dat prin nume şi un port dat ca număr prin argumentele programului.
- Deschide fluxuri de intrare şi de ieşire a datelor.
- Deschide un flux de intrare a datelor de la consolă.
- Până când nu citeşte de la consolă mesajul "end":
  - citeşte un mesaj de la consolă;
  - transmite mesajul serverului;
  - recepţionează mesajul trimis înapoi de server;
  - scrie mesajul de la server pe ecranul monitorului în fereastra lui;
- Scrie în fereastra lui că închide legătura (canalul de comunicaţie).
- Închide fluxul de intrare a datelor de la consolă.
- Închide legătura (soclu şi fluxurile de intrare şi de ieşire).

Serveri iterativi şi serveri concurenţi

Un server iterativ realizează o conexiune şi o încheie înainte ca s-o realizeze pe următoarea (eventual cu un alt client).

Un server concurent realizează câte un nou fir pentru fiecare cerere nouă de conexiune. El serveşte virtual (aparent) în acelaşi timp mai mulţi clienţi. De obicei serverii comerciali lucrează concurent.

Server multifir

Serverul multifir are un fir care rulează tot timpul metoda *run* (mai precis bucla *for*) aşteptând conexiuni cu alţi clienţi.

Specificaţia serverului multifir este:

- Deschide o legătură prin soclu pe un port dat.
- Aşteaptă la infinit să fie contactat de un client pe portul dat:
  - Dacă este contactat creează o nouă legătură cu clientul pe un nou fir.

Legătura (firul pentru comunicaţie cu clientul) realizează:

- Deschide un flux de intrare de date pe soclul dat.
- Deschide un flux de ieşire de date pe soclul dat.
- Cât timp serverul recepţionează mesaje diferite de "end":
  - recepţionează mesajul clientului;
  - scrie mesajul recepţionat pe ecran în fereastra serverului ;
  - trimite înapoi mesajul cu un şir de caractere în faţă conţinând numele serverului.
- Comunică clientului că închide legătura.
- Scrie în fereastra serverului că închide legătura.
- Închide legătura (soclul şi fluxurile asociate).

Specificaţia clientului este:

- Deschide o legătură cu un calculator (server) dat prin nume şi un port dat ca număr prin argumentele programului.
- Deschide un flux pentru citirea datelor de la consolă.
- Până când nu citeşte de la consolă un mesaj "end":
  - citeşte un mesaj de la consolă;
  - citeşte timpul curent;
  - transmite mesajul serverului;
  - recepţionează mesajul trimis înapoi de server;
  - citeşte timpul curent şi afişează pe ecran durata schimbului de informaţii între client-server-client;
  - scrie mesajul de la server pe ecranul monitorului în fereastra lui.
- Scrie în fereastra lui că închide legătura (canalul de comunicaţie).
- Închide fluxul pentru citirea datelor de la consolă.
- Închide legătura (soclul şi fluxurile asociate lui).

Comunicarea prin datagrame ➔ UDP

Serviciul UDP este unul nefiabil, dar cu viteză de transmisie mare. Nu se asigură certitudinea livrării pachetelor de date şi nu există mecanisme implementate pentru detectarea pierderii sau duplicării pachetelor (adică datagramelor).
Protocolul UDP este un ***serviciu fără conexiune*** în care datagramele sunt transmise ca şi pachete separate.
Mărimea unei datagrame este limitată de numărul de octeţi care pot fi transmişi într-o singură tranzacţie.
În cazul serviciului datagram, expeditorul depune datele la un capăt al canalului, iar acestea sunt preluate din când în când şi sunt furnizate la celălalt capăt de unde pot fi preluate de destinatar.

Clasa pentru crearea soclurilor este *DatagramSocket*.

Un soclu datagram este un punct pentru transmiterea şi recepţionarea pachetelor într-un serviciu fără conexiune. Fiecare pachet este adresat şi rutat individual. Clasa DatagramSocket are constructorii: *DatagramSocket()*, *DatagramSocket(int)* şi *DatagramSocket(int,InetAddress)*. Ultimul constructor creează un soclu pentru datagrame limitat la adresa specificată. Clasa are metodele: *close()*, *getLocalAddress()*, *getLocalPort()*, *getSoTime()*, *receive(DatagramPacket)*, *send(DatagramPacket)* şi *setSoTime(int)*.

Exemple de utilizare sunt:

DatagramSocket serverSocket=new DatagramSocket(1234);

DatagramSocket clientSocket=new DatagramSocket();

Recepţionarea datagram-elor se face cu:

byte[] b=new byte[1024];
DatagramPacket pac=new DatagramPacket(b,b.length);
clientSocket.receive(pac);

Metoda *receive* va bloca firul până când sosesc datele şi acestea sunt disponibile.

Transmiterea datelor se face pe adresa completă obţinută din numele calculatorului gazdă cu una dintre metodele clasei **InetAddress**. Metodele respective sunt statice deci sunt apelate prin intermediul clasei:

InetAddress adr=InetAddress.getByName(String);

Pentru transmiterea pachetului. acesta trebuie mai întâi să fie creat cu:

```
DatagramPacket sendPac=
new DatagramPacket(sendBuf,sendBuf.length,adr,port);
clientSocket.send(sendPac);
```

Se observă că se pun în pachet mesajul, adresa şi portul expeditorului, prin urmare acestea vor putea fi extrase de către destinatar.

Serverul transmite date astfel:

```
DatagramPacket sendPac=new
DatagramPac(sendBuf,sendbuf.length,receivePac.getPort(),
        receivePac.getPort());
serverSocket.send(sendPac);
```

7.4. Comunicația RT

Internet-ul în varianta sa clasică furnizează:

- servicii de date fără restricţii temporale
- servicii de email, transfer de fişiere, WWW, etc.
- unicast – distribuirea la un singur destinatar
- o clasă de servicii singulare

Din cauza nerespectării cerinţelor temporale s-au adăugat Internet-ului aşa numitele ***Serviciile integrate*** care oferă facilităţi pentru:

- servicii de timp-real (audio-video)
- rezervarea resurselor
- servicii diferenţiate
- multicast

Serviciile integrate susţin:

- serviciul de datagrame
- serviciul încărcării controlate: performanţa să fie la fel de bună ca într-o reţea de transport de datagrame neîncărcată – fără asigurarea cantitativă

În general, formele de comunicare sunt:

- unicast – distribuire de la o sursă la o destinaţie
- broadcast – distribuire largă – la toate gazdele dintr-o reţea (mică, locală)
- directed broadcast – distribuire largă direcţionată – la toate gazdele dintr-o reţea îndepărtată
- multicast – distribuire multiplă – la mai mulţi destinatari (engl.: recipients) – la un grup de destinatari (înregistraţi ca atare)

Ca aplicaţii de timp-real la forma multicast se întâlnesc:

- distribuiri audio-video:
  - unul la mai mulţi
  - simetric – toţi la toţi (de exemplu, la aplicaţii de teleconferinţe)
- simulări distribuite
- descoperiri de resurse
- distribuiri de fişiere:
  - preţuri ale produselor din pieţe (engl.: stock market quotes)
  - produse software noi

Principalele **teme ale comunicaţiei de timp-real** sunt:

- ***specificarea cerinţelor*** surselor care susţin traficul (comunicaţia)
- ***protocoale pentru crearea şi întreţinerea rezervării resurselor*** (de exemplu, RSVP=Resource ReSerVation Protocol)
- ***protocoale de rutare*** care susţin calitatea specificată a serviciilor şi multicast-ul (de exemplu, ST2+=Stream Protocol)
- ***protocoale de transport pentru controlul erorilor şi fluxurilor*** (de exemplu, RTP=Real-time Transport Protocol)
- ***controlul accesului*** – conexiuni bazate pe utilizare şi pierderea pachetelor
- ***planificarea pachetelor*** pentru a furniza servicii de calitate specificate (***QoS=Quality of Service***) – punerea la coadă bazată pe ponderare (engl.: Weighted Fair Queueing)

Aplicaţiile de timp-real au nevoie de **servicii garantate** care se referă la:

- limitarea (garanatarea) fermă a capacităţii de transfer a datelor şi întârzierii lor de la sursă la destinaţie
- limitarea (garantarea) întârzierilor (în noduri) - fiecare element din cale (din traseu) trebuie să producă o întârziere limitată

Ca şi **caracteristici negative** (dar care pot fi acceptate) ale utilizării serviciilor garantate sunt:

- nu totdeauna sunt implementabile (mai precis nu în orice context) – de exemplu în reţeaua Ethernet nu sunt acceptate toate protocoalele pentru servicii de timp-real
- nu se minimizează întârzierea startării transmiterii şi întârzierea medie de transmitere a pachetelor (sau cadrelor)

Realizarea aplicaţiilor de timp-real necesită specificarea precisă a cerinţelor temporale.
Este nevoie de specificarea caracteristicilor fluxurilor care constă din:

- specificarea traficului:
  - rata maximă (de vârf, engl.: peak rate) – poate fi necunoscută sau nespecificată uneori
  - rata mănunchiurilor de pachete (engl.: bucket rate)
  - dimensiunea mănunchiurilor
  - dimensiunea maximă a unei datagrame

- unitatea minimă tratată sau controlată – toate datagramele mai mici sunt contorizate ca octeţi

- specificarea calităţii serviciului: rata şi întârzierea stagnărilor

Ultima specificaţie se referă numai pentru servicii cu rate garantate.

Clasificatorul pachetelor examinează fiecare pachet care intră şi determină clasa din care face parte. Unele pachete pot avea atributul *preemtibil* şi prin urmare trebuie să se rechiziţioneze canalul de comunicaţie pentru ele.

Planificatorul pachetelor gestionează cozile şi temporizările diferitelor fluxuri. Controlul admiterii determină dacă poate fi acceptată intrarea unui nou flux fără să afecteze fluxurile deja existente (care au intrat anterior).

Protocolul RSVP (acronim de la Resource ReSerVation Protocol)

Rezervarea resurselor se realizează prin încărcarea diferenţiată a nodurilor. ➔ tratarea preferenţială a pachetelor care sunt considerate importante.

Se rezervă o parte specificată din capacitatea de transfer a nodurilor.

Există mai multe protocoale de rezervare care sunt orientate pe transmiţător şi receptor.

RSVP este un protocol de setare a rezervării şi care realizează semnalizările în Internet.

El realizează transportul cererilor de rezervare în reţea.

RSVP acţionează prin comunicarea ruterilor de pe traseul pachetelor să inspecteze traficul

UDP şi TCP şi să dea prioritate acelor pachete care au cereri de bandă de frecvenţă.

Mesajele RESV conţin:

- specificaţia fluxurilor:
  - planificatorul pachetelor utilizat
  - caracteristicile serviciilor garantate care includ:
    - rata maximă
    - rata mănunchiurilor
    - dimensiunea mănunchiurilor de pachete
    - dimensiunea maximă a datagramelor
    - unitatea minimă controlată
    - frecvenţa şi întârzierile stagnărilor
- specificaţia filtrului:
  - definesc pachetele din flux
  - clasificatorul pachetelor

Rezervarea poate fi realizată cu:

- filtru fix – când există câte o conductă pentru fiecare sursă
- filtru „wildcard” – când este o conductă pentru toate sursele dintr-o sesiune
- utilizare în comun specificată – în care sursele sunt identificate în mod explicit

Protocolul RTP = Real-time Transport Protocol

Este un protocol de nivel aplicaţie utilizat pentru transmiterea şi recepţionarea în timp real a datelor (inclusiv imagini şi sunete).

RTP defineşte structura datelor transmise şi mecanismele de control pentru transmiterea datelor (Real-time Transport Control Protocol).

Protocolul RTP nu rezolvă problema asigurării timpilor de livrare a mesajelor şi alte mecanisme de control, el folosind în acest scop alte protocoale.

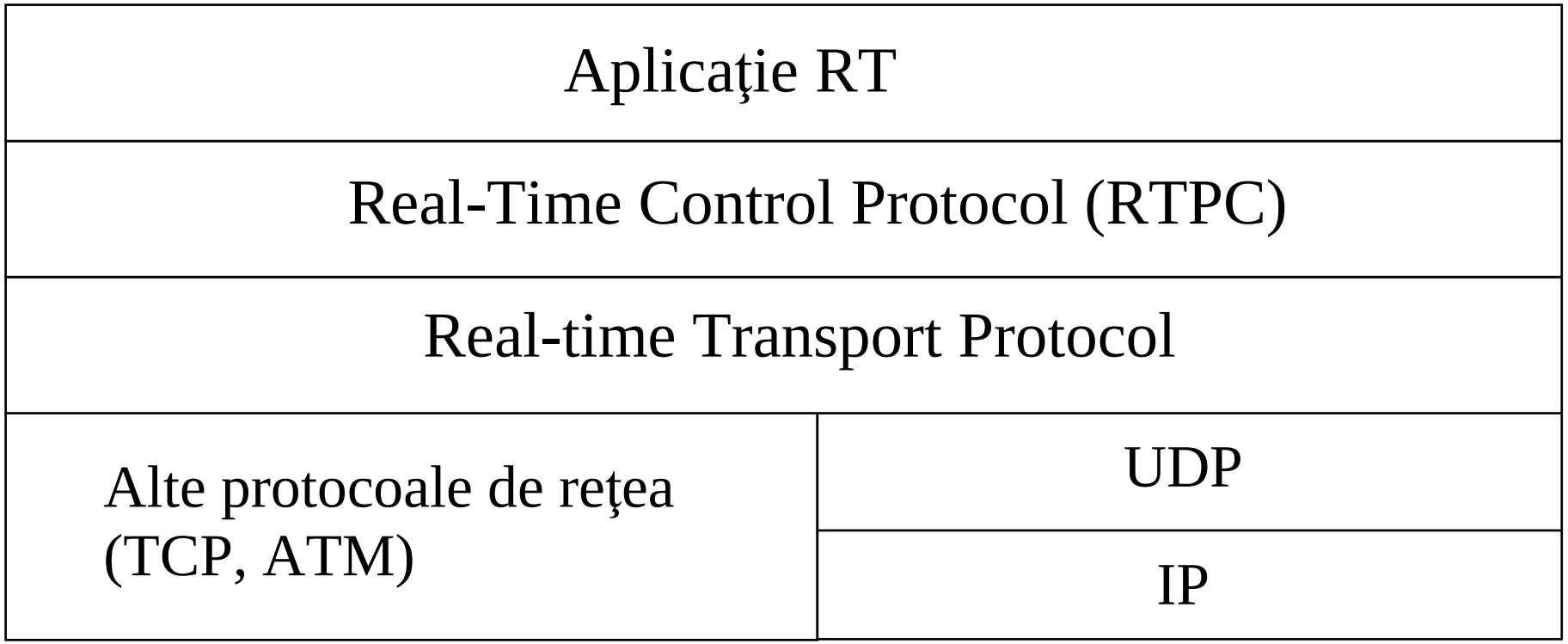
Protocolul RTP realizează:

- distribuirea unicast sau multicast
- identificarea tipului sursei şi a încărcăturii (adică a tipului de date transmise)
- secvenţierea corectă a mesajelor (sau datelor)
- temporizarea şi sincronizarea distribuirii mesajelor
- reunirea surselor multiple pentru combinarea fluxurilor produse de un mixer RTP
- translaţia fluxurilor – de la viteze ridicate la viteze mai coborâte

Protocoalele RTP şi RTPC lucrează în strânsă legătură unde:

- RTP este responsabil pentru transmiterea datelor
- RTPC monitorizează calitatea serviciilor

Arhitectura generală RTP:



Arhitectura generală la procesul de transmisie a datelor utilizând RTP:

Supervizarea şi controlul proceselor de la distanţă

Implementarea în limbajul Java a aplicaţiilor distribuite

Implementarea cu protocolul TCP

Clasa	Scop
URL	Reprezintă un URL
URLConnection	Returnează continutul adresat de obiectele URL
Socket	Creează un soclu (socket) TCP
ServerSocket	Creează un soclu pentru server (server socket) TCP
DatagramSocket	Creează un soclu (socket) UDP
DatagramPacket	Reprezintă o datagrama trimisă printr-un obiect DatagramSocket
InetAddress	Reprezintă numele unui PC din reţea, respectiv IP-ul corespunzător

7.6. Obiecte distribuite

Clase și obiecte
Structura aplicației Java

Obiecte distribuite

Un obiect care apelează o metodă a altui obiect trebuie să ii știe adresa

- Cum se rezolvă funcționalitatea garbage collector ?
- Cum poate fi rezolvată comparația obiectelor ? în Java: *RemoteObject::equals()*

Metode cunoscute:

- Remote Method Invocation (RMI) în Java
- Remote Procedure Call (RPC) în C

Distributed object identification (Remote Object IDentifier), în Java:
ROID = endpoint(Java Virtual Machine) + identificator (ObjID)

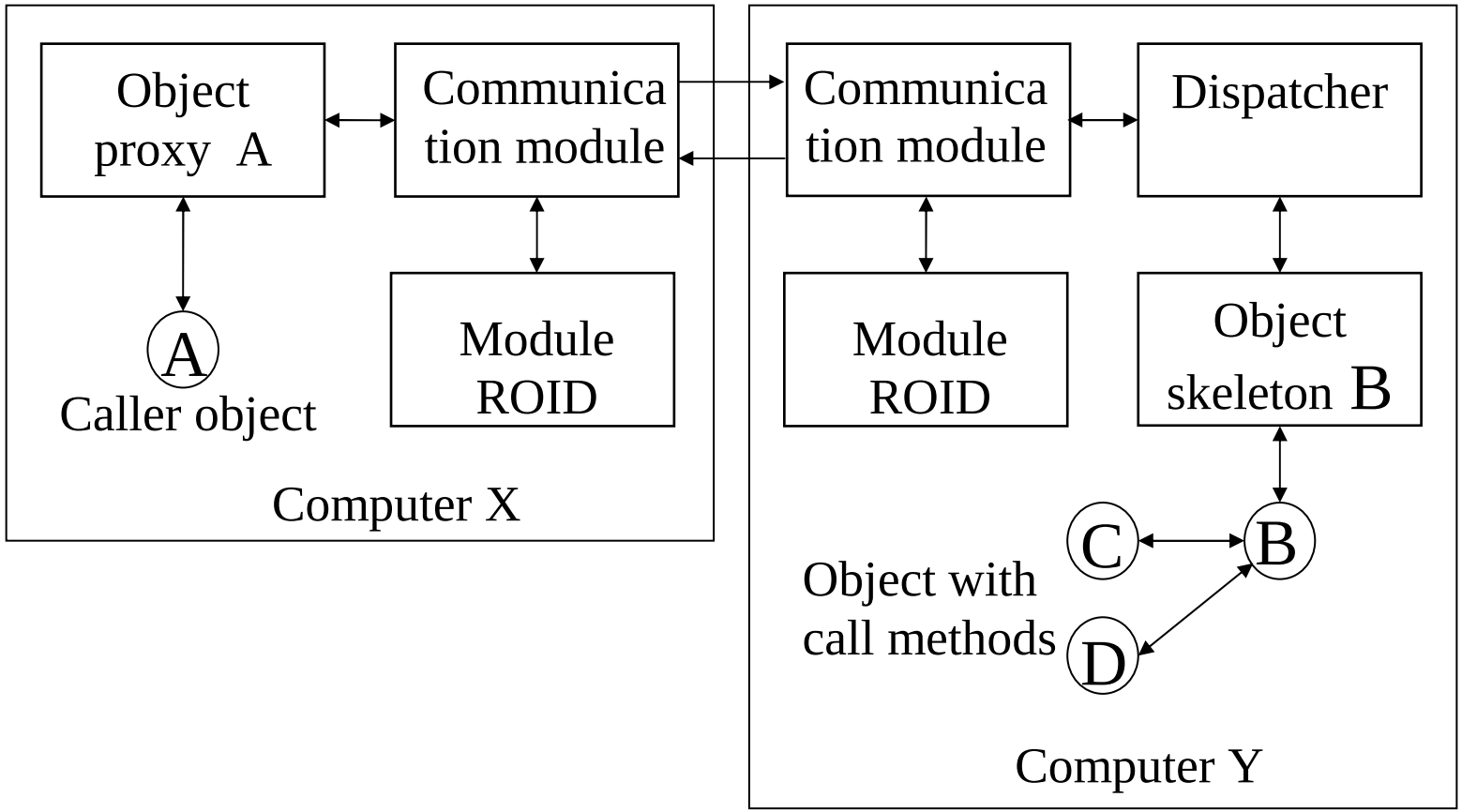


Fig. 7.16. Schema de comunicaţie între obiecte distribuite

Garbage Collector Distribuit

Colectarea reziduurilor (rezolvată local bine în Java) în aplicaţiile distribuite este mai complicată datorită necesităţii informării componentelor din mediul distribuit dacă mai trebuie utilizate obiectele în viitor. Un colector de reziduuri distribuit construit cu module ROID trebuie să realizeze următoarele:

- să ţină evidenţa numărului de staţii îndepărtate înregistrate ROID pentru fiecare obiect local (să menţină o tabelă de evidenţă a utilizărilor obiectelor)
- să informeze celelalte module ROID despre generarea/ştergerea din ROID a obiectelor lor locale (prin utilizarea unor metode de tip *addRef()* şi *removeRef()*)
- să colecteze reziduurile obiectelor care nu mai sunt referite nici local nici îndepărtat
- în reţelele de comunicaţie nefiabile să realizeze numărarea referinţelor (cu metode de tip *addROID()* şi *removeROID()*)

Execuţia sincronă şi asincronă a metodelor

activ sau pasiv.

Acest ***transfer asincron al execuţiei*** este mai complicat deoarece implică întreruperea execuţiei curente şi înlocuirea ei cu cea cerută.

Obiecte **Real_time**

Obiecte active
Obiecte pasive

În aplicaţiile de timp-real se utilizează două tipuri de *obiecte: active şi pasive*.

Obiectele active pot fi antrenate de:

- evenimente produse de componente (hardware sau software) externe calculatorului
- timp – când timpul sistemului a ajuns la o anumită valoare, sau a trecut un anumit interval de timp de la un eveniment precedent

Un caz mai complicat este acela în care un obiect transmite altui obiect secvenţa de instrucţiuni pe care s-o execute realizând ceea ce se numeşte ***migrarea codului.***

7.7. Obiecte distribuite Java

Implementarea folosind tehnica RMI

Aplicaţia server

Se defineşte interfaţa pe care o implementează obiectele distribuite. Această interfaţă va conţine metodele care vor putea fi apelate de la distanţă prin intermediul mecanismelor RMI. Interfaţa va trebui să extindă interfaţa ***RemoteInterface.*** Metodele definite în cadrul acestei interfeţe vor trebui să aibă adăugată clauza *throws RemoteException*.

Se construieşte clasa care implementează interfaţa definită la pasul anterior. Această clasă trebuie să extindă clasa ***UnicastRemoteObject***.

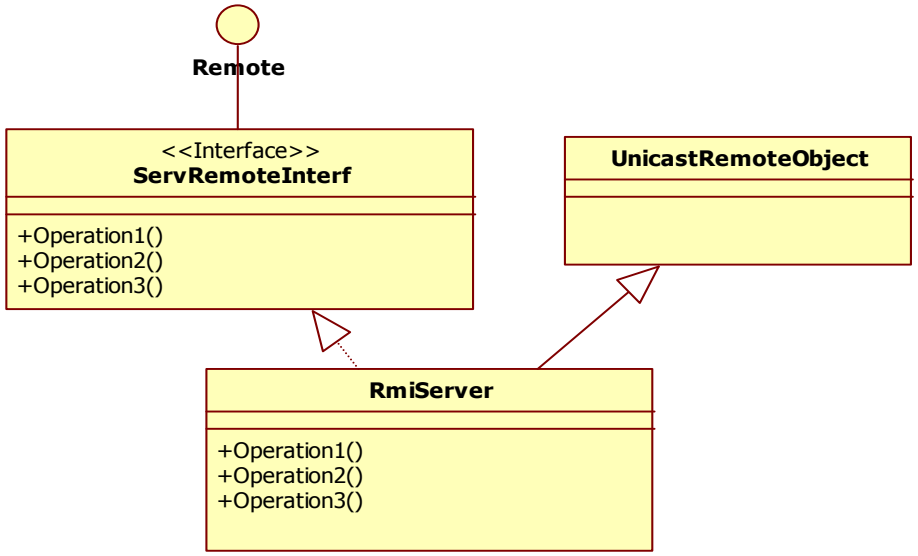
Pentru a putea face accesibil de la distanţă un obiect prin intermediul mecanismului RMI trebuiesc realizaţi următorii paşi suplimentari:

1. Instalarea în cadrul aplicaţiei ce urmează a construi şi înregistra obiecte distribuite a unui manager de securitate folosind clasa ***java.rmi.RMISecurityManager***.
2. Lansarea în execuţie fie de la consolă, fie direct din cadrul aplicaţiei a utilitarului ***rmiregistry.*** Lansarea de la consolă se face executând comanda ***rmiregistry port*** (unde port reprezintă portul pe care utilitarul va aştepta conexiuni – sau cereri de accesare a obiectelor distribuite).

Lansarea din cadrul aplicaţiei se face cu instrucţiunea *LocalRegistry.createRegistry(port)* (unde port are aceeaşi semnificaţie ca şi în cazul lansării de la consolă).

RmiServer are 3 metode care pot fi apelate de la distanță (remote).

RmiServert ref=new RmiServer();
try{
Naming.rebind(„serverPlace”,ref);
} catch
(RemoteException e) {}
catch (MalformedURLException me) {}
catch (Exception ex){}



Clientul implementează:

try {
ServRemoteInterf remoteObject=
(ServRemoteInterf)Naming.lookup(„serverPlace”);
} catch (Exception e) {}
remoteObject.Operation1();

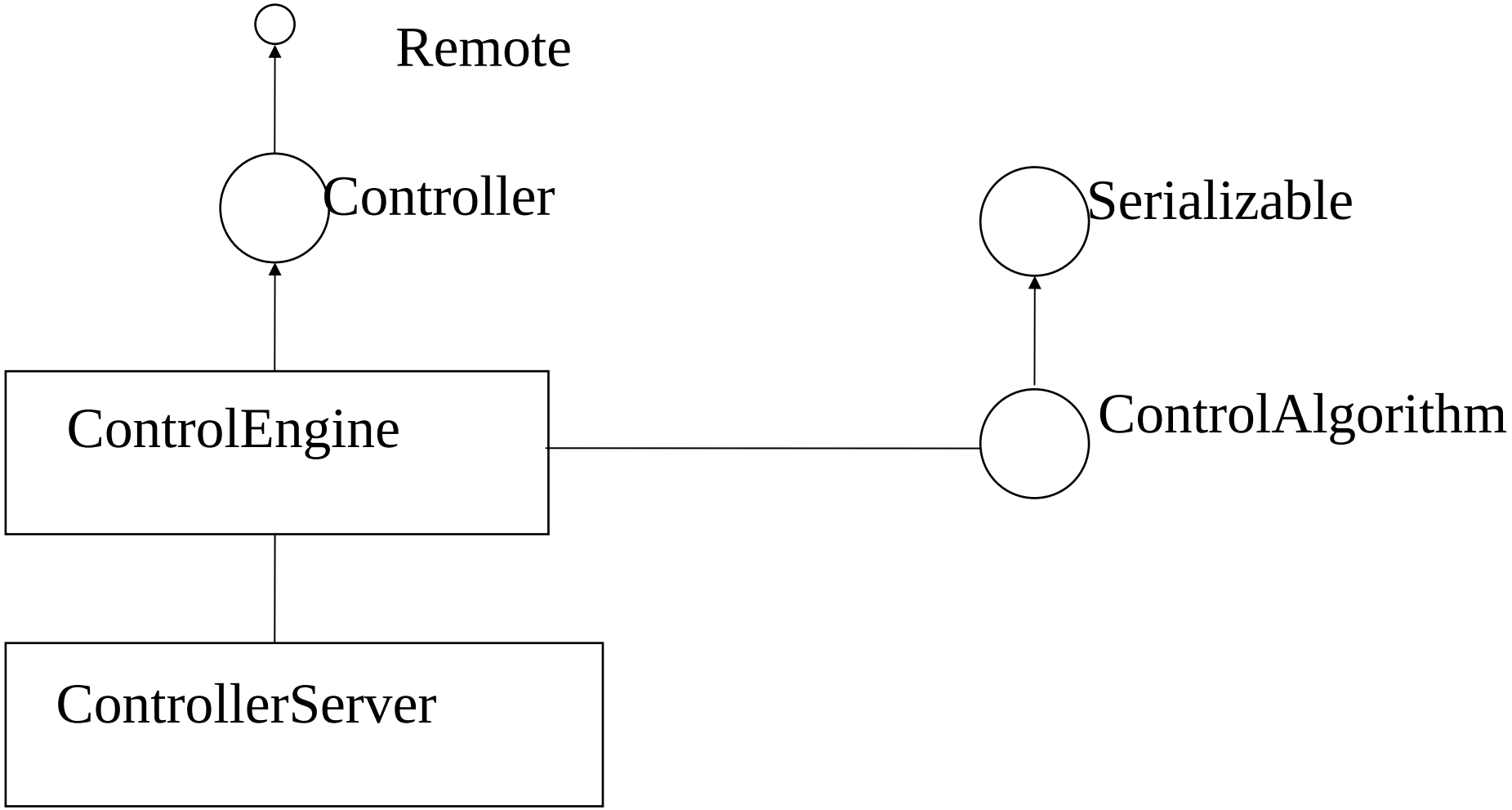


Fig. 7.18. Diagrama UML a claselor pentru aplicaţia server

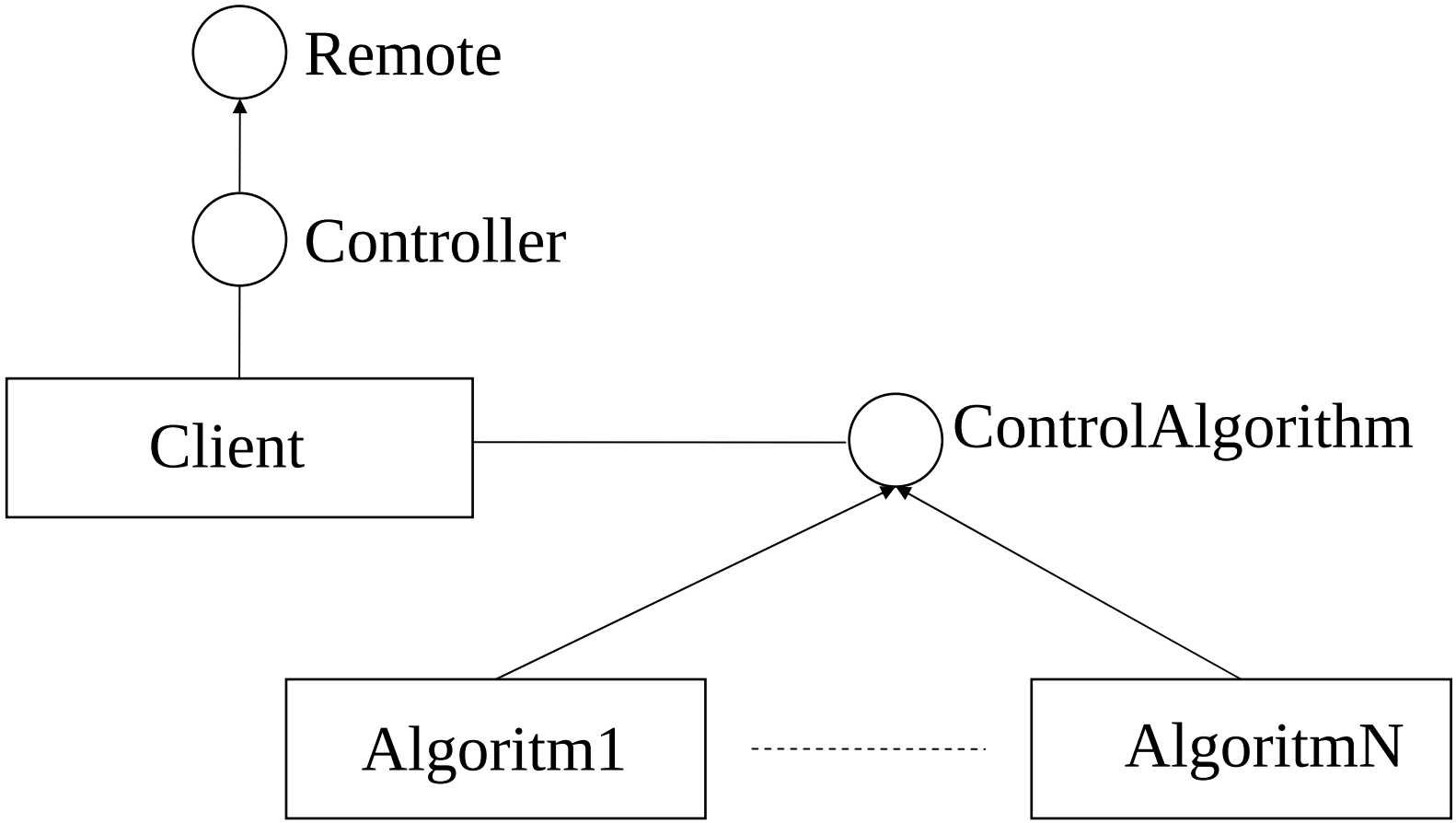
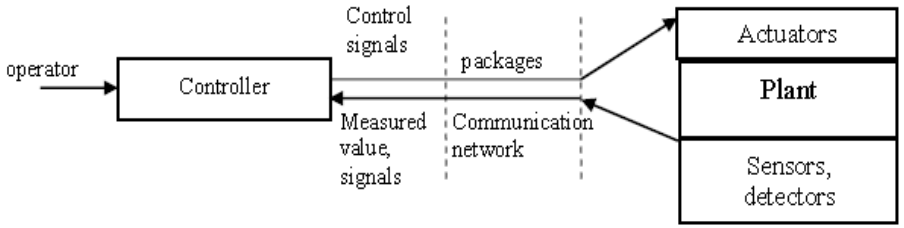
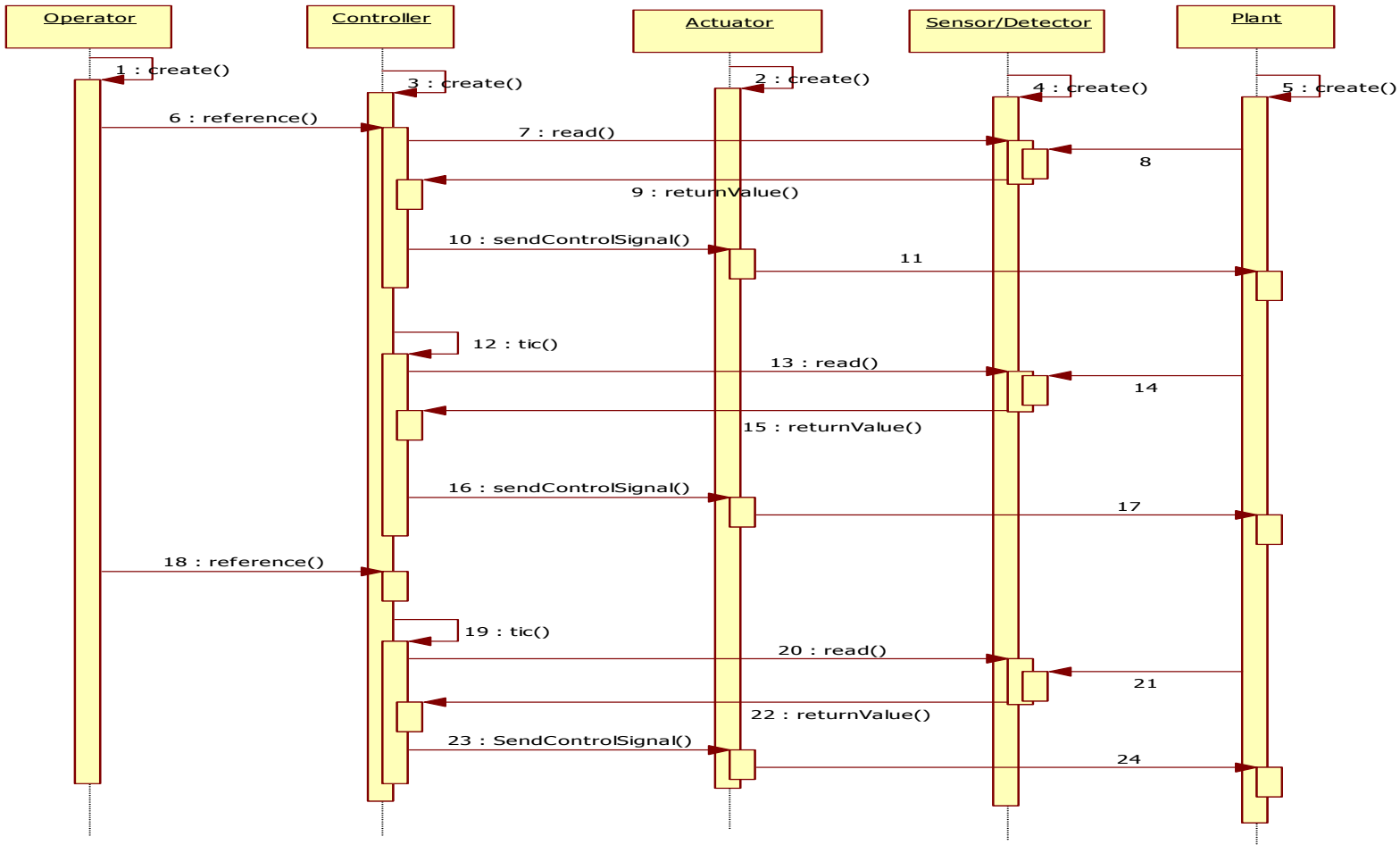


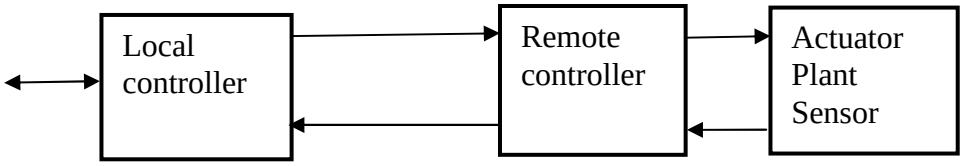
Fig. 7.19. Diagrama UML a claselor pentru aplicaţia client

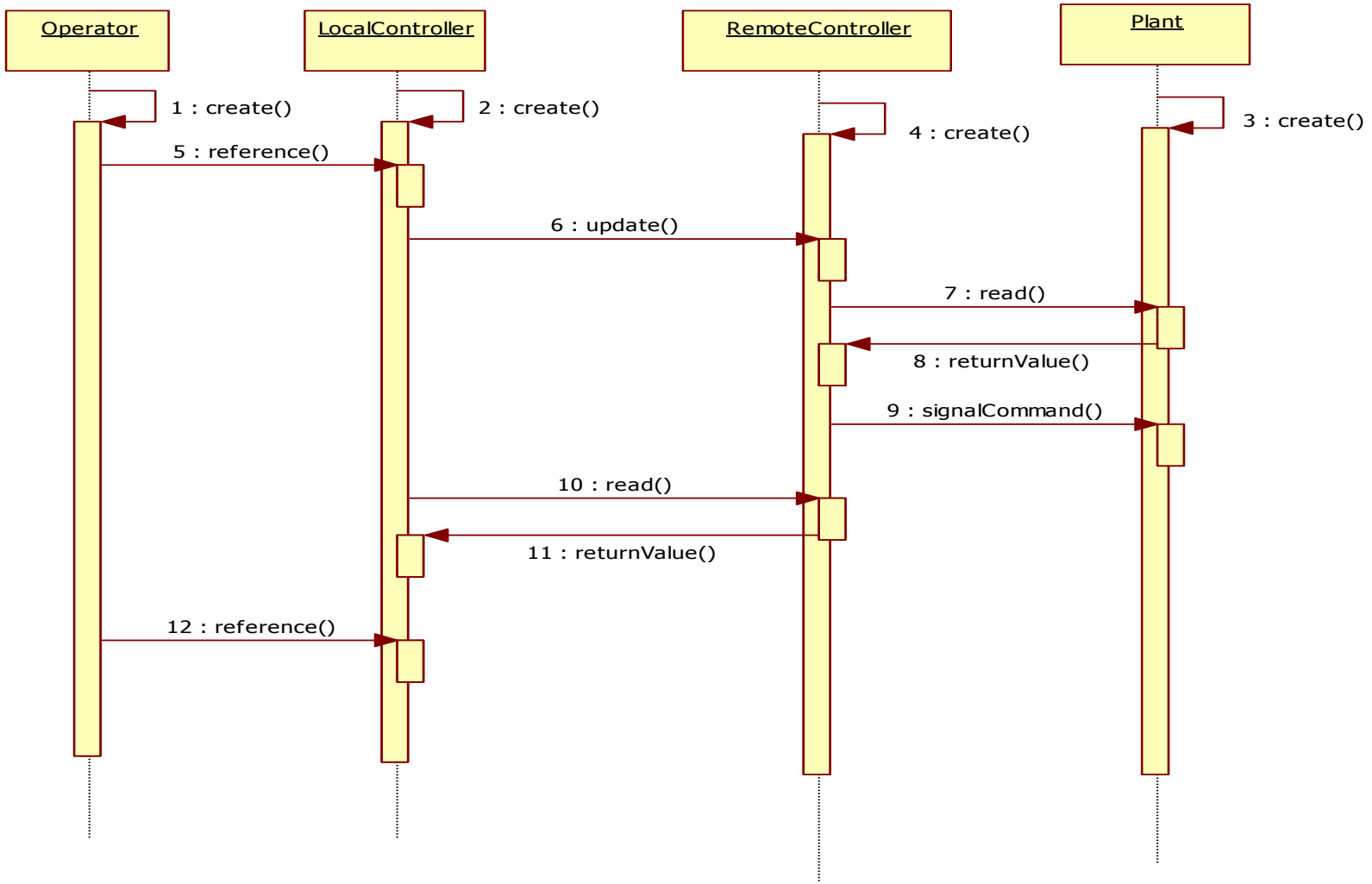
Implementarea controlului distribuit
Controler local și sistem la distanță (remote).





Local controller + remote controller





Temă de casă

Concepeți o aplicație de control (supervisor-controller) care schimbă algoritmul de control la distanță pentru o intersecție (Urban vehicle traffic control - UVTC) de la 2 faze la 3 faze – plus una pentru pietoni.

Cerințe:

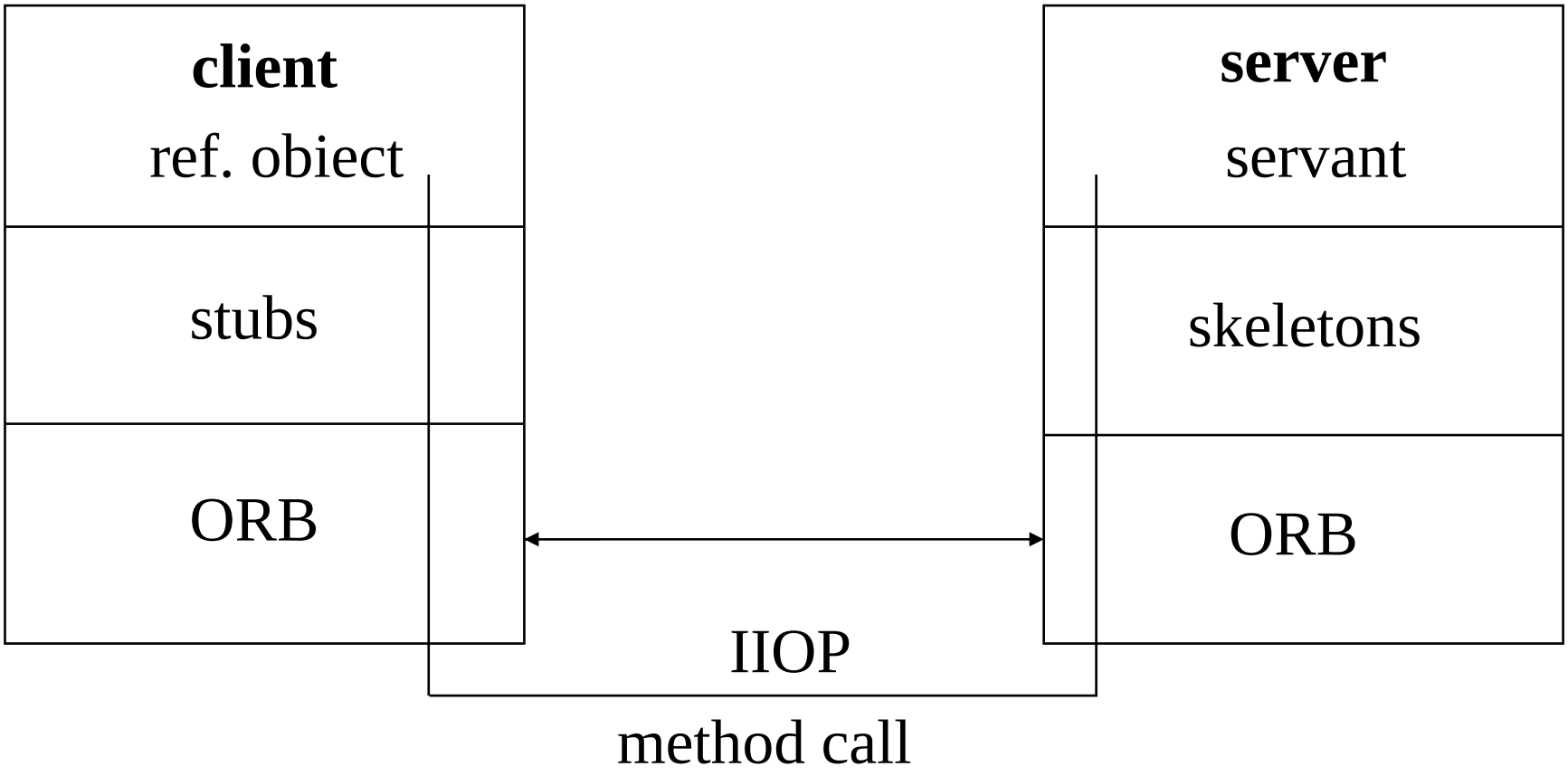
1. Specificarea UVT
2. Specificarea clientului
3. Algoritmul de control
4. Specificarea serverului
5. Diagrame de clase
6. Diagrama de comunicare a obiectelor
7. Diagrama componentelor
8. Diagrama de secvențe
9. Exemple de cod relevant

FLETPN
Pre & Post matrix
Delay vector
FLRS

CORBA

CORBA = *Common Object Resource Broker Architecture* şimprovement of *Core Object Model*.

ORB (acronim de Object Request Broker)



IDL = Interface Definition Language CORBA.

Codul ***stub*** foloseşte ORB pentru identificarea maşinii care execută obiectul server şi cere o conexiune către acesta. Când codul stub are conexiunea, el trimite apelul şi parametrii prin referinţa obiectului la codul *skeleton* (schelet) cu care s-a legat.

Codul ***skeleton*** transformă apelul şi parametrii într-un format de implementare specificat şi apelează obiectul. Pe aceeaşi cale sunt returnate obiectului apelant rezultatele şi excepţiile (dacă apar).

CORBA nu susţine în mod direct tranzacţii, gestiunea concurenţei, multiplicarea şi copierea obiectelor, dar există servicii suplimentare care pot fi instalate şi rezolvă aceste probleme.

Componentele sistemului ORB comunică între ele prin intermediul protocolului *IIOP (acronim de la Internet InterORB Protocol).* IIOP poate fi un protocol de transport fie pentru aplicaţii distribuite scrise în IDL, fie în Java.

Un client poate invoca numai metodele care sunt specificate într-o interfaţă CORBA.

Interfeţele obiectelor CORBA se bazează pe o interfaţă IDL care defineşte un anumit tip de obiecte. Sintaxa IDL este asemănătoare cu cea a limbajului Java. Interfeţele IDL pot fi translatate în Java folosind **compilatorul *idlj*** (prescurtare de la IDL to Java).

IDL este un limbaj pur declarativ pentru specificare interfeţelor operaţionale necesare în aplicaţii distribuite. Printre limbajele acceptate pentru translatare sunt: C, C++, COBOL, Ada şi Java.

Real-Time CORBA

Real-Time CORBA 1.0. (standard) – prevede lucrul cu priorităţi fixe

Real-Time CORBA 2.0. (standard) – prevede lucrul cu priorităţi dinamice

➔ dynamic scheduling

Se presupune că sistemul de operare susţine planificarea bazată pe priorităţi în noduri şi nu se presupune existenţa unor facilităţi de T-R în reţea

Pt. full-R-T tebuie ca tot sistemul să fie proiectat şi construit astfel încât să aibă caractersitici de T-R.

R-T CORBA defineşte un fir ca o entitate planificabilă. El are:

- CORBA Priority
- NATIV Priority

*

**

****

******

*** Sfârșit ***

******

****

**

*