

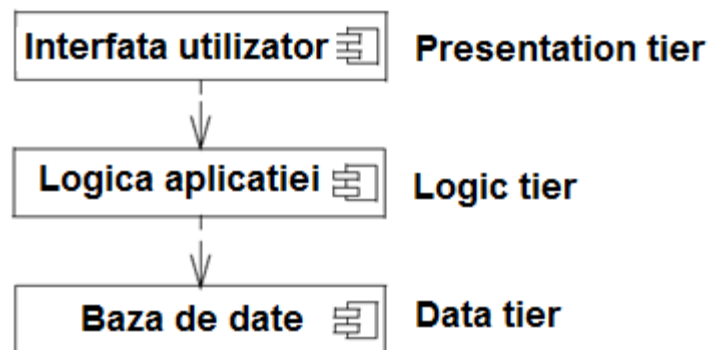
Proiectarea arhitecturala -2

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Ingineria programelor* – UPB, Automatică și Calculatoare
2020-2021

Arhitectura *Three-tier* (1)

- ❖ Arhitectura ierarhica închisă, în 3 niveluri.
- ❖ Specifică sistemelor care includ o baza de date.
- ❖ Subsistemele sunt organizate în 3 niveluri ierarhice:
 - **Nivelul interfață utilizator (*Presentation tier*)** – responsabil cu interacțiunea prin ferestre, form-uri, pagini web, s.a.
 - **Nivelul aplicație (*Application logic/ Logic tier*)**, numit si *Middleware* – include prelucrările specifice aplicației si comunicarea dintre nivelul *interfață* și nivelul *stocare*
 - **Nivelul stocare (*Storage/ Data tier*)** – sistemul de gestiune a bazei de date – asigura stocarea si regasirea obiectelor persistente.



Arhitectura *Three-tier* (2)

❖ Cele 3 niveluri sunt de regula alocate pe noduri hardware distincte.

▪ Arhitectura este des folosita în sistemele bazate pe Web:

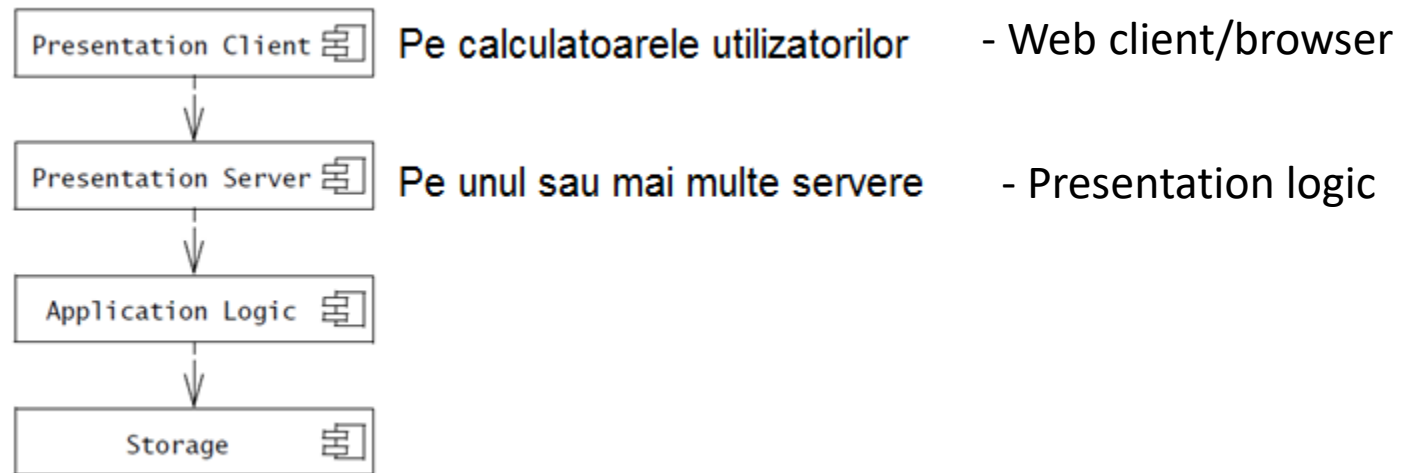
- Web Browsers implementează nivelul Interfață utilizator
- Web server – trateaza cererile provenite de la web browser: implementeaza nivelul aplicatie
- Sistemul de gestiune al bazei de date – asigura gestiunea datelor persistente

❖ Avantajele arhitecturii:

- *Nivelul stocare*, analog unui subsistem *Repository*, **poate fi partajat de diferite aplicatii** care utilizeaza aceeasi baza de date.
- Separarea nivelului *interfata* de nivelul *aplicatie* permite **existenta mai multor interfețe utilizator** pentru subsistemul/subsistemele care implementeaza logica aplicatiei.
- **Fiecare dintre cele 3 niveluri poate fi îmbunatatit sau înlocuit independent**, în cazul unei schimbari a cerintelor sau a unei schimbari tehnologice.

Arhitectura *Four-tier* (1)

- ❖ Este o variantă a arhitecturii Three-tier în care nivelul *Interfata utilizator* este descompus în:
 - nivelul *Prezentare Client* (Presentation Client Layer) – operatii minimale de interactiune
 - nivelul *Prezentare Server* (Presentation Server Layer) – implementeaza logica prezentarii



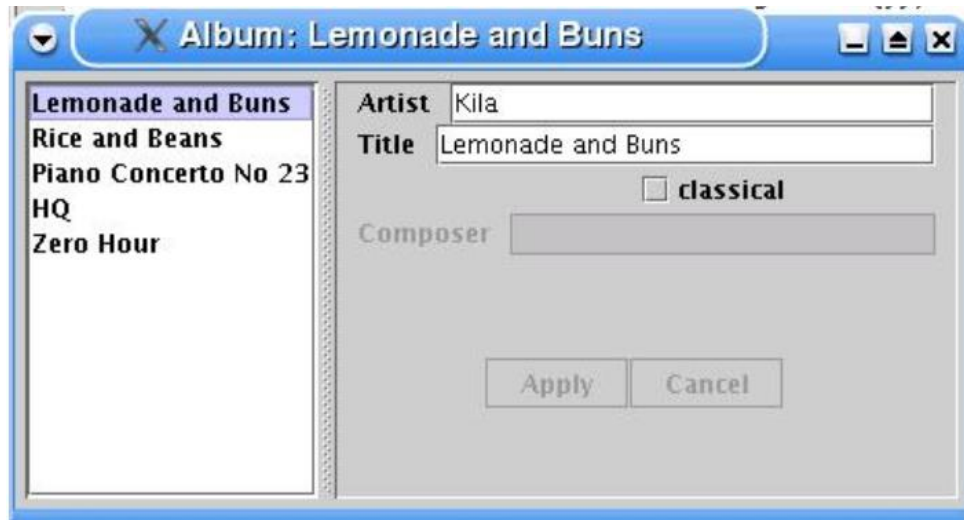
■ Avantaje:

- Cele 2 niveluri Prezentare pot fi implementate in limbaje de programare diferite si modificate independent.
- Nivelul *Presentation Client* poate oferi o gama larga de interfete utilizator (pentru diferite tipuri de dispozitive) care pot partaja elemente ale nivelului *Presentation server*
- Mai scalabila decat arhitectura în 3 niveluri.
- Mai usor de testat

Arhitectura *Four-tier* (2)

Logica prezentarii - exemplu:

- Alegerea din lista din stanga ferestrei determina datele care se vor afisa in partea dreapta a ferestrei și titlul ferestrei.
 - Câmpul Composer poate fi completat numai dupa ce s-a bifat check-boxul “classical”.
 - Butoanele Apply si Cancel pot fi folosite numai dupa ce s-au completat toate campurile.
- ❖ Toate verificarile legate de afișările din partea dreaptă a ferestrei și posibilitatile de interacțiune pot fi efectuate de nivelul “Presentation logic”.



Arhitecturi orientate pe interacțiune

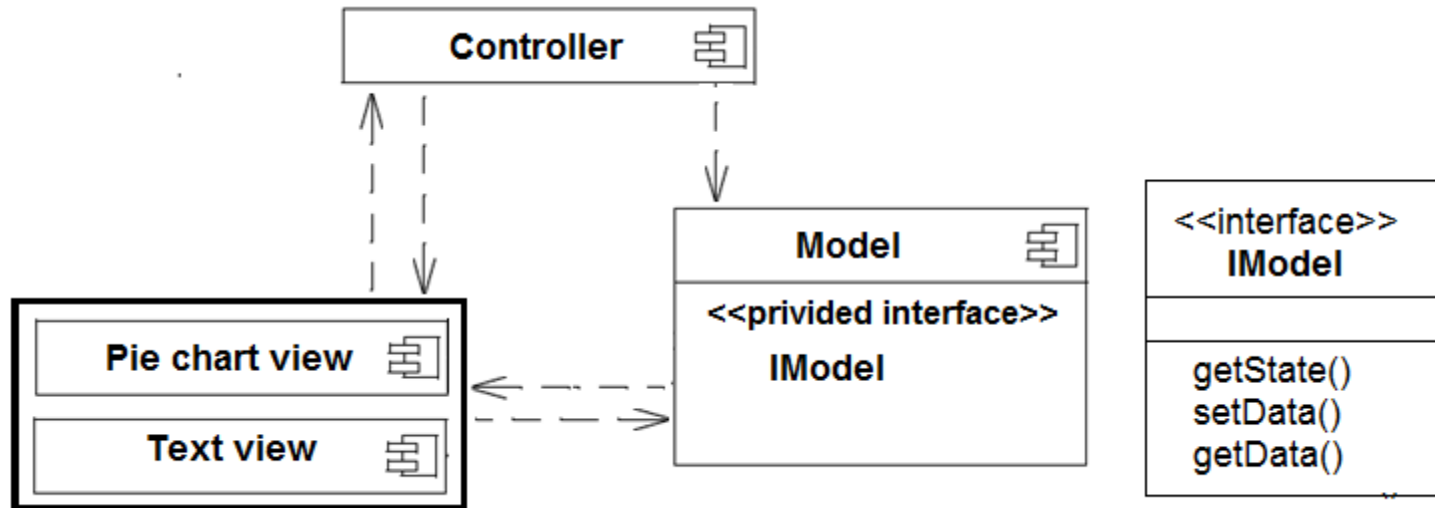
Model-View-Controller (1)

❖ **Scopul:** separarea interfeței cu utilizatorul de datele și logica aplicației.

Principalul stil arhitectural în această categorie: Model-View-Controller (MVC).

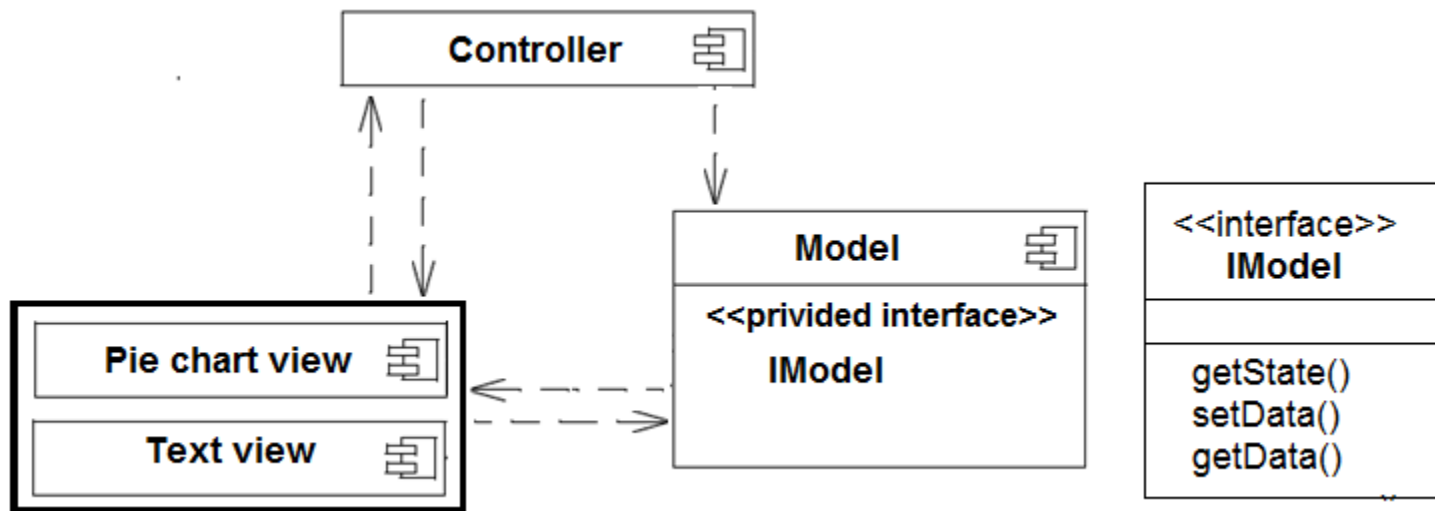
❖ O arhitectura MVC este alcătuită din 3 tipuri de subsisteme:

- Subsistemul **Model**: conține o reprezentare a datelor specifice aplicației, împreună cu:
 - Operațiile de acces la date
 - Operațiile de actualizare a datelor, conform interacțiunii cu utilizatorul și logicii aplicației



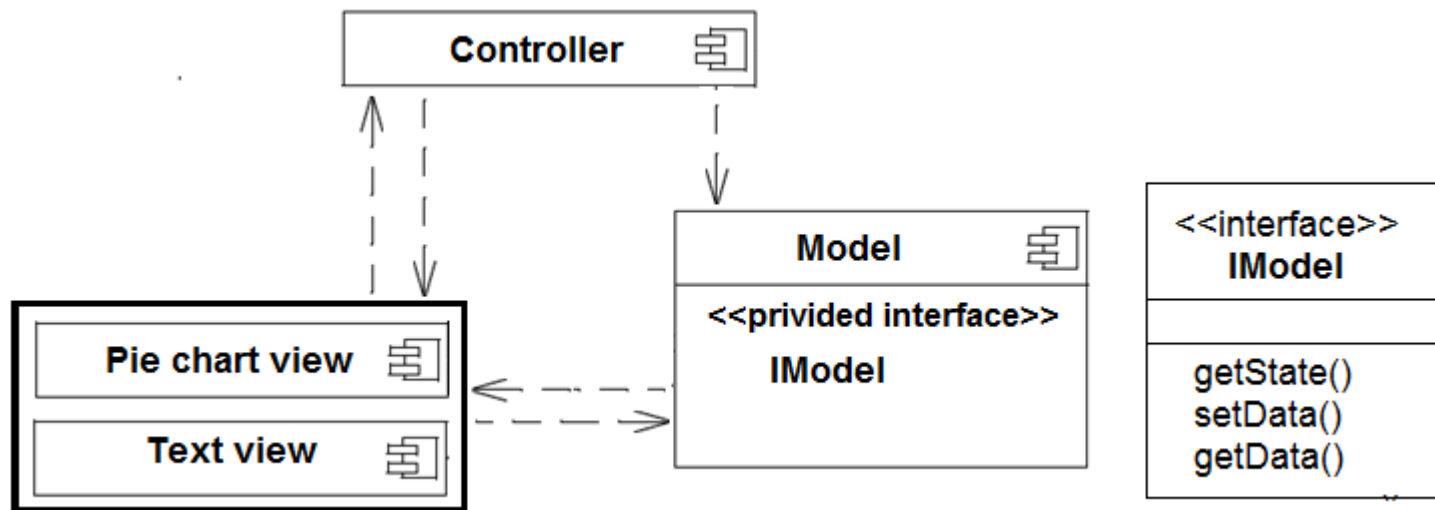
Model-View-Controller (2)

- Subsisteme **View**. Un subsistem View:
 - este asociat cu o fereastră ecran prin care utilizatorul interacționează cu sistemul
 - prezintă o vedere a modelului în fereastra asociată
 - transmite **Controller**-ului mesaje eveniment generate de interacțiunea cu utilizatorul
 - actualizează prezentarea datelor aplicației în fereastră, la orice modificare a lor



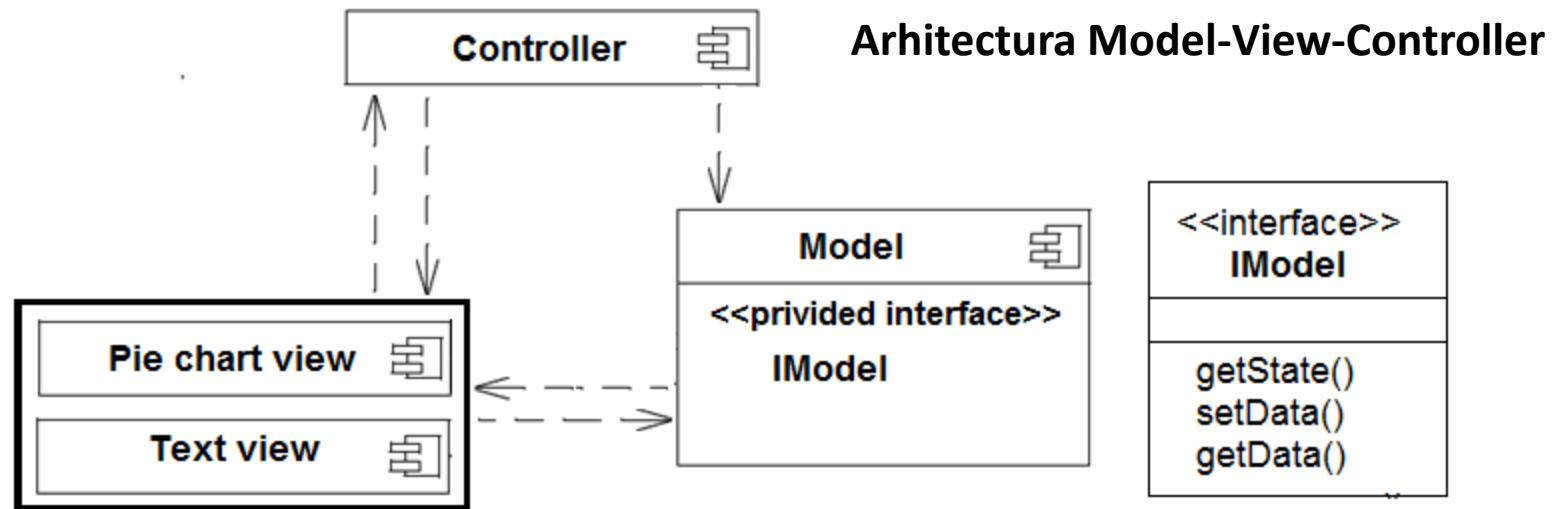
Model-View-Controller (3)

- Subsistemul **Controller**: procesează evenimentele generate de interacțiunea cu utilizatorul și transmite mesaje la:
 - **Model**, pentru accesarea datelor sau modificarea datelor în conformitate cu interacțiunea
 - **View**, pentru actualizarea vederii prezentate în fereastră.



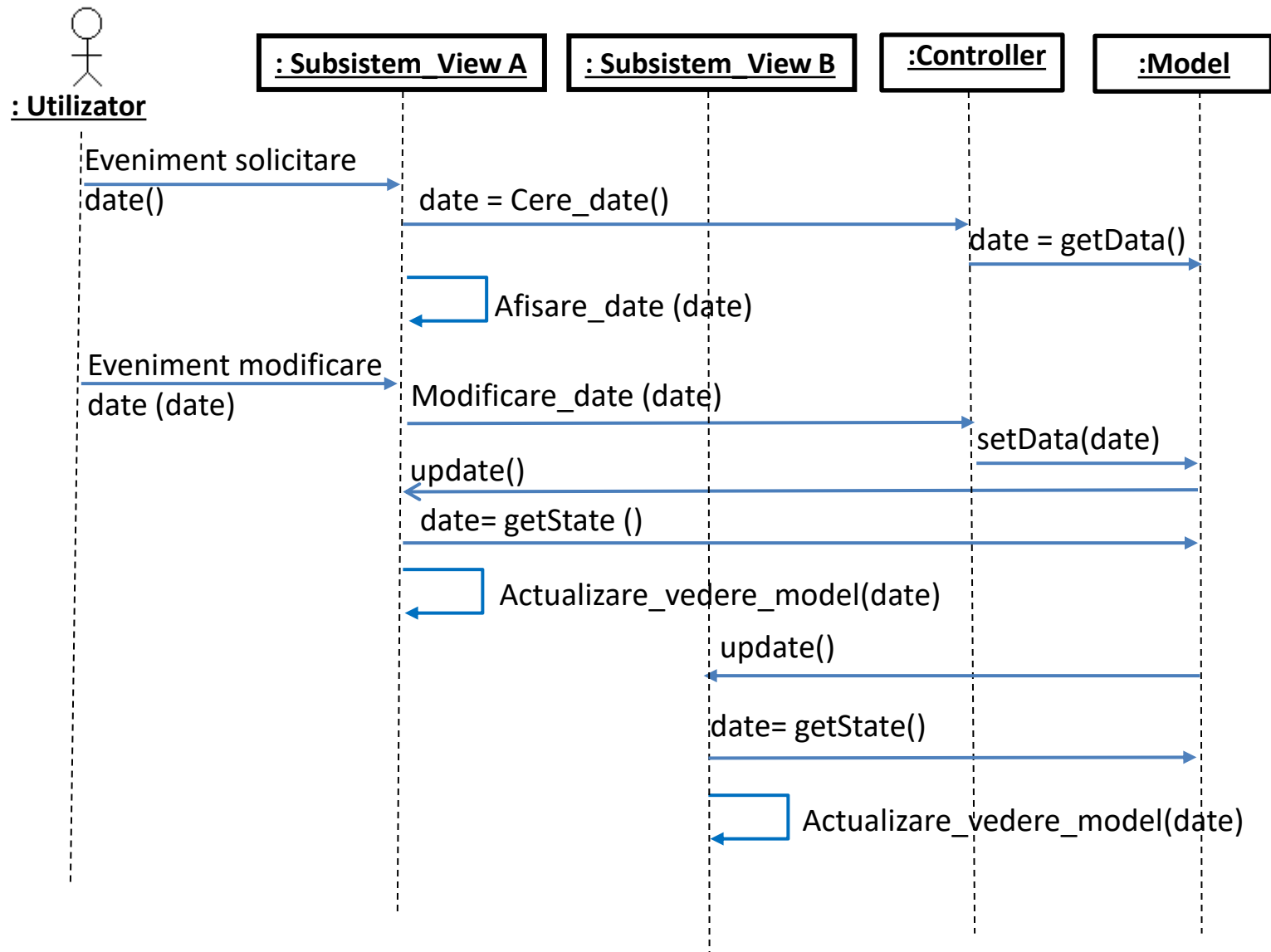
Model-View-Controller (4)

- ❖ MVC este un caz particular de *Repository activ*, unde subsistemul **Model** implementeaza depozitul (structura de date centrala) iar subsistemul **Controller** dictează fluxul controlului.



Model-View-Controller (5)

- scenariu de utilizare -



Model-View-Controller (6)

- ❖ Arhitectura MVC este adecvata sistemelor interactive, mai ales atunci cand sunt necesare mai multe vederi ale modelului.
- ❖ **Principalele avantaje ale arhitecturii:**
 - ofera flexibilitate în adaptarea sistemului: interfețele utilizator (View și Controller) sunt mult mai supuse schimbarilor decat modelul (datele specifice aplicatiei).
 - extensia simpla a sistemului pentru noi tipuri de clienti (componente View)
 - ușurează mentenanța codului
 - componentele sistemului pot fi dezvoltate și testate separat
 - adecvată aplicațiilor web
- ❖ **Dezavantaj:** poate conduce la cod suplimentar și creșterea complexității aplicației atunci cand modelul de date și interacțiunea cu utilizatorul sunt simple.

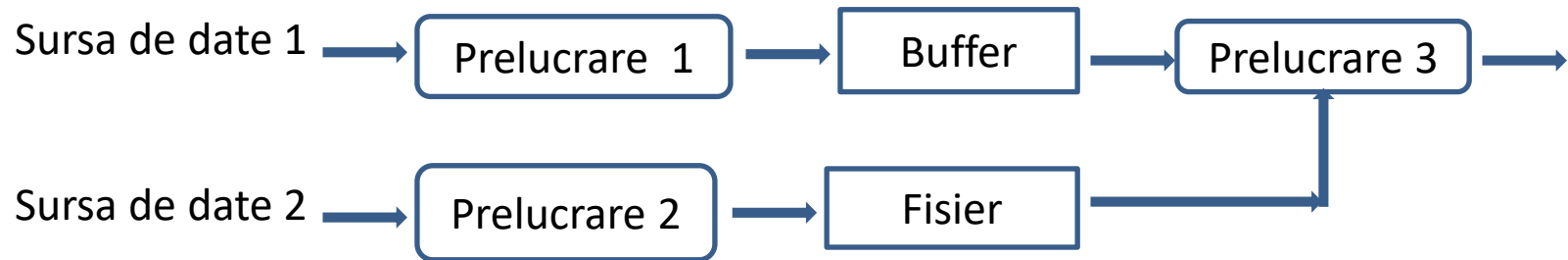
Arhitecturi centrate pe fluxul datelor: *Pipeline*

- **Fluxul global al controlului** este structurat în mai multe **etape de transformare a datelor** de intrare, pentru a produce anumite rezultate.
- **Fiecare etapă de transformare este o prelucrare simplă** si este alocată unui subsistem.
- **Comunicarea dintre subsistemele de transformare a datelor** se face prin **conectori** ca: buffer, coadă de mesaje, ș.a.
- Arhitecturi tipice bazate pe fluxul datelor:
 - Prelucrarea secvențială – pipeline secvențial (classic)
 - Pipe and filter – pipeline ne-secvențial



Pipeline secvential

- **Execuția unei etape de transformare a datelor se poate iniția numai după ce s-a terminat executia etapei precedente:** Prelucrare 3 se execută după Prelucrare1 **sau** după Prelucrare 2.
- Comunicarea între subsistemele de transformare se face prin buferi sau fișiere temporare.



❖ **Avantaje:**

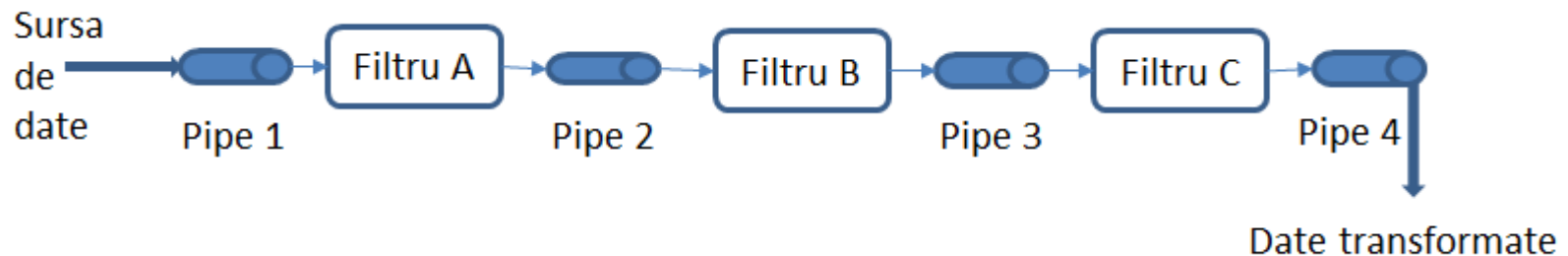
- Divizarea în subsisteme este simplă: fiecare subsistem poate fi un program independent care primește date de intrare și produce date de ieșire.
- Subsistemele de transformare a datelor pot fi reutilizate.

❖ **Dezavantaje**

- Nu furnizează suport pentru procesarea paralelă și interactivă.
- Latență în procesare și throughput coborât.

Arhitectura *Pipe and filter* (1)

- ❖ **Scopul arhitecturii:** descompunerea unei prelucrări complexe în prelucrări simple separate, care transformă datele incremental și pot fi executate în paralel.
- Arhitectura este alcatuită din:
 - O sursă de date
 - Mai multe etape de procesare, numite “filtre”, fiecare executnd o prelucrare simplă
 - Conectori, numiti “pipes”, prin care sunt transferate datele între filtre
 - Filtrele si conectorii formeaza un « pipeline »
 - Un receptor al datelor prelucrate în pipeline

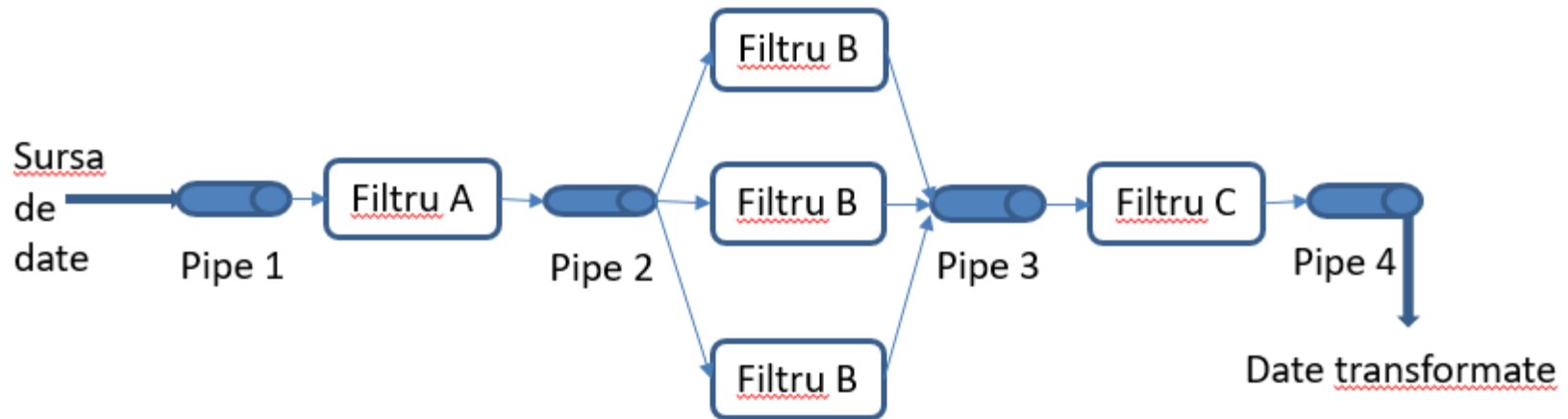


Arhitectura *Pipe and filter* (2)

- In general, intrările și ieșirile filtrelor sunt structurate ca fluxuri de date (streams), iar conectorii sunt implementati ca bufere FIFO.
- **Avantajele structurarii datelor în fluxuri:**
 - Fiecare filtru își începe executia imediat ce s-au primit date în conectorul său de intrare și se execută atâta timp cât exista date în conectorul respectiv.
 - Un filtru își poate începe executia chiar daca filtrul anterior în secventa de procesare, care produce fluxul său de intrare, nu și-a terminat executia.
 - Filtrele din pipeline pot fi executate în paralel.
- Filtrele se pot executa pe resurse de calcul diferite: filtrele intensiv computationale pot rula pe hardware de înalta performanta, altele pe hardware obisnuit.
- Filtrele pot fi executate pe calculatoare aflate în locatii geografice diferite, în apropierea resurselor de care au nevoie.

Arhitectura *Pipe and filter* (3)

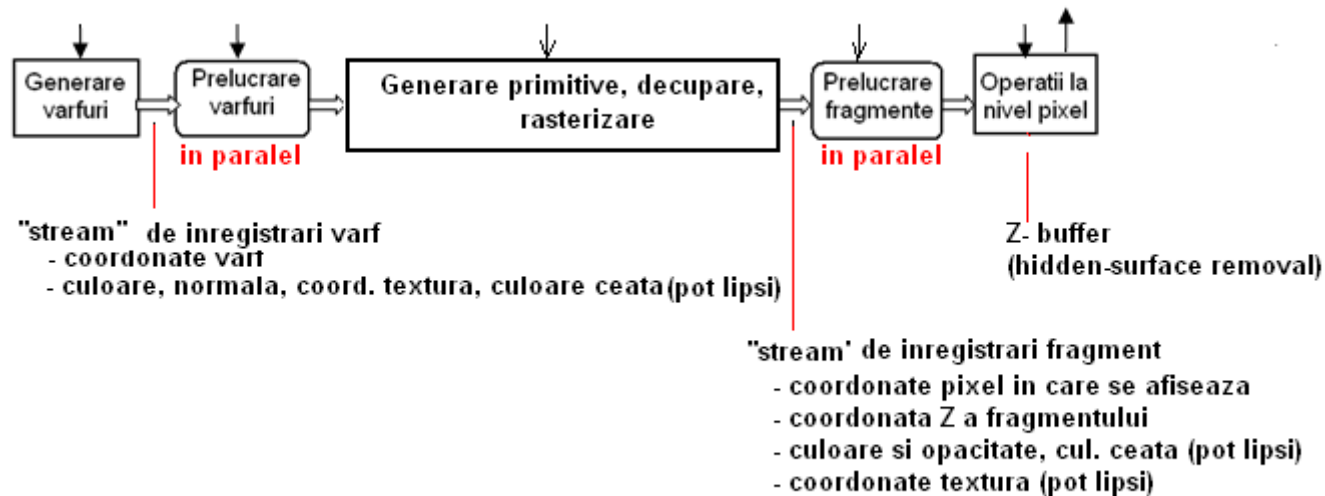
- Mai multe instanțe ale unui filtru pot fi executate în paralel pe aceeași mașină sau pe mașini diferite.



- Cele trei instanțe ale filtrului B se execută în paralel pentru date diferite provenite din același conector de intrare.
- Datele produse sunt memorate în același conector de ieșire.

Arhitectura *Pipe and filter* (4)

Exemplu: Graphics pipeline



- Filtrul “Generare varfuri” produce un flux de inregistrari “varf”.
- Inregistrările “varf” sunt prelucrate în paralel de instante ale filtrului “Prelucrare varfuri”(Vertex shader). Instantele sunt executate de procesoare diferite, pentru date (varfuri) diferite.

Arhitectura *Pipe and filter* (5)

❖ Avantajele arhitecturii

- Permite **paralelismul** și asigură o rată înaltă de execuție a tascurilor de prelucrare a unor volume mari de date.
- Permite **reutilizarea**: filtrele implementează prelucrări simple, care pot fi reutilizate în diferite aplicații.
- **Mentenanța** sistemului este mai simplă:
 - pot fi efectuate mai ușor modificări
 - filtrele sunt slab cuplate
- Oferă **flexibilitate în proiectarea sistemului**, fiind posibilă atât execuția secvențială cât și cea paralelă a filtrelor.
- **Toleranța la caderi**: dacă o instanță a unui filtru cade sau mașina pe care se execută devine nedisponibilă, prelucrarea aflată în curs pe acea instanță poate fi alocată altei instanțe.

Arhitectura *Pipe and filter* (6)

❖ Dezavantajele arhitecturii

- Flexibilitatea crescută a arhitecturii poate introduce complexitate, mai ales atunci când filtrele dintr-un pipeline sunt distribuite pe diferite mașini.
- Este necesară o infrastructură care să asigure că datele transferate între filtre nu se pierd.
- Nu este adecvată interacțiunilor.
- Poate introduce un overhead datorită transferului datelor între filtre.
- Pot să apară probleme în cazul instanțelor de filtre executate în paralel, atunci când una dintre instanțe cade: înainte de a cădea a modificat o informație de stare sau deja a postat un mesaj pentru etapa următoare.

Arhitectura dirijata de evenimente (1)

- Arhitectură distribuită frecvent utilizată.
- Adecvată sistemelor care colectează date în timp real și produc acțiuni ca răspuns la datele achiziționate.
- Alcătuită din componente care procesează evenimente asincron, în paralel.

Intr-un sistem bazat pe IOT exista 2 tipuri de elemente (“obiecte”):

- senzorii, care detecteaza producerea anumitor evenimente și genereaza semnale electrice
- elemente de actionare (actuators), care produc acțiuni ca răspuns la evenimentele detectate de senzori.

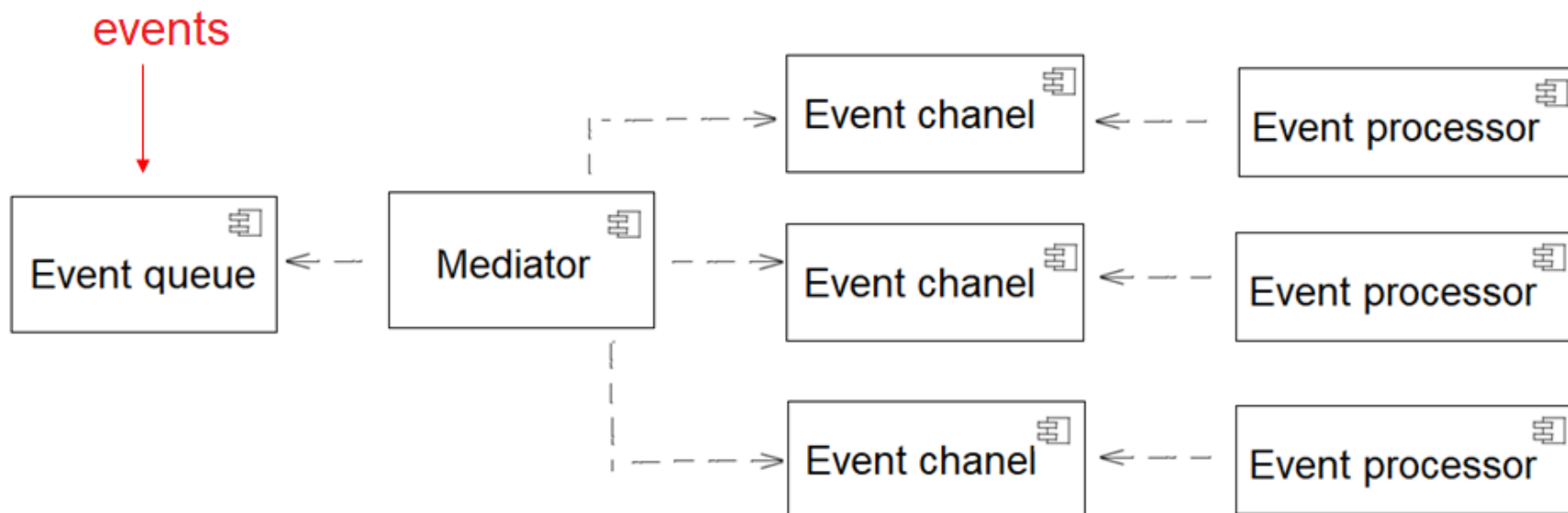
Exemplu:

- un termometru amplasat într-un spațiu este un senzor care produce un eveniment (semnal electric) atunci cand temperatura depășeste o anumită limită.
- rezultatul tratării evenimentului este comanda pornirii aparatului de aer condiționat

Arhitectura dirijata de evenimente (2)

Topologia Mediator

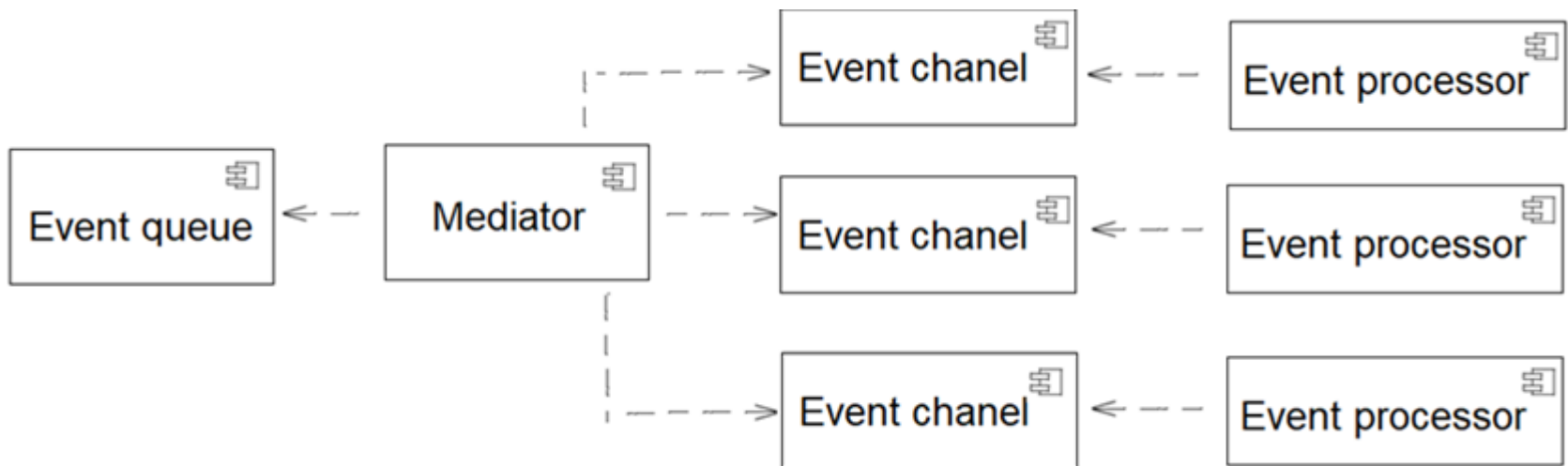
- 4 tipuri de componente: coada de evenimente, mediatorul, canale eveniment și procesoare de evenimente.
- Mediatorul**
 - *preia evenimente* din **coada de evenimente** și descompune tratarea fiecarui eveniment de intrare în pași de procesare care pot fi executați secvențial sau în paralel.
 - *genereaza evenimente de procesare* care sunt *transmise în canale eveniment specifice*.



Arhitectura dirijata de evenimente (3)

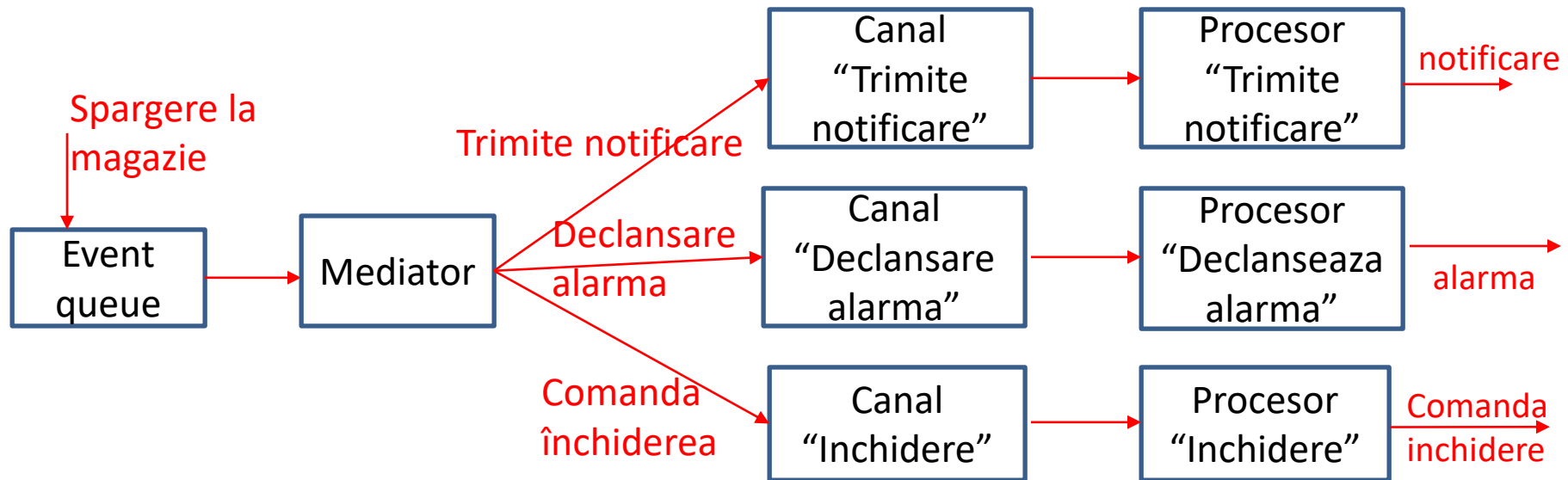
Topologia Mediator

- Fiecare **procesor de evenimente** preia evenimentele dintr-un canal de evenimente și efectuează prelucrări specifice, care implementează logica aplicatiei.
- **Canalele de evenimente** pot fi:
 - de tip “coada de mesaje” – evenimentele dintr-un canal sunt preluate de un singur processor;
 - de tip “topics and subscriptions” (un mesaj este transmis la mai mulți consumatori înregistrați pentru un anumit tip de mesaj) - evenimentele dintr-un canal sunt preluate și tratate în paralel de mai multe procesoare de evenimente.



Arhitectura dirijata de evenimente (4)

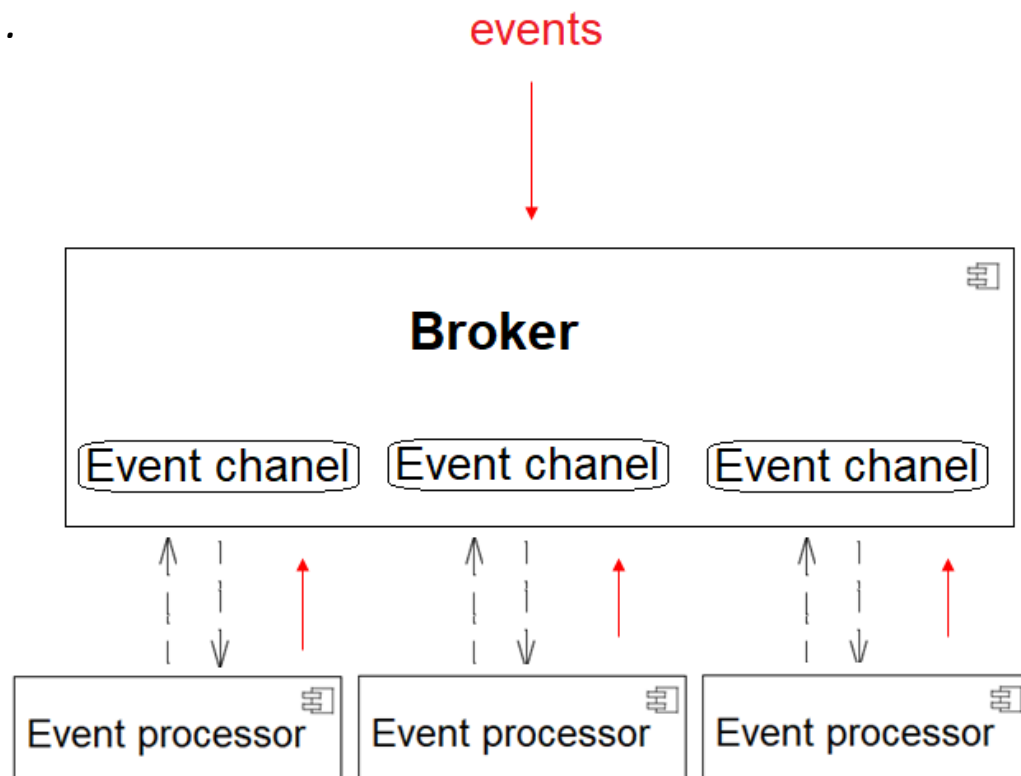
Exemplu de prelucrare într-un sistem cu topologie Mediator:



Arhitectura dirijata de evenimente (5)

Topologia Broker

- Broker-ul preia evenimentele și le dirijează către procesoarele de evenimente prin canale de evenimente specifice.
- Canalele de evenimente sunt conținute în componenta Broker și pot fi de tip “coada de mesaje” sau “*topics and subscriptions*”.
- Fiecare procesor de evenimente prelucrează un eveniment și poate genera un nou eveniment pe care-l trimite la Broker într-un canal de evenimente specific.



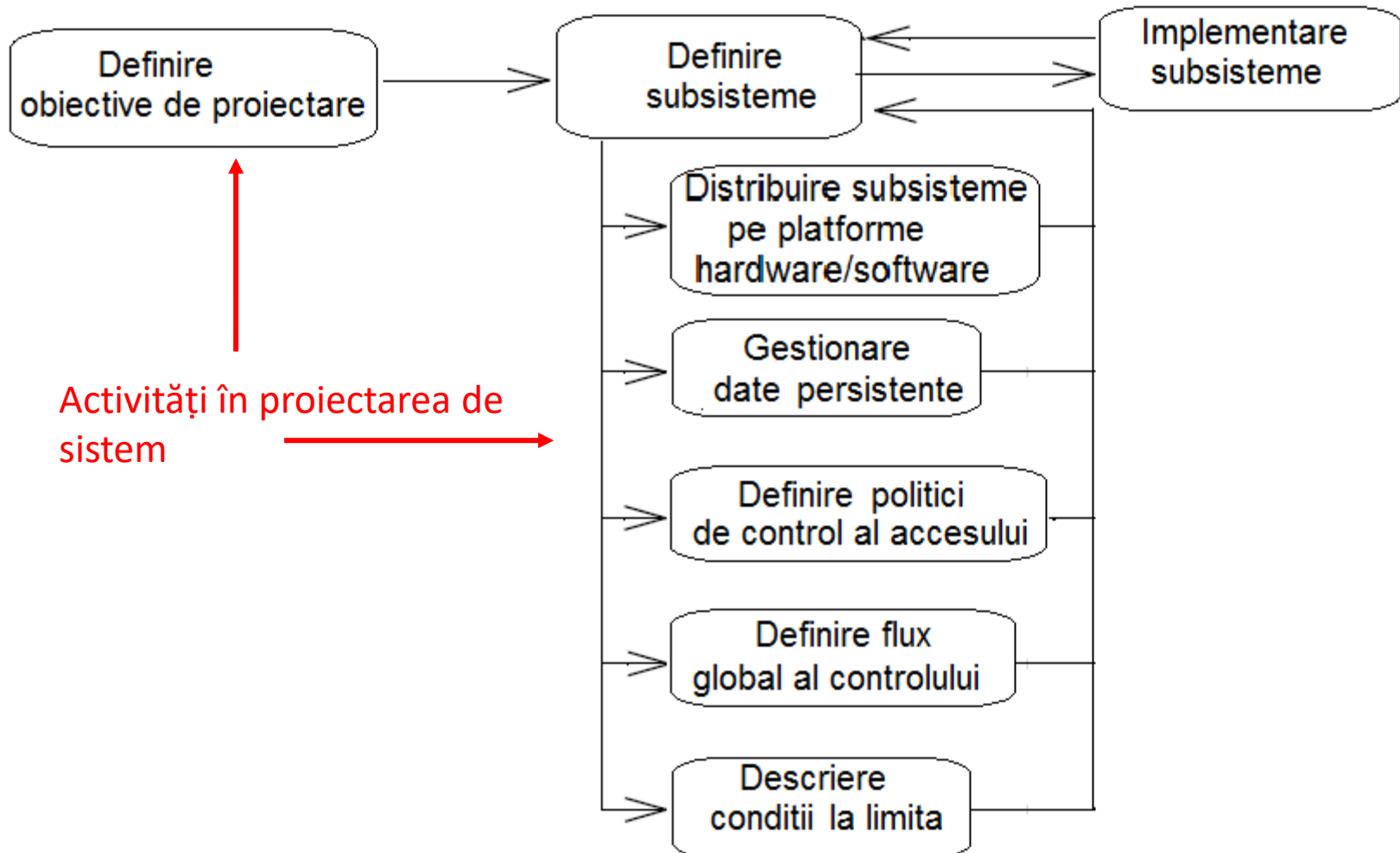
Arhitectura dirijata de evenimente (6)

avantaje si dezavantaje

- **Complexă:** prelucrare asincronă, distribuită.
- Lipsa de atomicitate în procesarea unei tranzactii (eveniment): **dificil de divizat procesarea în pasi executati în paralel.**
- **Scalabilitate mare.**
- Adecvata atat pentru aplicatii mici cat si pentru aplicatii foarte mari.
- **Componente foarte decuplate.**
- Componentele ***Event processor*** pot fi modificate fara a afecta pe celelalte.
In cazul topologiei *Mediator* schimbarea unei componente *Event processor* poate necesita o schimbare a componentei Mediator.
- **Testare dificila:** trebuie generate evenimente pt a simula natura asincrona a prelucrarilor.
- **Performanță înaltă** datorită prelucrărilor asincrone paralele care depășesc costul transferului de mesaje prin cozi.
- Dezvoltarea poate fi complicată: natura asincronă, tratarea condițiilor de cădere, procesoare de evenimente care nu răspund, etc.

Definirea arhitecturii sistemului

- proces iterativ -



Rafinarea descompunerii în subsisteme

Descompunerea initiala este ajustata iterativ in timpul celorlalte activitati ale proiectarii de sistem, urmarindu-se realizarea obiectivelor de proiectare:

- ❖ **Utilizarea de componente existente** (“Off-the-shelf”): descompunerea initiala este ajustata in acest scop. Astfel de componente pot realiza servicii complexe mai economic decat daca ar fi dezvoltate. Exemple: pachete de interfata utilizator, sisteme de baze de date, s.a.
- ❖ **Alocarea subsistemelor pe hardware**: atunci cand un sistem este distribuit pe mai multe noduri pot sa apara necesare si alte subsisteme pentru rezolvarea unor aspecte de fiabilitate si performanta.
- ❖ **Managementul datelor persistente**: pot fi necesare unul sau mai multe subsisteme de management al datelor persistente.
- ❖ **Politica de control al accesului utilizatorilor la resurse** poate influenta distributia acestora in subsisteme.
- ❖ Fluxul global al controlului are impact asupra interfetelor subsistemelor.
- ❖ **Conditiiile limita**: pornirea/oprirea sistemului, tratatea conditiilor speciale – **pot adauga subsisteme**

Distribuirea subsistemelor pe platforme hardware/software

- Multe sisteme se executa pe mai multe dispozitive de calcul si depind de accesul la Internet.
- Alocarea subsistemelor pe noduri hardware are consecinte importante asupra performantei si complexitatii sistemului: trebuie efectuata la inceputul proiectarii de sistem.
- Selectarea configuratiei hardware include si selectarea masinii virtuale pe care se va executa sistemul: sistemul de operare si alte componente necesare (sistemul de gestiune a bazei de date, sistemul de comunicare, s.a.)
- Selectarea configuratiei hardware si a masinii virtuale poate fi restrictionata de client: hardware existent, considerente de cost.
- Alte criterii: anumite componente trebuie sa se execute in locatii specifice (de ex., software pentru un bancomat), trebuie achizitionate echipamente de la un anumit producator, trebuie asigurate anumite conditii de comunicare, etc.
- Pentru reprezentarea alocarii subsistemelor pe echipamente si platforme software se folosesc **diagrame UML de distributie**.

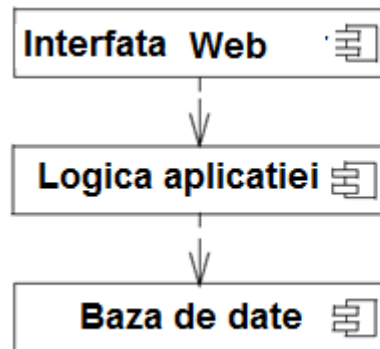
Identificarea și gestionarea datelor persistente

Date persistente:

- Date care nu se distrug la terminarea executiei aplicatiei care le-a creat
- Pot fi regasite si actualizate in cursul mai multor executii si posibil de mai multe aplicatii

Mecanisme pentru asigurarea persistentei:

- Sistem de fisiere: ieftin, simplu de implementat, gestiune de nivel coborat
- Bază de date: flexibil, scalabil, portabil, suporta scrieri/citiri concurente



Arhitectura SGCB (Sistemul de gestiune a cartilor din mai multe biblioteci)

Date utilizatori si carti

Gestionarea datelor persistente

❖ **Intrebari care influenteaza proiectarea unei baze de date**

- Care este rata de cereri estimata?
- Cat de des este accesata baza de date?
- Care este volumul tipic al datelor transferate la o cerere?
- Trebuie sa fie o baza de date distribuita?
- Datele trebuie sa fie arhivate?


❖ **Maparea modelului obiect UML pe o baza de date relationala:**

- Datele se memoreaza în tabele alcatuite din mai multe randuri
- Fiecare coloana a unei tabele reprezinta un atribut
- **Atributele unei clase corespund coloanelor unei tabele**
- **Un rand din tabela corespunde valorilor atributelor unei instante a clasei**
- **Asocierile dintre doua clase se implementeaza prin relatii între tabele.**

Controlul accesului

- ❖ Intr-un sistem multi-utilizator fiecare tip de utilizatori (actor) are anumite drepturi de acces la resursele sistemului.
- Se determina obiectele partajate de actori: fişiere, procese, baza de date, etc.
- Se defineşte mecanismul de control al accesului la obiectele partajate.
- In functie de cerintele de securitate se stabilesc regulile de autentificare a utilizatorilor si necesitatile de criptare a unor date.
- Accesul diferitilor actori la obiectele partajate poate fi modelat prin **matricea de control a accesului**:

Actor	Obiect A	Obiect B	Obiect C
Actor 1	Oper1A() Oper2A()		Oper1C() Oper3C()
Actor 2	Oper1A() Oper2A() Oper3A()	Oper1B() Oper2B() Oper3B()	
Actor 3	Oper3A()		Oper1C()



Drepturile de acces (operatiile pe care le poate efectua fiecare actor cu fiecare obiect partajat)

Fluxul global al controlului (1)

Fluxul controlului: secventierea actiunilor/operatiilor la executia sistemului.

Sunt 3 mecanisme de control al fluxului operatiilor într-un sistem:

- Dirijat procedural (procedure-driven control)
- Dirijat de evenimente (event driven control)
- Bazat pe fire de executie (thread-uri)

Control dirijat procedural

- Operatiile asteapta intrarile de la un actor atunci cand le sunt necesare.
- Este folosit in sistemele mai vechi si cele implementate in limbaje procedurale (cum sunt C, Pascal, Basic).
- Dificil de utilizat in limbajele orientate obiect, secventierea operatiilor fiind distribuita in seturi mari de obiecte.

Fluxul global al controlului (2)

Exemplu de flux al controlului dirijat procedural (Java): sistemul afiseaza mesaje si asteapta introducerea datelor de catre utilizator.

Stream in, out;

String userid, passwd;

out.println("Login:"); in.readLine(userid);

out.println("Password:"); in.readLine(passwd);

Control dirijat de evenimente

- Sistemul contine o bucla principala in care se asteapta un eveniment extern.
- Atunci cand se produce un eveniment, el este transferat obiectului corespunzator.
- Evenimentele pot fi tratate secvential sau in paralel
- Este o structura de control simpla, care centralizeaza toate intrarile in bucla principala.
- Este uzuala in sistemele bazate pe evenimente generate de interfata grafica utilizator
- Nu este adecvata implementarii secventelor in mai multi pasi.

Fluxul global al controlului (3)

Exemplu de flux al controlului dirijat de evenimente (Java):

Iterator subscribers, eventStream;

Subscriber subscriber; Event event; EventStream eventStream;

```
while (eventStream.hasNext()) {
```

```
    /* se extrage evenimentul din eventStream si se transmite obiectelor care s-au  
    inregistrat pentru acel eveniment */
```

```
    event = eventStream.next();
```

```
    subscribers = dispatchInfo.getSubscribers(event);
```

```
    while (subscribers.hasNext()) {
```

```
        subscriber = subscribers.next();
```

```
        subscriber.process(event);
```

```
    }
```

Obs: Urmatorul eveniment se extrage din eventStream numai dupa ce evenimentul curent a fost procesat de toate obiectele inregistrate.

Fluxul global al controlului (4)

Control bazat pe fire de executie

- Threadurile sunt executate in paralel
- Pot fi create/distruse la diferite momente de timp
- Intr-un thread se pot astepta intrari de la un actor.

Exemplu (Java): evenimentele sunt tratate in paralel

```
Thread thread; Event event; EventStream eventStream; EventHandler eventHandler;  
boolean done;
```

```
while (eventStream.hasNext()) {  
    event = eventStream.next();  
    eventHandler = new EventHandler(event)  
    thread = new Thread(eventHandler);  
    thread.start(); / * porneste executia threadului */  
}
```

- Sistemele bazate pe fire de executie sunt mai greu de depanat si testat.

Lecturi suplimentare – stiluri arhitecturale

1. <https://openclassrooms.com/en/courses/6397806-design-your-software-architecture-using-industry-standard-patterns/6896176-layered-architecture>

2. <https://www.jinfony.com/resources/bi-defined/3-tier-architecture-complete-overview/>

3. <https://martinfowler.com/eaDev/OrganizingPresentations.html>

4. <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html>