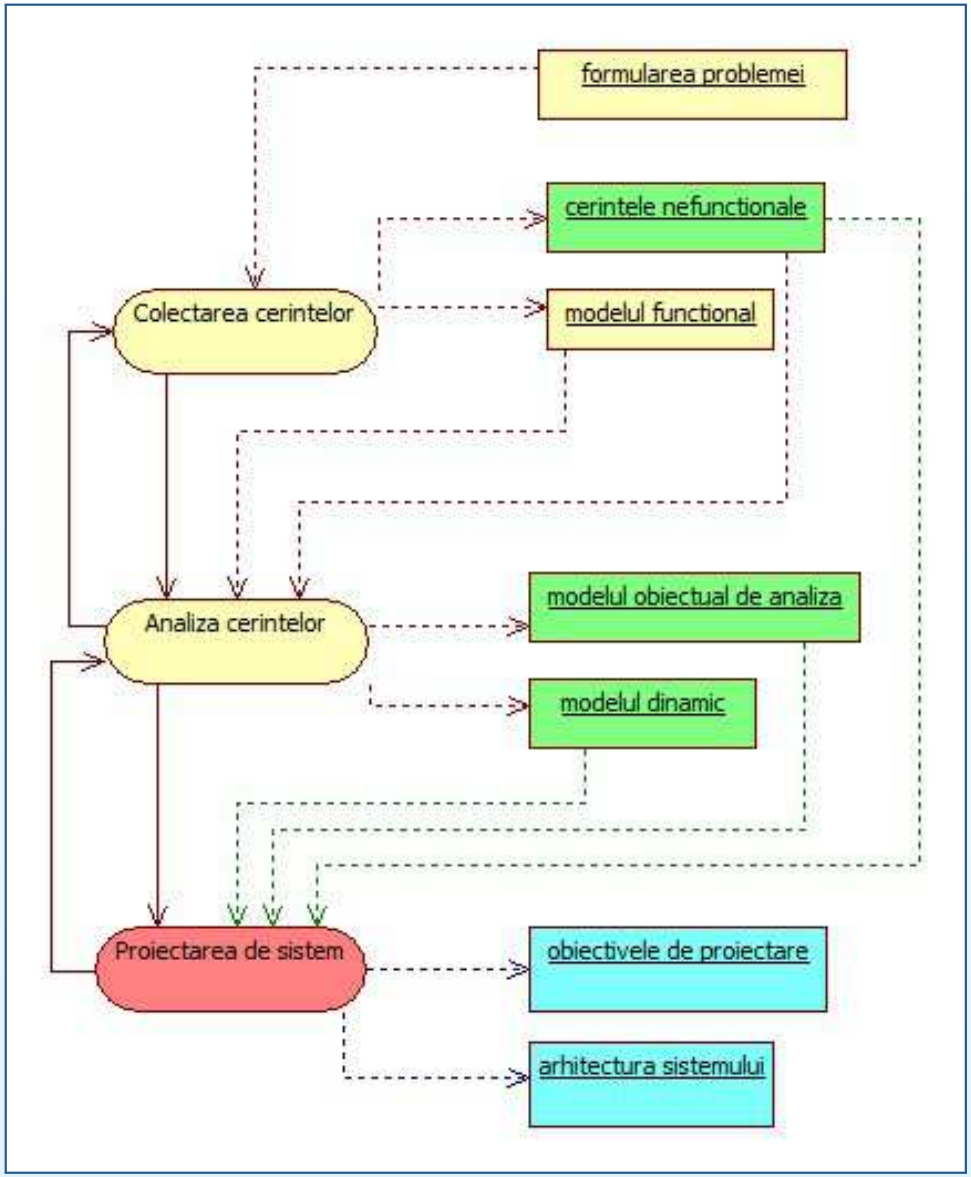
Curs 5

Proiectarea de sistem (I)

Suport de curs bazat pe ***B. Bruegge and A.H. Dutoit***

"Object-Oriented Software Engineering using UML, Patterns, and Java"

Proiectarea de sistem



Proiectarea de sistem (cont.)

- Procesul de transformare a modelului rezultat din ingineria cerinţelor într-un model arhitectural al sistemului
- Produse ale proiectării de sistem
  - *Obiectivele de proiectare* (eng. *design goals*)
    - Calităţi ale sistemului pe care dezvoltatorii trebuie să le optimizeze
    - Derivate din cerinţele nefuncţionale
  - *Arhitectura sistemului*
    - Subsistemele componente (de dimensiuni mai mici, asignabile unei subechipe de dezvoltare)
    - Responsabilităţile subsistemelor şi dependenţele între ele
    - Maparea subsistemelor la hardware
    - Strategii de dezvoltare: fluxul global de control, strategia de gesionare a datelor cu caracter persistent, politica de control a accesului

Proiectarea de sistem (cont.)

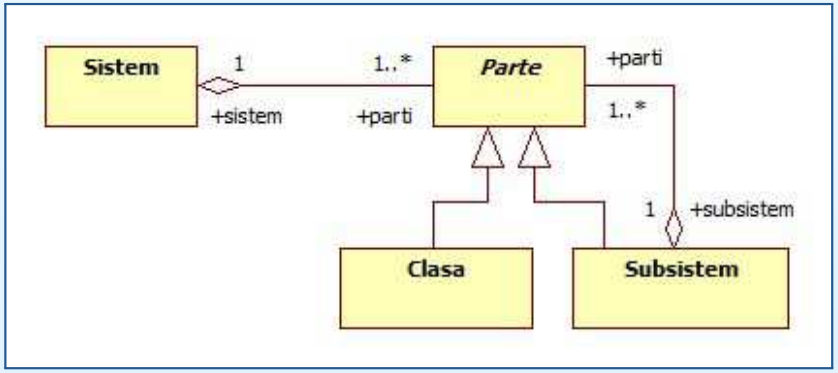
- Activităţi ale proiectării de sistem
  - *Identificarea obiectivelor de proiectare*
    - Identificarea şi prioritizarea acelor calităţi ale sistemului pe care dezvoltatorii trebuie să le optimizeze
  - *Descompunerea iniţială a sistemului*
    - Pe baza modelului funcţional şi a modelelor de analiză
    - Bazată pe utilizarea unor stiluri arhitecturale standard
  - *Rafinarea descompunerii iniţiale în vederea atingerii obiectivelor de proiectare*
    - Rafinarea arhitecturii de la pasul anterior, până la îndeplinirea tuturor obiectivelor de proiectare
- Analogie cu proiectarea arhitecturală a unei clădiri
  - Componente: camere vs. subsisteme
  - Interfeţe: pereţi/uşi vs. servicii
  - Reproiectare: mutarea pereţilor vs. schimbarea subsistemelor/interfeţelor

Concepte în proiectarea de sistem

- Subsisteme şi clase
- Servicii, interfeţe şi API-uri
- Coeziune şi cuplare
- Stratificare şi partiţionare
- Stiluri arhiecturale şi arhitecturi software

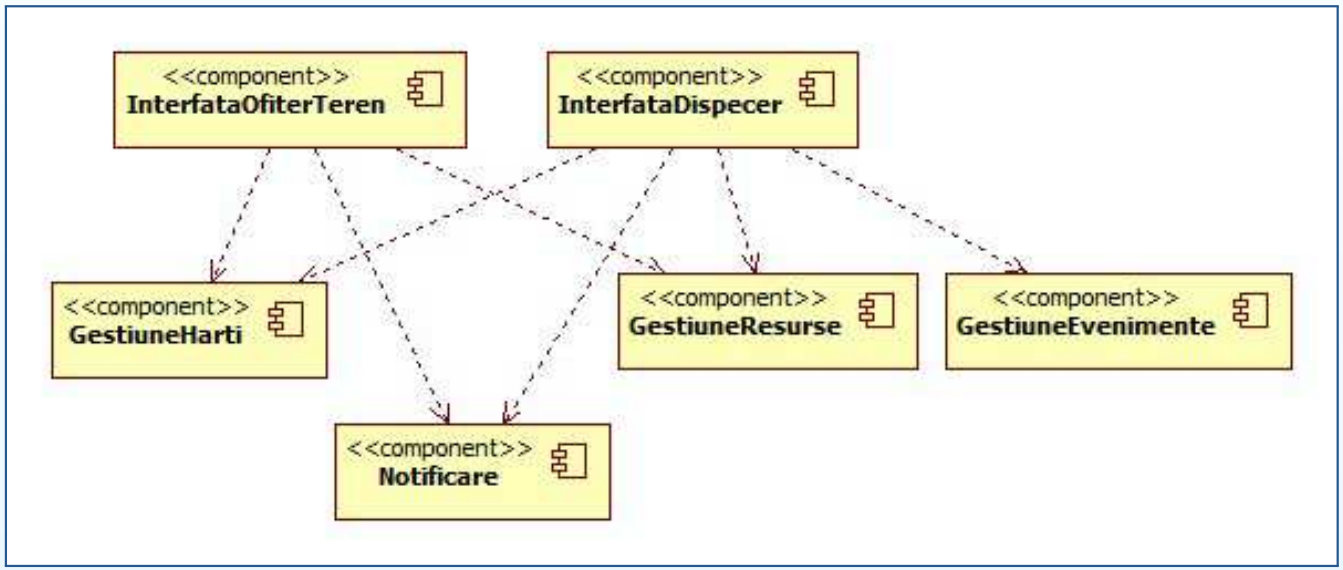
Subsisteme şi clase

- *Subsistem* = parte înlocuibilă a unui sistem (constând într-un număr de clase din domeniul soluţiei), caracterizată prin interfeţe bine definite, care încapsulează starea şi comportamentul claselor componente
  - Descompunerea în subsisteme permite gestionarea complexităţii ("divide et impera")
  - Un subsistem se dezvoltă, de regulă, de către un programator sau o echipă de dezvoltare
  - Prin descompunerea sistemului în subsisteme (relativ) independente, se permite dezvoltarea (relativ) concurentă a acestora
  - Sistemelor complexe le corespund mai multe nivele de descompunere (*Composite* pattern)



Subsisteme şi clase (cont.)

- Ex.: descompunere în subsisteme a sistemului SGA (diagramă UML de componente)



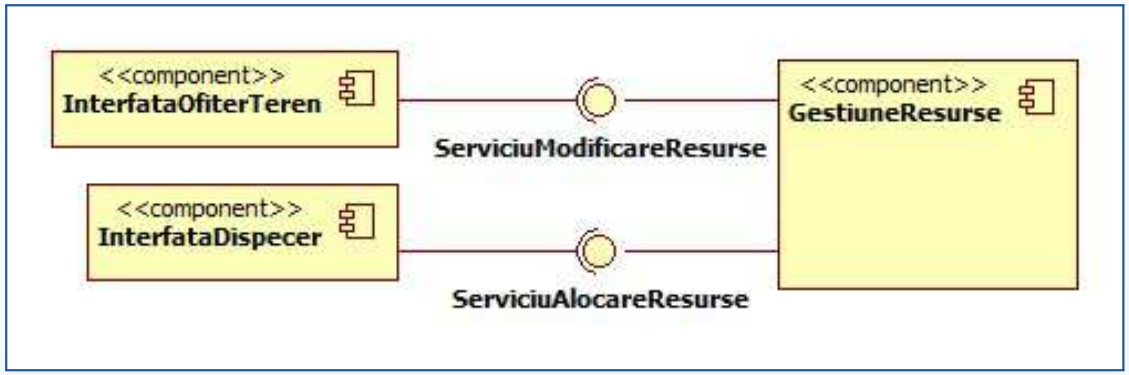
  - Subsistemele sunt reprezentate ca şi componente UML, cu relaţii de dependenţă între ele
  - O componentă UML poate reprezenta
    - O componenta logică = un subsistem ce nu are un echivalent runtime
    - O componenta fizică = un subsistem ce are un echivalent runtime

Servicii, interfeţe şi API-uri

- *Serviciu* = mulţime de operaţii înrudite (definite cu acelaşi scop)
  - Subsistemele sunt caracterizate de serviciile oferite altor subsisteme
  - Ex.: un subsistem care oferă un serviciu de notificare va defini operaţii de tipul *LookupChannel*(), *SubscribeToChannel*(), *UnsubscribeFromChannel*(), *SendNotification*()
  - Serviciile se identifică în timpul proiectării de sistem
- *Interfaţă* (a unui subsistem) = mulţime de operaţii UML înrudite, complet specificate (nume, tipuri parametri, tip returnat)
  - Rafinare a unui serviciu, specifică interacţiunile şi fluxul de informaţii dinspre şi înspre frontiera subsistemului (nu şi în interiorul acestuia)
  - Interfeţele se definesc în timpul proiectării obiectuale
- *API* (*Application Programming Interface*) = specificare a unei interfeţe subsistem într-un limbaj de programare
  - API-urile se definesc în etapa de implementare

Servicii, interfeţe şi API-uri (cont.)

- Ex.: Interfeţe/servicii oferite (eng. *provided*) şi solicitate/utilizate (eng. *required*)



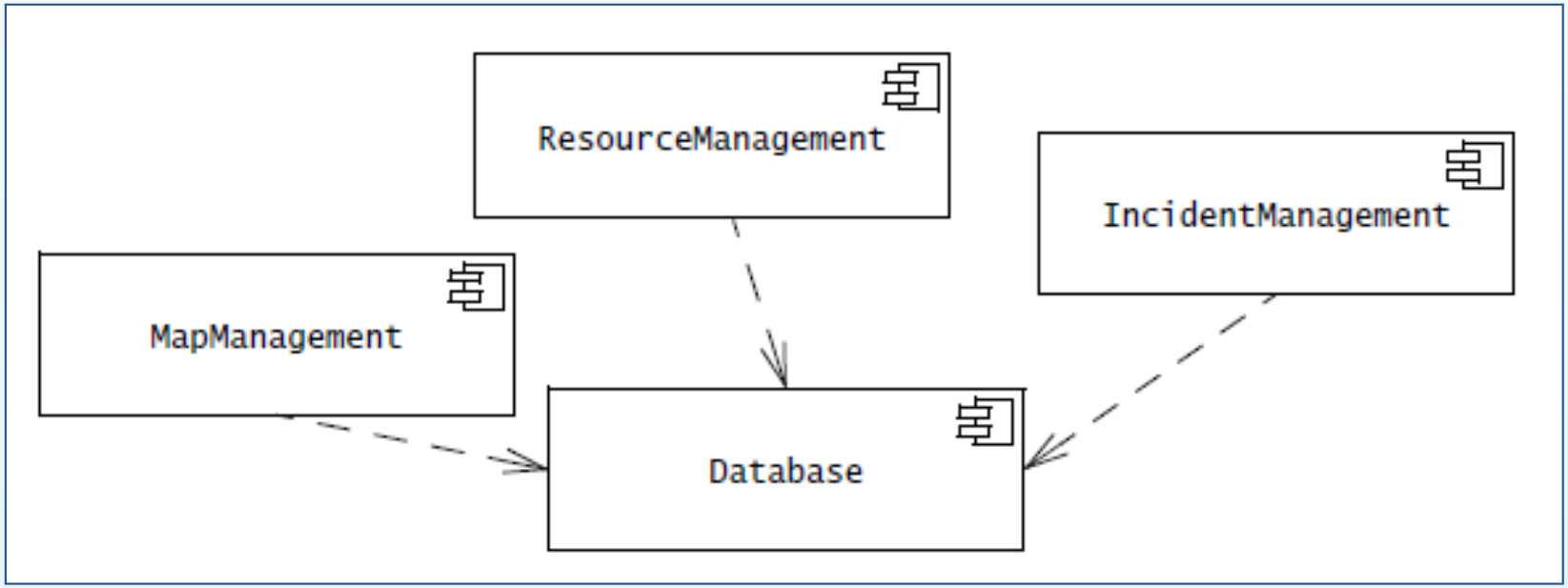
  - Notaţia UML: conectori *ball-and-socket*
    - Ball (lollipop) = interfaţă oferită, socket = interfaţă solicitată
    - Dependenţele dintre subsisteme se reprezintă prin cuplarea conectorilor ball cu cei socket
    - Reprezentare utilizată în momentul în care descompunerea în subsisteme e relativ stabilă şi focusul se schimbă de pe identificarea subsistemelor pe identificarea serviciilor (anterior se folosesc relaţii UML de dependenţă)

Coeziune şi cuplare

- *Cuplare* (eng. *coupling*) = măsură a dependenţei dintre două subsisteme
  - *Cuplare slabă* (eng. *low coupling*) - număr mic de dependenţe (subsisteme relativ independente)
  - *Cuplare strânsă* (eng. *strong coupling*) - număr mare de dependenţe (schimbările efectuate într-un sistem îl vor afecta şi pe celălalt)
  - Dezirabilă într-o descompunere: cuplarea slabă
  - Reducerea cuplării conduce, în general, la creşterea complexităţii prin introducerea de subsisteme suplimentare
- *Coeziune* (eng. *coesion*) = măsură a dependenţelor din interiorul unui subsistem
  - *Coeziune înaltă* (eng. *high coesion*) - subsistemul conţine un număr mare de clase puternic relaţionate şi care efectuează sarcini similare
  - *Coeziune slabă* (eng. *low coesion*) - subsistemul conţine un număr de clase nerelaţionate
  - Dezirabilă într-o descompunere: coeziunea înaltă
  - Creşterea coeziunii (prin descompunere în subsisteme cu coeziune înaltă) conduce şi la creşterea cuplării, prin dependenţele nou introduse

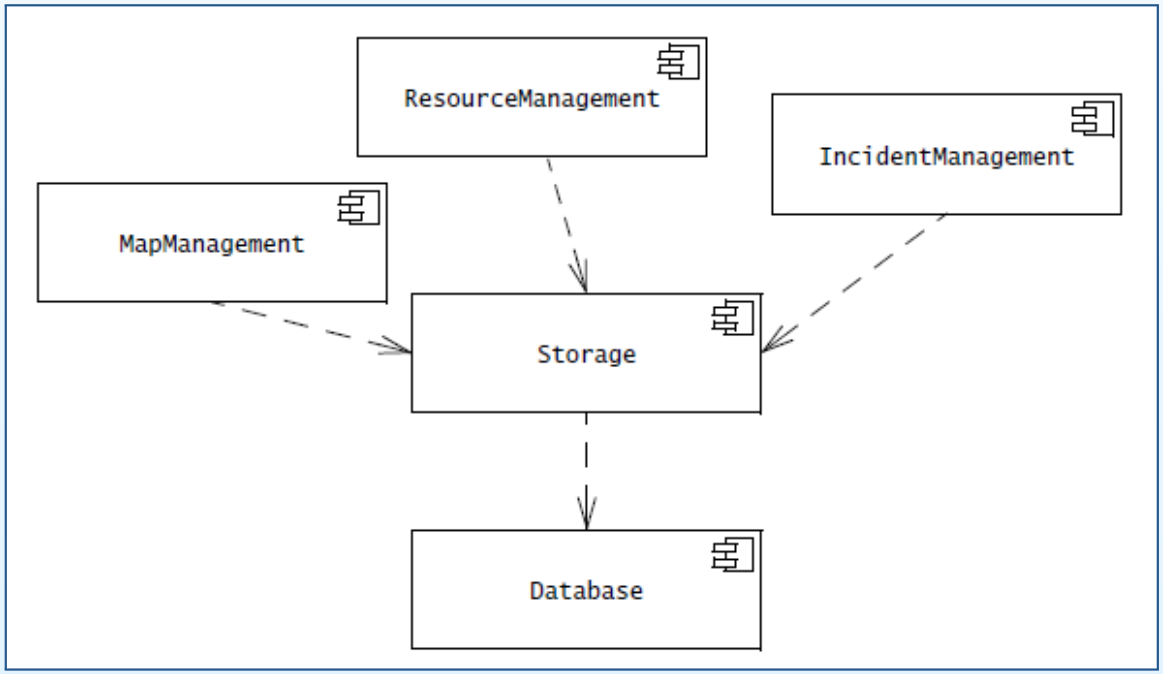
Cuplare

- Ex.: Subsisteme stâns cuplate
  - Subsistemele de gestiune trimit SGBD-ului comenzi SQL
  - Trecerea la un alt SGBD sau schimbarea strategiei de persistenţă (fişiere text) determină modificări la nivelul tuturor celor trei subsisteme de gestiune



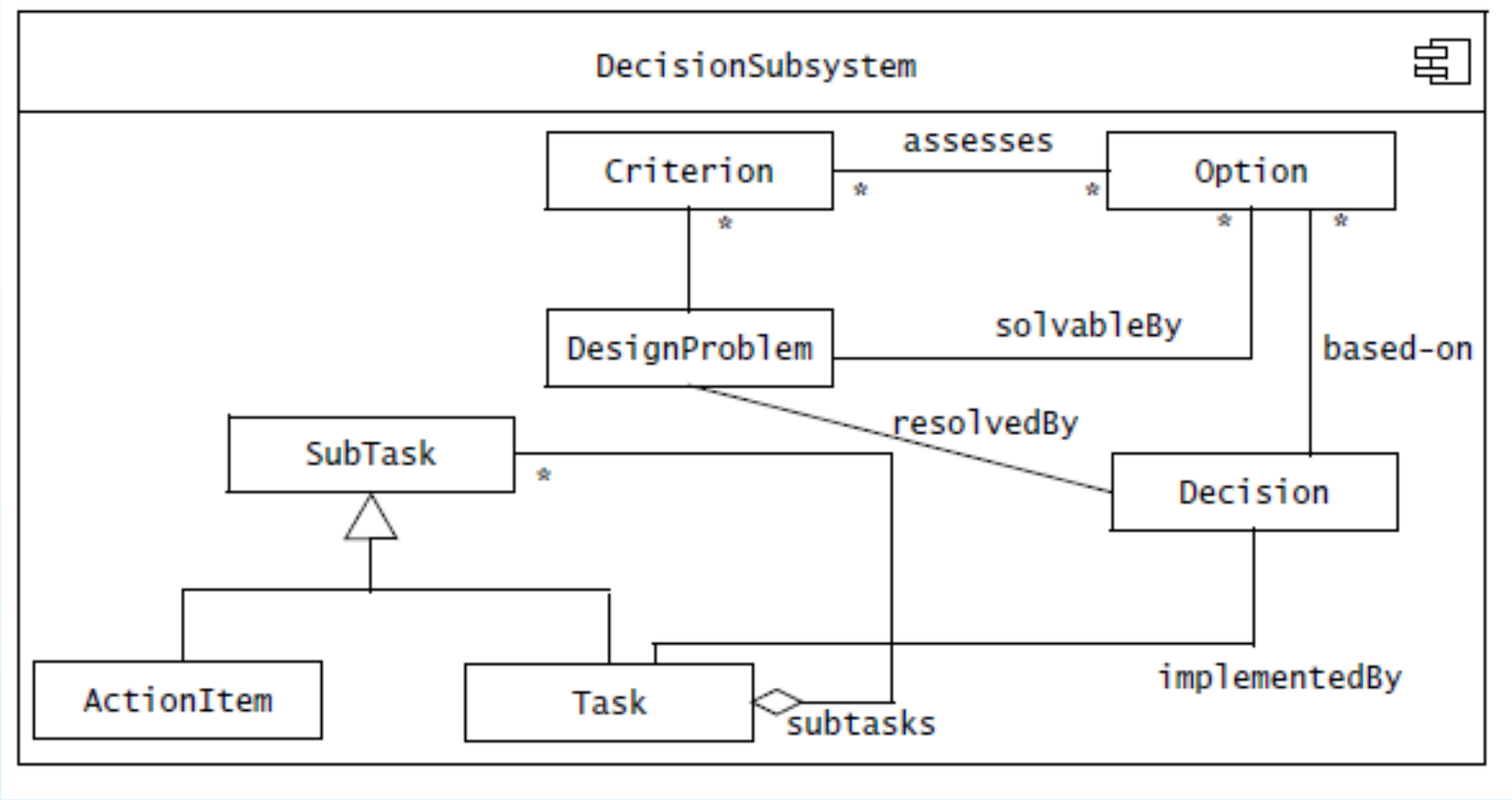
Cuplare (cont.)

- Ex.: Reducerea cuplării prin inserarea unui subsistem suplimentar
  - Noul subsistem izolează subsistemele de gestiune de SGBD
  - Subsistemele de gestiune utilizează doar serviciile oferite de noul subsistem, care va fi responsabil cu trimiterea de comenzi SQL spre SGBD
  - Trecerea la un alt SGBD sau schimbarea strategiei de persistenţă va determina doar modificări la nivelul subsistemului introdus



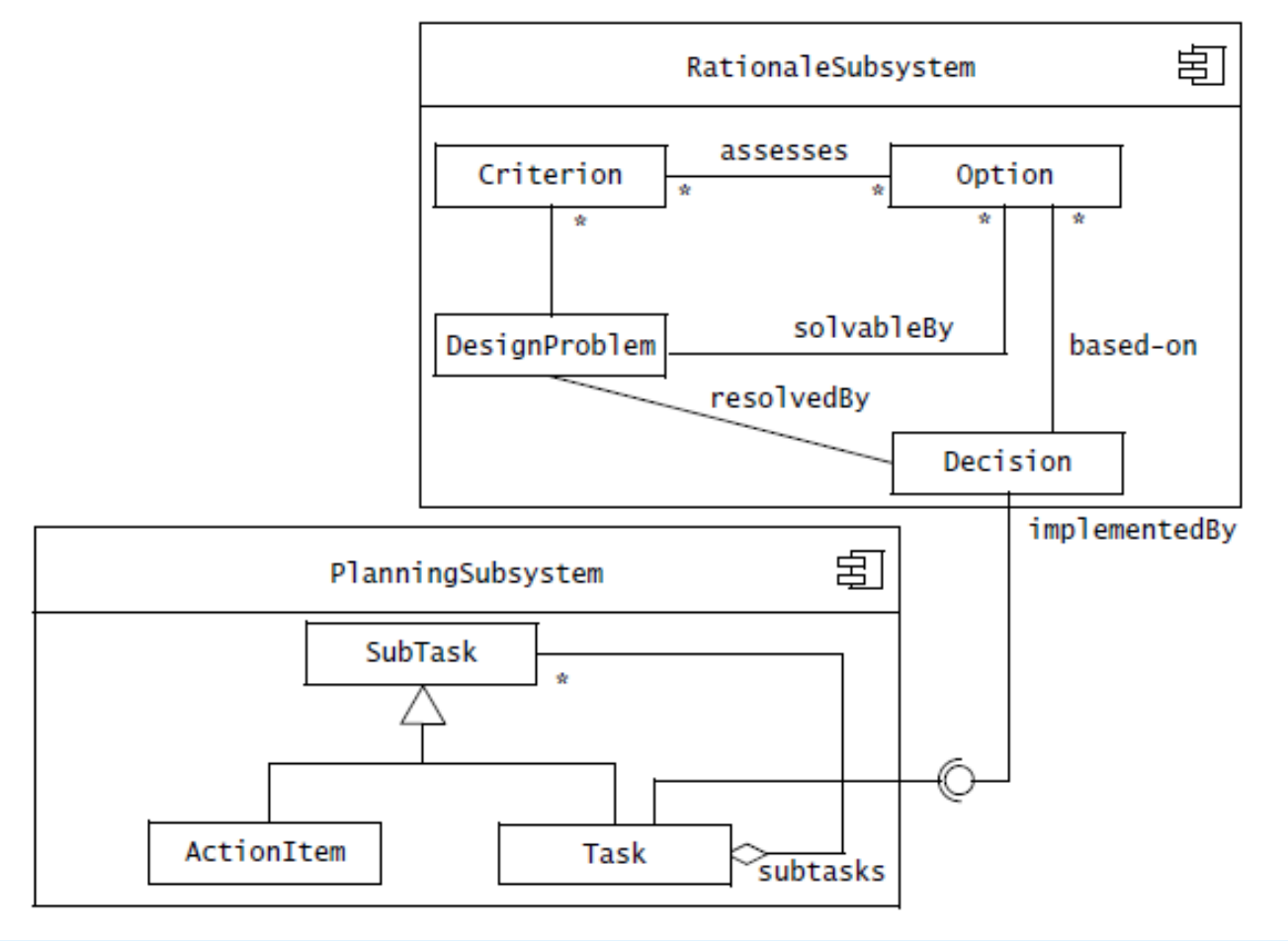
Coeziune

- Subsistem cu coeziune slabă
  - Clasele componente pot fi partiţionate în două submulţimi slab cuplate



Coeziune (cont.)

- Creşterea coeziunii prin descompunerea subsistemului iniţial



Stratificare şi partiţionare

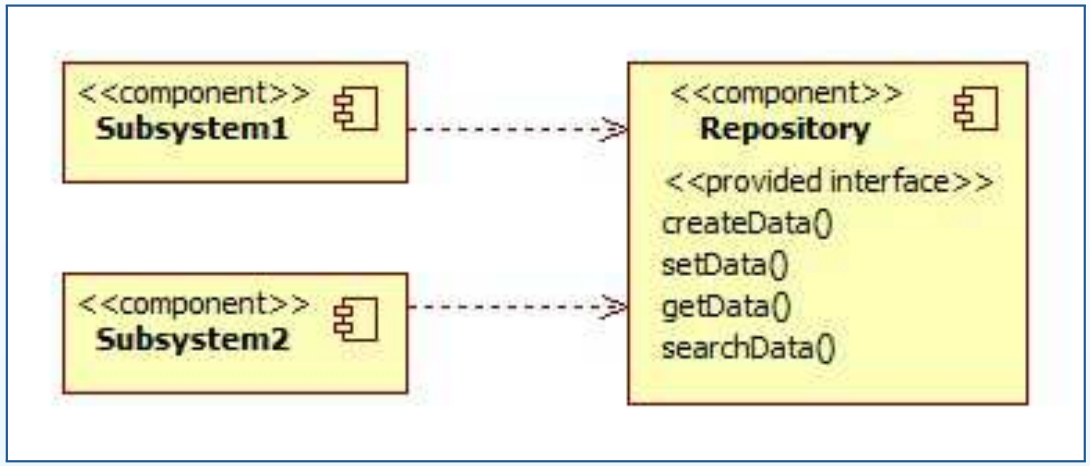
- *Descompunere ierarhică* a unui sistem (*stratificare*)
  - Generează o mulţime ordonată de *straturi* (eng. *layers*)
  - Un strat reprezintă un grup de subsisteme ce oferă servicii înrudite, eventual utilizând servicii dintr-un alt strat
  - Straturile sunt ordonate: un strat poate accesa doar servicii ale straturilor inferioare
    - *Arhitectură închisă* - fiecare strat poate accesa doar servicii din stratul imediat inferior (scop = modificabilitate/flexibilitate)
    - *Arhitectură deschisă* - un strat poate accesa servicii din oricare dintre straturile inferioare (scop = eficienţă)
- *Partiţionare*
  - Generează un grup de subsisteme la acelaşi nivel (eng. *peers*), fiecare dintre ele fiind responsabil de o categorie diferită de servicii
- În general, o descompunere a unui sistem complex este rezultatul atât al stratificării, cât şi al partiţionării

Stiluri arhitecturale şi arhitecturi software

- *Descompunerea sistemului* (eng. *system decomposition*) = identificarea subsistemelor, a serviciilor şi a relaţiilor între acestea
- *Stil arhitectural* (eng. *architectural style*) = şablon de descompunere a sistemelor
- *Arhitectură software* (eng. *software architecture*) = instanţă a unui stil arhitectural
- Exemple de stiluri arhitecturale
  - Repository
  - Model-View-Controller
  - Client-Server
  - Peer-to-Peer
  - Three-tier architecture
  - Four-tier arhitecture
  - Pipes and filters

Repository

- Subsistemele accesează şi modifică o singură structură de date centralizată, denumită *repository*

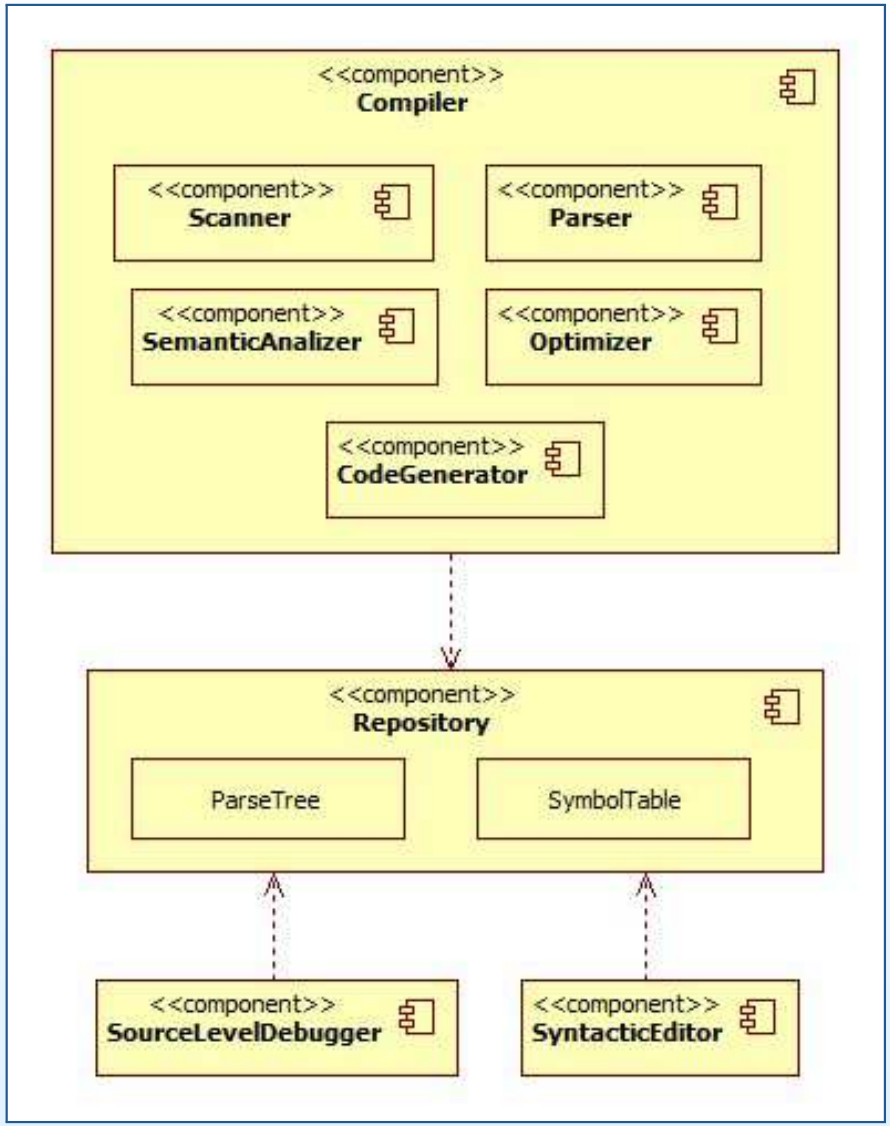


  - Subsistemele sunt relativ independente şi interacţionează doar prin intermediul repository-ului (cuplare slabă)
  - Fluxul de control poate fi dictat de repository (prin triggere) sau de către subsisteme (prin blocaje şi sincronizări)

Repository (cont.)

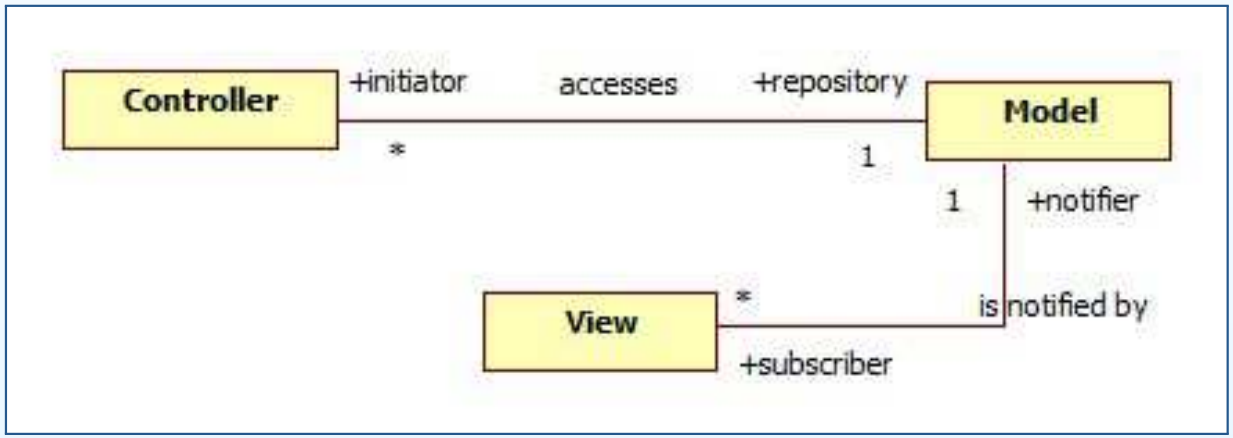
- Avantaje
  - Arhitectură utilă în cazul sistemelor cu necesităţi de procesare complexe, în continuă schimbare
  - Odată definit repository-ul, pot fi oferite servicii noi prin definirea de subsisteme adiţionale
- Dezavantaje
  - Subsistemele şi repository-ul sunt strâns cuplate, facând dificilă modificarea repository-ului fără a afecta subsistemele
  - Probleme de performanţă
- Utilitate
  - Sisteme de gestiune a bazelor de date
  - Compilatoare şi medii integrate de dezvoltare (eng. *Integrated Development Environments*, *IDEs*)

Exemplu de arhitectură Repository pentru un IDE



Model-View-Controller (MVC)

- Subsistemele sunt încadrate în una din trei categorii
  - *Model* - reprezintă informaţii/cunoştinţe din domeniul problemei
  - *View* - afisează aceste informaţii utilizatorului
  - *Controller* - translatează interacţiunile cu *view*-ul în acţiuni asupra modelului



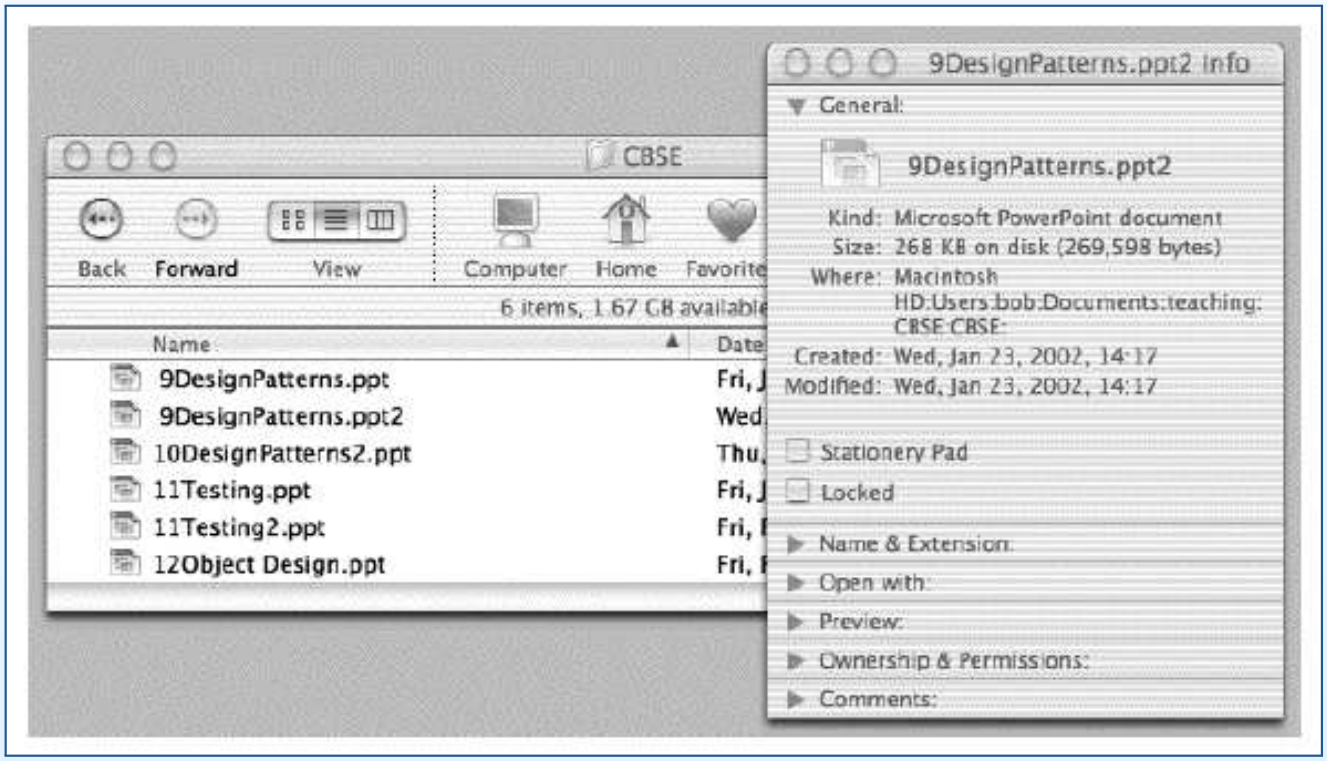
- Subsistemele model nu depind de nici un subsistem view sau controller
  - Modificările produse la nivelul acestora sunt propagate în subsistemele view prin intermediul unui protocol de înscriere/ notificare
  - Funcţionalitatea de înscriere/notificare este realizată, de obicei, cu ajutorul şablonului de proiectare *Observer*

Model-View-Controller (cont.)

- Justificare
  - Interfaţa utilizator (view-urile şi controller-ele) sunt mult mai predispuse spre schimbare decât informaţiile din model
  - Pot fi adăugate vederi noi, fără a modifica în alt fel sistemul
- Utilitate
  - Sisteme interactive, mai ales cele care necesită diferite vederi ale aceluiaşi model
- MVC este un tip particular de Repository, în care modelul corespunde structurii repository, iar fluxul de control este dictat de către obiectele controller

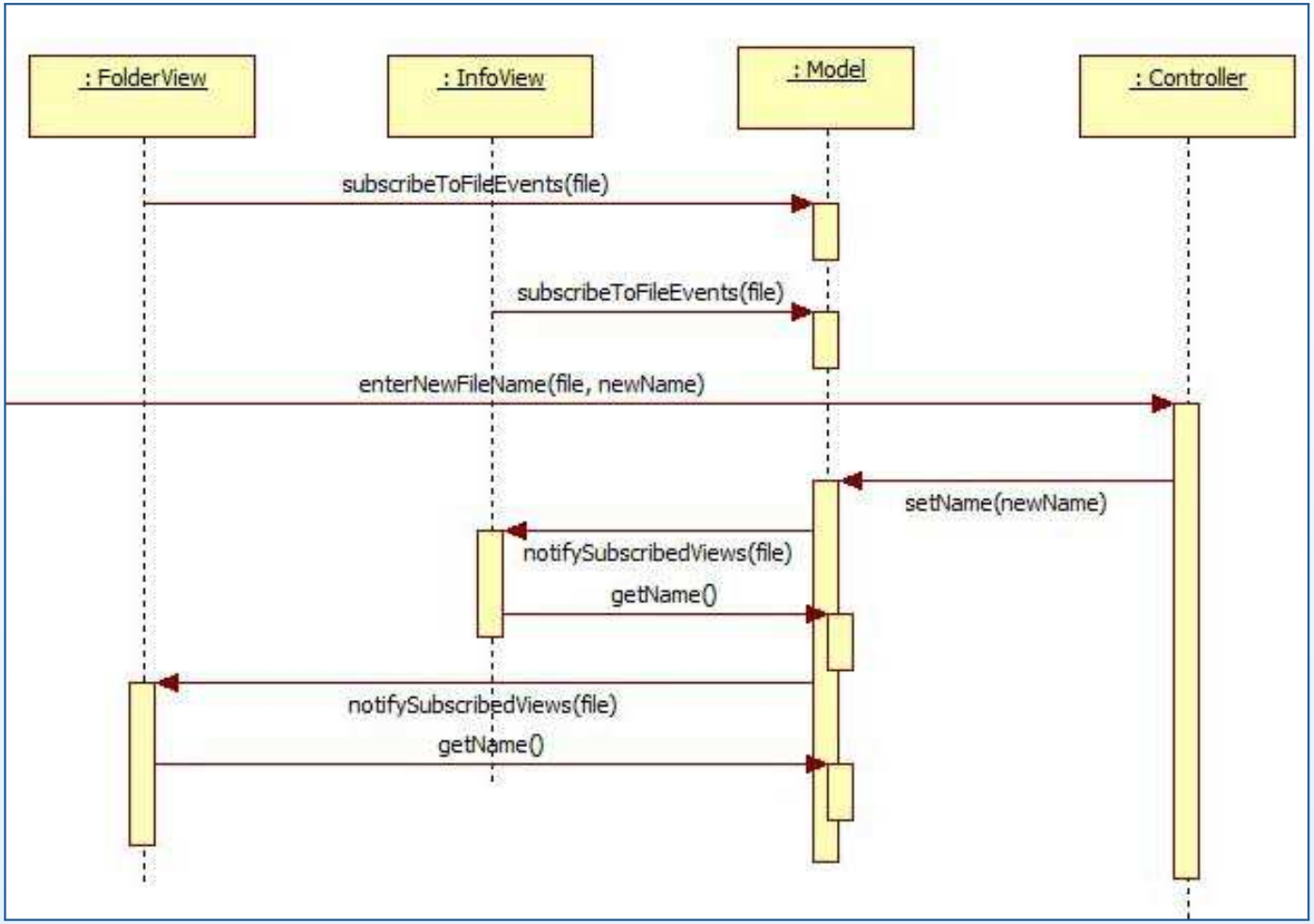
Exemplu de arhitectură MVC

- Modelul este reprezentat de fişierul *9DesignPatterns.ppt2*
- Una dintre vederi este fereastra CBSE, care listeaza conţinutul directorului cu acelaşi nume, cealalta este fereastra Info, care afişează informaţii relativ la fişierul *9DesignPatterns.ppt2*
- Schimbarea numelului fişierului determină actualizarea ambelor vederi



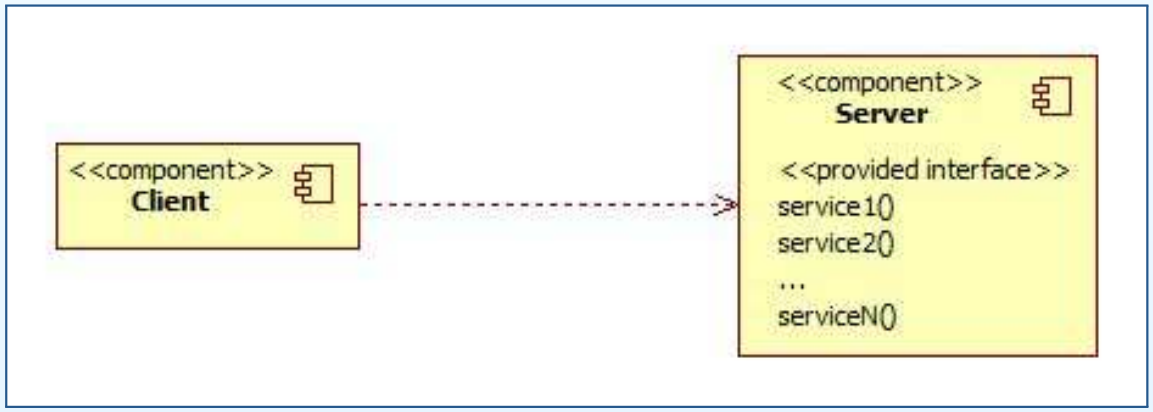
Exemplu de arhitectură MVC (cont.)

- Secvenţa de interacţiuni aferentă schimbării numelui fişierului *9DesignPatterns.ppt2*



Client-Server

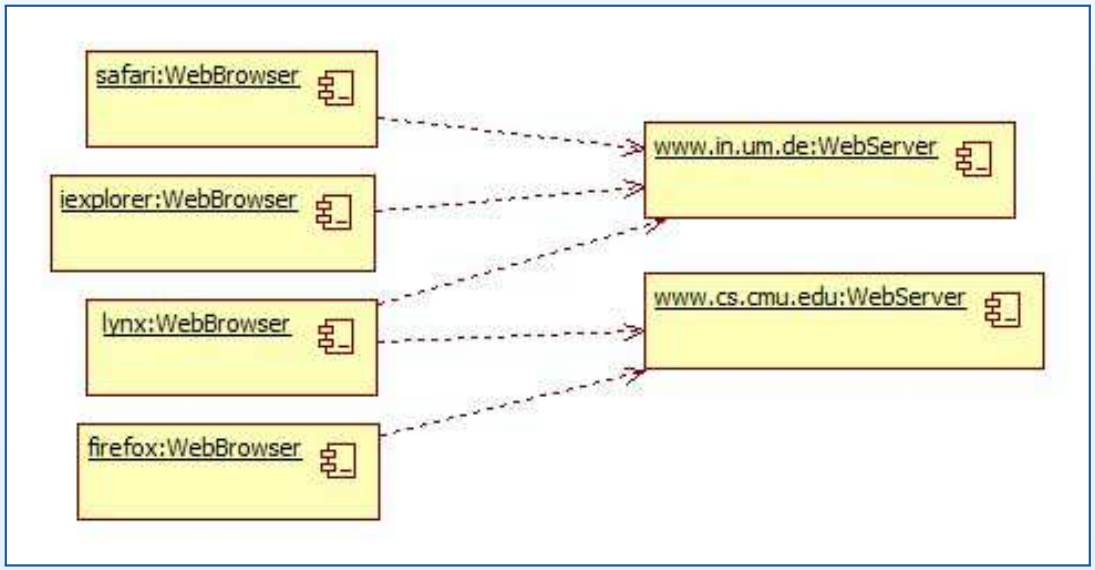
- Un subsistem, numit *server*, oferă servicii instanţelor unor alte subsisteme, numite *clienţi*, care sunt responsabile de interacţiunea cu utilizatorii
  - Solicitarea unui serviciu se face, de obicei, printr-un mecanism de apel la distanţă (Java RMI, CORBA, HTTP)
  - Fluxurile de control din server şi clienţi sunt independente, cu excepţia sincronizărilor pentru gestionarea cererilor şi primirea răspunsurilor



- Utilitate
  - Sisteme distribuite complexe, care gestionează un volum mare de date

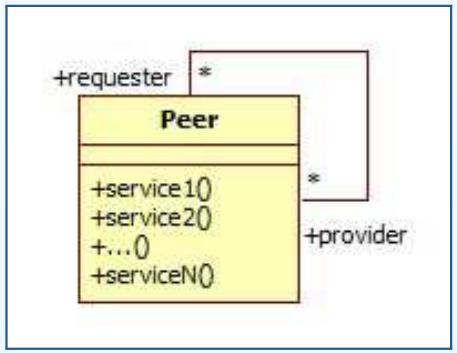
Exemple de arhitecturi client-server

- Un sistem informaţional cu o bază de date centralizată
  - Clienţii sunt responsabili de colectarea inputurilor utilizator, validarea acestora şi iniţierea tranzacţiilor cu baza de date
  - Serverul este responsabil de executarea tranzacţiilor şi garantarea integrităţii datelor
  - Îm acest caz, stilul client/server este o particularizare a stilului repository, în care structura de date centralizată este gestionată de un proces
- WWW - un client accesează date oferite de diverse servere



Peer-to-peer

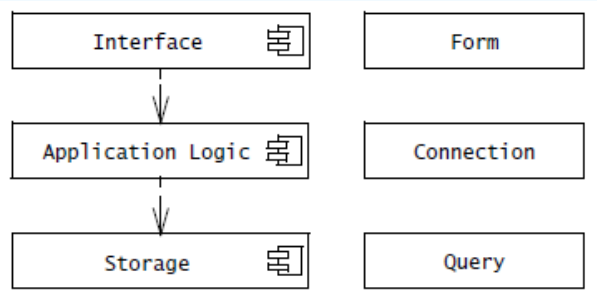
- Generalizare a stilului arhitectural client-server: un subsistem poate juca atât rol de client, cât şi de server (fiecare subsistem poate solicita şi oferi servicii)
  - Fluxurile de control sunt independente, cu excepţia sincronizărilor pe cereri



- Exemple
  - O bază de date care acceptă cereri de la o aplicaţie, dar o şi notifică atunci când se produc schimbări asupra datelor

Three-tier architecture

- Subsistemele sunt organizate pe trei straturi/nivele
  - *interfaţă* utilizator - include toate obiectele *boundary* care mediază interacţiunea cu utilizatorii (ferestre, forme, pagini Web, etc.)
  - *logica aplicaţiei* - include toate obiectele *entity* şi *control* care realizează verificările, procesările şi notificările cerute de aplicaţie
  - *accesul la date* - gestionează şi oferă acces la datele cu caracter persistent



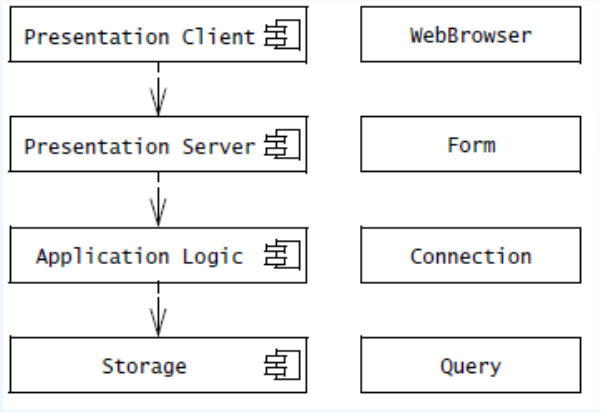
- Avantaje
  - Nivelul de acces la date joacă rolul repository-ului din stilul arhitectural omonim şi poate fi partajat de către aplicaţiile care operează asupra respectivelor date
  - Separarea dintre logica aplicaţiei şi interfaţă permite modificări ale interfeţei fără a afecta logica aplicaţiei

MVC vs. Three-tier architecture

- Stilul arhitectural MVC este nonierarhic (triangular)
  - Subsistemul *view* trimite cereri către subsistemul *controller*
  - Subsistemul *controller* actualizează subsistemul *model*
  - Subsistemul *view* este notificat de către subsistemul *model*
- Stilul arhitectural 3-tier este ierarhic (liniar)
  - Nivelul de prezentare nu comunică niciodată direct cu cel de date (arhitectură închisă)
- MVC nu acoperă problema persistenţei datelor

Four-tier architecture

- O variaţie a stilului arhitectural three-tier, în care nivelul *interfaţă* se descompune în
  - *prezentare client* - localizat pe maşinile client
  - *prezentare server* - localizat pe unul sau mai multe servere



- Avantaje
  - Pe nivelul prezentare client pot exista diferiţi clienţi, o parte a obiectelor *boundary* fiind reutilizate
  - Ex.: un sistem bancar include pe nivelul de prezentare client o interfaţă Web pentru utilizatori, un ATM şi o interfaţă desktop pentru angajaţii băncii. Formele partajate de toţi clienţii sunt definite şi procesate la nivelul prezentare server, eliminând redundanţa între clienţi

Pipes and filters

- *Pipeline* - lanţ de unităţi de procesare (procese, thread-uri, ...) aranjate astfel încât output-ul uneia reprezintă input-ul următoarei
- *Pipes and filters* - stil arhitectural constând din două tipuri de subsisteme, denumite *pipes* (canale) şi *filters* (filtre)
  - *Filter* - subsistem care efectuează un pas de procesare
  - *Pipe* - conexiune dintre doi paşi de procesare
- Fiecare subsistem filtru are un canal de intrare şi unul de ieşire
  - Datele preluate din canalul de intrare sunt procesare de către filtru şi trimise canalului de ieşire
- Ex. Unix shell: `ls -a | cat`