



Tema 3: Disseny Patrons arquitectònics

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2021/2022



UNIVERSITAT DE
BARCELONA

Temari

1	Introducció al procés de desenvolupament del software	
2	Anàlisi de requisits i especificació	
3	Disseny	3.1 Introducció
4	Del disseny a la implementació	3.2 Patrons arquitectònics
5	Ús de frameworks de testing	3.2 Criteris de Disseny: G.R.A.S.P. 3.4 Principis de Disseny: S.O.L.I.D. 3.5 Patrons de disseny

Tipus de patró

- i **Patrons d'arquitectura:** aspectes fonamentals d'un sistema software. Especifiquen un conjunt predefinit de subsistemes amb les seves responsabilitats i recomanacions. Usats en el disseny a gran escala i de gra guixut.
 - *Per exemple:* patró de **Capes**
- **Patrons de disseny:** utilitzats en el disseny dels subsistemes, especifiquen objectes i s'usen en frameworks de petita i mitjana escala. (micro-arquitectura)
 - *Per exemple,* patró **Façana** (*Facade*) per connectar les capes o el patró **Estratègia** per permetre algorismes connectables
- **Patrons d'estils o Idioms:** solucions de baix nivell orientades a la implementació, a un entorn de desenvolupament o a un llenguatge.
 - *Per exemple,* patró **Singleton** per fer una única instància d'una classe.

[Buschmann96]

3.2. Patrons arquitectònics

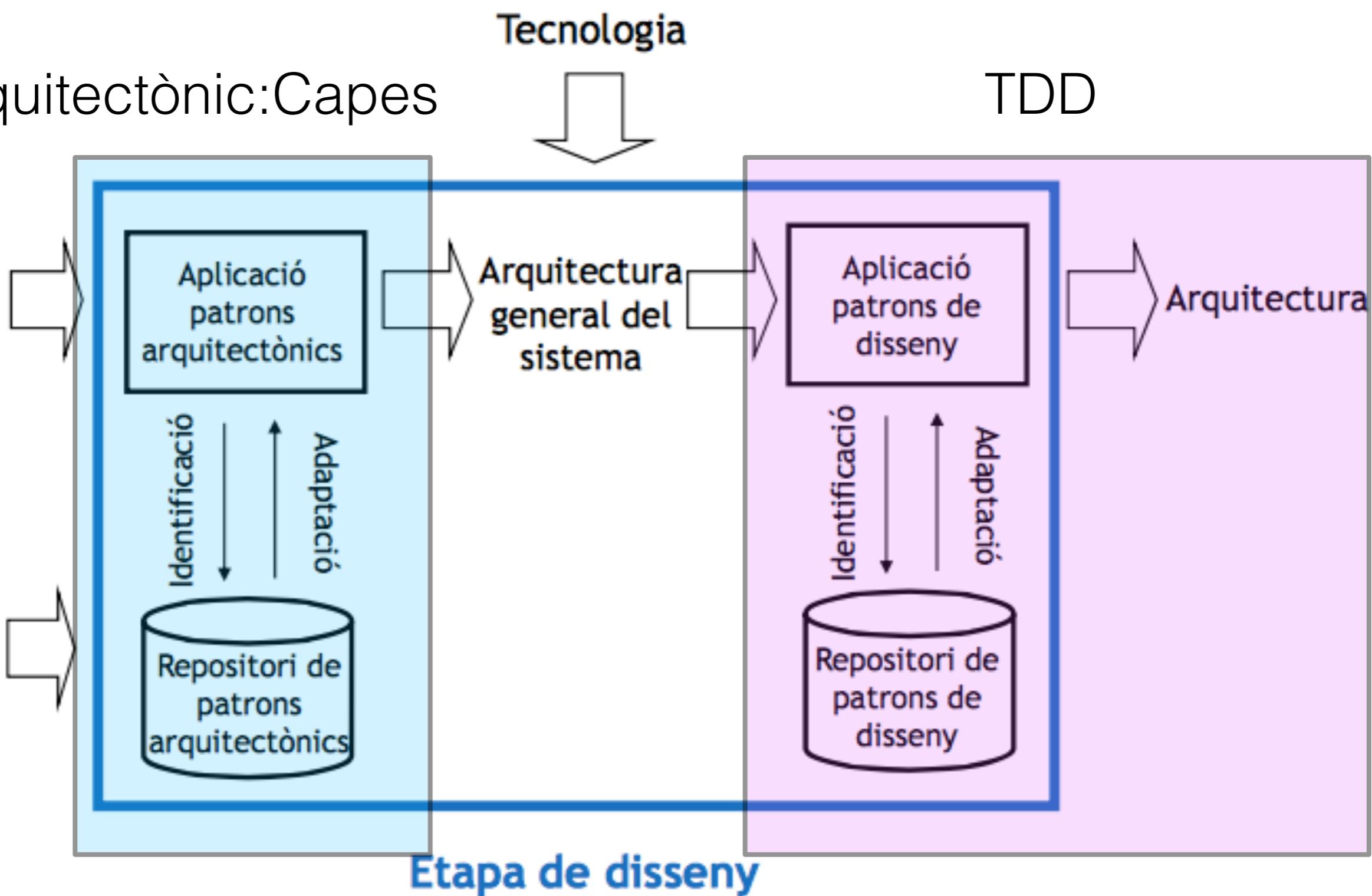
Patró arquitectònic:Capes

Tecnologia

TDD

Especificació

Requisits no
funcionals



3.2. Patrons arquitectònics

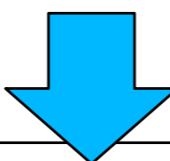
L'arquitectura del sistema és l'organització i estructura de les components i les seves connexions.

A partir de:

- Propietats que es volen assolir amb l'arquitectura (*requisits funcionals i no funcionals*)
- Recursos tecnològics disponibles

família de llenguatges de programació

família de sistema gestor de bases de dades, etc.



Els patrons (arquitectònics) s'usaran per definir l'arquitectura lògica del sistema (disseny a gran escala)

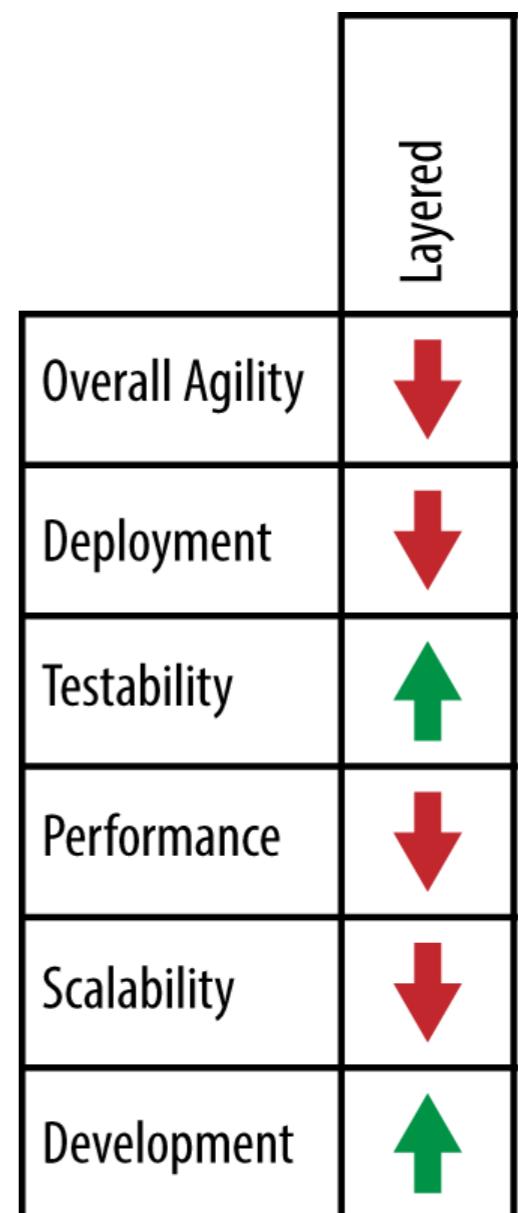
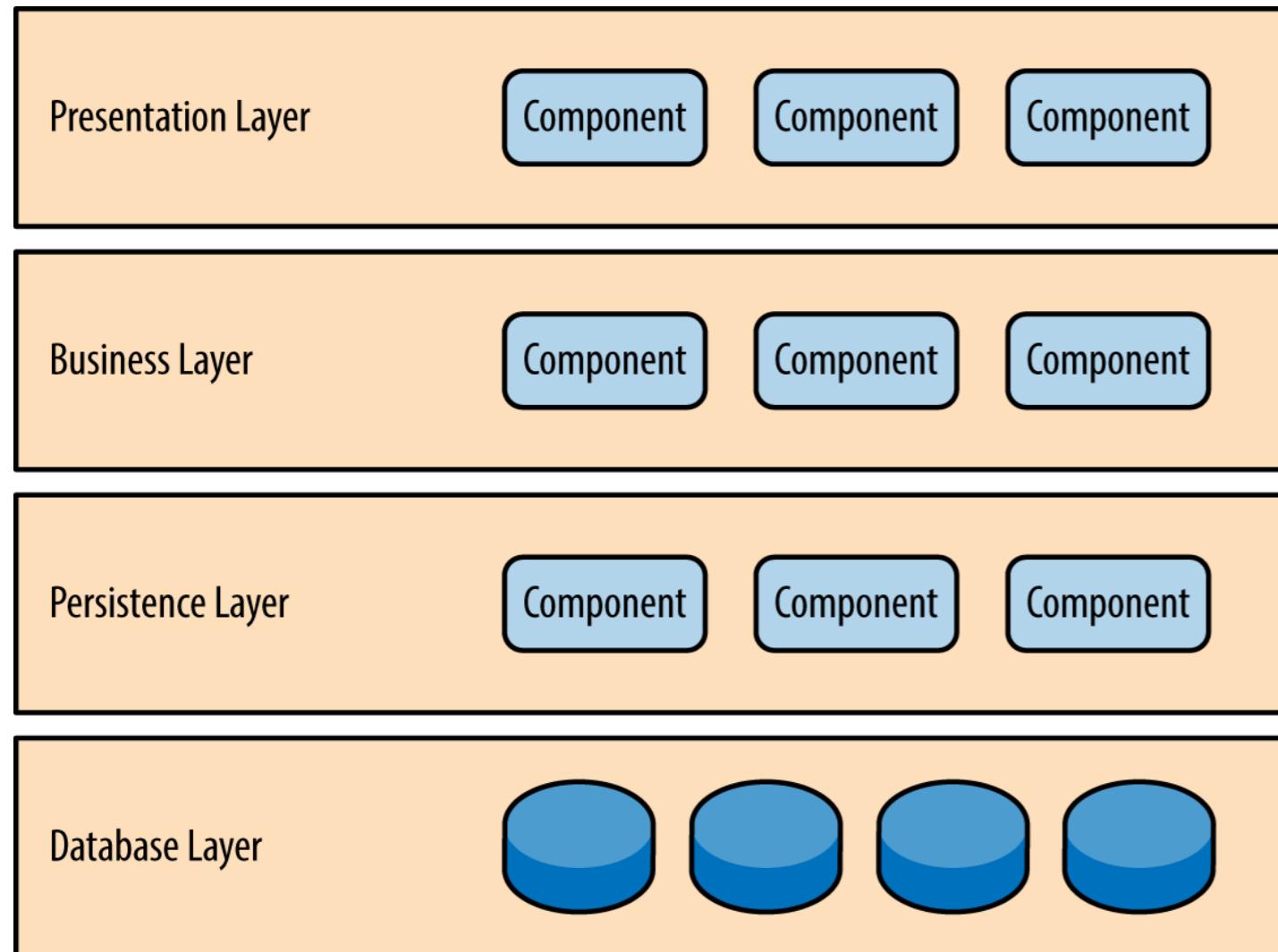
3.2. Patrons arquitectònics

- Patró per capes (Layers)
- Patró microkernels
- Patró events
- Patró microservices
- Patró space-based

3.2. Patrons arquitectònics

- **Patró per capes (Layers)**

Un sistema gran que requereix ser descomposat en grups de tasques (components), tals que cada grup de tasques està a un nivell determinat d'abstracció (o **capa**).



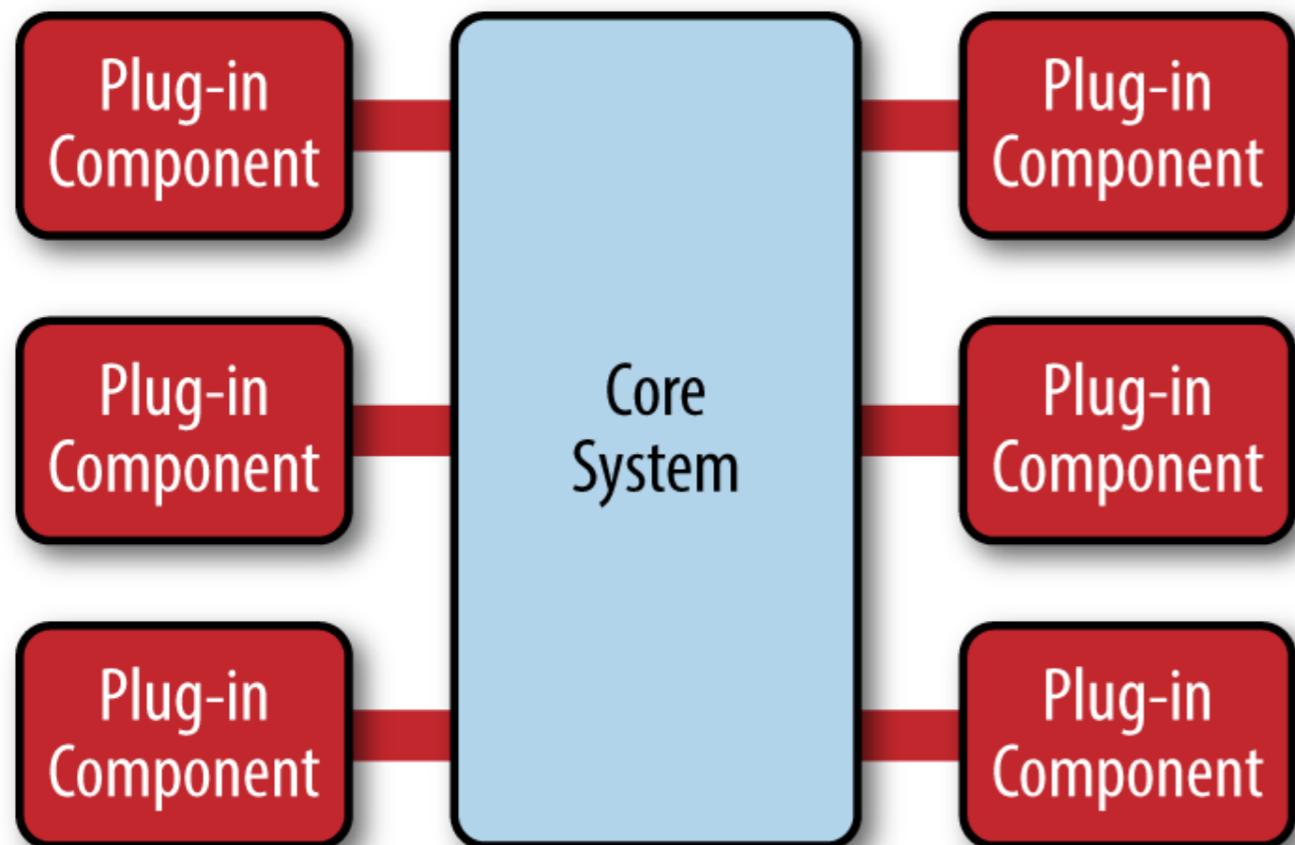
*architecture sinkhole anti-pattern
(regla del 20-80).*

3.2. Patrons arquitectònics

- Patró per capes (Layers)

- **Patró microkernels**

Patró de plug-in, La lògica de l'aplicació està dividida en diferents plugins o microkernels independents entre si.



Microkernel
Overall Agility
Deployment
Testability
Performance
Scalability
Development

Green arrows point upwards from the first five rows, indicating positive outcomes. A red arrow points downwards from the last row, indicating a negative outcome.

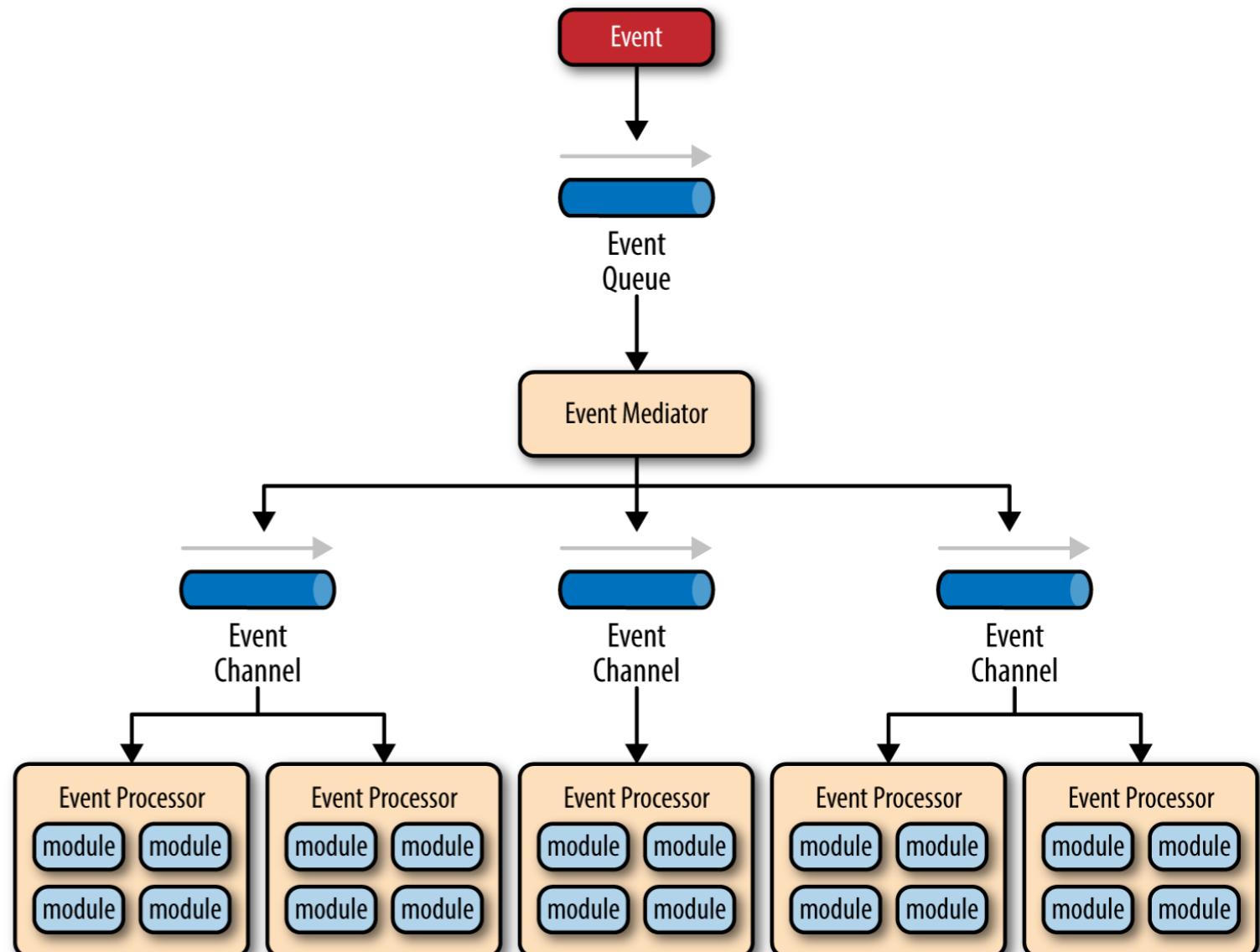
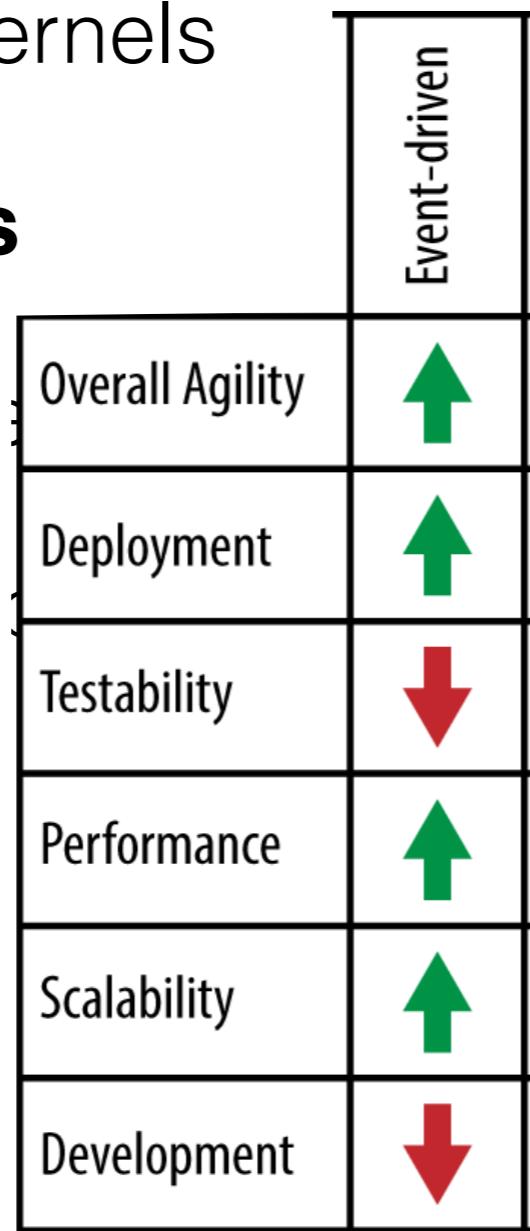
3.2. Patrons arquitectònics

- Patró per capes (Layers)

- Patró microkernels

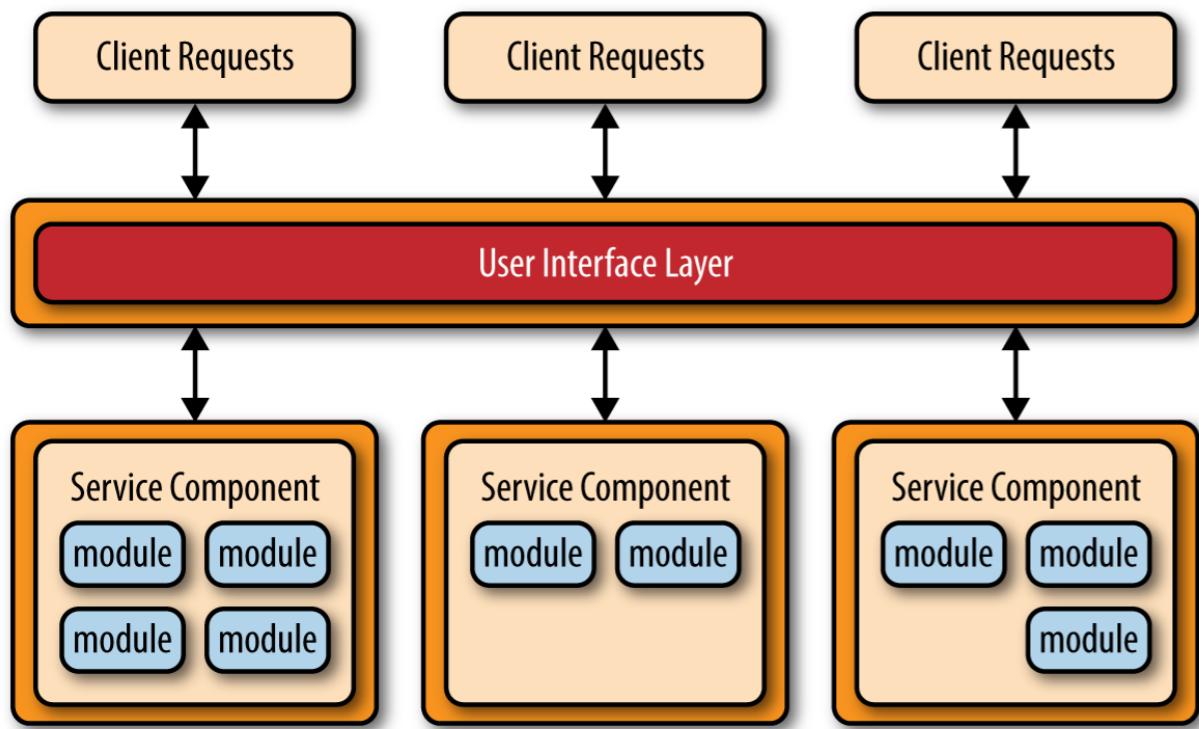
- **Patró events**

Patró per arquitectures asíncrones molt escalables.
Només es guarden els events que han passat, no les dades canviades.
Hi han dues arquitectures bàsiques: el Mediator i el Broker



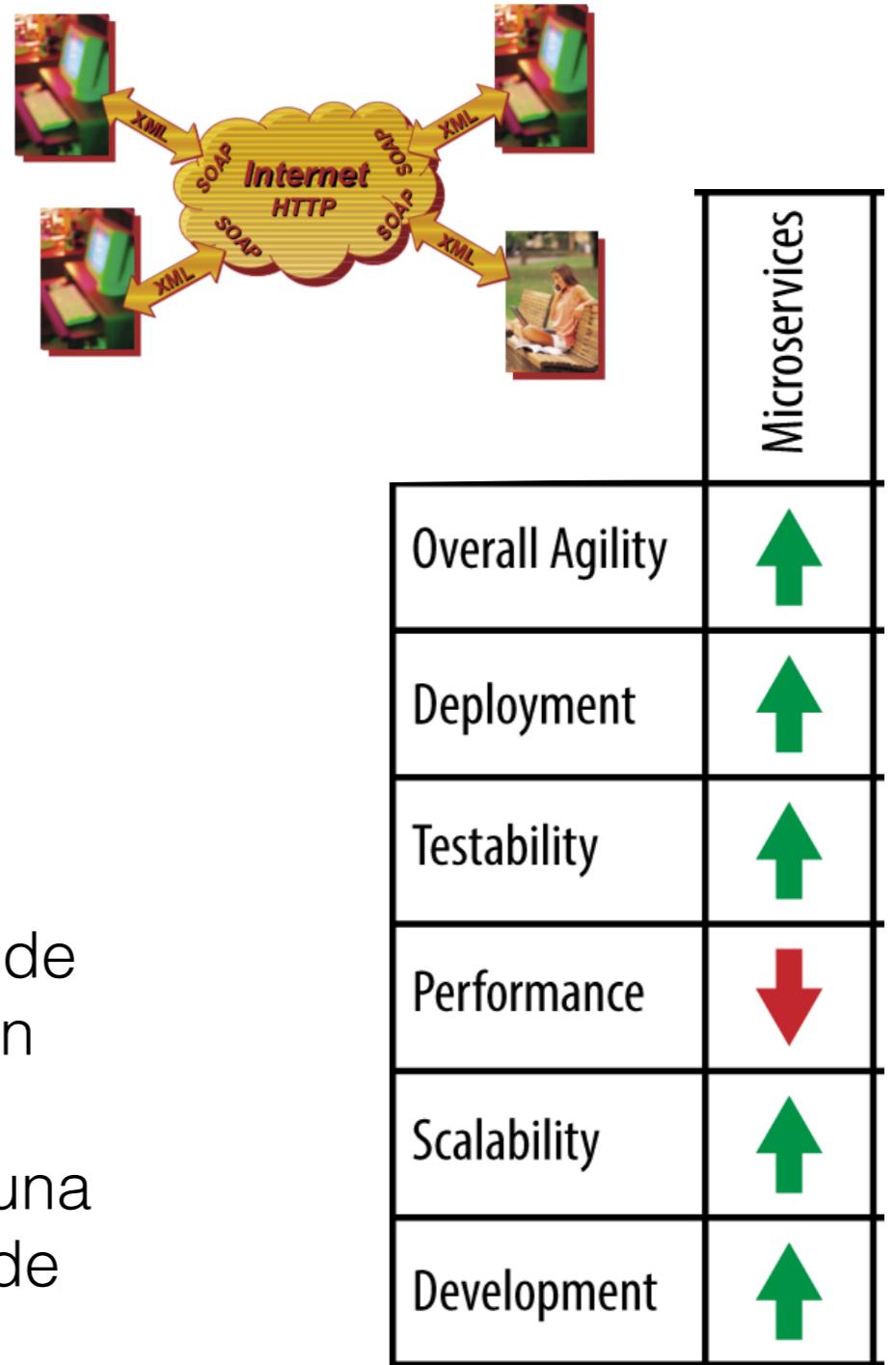
<https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing>

3.2. Patrons arquitectònics

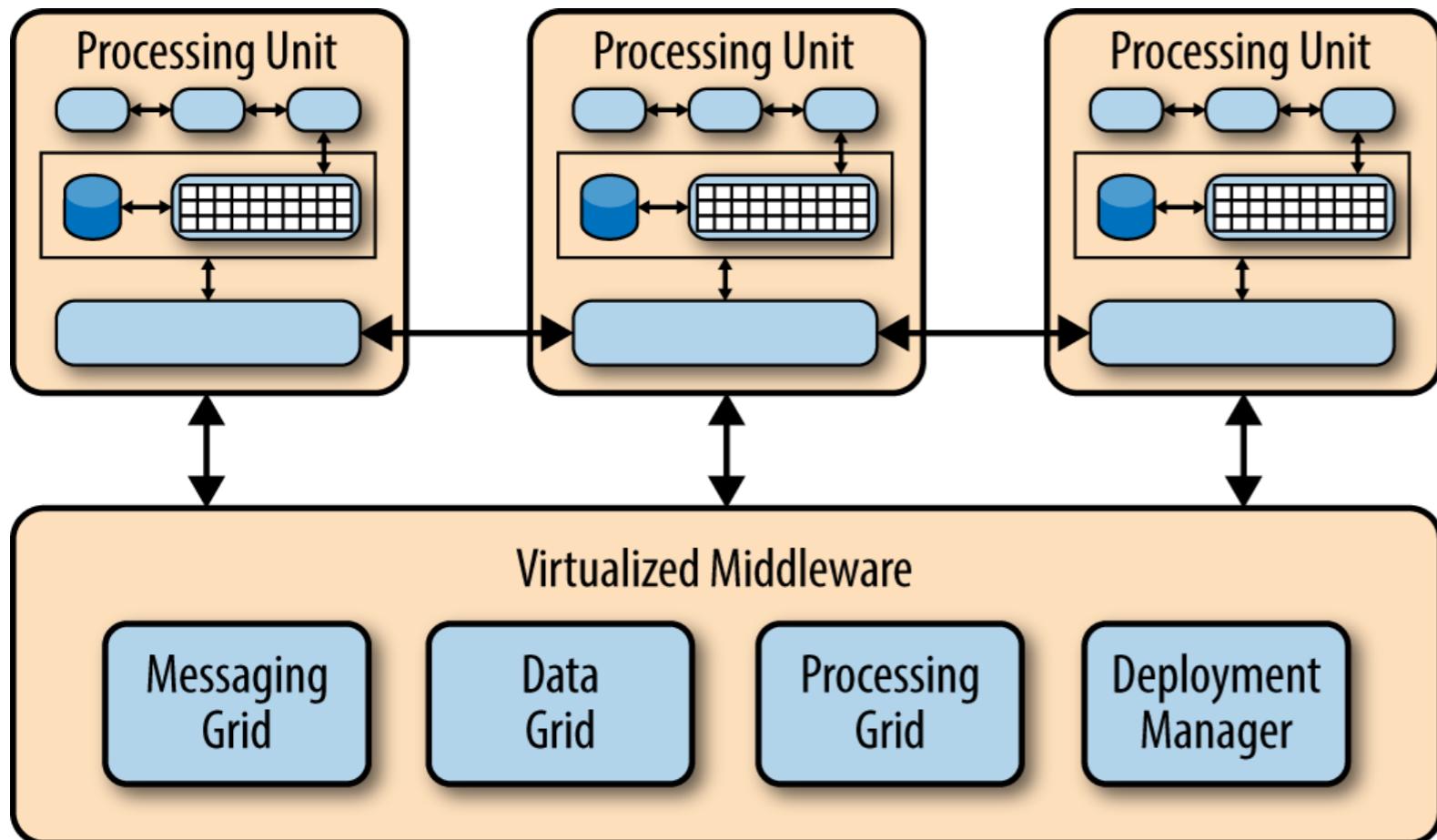


- **Patró microservices**

Arquitectura que ofereix **serveis** als components de l'aplicació mitjançant protocols de comunicació en xarxa. El principi bàsic és la independència dels clients, productes i les tecnologies. Un servei és una funcionalitat a la que es pot accedir remotament de forma immediata.



3.2. Patrons arquitectònics

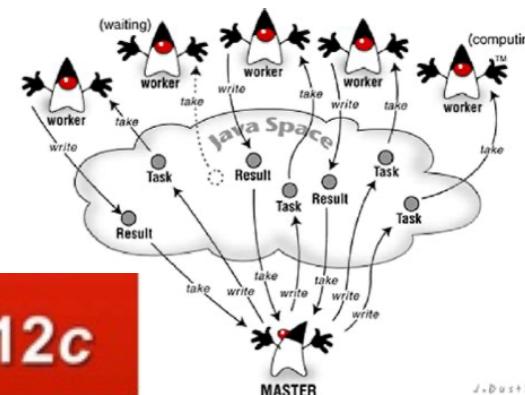


Space-based
Overall Agility
Deployment
Testability
Performance
Scalability
Development

... ↑ ↑ ↓ ↑ ↑ ↓

- **Patró space-based**

També coneぐut com Cloud architecture pattern. No hi ha una base de dades centralitzada sino que està distribuïda i replicada en grids. Les dades de l'aplicació es repliquen en els diferents unitats de procés.



IBM Cloud
Object Storage

Oracle Coherence 12c

3.2. Patrons arquitectònics



Tenda en línia:

- molts usuaris
- càlculs senzills
- elevat nombre de enviaments
- Interfície web genèrica
- Base de Dades de comandes i inventari

Microservices



Control de lloguers

- menys usuaris
- càlculs més complexs
- Interfície gràfica complexa
- Base de Dades complexa
- Integració amb paquets de serveis

Microkernel

3.2. Patrons arquitectònics



Reserves de butaques i control d'aforament i cancel·lacions:

- Relativament pocs usuaris
- Càlculs poc complexes
- Base de Dades canviable
- Seguint dels canvis

Events

Xarxa social:

- Petita aplicació web
- Molts usuaris
- Càrrega de dades contínua
- No es necessita una base de dades centralitzada
- Diferents tipus de dades utilitzades (per capes)

Space-based
Per capes

Gestió despeses PYME:

- pocs usuaris
- càlculs més complexs
- Preveure ampliació futura
- Base de Dades accessible per pocs usuaris
- Integració amb petites aplicacions

3.2. Patrons arquitectònics

	Layered	Event-driven	Microkernel	Microservices	Space-based
Overall Agility	↓	↑	↑	↑	↑
Deployment	↓	↑	↑	↑	↑
Testability	↑	↓	↑	↑	↓
Performance	↓	↑	↑	↓	↑
Scalability	↓	↑	↓	↑	↑
Development	↑	↓	↓	↑	↓

3.2. Patrons arquitectònics

Nom del patró: **Patró per Capes**

Problema

- Els canvis de codi es propaguen al llarg de tot el sistema
- La lògica de l'aplicació es barreja amb la interfície d'usuari
- La lògica de l'aplicació es lliga al serveis tècnics
- Alt acoblament entre diferents àrees d'interès: difícil manteniment i poca flexibilitat

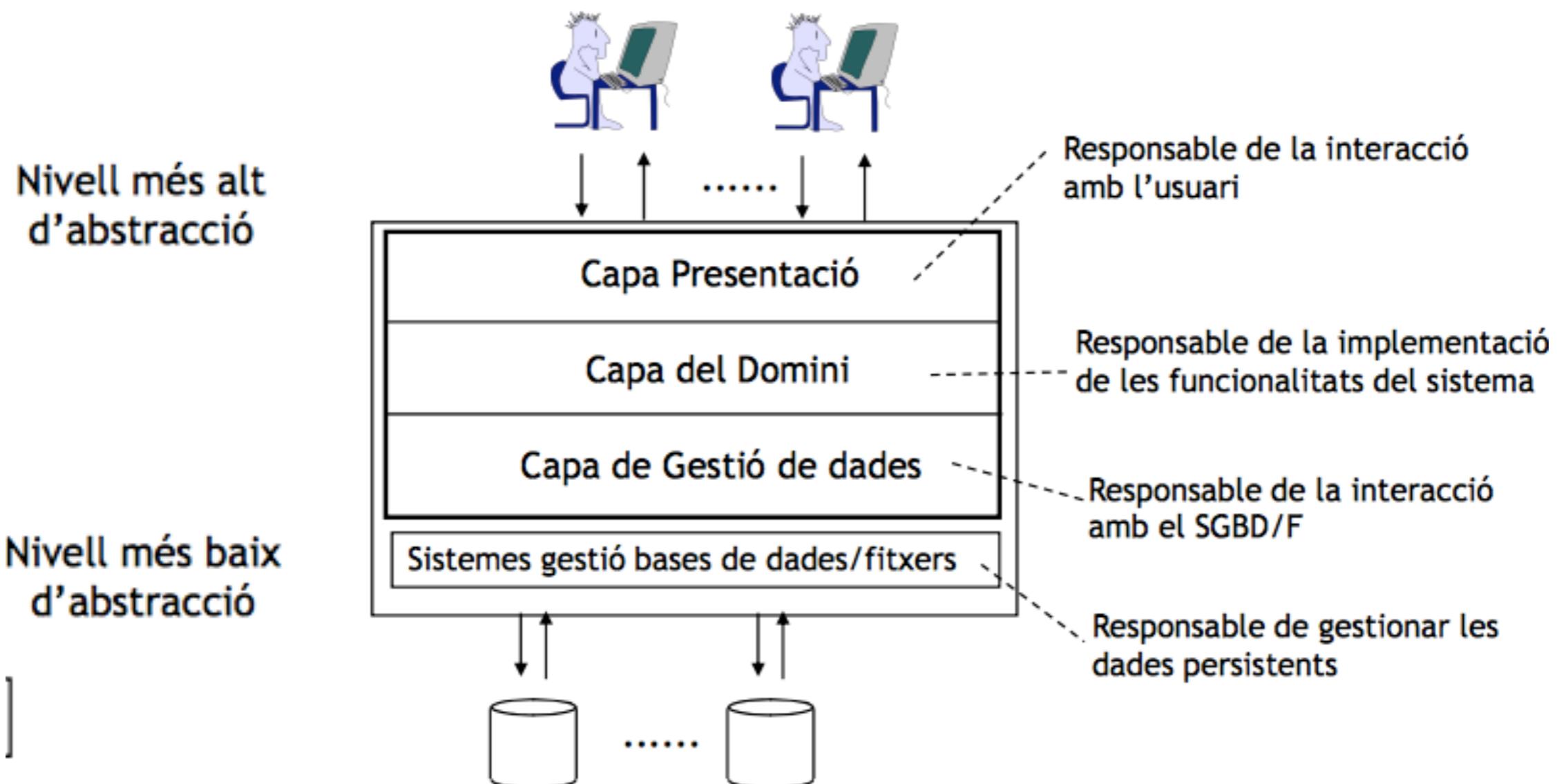
Solució

- **Aspecte estàtic:** estructura per capes separades de diferents responsabilitats (Les capes altes són més específiques i les baixes més tècniques i generals)
- **Aspecte dinàmic:** col·laboració i acoblament de capes altes a baixes

3.2. Patrons arquitectònics

Patró per capes:

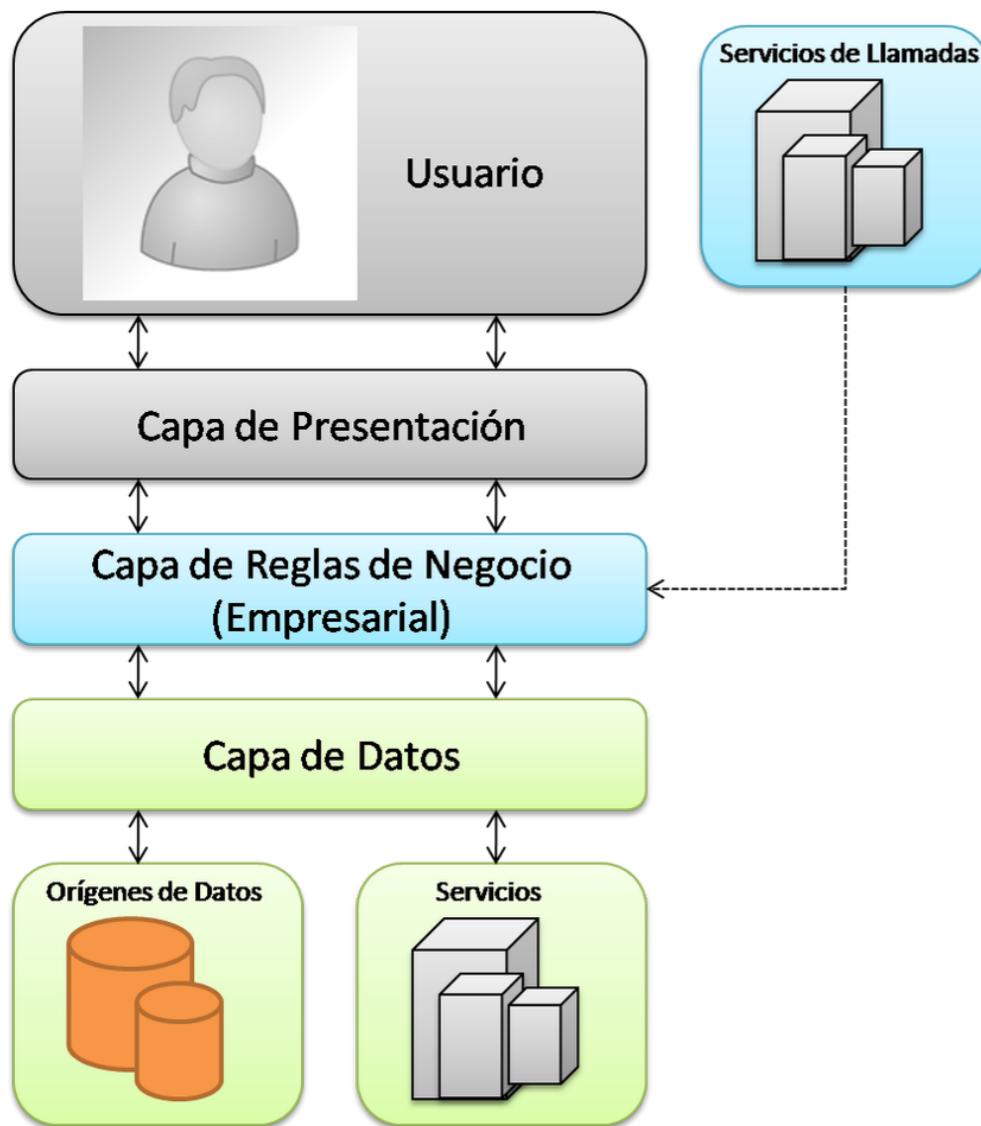
Un sistema gran que requereix ser descomposat en grups de tasques (components), tals que cada grup de tasques està a un nivell determinat d'abstracció (o **capa**).



3.2. Patrons arquitectònics

Patró per capes:

Un sistema gran que requereix ser descomposat en grups de tasques (components), tals que cada grup de tasques està a un nivell determinat d'abstracció (o **capa**).



Són només conceptuais i no tenen per què correspondre a l'estruccura de la implementació

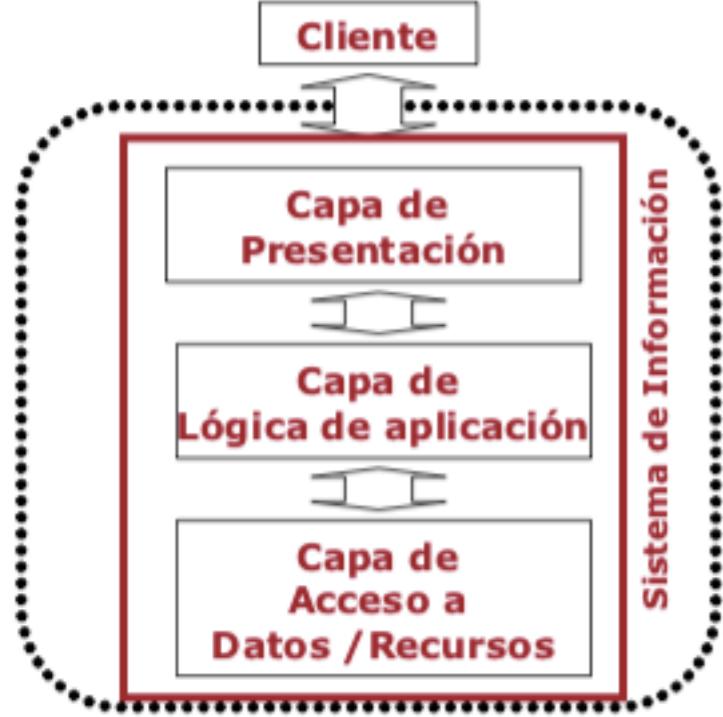
En el moment d'implementar-se, les capes poden quedar-se en un o més nivells (**tiers**)

Les més habituals:

1-tier, 2-tier, 3-nivells o N-nivells

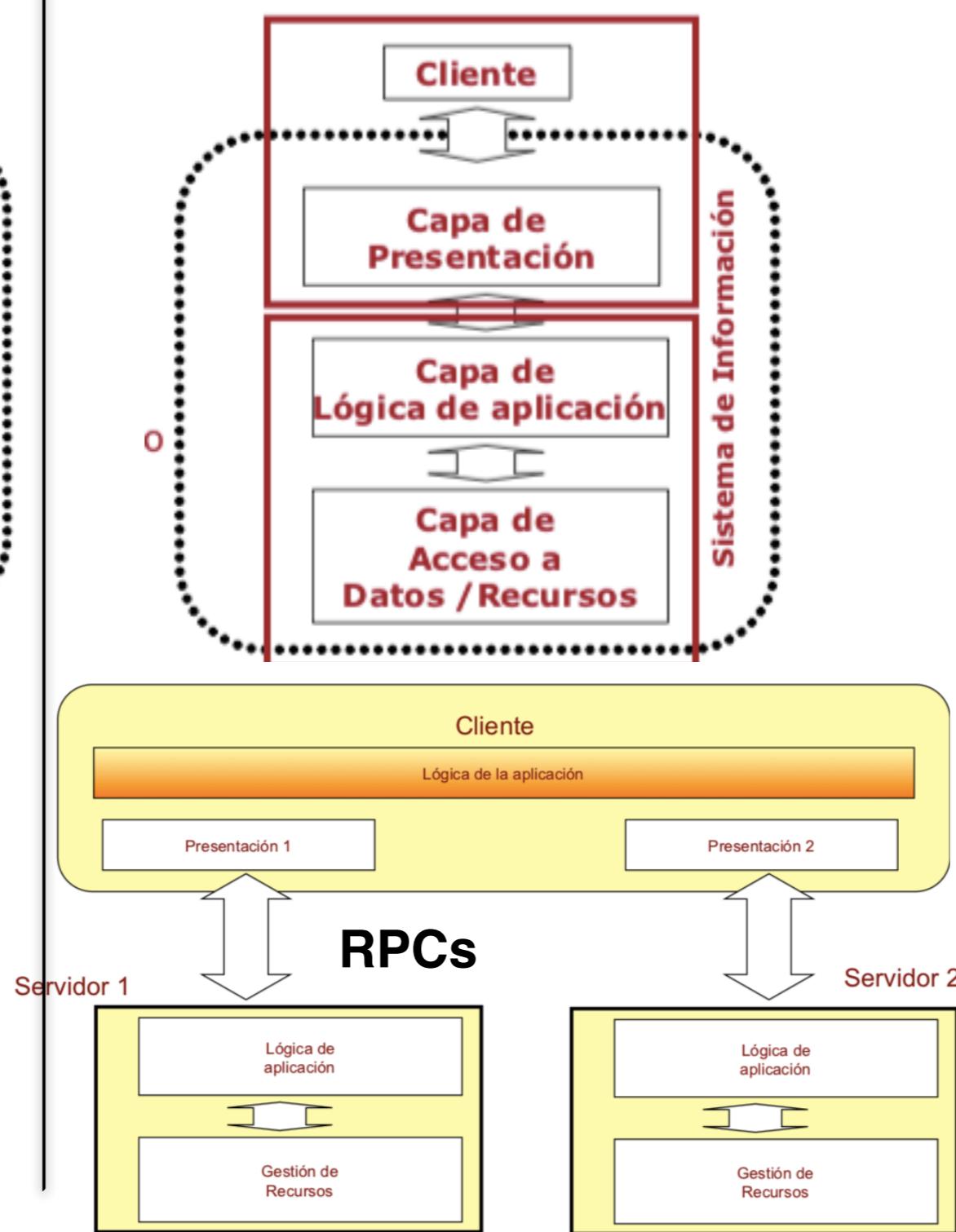
3.2. Patrons arquitectònics

1-tier

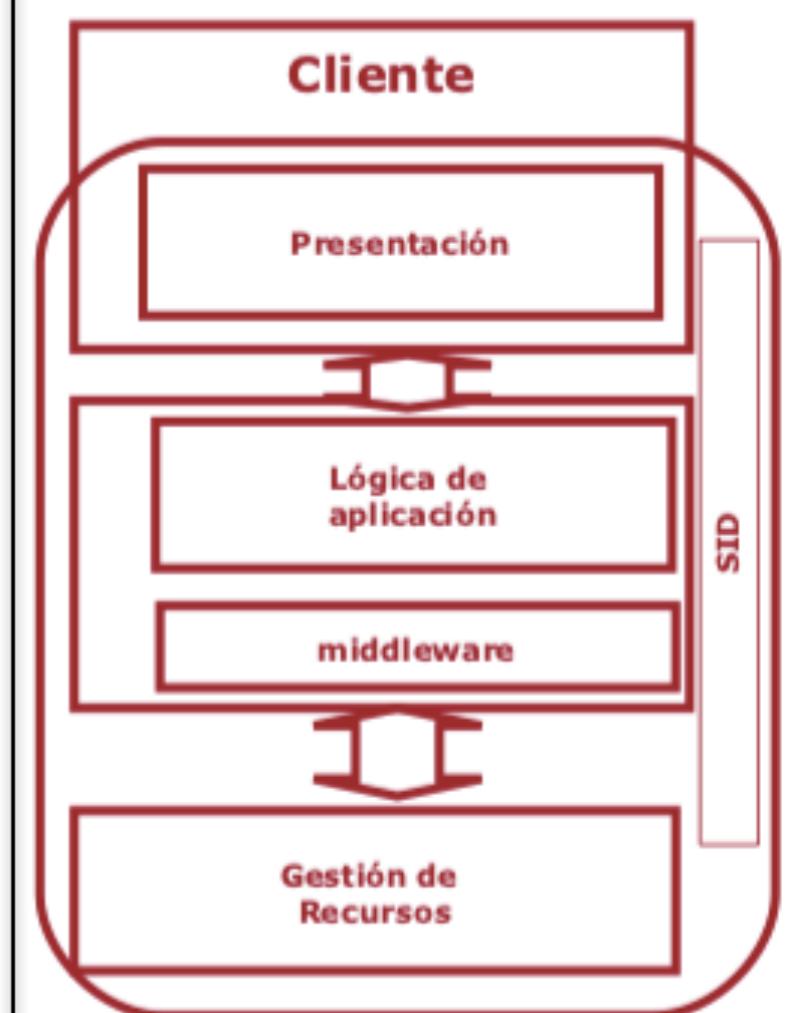


Sistemes monolítics

2-tier



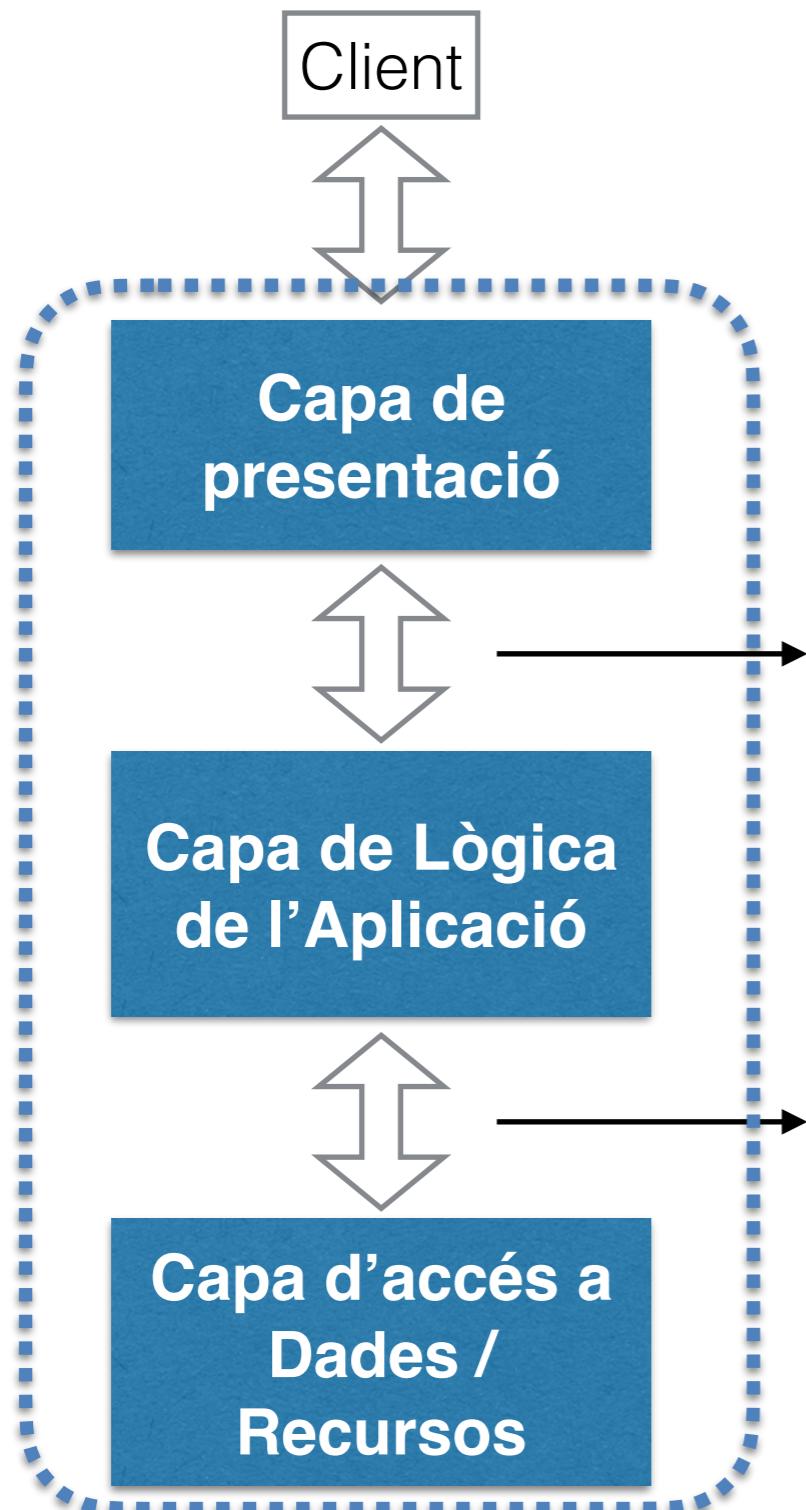
3-tier



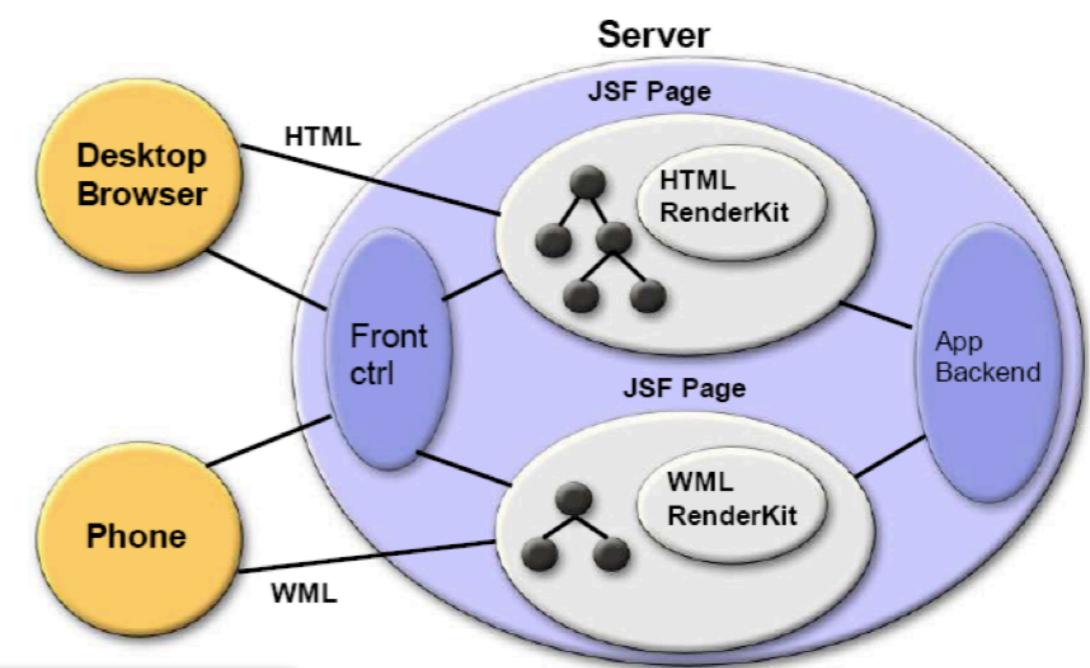
DCOM

3.2. Patrons arquitectònics

Patró per capes:



Patró Model-Vista-
Controlador



Patró Data
Access Object



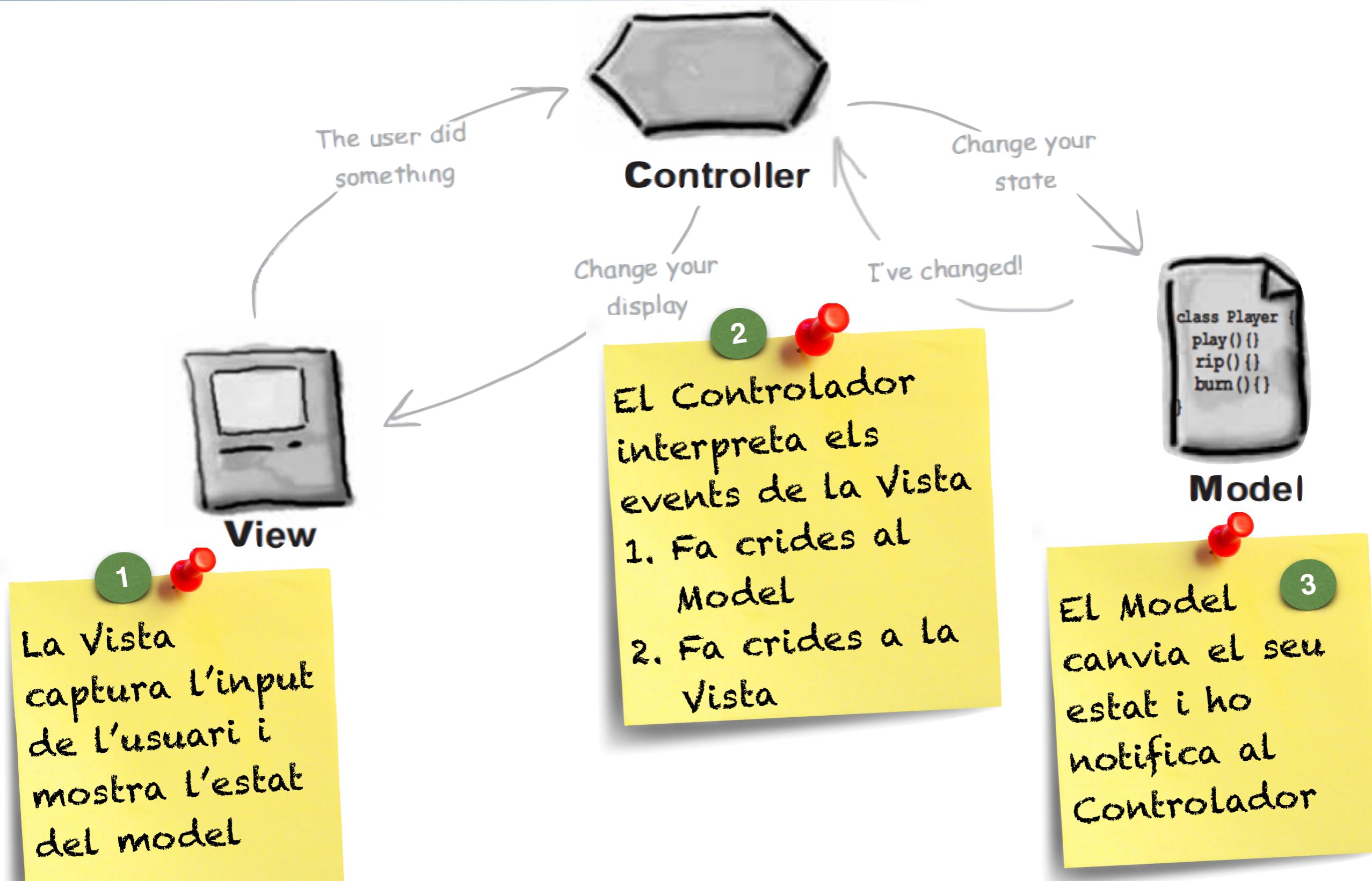
HIBERNATE



JDBC Templates

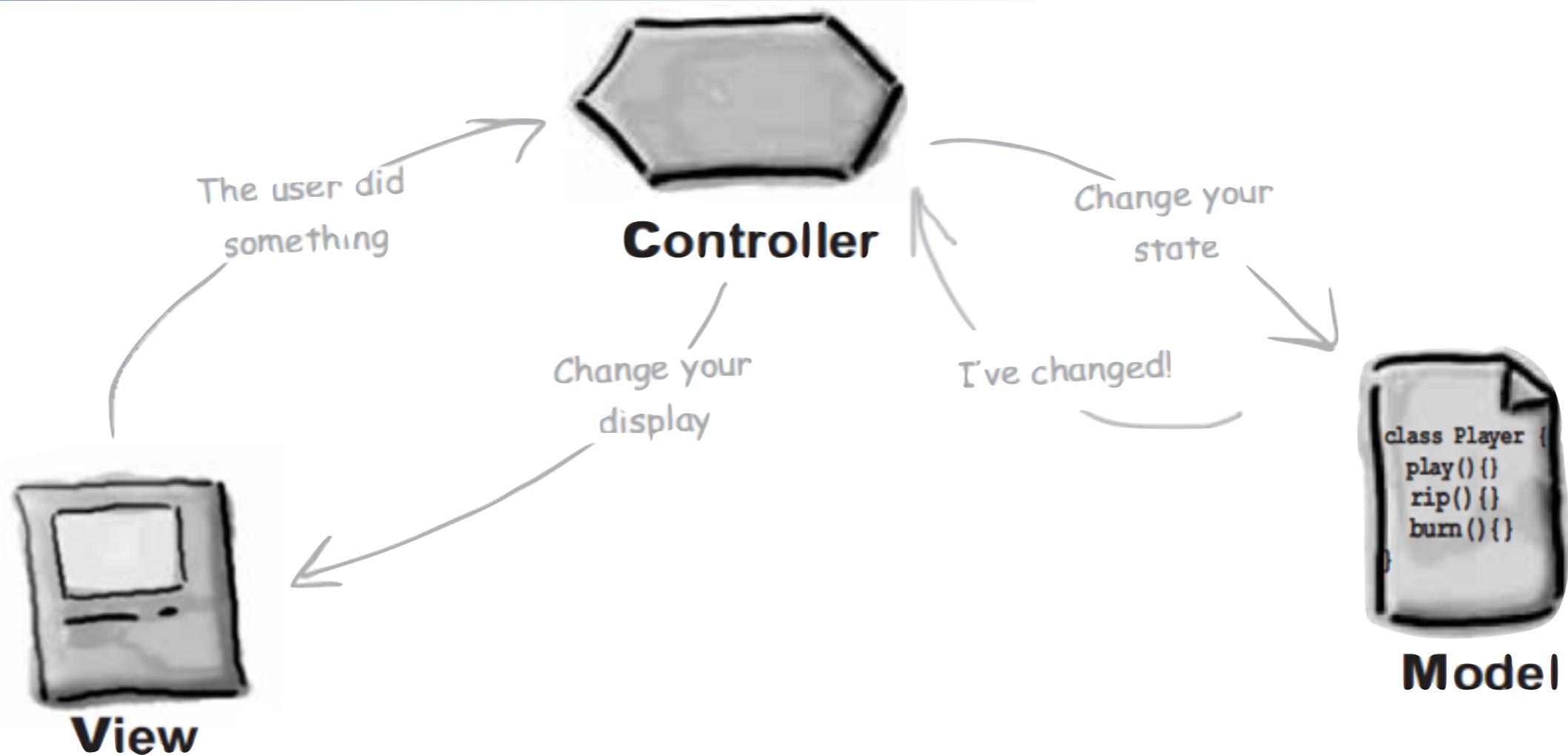
Model-Vista-Controlador

Aproximació senzilla



Model-Vista-Controlador

Aproximació senzilla



Problema:

- Quan hi han actualitzacions en el model (model dinàmic) que no estan fetes des de la vista, com pot el model notificar canvis a la Vista?
- El Controlador centralitza tota la comunicació produïnt retards a vegades innecessaris.

3.2. Models arquitectònics

Model - Vista - Controlador

VISTA:

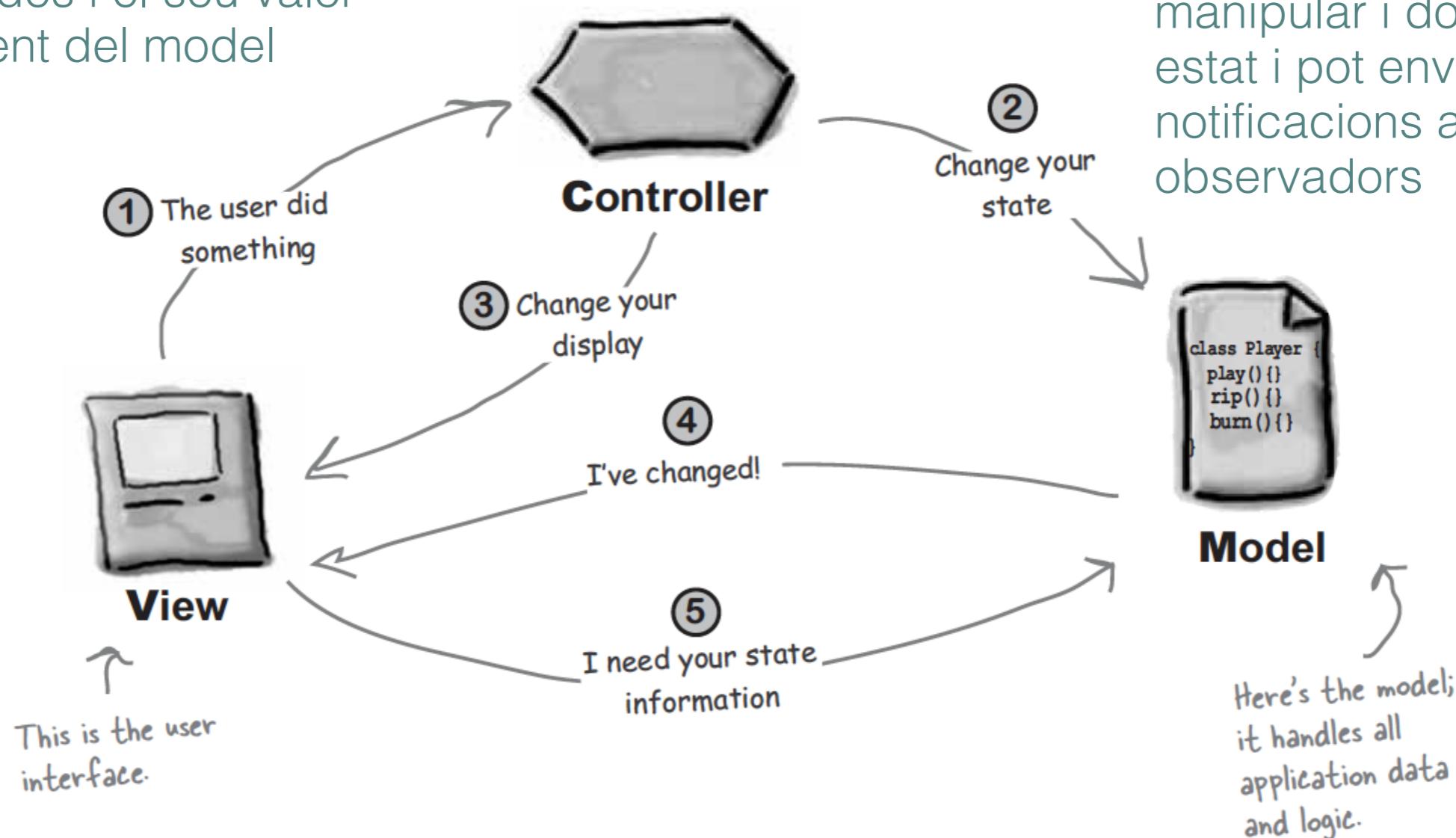
Dóna la presentació del model. La vista normalment mostra l'estat de les dades i el seu valor directament del model

CONTROLADOR:

Agafa l'entrada de l'usuari i li dóna el què significa al model

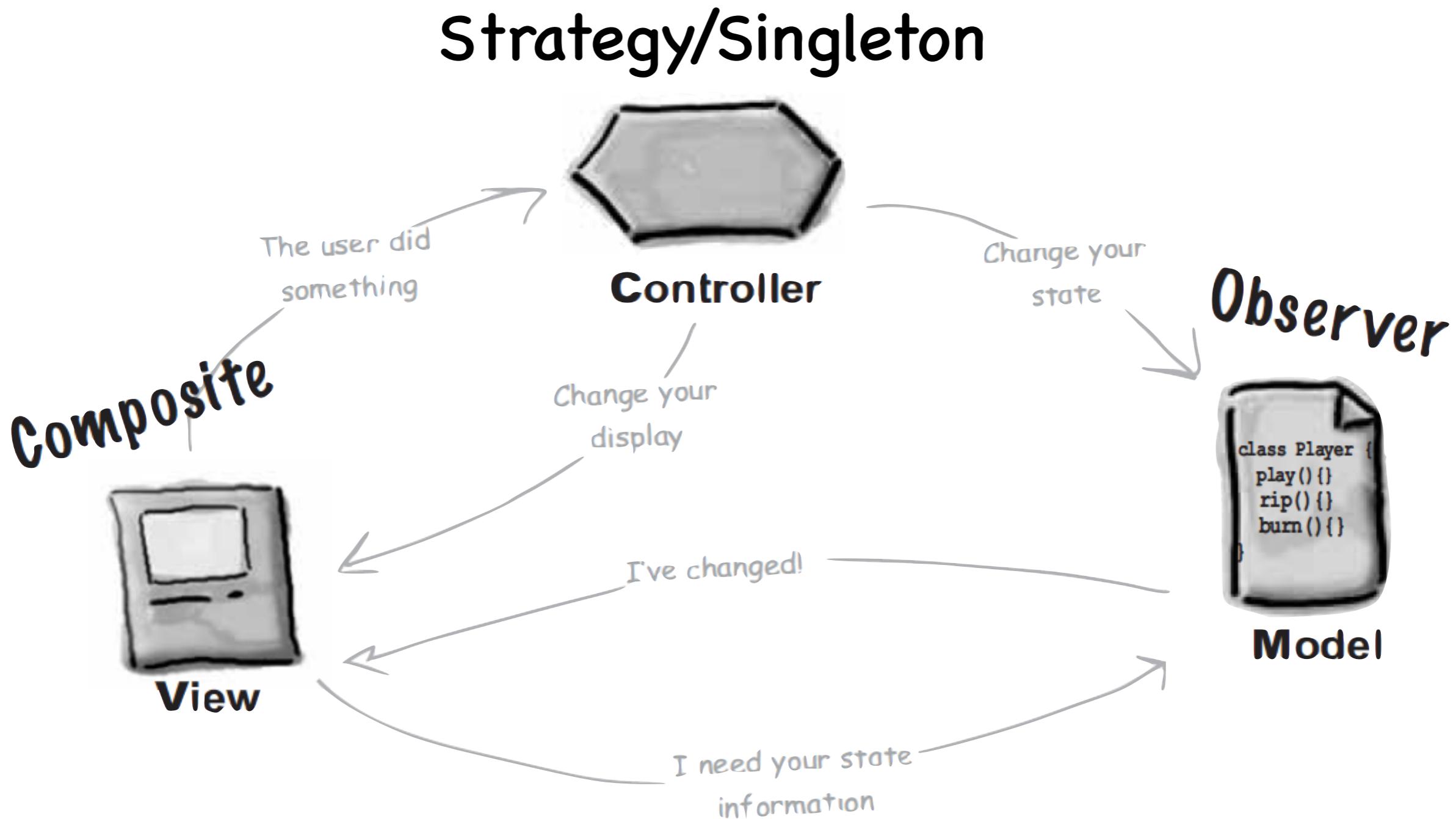
MODEL:

El model guarda totes les dates, l'estat i la lògica de l'aplicació. Dóna una interfície per manipular i donar el seu estat i pot enviar notificacions als observadors



Model-Vista-Controlador

Aproximació general - Principals patrons

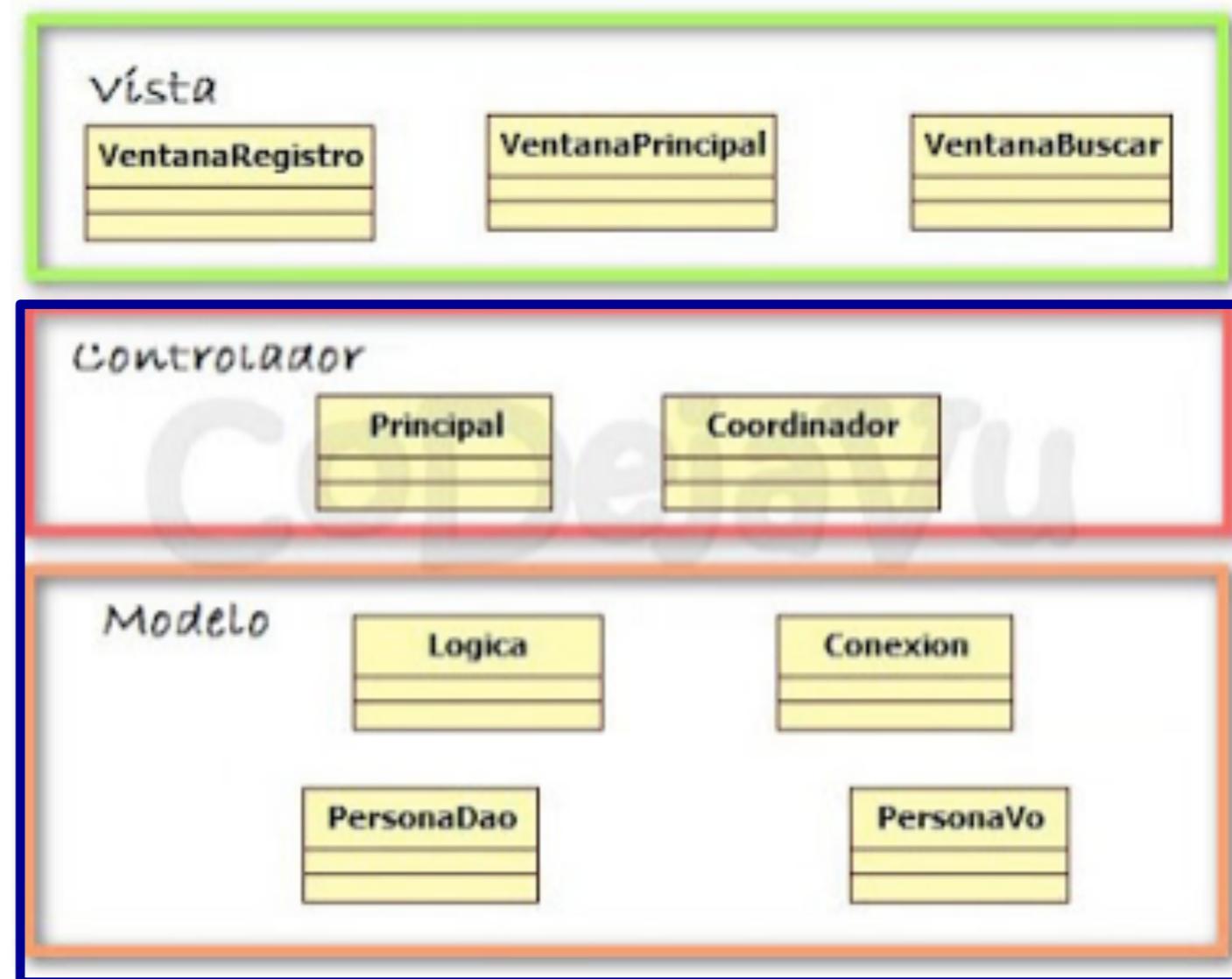


3.2. Patrons arquitectònics

Com encaixa el Model-Vista-Controlador en el model per capes?

Capa d'interfície o de presentació

Capa d'aplicació i domini



Exemple: <http://www.jc-mouse.net/java/3-en-raya-java-con-mvc-y-netbeans>

Model Vista Controlador (MVC)

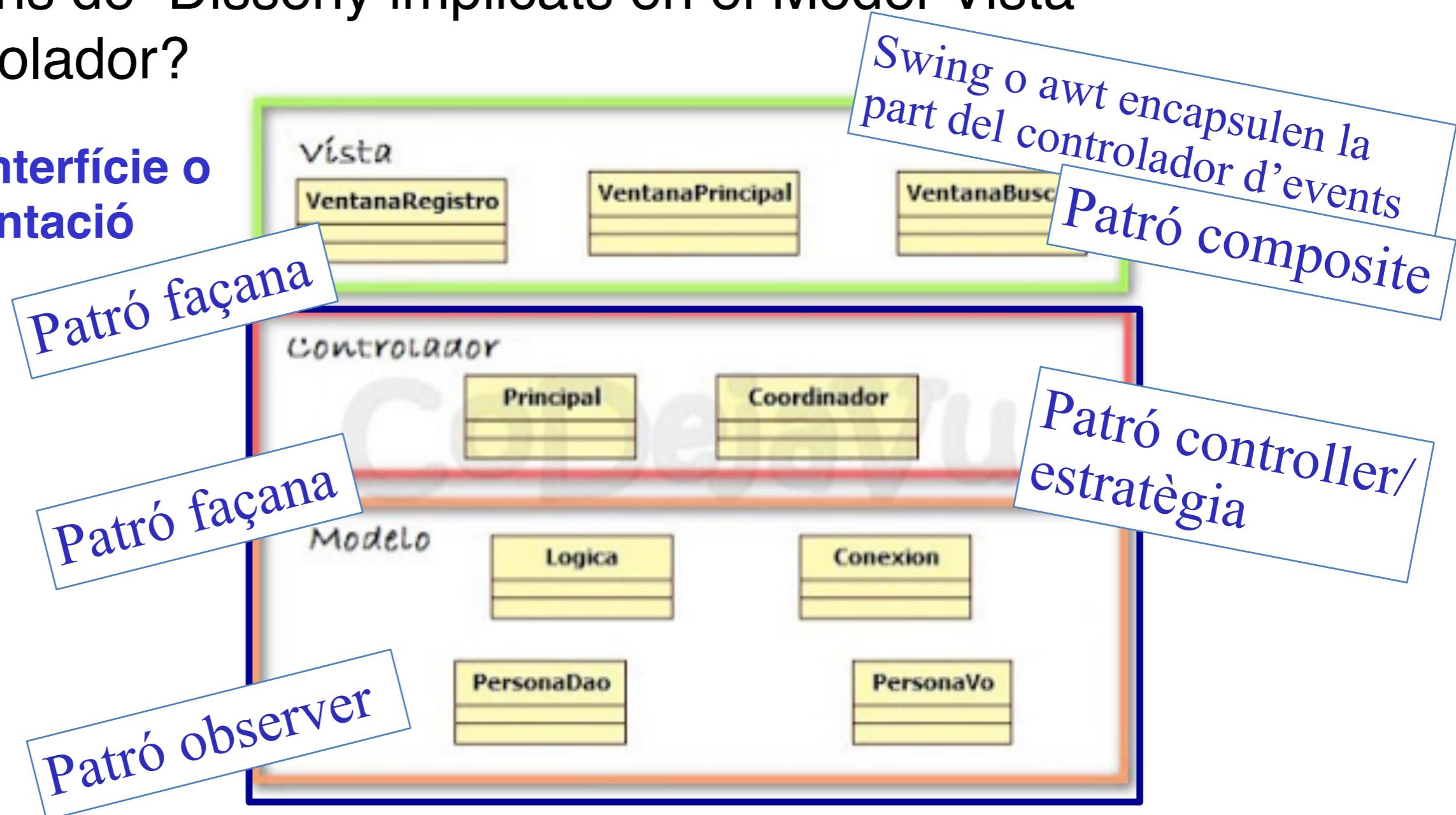
- Qualsevol arquitectura fa la distinció entre capa de presentació i capa de domini (o d'aplicació)
- Un sistema orientat a objectes típic es divideix en tres capes arquitectòniques:
 - **Interfície d'usuari (capa de presentació).** Interfície gràfica i finestres. (**Vista**) Deleguen les tasques a un **Controlador**.
 - **Lògica d'aplicació i objectes del domini.** El **Controlador** s'encarrega de processar les peticions de la capa de presentació i el **Model** s'encarrega dels objectes que representen conceptes del domini i tenen com a missió satisfer els requisits
 - **Serveis tècnics.** Objectes de propòsit general i subsistemes que proveeixen de suport tècnic

3.2. Patrons arquitectònics

Patrons de Disseny implicats en el Model-Vista-Controlador?

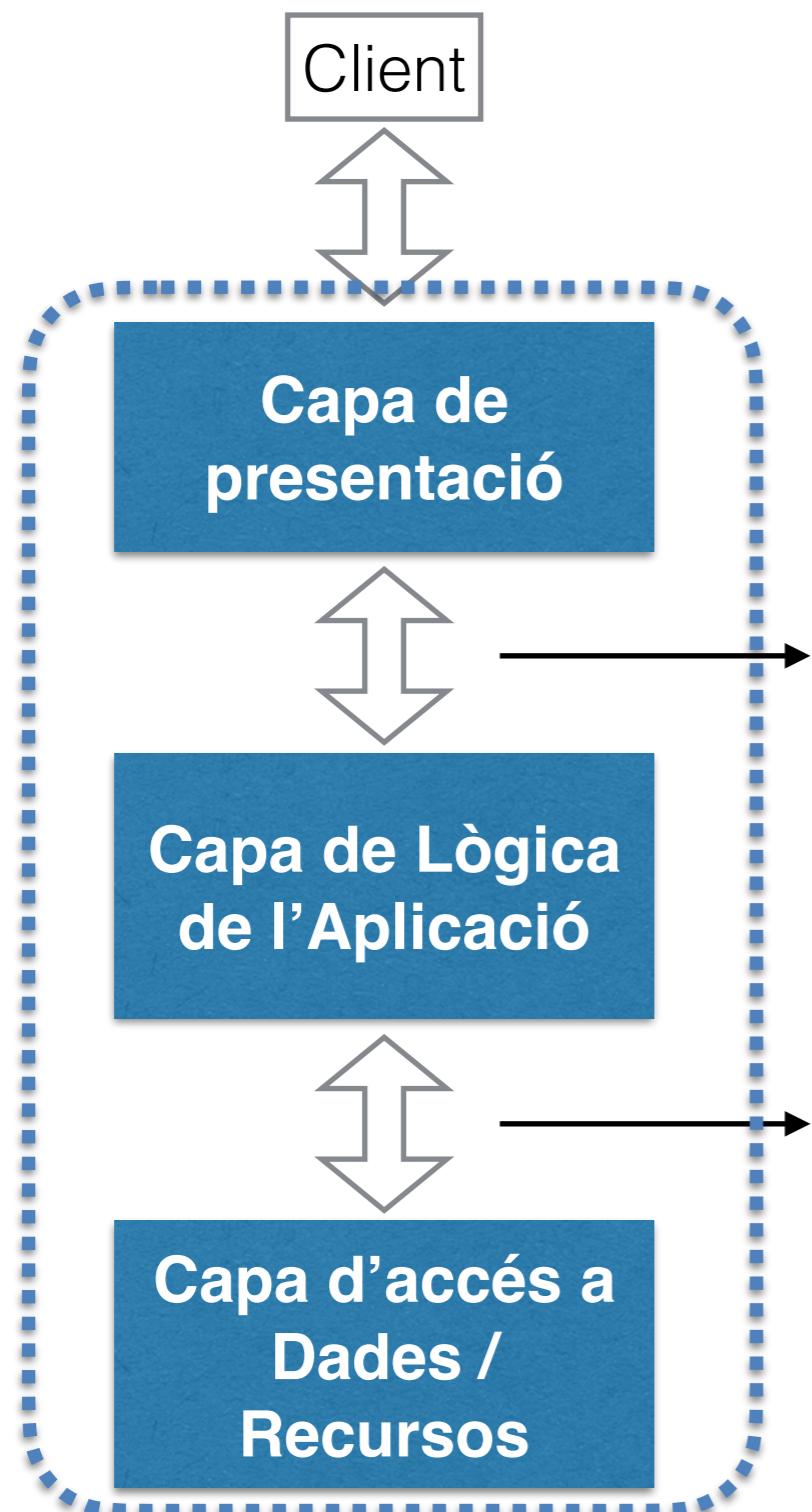
Capa d'interfície o
de presentació

Capa d'aplicació
i domini

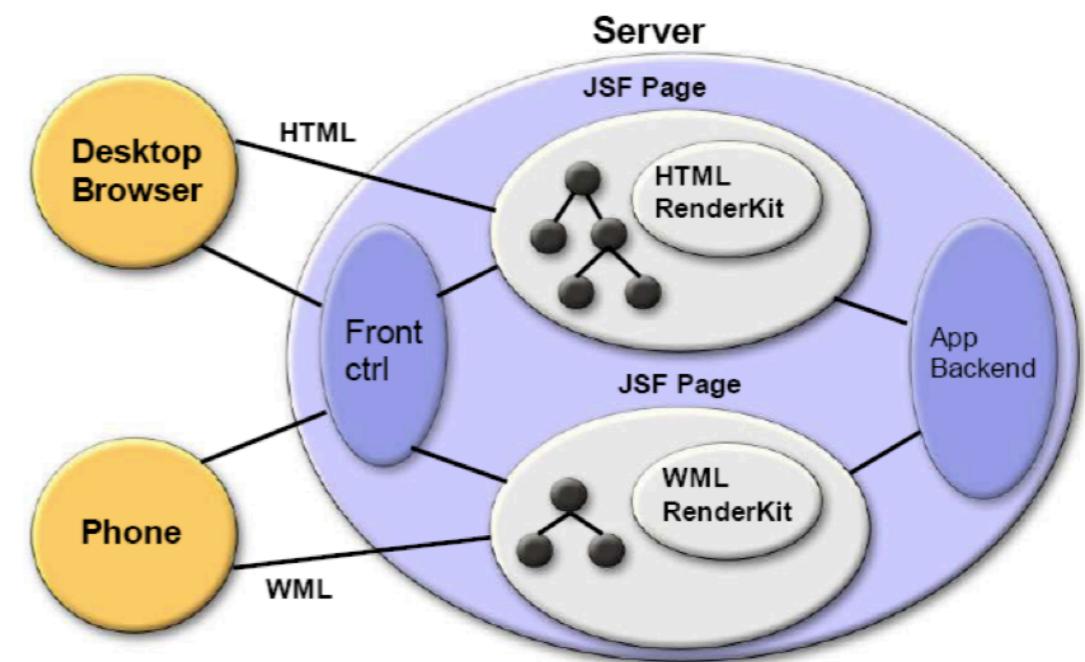


3.2. Patrons arquitectònics

Patró per capes:



Patró Model-Vista-Controlador



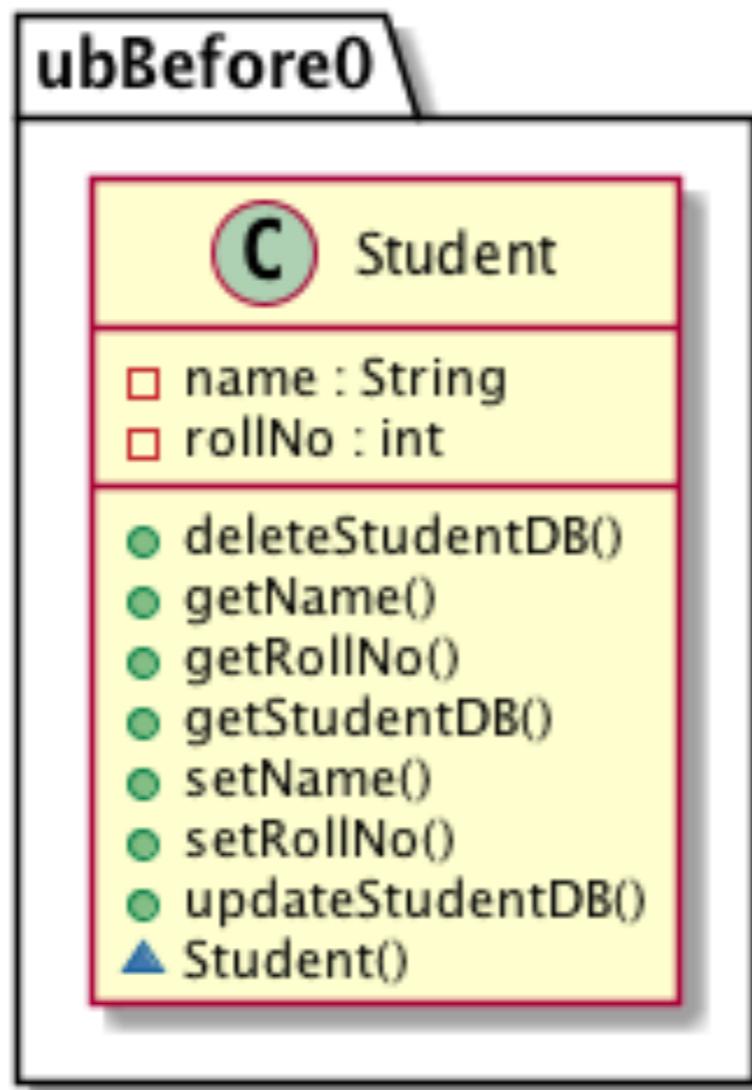
Patró Data Access Object



JDBC Templates

3.2. Patrons arquitectònics

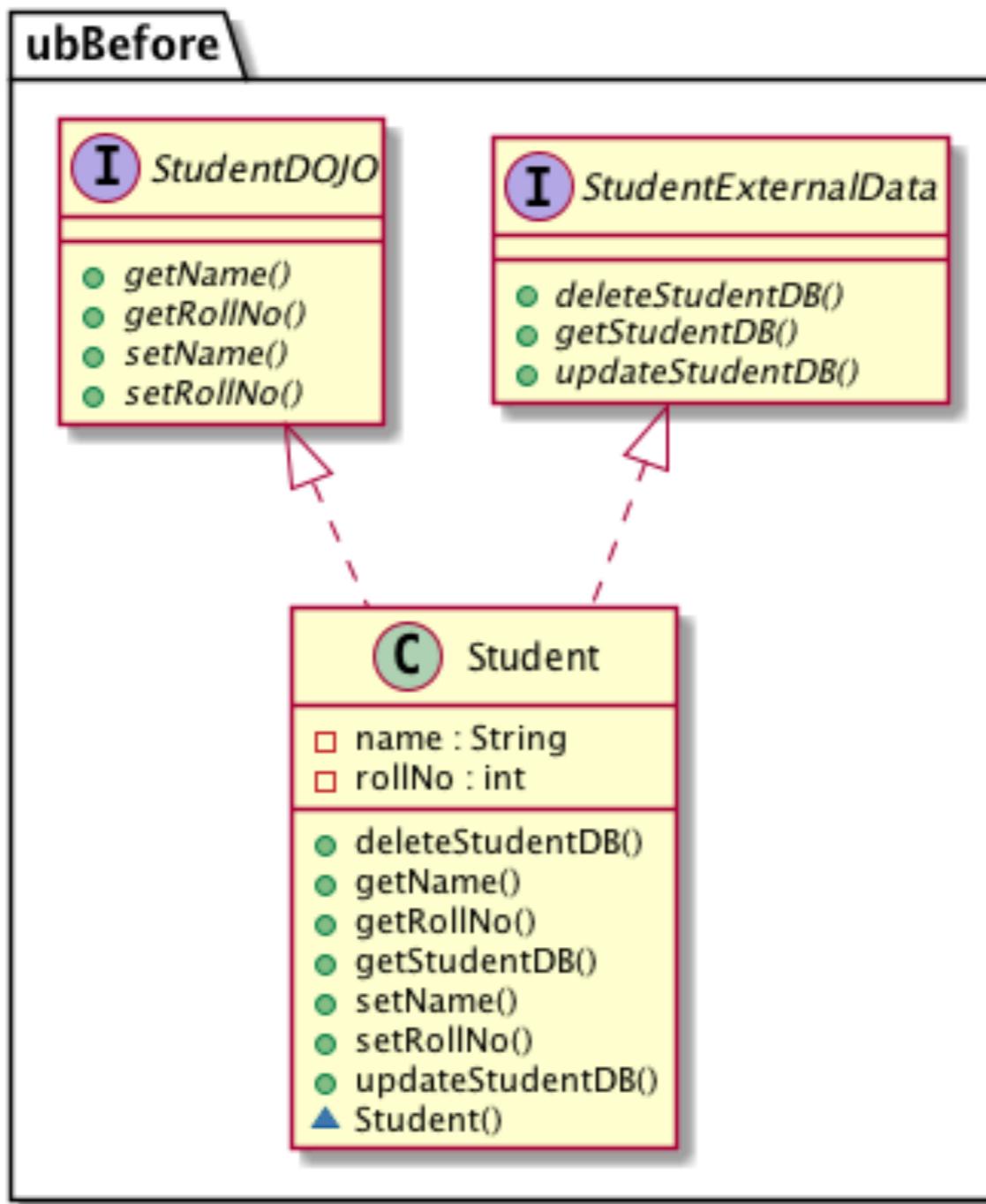
Comunicació entre la capa de lògica i la capa de persistència:



Cohesió?
Acoblament?

3.2. Patrons arquitectònics

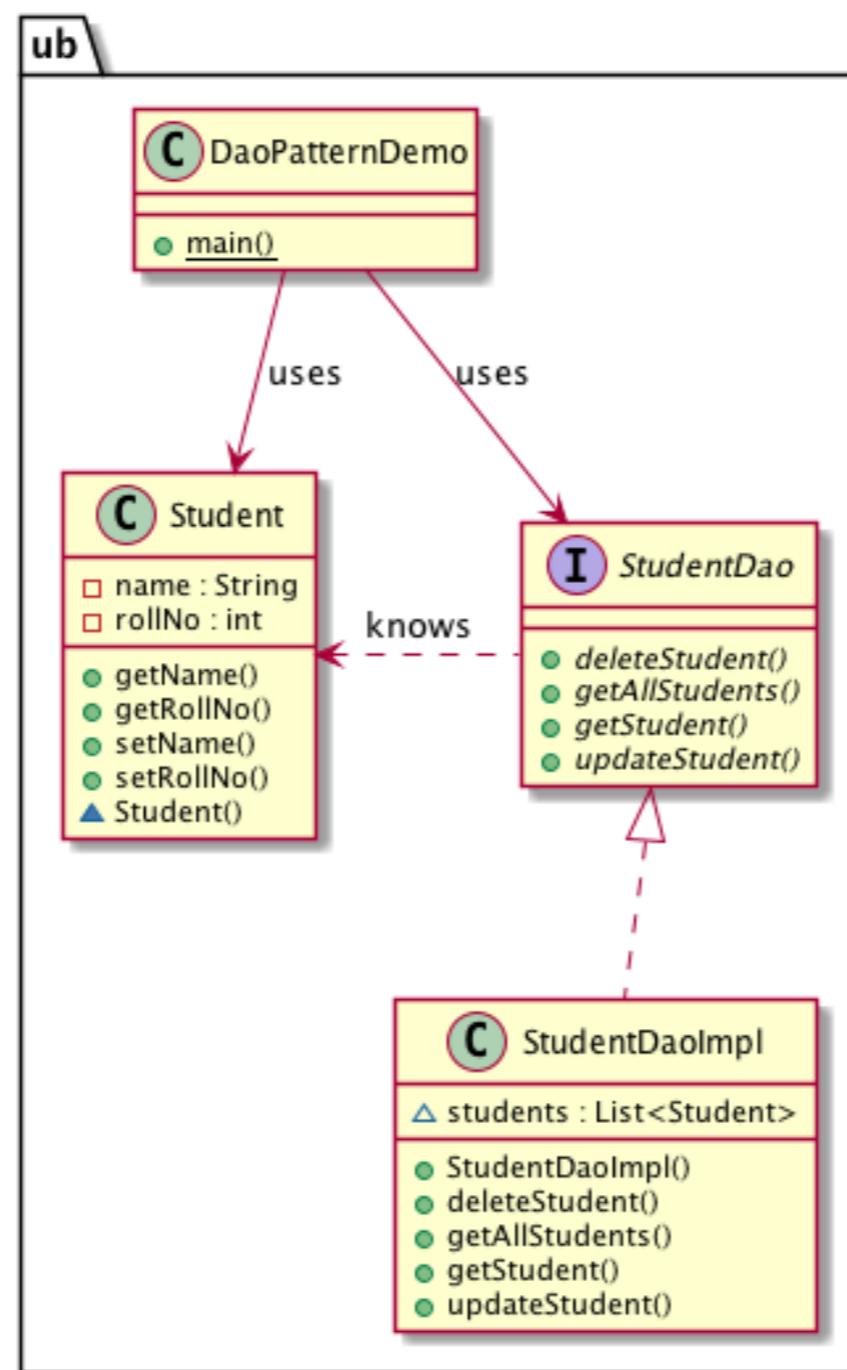
Comunicació entre la capa de lògica i la capa de persistència:



Cohesió?
Acoblament?

3.2. Patrons arquitectònics

Comunicació entre la capa de lògica i la capa de persistència:



3.2. Patrons arquitectònics

Nom del patró: **Patró Data Access Object (DAO)**

Problema

- Accés als components de baix nivell acoblat a la lògica de l'aplicació
- Poc flexible a canvis de components de baix nivell (rigidesa)
- Viscositat a nivell de disseny de l'aplicació

Solució

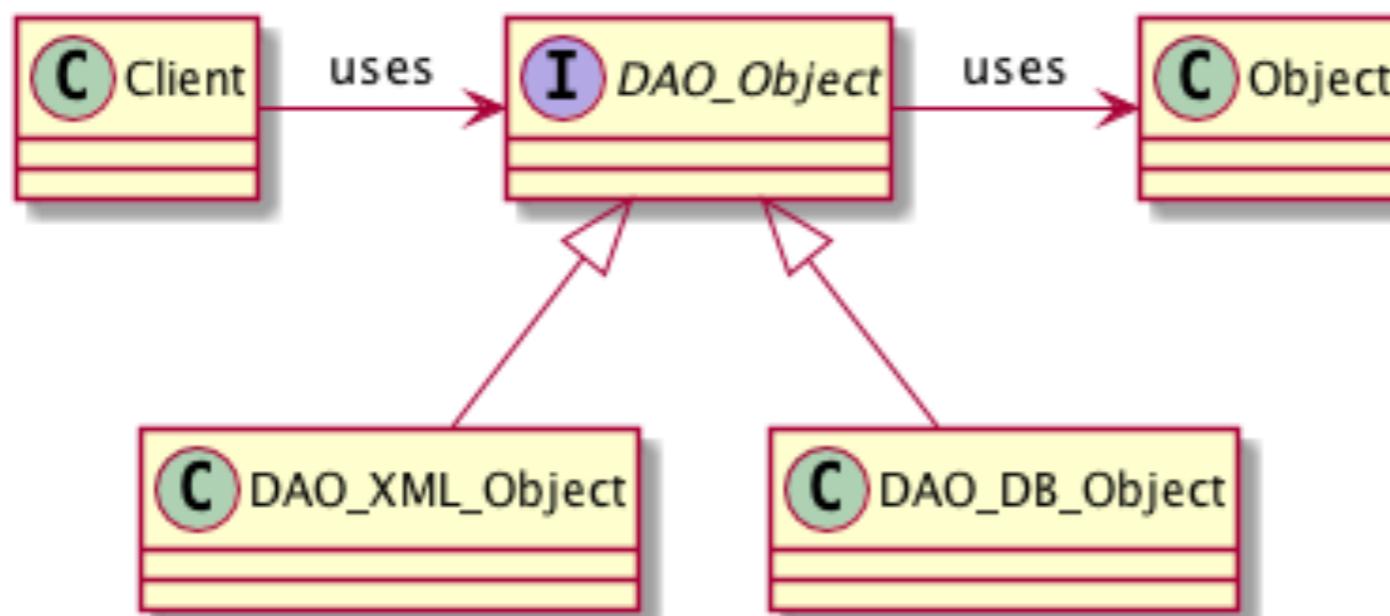
- **Aspecte estàtic:** Tot accés a Bases de Dades es fa mitjançant DAO. Encapsula les operacions CRUD dels objectes de l'aplicació
- **Aspecte dinàmic:** col·laboració i desacoblamet de la capa d'aplicació a la capa de persistència

3.2. Patrons arquitectònics

Patró Data Access Object: (DAO)

Utilitzat per separar l'accés a dades o recursos a baix nivell des de les funcionalitats. Els participants són:

- **Data Access Object Interface** - Defineix les operacions stàndard que s'han de fer en els objectes del model
- **Data Access Object classe d'implementació** - Realitza la implementació concreta sobre la base de dades /XML o la tecnologia concreta de persistència.
- **Object**: (POJO: Plain Old Java Object) - Objecte que conté les dades extretes de la classe DAO, i permet mètodes de get/set.



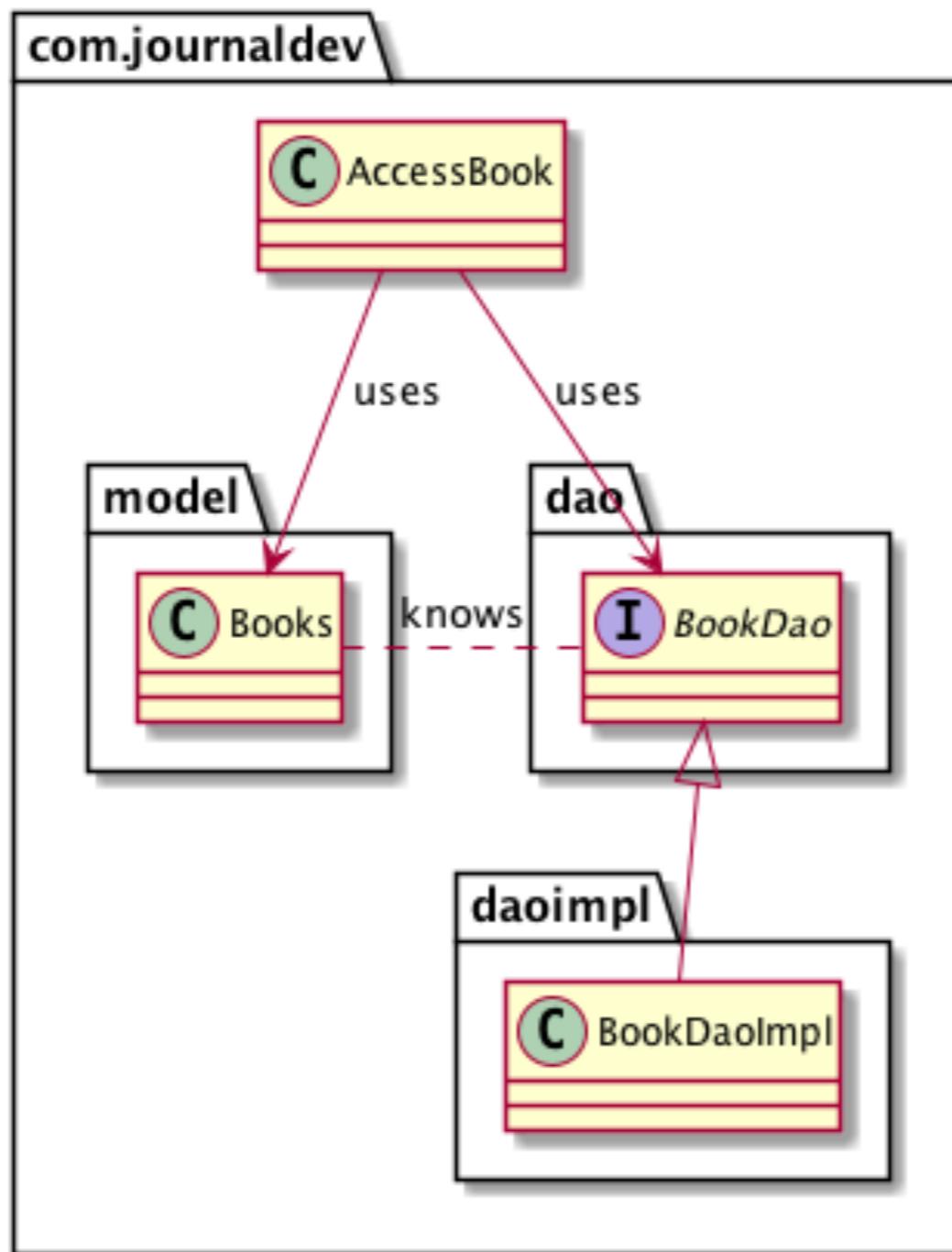
3.2. Patrons arquitectònics

Patró Data Access Object: (DAO) Interfície DAO

```
public interface GenericDao<T> {  
  
    /** Persist the newInstance object into database */  
    public void add(T newInstance);  
  
    /** Retrieve an object that was previously persisted to the database */  
    public T read();  
  
    public List<T> getAll();  
  
    /** Save changes made to a persistent object. */  
    public void update(T transientObject);  
  
    /** Remove an object from persistent storage in the database */  
    public void delete(T persistentObject);  
}
```

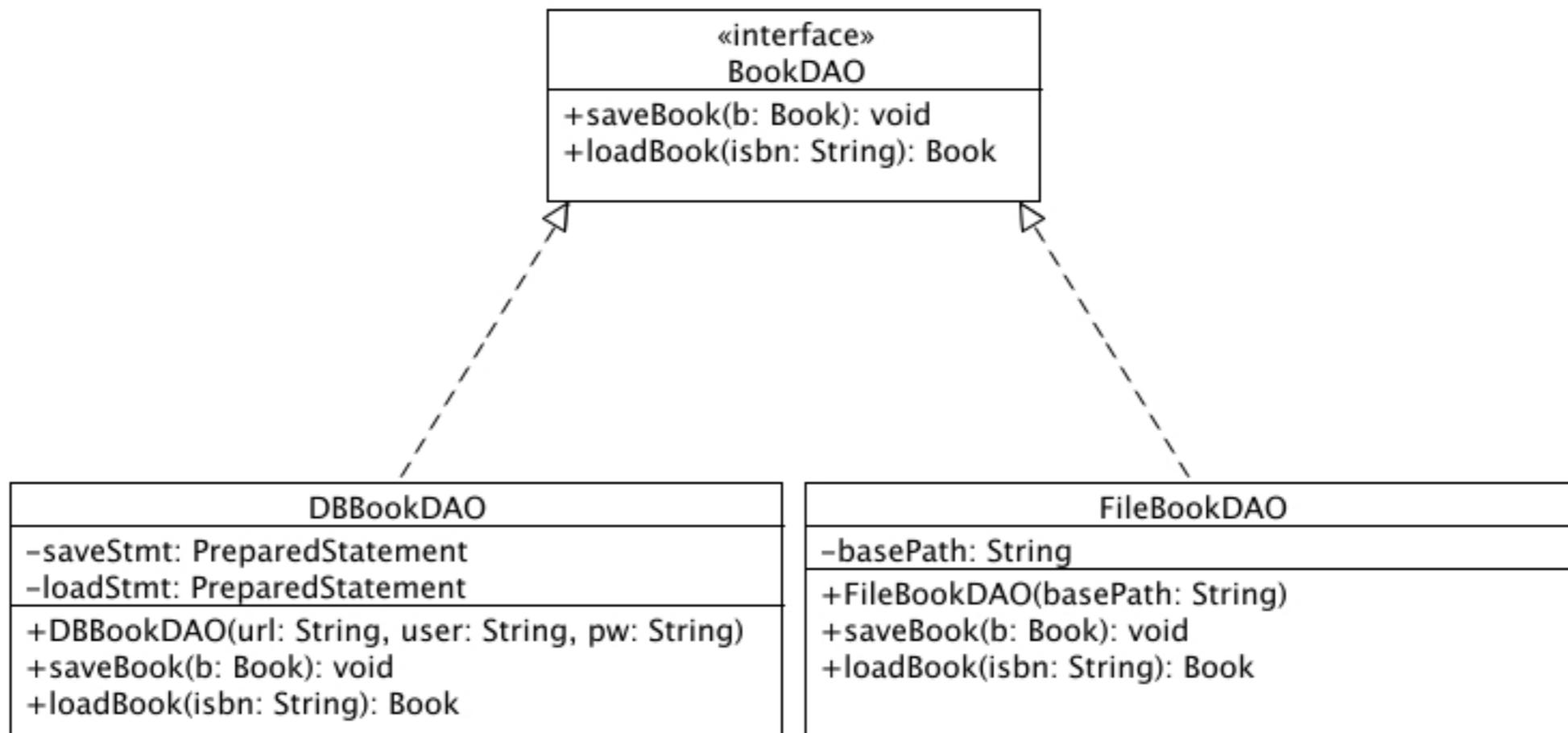
3.2. Patrons arquitectònics

Patró Data Access Object: (DAO) Exemples



3.2. Patrons arquitectònics

Patró Data Access Object: (DAO) Exemples



<http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/dao.html>

3.2. Patrons arquitectònics

Patró Data Access Object: (DAO) Exemples i variacions

```
public class DBBookDAO implements BookDAO {  
  
    private PreparedStatement saveStmt;  
    private PreparedStatement loadStmt;  
  
    public DBBookDAO(String url, String user, String pw) {  
        Connection con = DriverManager.getConnection(url, user, pw);  
        saveStmt = con.prepareStatement("INSERT INTO books(isbn, title, author) "  
            + "VALUES (?, ?, ?)");  
        loadStmt = con.prepareStatement("SELECT isbn, title, author FROM books "  
            + "WHERE isbn = ?");  
    }  
  
    public Book loadBook(String isbn) {  
        Book b = new Book();  
        loadStmt.setString(1, isbn);  
        ResultSet result = loadStmt.executeQuery();  
        if (!result.next()) return null;  
  
        b.setIsbn(result.getString("isbn"));  
        b.setTitle(result.getString("title"));  
        b.setAuthor(result.getString("author"));  
        return b;  
    }  
  
    public void saveBook(Book b) {  
        saveStmt.setString(1, b.getIsbn());  
        saveStmt.setString(2, b.getTitle());  
        saveStmt.setString(3, b.getAuthor());  
        saveStmt.executeUpdate();  
    }  
}
```

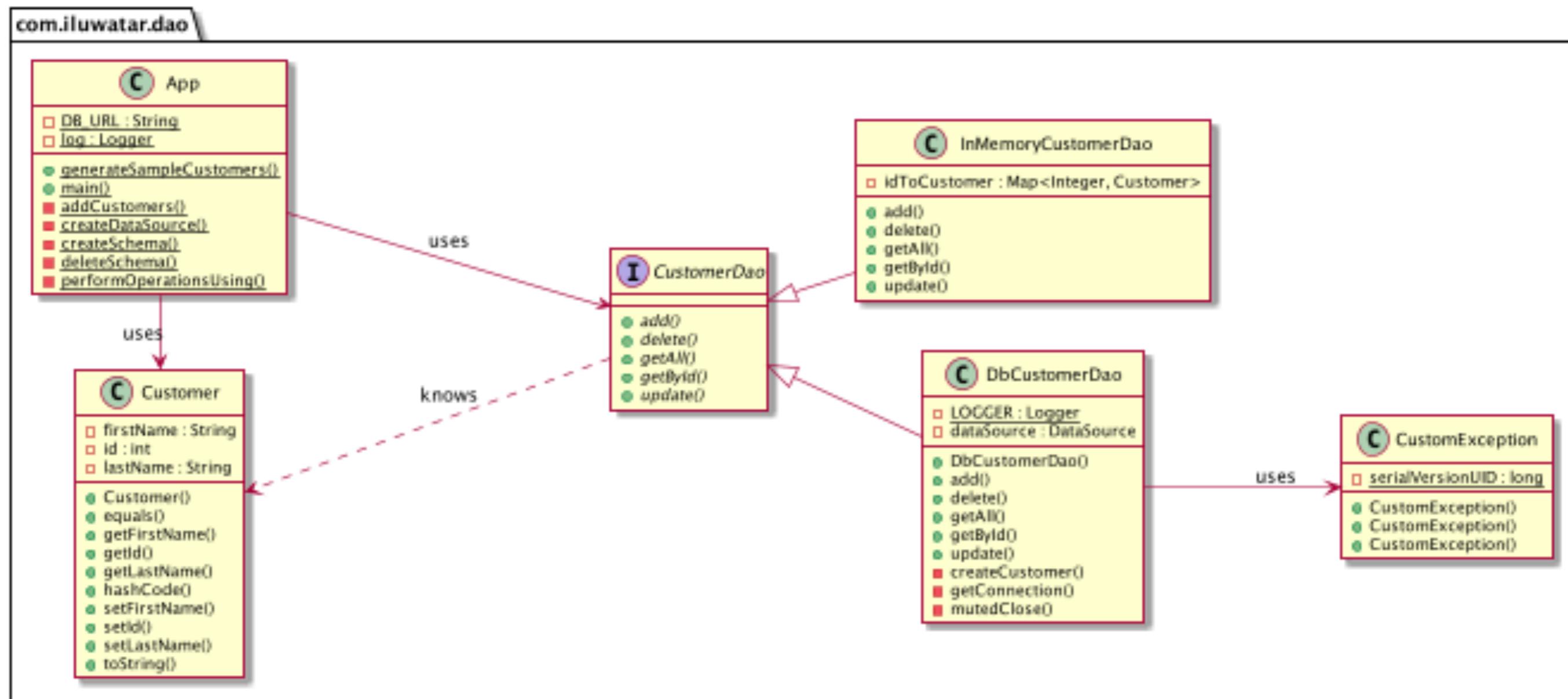
3.2. Patrons arquitectònics

Patró Data Access Object: (DAO) Exemples i variacions

```
public class FileBookDAO implements BookDAO {  
  
    private String basePath;  
  
    public FileBookDAO(String basePath) {  
        this.basePath = basePath;  
    }  
  
    public Book loadBook(String isbn) {  
        FileReader fr = new FileReader(basePath + isbn);  
        BufferedReader br = new BufferedReader(fr);  
        Book b = new Book();  
        String rISBN = br.readLine();  
        String rTitle = br.readLine();  
        String rAuthor = br.readLine();  
  
        if (rISBN.startsWith("ISBN: ")) {  
            b.setISBN(rISBN.substring("ISBN: ".length()));  
        } else {  
            return null;  
        }  
        if (rTitle.startsWith("TITLE: ")) {  
            b.setTitle(rTitle.substring("TITLE: ".length()));  
        } else {  
            return null;  
        }  
        if (rAuthor.startsWith("AUTHOR: ")) {  
            b.setAuthor(rAuthor.substring("AUTHOR: ".length()));  
        } else {  
            return null;  
        }  
        return b;  
    }  
}
```

3.2. Patrons arquitectònics

Patró Data Access Object: (DAO) Exemples



3.2. Patrons arquitectònics

Patró per capes:

