13.4 Che cos'è un'architettura software

Si è accennato alle architetture logiche e alle architetture di rilascio, e ora è il momento di dare una definizione di **architettura software**. Una è la seguente:

> Un'architettura è l'insieme delle decisioni significative sull'organizzazione di un sistema software, la scelta degli elementi strutturali da cui è composto il sistema e delle relative interfacce, insieme al loro comportamento specificato dalle collaborazioni tra questi elementi, la composizione di questi elementi strutturali e comportamentali in sottosistemi via via più ampi, e lo stile architetturale che guida questa organizzazione (questi elementi e le loro interfacce, le loro collaborazioni e la loro composizione). [BRJ99]

Indipendentemente dalla definizione (e ce ne sono diverse) il tema comune in tutte le definizioni di architettura software è che essa ha a che fare con la larga scala: le cosiddette Grandi Idee nelle motivazioni, vincoli, organizzazione, pattern, responsabilità e connessioni di un sistema (o un sistema di sistemi).

La Figura 13.2 mostra un'architettura logica (parziale) a strati, disegnata con la notazione di un **diagramma dei package** di UML.

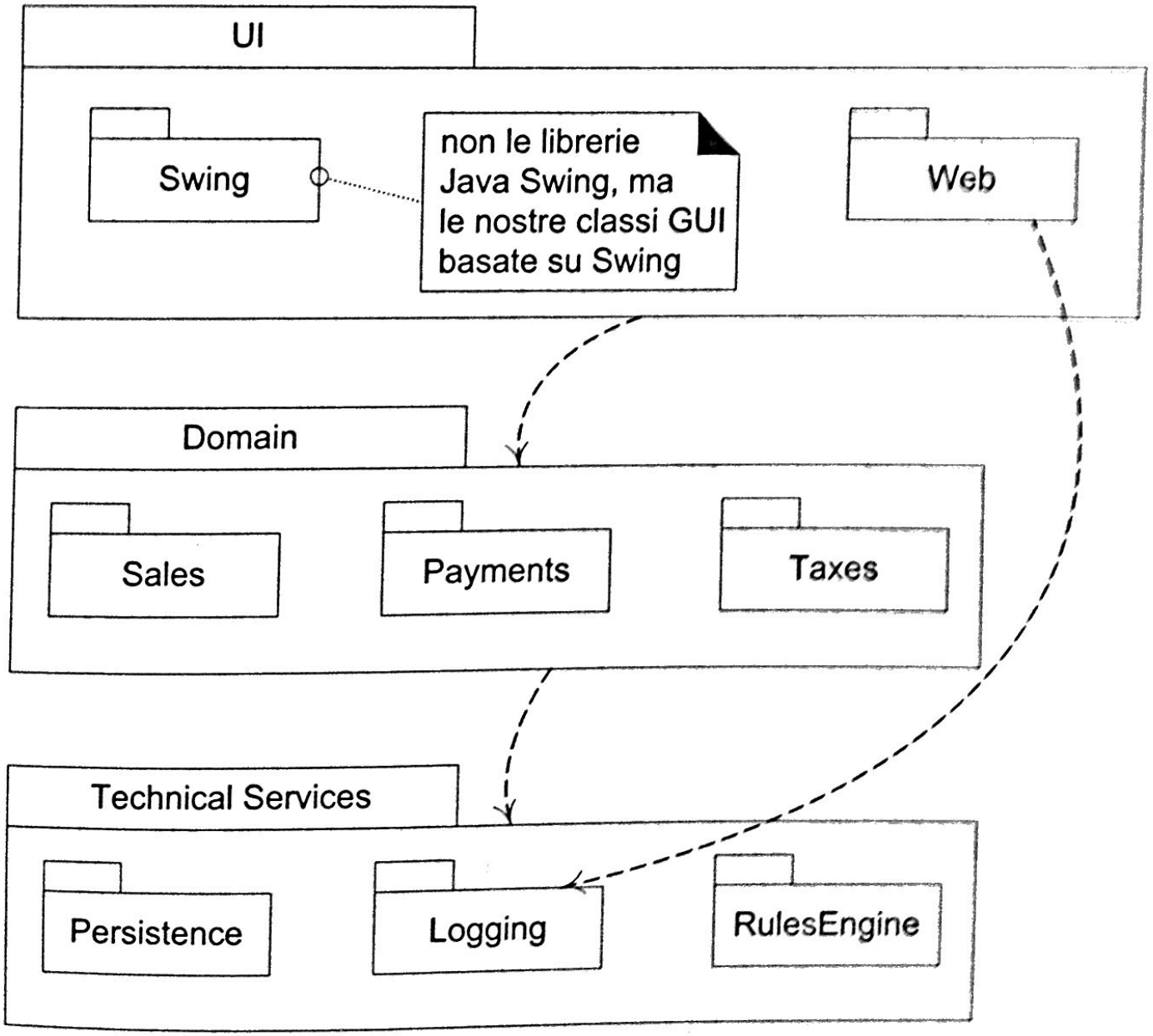

Figura 13.2 Strati mostrati con la notazione di un diagramma dei package di UML.

13.2 Che cosa sono l'architettura logica e gli strati

L'**architettura logica** è l'organizzazione su larga scala delle classi software in package (o namespace), sottosistemi e strati. È chiamata architettura *logica* poiché non ci sono decisioni da prendere su come questi elementi siano distribuiti sui processi o sui diversi computer fisici di una rete (queste ultime decisioni fanno parte dell'**architettura di deployment**).

Uno **strato** è un gruppo a grana molto grossa di classi, package o sottosistemi, che ha delle responsabilità coese rispetto a un aspetto importante del sistema. Inoltre gli strati sono organizzati in modo tale che strati "più alti" (come lo strato dell'interfaccia utente) ricorrano ai servizi degli strati "più bassi", mentre normalmente non avviene il contrario.

Gli strati di un sistema orientato agli oggetti comprendono normalmente i seguenti.

- **User Interface (Interfaccia utente).**
- **Application Logic e Domain Objects (Logica applicativa e Oggetti di dominio).** Oggetti software che rappresentano concetti del dominio (per esempio, una classe software *Sale*) che soddisfano i requisiti dell'applicazione, come calcolare il totale di una vendita.
- **Technical Services (Servizi tecnici).** Oggetti e sottosistemi d'uso generale che forniscono servizi tecnici di supporto, come l'interfacciamento con una base di dati o il logging degli errori. Questi servizi sono solitamente indipendenti dall'applicazione e riusabili in diversi sistemi.

In un'**architettura a strati stretta**, uno strato può solo richiamare i servizi dello strato immediatamente sottostante. Questa struttura è comune nei protocolli di rete organizzati a pila, ma non nei sistemi informatici, che normalmente utilizzano un'**architettura a strati rilassata**, in cui uno strato più alto può richiamare i servizi di strati più bassi di diversi livelli. Per esempio, lo strato UI può fare chiamate allo strato della logica applicativa direttamente sottostante, ma anche agli elementi di uno strato di servizi tecnici più basso, per il logging e così via.
Un'architettura logica non deve necessariamente essere organizzata a strati, ma molto spesso lo è; per questo motivo viene introdotta in questo capitolo.

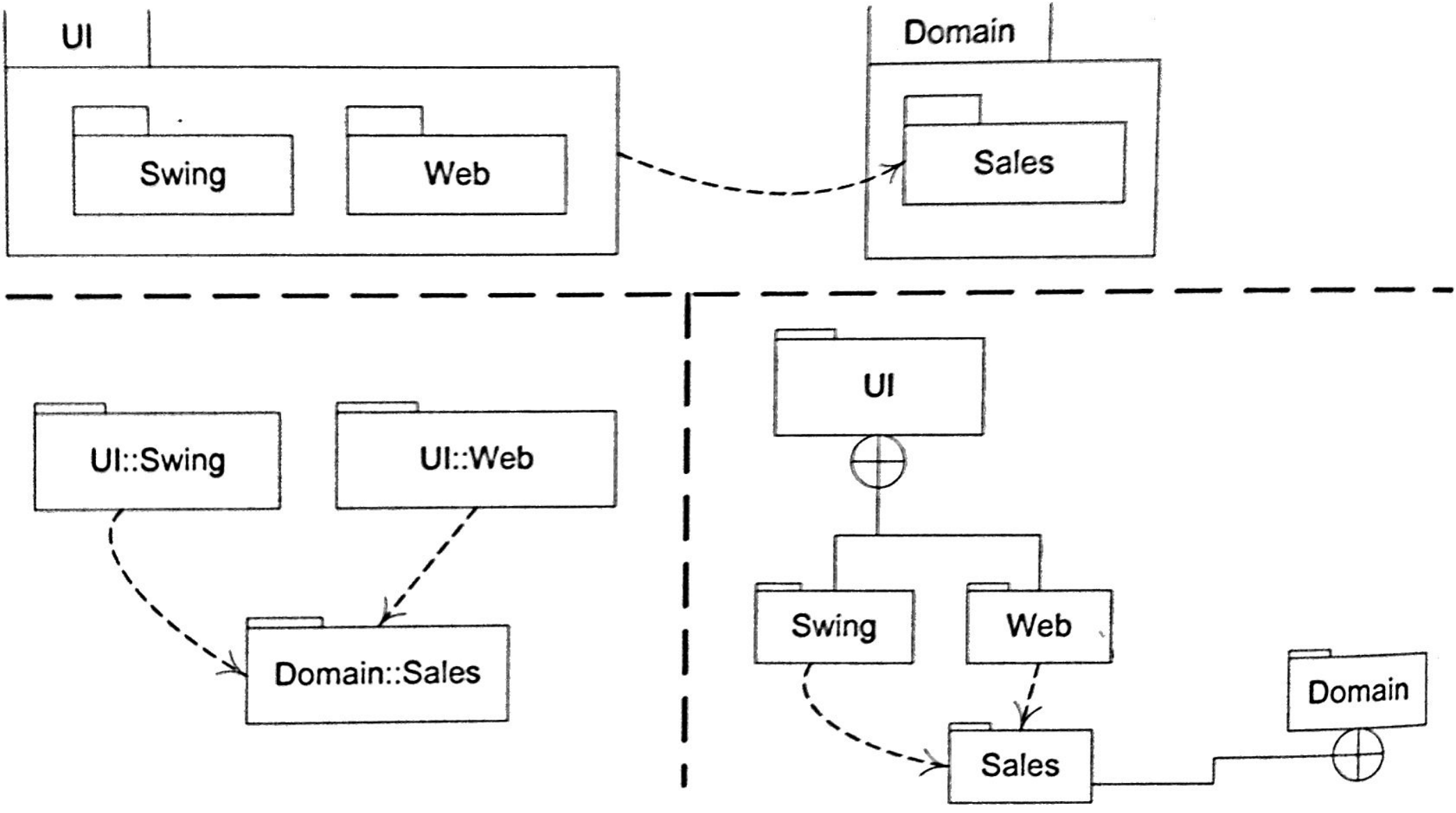

Figura 13.3 Approcci UML alternativi per mostrare l'annidamento dei package, utilizzando package immersi, nomi completamente qualificati UML e il simbolo della croce cerchiata.

13.6 Linea guida: progettazione con gli strati

Le idee essenziali dell'utilizzo degli strati [BMRSS96] sono semplici.

- Organizzare la struttura logica su larga scala di un sistema in strati separati con responsabilità distinte e correlate, con una separazione netta e coesa degli interessi, come per esempio il fatto che gli strati "inferiori" sono servizi generali e di basso livello, mentre gli strati superiori sono più specifici per l'applicazione.

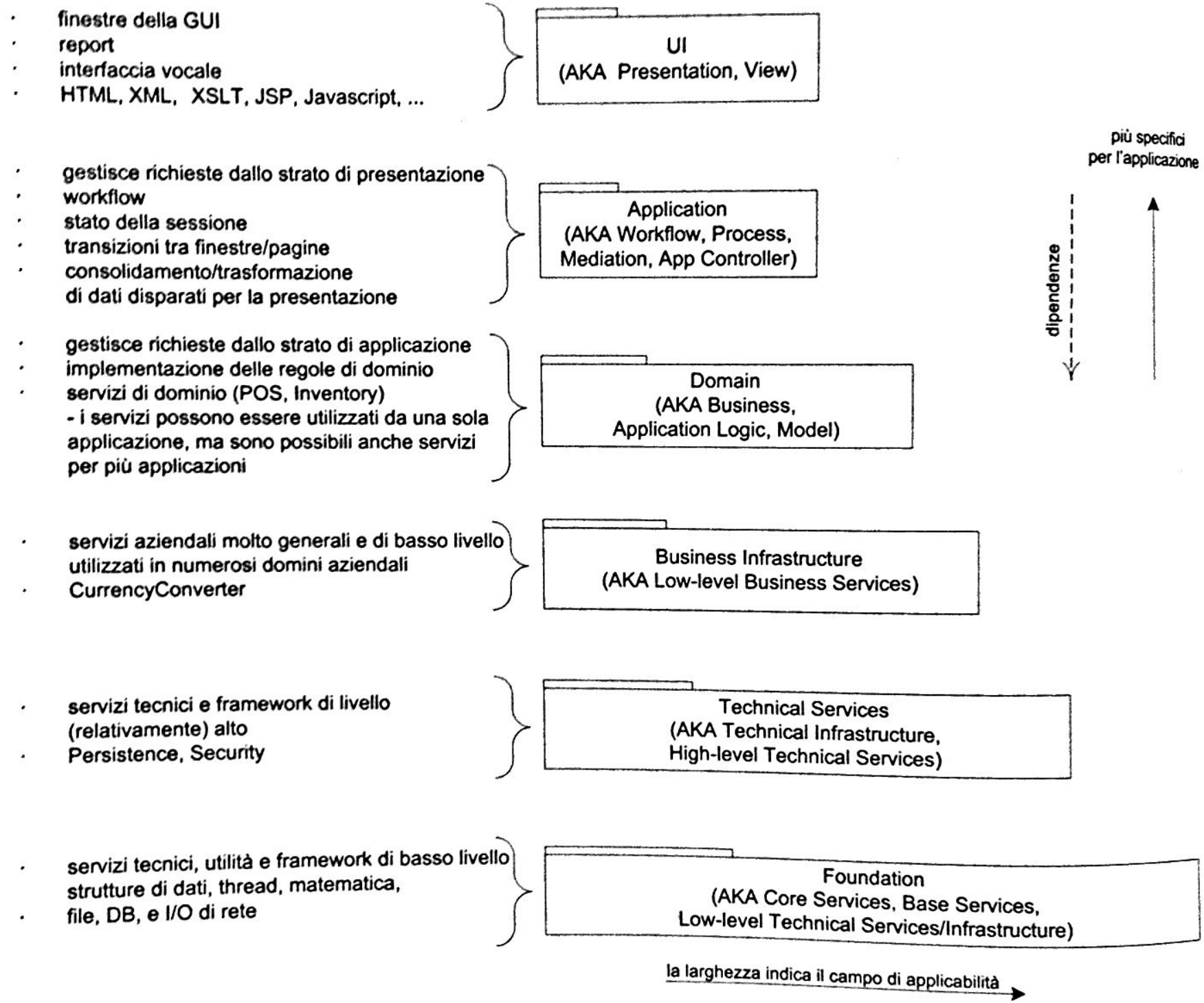

Figura 13.4 Scelta comune per gli strati nell'architettura logica di un sistema informatico.[1]

Codice: corrispondenza tra strati e package UML e l'organizzazione del codice

I linguaggi OO più diffusi (Java, C#, C++, Python, ...) forniscono il supporto ai package (chiamati namespace in C# e C++).
Il seguente esempio, che usa Java, mostra la corrispondenza tra package UML e codice. Gli strati e i package mostrati nella Figura 13.2 sono fatti corrispondere ai nomi dei package Java, nel modo seguente (si noti come il nome dello strato è usato come parte del nome del package Java):

```
// --- Strato UI

com.mycompany.nextgen.ui.swing
com.mycompany.nextgen.ui.web

// --- Strato DOMAIN

// package specifici del progetto NextGen
com.mycompany.nextgen.domain.sales
com.mycompany.nextgen.domain.payments

// --- Strato TECHNICAL SERVICES

 // il nostro strato di persistenza (accesso alla base di dati)
com.mycompany.service.persistence

 // terze parti
org.apache.log4j
org.apache.soap.rpc

 // --- Strato FOUNDATION

 // package foundation creati dal nostro team
com.mycompany.util
```

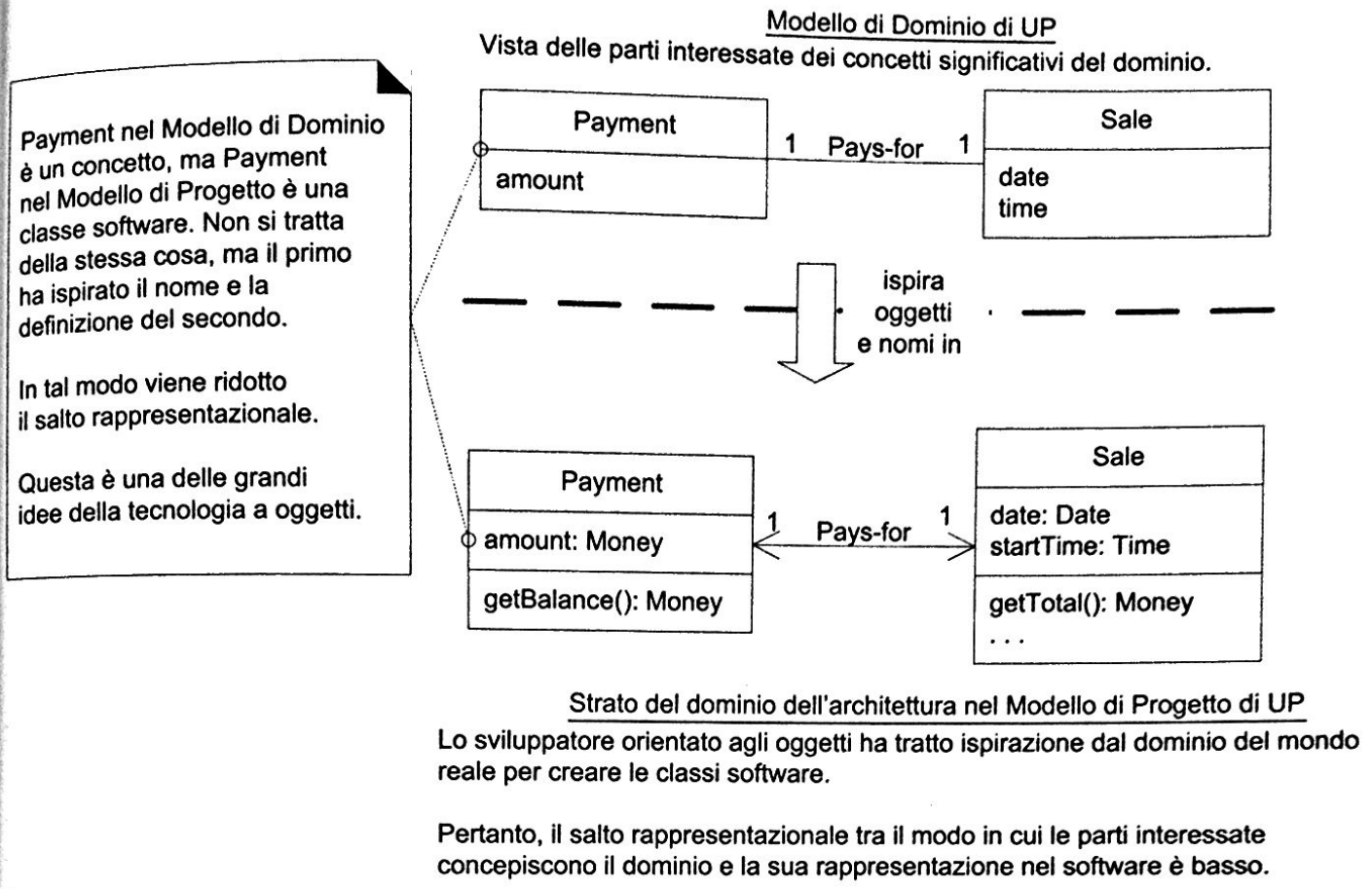

Figura 13.5 Relazione tra lo strato del dominio e il modello di dominio.

rappresentazionale basso tra il dominio del mondo reale e il progetto software. Per esempio, una vendita *Sale* nel Modello di Dominio di UP contribuisce a ispirare la creazione di una classe software *Sale* nello strato del dominio del Modello di Progetto di UP.

Definizione: livelli, strati e partizioni

La nozione originaria di **livello** (**tier**) nell'architettura era uno strato logico, non un nodo fisico; tuttavia questo termine è stato ampiamente utilizzato per indicare un nodo fisico di elaborazione (o un cluster di nodi), come il "livello client" (il computer client). Questo libro evita di usare questo termine per motivi di chiarezza, ma lo si tenga a mente quando si legge la letteratura sulle architetture.

Si dice che gli **strati** di un'architettura rappresentano le sezioni verticali, mentre le **partizioni** rappresentano una divisione orizzontale di sottosistemi relativamente paralleli di uno strato. Per esempio, lo strato *Technical Services* può essere diviso in partizioni quali *Security* e *Reporting* (Figura 13.6).

Linea guida: non mostrare le risorse esterne come lo strato più basso

La maggior parte dei sistemi si basa su risorse o servizi esterni, come una base di dati MySQL per l'inventario o un servizio di naming e directory Novell LDAP. Si tratta di componenti di implementazione *fisici*, non di uno strato dell'architettura *logica*.

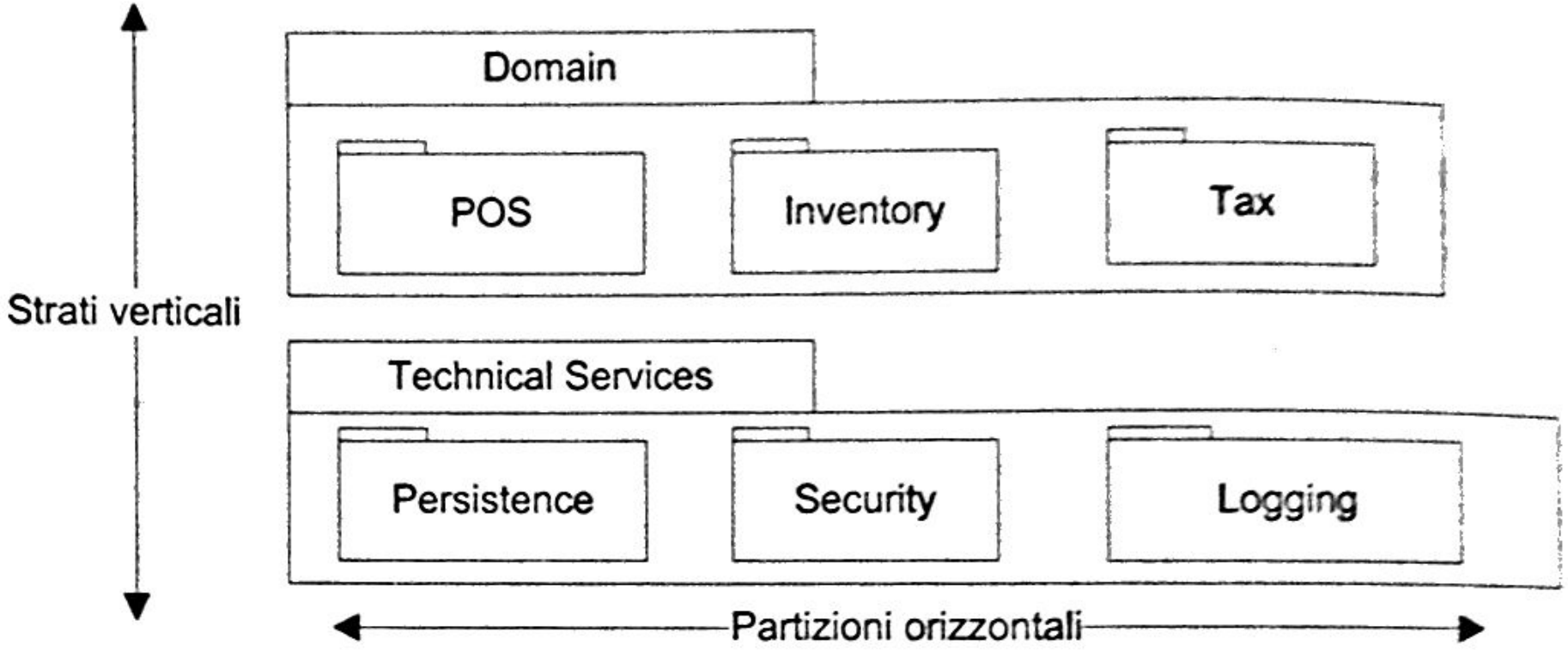

Figura 13.6 Strati e partizioni.

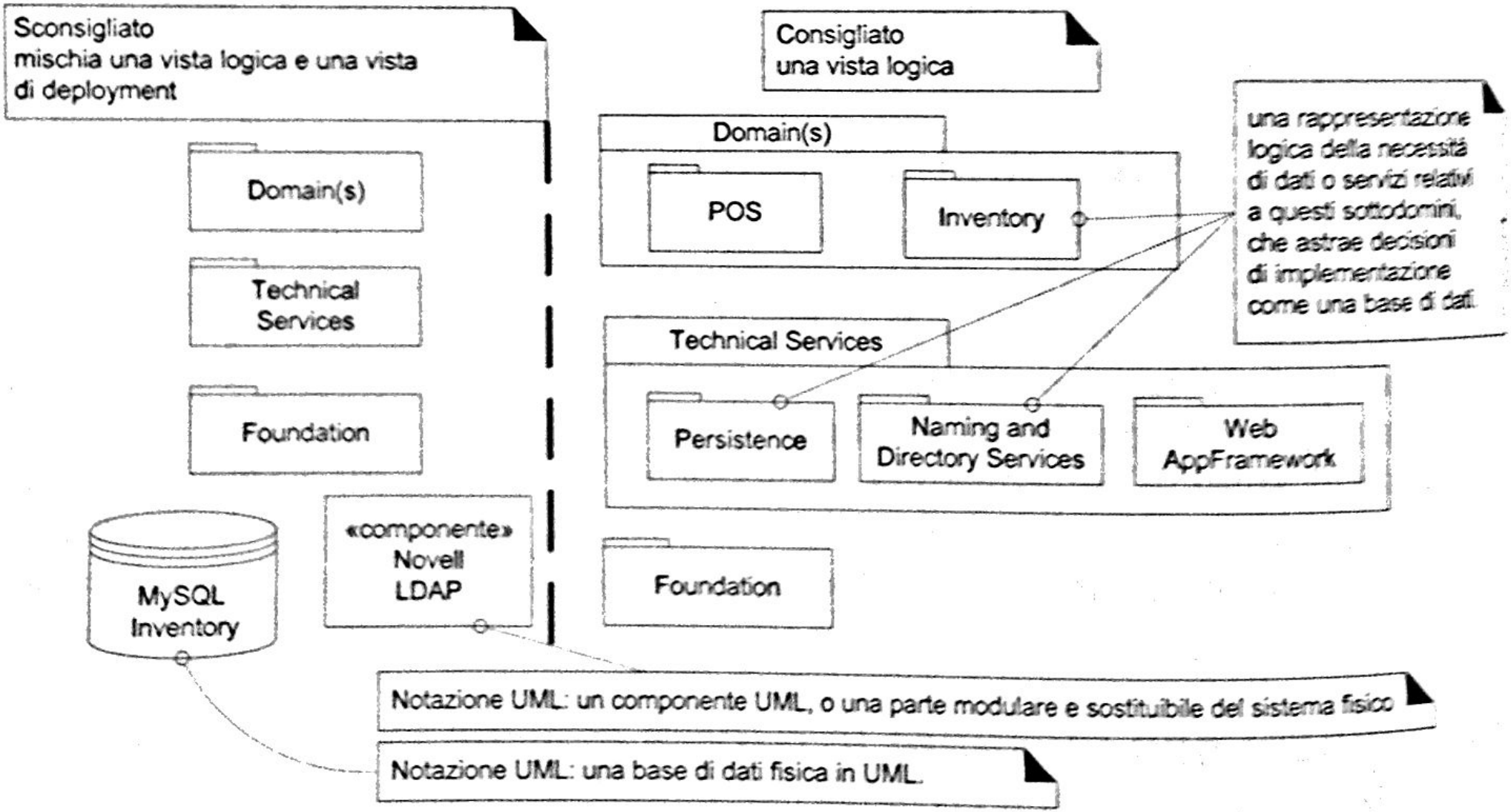

Figura 13.7 Mischiare le viste dell'architettura.

13.7 Linea guida: Principio di Separazione Modello-Vista

Che tipo di visibilità devono avere gli altri package verso lo strato UI? Come devono comunicare le classi non UI con le classi UI?

Linea guida: Principio di Separazione Modello-Vista

Questo principio è costituito da almeno due parti.

1. Gli oggetti non UI non devono essere connessi o accoppiati direttamente agli oggetti UI. Per esempio, non permettere a un oggetto software *Sale* (un oggetto di "dominio", non UI) di avere un riferimento a un oggetto finestra *JFrame* di Java Swing (un oggetto UI). Infatti le finestre sono relativa ad una applicazione specifica, mentre (idealmente) gli oggetti non appartenenti all'interfaccia grafica possono essere riusati in nuove applicazioni o vi si può accedere mediante una nuova interfaccia.

2. Non mettere logica applicativa (come il calcolo delle imposte) nei metodi di un oggetto dell'interfaccia utente. Gli oggetti UI dovrebbero solo inizializzare gli elementi dell'interfaccia utente, ricevere eventi UI (come un clic del mouse su un pulsante), e delegare le richieste di logica applicativa agli oggetti non UI (come gli oggetti di dominio).

In questo contesto, **modello** è un sinonimo per lo strato degli oggetti del dominio (è un vecchio termine OO della fine degli anni '70). **Vista** è un sinonimo per gli oggetti dell'interfaccia utente, come finestre, pagine web, applet e report.

Il principio di **Separazione Modello-Vista**[2] afferma che gli oggetti del modello (dominio) non devono avere una conoscenza *diretta* degli oggetti della vista (UI), almeno in quanto oggetti della vista. Per esempio, un oggetto *Register* o *Sale* non dovrebbe inviare

I messaggi inviati dallo strato UI allo strato del dominio saranno i messaggi mostrati negli SSD, come *enterItem*.

Per esempio, usando Java Swing, è possibile che una classe finestra della GUI di nome *ProcessSaleFrame* nello strato UI rilevi gli eventi del mouse e della tastiera che richiedono l'inserimento di un articolo; in questo caso, l'oggetto *ProcessSaleFrame* invierà un messaggio *enterItem* a un oggetto software dello strato del dominio, per esempio *Register*, per far eseguire la logica applicativa (Figura 13.8).

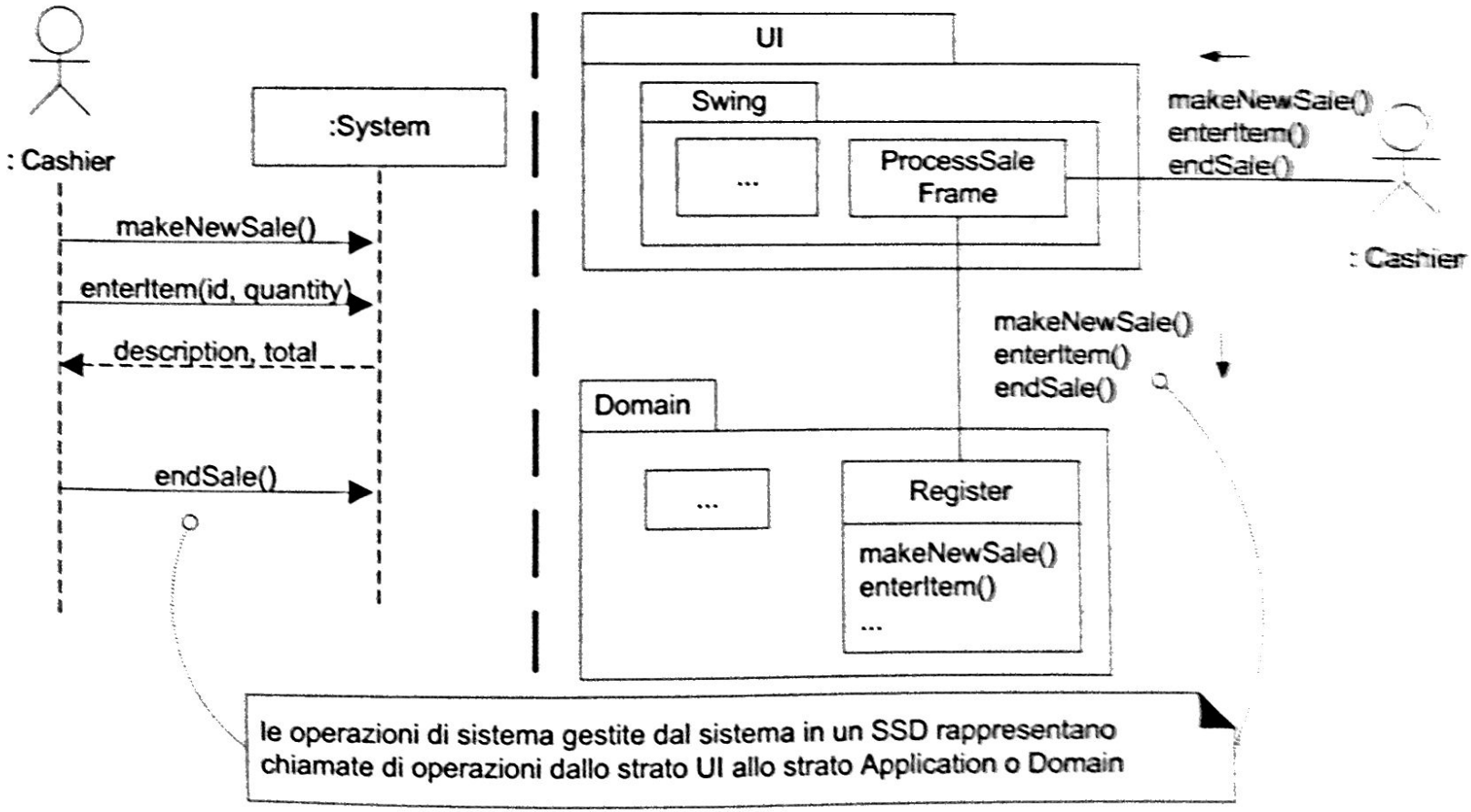

Figura 13.8 · Operazioni di sistema negli SSD e in termini di strati.