Luca Cabibbo

Architettura dei Sistemi Software

Pattern architetturale Layers

dispensa asw330

marzo 2021

Ogres are like onions.
Onions have layers. Ogres have layers.
You get it? We both have layers.

Shrek

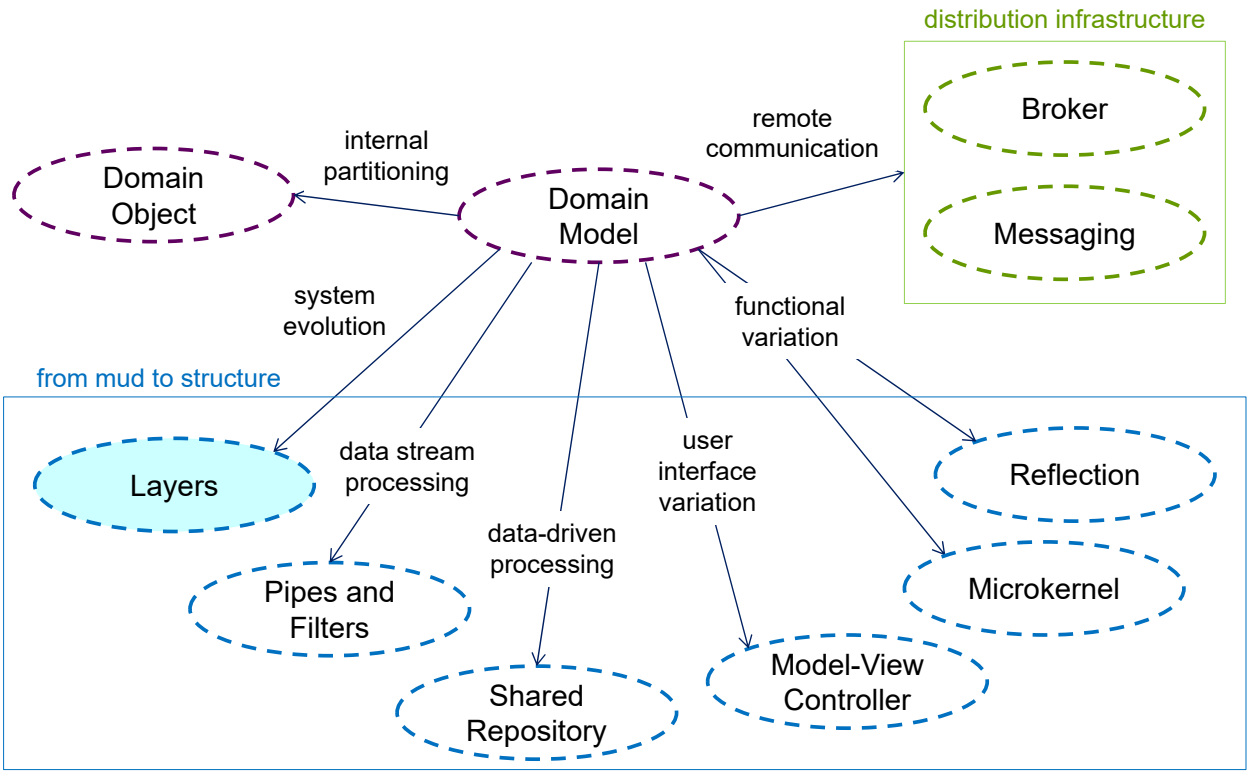
- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 17, Pattern architetturale Layers
- [POSA1] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. **Pattern-Oriented Software Architecture (Volume 1)**: A System of Patterns. Wiley, 1996.
- [POSA4] Buschmann, F., Henney, K., and Schmidt, D.C. **Pattern-Oriented Software Architecture (Volume 4)**: A Pattern Language for Distributed Computing. Wiley, 2007.
- Bachmann, F., Bass, L., and Nord, R. Modifiability Tactics. Technical report CMU/SEI-2007-TR-002. 2007.

- Obiettivi e argomenti

- Obiettivi
  - presentare Layers – il pattern architetturale POSA più diffuso
  - fare qualche considerazione generale sui pattern architetturali – dato che questo è il primo pattern architetturale “concreto” che viene presentato
  - discutere brevemente la relazione tra il pattern Layers e le tattiche per la modificabilità
- Argomenti
  - Layers (POSA)
  - Layers e tattiche per la modificabilità
  - discussione

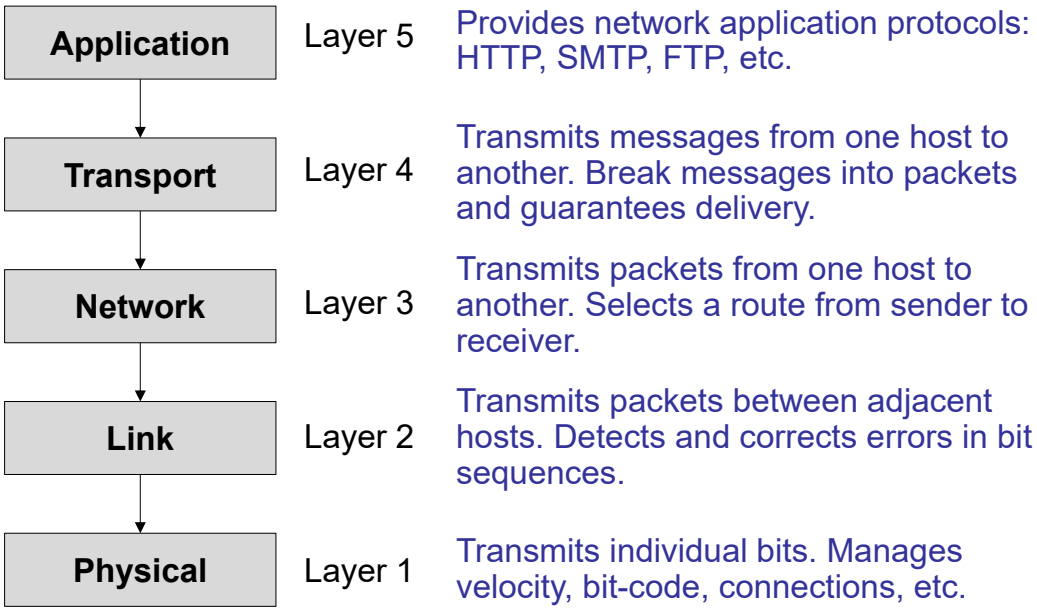
* Layers (POSA)



Layers

- Il pattern architetturale **Layers** – noto anche come *architettura a strati*
  - nella categoria “system evolution” di [POSA4]
  - aiuta a strutturare applicazioni che possono essere decomposte in gruppi di compiti, a diversi livelli di astrazione

Esempio



Layers

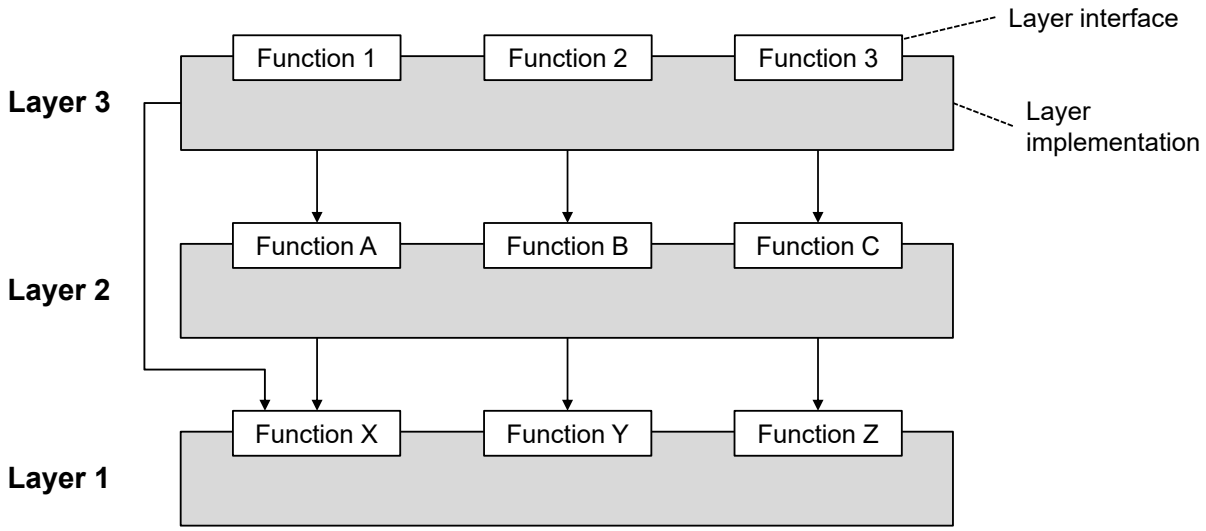
- Contesto
  - un sistema grande – richiede di essere decomposto
  - le diverse parti devono poter essere sviluppate e devono poter evolvere in modo indipendente
- Problema
  - il sistema deve essere decomposto in parti che possano essere sviluppate e fatte evolvere in modo indipendente tra loro
    - è richiesta modificabilità, riusabilità e/o portabilità
  - la decomposizione deve essere basata su una separazione degli interessi ragionata – e non deve penalizzare altre qualità

Layers

- Soluzione
  - decomponi il sistema in una gerarchia verticale di elementi software, chiamati *strati*
  - ciascuno strato ha una responsabilità distinta e ben specifica
  - costruisci le funzionalità di ciascuno strato in modo che dipendano solo da se stesso o dagli strati inferiori
  - fornisci, in ciascuno strato, un'interfaccia che è separata dalla sua implementazione – e basa la comunicazione tra strati solo su queste interfacce

Layers

- Soluzione – struttura statica della soluzione



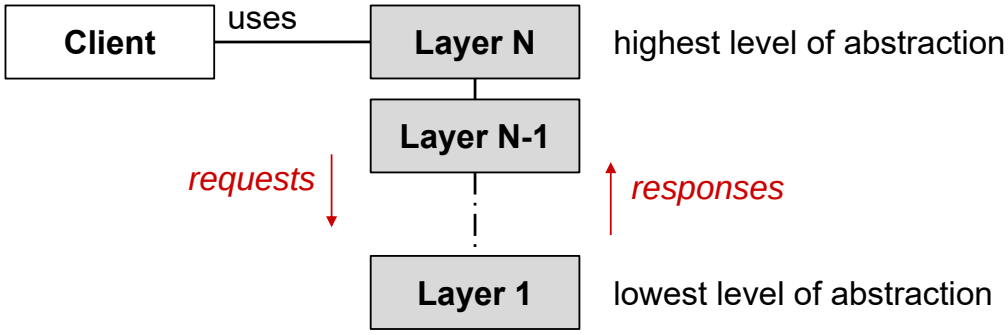
  - ma di che natura sono gli strati?

Pattern, soluzione e scenari

- La *soluzione* di un pattern descrive il “principio” o l’“idea risolutiva” fondamentale del pattern
  - comprende sia la *struttura statica* che il *comportamento dinamico* del pattern
- Gli aspetti dinamici di un pattern possono essere descritti sotto forma di scenari
  - ciascuno *scenario* descrive un possibile comportamento dinamico archetipale della soluzione
  - gli scenari possono anche descrivere modi diversi di applicare il pattern

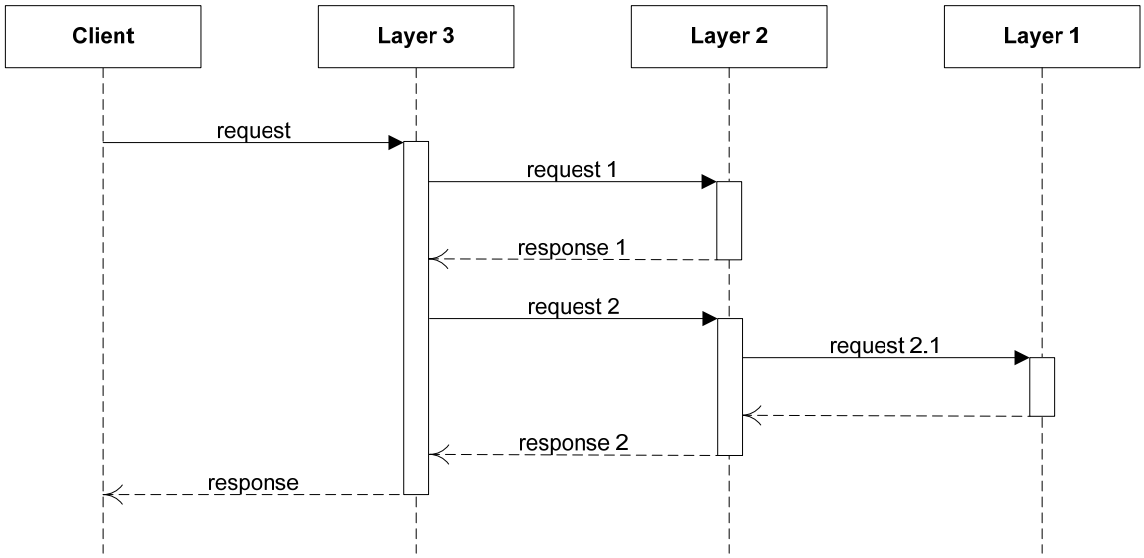
Scenario 1 – comunicazione top-down

- Lo scenario di Layers più comune è la *comunicazione top-down* – di tipo *richiesta-risposta*



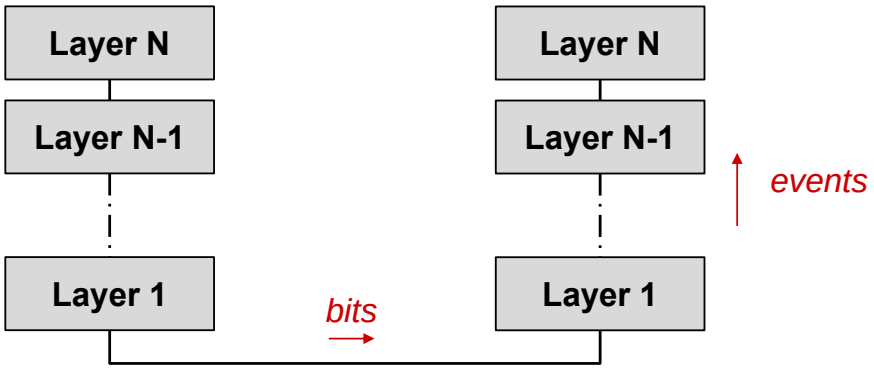
Scenario 1 – comunicazione top-down

- Lo scenario di Layers più comune è la *comunicazione top-down* – di tipo *richiesta-risposta*



Scenario 2 – comunicazione bottom-up

- Un altro scenario comune è la *comunicazione bottom-up* – basata sulla *notifica di eventi*



Altri scenari

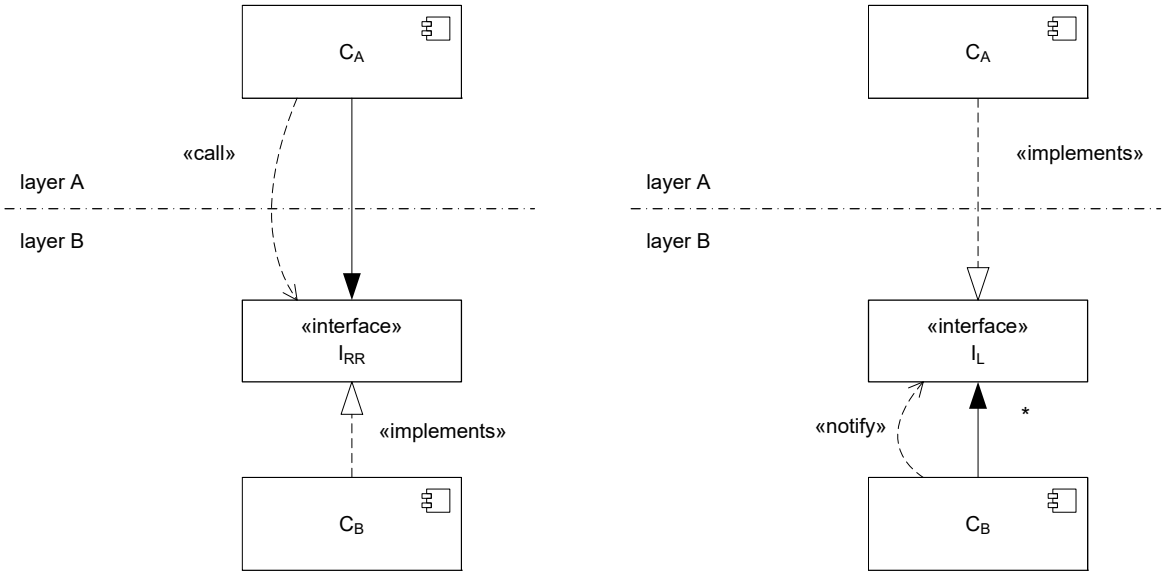
- Sono possibili anche altri scenari
  - una richiesta allo strato N viene servita in uno strato intermedio – ad es., nello strato N-1 o N-2
  - una notifica dallo strato 1 viene gestita in uno strato intermedio – ad es., nello strato 2 o 3
  - una richiesta viene gestita da un sottoinsieme degli strati delle due pile

Scenari – discussione

- Gli scenari di Layers sono tutti basati su due modalità di comunicazioni – la comunicazione *richiesta-risposta* e la *notifica di eventi*
  - in corrispondenza, ciascuno strato può definire diversi tipi di interfacce
    - in uno strato più basso, verso l'alto
      - un'interfaccia fornita, per ricevere l'invocazione di operazioni
      - un'interfaccia fornita, per notificare eventi
    - in uno strato più alto, verso il basso
      - un'interfaccia richiesta, per invocare operazioni
      - un'interfaccia richiesta, per accettare notifiche di eventi

Scenari – discussione

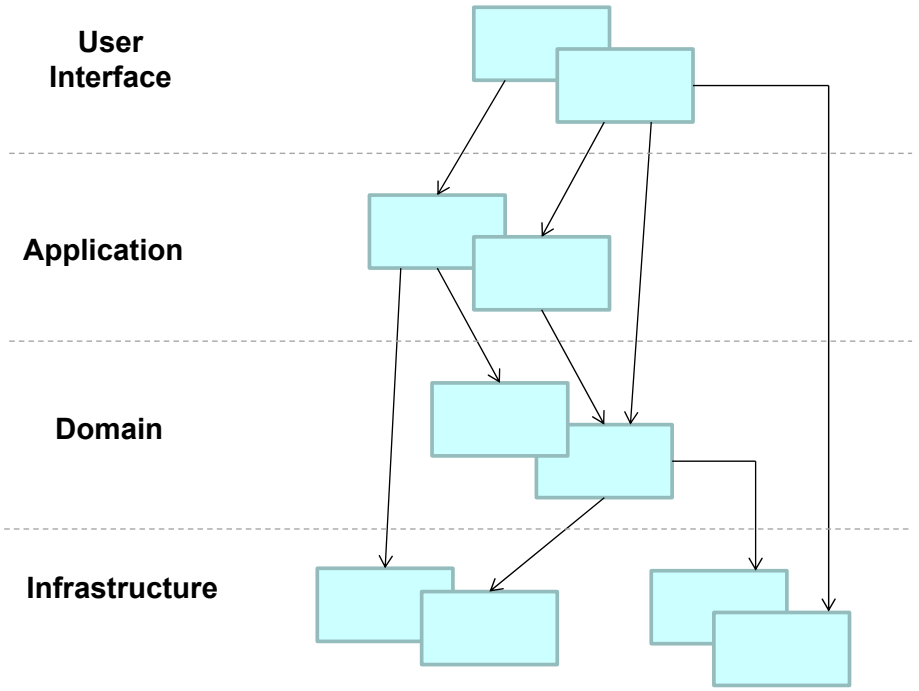
- Nell'architettura a strati, le dipendenze sono comunque dall'alto verso il basso



- Applicazione di Layers

- Definisci il criterio di partizionamento delle funzionalità
  - discusso più avanti
- Determina gli strati
  - determina il numero di strati e le loro responsabilità
- Specifica i servizi offerti da ciascuno strato
- Raffina la definizione degli strati – in modo iterativo
- Definisci l'interfaccia di ciascuno strato
  - quali i servizi offerti da ciascuno strato? quali le notifiche accettate?
- Struttura individualmente gli strati
  - discusso più avanti
- Specifica la comunicazione tra strati – top-down o bottom-up
- Disaccoppia gli strati – usa opportuni design pattern
- Progetta una strategia per la gestione degli errori

- Esempio: Layered Architecture [DDD]



Layered Architecture (DDD)

- Un esempio comune – la *Layered Architecture* [DDD]
  - *presentation layer* – UI o presentazione
  - *application layer* – applicazione
  - *domain layer* – dominio
  - *infrastructure layer* – infrastruttura

Layered Architecture

- Il *domain layer* rappresenta il modello di dominio
  - gli oggetti di dominio – liberi dalla responsabilità di visualizzarsi, di memorizzarsi e di gestire compiti dell'applicazione – sono focalizzati sulla rappresentazione del modello di dominio
  - è responsabile di gestire lo stato complessivo del sistema

Layered Architecture

- L'*application layer* definisce i compiti che il sistema è chiamato a fare
  - due varianti principali
    - un insieme di facade sottili, che delegano lo svolgimento delle operazioni di sistema a oggetti opportuni dello strato del dominio
    - un insieme di classi più spesse che definiscono degli "operation script" – che implementano direttamente la logica applicativa, ma delegano la logica di dominio allo strato del dominio
  - è responsabile di gestire lo stato delle conversazioni/sessioni con i suoi client

- Criteri di decomposizione

- Nell'applicazione di Layers, è necessario scegliere un criterio di decomposizione delle funzionalità
  - in generale, la decomposizione delle funzionalità dovrebbe garantire che ciascun gruppo di funzionalità sia incapsulato in uno strato e che possa essere sviluppato ed evolvere indipendentemente dagli altri gruppi di funzionalità
  - in pratica, ci sono diversi criteri specifici comuni per la decomposizione delle funzionalità – astrazione, granularità, distanza dall'hardware e tasso di cambiamento

Criteri di decomposizione

- Criteri di decomposizione comuni
  - il criterio dell'*astrazione* può essere applicato nei sistemi che si devono occupare della gestione di diversi aspetti, a livelli di astrazione differenti
    - questo criterio viene utilizzato anche per motivare una decomposizione tecnica in strati come presentazione, logica applicativa e infrastruttura/servizi tecnici – che sono responsabilità a livelli di astrazione differenti

Criteri di decomposizione

- Criteri di decomposizione comuni
  - il criterio della *granularità* può portare a una suddivisione in uno strato con oggetti o servizi di business che vengono usati da uno strato di processi di business
  - il criterio della *distanza dall'hardware* può portare a una suddivisione con uno strato che definisce astrazioni del sistema operativo, uno strato di protocolli di comunicazione e uno strato di funzionalità applicative
  - il criterio del *tasso di cambiamento atteso* suggerisce di separare le funzionalità – mettendo in basso le cose più stabili e in alto quelle meno stabili (perché?)
  - la decomposizione in strati può anche fare riferimento alla combinazione di diversi criteri

Sul numero degli strati

- Ma quel è il numero “giusto” di strati da utilizzare?
  - l’obiettivo è favorire un’evoluzione indipendente degli strati
  - troppi pochi strati potrebbero non separare abbastanza i differenti aspetti che il sistema deve gestire
  - troppi strati potrebbero frammentare eccessivamente l’architettura del software – e rendere difficile la sua evoluzione
  - è una scelta difficile da cambiare

- Layers e team di sviluppo

- Se Layers viene applicato come decomposizione di primo livello, gli strati vengono in genere assegnati a team di sviluppo separati
  - per favorire un’evoluzione indipendente degli strati, la decomposizione in strati non può essere indipendente dall’organizzazione dei team
  - una considerazione importante riguarda il costo del coordinamento – quello inter-team è maggiore di quello intra-team – pertanto
    - la maggior parte delle modifiche attese dovrebbero avere impatto solo su singoli strati
    - ogni strato dovrebbe dipendere solo in modo debole dagli strati inferiori

Layers e team di sviluppo

- Una critica al pattern Layers, legata alla legge di Conway
  - alcune organizzazioni adottano team di sviluppo mono-funzionali, con riferimento a un'applicazione "standard" dell'architettura a strati – con presentazione, logica applicativa e infrastruttura – indipendentemente dal dominio applicativo dei sistemi software che sviluppano
  - in genere, queste organizzazioni tendono a produrre sistemi software con un accoppiamento elevato – anziché con parti indipendenti
    - il problema non è nell'architettura a strati – ma nel modo in cui essa viene applicata!

- Conseguenze

- Modificabilità
  - ☺ la modificabilità può essere alta
  - ☺ è possibile sostenere la portabilità
  - 😐 la modificabilità dipende da come i cambiamenti si ripercuotono sul sistema
  - ☹ alcuni cambiamenti potrebbero coinvolgere molti strati
  - ☹ è difficile cambiare la scelta degli strati e l'assegnazione di responsabilità agli strati

Conseguenze

- Prestazioni
  - ☹ la comunicazione tra strati può penalizzare le prestazioni
  - 😐 allocare ciascuno strato a un diverso processo non migliora necessariamente le prestazioni
  - ☺ si possono migliorare le prestazioni associando un diverso thread di esecuzione a ciascun evento da elaborare
- Affidabilità e disponibilità
  - ☹ se le richieste devono essere elaborate da molti strati, la verifica sistema può essere più difficile
  - ☺ uno strato più alto di gestire guasti che si verificano negli strati inferiori
  - ☺ è possibile introdurre degli strati intermedi per effettuare il monitoraggio del sistema

Conseguenze

- Sicurezza
  - ☺ è possibile inserire strati per introdurre opportuni meccanismi di sicurezza
- Altre conseguenze
  - ☺ possibilità di riusare strati
  - 😐 può aumentare gli sforzi iniziali e la complessità del sistema – ma questi sforzi sono poi di solito ripagati
  - ☹ può essere difficile stabilire la granularità/il numero/il livello di astrazione degli strati

- Discussione

- [POSA4] colloca Layers nella categoria “system evolution”
  - per sostenere la modificabilità, è basato sui principi di modularità e di separazione degli interessi
    - un progetto è *modulare* se è caratterizzato da accoppiamento basso e coesione alta
    - il principio di *separazione degli interessi* tende a separare interessi diversi in elementi differenti

- Usi conosciuti

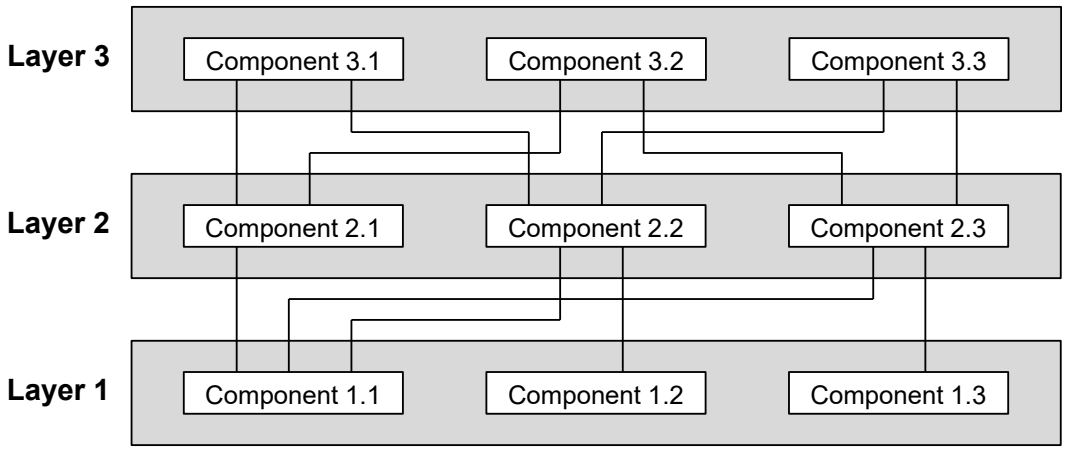
- Il pattern Layers è applicato in modo pervasivo
  - ad es., nei sistemi informativi e nei sistemi basati su macchine virtuali
  - nell’architettura client-server, a due o più livelli
  - nell’architettura ANSI a tre livelli dei DBMS (esterno, logico, interno)
    - che sostiene l’indipendenza dei livelli
  - nell’architettura a componenti e nell’architettura orientata ai servizi
  - nell’architettura del cloud
  - ...

- Sull'applicazione di Layers

- Il pattern Layers può essere applicato
  - come decomposizione di primo livello, per la strutturazione di un intero sistema
  - come decomposizione di secondo livello, per un componente o un sotto-sistema – infatti è comune organizzare a strati il codice dei singoli componenti architetturali

Sull'applicazione di Layers

- Gli strati possono anche essere strutturati internamente



* Layers e tattiche per la modificabilità

- *Increase semantic coherence – increase cohesion*
  - le responsabilità vengono assegnate agli strati in modo che abbiano una qualche coerenza semantica
- *Encapsulate – reduce coupling*
  - ciascuno strato espone i propri servizi mediante un'interfaccia
- *Abstract common services – reduce coupling*
  - uno strato può fornire servizi comuni e astratti agli strati superiori
  - gli strati possono definire una "scala" di servizi via via più astratti

Layers e tattiche per la modificabilità

- *Use an intermediary – reduce coupling*
  - uno strato può avere lo scopo di definire un'interfaccia nei confronti di uno strato inferiore – ad es., una facade
- *Restrict dependencies – reduce coupling*
  - uno strato può costituire un intermediario tra gli strati più in alto e gli strati più in basso
  - si noti che l'*architettura a strati rilassata* è basata sulla rimozione di intermediari e di *Restrict dependencies*

* Discussione

- Layers è un pattern architetturale fondamentale, che viene usato nell'organizzazione di molti sistemi software
  - Layers, come gli altri pattern architetturali
    - identifica alcuni tipi specifici di elementi e di possibili modalità di interazione tra questi elementi
    - descrive criteri per effettuare la decomposizione sulla base di questi tipi di elementi e delle possibili relazioni tra essi
    - il criterio specifico di identificazione degli elementi/ componenti può far riferimento a qualche modalità di modellazione del dominio del sistema
    - discute la possibilità di raggiungere (o meno) alcuni attributi di qualità