Dal modello di analisi si arriva alla realizzazione di un modello di progettazione del sistema. Tale modello gli sviluppatori definiscono gli obiettivi di progettazione del progetto e scompongono il sistema in sottosistema più piccoli in modo da poter essere realizzati dai singoli team. Selezionano strategie per la costruzione del sistema, come la strategia hardware/software, la strategia di gestione dei dati persistente, il flusso di controllo globale,ecc.. Come risultato si avrà un modello che include una decomposizione del sottosistema e una chiara descrizione di ciascuna di queste strategie.
SD non è algoritmica quindi bisogna fare un compresso tra molti obiettivi di progettazione che spesso sono in conflitto tra loro.
La progettazione del sistema è scomposta in diverse attività, quali:

1. Identificare gli obiettivi di progettazione (design goal) : gli sviluppatori identificano e danno la priorità alle qualità del sistema che dovrebbero ottimizzare.
2. Progettare la decomposizione iniziale del sottosistema:gli sviluppatori decompongono il sistema in parti più piccole in base al caso d'uso e ai modelli di analisi. Gli sviluppatori utilizzano stili architettonici standard come punto di partenza durante tale attività.
3. Affinare la scomposizione del sottosistema per raggiungere gli obiettivi di progettazione: la decomposizione iniziale di solito non soddisfa tutti gli obiettivi di progettazione. Gli sviluppatori lo perfezionano fino al raggiungimento di tutti gli obiettivi.

SD crea dei prodotti che nel modello di analisi non sono state fatte come:

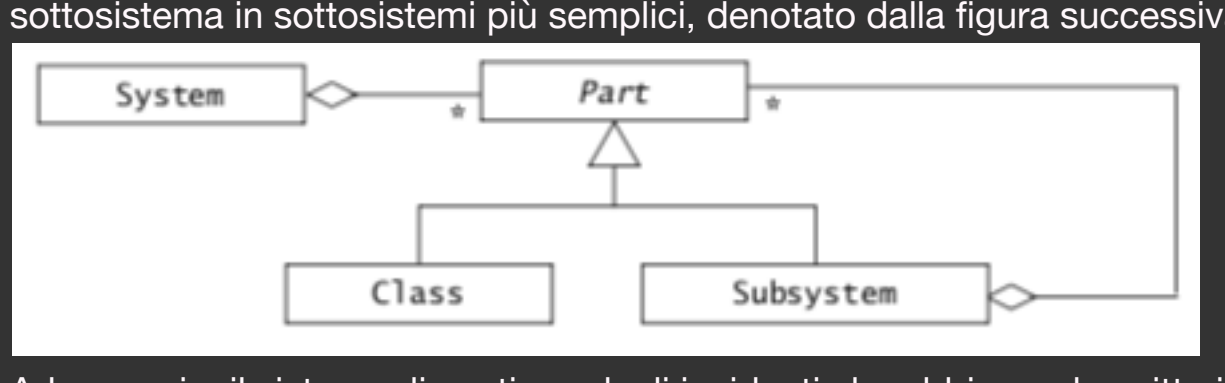
- progettare obiettivi, descrivendo la qualità del sistema che gli sviluppatori dovrebbero ottimizzare.
- architettura del software, che descrive la decomposizione del sottosistema in termini di responsabilità del sottosistema, dipendenze tra sottosistemi, mappatura dei sottosistemi all'hardware e principali decisioni politiche come flusso di controllo, controllo degli accessi e archiviazione dei dati.
- casi d'uso boundary che descrivono la configurazione del sistema, l'avvio, l'arresto e la gestione delle eccezioni.

perché il modello di analisi pensa soltanto a descrivere il sistema dal punto di vista degli attori come:

- una serie di requisiti e vincoli non funzionali
- un modello di caso d'uso
- un modello a oggetti
- un diagramma di sequenza

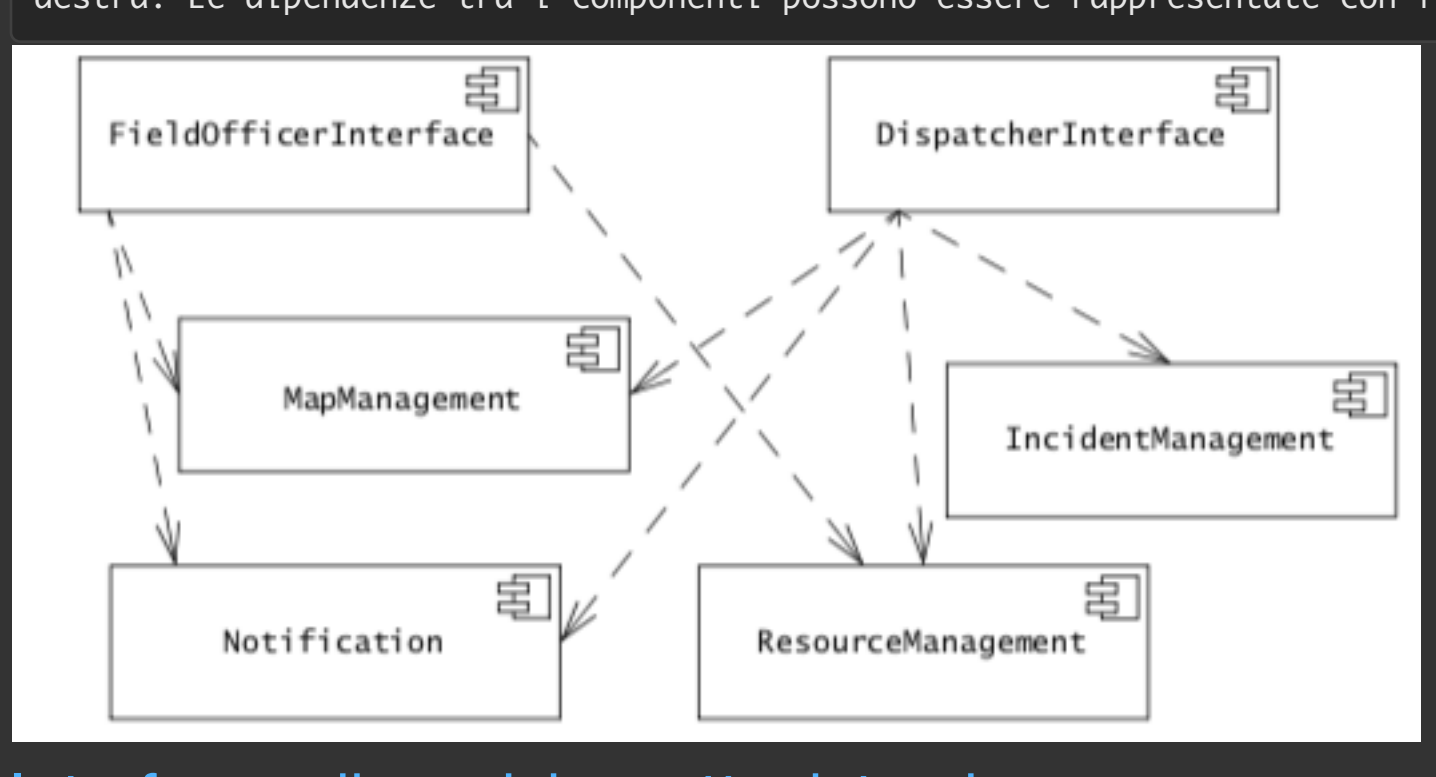
Sottosistemi e classi

Per ridurre la complessità della soluzione, decomponiamo un sistema in parti più semplici, chiamate sottosistemi, costituite da un numero di classi del dominio della soluzione.
In genere corrisponde alla quantità di lavoro che può affrontare un singolo sviluppatore o un singolo team di sviluppo. Decomponendo il sistema in sottosistemi indipendenti, i team simultanei possono lavorare su singoli sottosistemi con un sovraccarico di comunicazione minimo. Nel caso di sottosistemi complessi, applichiamo ricorsivamente questo principio e scomponiamo un sottosistema in sottosistemi più semplici, denotato dalla figura successiva.



Ad esempio, il sistema di gestione degli incidenti che abbiamo descritto in precedenza può essere scomposto in un sottoinsieme DispatcherInterface che realizza l'interfaccia utente per Dispatcher; un sottosistema FieldOfficerInterface che realizza l'interfaccia utente per FieldOfficer; un sottosistema IncidentManagement, responsabile della creazione, modifica e archiviazione degli incidenti; un sottosistema ResourceManagament, responsabile del monitorato delle risorse disponibili, un MapManagement per la rappresentazione di mappe e posizioni e un Notification che implementa la comunicazione tra i terminali FieldOfficer e le stazioni Dispatcher.
Modellazione dei sottosistemi in UML

```
Questa decomposizione viene mostrata utilizzando componenti UML. I componenti sono rappresentati come rettangoli con l'icona del componente nell'angolo in alto a destra. Le dipendenze tra i componenti possono essere rappresentate con frecce a bastoncino tratteggiate.
```



Interfacce di servizi e sottosistemi

Un sottosistema è caratterizzato dai servizi che fornisce ad altri sottosistemi.
Un servizio è un insieme di operazioni correlate che condividono uno scopo comune.
Esempio: un sottosistema che fornisce un servizio di notifica definisce le operazioni per inviare notifiche, cercare canali di notifica e iscriversi e annullare l'iscrizione a un canale.
L'insieme delle operazioni di un sottosistema disponibili per altri sottosistemi costituisce l'interfaccia del sottosistema. Tale interfaccia include il nome delle operazioni, i loro parametri, i loro tipi e i loro valori di ritorno. Operazione che viene fatta nell'OD.

```
Quindi SD si concentra sulla definizione dei servizi forniti da ciascun sottosistema, ovvero enumerando le operazioni, i loro parametri e il loro comportamento di alto livello.
```

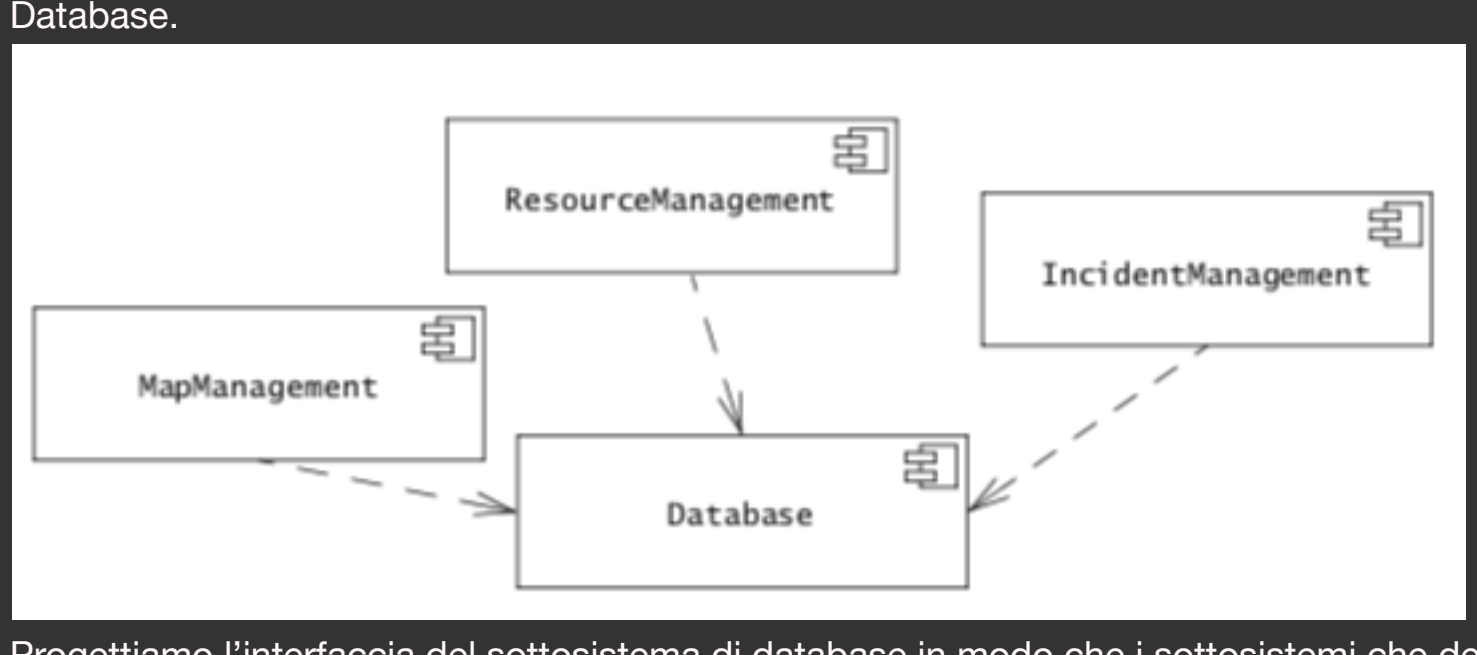
La definizioni dei sottosistemi in termini di servizi forniti ci aiuta a concentrarci sulla sua interfaccia anziché sulla sua implementazione. Quando si scrive un'interfaccia del sottosistema, è necessario cercare di ridurre al minimo la quantità di informazioni fornite sull'implementazioni come non bisogna far riferimento alla struttura di dati interna quali array, list, table hash. Questo ci consente di ridurre al minimo l'impatto del cambiamento quando rivediamo l'implementazioni di un sottosistema.

Accoppiamento e coesione

L'accoppiamento è il numero di dipendenze tra due sottosistemi. Se due sottosistemi sono accoppiati liberamente, sono indipendenti, quindi le modifiche a uno dei due sottosistemi avranno un impatto minimo sull'altro. Ma se fossero fortemente accoppiati, le modifiche poste a un sottosistema avrà un forte impatto all'altro sottosistema.

```
In tutto ciò, è desiderabile che tra i due sottosistemi non siano fortemente accoppiati ( loosely coupled) in modo da ridurre l'impatto sugli errori o cambiamenti futuri posto tra due sottosistemi.
```

Supponiamo , considerando la figura nominata in precedenza, che vogliamo immagazzinare tutti i dati persistenti in un database relazionale. Ciò ci porta a creare un nuovo sottosistema chiamato Database.



Progettiamo l'interfaccia del sottosistema di database in modo che i sottosistemi che devono archiviare i dati emettono dei comandi query, come SQL. Ma ciò porta un elevato accoppiamento tra il sottosistema e i tre sottosistemi client poiché qualunque cambiamento nel modo i cui i dati vengono archiviati richiederà modifiche nei corrispettivi sottosistemi client. Ad esempio. se cambiamo i fornitori di database dovremo cambare i sottosistemi per utilizzare un dialetto diverso dalla linqua query.
Quindi per ridurlo, creiamo un nuovo sottosistema "Storage" che protegge il database dagli altri sottosistemi. Pertanto qualunque modifica subirà il database, basterà modificare soltanto il sottosistema storage.
Con questo però abbiamo aumentato la complessità. Riducendo l'accoppiamento, gli sviluppatori possono introdurre molti livelli inutili di astrazioni che consumano tempo di sviluppo e tempo di elaborazione. Bisogna ridurre l'accoppiamento ogniqualvolta è probabile che qualsiasi sottosistemi cambi.

La coesione è il numero di dipendenze all'interno di un sottosistema. Se un sottosistema contiene molti oggetti correlati tra loro ed eseguono attività simili, la sua coesione è elevata. Mentre se un sottosistema contiene oggetti non correlati, la sua coesione è bassa.

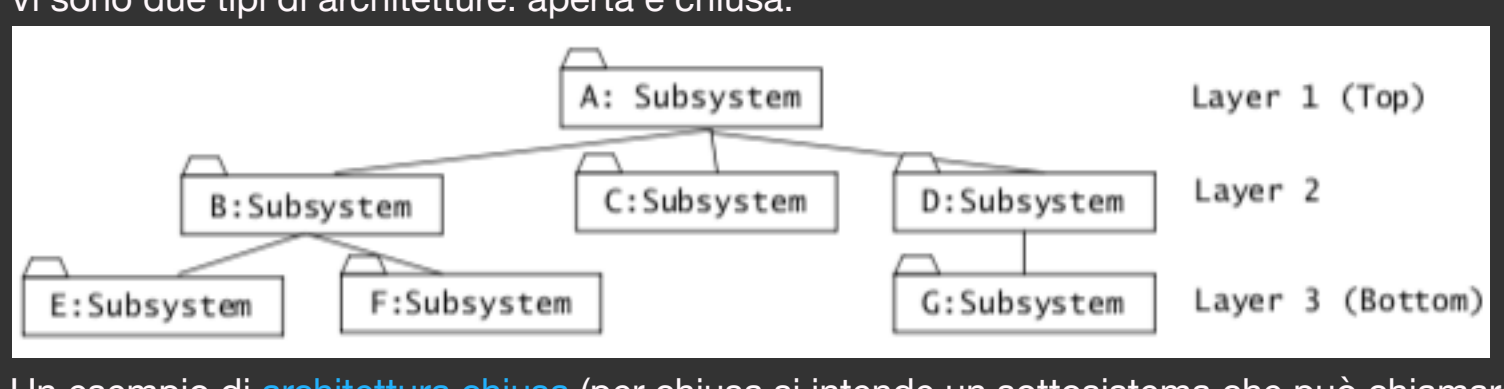
```
Quindi è desiderabile che la decomposizione di un sistema crei dei sottosistemi con elevata coesione.
```

In generale, esiste un compresso tra coesione e accoppiamento. Spesso è possibile aumentare la coesione decomponendo il sistema in sottosistemi più piccoli. Tuttavia, ciò aumenta anche l'accoppiamento all'aumentare del numero di interfacce.

```
Una buona euristica è che gli sviluppatori possono gestire 7+2 concetti a qualsiasi livello di astrazione. Se ci sono più di nove sottosistemi a un dato livello di astrazione o se un sottosistema fornisce più di nove servizi, è necessario considerare di rivedere la decomposizione. Una buona progettazione può spesso essere realizzata con soli tre strati.
```

Layers e Partition

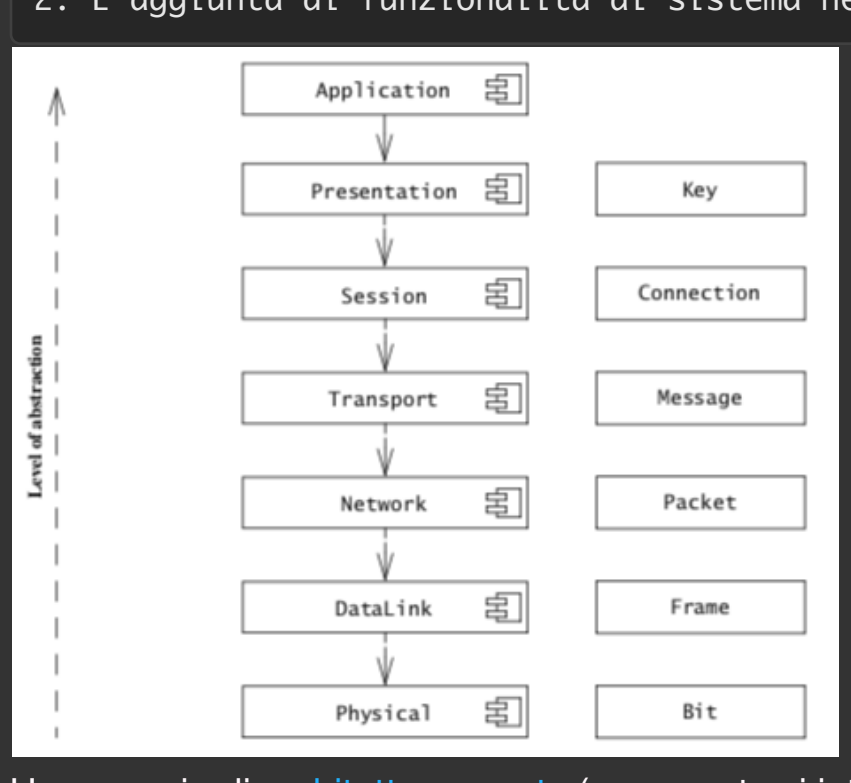
Una decomposizione gerarchica di un sistema produce un insieme ordinato di livelli. Un livello è un raggruppamento di sottosistemi che forniscono servizi correlati, realizzato utilizzando servizi di un altro livello. I livelli sono ordinati in quanto ogni livello può dipendere solo da livelli di ordine inferiore e non ha conoscenza dei livelli sopra di esso. Il livello che non dipende da nessun altro è chiamato livello inferiore e il livello non utilizzato da nessun altro è chiamato livello superiore.
Vi sono due tipi di architetture: aperta e chiusa.



Un esempio di architettura chiusa (per chiusa si intende un sottosistema che può chiamare solo le operazioni dello strato sottostante) è il modello OSI che è composto da sette strati. Ogni livello è responsabile dell'esecuzione di una funzione ben definita. Inoltre, ogni livello fornisce i propri servizi utilizzando i servizi del livello sottostante.

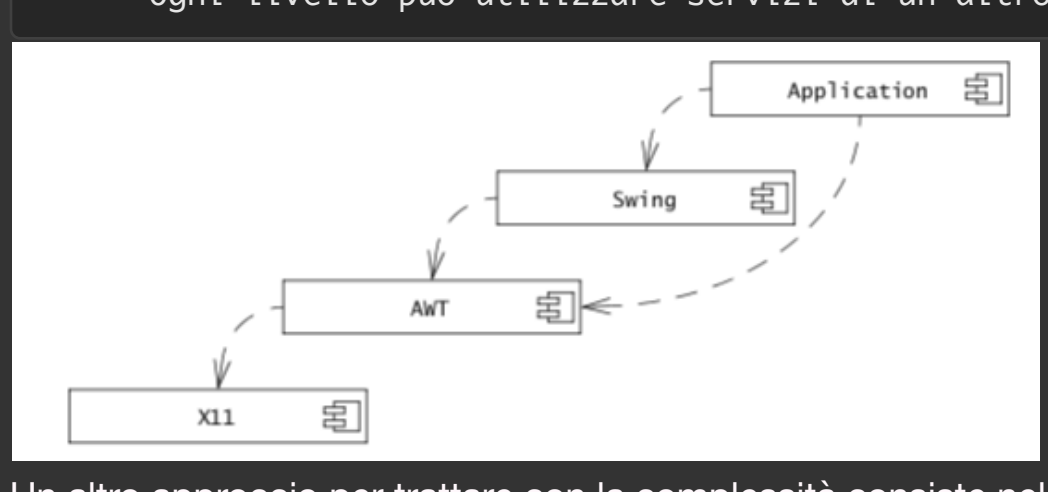
```
Vantaggi delle architetture chiuse:
1. Basso accoppiamento tra le componenti
    -Ogni layer conosce soltanto l'interfacce dei livelli sovrastanti. Quindi ogni modifica subita nel livelli più profondi non avranno un impatto nei livelli superiori.
2. Integrazione e testing incrementale

Svantaggi:
1. Ogni livello aggiunge overhead in termini di tempo e memoria che può rendere difficile soddisfare requisiti non funzionali
2. L'aggiunta di funzionalità al sistema nelle revisioni successive può rivelarsi difficile.
```



Un esempio di architettura aperta (per aperta si intende un sottosistema che può chiamare servizi dei sottosistemi degli strati sottostanti) è il toolkit di interfaccia utente Swing per Java. Il livello più basso è fornito dal sistema operativo o da un sistema a finestra e fornisce una gestione di base delle finestre. AWT è un'interfaccia di finestra astratta fornita da Java per proteggere le applicazioni da piattaforme di finestre specifiche. Swing è una libreria di oggetti dell'interfaccia utente che offre una vasta gamma di servizi, dai pulsanti alla gestione della geometria. Un'applicazione di solito accede solo all'interfaccia Swing. Tuttavia, il livello Applicazione può ignorare il livello Swing e accedere direttamente ad AWT.

```
In generale, l'apertura dell'architettura consente agli sviluppatori di bypassare i livelli superiori per affrontare i colli di bottiglia delle prestazioni.
Vantaggi:
1. Efficienza a runtime:
    - Ogni livello può utilizzare servizi di un altro livello bypassando livelli intermedi.
```

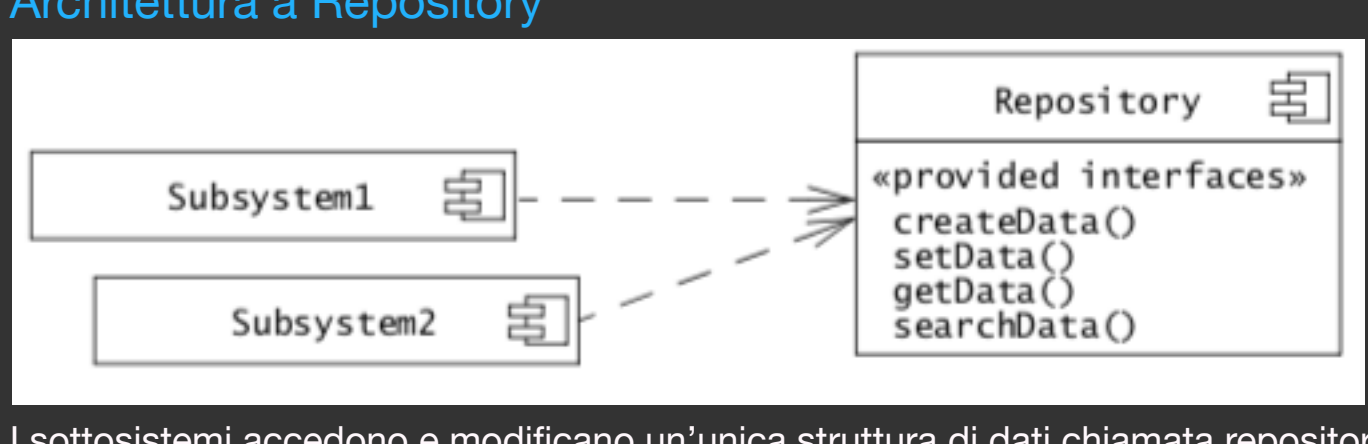


Un altro approccio per trattare con la complessità consiste nel partizionare il sistema in sottosistemi peer(pari) fra loro, ognuno responsabile di differenti classi di servizi. Ad esempio, un sistema di bordo per un'auto potrebbe essere scomposto in un servizio di viaggio che fonirsce indicazioni in tempo reale al conducente, un servizio di preferenze individuali che ricorda la posizione del sedile del conducente,ecc.. Ogni sottosistema dipende vagamente dagli altri, ma spesso può operare in modo isolato.
In generale una decomposizione in sottosistemi è il risultato di un'attività di partitioning e di layering. Prima si suddivide il sistema in sottosistemi al top-level che sono responsabili di specifiche funzionalità (partitioning). Poi, ogni sottosistema è organizzato in diversi layer, se il livello di complessità lo richiede, finché non sono semplici abbastanza da poter essere implementate da un singolo sviluppatore (layering).

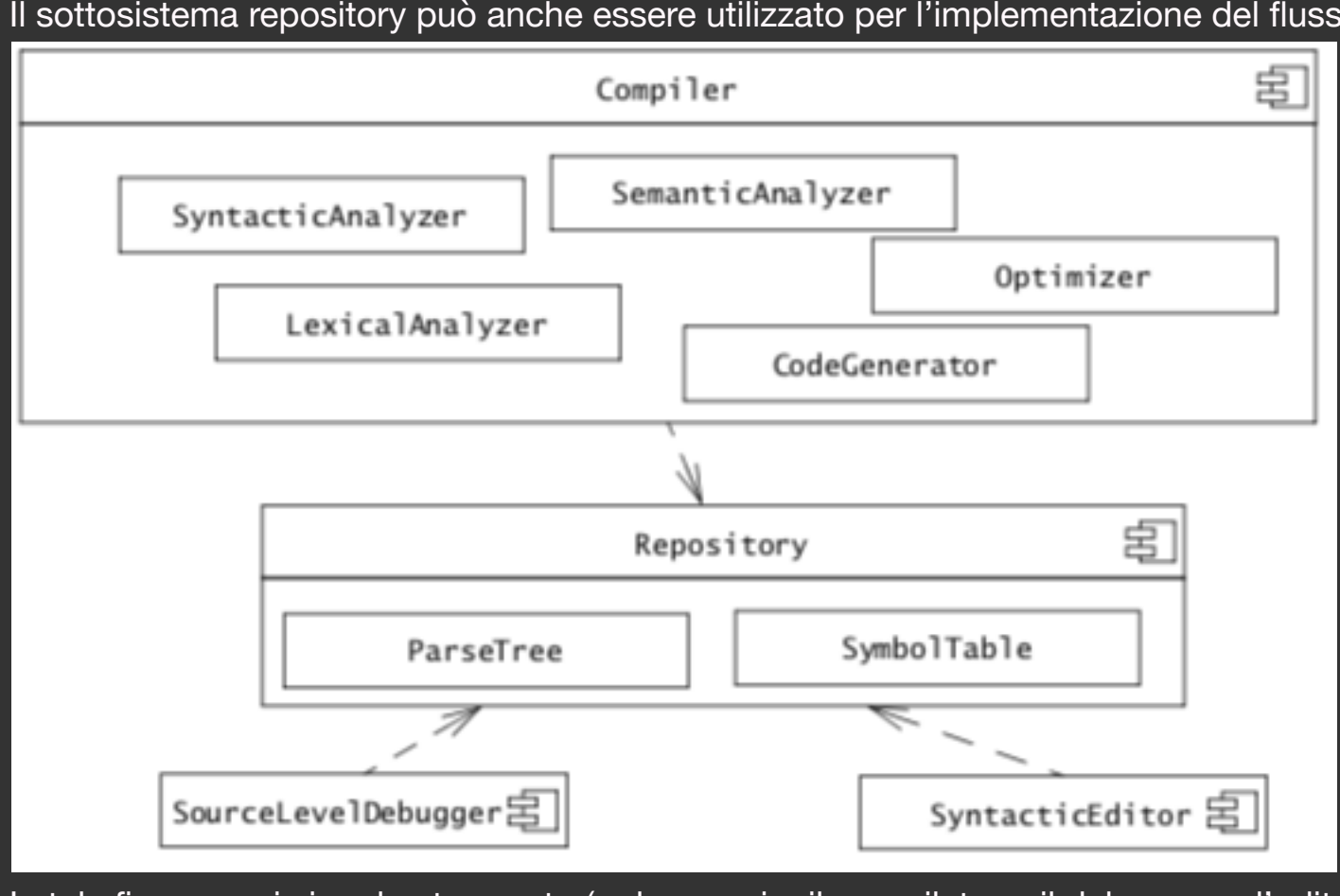
Stili architettonici

Con l'aumentare della complessità dei sistemi, decomporli tali sistemi diventa sempre più difficile quindi è emerso il concetto di architettura software. Un'architettura software include la decomposizione del sistema, il flusso di controllo globale, la gestione delle boundary condition e i protocolli di comunicazioni tra i sottosistemi.

Architettura a Repository



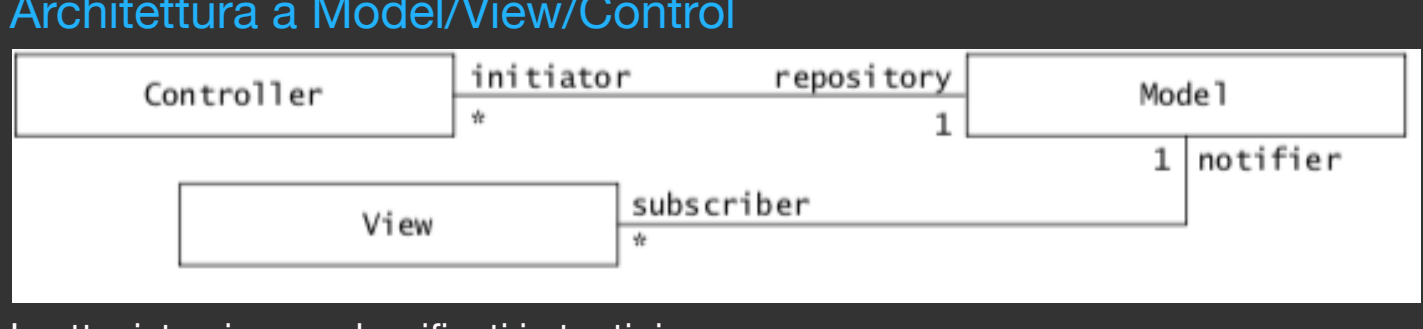
I sottosistemi accedono e modificano un'unica struttura di dati chiamata repository centrale. I sottosistemi sono indipendenti e interagiscono solo attraverso il repository (loosely coupled) . Il repository non ha conoscenze degli altri sottosistemi. I repository sono in genere utilizzati per i sistemi di gestione dei database, come un sistema di gestione stipendi o un sistema bancario; la posizione centrale dei dati semplifica la gestione dei problemi di concorrenza e integrità tra i sottosistemi. Anche compilatori e gli ambienti di sviluppo software; i diversi sottosistemi di un compilatore accedono e aggiornano un albero di analisi centrale e una tabella dei simboli.
Il sottosistema repository può anche essere utilizzato per l'implementazione del flusso di controllo globale.



In tale figura ogni singolo strumento (ad esempio, il compilatore, il debugger e l'editor) viene invocato dall'utente. Il repositoru garantisce solo che gli accessi simultanei siano serializzati.
Oppure il repository può essere utilizzato per invocare i sottosistemi in base allo stato della strutta dati centrali (chiamati sistemi di lavagna).

```
I repository sono adatti per applicazioni con complesse attività di elaborazioni dati in costante evoluzione.
Il principale svantaggio dei sistemi di repository è che il repository centrale può rapidamente diventare un collo di bottiglia. L'accoppiamento tra ciascun sottosistema e il repository è elevato, rendendo difficile la modifica del repository senza influire su tutti i sottosistemi.
```

Architettura a Model/View/Control



I sottosistemi sono classificati in tre tipi:

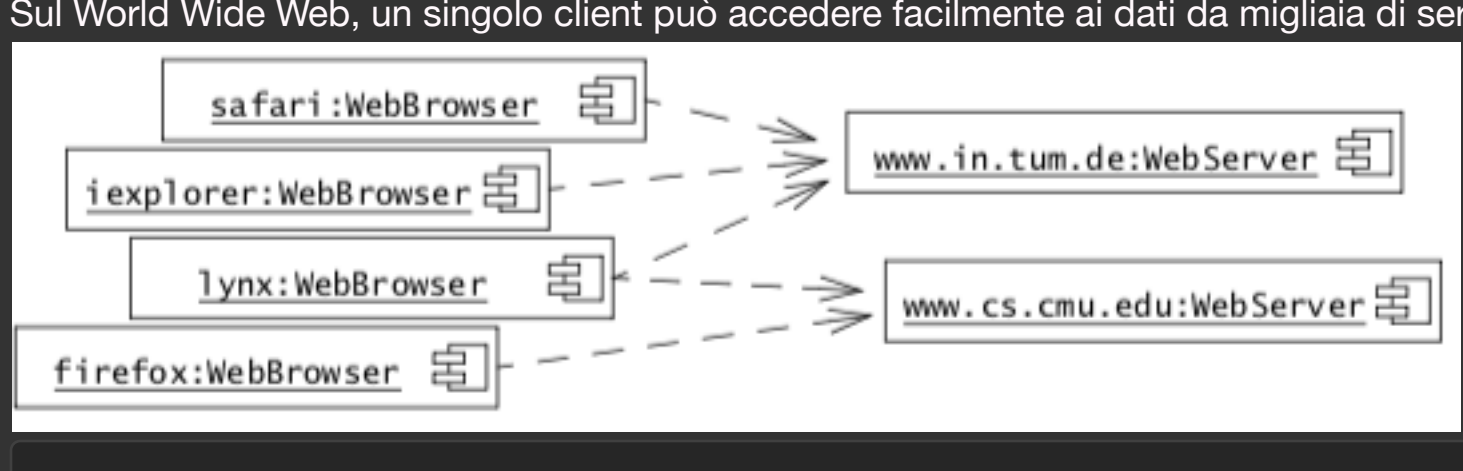
1. i sottosistemi Model mantengono la conoscenza del dominio. Sono sviluppati in modo da non dipendere da alcuni sottosistemi View e Control.
2. I sottosistemi View mostrano i dati all'utente
3. i sottosistemi Control gestiscono la sequenza di interazioni con l'utente.

MVC è un caso speciale del repository in cui Model implementa la struttura dati centrale e gli oggetti Control determinano il flusso di controllo. La logica tra la spedizione di Model, View e Control è che le interfacce utente, ovvero View e il Control, sono molto più spesso soggette a modifiche rispetto alla conoscenza del dominio, ovvero il Model. Inoltre, rimuovendo qualsiasi dipendenza dal modello dalla vista con il protocollo di sottoscrizione/notifica, le modifiche alle viste non hanno alcun effetto sui sottosistemi del modello.

```
Questo modello è adatto per sistemi interattivi, tuttavia introduce lo stesso collo di bottiglia come "repository".
```

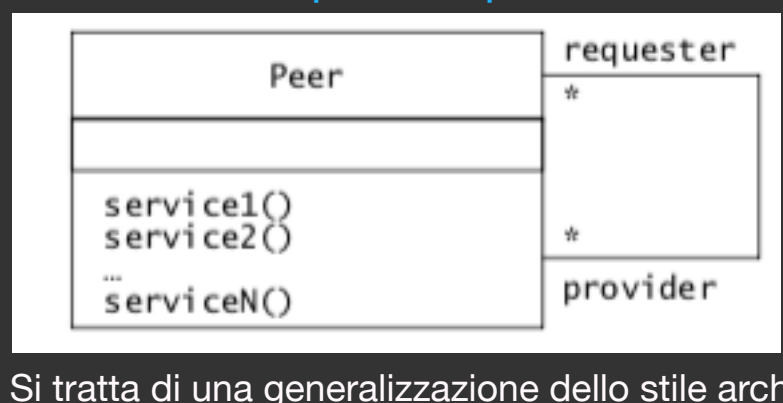
Architettura a Client/Server

Il sottosistema Server, fornisce servizi alle istanze di altri sottosistemi chiamati client che sono responsabili dell'interazione con l'utente. La richiesta di un servizio viene in genere effettuata tramite un meccanismo di chiamata di procedura remota o un broker di oggetti comune. Il flusso di controllo nei client e nei server è indipendente, ad eccezione della sincronizzazione per la gestione delle richieste o la ricezione dei risultati.
Un sistema informativo con un database centrale è un esempio di uno stile architettonico client / server. I clienti sono responsabili della ricezione di input da parte dell'utente, dell'esecuzione dei controlli di intervallo e dell'avvio delle transazioni del database quando vengono raccolti tutti i dati necessari. Il server è quindi responsabile dell'esecuzione della transazione e della garanzia dell'integrità dei dati.
Sul World Wide Web, un singolo client può accedere facilmente ai dati da migliaia di server diversi.

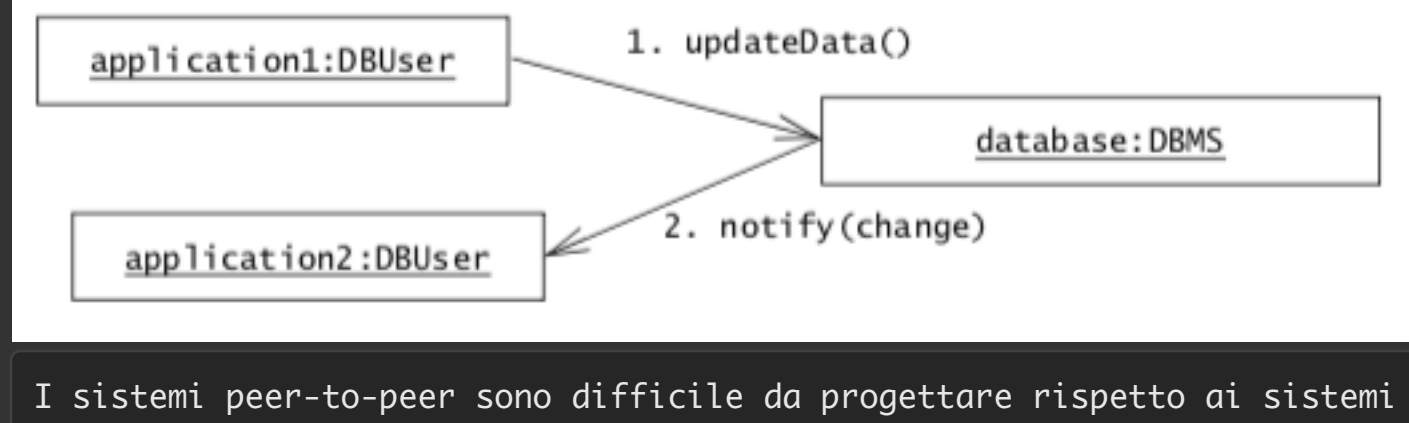



```
Sono adatti per sistemi distribuiti che gestiscono grandi dimensioni quantità di dati
```

Architettura a peer-to-peer

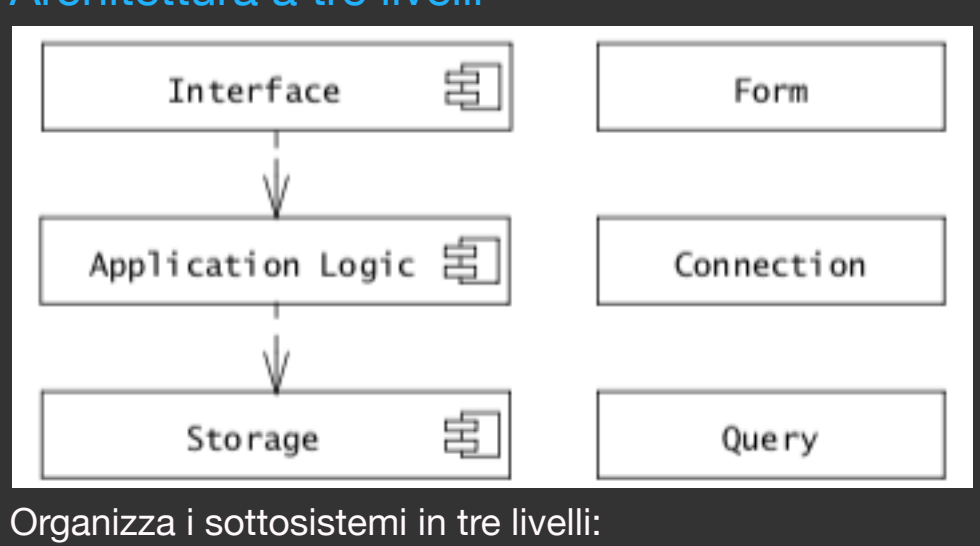


Si tratta di una generalizzazione dello stile architettonico client/server in cui i sottosistemi possono agire sia come client che come server. Il flusso di controllo all'interno di ciascun sottosistema è indipendente dagli altri ad eccezione delle sincronizzazione su richiesta.
Un esempio di tale architettura è un database che accetta richieste dall'applicazione e notifica all'applicazione ogni volta che alcuni dati vengono modificati.




```
I sistemi peer-to-peer sono difficile da progettare rispetto ai sistemi client-server perché introducono la possibilità di deadlock e complicano il flusso di controllo.
```

Architettura a tre livelli



Organizza i sottosistemi in tre livelli:

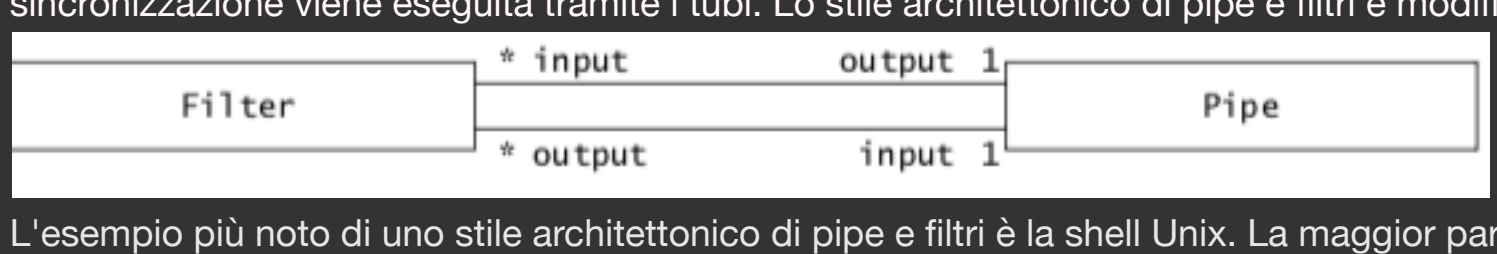
1. il livello Interface include tutti gli oggetti Boundary che si occupano dell'utente, incluso finestre, moduli, pagine Web e cosa via.
2. Il livello Application Logic include tutti gli oggetti di controllo e entità, realizzando l'elaborazione, il controllo delle regole e le notifiche richieste dall'applicazione.
3. Il livello Storage realizza l'archiviazione, il recupero e la query di oggetti persistenti.

Lo stile architettonico a tre livelli fu inizialmente descritto negli anni '70 per i sistemi di informazione. Il livello Storage, analogo al sottosistema Repository nello stile architettonico del repository, può essere condiviso da diverse applicazioni che operano sugli stessi dati.

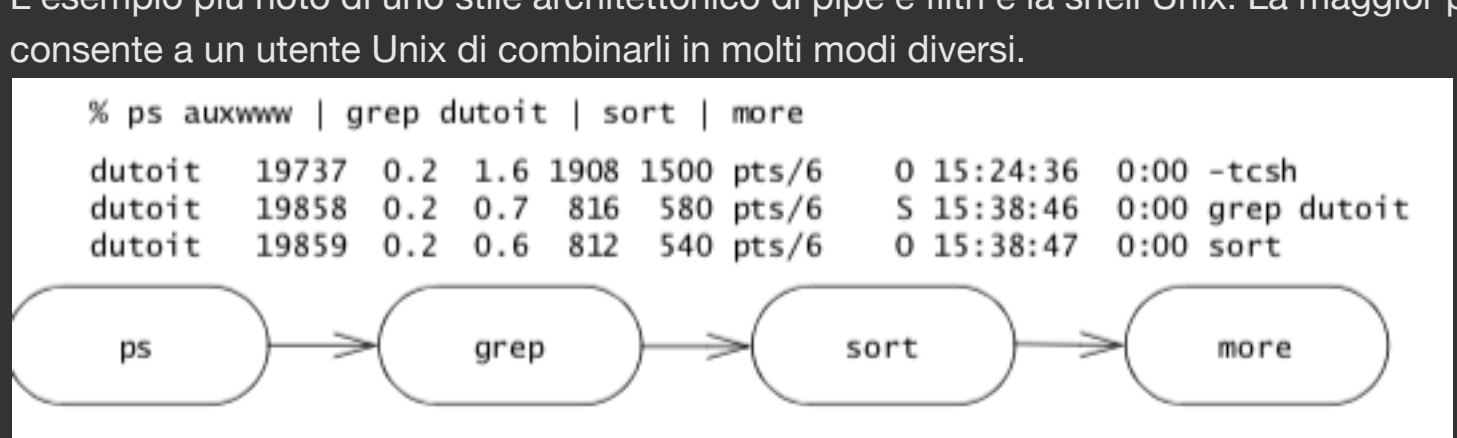
```
A sua volta, la separazione tra il livello dell'interfaccia e il livello della logica dell'applicazione consente lo sviluppo o la modifica di diverse interfacce utente per la stessa logica dell'applicazione.
```

Architettura a Pipe e Filter

I sottosistemi elaborano i dati ricevuti da una serie di input e inviano i risultati ad altri sottosistemi tramite una serie di output. I sottosistemi sono chiamati "Filter" e le associazioni tra i sottosistemi sono chiamate "Pipe". Ogni filtro conosce solo il contenuto e il formato dei dati ricevuti sulle pipe di input, non i filtri che li hanno prodotti. Ogni filtro viene eseguito contemporaneamente e la sincronizzazione viene eseguita tramite i tubi. Lo stile architettonico di pipe e filtri è modificabile: i filtri possono essere sostituiti con altri o riconfigurati per raggiungere uno scopo diverso.



L'esempio più noto di uno stile architettonico di pipe e filtri è la shell Unix. La maggior parte dei filtri sono scritti in modo tale da leggere i loro input e scrivere i loro risultati su pipe standard. Ciò consente a un utente Unix di combinarli in molti modi diversi.



L'output di ps (stato del processo) viene inserito in grep (ricerca di un modello) per rimuovere tutti i processi che non sono di proprietà di un utente specifico. L'output di grep (ovvero i processi di proprietà dell'utente) viene quindi ordinato per ordinamento e inviato a more, che è un filtro che visualizza il suo input su un terminale, uno schermo alla volta.
Gli stili di pipe e filtri sono adatti per sistemi che applicano trasformazioni a flussi di dati senza intervento da parte degli utenti.

```
Non sono adatti per sistemi che richiedono interazioni più complesse tra componenti, come un sistema di gestione delle informazioni o un sistema interattivo.
```

Identificazione dei sottosistemi

La ricerca di sottosistemi durante la progettazione del sistema è simile alla ricerca di oggetti durante l'analisi. La decomposizione del sottosistema viene rivista ogni volta che vengono affrontati nuovi problemi. Le prime iterazioni sulla decomposizione possono introdurre cambiamenti drastici nel modello di progettazione del sistema. Questi sono gestiti meglio attraverso il "brainstorming".

Euristica per il raggruppamento di oggetti in sottosistemi

- Assegnare oggetti identificati in un caso d'uso nello stesso sottosistema;
- Creare un sottosistema dedicato per gli oggetti utilizzati per spostare i dati tra i sottosistemi;
- Ridurre al minimo il numero di associazioni che attraversano i confini del sottosistema;
- Tutti gli oggetti nello stesso sottosistema devono essere funzionalmente correlati.
