Luca Cabibbo

Architettura dei Sistemi Software

Pattern architetturale Pipes and Filters

dispensa asw340

marzo 2021

Sam had a strange feeling
as the slow gurgling stream slipped by:
his old life lay behind in the mists,
dark adventure lay in front.

J.R.R. Tolkien

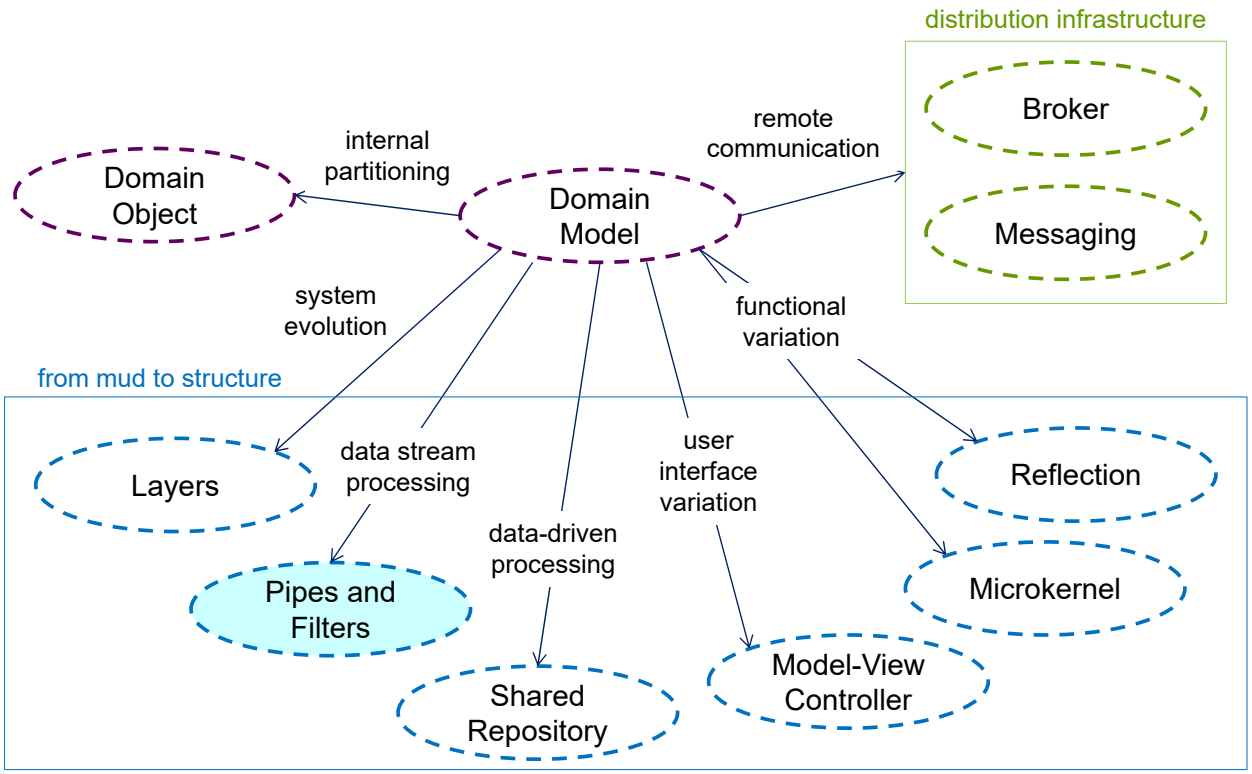
- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 18, Pattern architetturale Pipes and Filters
- [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.

- Obiettivi e argomenti

- Obiettivi
  - presentare il pattern architetturale Pipes and Filters
- Argomenti
  - Pipes and Filters (POSA)
  - discussione

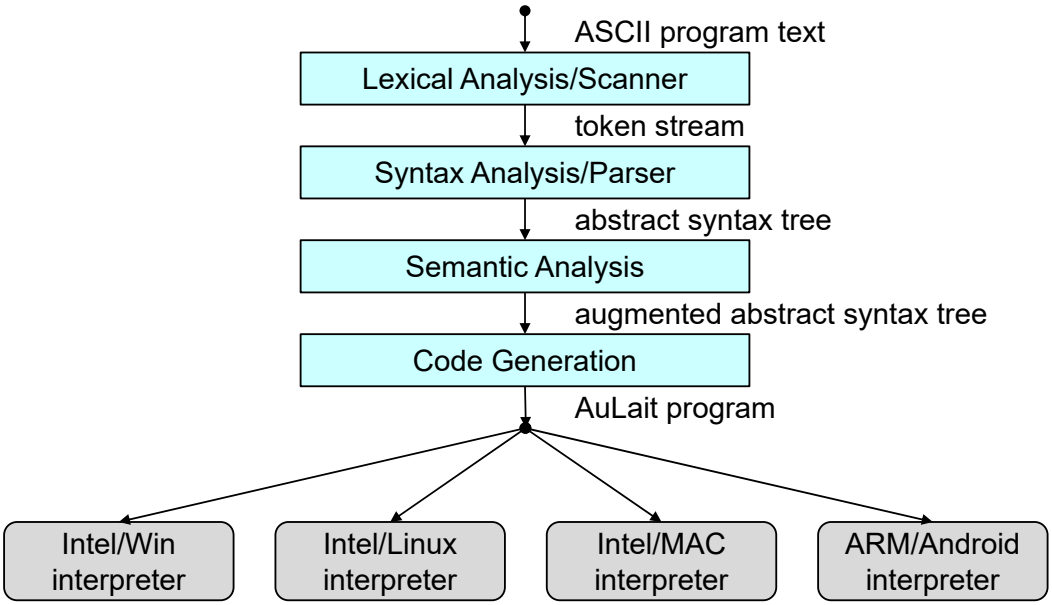
* Pipes and Filters [POSA]



Pipes and Filters

- Il pattern architetturale **Pipes and Filters** – chiamato anche *architettura a pipeline*
  - nella categoria “data stream processing” di [POSA4]
  - aiuta a strutturare sistemi (o componenti) che devono effettuare l’elaborazione di flussi di dati – ovvero, che devono trasformare un flusso di dati in ingresso in un flusso di dati in uscita
  - in corrispondenza, si parla di applicazioni guidate da flussi di dati (data-flow-driven application)

Esempio



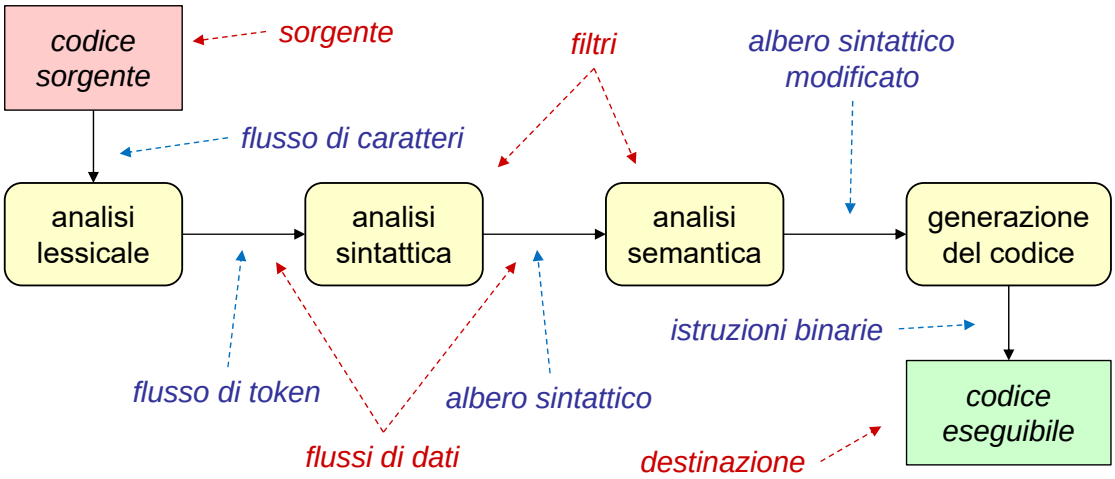
Pipes and Filters

- Contesto
  - un sistema (o componente) deve elaborare flussi di dati
- Problema
  - l'applicazione deve elaborare flussi di dati – effettuandone una trasformazione
  - una decomposizione basata su meccanismi di tipo richiesta-risposta non è adeguata
  - la trasformazione può essere specificata come una sequenza di passi di elaborazione – per essere effettuata in modo incrementale
  - va sostenuta la modificabilità della trasformazione
  - vanno evitate penalizzazioni nelle prestazioni

Pipes and Filters

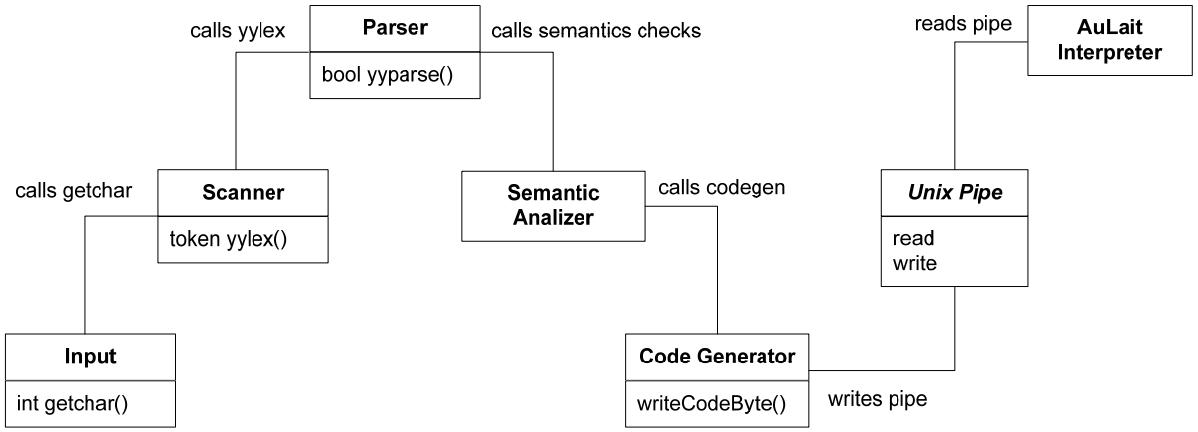
- Soluzione
  - suddividi la trasformazione in una sequenza di passi di elaborazione dei dati auto-contenuti
  - implementa ciascun passo di elaborazione con un "filtro"
    - ogni *filtro* (*filter*) è un componente che effettua un'elaborazione (di solito semplice) per operare una trasformazione (parziale e incrementale) del flusso di dati complessivo – un nome migliore è "processor"
  - definisci la trasformazione complessiva collegando i filtri in una *pipeline* di elaborazione dei dati
  - nella pipeline, collega i filtri successivi mediante delle "pipe"
    - ogni *pipe* è un connettore che è buffer di dati intermedio che collega una coppia di filtri

Esempio



Esempio

- Il compilatore può usare
  - alcuni filtri standard – come lex e yacc
  - ulteriori filtri specifici – ad es., per la generazione del codice
  - le pipe di Unix

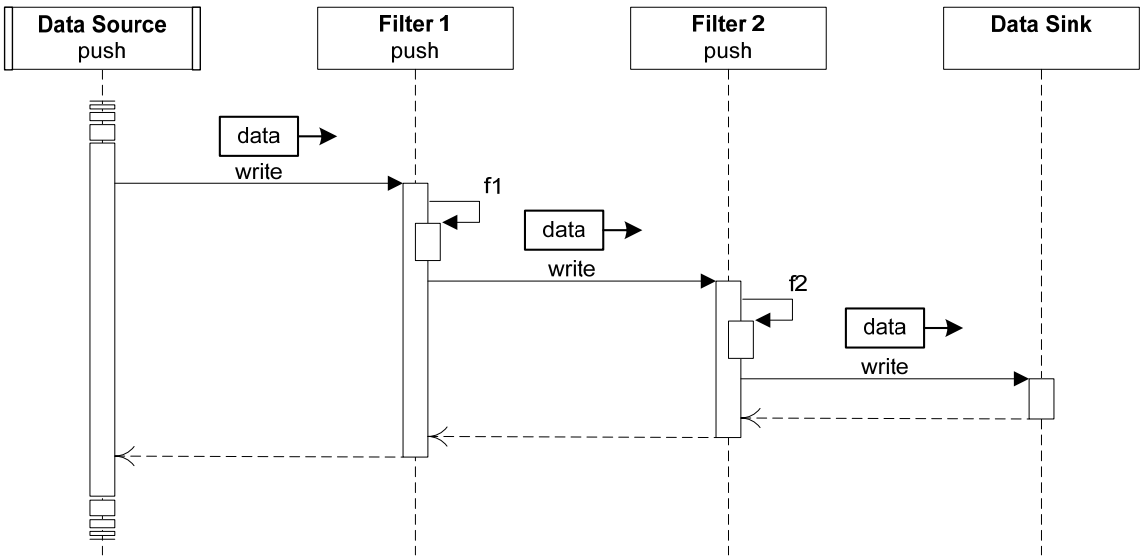


Scenari

- Sono possibili diverse realizzazioni (scenari) di Pipes and Filters
  - il flusso dei dati va sempre nella stessa direzione, dalla sorgente verso la destinazione
  - tuttavia, la gestione del controllo cambia da scenario a scenario
  - in alcuni scenari (1,2,3), c'è un solo elemento attivo
  - nello scenario (4) ci sono più filtri/elementi attivi – che vivono in processi e/o thread diversi

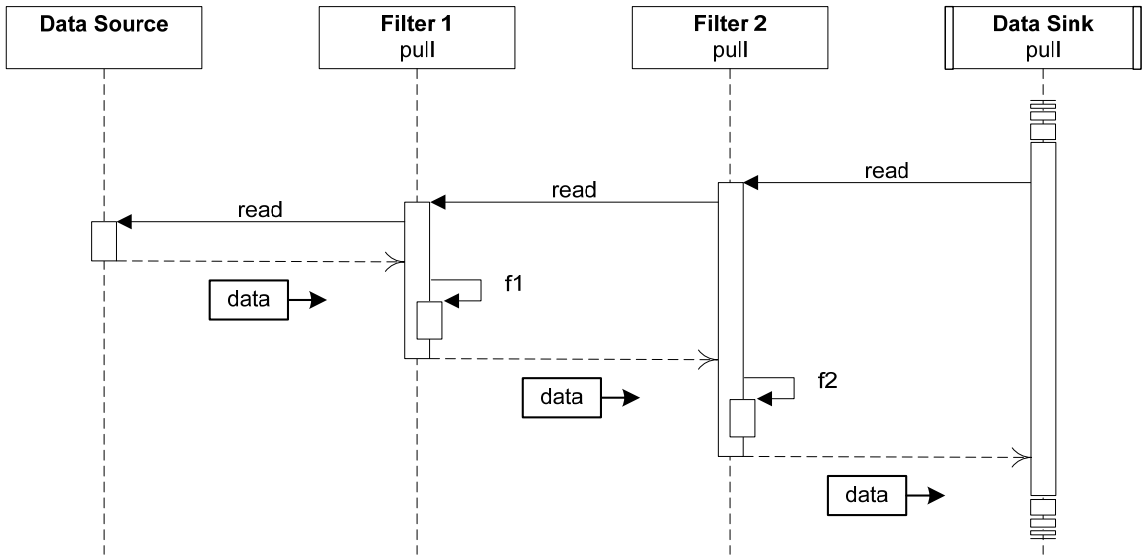
Scenario 1

- *Pipeline di tipo push*



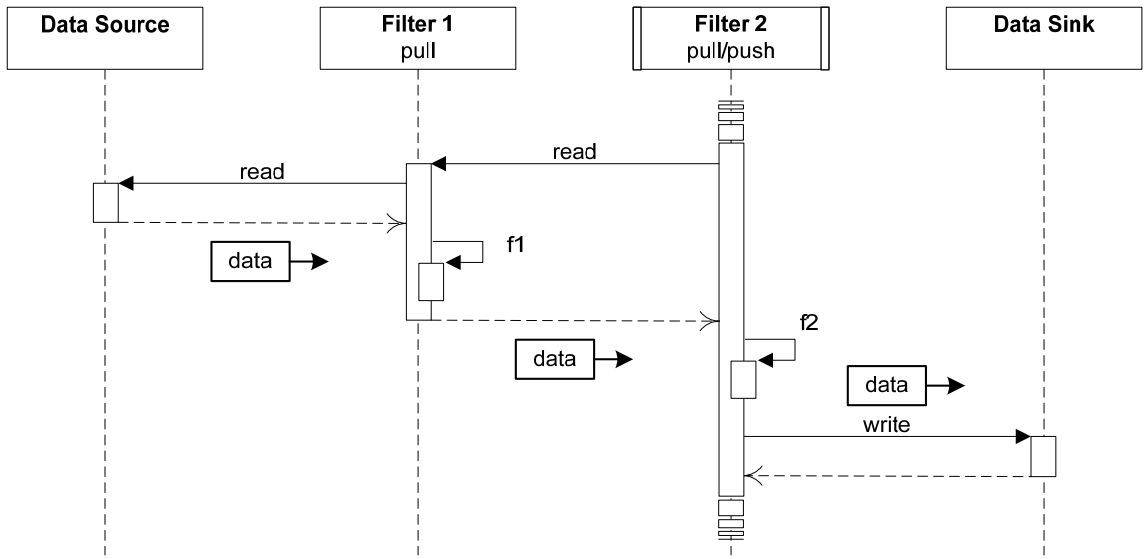
Scenario 2

- *Pipeline di tipo pull*



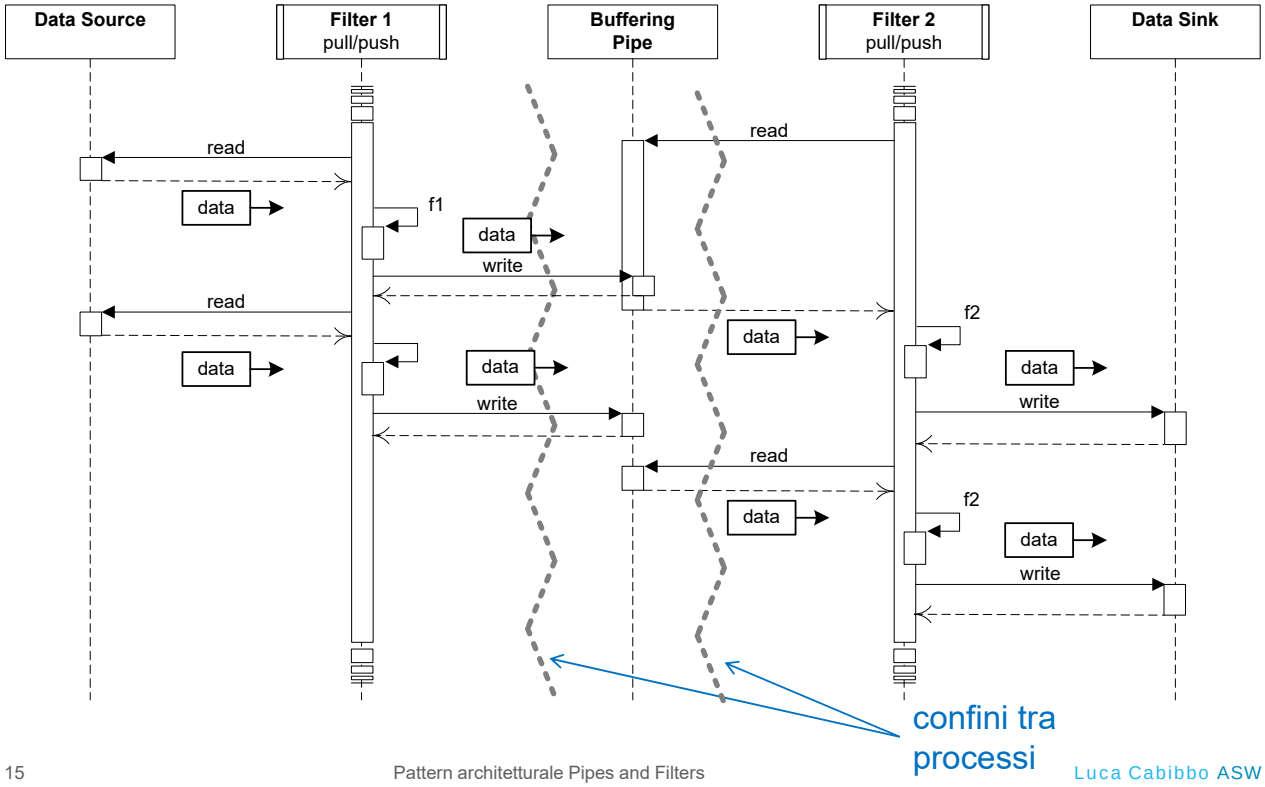
Scenario 3

- Una pipeline mista pull-push



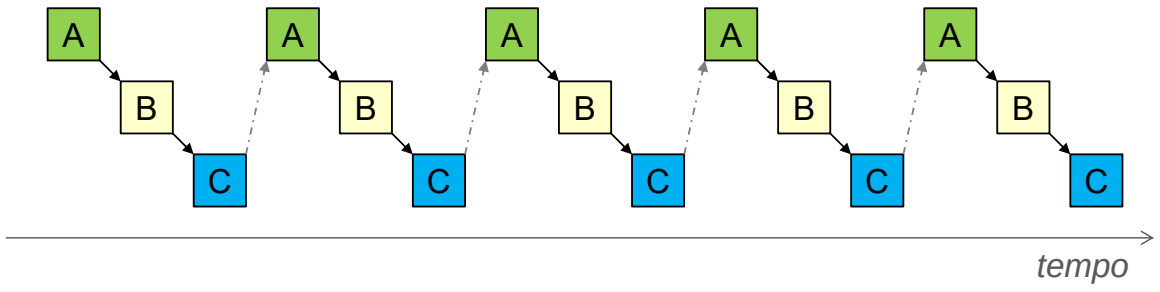
Scenario 4

- Una pipeline con filtri attivi

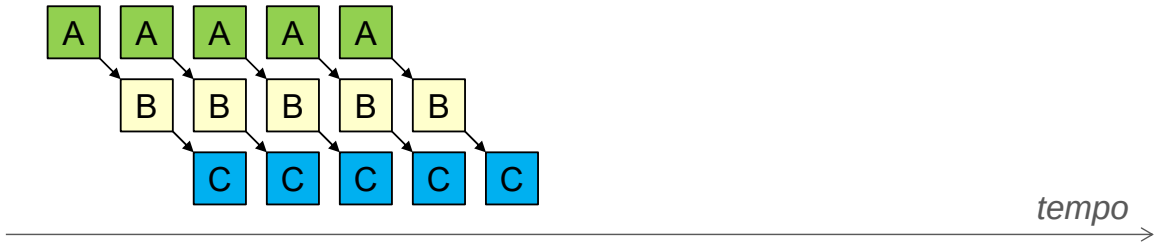


Confronto tra scenari

- pipeline di tipo push



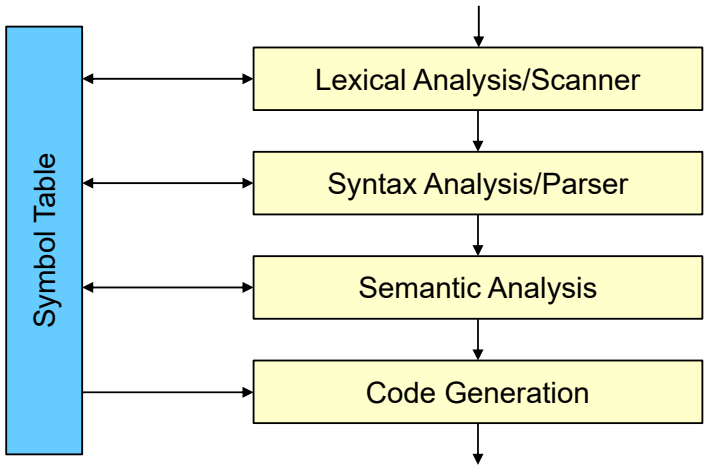
- pipeline con più filtri attivi



Pipes and Filters – Applicazione

- Suddividi il compito da svolgere in un gruppo di passi di elaborazione
- Definisci il formato dei dati scambiati dai filtri
- Decidi come implementare le connessioni pipe
  - quali filtri sono attivi?
  - come sono attivati i filtri passivi?
  - come sono implementate le pipe?
- Progetta e implementa i filtri
- Progetta per la gestione degli errori
- Metti in piedi la pipeline di elaborazione

Esempio

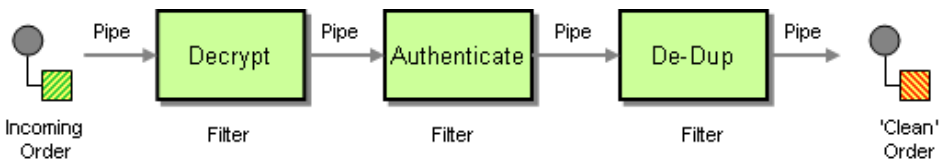


Esempio

- Il pattern Pipes and Filters è usato nello scripting in Unix
  - ad esempio
    - cat document.txt | tr -cs A-Za-z '\n' | tr A-Z a-z | \
      sort | uniq -c | sort -rn | sed 10q
    - determina le 10 parole più frequenti di un file di testo, e visualizza un elenco ordinato di queste parole insieme alla loro frequenza

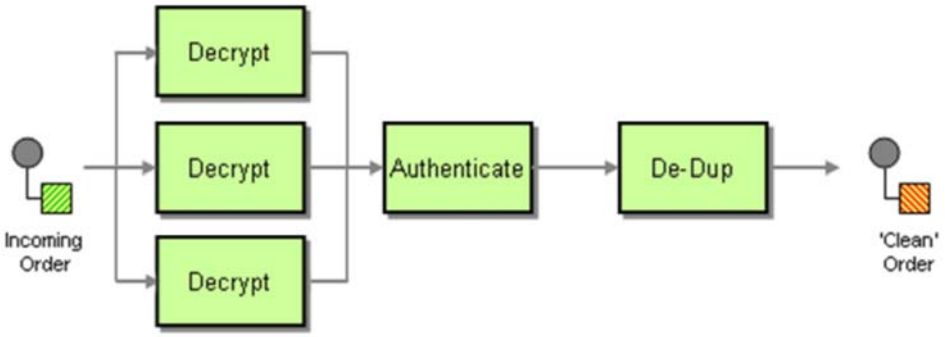
Esempio

- Un sistema riceve ordini sotto forma di messaggi
  - gli ordini sono cifrati e contengono un certificato che garantisce l'identità del cliente
  - è possibile che alcuni messaggi arrivino ripetuti
- Si supponga di voler trasformare questo flusso di messaggi in
  - un flusso di ordini "in chiaro", senza informazioni ridondanti e senza duplicati



Esempio (cont.)

- Si supponga inoltre che l'operazione di decifratura sia molto più lenta di quella di autenticazione
  - vogliamo evitare una penalizzazione delle prestazioni



- Discussione

- Pipes and Filters suggerisce una decomposizione dell'elaborazione in passi di elaborazione distinti – implementati da filtri, collegati in una pipeline
  - sostiene un approccio di elaborazione incrementale – ogni filtro esegue solo una parte dell'elaborazione
  - i filtri possono evolvere in modo indipendente
  - i filtri possono operare in modo concorrente
  - talvolta è possibile replicare filtri

Discussione

- Anche le pipe svolgono un ruolo importante
  - sono connettori – per la comunicazione, il coordinamento e la sincronizzazione tra filtri
  - sostengono un accoppiamento debole tra filtri
  - sono possibili diverse implementazioni – ad es., come code oppure come canali per messaggi

Discussione

- L'applicazione di Pipes and Filters può essere guidata da una modellazione del dominio di tipo data-flow – in termini di flussi di dati e di attività
  - ciascun filtro è un Domain Object che rappresenta un'attività o un compito nel dominio
  - ciascuna pipe implementa un flusso di dati tra filtri
    - una pipe potrebbe essere un Domain Object che rappresenta una movimentazione di dati o una politica di buffering

- Conseguenze

- Modificabilità
  - ☺ sostiene la modificabilità della trasformazione – nella misura in cui la nuova trasformazione può essere realizzata come "ridefinizione" della pipeline
  - ☺ la flessibilità è basata su
    - la possibilità di aggiungere/sostituire/rimuovere filtri – da una libreria di filtri preesistenti – oppure di definire nuovi filtri o di modificare filtri esistenti
    - la presenza e l'uso di pipe
    - la possibilità di riorganizzare la pipeline in diversi momenti della vita del sistema
  - ☹ non sempre la modificabilità è alta

Conseguenze

- Prestazioni
  - ☺ i filtri sono buone unità di concorrenza
  - ☺ è possibile replicare filtri
  - ☺ potrebbe non essere necessario usare file per i risultati intermedi
  - ☺ le necessità di sincronizzazione tra filtri sono limitate
  - ☹ c'è un overhead dovuto al trasferimento continuo dei dati tra i filtri – nonché ai cambiamenti di contesto e del formato dei dati
  - 😐 il guadagno di efficienza potrebbe essere solo un'illusione

Conseguenze

- Affidabilità
  - 😐 difficile fare considerazioni generali
  - ☹ se i dati devono essere elaborati da molti filtri, la verifica è più difficile
  - ☺ può essere facile verificare ogni filtro in isolamento
  - ☺ l'affidabilità può essere sostenuta da un'opportuna infrastruttura di esecuzione

Conseguenze

- Sicurezza
  - ☺ i sistemi basati su Pipes and Filters e i suoi componenti hanno in genere un'interfaccia piccola e ben definita
  - ☺ può essere semplice introdurre meccanismi di sicurezza sull'intero sistema o sui singoli componenti
- Altro
  - ☺ possibilità di riusare componenti filtro
  - ☹ la condivisione di dati tra filtri è difficile
  - ☹ la gestione degli errori è difficoltosa
  - ☹ non è adatto a sistemi interattivi

- Usi conosciuti

- Alcuni usi conosciuti del pattern Pipes and Filters
  - nello scripting in Unix
  - nel server web Apache, nell'elaborazione di richieste
  - in sistemi di calcolo scientifico
  - Pipes and Filters è alla base delle infrastrutture di messaging e dei sistemi di workflow – importanti nelle architetture a servizi

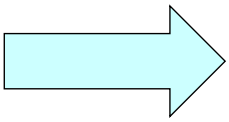
Variante: Tee and join pipeline system

- Il pattern architetturale *Tee and join pipeline system* è una variante di Pipes and Filters
  - i filtri possono avere più ingressi e/o più uscite
    - tee di Unix
  - l'elaborazione è un grafo diretto

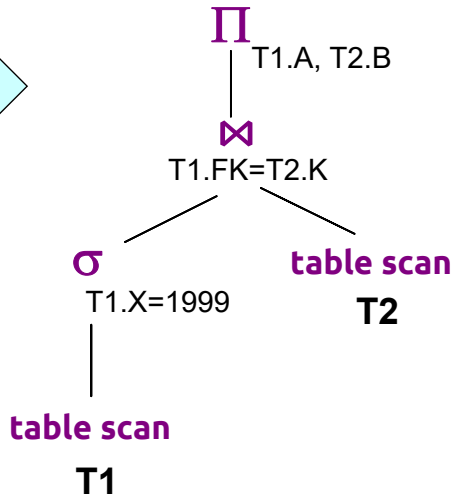
Esempio: esecuzione di query relazionali

SQL Query:

```
SELECT  T1.A, T2.B
FROM    T1, T2
WHERE   T1.FK=T2.K
AND     T1.X=1999;
```



Execution Plan:



Altri usi

- Pipes and Filters è utilizzato in numerosi modelli computazionali per l'elaborazione e l'analisi di dati provenienti da sorgenti di dati grandi e/o diversificate
  - alcuni esempi notevoli
    - MapReduce – a programming model for processing large data sets with a parallel, distributed algorithm on a cluster
    - Logstash – a server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite repository
  - in questi sistemi, le elaborazioni da eseguire sono specificate mediante una sequenza di trasformazioni elementari
  - questi sistemi possono supportare prestazioni, scalabilità e affidabilità

* Discussione

- Pipes and Filters è un altro pattern architetturale fondamentale
  - guida la decomposizione di sistemi (o di componenti di sistemi) che devono elaborare flussi di dati
  - Pipes and Filters, 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