Luca Cabibbo

Architettura dei Sistemi Software

Vedremo che l'architettura a micro servizi forse è più legata a questo pattern architetturale che non all'architettura a strati

Pattern architetturale Pipes and Filters

dispensa asw340

ottobre 2024

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

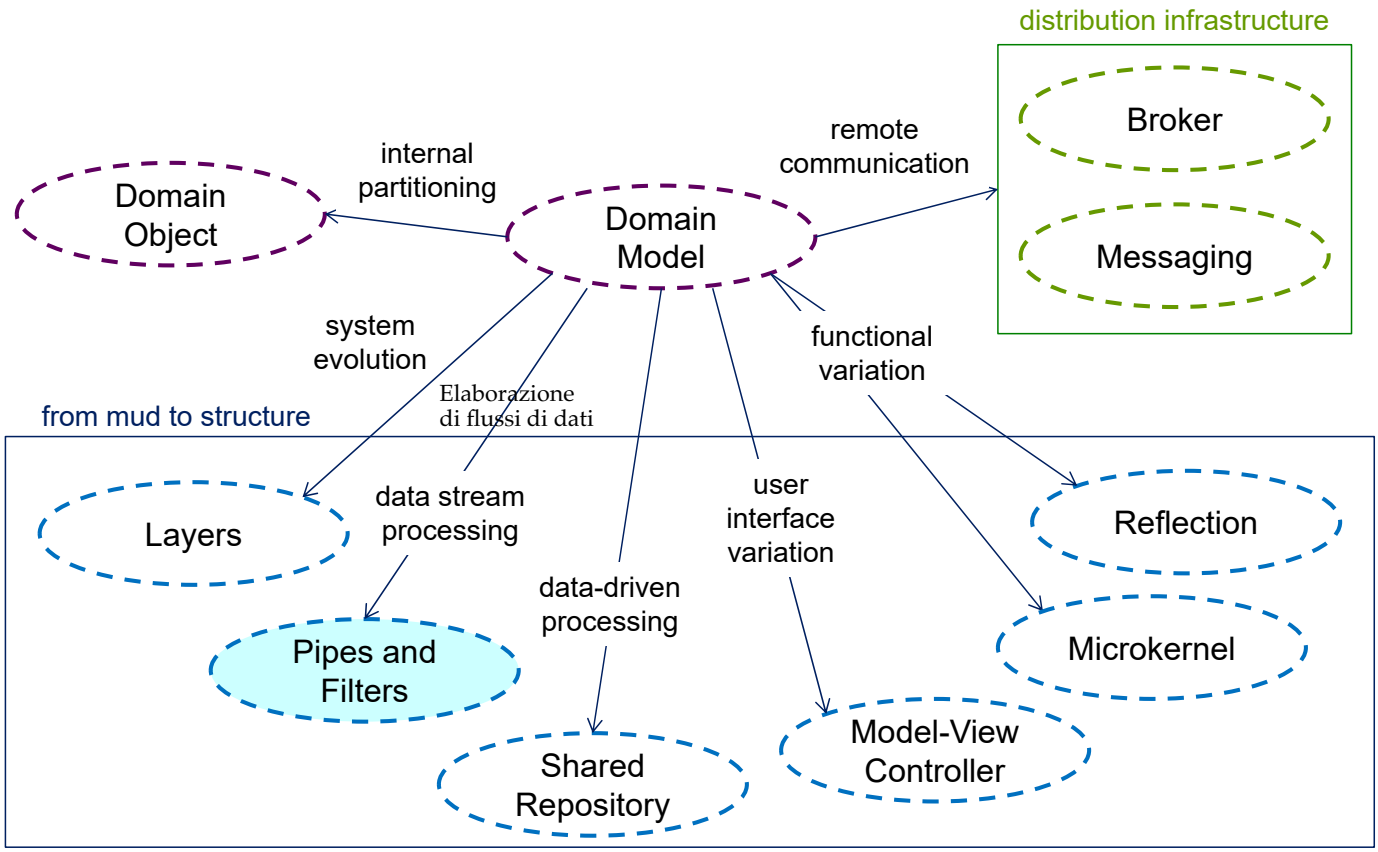
- Riferimenti

- 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

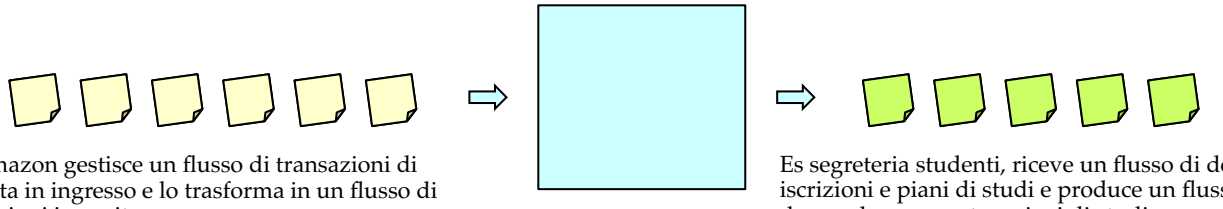
* Pipes and Filters [POSA]



Pipes and Filters

Le pipe sono connettori, tubi letteralmente, i filtri sono componenti

- 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



Es Amazon gestisce un flusso di transazioni di vendita in ingresso e lo trasforma in un flusso di spedizioni in uscita

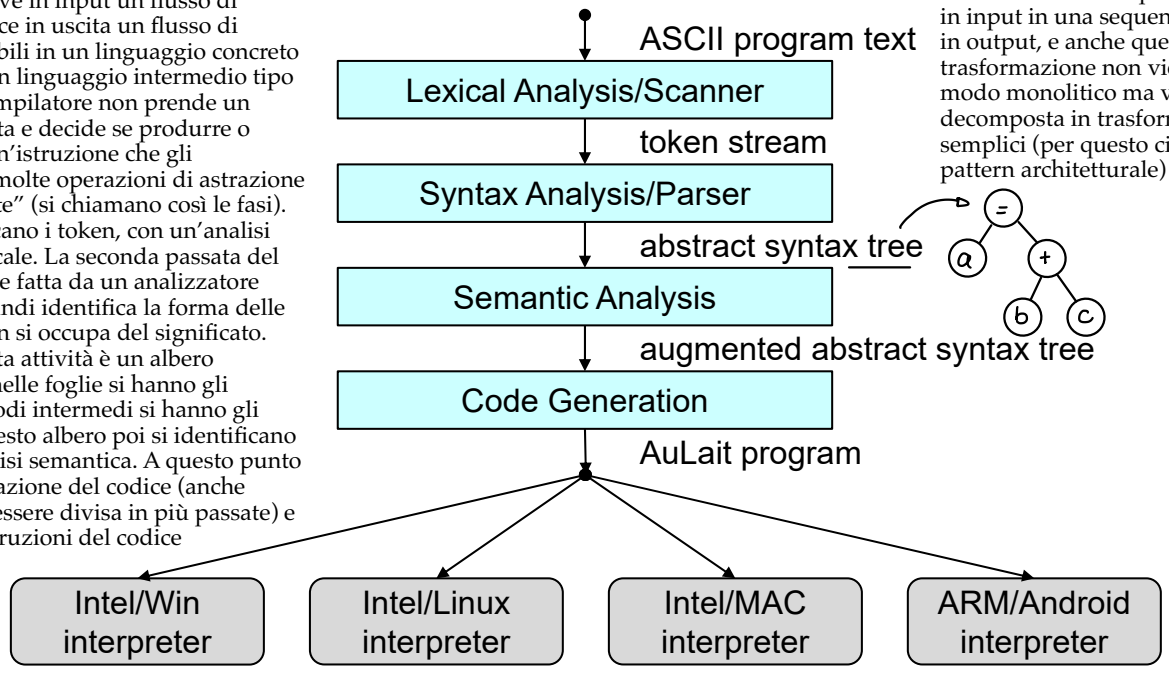
Es segreteria studenti, riceve un flusso di domande di iscrizioni e piani di studi e produce un flusso di domande approvate e piani di studi approvati (o meno)

  - elaborazioni di questo tipo sono molto comuni e rilevanti
  - in corrispondenza, si parla di applicazioni guidate da flussi di dati (data-flow-driven application)

Esempio

- Un compilatore portabile per un nuovo linguaggio di programmazione (Mocha) – basato su un linguaggio intermedio (AuLait)

Compilatore riceve in input un flusso di caratteri e produce in uscita un flusso di istruzioni eseguibili in un linguaggio concreto x86 o x64 ecc o un linguaggio intermedio tipo pipecode. Un compilatore non prende un carattere alla volta e decide se produrre o meno in uscita un'istruzione che gli corrisponde. Fa molte operazioni di astrazione e fa varie "passate" (si chiamano così le fasi). Prima si identificano i token, con un'analisi quindi solo lessicale. La seconda passata del compilatore viene fatta da un analizzatore sintattico che quindi identifica la forma delle frasi anche se non si occupa del significato. L'output di questa attività è un albero sintattico in cui nelle foglie si hanno gli operandi e nei nodi intermedi si hanno gli operatori. Da questo albero poi si identificano i tipi con un'analisi semantica. A questo punto avviene la generazione del codice (anche questa fase può essere divisa in più passate) e si generano le istruzioni del codice



Quindi anche qui si tratta di trasformare una sequenza di caratteri in input in una sequenza di istruzioni in output, e anche questa trasformazione non viene fatta in modo monolitico ma viene decomposta in trasformazioni più semplici (per questo ci interessa il pattern architetturale)

Pipes and Filters

- Contesto
  - un sistema (o componente) deve elaborare flussi di dati
- Problema

Nota ancora che il problema NON è espresso in termini della soluzione!

  - l'applicazione deve elaborare flussi di dati – per effettuare una trasformazione da un flusso di dati d'ingresso a un flusso di dati in uscita
  - una decomposizione basata su meccanismi di tipo richiesta-risposta non è adeguata

Nel caso del compilatore non si può agire sulla singola "richiesta", cioè sul singolo elemento del flusso di input, in questo caso non si può elaborare un output su ogni singolo carattere in input

C'è questa possibilità

  - la trasformazione può essere specificata come una sequenza di passi di elaborazione – per essere effettuata in modo incrementale

Cioè svolta per passaggi successivi che potrebbero essere svolti dallo stesso ente o da enti diversi

E si tiene in conto che possono esserci modifiche di alcuni passaggi della trasformazione

  - 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

Perché è vero per ipotesi, vedi slide precedente

  - implementa ciascun passo di elaborazione con un "filtro"
    - un *filtro* (*filter*) è un componente che effettua un'elaborazione (di solito semplice) del proprio flusso di dati – un nome migliore è "processor"

Perché fa un'elaborazione parziale della trasformazione, non è un filtro che fa passare alcune cose e altre no

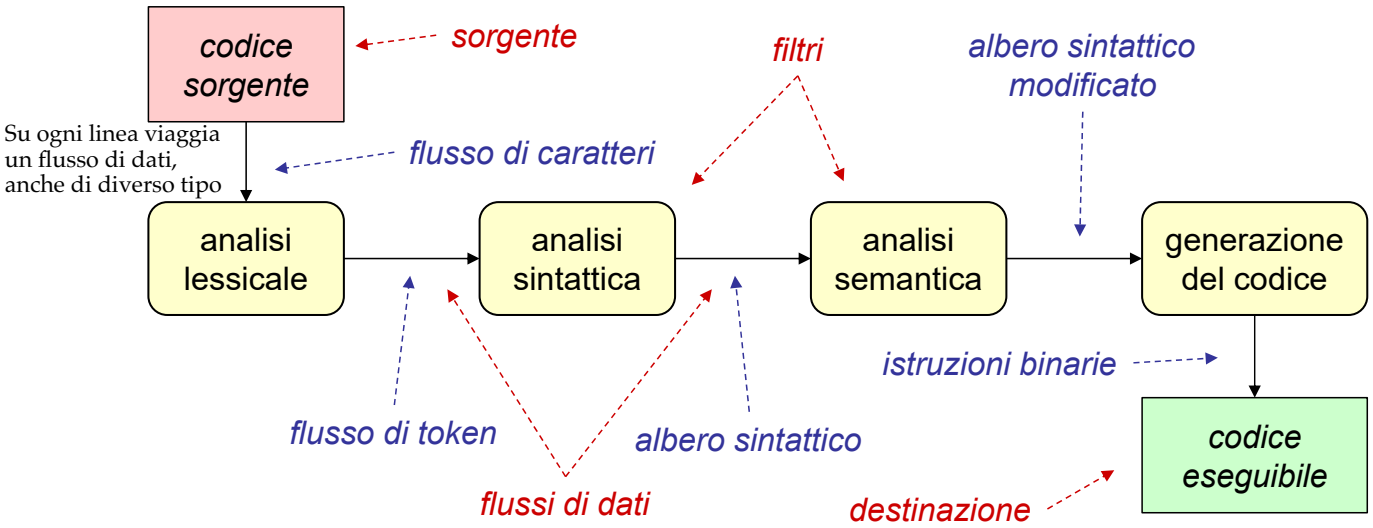
      - ha obiettivo di operare una trasformazione (parziale e incrementale) del flusso di dati complessivo

Si occupa solo di aspetti funzionali

  - definisci la trasformazione complessiva collegando i filtri, in sequenza, in una *pipeline* di elaborazione dei dati
  - nella pipeline, collega i filtri successivi mediante delle "pipe"
    - una *pipe* è un connettore che è un buffer di dati intermedio che collega una coppia di filtri

La trasformazione è una sequenza di passi, i passi sono rappresentati dai filtri, i filtri sono collegati in una singola pipeline TRAMITE delle pipe che si posizionano tra due filtri successivi. Ogni pipe è un connettore che non esegue trasformazioni, è un buffer di dati

Esempio

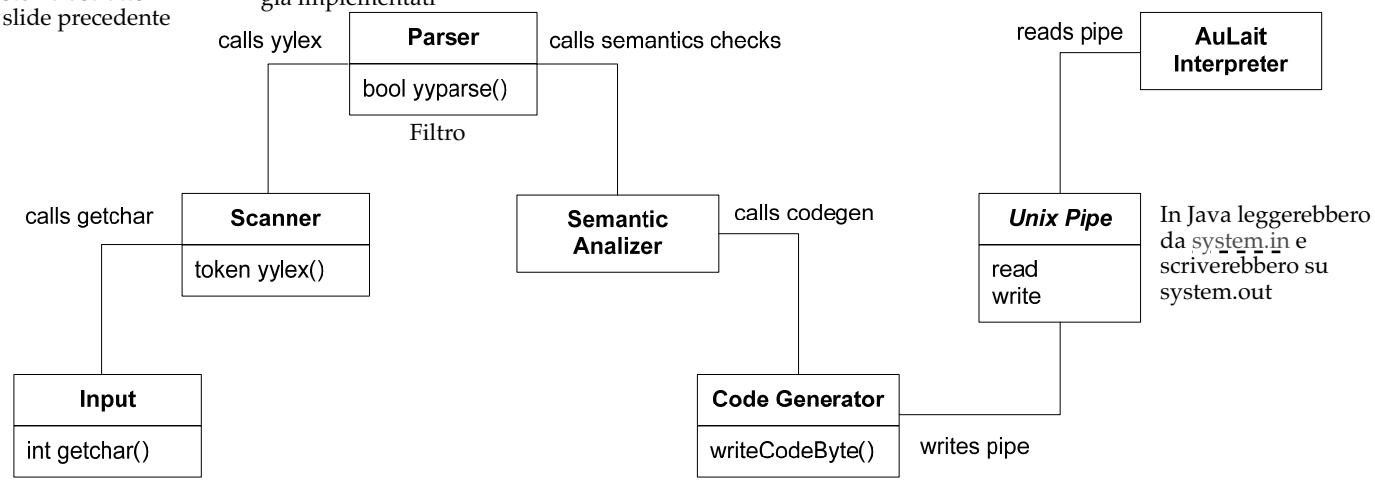


Esempio

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

Realizzazione concreta del sistema astratto nella slide precedente

Nota che un aspetto fondamentale di pipe and filters è quello di RIUTILIZZARE filtri già implementati



Scenari

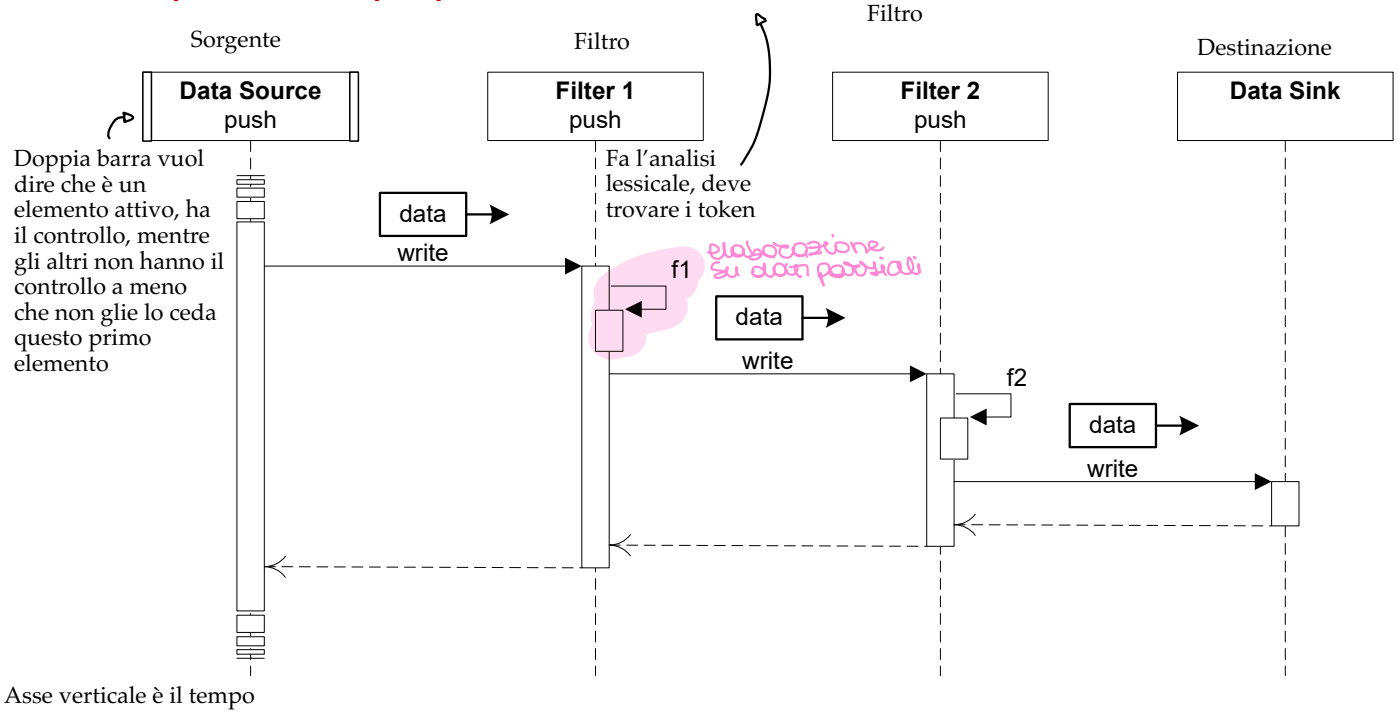
Gli aspetti dinamici sono descritti da quattro scenari

- 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 – in processi e/o thread diversi

Scenario 1

Se ad esempio ricevesse p u b l i c decide cosa ha ricevuto quando riceve il carattere successivo alla c. Se riceve spazio capisce che ha ricevuto un token intero che è una parola chiave e la passa all'analizzatore sintattico

- *Pipeline di tipo push*

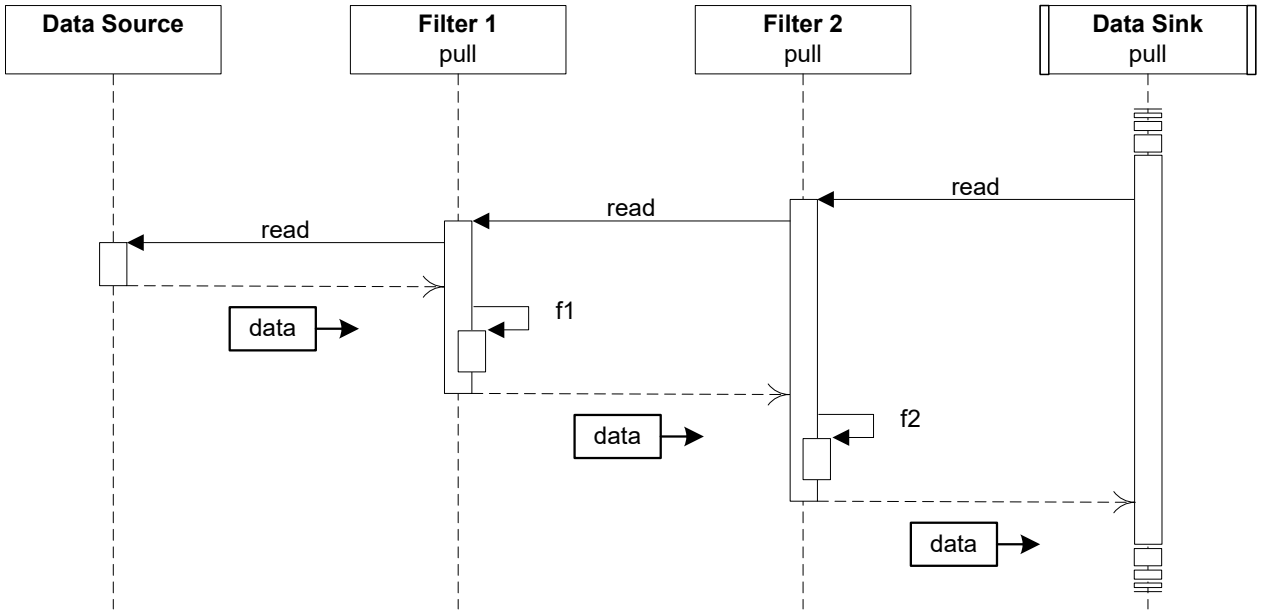


Scenario 2

- *Pipeline di tipo pull*

I dati si muovono sempre da sinistra a destra (da source a destination) ma il controllo si muove in modo diverso

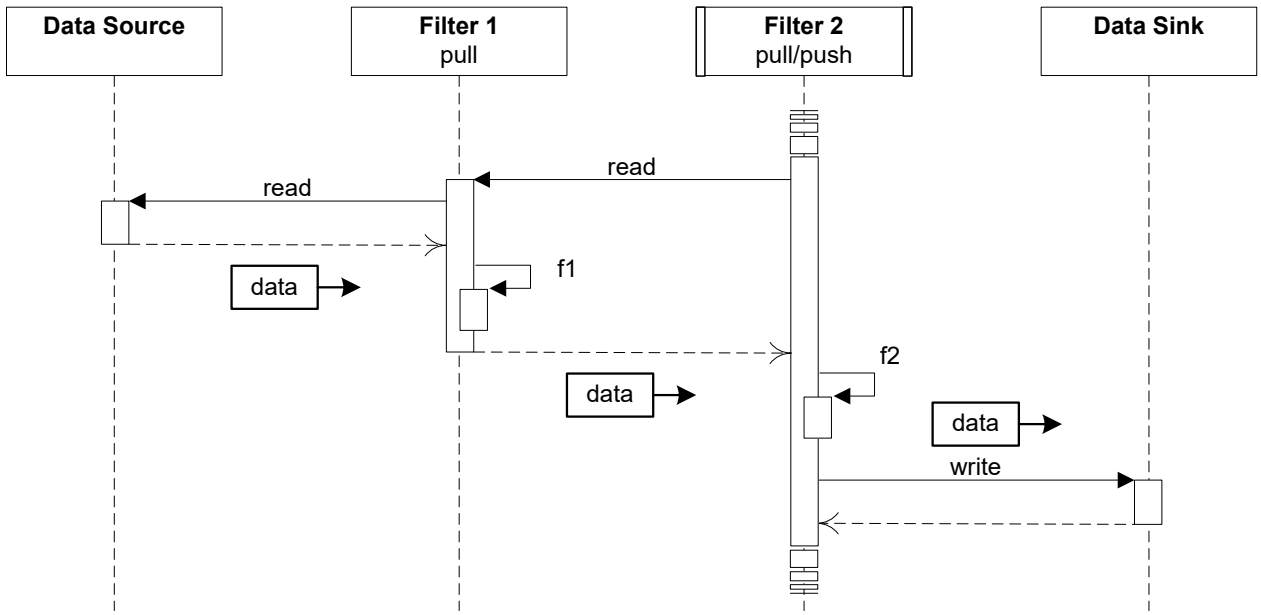
L'elemento attivo qui è la destinazione



Scenario 3

- Una pipeline mista pull-push

L'elemento attivo è uno dei filtri. Questo filtro è di tipo pull/push, TUTTI quelli a sinistra sono di tipo pull, TUTTI quelli a destro sono di tipo push



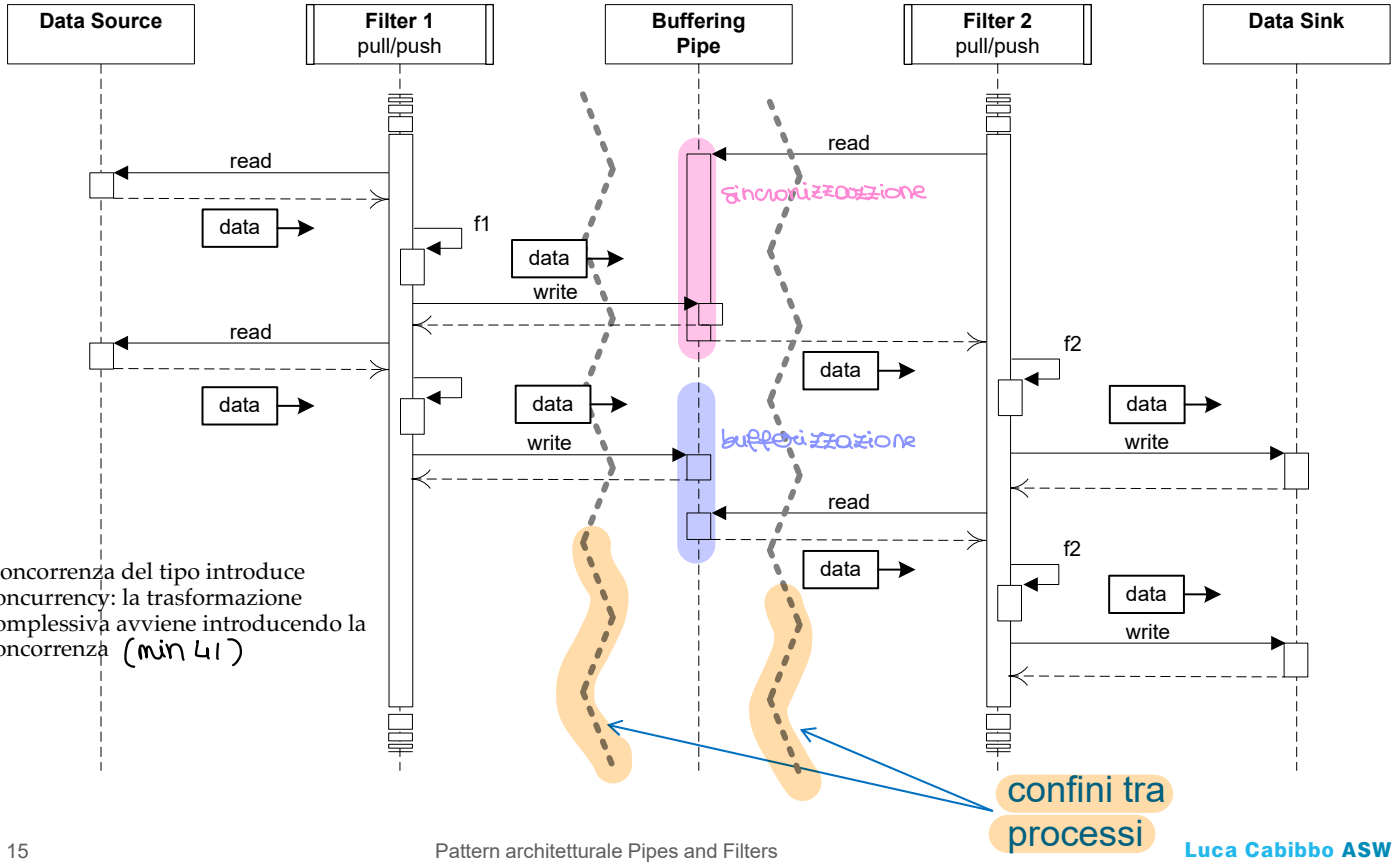
Cioè chiede a oltranza, quando ha informazioni sufficienti scrive

Scenario 4

- Una pipeline con filtri attivi

Ci sono più filtri attivi, se non addirittura tutti, che operano TUTTI in modalità pull/push. Chiede un carattere finché non può produrre un token (o un'altra cosa tipo un pezzo di albero sintattico) da scrivere

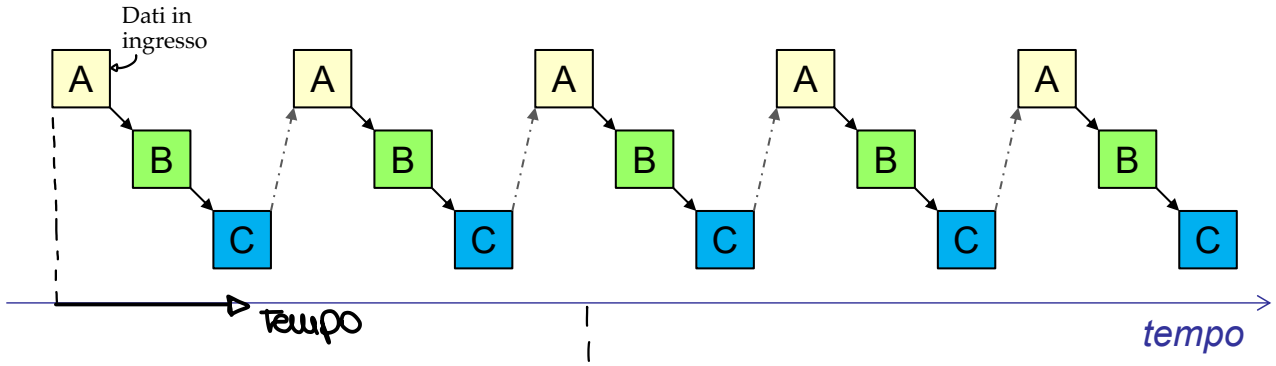
La buffering pipe, siccome i filtri vivono in processi diversi, serve a sincronizzarli. Cioè se un filtro è troppo lento o troppo veloce vanno memorizzati dati per stabilizzare i ritmi di lavoro. Supporta la comunicazione interprocesso



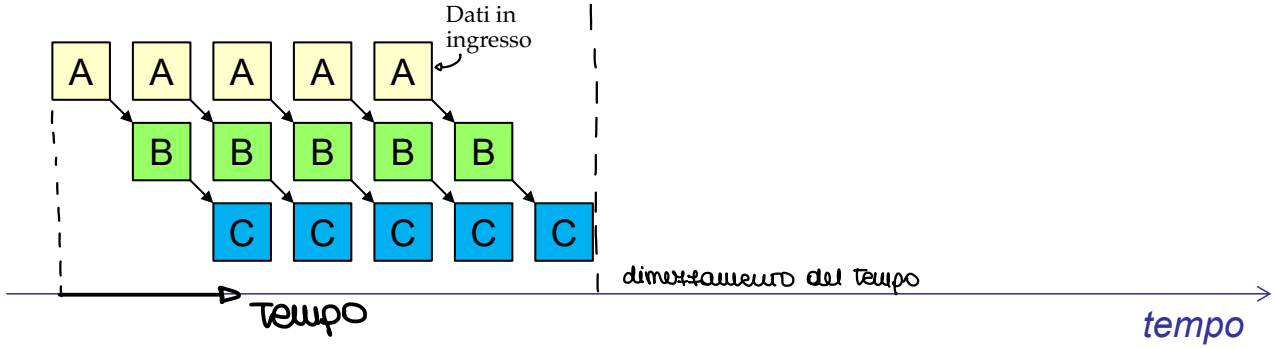
Concorrenza del tipo introduce concurrency: la trasformazione complessiva avviene introducendo la concorrenza (min 41)

Confronto tra scenari

- pipeline di tipo push



- pipeline con più filtri attivi



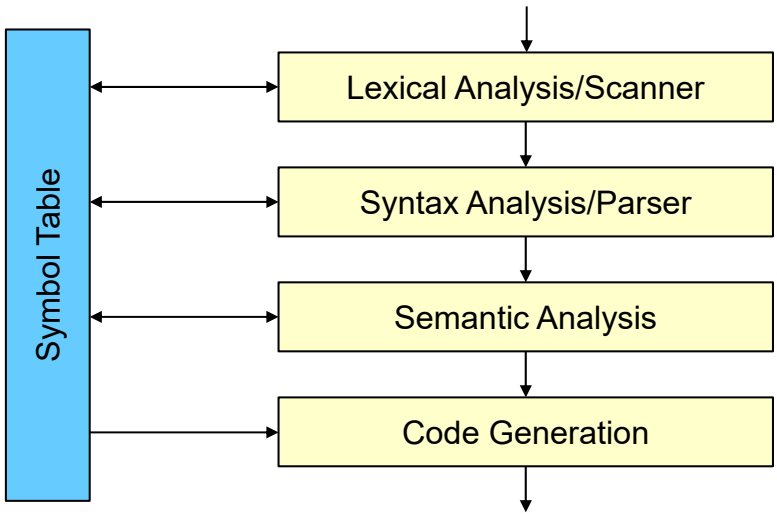
Pipes and Filters – Applicazione

- 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? Se in modalità push o pull
  - come sono implementate le pipe? Se ce ne sono alcuni che posso riutilizzare
- Progetta e implementa i filtri
- Progetta per la gestione degli errori
- Metti in piedi la pipeline di elaborazione

Qui è più complesso di layers, perché l'errore viene sollevato dal primo filtro e questo non tiene in conto delle trasformazioni dei filtri successivi (es se mi dimentico di mettere un parametro l'errore mi dice che manca una parentesi tonda, non che c'è un parametro mancante) per cui non viene fornito contesto dell'errore

Esempio

- Il compilatore può essere basato su Pipes and Filters – ma viene utilizzata anche una tavola dei simboli condivisa



inizio 2ª parte registrazione

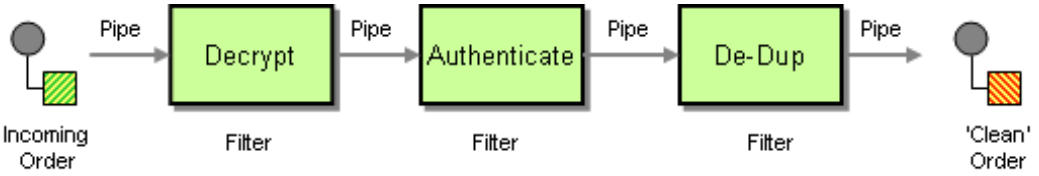
Esempio

- 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

L'obiettivo e la migliore situazione per usare pipe and filters è definire filtri per cui quando ho bisogno di diverse trasformazioni non devo scriverne di nuovi ma devo solo cambiarli, cambiarne l'ordine, o usarne altri già disponibili

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

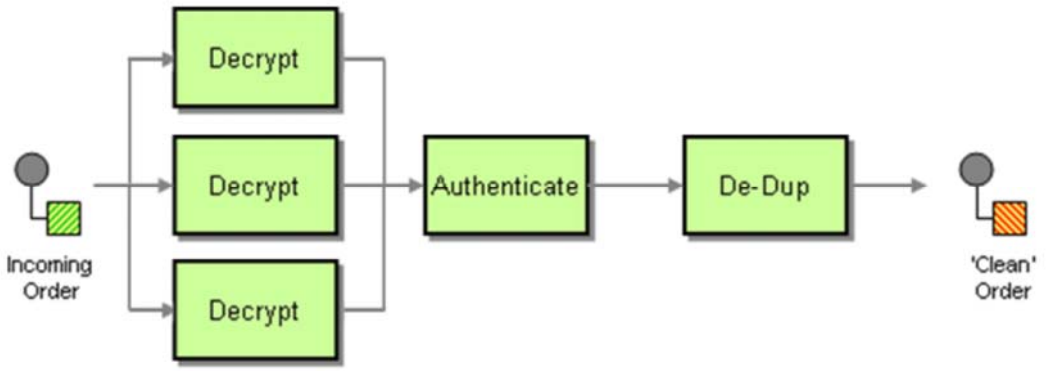


Filtri di questo tipo sono riutilizzabili, questo è fortemente desiderato in questo tipo di architettura. I FILTRI CHE FANNO UNA COSA SOLA SONO PIÙ FACILMENTE RIUTILIZZABILI e danno vita ad una libreria di filtri desiderabile

Esempio (cont.)

Molto spesso i filtri coinvolti viaggiano a velocità diversa. I tempi sono sempre legati al filtro più lento, quindi per non penalizzare le prestazioni va valutata la pipeline nell'insieme (non sempre è possibile aggirare vincoli del genere)

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



Es ogni decriptazione ci mette 3 unità di tempo, mentre un'autenticazione e un'operazione di de-duplicazione impiegano una unità di tempo ciascuna

- 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

- In Pipes and Filters 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

minuto 10.50

- 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

    Cioè se devo cambiare la pipeline, l'ordine dei filtri o i parametri dei filtri ma senza dover definire nuovi filtri o modificarne. Se invece devo modificare filtri si vuole avere filtri che fanno una cosa sola, coesi, debolmente connessi con gli altri.
  - ☺ 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 Defer binding
  - ☹ 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

Variante non necessariamente sequenziale

- 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

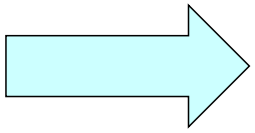
Esempio: esecuzione di query relazionali

Albero di esecuzione a cui si associa un piano di esecuzione che decide come eseguire questa interrogazione

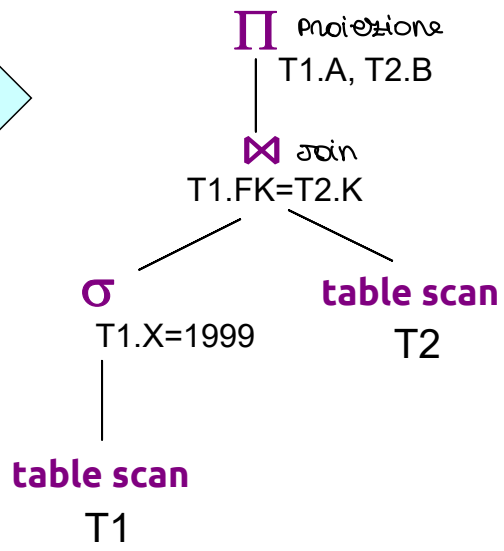
SQL Query:

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

Gli operatori sono implementati da filtri



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
    - Data analytics
      - 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