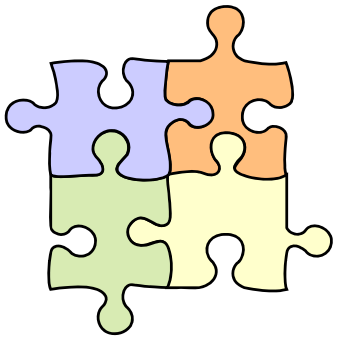
Luca Cabibbo

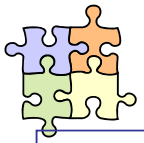


Architetture Software

Stili architetturali e pattern

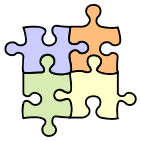
Dispensa AS 11

ottobre 2008



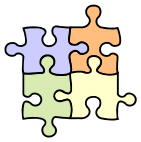
- Fonti

- [SSA] Chapter 11, Using Styles and Patterns
- [GoF] Design Patterns – Elementi per il riuso di software a oggetti
- [POSA] Pattern-Oriented Software Architecture – A System of Patterns
- [POSA4] Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing



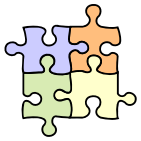
- Obiettivi e argomenti

- Obiettivi
  - comprendere il ruolo dei pattern software nel contesto della definizione dell'architettura
- Argomenti
  - pattern software
  - stili, pattern e idiomi
  - alcuni stili architetturali comuni
  - benefici nell'uso degli stili architetturali
  - stili e descrizioni architetturali
  - design pattern e idiomi nell'architettura



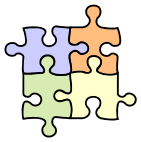
* Pattern software

- Lo scopo di un *pattern software* è
  - condividere una soluzione provata ed ampiamente applicabile ad un particolare problema di progettazione, descritta in una forma standard che possa essere facilmente riusata
- Una possibile descrizione strutturata
  - nome
  - interessi – la situazione in cui il pattern può essere applicato
  - problema – e forze in gioco
  - soluzione
  - conseguenze – risultati (positivi e negativi) e compromessi



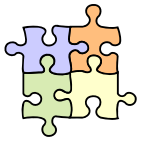
Esempio: Adapter [GoF]

- **Adapter** [GoF]
  - adatta l'interfaccia di un elemento di un sistema ad una forma richiesta da uno dei suoi client
  - scopo – convertire l'interfaccia di una classe in un'altra interfaccia richiesta dal client – ***Adapter*** consente a classi diverse di operare insieme quando ciò non sarebbe altrimenti possibile a causa di interfacce incompatibili [GoF]
- Contesto
  - bisogna connettere diversi elementi eterogenei



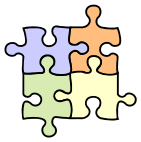
Esempio: Adapter

- Problema (e forze)
  - un elemento (*client*) potrebbe usare i servizi offerti da un altro elemento (*adaptee*), ma l'interfaccia dell'adaptee non è adatta al client
    - ad es., il client è .NET mentre l'adaptee è Java
  - il client potrebbe usare direttamente l'adaptee – ma il conseguente accoppiamento stretto è indesiderato
    - ad es., cambiamenti nell'adaptee richiedono cambiamenti nel client – client diversi devono cambiare in modo diverso
  - preferibile l'uso di un intermediario (adattatore)
    - l'adattatore dovrebbe solo fornire un servizio di adattamento e "traduzione" – e non fornire direttamente delle funzionalità
    - l'uso dell'adattatore non dovrebbe avere effetti negativi sulle qualità del servizio sottostante – ad es., sicurezza, prestazioni, affidabilità, ...

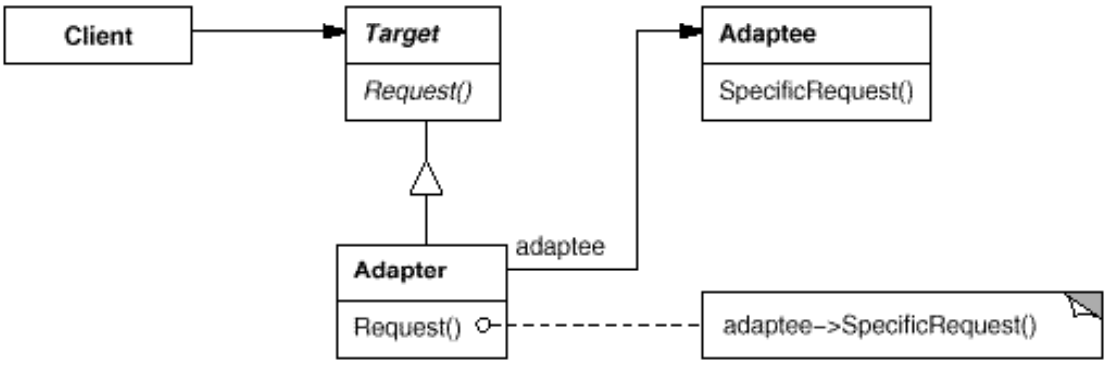


Esempio: Adapter

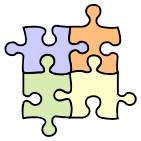
- Soluzione
  - introdurre un terzo elemento separato tra il client e l'adaptee – un *adapter* (adattatore)
    - quando il client vuole comunicare con l'adaptee – il client comunica con l'adattatore e l'adattatore comunica con l'adaptee
    - il ruolo dell'adattatore è semplicemente di interpretare richieste del client, trasformarle in richieste all'adaptee, ottenere risposte dall'adaptee, trasformarle in risposte al client
    - l'adattatore ha in generale un'interfaccia (*target*) diversa da quella dell'adaptee – ha un'interfaccia gradita al client



Esempio: Adapter

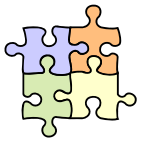


- Attenzione
  - [SSA] chiama *target* quello che [GoF] chiama *adaptee*



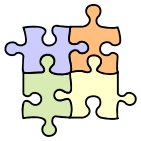
Esempio: Adapter

- Conseguenze
  - ☺ disaccoppiamento delle implementazioni del client e dell'adaptee – l'implementazione di ciascun elemento può variare senza che questo si ripercuota sull'altro elemento
  - ☺ l'adaptee può essere usato da diversi tipi di client, ciascuno col suo adattatore
  - ☹ l'indirezione addizionale potrebbe ridurre l'efficienza
  - ☹ ci potrebbe essere un aumento nell'overhead per la manutenzione se cambia il servizio offerto dall'adaptee – e quindi oltre all'adaptee (ed eventualmente al client) deve cambiare anche l'adattatore



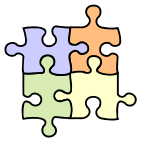
Esempio: Observer [GoF]

- **Observer** [GoF] – **Publisher/Subscriber** [POSA]
  - definisce una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato, tutti gli oggetti dipendenti da questo siano notificati e aggiornati automaticamente
- Contesto
  - un elemento (*publisher*) crea informazioni che sono di interesse per altri elementi (*subscriber*)



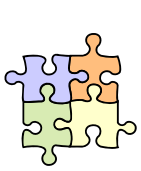
Esempio: Observer

- Problema
  - diversi tipi di oggetti *subscriber* (abbonato) sono interessati ai cambiamenti di stato o agli eventi di un oggetto *publisher* (editore)
  - ciascun subscriber vuole reagire in un modo proprio quando un publisher genera un evento
  - il publisher vuole mantenere un accoppiamento basso verso i suoi subscriber

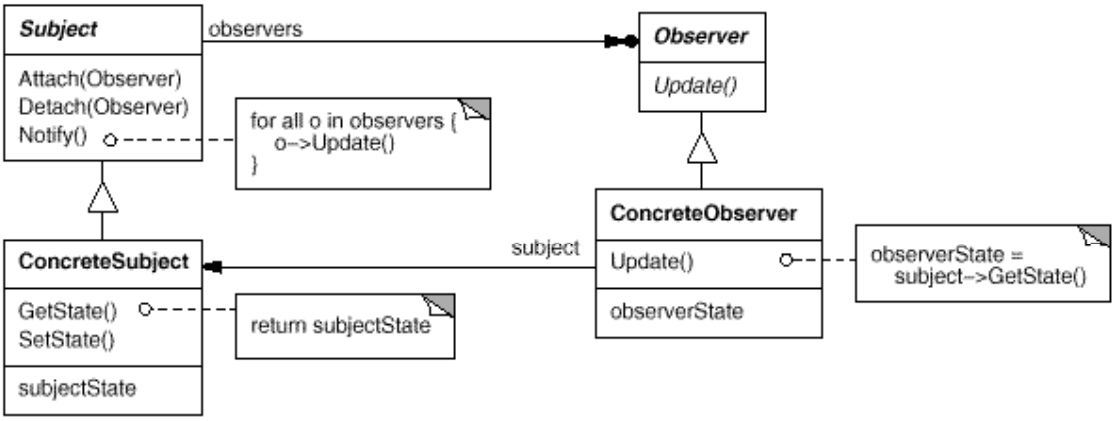


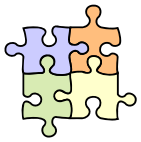
Esempio: Observer

- Soluzione
  - definisci un'interfaccia "subscriber" o "listener" (ascoltatore)
  - i subscriber implementano questa interfaccia
  - il publisher registra dinamicamente i subscriber che sono interessati ai suoi eventi – e li avvisa quando si verifica un evento
    - la notifica può contenere tutti i dettagli dell'evento – oppure dire semplicemente "qualcosa è cambiato" – poi il subscriber, se interessato, interroga il publisher
  - componenti – publisher e subscriber
  - connettori – un canale affidabile per la trasmissione delle notifiche



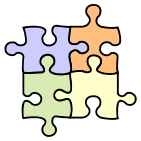
Esempio: Observer





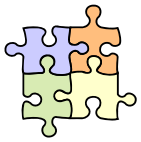
Esempio: Observer

- Conseguenze
  - ☺ accoppiamento debole (astratto e minimale) tra il publisher e i suoi subscriber
  - ☺ supporto per comunicazione broadcast
  - ☺ possibilità di aggiungere/rimuovere i subscriber dinamicamente
  - ☺ rimuove la necessità del polling da parte dei subscriber
  - ☹ potrebbe essere difficile comprendere le relazioni di dipendenza tra i vari elementi
  - ☹ effetto imprevedibile degli aggiornamenti – una modifica in un publisher può scatenare una catena di aggiornamenti e sincronizzazioni su tutti i subscriber
  - ☹ implementazione complessa – se è richiesta una consegna affidabile dei messaggi



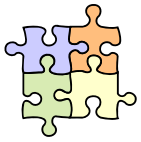
* Stili, pattern e idiomi

- Tre categorie di pattern software
  - stili architetturali
  - design pattern
  - idiomi
- Non completamente indipendenti



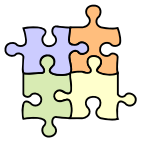
Stili, pattern e idiomi

- 🕮 [SSA]
  - uno **stile** (o **pattern**) **architetturale** esprime una schema per l'*organizzazione strutturale fondamentale di sistemi software*
  - fornisce un insieme di tipi di elementi predefiniti, specifica le loro responsabilità, e comprende regole e linee guida per organizzare le relazioni tra di essi
- Esempi di stili architetturali
  - Layers, Client/Server, Peer-to-Peer, ...
- Detto in altro modo
  - ciascun pattern architetturale guida nella scelta della decomposizione del sistema in elementi di un certo tipo e delle relazioni tra questi elementi
  - discute le possibilità di raggiungere (o meno) certi obiettivi di qualità



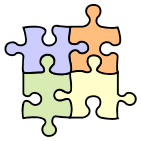
Stili, pattern e idiomi

- [SSA]
  - un **design pattern** fornisce uno schema per *raffinare gli elementi di un sistema software o le relazioni tra di essi*
  - descrive una struttura che ricorre comunemente di elementi di progetto interconnessi, che risolvono un problema di progettazione generale in un contesto particolare
- Esempi di design pattern
  - Singleton, Adapter, Proxy, ...
- Detto in altro modo
  - i design pattern possono essere usati nella progettazione di dettaglio di singoli elementi architetturali – nonché delle interazioni tra di essi



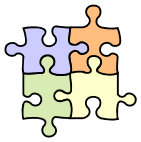
Stili, pattern e idiomi

- [SSA]
  - un **idioma** è un pattern di basso livello, specifico di un linguaggio di programmazione
  - descrive come implementare *aspetti particolari di elementi o delle relazioni tra essi*, *usando* una caratteristica di *un certo linguaggio*
- Esempi di idiomi
  - come implementare Singleton in Java
    - variabile di classe *instance*, metodo di classe *getInstance*, costruttore privato
  - counted pointer
    - per gestire oggetti condivisi in C++ – ogni oggetto sa quanti oggetti lo referenziano – quindi sa quando può essere deallocato
    - utile in Java?



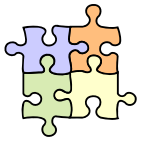
Pattern e linguaggi di pattern

- I pattern non sono tra loro indipendenti
  - alcuni pattern sono alternativi – uso uno oppure l'altro
  - altri pattern sono sinergici – se uso uno è utile usare anche l'altro
  - altri possono essere usati in gruppi più complessi
  - utile ragionare anche sulle relazioni tra pattern
- Un *pattern language* (*linguaggio di pattern*)
  - una famiglia di pattern correlati
    - con una discussione sulle loro correlazioni
  - ne esistono diversi – specifici per la progettazione di certi tipi di sistemi o per certi tipi di requisiti
    - ad es., linguaggio di pattern per la sicurezza
    - ad es., linguaggio di pattern per sistemi distribuiti [POSA4]



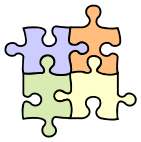
Ruolo dei pattern software

- Alcuni dei ruoli svolti da pattern (e linguaggi di pattern)
  - deposito di conoscenza
  - esempi di buone pratiche
  - un linguaggio per discutere problemi di progettazione
  - un aiuto alla standardizzazione
  - una sorgente di miglioramento continuo
  - incoraggiamento alla generalità
- Il ruolo principale dal punto di vista delle architetture software
  - riduzione del rischio
  - incremento della produttività, della standardizzazione e della qualità



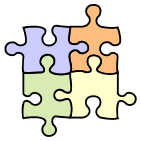
* Alcuni stili architetturali comuni

- Molti stili architetturali comuni
  - client/server a due o più livelli, peer-to-peer, layers, oggetti distribuiti, ...
- Vengono ora descritti brevemente alcuni stili architetturali comuni
  - client/server
  - layers
  - pipes and filters
- La discussione sarà ripresa, ampliata ed approfondita nel seguito del corso



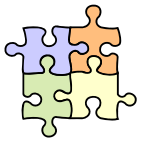
- Client/server

- **Client/server** è uno stile architetturale molto diffuso
- Due tipi di elementi/componenti – sono processi
  - un *server* che offre uno o più *servizi* mediante un'interfaccia ben definita
  - uno o più *client* che usano i servizi come parte delle loro operazioni
- Un altro tipo di elementi/connettori
  - un formato e un *protocollo* – definito dall'interfaccia del server
- Nota
  - client e server possono essere su calcolatori diversi – ma potrebbero anche essere sullo stesso calcolatore



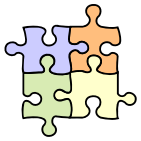
Client/server

- Esempi di uso
  - molti servizi di Internet, accesso alle basi di dati, ...
- Conseguenze
  - ☺ condivisione di risorse, centralizzazione di elaborazione complessa o sensibile, ...
  - ☹ overhead della comunicazione, ...
- Alcune varianti comuni
  - server senza stato e client con stato
  - server con stato e client senza stato
  - client/server a più livelli – può essere considerato un altro stile



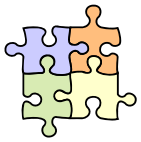
- Layers [POSA]

- **Layers** [POSA]
- Un singolo tipo di elemento (sono solitamente moduli) – lo *strato*
- Organizzazione degli elementi
  - elementi organizzati in una pila di strati
  - ogni strato fornisce servizi allo strato superiore e richiede servizi allo strato inferiore
  - strati organizzati per livello di astrazione
- Conseguenze
  - ☺ riuso di strati, buona separazione degli interessi, facilità di manutenzione, portabilità, ...
  - ☹ diminuzione della flessibilità di implementazione, possibile riduzione dell'efficienza, ...



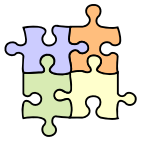
- Pipes and Filters [POSA]

- **Pipes and Filters** [POSA]
- Contesto
  - un sistema che deve elaborare un flusso di dati – ad es., un compilatore
- Problema
  - un sistema deve elaborare un flusso di dati
  - l'elaborazione può essere organizzata in una sequenza di trasformazioni
  - la specifica delle singole trasformazioni può cambiare nel tempo
  - l'uso di un singolo processo non è consigliato



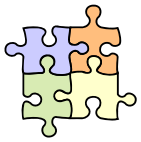
Pipes and Filters

- Forze
  - devono essere possibili cambiamenti futuri che corrispondono alla modifica o riorganizzazione dei passi di trasformazione
  - piccoli passi di trasformazione sono più facilmente riusabili di grandi passi
  - è possibile che ci siano diverse sorgenti di dati
  - passi non consecutivi non devono condividere informazioni
  - la memorizzazione esplicita di risultati intermedi dovrebbe essere evitata
  - l'elaborazione concorrente di passi non dovrebbe essere vietata



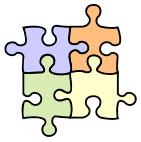
Pipes and Filters

- Soluzione (idea)
  - organizza la trasformazione complessiva in una sequenza di passi
  - collega i passi con il flusso di dati del sistema



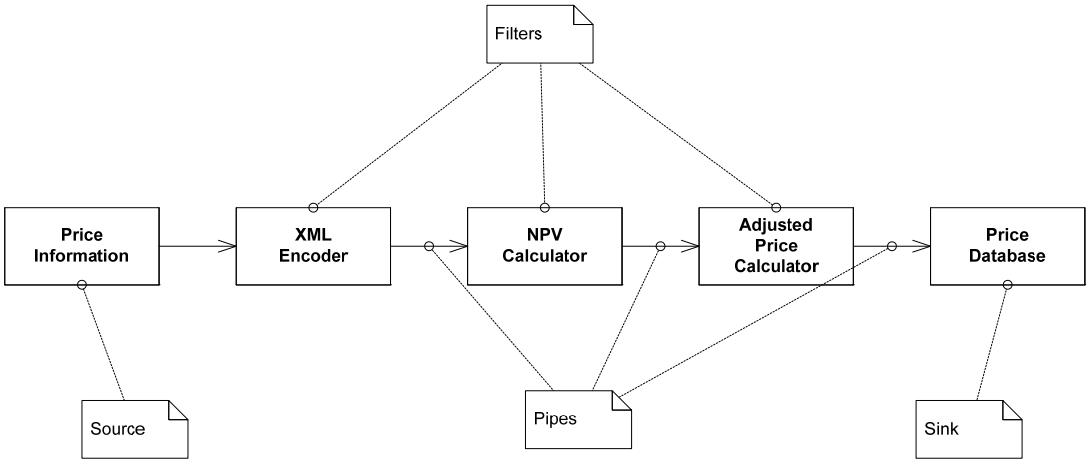
Pipes and Filters

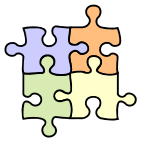
- Soluzione
  - l'elaborazione è svolta da componenti *filtro* – che consumano ed elaborano dati in modo incrementale – con un flusso di ingresso ed un flusso di uscita
    - ogni filtro svolge una singola attività di elaborazione
  - i componenti filtro sono connessi mediante *pipe* (tubi) – che collegano flussi di uscita con flussi di ingresso consecutivi
    - le pipe sono l'unico modo consentito per connettere i filtri
    - le pipe definiscono un formato standard per i dati che possono attraversarle
  - la sequenza di filtri combinati da pipe è chiamata una *pipeline* di elaborazione



Esempio

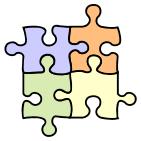
- Un flusso di informazioni su prezzi deve essere memorizzato in una base di dati
  - la rappresentazione nella base di dati è diversa da quella originale
  - le informazioni devono essere arricchite





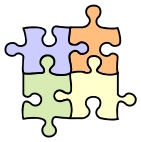
Pipes and Filters

- Conseguenze
  - ☺ non sono necessari file intermedi – ma possibili
  - ☺ l'implementazione dei filtri può essere cambiata senza modificare gli altri elementi del sistema
  - ☺ la ricombinazione di filtri esistenti può consentire la definizione di nuove pipeline di elaborazione
  - ☺ i filtri possono essere riusati in situazioni diverse
  - ☺ possibile l'elaborazione parallela, con filtri eseguiti in modo concorrente
  - ☹ la condivisione di informazioni di stato è difficile
  - ☹ la trasformazione dei dati in un formato inter-filtro comune può richiedere un overhead
  - ☹ la gestione degli errori è difficile



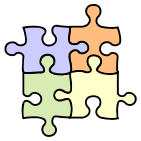
* Benefici nell'uso degli stili architetturali

- Benefici nel basare un'architettura su uno stile riconoscibile
  - selezione di una soluzione provata e ben compresa, che definisce i principi organizzativi del sistema
  - più facile comprendere l'architettura e le sue caratteristiche – ovvero il modo in cui sono controllate le varie qualità
- Possibili usi degli stili architetturali
  - soluzione di progetto per il sistema in discussione
  - base per l'adattamento
  - ispirazione per una soluzione correlata
  - motivazioni per un nuovo stile
- È possibile che un'architettura sia basata su più stili
  - ma in genere uno è dominante



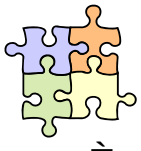
Esempio

- Si consideri un sistema di trading finanziario
  - permette agli utenti di eseguire transazioni di compravendita di azioni
  - notifica informazioni (ad es., prezzi correnti) agli utenti
- Stili applicabili
  - client/server – per la gestione delle transazioni
    - elaborazione sicura, scalabile, affidabile, buone prestazioni
  - publisher/subscriber – per il broadcast delle informazioni
    - diffusione delle informazioni efficace, flessibile, asincrona
  - layers – per l'organizzazione delle singole applicazioni
    - portabilità, modificabilità



* Stili e descrizioni architetturali

- Uno stile architetturale viene espresso in modo più o meno evidente nelle varie viste
  - utile annotare i modelli che formano l'AD con l'indicazione degli stili adottati e delle relative motivazioni



* Design pattern e idiomi nell'architettura

- È chiaro il ruolo degli stili architetturali nella definizione di un'architettura
  - quale il ruolo dei design pattern e degli idiomi nella definizione di un'architettura?
  - utili (soprattutto i design pattern) nel descrivere le connessioni tra elementi architetturali
    - l'architettura deve comprendere una descrizione delle relazioni tra elementi ed indicare un approccio uniforme nella loro realizzazione
    - ad es., gli strati comunicano dall'alto verso il basso con Facade, l'accesso ai dati persistenti mediante DAO, trasferimento di dati tra livelli mediante DTO, ...
  - strumento di comunicazione tra architetto e progettista