



Luca Cabibbo
Architettura
dei Sistemi
Software

Altri pattern architetturali POSA

dispensa asw350
ottobre 2024

*If I speak of myself in different ways,
that is because I look at myself
in different ways.*
Michel de Montaigne



- Riferimenti

- Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
 - Capitolo 19, **Altri pattern architetturali POSA**
- [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 alcuni ulteriori pattern architetturali POSA comuni

▣ Argomenti

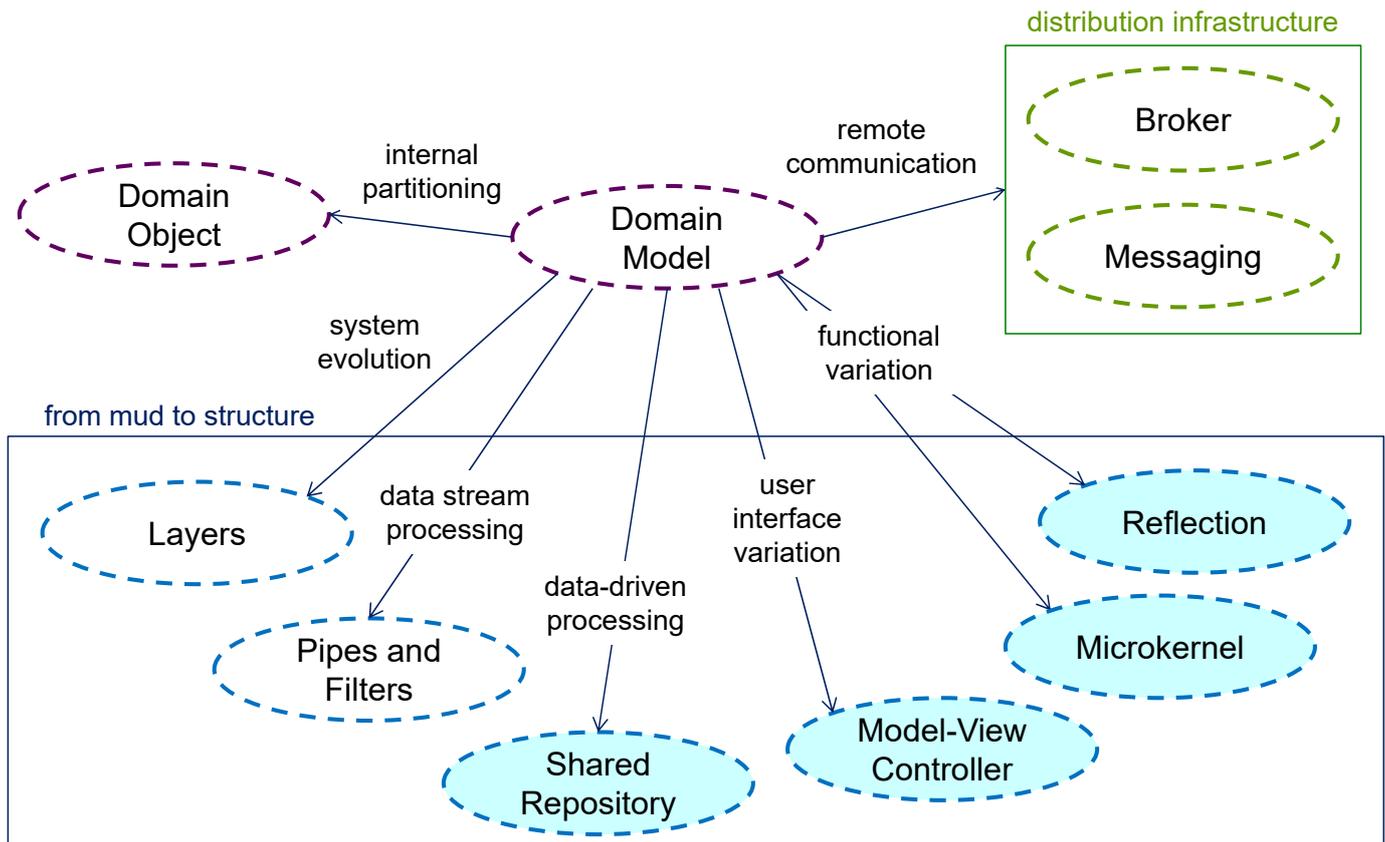
- introduzione
- Model-View-Controller (POSA)
- Microkernel (POSA)
- Reflection (POSA)
- Shared Repository (POSA4)
- discussione



* Introduzione

▣ Presentiamo ora degli ulteriori pattern architetturali POSA

- che si concentrano su alcuni aspetti specifici dei sistemi software – come interfacce grafiche e gestione di dati condivisi



* Model-View-Controller (POSA)

- Il pattern architetturale **Model-View-Controller (MVC)**
 - nella categoria “user interface variation” di [POSA4]
 - divide un’applicazione interattiva in tre tipologie di componenti
 - un *modello*
 - una o più *viste*
 - uno o più *controller*
 - in cui un’*interfaccia utente* è formata da una vista e un controller



Esempio

Brown	4003
Red	14535
Green	3927
Yellow	5235
Blue	16637
Dark Blue	5701
Aquamarine	3465
Black	2850
Others	1381

votes (k)

Brown	6,9%
Red	25,2%
Green	6,8%
Yellow	9,1%
Blue	28,8%
Dark Blue	9,9%
Aquamarine	6,0%
Black	4,9%
Others	2,4%

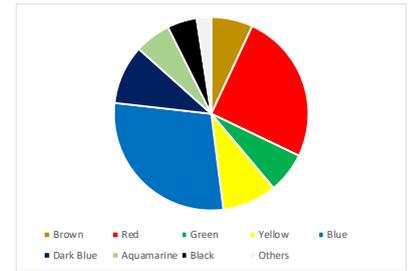
percentage

Brown	52
Red	189
Green	51
Yellow	68
Blue	216
Dark Blue	74
Aquamarine	45
Black	37
Others	18

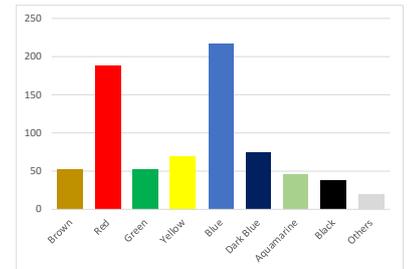
seats

Core Data

Party	Votes (k)	Seats
Brown	4003	52
Red	14535	189
Green	3927	51
Yellow	5235	68
Blue	16637	216
Dark Blue	5701	74
Aquamarine	3465	45
Black	2850	37
Others	1381	18



pie chart



bar chart



parliament



Model-View-Controller

Contesto

- applicazione interattiva – con interfaccia utente flessibile

Problema

- le interfacce utente (UI) sono soggette a richieste di cambiamento frequenti – più frequenti delle altre funzionalità fondamentali dell'applicazione
- cambiamenti nella UI non si devono ripercuotere sulle funzionalità fondamentali dell'applicazione
- inoltre, un'applicazione può avere più UI



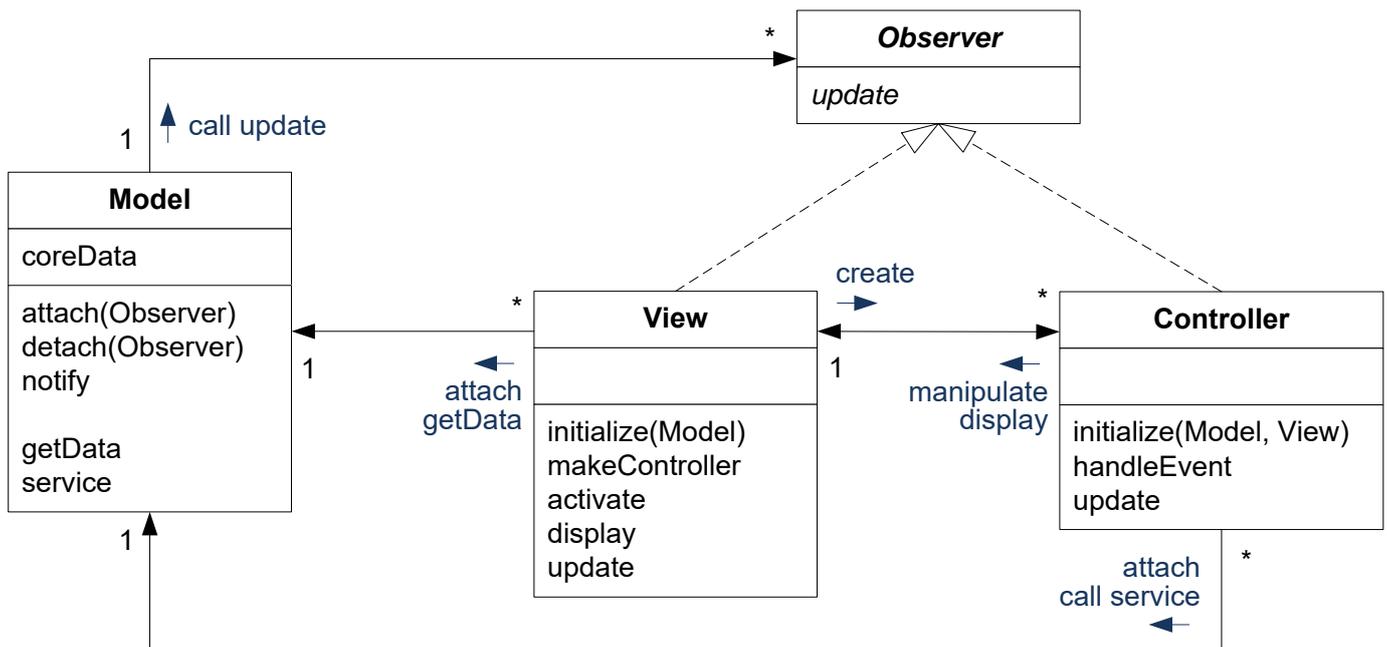
Model-View-Controller

□ Soluzione

- dividi l'applicazione interattiva in tre parti disaccoppiate
 - il *modello* contiene i dati e le funzionalità di base
 - le *viste* mostrano informazioni agli utenti
 - i *controller* gestiscono le richieste degli utenti
 - ogni vista ha un suo controller
- garantisci inoltre la coerenza delle tre parti con l'aiuto di un meccanismo di propagazione dei cambiamenti



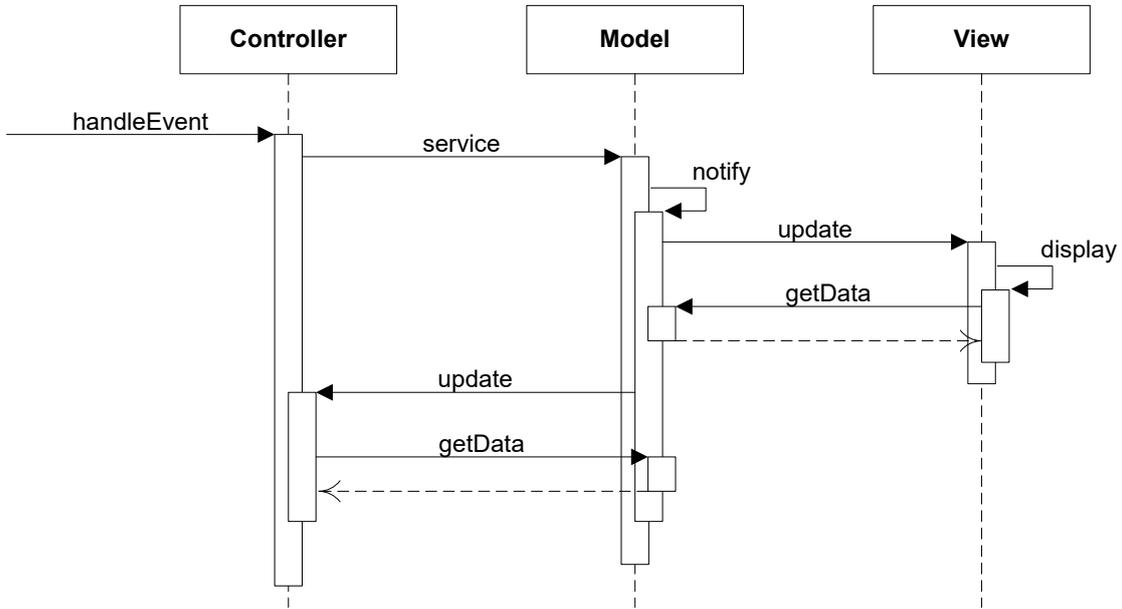
Struttura



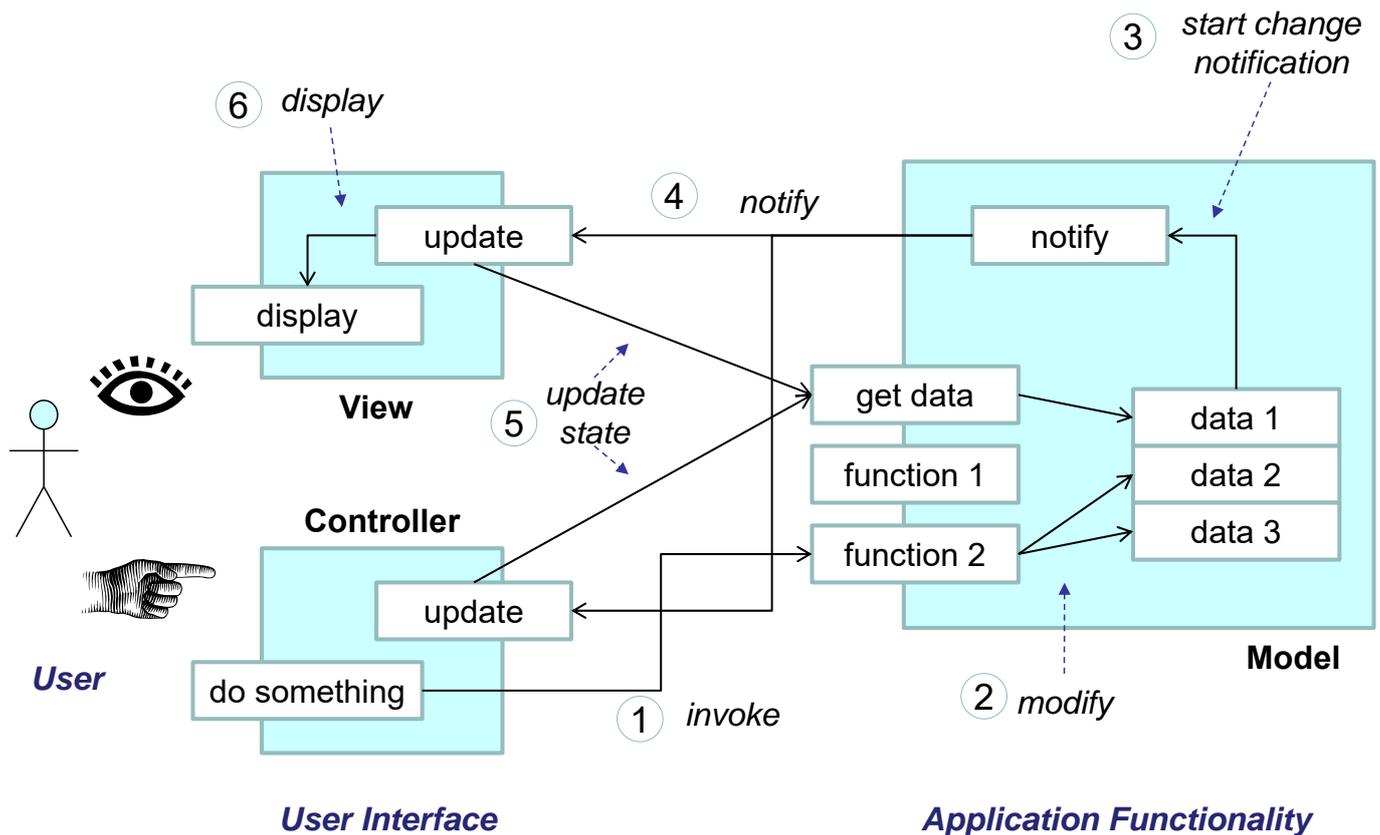


Scenario 1

Scenario relativo alla propagazione dell'input



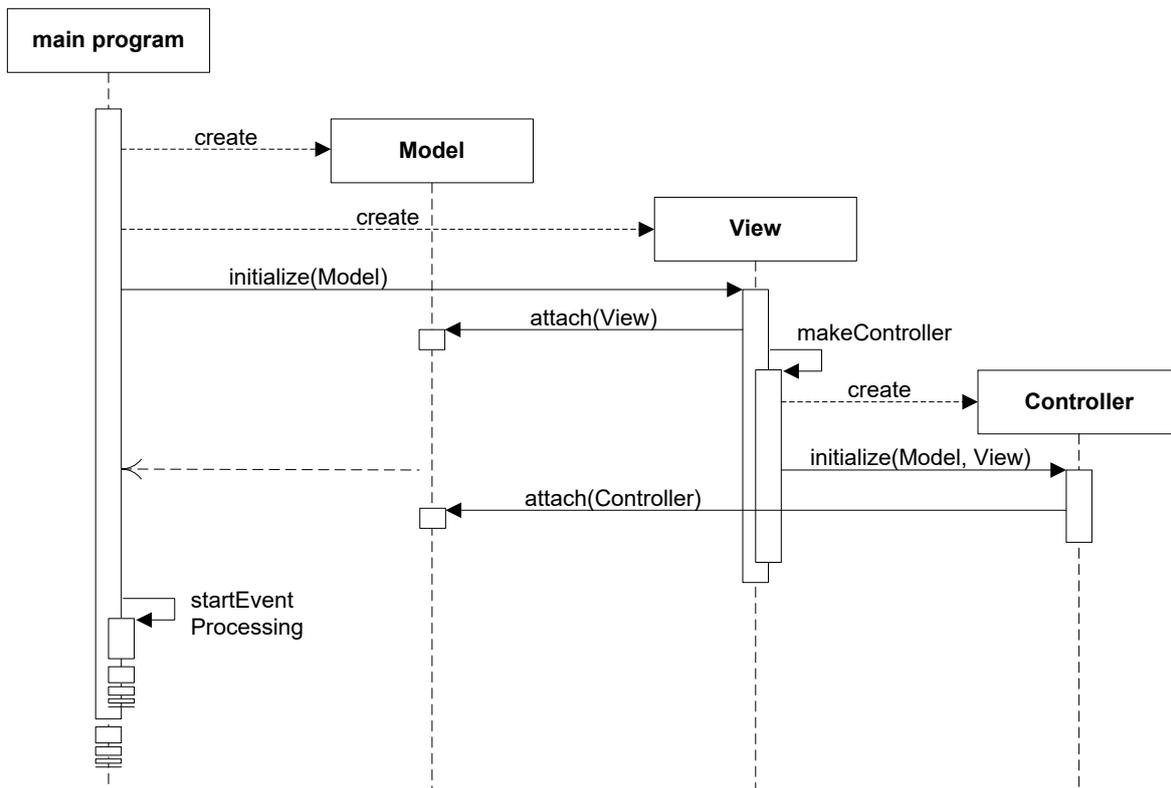
Scenario 1





Scenario 2

Scenario relativo all'inizializzazione della triade MVC



13

Altri pattern architetturali POSA

Luca Cabibbo ASW



Model-View-Controller

Alcuni linee guida

- incapsula le funzionalità fondamentali dell'applicazione nel modello – in modo indipendente dalle modalità di visualizzazione e dai meccanismi di interazione
- partiziona il modello in un insieme di Domain Object
- associa ogni “pezzo coerente di informazioni” del modello a una o più viste auto-contenute
- associa ciascuna vista con un insieme separato di controller
- consenti all'utente di interagire con l'applicazione solo tramite i controller
- collega modello, vista e controller con un meccanismo di propagazione dei cambiamenti

14

Altri pattern architetturali POSA

Luca Cabibbo ASW



Conseguenze

▣ Benefici

- 😊 possibili viste multiple sullo stesso modello
- 😊 possibili modalità di interazioni multiple su una vista
- 😊 sincronizzazione delle viste
- 😊 viste e controller plug-and-play
- 😊 look-and-feel plug-and-play
- 😊 possibilità di sviluppare/riusare framework



Conseguenze

▣ Possibili inconvenienti

- 😞 aumento della complessità
- 😞 rischio di numero eccessivo di aggiornamenti
- 😞 accoppiamento tra vista e controller – e di vista e controller con il modello
- 😞 inefficienza nell'accesso ai dati da parte delle viste
- 😞 dipendenza dalla piattaforma – il porting di un sistema MVC su una piattaforma diversa può essere difficoltoso
- 😊 esistono varianti di MVC considerate più flessibili/portabili o più adeguate per gli strumenti attuali per lo sviluppo di interfacce utente – ad es., PAC, MVP, ...



- Usi conosciuti

- Alcuni usi conosciuti del pattern MVC
 - MVC è alla base di molti framework per la definizione di interfacce utente
 - ad es., quello di Smalltalk
 - anche le applicazioni web in Java possono essere strutturate secondo MVC – ad es., Spring Web MVC
 - spesso sono usati dei pattern derivati da MVC
 - ad es., Model-View-Presenter in .NET



* Microkernel (POSA)

- Il pattern architetturale **Microkernel**
 - nella categoria “functional variation” di [POSA4]
 - sostiene lo sviluppo di una “famiglia” (o “linea”) di applicazioni o prodotti software
 - le diverse applicazioni sono tutte basate sulla una stessa architettura e un unico nucleo funzionale – e vengono costruite in sede di compilazione o deployment



Microkernel

□ Contesto

- applicazione adattabile a diversi scenari

□ Problema

- alcune applicazioni devono esistere in versioni multiple
- malgrado le differenze, le diverse versioni dell'applicazione dovrebbero essere basate su una stessa architettura comune e uno stesso nucleo funzionale comune
- alcuni obiettivi di progetto
 - evitare variazioni architetturali
 - minimizzare lo sforzo di sviluppo ed evoluzione delle funzioni comuni
 - consentire nuove versioni e di variare le versioni esistenti



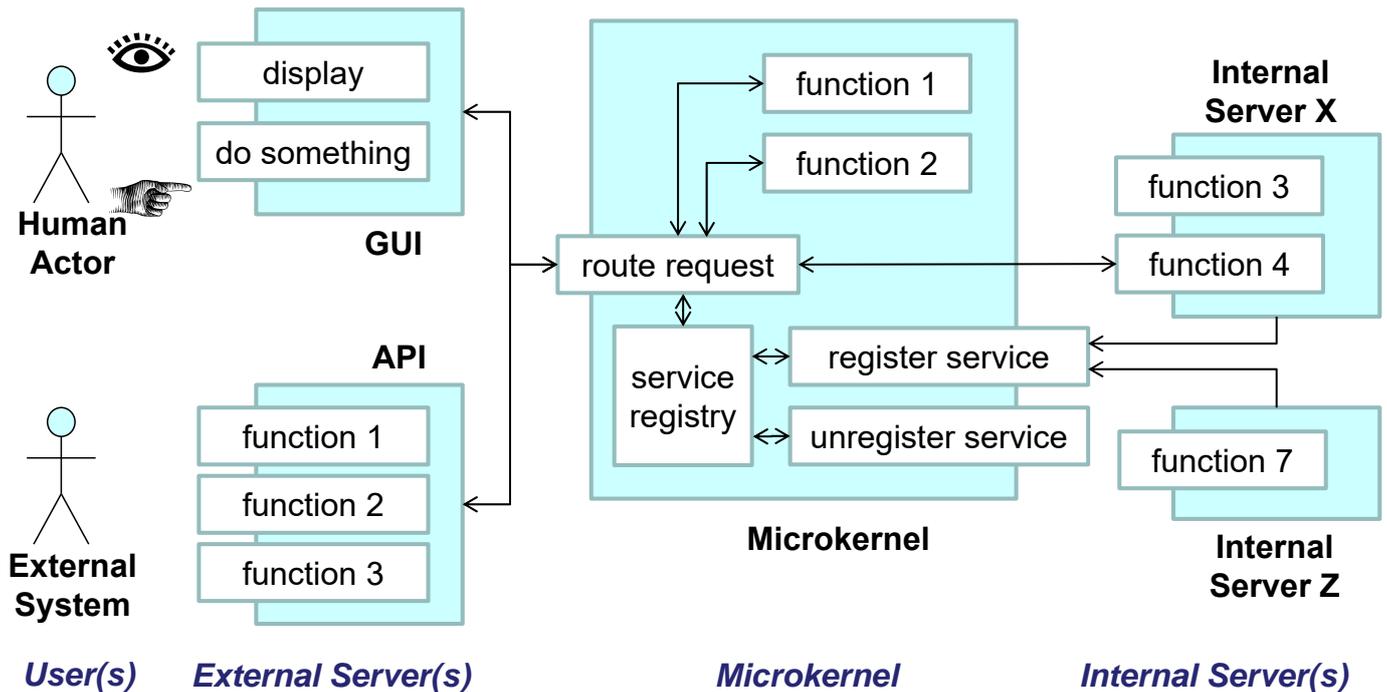
Microkernel

□ Soluzione

- componi le diverse versioni dell'applicazione estendendo un nucleo minimale comune – sulla base dei seguenti elementi
 - un *microkernel* – implementa le funzionalità condivise da tutte le versioni – fornisce l'infrastruttura per integrare le diverse versioni
 - uno o più *server interni* (IS) – ognuno implementa delle funzionalità auto-contenute, specifiche per una versione
 - uno o più *server esterni* (ES) – ognuno implementa un'interfaccia utente o un'API specifica per una versione
- ciascuna versione specifica dell'applicazione è composta dal microkernel, un insieme di IS e un insieme di ES



Struttura di una versione dell'applicazione



Microkernel

□ Dinamica

- in sede di deployment, il microkernel registra gli IS e le richieste che sono in grado di soddisfare
 - sono anche possibili varianti sul momento del collegamento
- i client effettuano richieste al sistema solo tramite l'interfaccia o l'API di un ES – che propaga le richieste al microkernel
- la gestione di una richiesta da parte del microkernel viene eseguita direttamente dal microkernel oppure delegata a un IS



Discussione

- Un'architettura basata su Microkernel
 - sostiene lo sviluppo, l'evoluzione e la gestione di versioni multiple di un'applicazione
 - garantisce che ciascuna versione possa essere effettivamente definita con riferimento al suo scopo specifico
 - sostiene la definizione di nuove versioni dell'applicazione
 - minimizza lo sforzo di sviluppo e di manutenzione delle applicazioni



- Usi conosciuti

- Alcuni usi conosciuti del pattern architetturale Microkernel
 - nella realizzazione del kernel dei sistemi operativi
 - nella realizzazione di linee di prodotto
 - le *architetture a plug-in* possono essere considerate una variante di Microkernel
 - ad es., l'architettura a plug-in di Eclipse (E3) oppure quella di Mozilla Firefox



* Reflection (POSA)

- Il pattern architetturale **Reflection**
 - nella categoria “functional variation” di [POSA4]
 - consente di cambiare la struttura e il comportamento di un sistema in modo dinamico, anche a runtime – sulla base della modifica di aspetti fondamentali, come le strutture di dati e i meccanismi di comunicazione

- *Reflection*
 - In computer science, reflection is the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime. [Wikipedia, 2018]



Reflection

- **Contesto**
 - un sistema che deve consentire delle variazioni – in qualunque momento, anche in produzione

- **Problema**
 - il sistema deve poter evolvere nel tempo
 - alcune modifiche possono avvenire in qualunque momento, e anche quando il sistema è in esecuzione in produzione
 - in questi casi non è accettabile realizzare le modifiche intervenendo sul codice
 - piuttosto, il sistema dovrebbe essere aperto a molti tipi di variazioni ed estensioni, in modo dinamico – su richiesta, in momenti appropriati, mediante un meccanismo uniforme



Reflection

□ Soluzione

- rappresenta informazioni sulle proprietà e gli aspetti variabili del sistema utilizzando un insieme di *meta-oggetti* o *meta-dati*
- suddividi il sistema in due parti (livelli) principali
 - un *meta-livello* – contiene i meta-oggetti
 - un *livello base* – comprende la logica applicativa fondamentale del sistema
- connetti il livello base con il meta-livello – in modo che cambiamenti di informazioni nel meta-livello influiscano sul comportamento effettivo del sistema
- fornisci un *protocollo* per amministrare e configurare dinamicamente gli oggetti del meta-livello

27

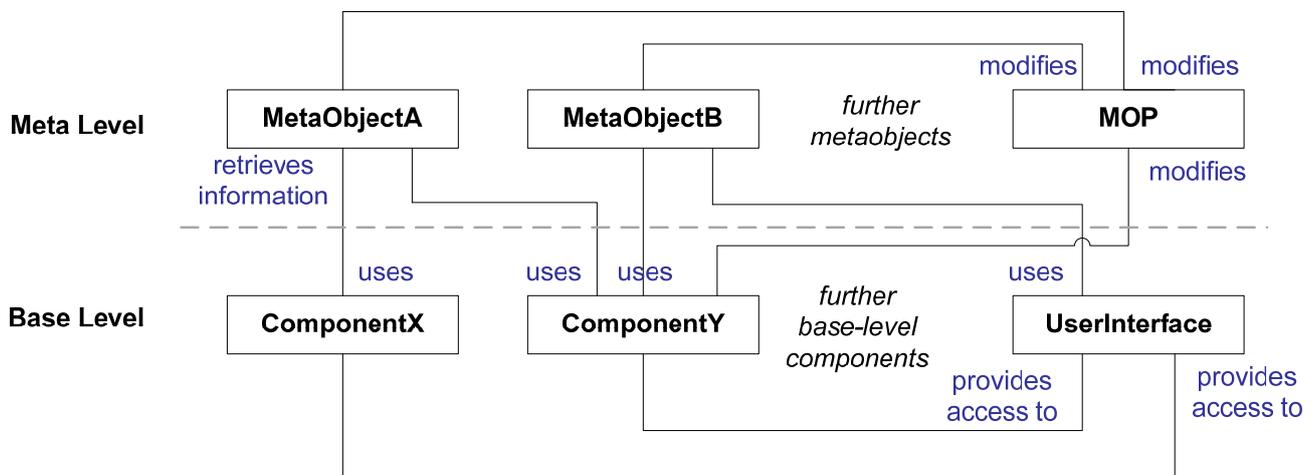
Altri pattern architetturali POSA

Luca Cabibbo ASW



Struttura

- La struttura della soluzione è basata su due livelli – meta e base



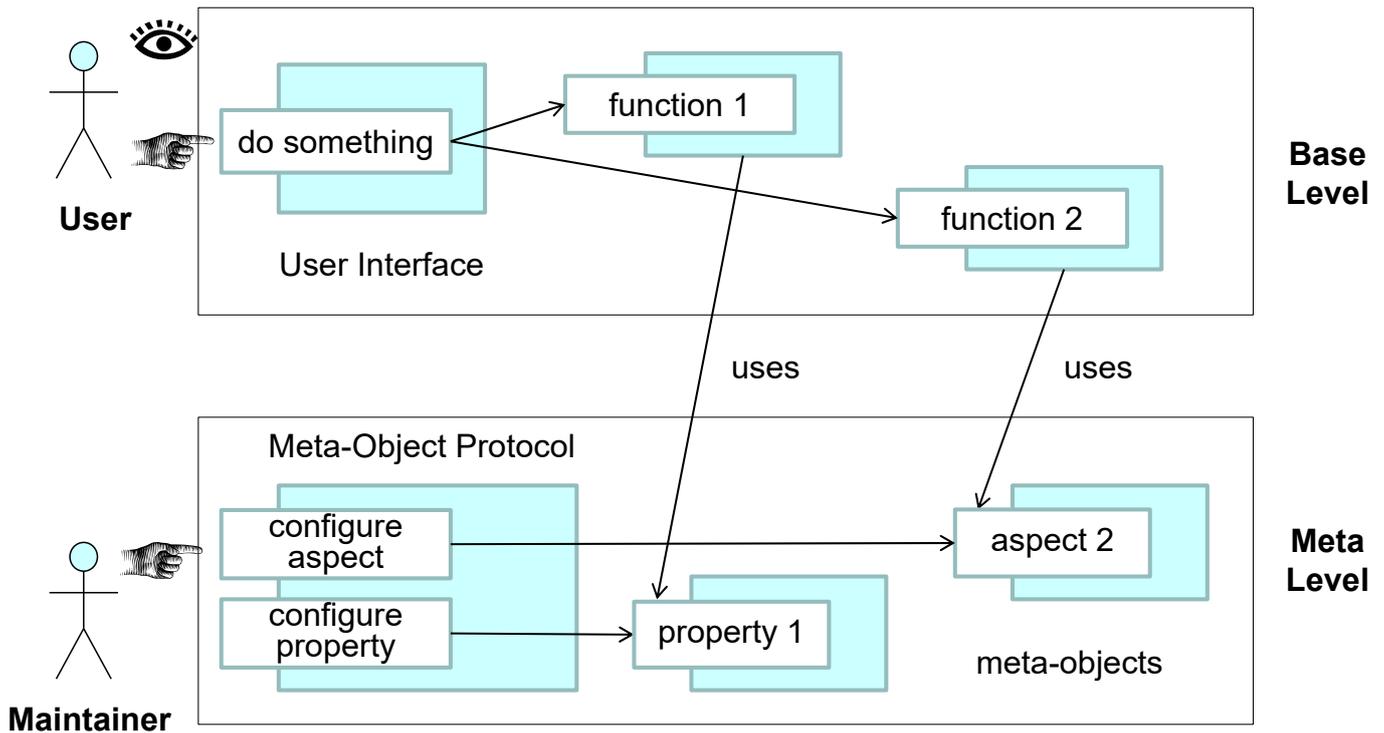
28

Altri pattern architetturali POSA

Luca Cabibbo ASW



Struttura



- questa figura è “girata” rispetto alla precedente

29

Altri pattern architetturali POSA

Luca Cabibbo ASW



Esempio

- Il catalogo di una base di dati in un DBMS relazionale
 - lo schema di una base di dati è rappresentato nel meta-livello mediante un *catalogo* – che descrive anche le corrispondenze tra livello logico e livello fisico
 - il livello base definisce la modalità di esecuzione dei comandi SQL – facendo riferimento agli schemi descritti al meta-livello e alle loro rappresentazione fisiche
 - le meta-informazioni vengono usate per realizzare il comportamento richiesto
 - le applicazioni usano il DBMS al livello base – in SQL
 - l'esecuzione dei comandi coinvolge sia il livello base che il meta-livello

30

Altri pattern architetturali POSA

Luca Cabibbo ASW



Reflection

▣ Alcune linee guida

- inizia progettando un'applicazione in cui non è prevista alcuna variabilità
- identifica i punti di variazione nella struttura e nel comportamento del sistema – e determina le informazioni che possono influire sul comportamento dell'applicazione
- rappresenta ciascun comportamento o struttura variabile in un meta-oggetto separato
- modifica l'implementazione di ciascun oggetto di dominio del progetto iniziale – in modo il suo comportamento sia determinato dai meta-oggetti
- definisci un MOP per gestire i meta-oggetti



Conseguenze

▣ Benefici

- 😊 facile modificare il sistema
- 😊 è possibile effettuare molte modifiche al sistema senza cambiare il codice sorgente

▣ Inconvenienti

- 😞 maggiore complessità
- 😞 minore efficienza
- 😞 non tutte le modifiche sono possibili



- Usi conosciuti

- Anche l'applicazione del pattern architetturale Reflection è pervasiva
 - molti linguaggi di programmazione prevedono meccanismi di riflessione
 - molti sistemi sono basati su una nozione di meta-dati o meta-linguaggio
 - ad es., i DBMS e gli IDE



* Shared Repository (POSA4)

- Il pattern architetturale **Shared Repository**
 - nella categoria “data-driven processing” di [POSA4]
 - guida la connessione tra componenti o applicazioni che operano su dei dati condivisi
 - il coordinamento tra i diversi componenti o applicazioni avviene tramite questi dati condivisi – e non tramite interazioni dirette



Esempio

- ❑ Si vogliono definire un gruppo di componenti applicativi relativi a diverse attività per la gestione degli ordini
 - ad es., ricezione ordini, evasione ordini, fatturazione, ricezione pagamenti, ricezione merci da fornitori, ...
- ❑ Questi componenti applicativi devono agire in modo coordinato
 - ad es., la ricezione di alcune merci da un fornitore potrebbe consentire l'evasione di alcuni ordini sospesi
- ❑ Tuttavia, il coordinamento diretto (point-to-point) tra componenti potrebbe non essere desiderato
 - potrebbe essere difficile esplicitare le modalità di coordinamento tra componenti
 - per il numero di “coordinamenti diretti” tra i componenti



Shared Repository

- ❑ Contesto
 - un'applicazione o sistema data-intensive – i cui componenti devono interagire in modo coordinato
- ❑ Problema
 - alcune applicazioni sono inerentemente guidate dai dati
 - i componenti dell'applicazione devono interagire in modo controllato – anche se le loro interazioni non sono guidate da processi di business espliciti
 - è però possibile coordinare questi componenti con riferimento ai dati condivisi su cui essi operano



Shared Repository

□ Soluzione

- mantieni i dati in un *repository* centrale *condiviso* da tutti i componenti funzionali dell'applicazione
- fa guidare e coordinare il flusso di controllo della logica applicativa dai dati nel repository
 - i componenti lavorano direttamente con i dati del repository condiviso
 - la creazione, modifica o distruzione di dati da parte di un componente viene resa accessibile anche agli altri componenti, che possono reagire di conseguenza
- se il repository condiviso è una base di dati condivisa, allora si parla di *shared database*

37

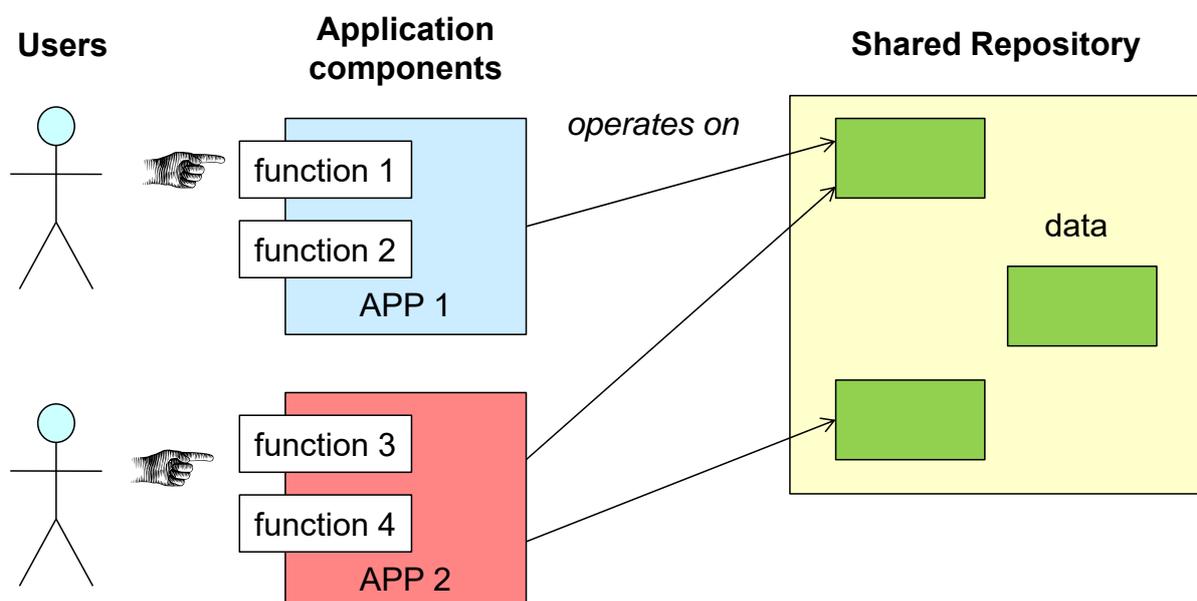
Altri pattern architetturali POSA

Luca Cabibbo ASW



Shared Repository

□ Struttura della soluzione



38

Altri pattern architetturali POSA

Luca Cabibbo ASW



Shared Repository

□ Discussione

- Shared Repository consente l'integrazione di elementi funzionali con un flusso di controllo guidato dai dati
- il repository potrebbe essere una base di dati o una collezione di oggetti in memoria
- i dati gestiti dal repository possono essere dei Domain Object
- l'accesso ai dati condivisi dovrebbe essere opportunamente sincronizzato
- può essere utile un meccanismo di notifica dei cambiamenti
- "il modo in cui l'architettura memorizza, manipola, gestisce e distribuisce informazioni" è certamente un interesse rilevante per le applicazioni data-intensive
- i componenti dell'applicazione non sono accoppiati direttamente tra loro – ma attenzione all'accoppiamento indiretto tramite il repository!

39

Altri pattern architetturali POSA

Luca Cabibbo ASW



Conseguenze

□ Modificabilità/evolubilità

- 😊 incoraggia un accoppiamento debole tra le varie applicazioni
- 😊 la struttura del repository può essere descritta da meta-dati
- 😊 può essere opportuno che i componenti vedano la struttura logica (e non fisica) del repository
- 😞 i componenti sono accoppiati tra di loro mediante la struttura del repository

□ Affidabilità

- 😊 possibilità di centralizzare funzionalità di backup/recovery

□ Sicurezza

- 😞 la centralizzazione dei dati può causare problemi di sicurezza

40

Altri pattern architetturali POSA

Luca Cabibbo ASW



* Discussione

- Questo capitolo ha presentato alcuni ulteriori pattern architetturali POSA comuni – che si concentrano su alcuni aspetti specifici dei sistemi software, come le interfacce grafiche e la gestione di dati condivisi e persistenti
 - questi pattern, che sono descritti a un livello generale, sono indipendenti dalle possibili implementazioni e piattaforme – e sicuramente anche da specifiche tecnologie