



Luca Cabibbo

Architettura dei Sistemi Software

È un pattern di importanza minore, non è neanche un pattern POSA ma noi lo utilizziamo nel progetto!
Va applicato BENE, nel senso che il progetto viene dato con questa architettura e le modifiche che vanno fatte non devono eliminare questo tipo di architettura.

Architettura esagonale

dispensa asw360
ottobre 2024

*There must be a cause why snowflakes
have the shape of six-cornered starlets.
It cannot be chance. Why always six?
Johannes Kepler*



- Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
 - Capitolo 20, **Architettura esagonale**
- ❑ Cockburn, A. **Hexagonal Architecture**. 2005.
 - <https://alistair.cockburn.us/hexagonal-architecture/>
- ❑ Cockburn, A. and Garrido de Paz, J.M. **Hexagonal Architecture Explained: How the Ports & Adapters architecture simplifies your life, and how to implement it**. Humans and Technology Press, v0.9b. 2024.
- ❑ Vernon, V. **Implementing Domain-Driven Design**. Addison-Wesley, 2013.
- ❑ Richardson, C. **Microservices Patterns: With examples in Java**. Manning, 2019.



- Obiettivi e argomenti

❑ Obiettivi

- presentare l'architettura esagonale

❑ Argomenti

- introduzione
- architettura a strati e inversione delle dipendenze
- architettura esagonale
- discussione



* Introduzione

❑ L'architettura esagonale è un pattern architetturale comune

- è diffuso soprattutto nel contesto di DDD e dell'architettura a microservizi
- in un sistema distribuito, consente di definire l'architettura di un singolo componente o servizio software, nonché di gestire le interazioni di questi componenti o servizi

Abbiamo parlato di architettura a strati, che applica il principio di separazione degli interessi e dice che interessi diversi sono rappresentati da strati diversi in modo che il codice non sia mischiato. In particolare l'architettura a strati viene spesso applicata come una decomposizione tecnica in modo da separare per esempio la presentazione dall'accesso ai dati e dalla logica di business. Però l'architettura a strati può essere applicata come decomposizione di primo livello (se viene fatta una decomposizione tecnica questa normalmente non è una buona idea) o di secondo livello (quindi gli elementi architetturali vengono internamente decomposti a strati).

L'architettura esagonale è una variante in un certo senso dell'architettura a strati in modo da consentire una decomposizione del codice ancora più adatta nell'ottica dell'architettura a servizi. In particolare nell'architettura a servizi e a microservizi si dice che gli elementi architetturali di primo livello sono i servizi e i microservizi però è utile fare una decomposizione del codice, che potrebbe essere fatta con l'architettura a strati ma viene fatta meglio con l'architettura esagonale. Utilizziamo questo pattern soprattutto per la decomposizione interna dei microservizi (del codice dei microservizi).

In particolare in un sistema distribuito permette di definire l'architettura interna di questo servizio software ma anche di gestire le interazioni esterne di questi servizi.



Responsabilità di business e infrastrutturali

□ Responsabilità degli elementi software

- **responsabilità di business** – o **logica di business** Di competenza dei componenti

- una o più funzionalità applicative Quindi relativo ad una o più funzionalità

- **responsabilità infrastrutturali** Di competenza dei connettori

- sono relative all'accesso ai servizi infrastrutturali

- un **servizio infrastrutturale** è un servizio tecnico (non applicativo) – ad es., un database o un servizio di comunicazione distribuita

- dipendono da tecnologie specifiche e richiedono l'utilizzo di framework tecnici opportuni

! È una responsabilità relativa all'ACCESSO ai servizi infrastrutturali, cioè servizi TECNICI, tipo base di dati, servizio di comunicazione in rete, servizio di scambio di messaggi, servizio per la cifratura di dati ecc (tendenzialmente servizi che acquistiamo)

I servizi infrastrutturali sono TECNICI e richiedono utilizzo di framework tecnici opportuni



Responsabilità di business e infrastrutturali

□ Come allocare le responsabilità di business e infrastrutturali agli elementi software? Intuitivamente

- le responsabilità di business competono ai “componenti”
- le responsabilità infrastrutturali competono ai “connettori”

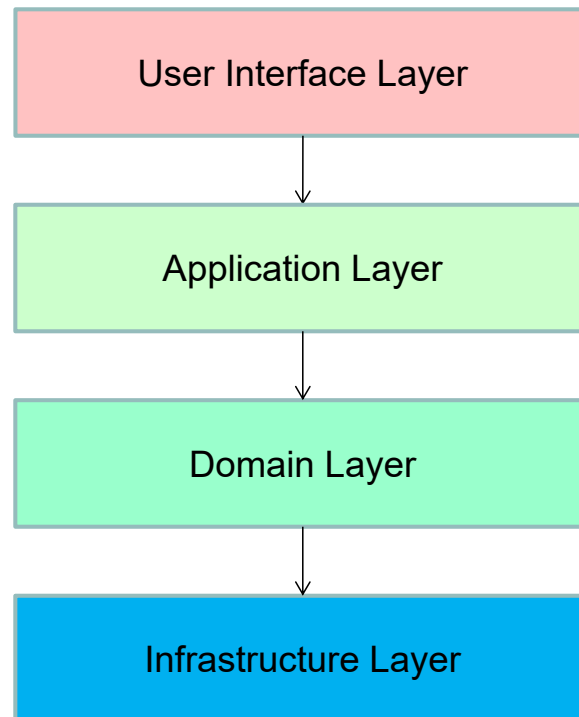
- le responsabilità di business dovrebbero essere disaccoppiate (non dovrebbero dipendere) da quelle infrastrutturali

Perché quelle di business sono di alto livello e quelle infrastrutturali sono di basso livello sempre con riferimento al principio di inversione delle dipendenze



* Architettura a strati e inversione delle dipendenze

- ❑ Consideriamo di nuovo l'architettura a strati "tradizionale" – ad es., la *Layered Architecture* di DDD



7

Architettura esagonale

Luca Cabibbo ASW



DIP e architettura a strati

- ❑ Consideriamo di nuovo anche il *principio di inversione delle dipendenze (DIP)*
 - i moduli di alto livello (più importanti) non dovrebbero dipendere dai moduli di basso livello (meno importanti) – piuttosto, entrambi dovrebbero dipendere da opportune astrazioni
 - le astrazioni non dovrebbero dipendere dai dettagli – piuttosto, i dettagli dovrebbero dipendere dalle astrazioni
 - l'architettura a strati tradizionale soddisfa il DIP?

8

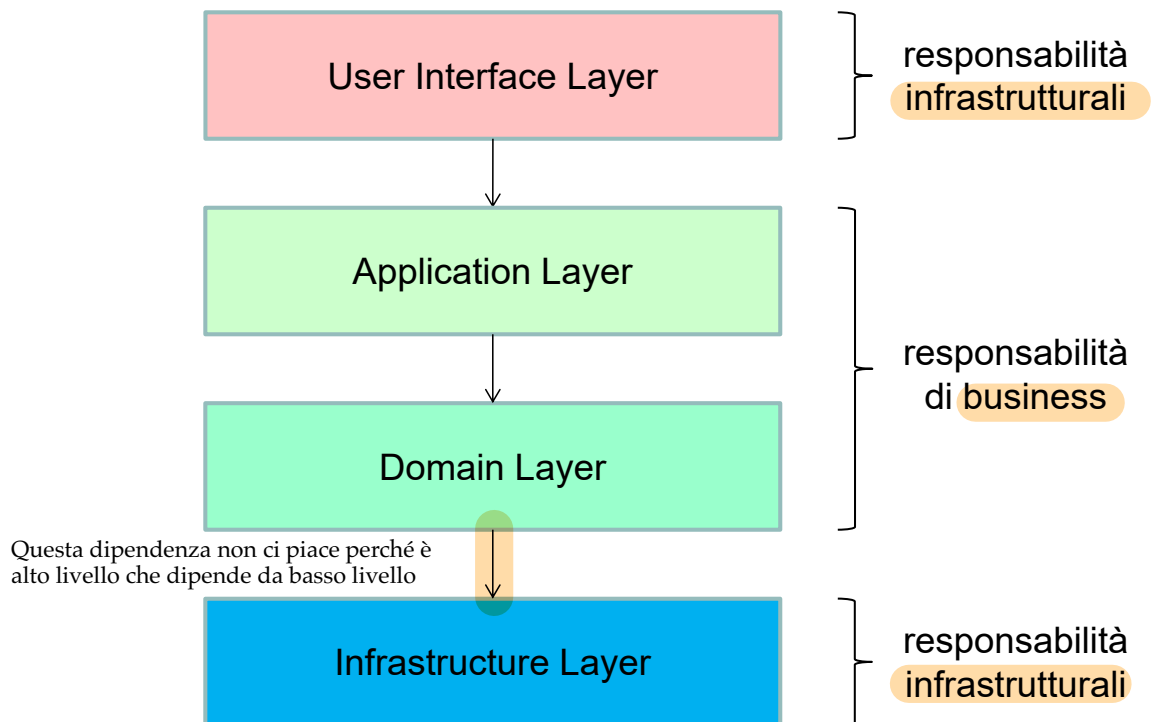
Architettura esagonale

Luca Cabibbo ASW



Architettura a strati e DIP

- L'architettura a strati tradizionale soddisfa il DIP?



9

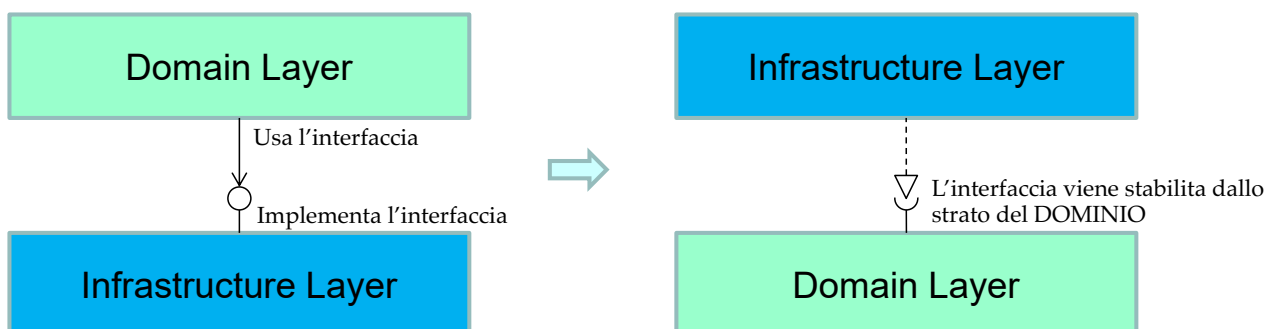
Architettura esagonale

Luca Cabibbo ASW



Architettura a strati e DIP

- L'architettura a strati tradizionale non soddisfa il DIP
 - è però possibile applicare il DIP, introducendo delle “opportune astrazioni”



- l'idea è definire delle interfacce per le responsabilità infrastrutturali negli strati di business in cui queste responsabilità sono richieste e utilizzate – e implementarle nello strato dell'infrastruttura
 - ad es., per l'accesso ai dati persistenti, oppure per l'invio di messaggi su un canale per messaggi

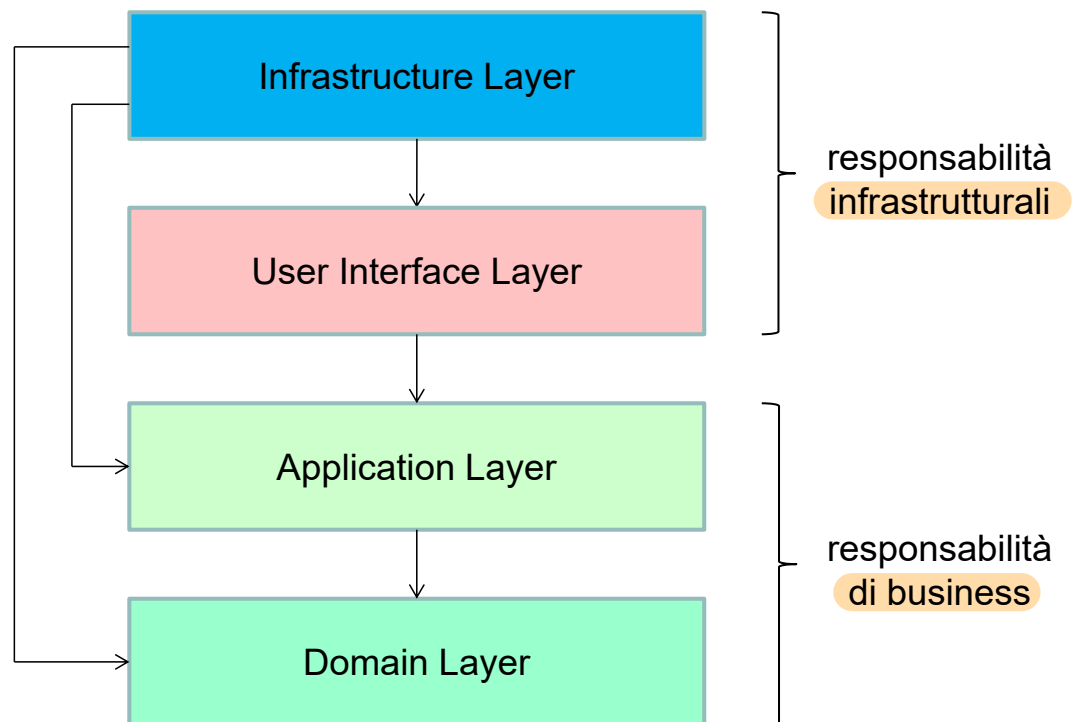
10

Architettura esagonale

Luca Cabibbo ASW



Architettura a strati e DIP



11

Architettura esagonale

Luca Cabibbo ASW



Verso l'architettura esagonale

- L'architettura esagonale, intuitivamente
 - 1 ■ ha origine nell'architettura a strati
 - 2 ■ a cui però viene prima applicato il DIP
 - 3 ■ e poi viene rimossa l'asimmetria sopra-sotto dell'architettura a strati

È solo un'interpretazione grafica: se si ha solo un sopra-sotto ci sono solo due versi in cui muoversi, mentre l'architettura esagonale vuole modificare questo vincolo

12

Architettura esagonale

Luca Cabibbo ASW



* Architettura esagonale

min 16

- ❑ Il pattern architetturale **Hexagonal Architecture** (*architettura esagonale*) – chiamato anche, in modo più descrittivo, **Ports and Adapters**
 - nel contesto di un sistema/software distribuito, aiuta a organizzare un singolo componente distribuito – che chiamiamo un **servizio** (*servizio applicativo*) o **componente esteso** ti ascolta
 - in pratica, ogni servizio ha sia responsabilità di business che responsabilità di presentazione e infrastrutturali
 - questo pattern sostiene
 - un accoppiamento debole tra le responsabilità di business e le altre responsabilità
 - un'interazione flessibile tra i servizi – tra di loro e con altre entità esterne

Componente tecnologico

13

Architettura esagonale

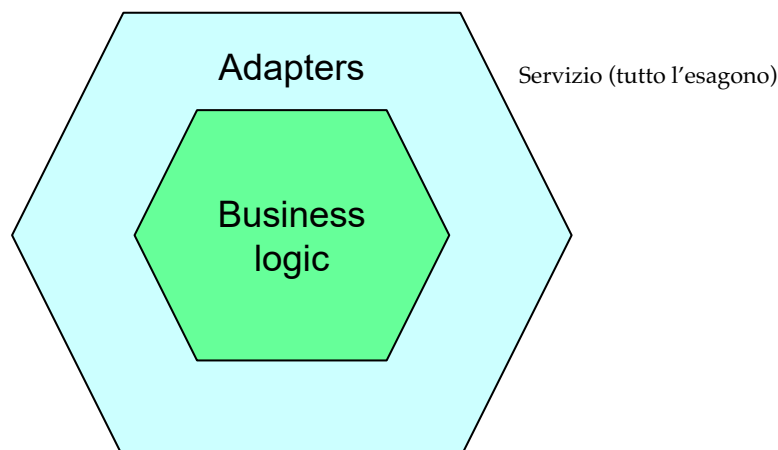
Luca Cabibbo ASW



Architettura esagonale

- ❑ Ogni servizio è rappresentato da un “esagono”
 - ogni servizio/componente esteso/esagono è inoltre suddiviso in due parti
 - **interno** (*inside*) – responsabilità di business Logica di business, componente
 - **esterno** (*outside*) – responsabilità di presentazione e infrastrutturali Adattatore, connettore

L'interno è la parte più importante perché ha valore di business, la parte fuori è meno importante quindi vogliamo che l'interno ne sia indipendente, vogliamo “proteggere” l'interno dall'esterno



14

Architettura esagonale

Luca Cabibbo ASW



Architettura esagonale

- ❑ L'**interno** di un servizio (**logica di business**) – chiamato anche **application**, **app** o **core** – implementa un insieme di **funzionalità** e responsabilità di business
 - implementa gli strati Domain e Application del servizio applicativo
 - **non** implementa responsabilità di presentazione o infrastrutturali
 - tuttavia le supporta, definendo delle interfacce utili per la presentazione e per l'accesso ai servizi infrastrutturali
- in pratica, l'interno è la parte del servizio attorno a cui vogliamo mettere dei confini
 - dentro c'è tutta e sola la logica di business
 - fuori c'è tutta e sola la tecnologia legata all'uso dei servizi infrastrutturali



Architettura esagonale

- ❑ Il servizio (la sua **logica di business**) ha la necessità di **interagire** con **diversi tipi di entità esterne** – chiamati anche **attori** Attore è tutto ciò che è esterno al servizio
 - ad es., gli utenti, i test automatizzati e la base di dati, oppure anche altri servizi o servizi di altre organizzazioni
 - si vuole che la logica di business possa interagire con altre entità esterne
 - con un accoppiamento basso – evitando l'intreccio tra la logica di business e le interazioni con le **entità esterne**
 - in modo flessibile

Sia altri servizi, che
utenti che servizi
infrastrutturali



Porte e adattatori

- La logica di business interagisce con le entità esterne tramite porte e adattatori

- ciascuna **porta** (definita nella logica di business) definisce un'interfaccia (fornita o richiesta) che rappresenta una modalità di interazione con la logica di business, con uno scopo specifico

Le porte sono definite all'interno della logica di business, all'interno dell'esagono, e sono intuitivamente delle interfacce che possono essere richieste o fornite. Ogni porta ha una finalità specifica e rappresenta una modalità specifica di interagire col servizio. Le interfacce fornite sono implementate dall'interno del servizio, sono dette porte inbound ma anche primary port perché possono essere usate dagli attori principali che sono gli utenti. Le interfacce richieste sono dette outbound o secondary port, rappresentano qualcosa di esterno che il servizio può richiedere

- due tipi di porte
 - **inbound port** – un'interfaccia fornita
 - è chiamata anche una **primary port** – perché consente ad un attore primario esterno di usare le funzionalità della logica di business
 - **outbound port** – un'interfaccia richiesta
 - è chiamata anche una **secondary port** – perché consente alla logica di business di usare le funzionalità di attore di supporto (secondario) esterno

17

Architettura esagonale

Luca Cabibbo ASW



Porte e adattatori

- La logica di business interagisce con le entità esterne tramite porte e adattatori

- esempi di porte per un servizio S
 - porte per interagire con i suoi utenti (sul web) e per ricevere chiamate remote da altri servizi Inbound
 - porte per accedere alla base di dati e per effettuare chiamate remote ad altri servizi ↗
Porta per parlare con la base di dati. Una porta di questo tipo è detto repository, è una porta outbound
 - porte per lo scambio asincrono di messaggi

Es porte che supportano chiamate rest ↖

Sono inbound o outbound a seconda che ricevano o inviino messaggi

Tutte le porte sono scritte ALL'INTERNO dell'esagono e quindi sono scritte in modo da non dipendere da tecnologie esterne

18

Architettura esagonale

Luca Cabibbo ASW



Porte e adattatori

C'è bisogno di chi implementi le interfacce richieste e di chi fa le chiamate al servizio tramite le sue porte inbound. Questo viene fatto dagli adattatori, che implementano le responsabilità infrastrutturali, e hanno lo scopo di adattare richieste che vengono dall'esterno del servizio in richieste accettate all'interno del servizio (inbound) o di trasformare le richieste interne in richieste verso l'esterno (outbound).

- ❑ La logica di business interagisce con le entità esterne tramite porte e adattatori
 - intuitivamente, gli adattatori implementano lo strato Presentation e lo strato Infrastructure del servizio
 - ciascun **adattatore** (definito nell'esterno del servizio) è relativo a una specifica porta, ed ha lo scopo di adattare le interazioni tra quella porta e un'entità esterna, utilizzando una tecnologia specifica
 - due tipi di adattatori – a seconda del tipo di porta a cui si riferiscono
 - **inbound adapter** – chiamato anche **primary adapter**
 - **outbound adapter** – chiamato anche **secondary adapter**
- Hanno la funzione di trasformare richieste che vengono dall'esterno del servizio in richieste che vengano accettate all'interno del servizio
- Hanno la funzione di trasformare richieste che vengono dall'interno del servizio in richieste verso l'esterno del servizio

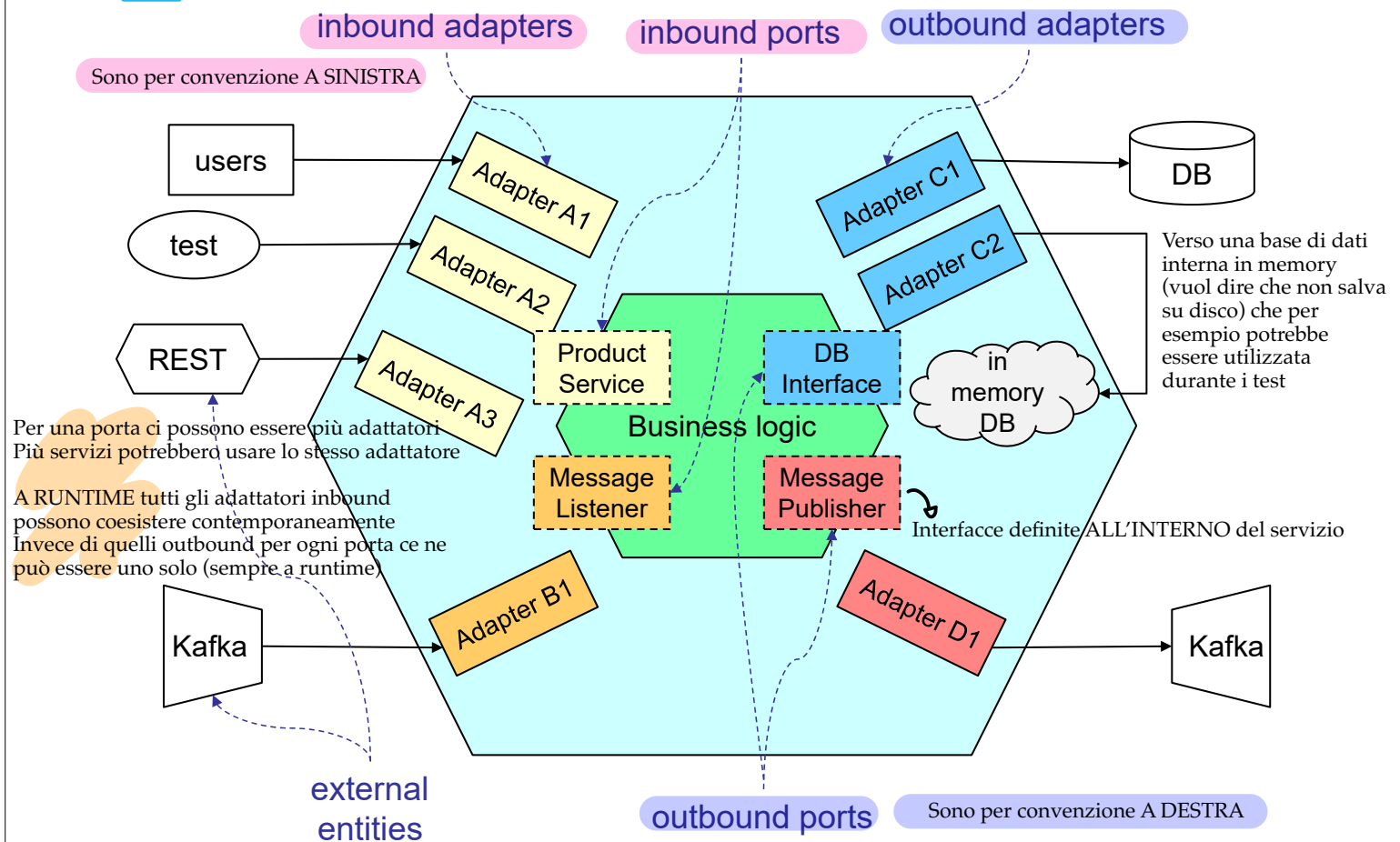


Porte e adattatori

- ❑ La logica di business interagisce con le entità esterne tramite porte e adattatori
 - esempi di adattatori per un servizio S
 - un controller Spring Web MVC
 - un adattatore per l'accesso a una base di dati MySQL – un altro per l'accesso a una base di dati inmemory
 - adattatori per ricevere ed effettuare chiamate REST – e altri per chiamate gRPC
 - adattatori per inviare e ricevere messaggi con Kafka



Architettura esagonale



21

Architettura esagonale

Luca Cabibbo ASW



Porte

- Una **porta** rappresenta una modalità di interazione con il servizio (con la sua logica di business), con uno scopo specifico
 - una nozione **volutamente generica e flessibile**
 - le porte sono rappresentate graficamente dai lati dell'esagono
 - ma non vuol dire che ogni servizio debba avere sei porte
 - ogni porta è in corrispondenza con un'interfaccia interna (un'API) della logica di business
 - ad es., in DDD potrebbero essere dei service, repository e application service

22

Architettura esagonale

Luca Cabibbo ASW



Adattatori

Servono ad adattare richieste che vengono dall'esterno del servizio verso l'interno del servizio. Interagiscono con l'esterno attraverso tecnologie specifiche e devono trasformare le richieste in richieste verso l'interno che NON dipendano da tecnologie specifiche

- ❑ Ciascun **adattatore** è relativo all'interazione con una specifica tipologia di entità esterna – adatta il tipo di interazione richiesto dall'entità esterna all'API interna dell'applicazione, e viceversa, utilizzando una tecnologia specifica
 - ad esempio
 - un controller web MVC, l'implementazione di un repository JPA, un controller REST, un endpoint per messaggi Kafka
 - per ciascuna porta ci possono essere più adattatori
 - ciascun adattatore può essere utilizzato anche da più entità esterne differenti




Scenari

- ❑ Alcuni scenari principali

1  un'entità esterna interagisce con un servizio attraverso un adattatore specifico, relativo a una porta specifica

Interazione
da fuori

- l'adattatore converte questo evento esterno in un'invocazione di un'operazione o in un messaggio, e lo passa alla logica di business tramite l'interfaccia di quella porta (inbound)

2  la logica di business di un servizio deve interagire con l'esterno

Interazione
verso fuori

- la logica di business interagisce con l'interfaccia di una porta (outbound) – l'adattatore associato a quella porta converte la richiesta o il messaggio in un formato appropriato per l'entità esterna
- in entrambi i casi, la logica di business è indipendente sia dalla natura dell'entità esterna che dagli specifici adattatori utilizzati e dalle tecnologie sottostanti

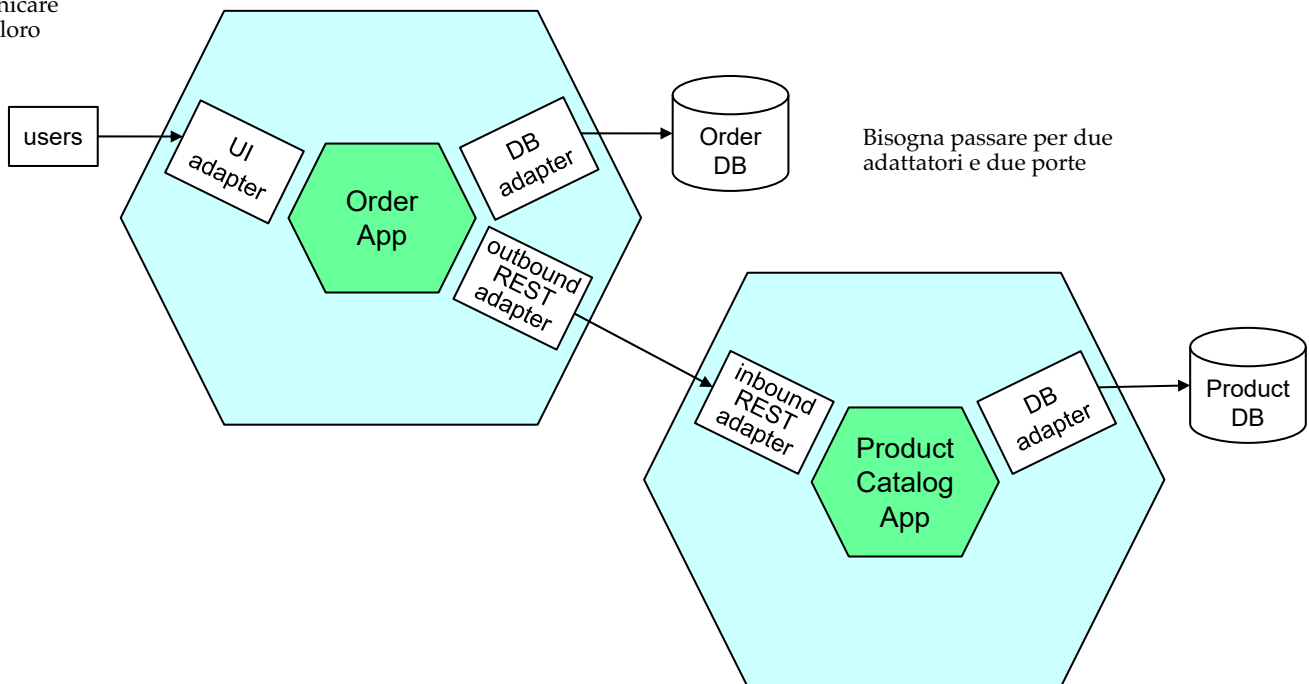


Scenari

Alcuni scenari principali

3 un sistema software distribuito composto da più servizi, che interagiscono tra di loro mediante porte e adattatori

Due servizi vogliono comunicare tra di loro



25

Arquitettura esagonale

Luca Cabibbo ASW



Conseguenze

L'architettura esagonale ha impatto positivo su diverse qualità

modificabilità e flessibilità

- 😊 l'accoppiamento debole tra l'interno e l'esterno di un servizio sostiene la possibilità di implementare e di far evolvere la logica di business in modo indipendente dagli adattatori
- 😊 l'indirezione fornita dagli adattatori consente un'interazione flessibile di un servizio con altre entità esterne e con altri servizi
- 😊 l'architettura esagonale è compatibile con team cross-funzionali

26

Arquitettura esagonale

Luca Cabibbo ASW



Conseguenze

- ❑ L'architettura esagonale ha impatto positivo su diverse qualità
 - verificabilità
 - 😊 l'ampio uso di interfacce sostiene l'utilizzo di test double
 - 😊 è possibile verificare separatamente la logica di business e gli adattatori
 - 😊 è possibile usare adattatori specifici per i test (ad es., per un inmemory database)
 - interoperabilità
 - 😊 grazie alle indirezioni fornite dagli adattatori
 - prestazioni
 - 😞 possono essere penalizzate dall'uso degli adattatori



Architettura esagonale e Layers

- ❑ Abbiamo detto che l'architettura esagonale ha origine nell'architettura a strati
 - ogni servizio è internamente basato su due “strati”
 - sostiene un isolamento delle responsabilità di business dalle responsabilità di presentazione e infrastrutturali
 - c'è però un'importante differenza con l'architettura a strati
 - nell'architettura a strati, gli strati possono essere allocati a team di sviluppo separati (team mono-funzionali)
 - nell'architettura esagonale, sono i servizi applicativi che vengono allocati a team di sviluppo separati (team cross-funzionali, che si occupano di interi servizi)
- Cioè l'intero esagono viene assegnato al team che quindi deve essere cross-funzionale



Architettura esagonale, servizi, componenti e connettori

- Nell'architettura esagonale, in un servizio
 - la logica di business (interno) ha responsabilità funzionali – da “componente”
 - gli adattatori (esterno) hanno responsabilità infrastrutturali – da “connettori”
 - viene dunque ancora effettuata una decomposizione tra componenti e connettori
 - si tratta però di una **decomposizione di secondo livello** – perché la decomposizione di primo livello è quella relativa ai servizi
 - un intero servizio viene anche chiamato “componente esteso” – perché comprende la sua logica di business (“componente”) insieme agli adattatori, che “estendono” il “componente” per renderlo un elemento software distribuito autonomo

29

Architettura esagonale

Luca Cabibbo ASW



Un ulteriore elemento – il configuratore

- Nell'architettura esagonale, c'è anche bisogno di un **configuratore** per connettere tutti gli elementi di un servizio – per connettere la logica di business con gli adattatori, tramite le porte
 - questo configuratore deve in genere
 - istanziare la logica di business
 - istanziare gli adattatori utilizzati
 - collegare ogni adattatore inbound con la corrispondente porta inbound
 - collegare ogni porta outbound con il corrispondente adattatore outbound
 - ad esempio, come configuratore è possibile usare un framework per l'iniezione delle dipendenze come Spring

All'avvio del servizio (cioè a runtime) crea il servizio, gli adattatori e collega gli adattatori con le porte (considerando il vincolo sulle porte outbound a runtime, vedi sopra)

30

Architettura esagonale

Luca Cabibbo ASW



- Architettura esagonale nelle esercitazioni

- ❑ Ciascun servizio applicativo (ad es., **restaurant-service**) viene strutturato con l'architettura esagonale, usando i seguenti package
 - un package di base del servizio – ad es., **asw.efood.restaurant-service**
 - un package che definisce l'interno (logica di business) del servizio, comprese tutte le sue porte – ad es., **asw.efood.restaurant-service.domain**
 - contiene le entità, i service e i repository (le interfacce) – service e repository sono porte
 - un package separato per ciascun adattatore – ad es.,
 - **asw.efood.restaurant-service.rest** definisce l'adattatore REST per ricevere invocazioni remote da altri servizi
 - **asw.efood.restaurant-service.accounting-client.rest** definisce l'adattatore REST per effettuare invocazioni remote al servizio **accounting-service** tramite REST

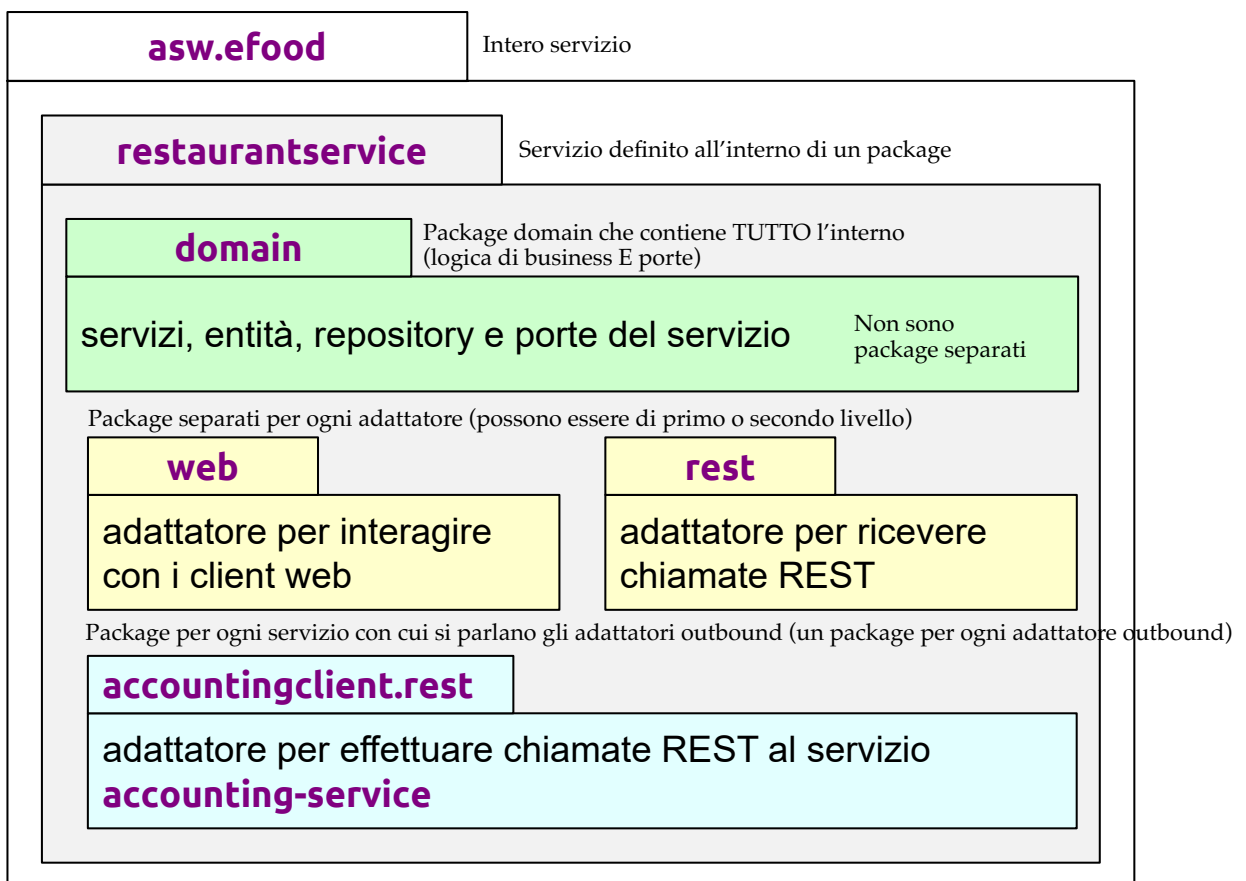
31

Architettura esagonale

Luca Cabibbo ASW



Architettura esagonale nelle esercitazioni



32

Architettura esagonale

Luca Cabibbo ASW



* Discussione

□ L'architettura esagonale

- nell'ambito di un servizio, sostiene un isolamento delle responsabilità di business da quelle di presentazione e infrastrutturali
- può essere applicata nella realizzazione di sistemi distribuiti – ad es., nei microservizi
- è compatibile con DDD – ogni esagono può essere utilizzato per implementare un Bounded Context
- è un'architettura flessibile, compatibile con altri pattern – come l'architettura a servizi, i microservizi, gli eventi di dominio e CQRS