Luca Cabibbo

Architettura dei Sistemi Software

Messaging

dispensa asw460

ottobre 2024

Asynchronous messaging architectures are powerful but require us to rethink our development approach.

Gregor Hohpe and Bobby Woolf

- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 26, Messaging
- [POSA4] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. **Pattern-Oriented Software Architecture (vol. 4)**: A Pattern Language for Distributed Computing. John Wiley & Sons, 2007
- [EIP] Hohpe, G. and Woolf, B. **Enterprise Integration Patterns**: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2004.
  - https://www.enterpriseintegrationpatterns.com/

- Obiettivi e argomenti

- Obiettivi
  - presentare il pattern architetturale Messaging per la comunicazione asincrona
  - presentare ulteriori pattern di supporto alla comunicazione asincrona
- Argomenti
  - introduzione
  - Messaging [POSA4]
  - pattern di supporto al Messaging
  - discussione

* Introduzione

- La comunicazione asincrona è supportata dai pattern architetturali *Messaging* [POSA4] e *Publisher-Subscriber* [POSA]
  - tra questi due pattern POSA ci sono molti punti in comune – ma anche alcune differenze (che però sono meno importanti dei punti in comune)
  - qui li presentiamo in modo unificato, come se fossero un singolo pattern architetturale – che chiamiamo *Messaging* [POSA4]

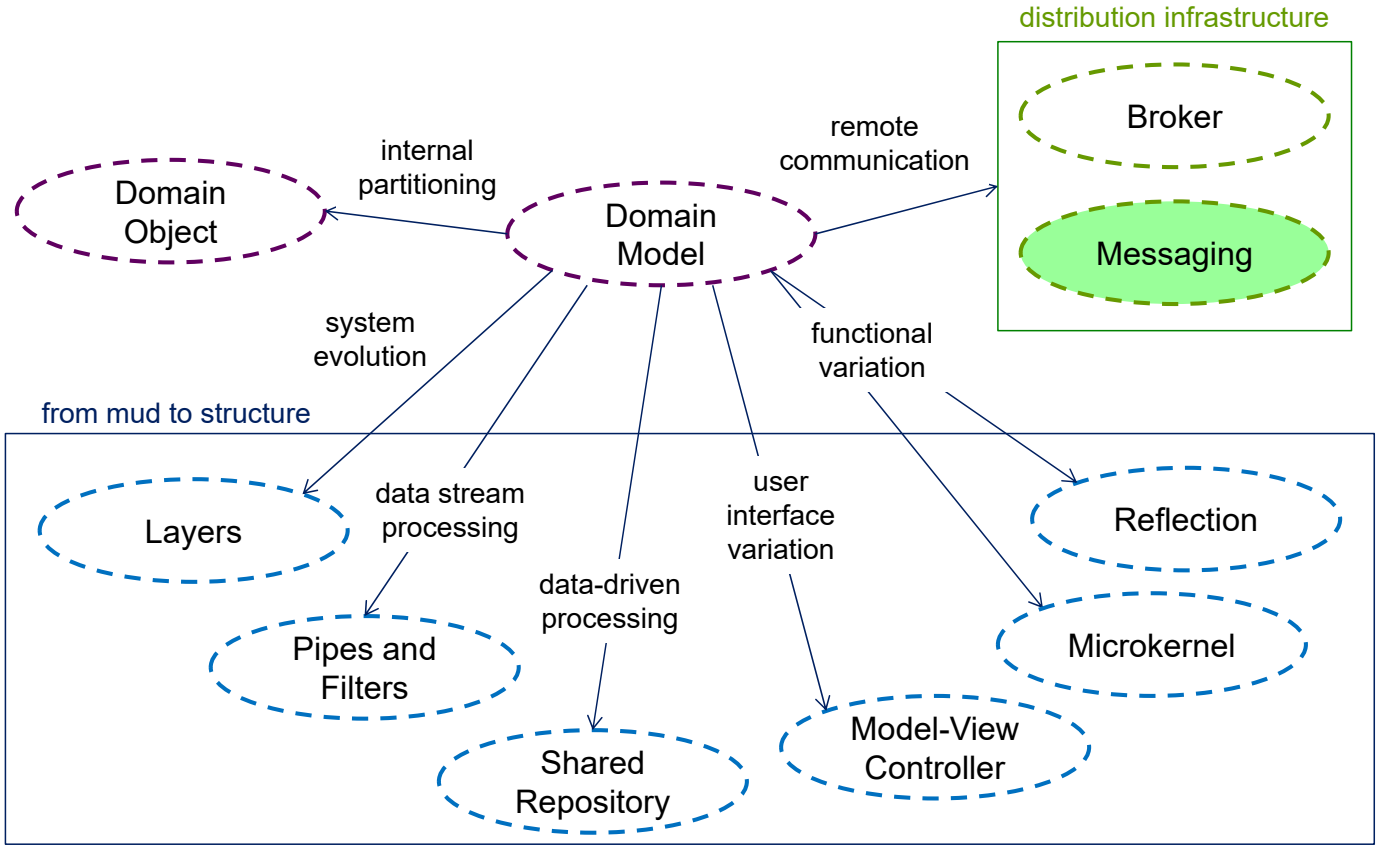
Messaging [POSA4]

- *Messaging* è un pattern architetturale [POSA4] della categoria "distribution infrastructure"
  - supporta lo stile di comunicazione della comunicazione asincrona
    - definisce un'infrastruttura che abilita la comunicazione asincrona
    - si occupa degli obiettivi e dell'applicazione della comunicazione asincrona nei sistemi software distribuiti

Broker parla dell'invocazione remota, messaging parla della comunicazione asincrona. Broker dice come funziona un broker che è un middleware per l'invocazione remota. Messaging fa due cose: dice come funziona un messaging broker che è il middleware e parla di un modo particolare ma importante di utilizzare la comunicazione asincrona. Sarebbe il caso forse di avere due pattern distinti, ognuno che parli di una cosa (come funziona il broker/ come utilizzare la comunicazione asincrona). In realtà in (libro) ci sono due pattern, ma ognuno parla delle due cose mischiandole. Allora nel corso queste due cose vengono presentate insieme, invece di separarle in modo non accurato.

L'obiettivo è sostenere lo sviluppo di sistemi basati sulla comunicazione asincrona. Questo pattern per messaging riguarda:
Come funziona la comunicazione asincrona
Come può essere utilizzata la comunicazione asincrona

Relazione con altri pattern [POSA]



Messaging

Vantaggi della comunicazione asincrona visti nella scorsa lezione:
Riduzione di accoppiamento (perché comunicazione è indiretta, asincrona e basata su abstract common services)
Supporto ad altre qualità: prestazioni, scalabilità, affidabilità

- Intuitivamente, il pattern Messaging organizza un sistema distribuito come un insieme di componenti che interagiscono tramite lo scambio di messaggi
  - i messaggi possono contenere dati, informazioni, metadati, richieste, risposte, informazioni di errore, notifiche di eventi, …
  - una comunicazione flessibile, in modalità molti-a-uno (point-to-point) oppure molti-a-molti (publish-subscribe)
  - lo scambio di messaggi avviene in modo indiretto e asincrono
  - questo stile di comunicazione rilassa accoppiamento e tipizzazione
  - complessivamente, sostiene la comunicazione asincrona e cerca di semplificarne l'utilizzo

Pattern di supporto al Messaging

Posa dedica un intero capitolo a questo argomento, e nel capitolo oltre a messaging e pub-sub ci sono altri pattern ma con una portata minore, sono pattern di supporto all'applicazione del messaging

- Il pattern architetturale Messaging è solo il "punto di ingresso" ai pattern [POSA4] per la comunicazione asincrona
  - [POSA4] presenta anche degli ulteriori pattern, di supporto a questo pattern architetturale principale, che affrontano alcuni problemi ricorrenti nell'utilizzo della comunicazione asincrona, e ne propongono delle soluzioni coerenti
  - la progettazione dei sistemi basati sulla comunicazione asincrona può essere così guidata da tutti questi pattern

* Messaging [POSA4]

- Il pattern architetturale **Messaging** – proposto inizialmente da [EIP], poi ripreso da [POSA4], nella categoria “distribution infrastructure”
  - definisce un’infrastruttura per la comunicazione basata sullo scambio di messaggi
  - per sostenere la comunicazione o l’integrazione in un sistema coerente di componenti debolmente accoppiati o indipendenti

Da definire dopo

Messaging

- Contesto

Non è l’unico contesto in cui si può applicare ma è quello che propone il libro

  - un sistema distribuito, in cui ci sono un insieme di componenti debolmente accoppiati o indipendenti

Cioè il componente sa quasi niente di un altro ma sa almeno della sua esistenza, oppure il componente non sa neanche che ne esistano altri

I componenti già ci sono, devo farli interagire. In effetti messaging può essere applicato anche se i componenti non ci sono

  - è necessaria un’infrastruttura di comunicazione flessibile, per abilitare la comunicazione tra questi componenti e la loro integrazione

Fare in modo che i diversi componenti messi insieme forniscano della capacità superiori a quelle fornite dai singoli componenti. È più forte del termine “composizione” dei componenti, e l’aspetto fondamentale nell’integrazione è l’indipendenza dei componenti iniziali

Messaging

- **Problema** Ricordati che nel problema non si parla della soluzione
  - alcuni sistemi distribuiti sono composti da componenti debolmente accoppiati e che devono operare in modo indipendente – al limite, sono completamente autonomi
  - è necessario far interagire e collaborare o integrare questi componenti

Nota che nella descrizione del problema dice propagare informazioni, non messaggi. Non si cita la soluzione

Questo punto è un altro problema

  - alcune informazioni devono poter essere propagate tra componenti – da componenti produttori (*producer* o *publisher*) a componenti consumatori (*consumer* o *subscriber*) – in modo che ogni componente possa reagire in modo appropriato alle informazioni di proprio interesse
  - l'interazione tra i componenti
    - deve essere flessibile Facilmente modificabile
    - deve essere basata su un accoppiamento debole
    - deve avvenire in modo affidabile

Messaging

- **Soluzione**

Rivedi la definizione di "bus" come broker

  - collega i componenti mediante un *bus per messaggi*, che consente uno scambio *asincrono* di *messaggi* tra i componenti
  - i componenti registrano presso il bus i tipi di messaggi di loro interesse
  - il bus per messaggi si occupa di inoltrare ciascun messaggio inviato sul bus dal suo produttore ai consumatori interessati a questo messaggio, in modo affidabile
    - l'inoltro è basato sulle registrazioni dei componenti ai tipi di messaggi di loro interesse

Messaging

- Soluzione
  - i messaggi possono incapsulare informazioni, strutture di dati o notifiche di eventi
  - i consumatori di messaggi possono usare le informazioni nei messaggi per guidare o coordinare le proprie computazioni
  - codifica i messaggi in modo che i componenti possano comunicare in modo debolmente accoppiato
  - realizza il collegamento tra i componenti anche mediante l'ausilio di ulteriori elementi software

Il sistema è formato da componenti e connettori. Quindi abbiamo parlato di componenti e messaggi, chi dovrà scambiare questi messaggi saranno proprio i connettori.
Intuitivamente, soprattutto se devo fare integrazione, questi connettori fungeranno da collante nella connessione tra diversi componenti, che non dovranno essere modificati per poter comunicare e per poter scambiare messaggi, da qui anche il mantenimento dell'accoppiamento debole.

Discussione

- Alcuni chiarimenti sul pattern architetturale Messaging
  - il problema affrontato è, in generale, di definire un'infrastruttura di comunicazione flessibile tra componenti debolmente accoppiati O al limite completamente indipendenti
  - in modo più specifico, questo pattern può essere applicato anche per supportare l'integrazione di componenti indipendenti o autonomi – e che devono rimanere tali Se un componente non conosce l'esistenza dell'altro non dovrà conoscerla neanche una volta implementata la comunicazione asincrona
    - ad es., comporre i componenti di un insieme di applicazioni esistenti per realizzare una nuova applicazione, con un maggior valore di business
    - quest'ultimo argomento è discusso in una successiva dispensa

Discussione

- Alcuni chiarimenti sul pattern architetturale Messaging
  - un'idea centrale della soluzione è di realizzare le interazioni tra i componenti di interesse sulla base dello scambio asincrono di messaggi, tramite un bus per messaggi
    - la comunicazione asincrona sostiene un accoppiamento debole tra i componenti – ma anche flessibilità e affidabilità nella comunicazione
    - i (tipi di) messaggi e i canali per messaggi (che formano il bus per messaggi) possono essere progettati applicando i pattern di supporto al Messaging

Non si parla di tipi di messaggi e canali per messaggi nella soluzione. Vedremo che se ne parlerà nei pattern di supporto

  - i diversi componenti si scambiano messaggi per guidare o coordinare le proprie computazioni
    - ad es., i messaggi possono codificare eventi – da propagare da componenti publisher a componenti subscriber, affinché questi ultimi possano reagire di conseguenza

Discussione

- Alcuni chiarimenti sul pattern architetturale Messaging
  - un altro aspetto fondamentale della soluzione è la parola iniziale "collega"
  - questo "collegamento" tra i componenti viene realizzato mediante l'introduzione di ulteriori elementi software (connettori), che fungono da "collante" tra i componenti preesistenti
    - di solito, i messaggi vengono scambiati dai nuovi elementi "collante" – e non "direttamente" dai componenti preesistenti
    - questi ulteriori elementi "collante" possono essere progettati applicando i pattern di supporto al Messaging

Conseguenze

- Benefici
  - ☺ modificabilità e flessibilità
  - ☺ prestazioni
  - ☺ affidabilità
- Inconvenienti
  - ☹ complessità
  - ☹ prestazioni – l'overhead può penalizzare le prestazioni
  - ☹ verifica e affidabilità

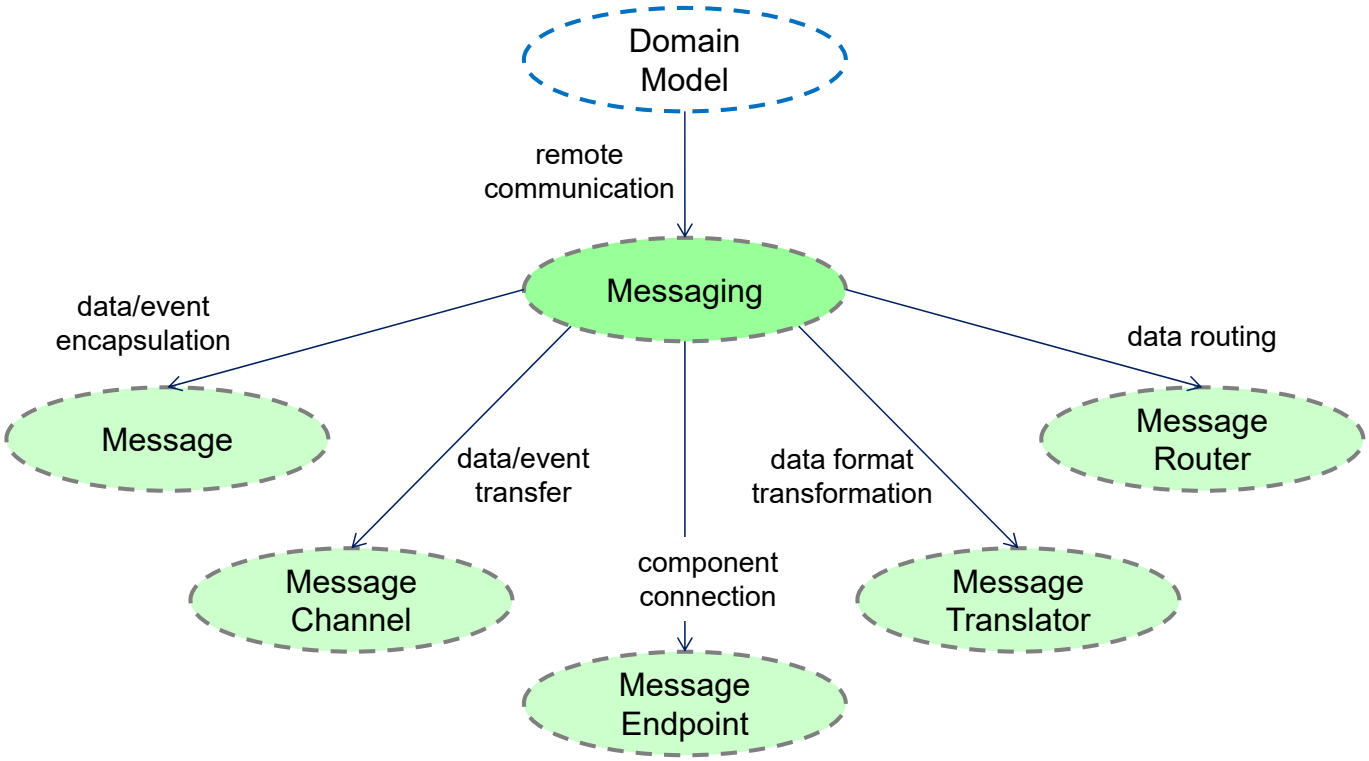
* Pattern di supporto al Messaging

- L'applicazione del Messaging è favorita da ulteriori pattern di supporto, che affrontano in modo coerente alcuni problemi ricorrenti della comunicazione asincrona
  - la comunicazione avviene mediante lo scambio di *messaggi*
  - la comunicazione avviene tramite *canali per messaggi*
  - i componenti vengono collegati ai canali mediante dei *message endpoint*
  - è anche possibile avere degli ulteriori componenti aggiuntivi – ad es., per la *trasformazione* o il *routing* di messaggi
  - questi pattern hanno, tra l'altro, lo scopo di mantenere basso l'accoppiamento tra i componenti

Vedere il Message come un pattern ci aiuta perché ci descrive non solo la soluzione ma anche il problema

Quindi non in modo indistinto

Pattern di supporto al Messaging



- Message [EIP]

- Problema
  - consentire uno scambio flessibile di informazioni tra componenti – per comunicare, ad es., dati, documenti, eventi o comandi

Bisogna sostenere la flessibilità perché questi componenti per realizzare l'integrazione potrebbero doversi scambiare vari tipi di dati

- Soluzione
  - incapsula le informazioni da scambiare in un messaggio (*message*)
  - il messaggio è formato da
    - un body (o payload) – contiene i dati effettivi
    - un header – specifica ulteriori metadati circa i dati trasmessi

Abbiamo detto per esempio che in alcuni casi i messaggi hanno una scadenza (ad es il messaggio è "voglio trasmettere è "è cambiata la quotazione di questa azione". L'evento che voglio comunicare fa parte del corpo, ma decidere che questa informazione è utile per 2,5,45 minuti, lo devo mettere nell'header; oppure l'identificativo di una transazione, o un tipo di certificato che va verificato prima dell'elaborazione, sono informazioni che vanno nell'header). In generale le informazioni che servono a supportare le qualità dell'applicazione sono nell'header

Message

- Conseguenze
  - \+ consente il trasferimento di strutture di dati, documenti ed eventi tra componenti
  - \+ sostiene un accoppiamento debole
  - \+ flessibilità – soprattutto se i messaggi sono autodescrittivi
  - – overhead nella codifica e decodifica dei messaggi

Perché conoscere il formato del messaggio è considerato un accoppiamento più debole del dover conoscere l'intera interfaccia di un altro componente per utilizzare dei servizi (cioè non solo le operazioni ma anche i parametri e i tipi di dati dei parametri)

Message Channel [POSA4]

- Problema
  - i messaggi contengono solo i dati che devono essere scambiati tra componenti
  - per sostenere un accoppiamento debole, i componenti non dovrebbero conoscere chi è interessato a quali messaggi
  - è comunque necessario un meccanismo per consentire lo scambio di messaggi tra componenti

Nè il tipo, né l'identità remota ecc non devono sapere nulla dell'altro componente

- Soluzione
  - non connettere direttamente i componenti che devono interagire – ma collegali tramite un canale per messaggi (*message channel*)

Message Channel

- Discussione
  - un canale per messaggi è un “indirizzo logico” presso cui i componenti e servizi possono inviare e ricevere messaggi
  - è comune l’uso di molti canali per messaggi
  - ci sono diversi tipi comuni di canali per messaggi – ad esempio
    - *canali punto-punto* (*point-to-point channel*)
    - *canali publish-subscribe* (*publish-subscribe channel*)
  - altri tipi di canali rappresentano dei modi comuni di usare i canali – ad esempio
    - un *canale per messaggi non validi* (*invalid message channel*) – per disaccoppiare la gestione di messaggi errati dal resto della logica applicativa

Message Channel

- Conseguenze
  - \+ sostiene un accoppiamento debole
  - \+ è possibile assegnare a un canale la responsabilità per alcuni attributi di qualità Es persistenza dei messaggi, durata…
  - – la gestione di un message channel richiede memoria e risorse di rete, nonché eventualmente anche memoria persistente

- Message Endpoint [POSA4]

- Problema
  - si vogliono far comunicare, mediante lo scambio di messaggi, componenti autonomi – ma una loro comunicazione diretta non è possibile o desiderabile
  - in ogni caso, si vuole sostenere un accoppiamento debole tra questi componenti e con la soluzione tecnologica
    - tra i componenti e la soluzione tecnologica di messaging Kafka, …
    - tra i componenti stessi
  - è però necessario abilitare questi componenti allo scambio di messaggi

Message Endpoint

- Soluzione

Cosa
- connetti i componenti con l'infrastruttura di comunicazione mediante dei connettori chiamati *message endpoint* ("estremità" per messaggi) che gli consentono di scambiare messaggi

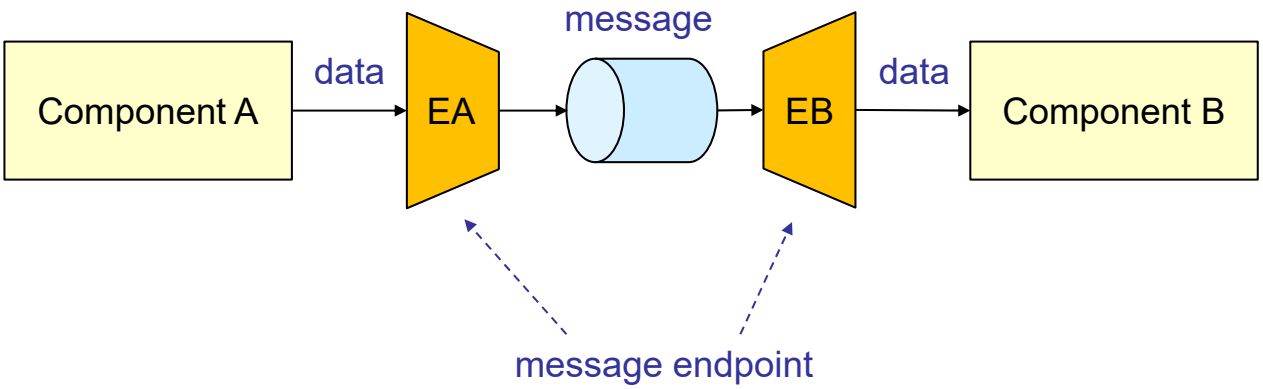
Come
- quando un componente deve comunicare dei dati a un altro componente, li passa al suo message endpoint – che invia un messaggio a un canale per messaggi
- questo messaggio non viene ricevuto direttamente dal componente destinatario, ma da un altro message endpoint – che ne estrae i dati e li passa al componente che li può elaborare

I componenti quindi non sanno che tra loro c'è uno scambio di messaggi, loro vedono solo i dati

- talvolta, non sono i componenti che comunicano direttamente con i loro message endpoint – piuttosto, sono i message endpoint che intercettano i dati dai componenti a cui sono associati

NON vogliamo accoppiare il componente al message endpoint, NON vogliamo modificare il componente, quindi dobbiamo fare un inversione di dipendenze e fare in modo che siano gli endpoint a prendere i dati dai componenti e non i componenti a mandarli all'endpoint

Message Endpoint



Ci sono due tipologie principali di message endpoint

Message Endpoint

i componenti potrebbero essere consapevoli che c'è uno scambio di messaggi ma non si occupano direttamente di tale scambio (delegando i message endpoint a farlo) in modo da avere comunque disaccoppiamento completo dalla tecnologia

- Discussione
  - un message endpoint è un connettore che incapsula tutto il codice per l'accesso alle API del message broker
    - un message endpoint che ha solo lo scopo di disaccoppiare i componenti dalla tecnologia di messaging utilizzata è chiamato un *messaging gateway* [EIP]
  - nell'architettura esagonale, un message endpoint può essere realizzato come un adattatore
    - un outbound adapter per l'invio di messaggi oppure un inbound adapter per la ricezione di messaggi
  - di solito, un message endpoint ha lo scopo di catturare le interazioni già previste dai componenti di interesse e di ricondurle a interazioni basate sullo scambio di messaggi
    - ad es., un adapter che riceve un messaggio per un comando e invoca un'operazione della logica di business

Message Endpoint

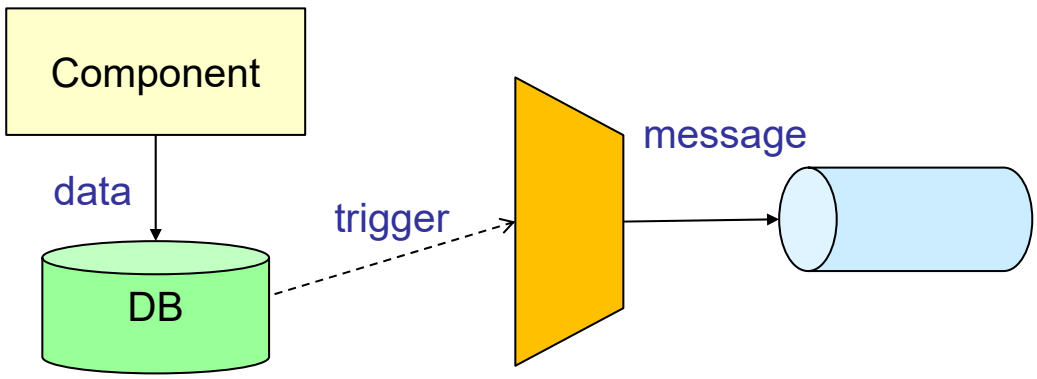
- Discussione
  - un *channel adapter* [EIP] è un altro tipo comune di message endpoint – che ha lo scopo di nascondere completamente a un componente l'infrastruttura per lo scambio di messaggi
  - molti elementi che fungono da "collante" in un sistema di integrazione di applicazioni, per "collegare" degli elementi preesistenti, sono dei channel adapter

Altro tipo di endpoint con lo scopo di nascondere completamente al componente l'infrastruttura per lo scambio di messaggi. Molti componenti sono proprio di questo tipo

Message Endpoint

- Discussione – alcuni esempi di channel adapter
  - in un'applicazione per basi di dati

Ogni volta che arriva un ordine lo salva nella base di dati, quindi ogni volta che arriva un ordine va inviato un messaggio che contenga le informazioni di quell'ordine



I trigger sono eventi associati ad elementi della base di dati (es se è stata aggiunta una riga a questa tabella e sono vere alcune condizioni, allora fai queste azioni). Questo è un modo per non modificare il componente in alcun modo, ma per generare un'azione in seguito ad un evento che vogliamo che produca una conseguenza, come l'aggiunta di un ordine

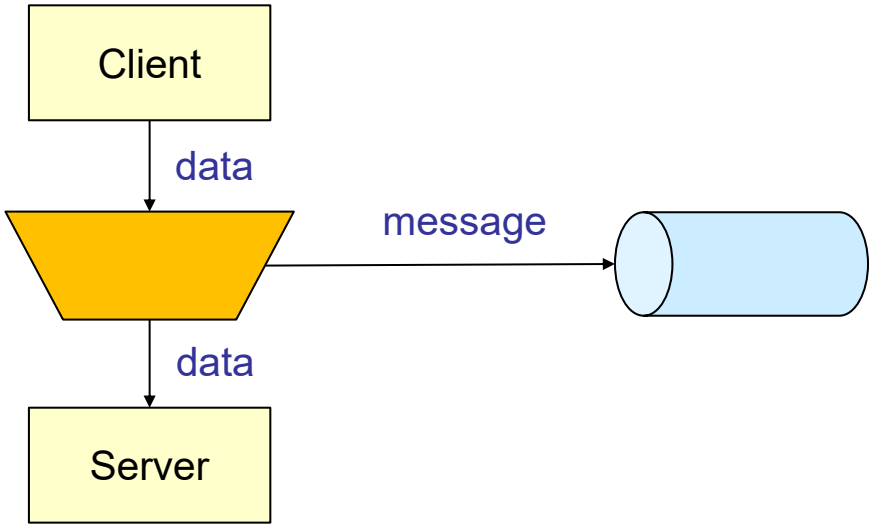
Message Endpoint

- Discussione – alcuni esempi di channel adapter
  - in un'applicazione client-server

Channel adapter che fa da intermediario tra client e server. Quando vengono chiamate funzioni di suo interesse cattura dei dati e svolge azioni.

Questo è per aggiungere comunicazione indiretta ANCHE nell'invocazione remota.

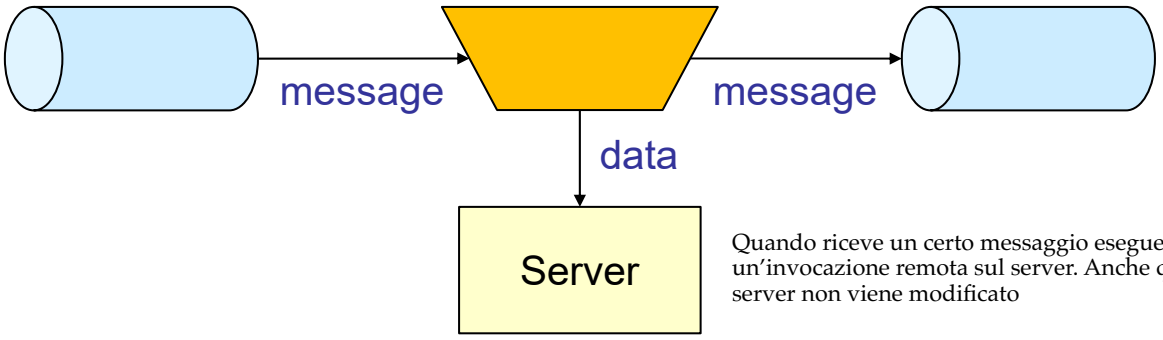
Anche qui non stiamo cambiando i componenti ma solo i connettori



Message Endpoint

- Discussione – alcuni esempi di channel adapter
  - in un'applicazione client-server

Quando vengono scambiati certi messaggi allora vengono invocate delle funzioni di un componente server



Quando riceve un certo messaggio esegue un'invocazione remota sul server. Anche qui il server non viene modificato

- Message Translator [POSA4]

- Problema
  - per sostenere un accoppiamento debole, può essere necessario trasformare un messaggio dal formato utilizzato dal suo produttore iniziale al formato compreso dai suoi consumatori finali – se questi formati sono differenti

    Accoppiamento debole tra i messaggi scambiati tra i vari componenti
- Soluzione
  - introduci dei traduttori di messaggi (*message translator*) tra i componenti, in grado di convertire messaggi da un formato all'altro

    Nasce nel contesto di XML, anche solo non per convertire (per es da XML a JSON) ma per modificare la struttura dello stesso messaggio XML
  - il message translator garantisce che il consumatore di un messaggio riceva il messaggio nel formato che preferisce

- Message Router [POSA4]

- Problema
  - i messaggi scambiati tra componenti devono essere instradati nell'infrastruttura per i messaggi in modo opportuno – ma i componenti non dovrebbero avere conoscenza del cammino di instradamento da utilizzare
- Soluzione
  - introduci dei *message router* che consumano messaggi da un canale e li re-inseriscono in altri canali, sulla base di alcune condizioni
  - un message router consente di muovere ciascun messaggio verso il consumatore o destinatario più opportuno

    Sulla base del messaggio stesso, basandosi o sull' header o sul body in base ai casi

* Discussione

- Il pattern architetturale Messaging sostiene la comunicazione asincrona – e la progettazione di sistemi software basati sulla comunicazione asincrona
  - suggerisce di organizzare un sistema distribuito come un insieme di componenti che interagiscono sulla base dello scambio di messaggi e di notifiche di eventi, in modo asincrono
  - l'applicazione di questo pattern fondamentale – per comporre un insieme di componenti indipendenti o autonomi – è favorita anche da ulteriori pattern di supporto alla comunicazione asincrona
  - questa famiglia di pattern architetturali può essere utilmente applicata anche nel contesto dell'integrazione di applicazioni