



Luca Cabibbo
Architettura
dei Sistemi
Software

Integrazione di applicazioni

dispensa asw465
ottobre 2023

*We believe that asynchronous messaging
will play an increasingly important role
in enterprise software development,
particularly in integration.*

Martin Fowler

1

Integrazione di applicazioni

Luca Cabibbo ASW



- Riferimenti

- [EIP] Hohpe, G. and Woolf, B. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. Addison-Wesley, 2004.
 - <https://www.enterpriseintegrationpatterns.com/>
- [POSA4] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. **Pattern-Oriented Software Architecture (vol. 4): A Pattern Language for Distributed Computing**. John Wiley & Sons, 2007

2

Integrazione di applicazioni

Luca Cabibbo ASW



- Obiettivi e argomenti

□ Obiettivi

- presentare brevemente il problema dell'integrazione di applicazioni
- illustrare uno studio di caso sull'applicazione del Messaging all'integrazione di applicazioni

□ Argomenti

- introduzione all'integrazione di applicazioni
- uno studio di caso di integrazione di applicazioni
- discussione



* Introduzione all'integrazione di applicazioni

- Le applicazioni “interessanti” vivono raramente in isolamento – piuttosto, devono spesso comunicare tra loro, per scambiarsi dati o servizi
 - in particolare, l'*integrazione di applicazioni* (*Enterprise Application Integration* o *EAI*) ha l'obiettivo di comporre i componenti di un insieme di applicazioni esistenti per realizzare una nuova applicazione, con un maggior valore di business



Approcci per l'integrazione di applicazioni

- Nel corso del tempo sono stati introdotti e utilizzati in pratica diversi approcci per l'integrazione di applicazioni
 - trasferimento di file
 - le applicazioni si scambiano dei file di dati condivisi
 - basi di dati condivise
 - le applicazioni memorizzano i dati che si devono scambiare in una base di dati condivisa
 - invocazioni remote
 - le applicazioni si scambiano i dati mediante l'invocazione di operazioni remote
 - messaging (con riferimento al pattern architetturale Messaging)
 - le applicazioni si scambiano dati, sotto forma di messaggi e in modo asincrono, collegandosi a un'infrastruttura per lo scambio di messaggi



Approcci per l'integrazione di applicazioni

- In questi anni il Messaging è divenuto in pratica l'approccio preferito per l'integrazione di applicazioni – perché supera numerosi limiti degli altri approcci
 - trasferimento di file
 - è poco tempestivo
 - spesso lo scambio di dati è tra singole coppie di applicazioni, con formati non compatibili con le altre applicazioni
 - basi di dati condivise
 - è difficile progettare una base di dati condivisa per l'integrazione di molte applicazioni
 - problemi di prestazioni
 - invocazioni remote
 - accoppiamento alto tra le applicazioni
 - l'integrazione sincrona è spesso caratterizzata da problemi di prestazioni e di affidabilità



Integrazione e messaging

- Oggi la comunicazione asincrona e il messaging sono elementi fondamentali nell'integrazione di applicazioni
 - l'integrazione avviene realizzando un'infrastruttura di comunicazione tra le applicazioni preesistenti, basata appunto sullo scambio di messaggi
 - il messaging viene spesso preferito ad altri approcci e tecnologie perché sostiene un accoppiamento basso tra i componenti e offre una maggior flessibilità
 - alcuni vantaggi sono accoppiamento debole, asincronia, consegna "immediata" (appena possibile), affidabilità, formati personalizzati, ...
 - in ogni caso, queste modalità di comunicazione sono importanti, in genere, nella composizione di componenti o servizi distribuiti indipendenti o autonomi



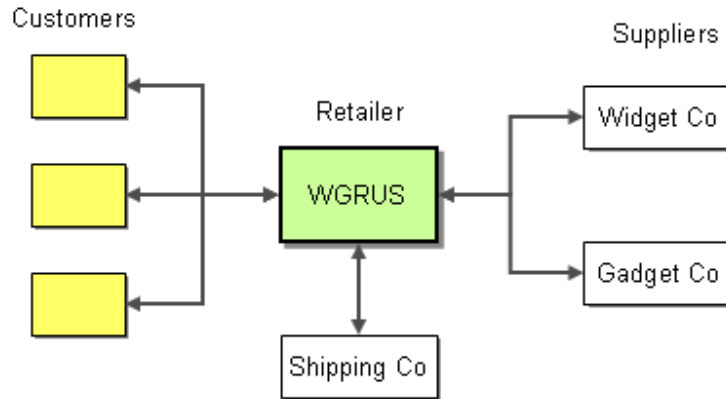
* Uno studio di caso di integrazione di applicazioni

- Presentiamo e discutiamo ora uno studio di caso (parziale) per l'integrazione di applicazioni basata sul messaging
 - si tratta dello studio di caso Widgets & Gadgets 'R Us (WGRUS) di [EIP]
 - <https://www.enterpriseintegrationpatterns.com/Chapter1.html>



WGRUS

- ❑ **Widgets & Gadgets 'R Us** è un rivenditore che acquista e rivende “widgets” e “gadgets”
 - Widgets & Gadgets 'R Us nasce dalla fusione di due aziende, Widget Co e Gadget Co



- **WGRUS** è il sistema software per Widgets & Gadgets 'R Us, che deve integrare alcuni componenti (preesistenti) dei sistemi informatici (preesistenti) di Widget Co e Gadget Co



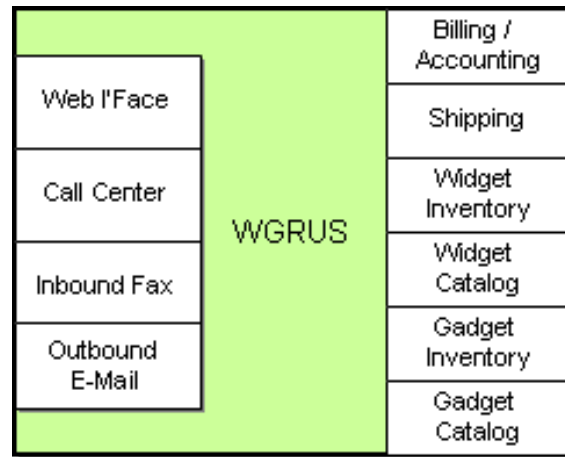
WGRUS

- ❑ Alcune funzionalità di WGRUS
 - inserimento ordini
 - elaborazione ordini
 - verifica stato di avanzamento di un ordine
 - gestione clienti
 - gestione catalogo prodotti
 - ...
- ❑ Qui consideriamo (parzialmente) solo la gestione degli ordini
 - inserimento, elaborazione, verifica stato di avanzamento degli ordini



WGRUS come problema di integrazione

- Come detto, il sistema WGRUS deve realizzare le varie funzionalità integrando alcuni componenti preesistenti
 - tra cui i sistemi preesistenti di Widget Co e Gadget Co – a loro volta composti da vari elementi

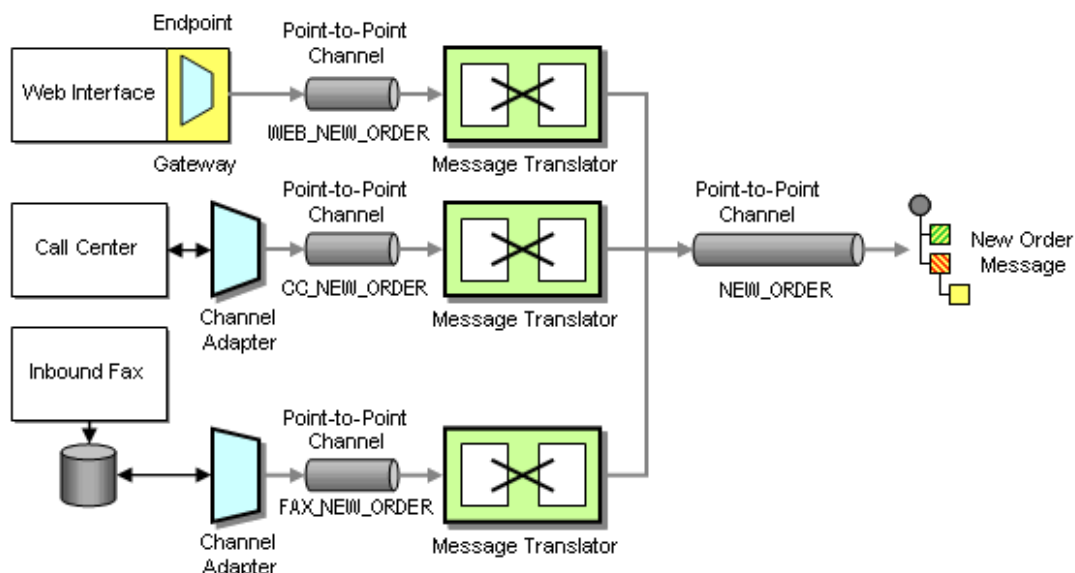


- a sinistra, sono mostrati i canali di interazione con i clienti
- a destra, i componenti applicativi preesistenti da riusare



- Ricezione di ordini

- Gli ordini possono essere ricevuti/immessi da vari client
 - un client web, un client per un addetto al telefono, ordini ricevuti via fax – ciascuno genera ordini con un formato diverso
 - si vuole invece avere un flusso di messaggi (unico e omogeneo) per tutti gli ordini

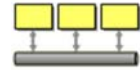




Pattern per l'EAI

Alcuni pattern per l'Enterprise Application Integration

▪ *Messaging*



- per integrare un insieme di applicazioni, in modo che possano scambiarsi informazioni e lavorare assieme

▪ *Message*



- un messaggio – ovvero, un tipo/flusso di messaggi

▪ *Message (Point-to-Point) Channel*

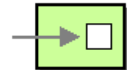


- un canale punto-punto (una coda) per lo scambio di messaggi



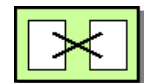
Pattern per l'EAI

▪ *Message Endpoint*



- collega un componente al sistema di messaging, per trasmettere/ricevere messaggi

▪ *Message Translator*



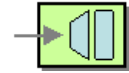
- una trasformazione che cambia il formato di un messaggio



Pattern per l'EAI

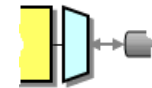
□ Esistono vari tipi di *Message Endpoint* – tra cui

▪ *Messaging Gateway*



- message endpoint che incapsula l'accesso al sistema di messaging, fornendo un'interfaccia con i metodi specifici del dominio applicativo – ma in modo indipendente dal sistema di messaging usato

▪ *Channel Adapter*



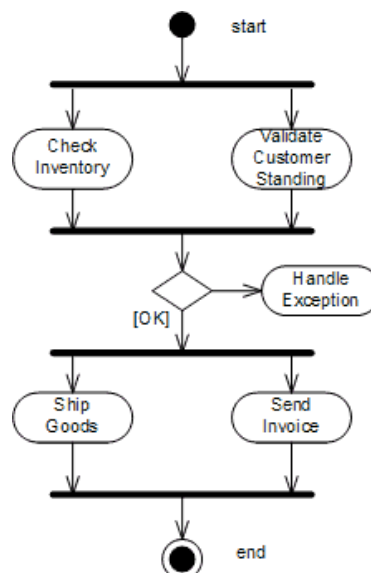
- message endpoint che realizza una connessione tra un'applicazione (di solito preesistente) e il sistema di messaging



- Elaborazione di ordini

□ Ora abbiamo un flusso consistente di ordini – l'elaborazione di un ordine richiede

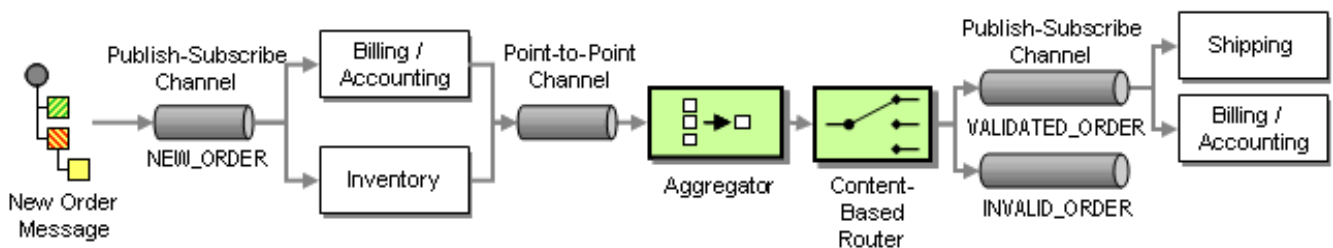
- verifica dello stato del cliente – nessun debito in sospeso
- verifica dell'inventario – disponibilità degli articoli ordinati
- se tutto ok, si può procedere





Elaborazione di ordini

- Come elaborare gli ordini?
 - ogni ordine viene inviato separatamente e in parallelo a contabilità e inventario per le verifiche
 - le due risposte devono essere poi aggregate
 - gli ordini confermati vanno poi inviati ai sistemi di spedizione e di fatturazione



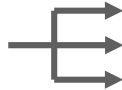
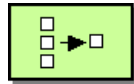
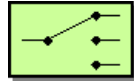
17

Integrazione di applicazioni

Luca Cabibbo ASW



Pattern per l'EAI

- **Publish-Subscribe Channel** 
 - un canale pub-sub per lo scambio di messaggi
- **Aggregator** 
 - combina il contenuto di messaggi diversi ma correlati
- **Content-Based Router** 
 - gira un messaggio a un opportuno canale, verso la sua destinazione, sulla base del contenuto del messaggio

18

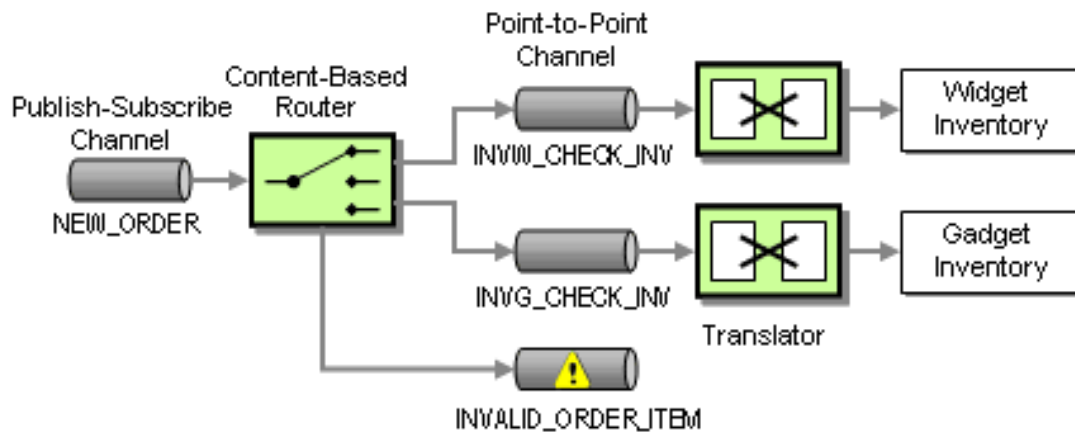
Integrazione di applicazioni

Luca Cabibbo ASW



- Controllo dell'inventario

- In realtà, ci sono due sistemi/funzionalità per il controllo dell'inventario
 - una per i Widget e una per i Gadget
 - ciascuna richiesta va instradata al sistema giusto
 - ipotesi (temporanea): una sola riga d'ordine per ordine
 - ipotesi (semplificativa): il primo carattere del codice del prodotto è W oppure G



19

Integrazione di applicazioni

Luca Cabibbo ASW



Pattern per l'EAI

- *Invalid Message Channel*
 - destinazione di messaggi non validi



20

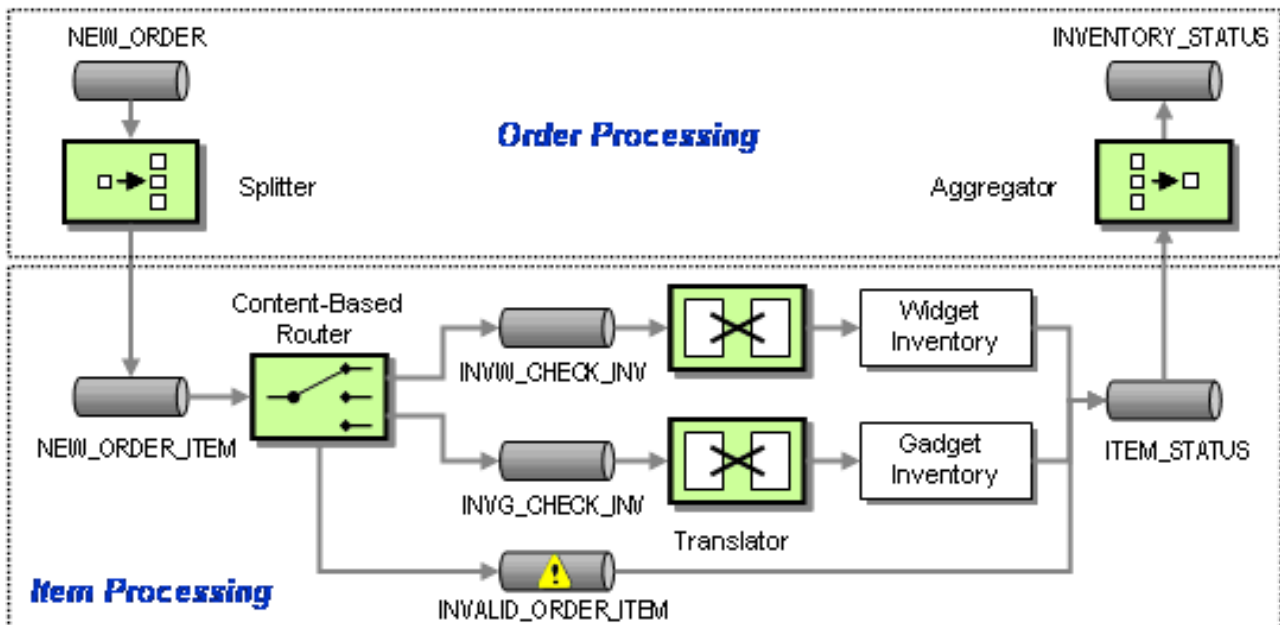
Integrazione di applicazioni

Luca Cabibbo ASW



- Ordini con più righe d'ordine

- In realtà, un ordine contiene normalmente più righe d'ordine
 - alcune saranno relative a widget, altre a gadget
 - la disponibilità delle merci va verificata riga d'ordine per riga d'ordine



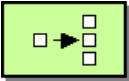
21

Integrazione di applicazioni

Luca Cabibbo ASW



Pattern per l'EAI

- **Splitter**
 - decompone un messaggio in un insieme di messaggi, ciascuno dei quali può richiedere (successivamente) una diversa elaborazione

22

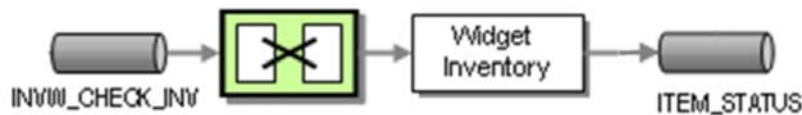
Integrazione di applicazioni

Luca Cabibbo ASW

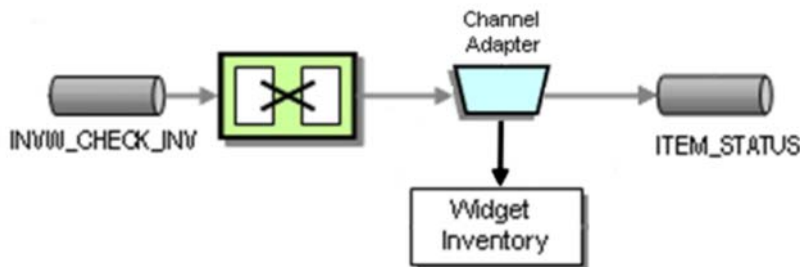


Un'osservazione

- Si consideri questa porzione del diagramma



- sembra che il componente preesistente Widget Inventory partecipi in prima persona alla soluzione di integrazione – ma è proprio lui a consumare e produrre messaggi direttamente?
- no, tale componente verrà probabilmente utilizzato tramite un opportuno message endpoint (ad es., un channel adapter)



- considerazioni analoghe si possono fare per l'accesso alle funzionalità di altri componenti preesistenti

23

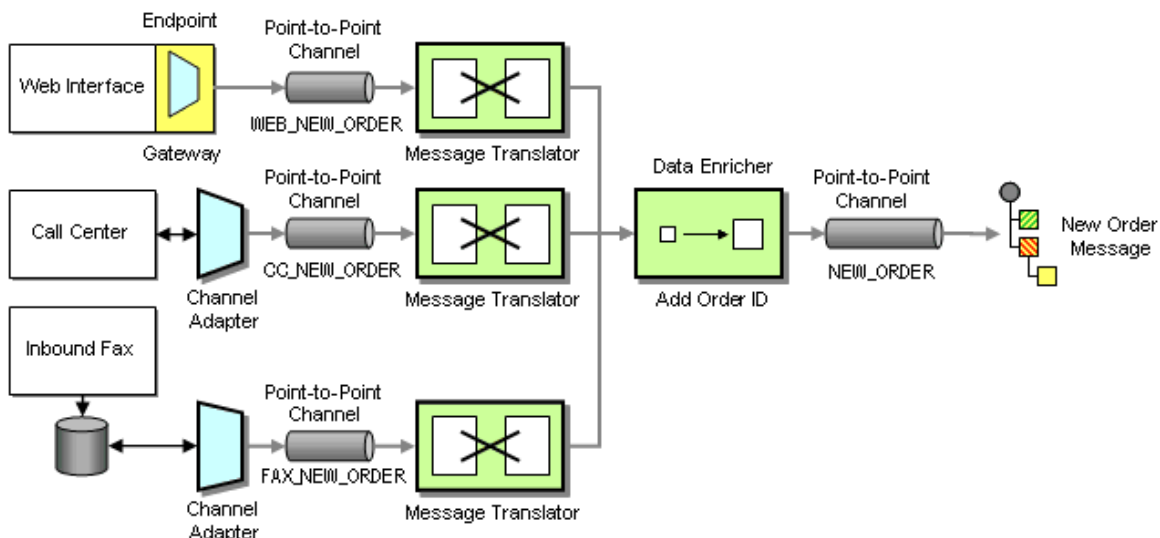
Integrazione di applicazioni

Luca Cabibbo ASW



- Identificatore d'ordine

- Messaggi elaborati separatamente possono essere ricombinati mediante un Aggregator sulla base di opportune informazioni di correlazione
 - ad es., un identificatore d'ordine
 - ma è necessario aggiungere un identificatore a ciascun ordine



24

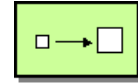
Integrazione di applicazioni

Luca Cabibbo ASW

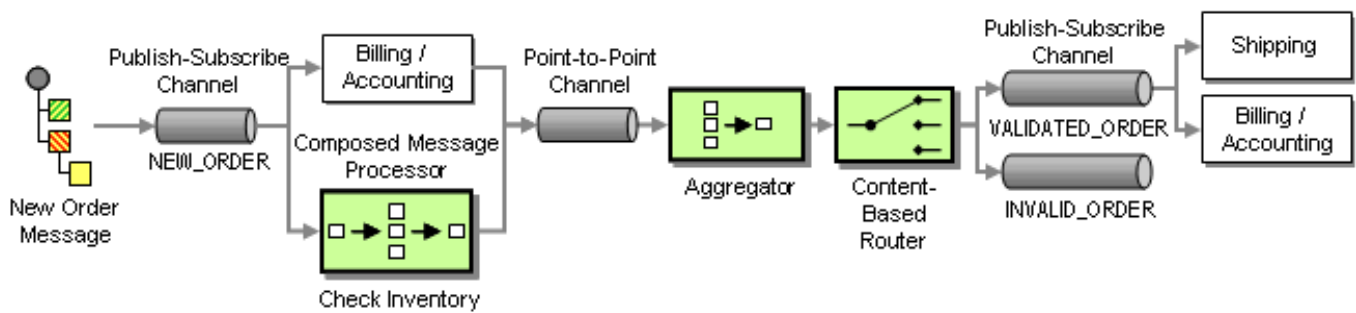


Pattern per l'EAI

- *Content Enricher*
 - aggiunge informazioni a un messaggio

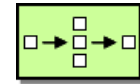


- Gestione degli ordini – rivista





- *Composed Message Processor*

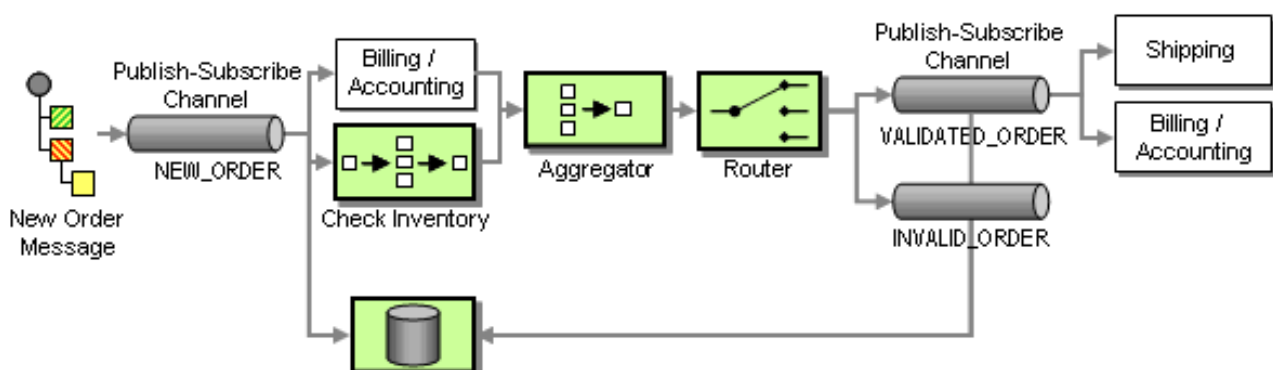


- mantiene il flusso di messaggi complessivo, anche se i diversi messaggi richiedono elaborazioni diverse



- Verificare lo stato di un ordine

- L'elaborazione di un ordine richiede lo svolgimento di varie attività
 - come è possibile verificare lo stato di avanzamento di un ordine? è stata effettuata la spedizione? è in attesa di prodotti? è bloccato perché il cliente ha debiti in sospeso?
 - è possibile rispondere conoscendo l'“ultimo” messaggio scambiato nel sistema circa l'ordine – questo può essere fatto memorizzando i messaggi rilevanti in un repository di messaggi





Pattern per l'EAI

▪ *Message Store*

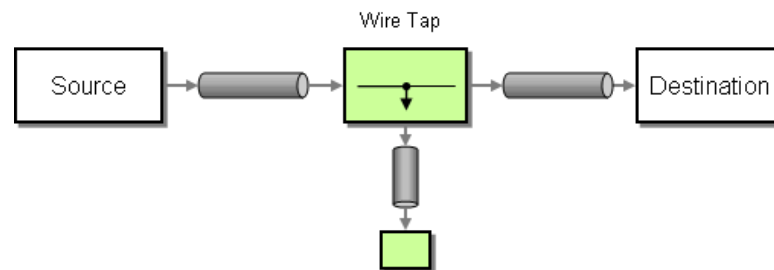


- quando viene inviato un messaggio nel sistema, viene inviato anche un messaggio duplicato e memorizzato in un repository di messaggi
 - semplice se il canale di cui bisogna memorizzare i messaggi è di tipo *Publish-Subscribe*

▪ *Wire Tap*



- per duplicare su più canali i messaggi inviati su un certo canale



* **Discussione**

- Lo studio di caso WGRUS esemplifica l'applicazione della comunicazione asincrona per l'integrazione di applicazioni, nonché l'applicazione di alcuni pattern di supporto allo scambio di messaggi
 - i componenti preesistenti non vengono collegati tra di loro direttamente
 - piuttosto, l'integrazione è basata sulla costruzione, il consumo, la trasformazione, lo splitting, l'aggregazione e il routing di messaggi, anche con riferimento a un certo numero di canali per messaggi
 - i componenti preesistenti non sono collegati direttamente nemmeno all'infrastruttura per lo scambio di messaggi – piuttosto, vengono collegati ad essa mediante dei message endpoint, che incapsulano l'accesso all'infrastruttura per i messaggi, e inoltre fungono da “collante” tra i componenti preesistenti



Discussione

- [EIP] presenta numerosi pattern per la comunicazione basata sullo scambio di messaggi – alcuni dei quali sono stati ripresi anche da [POSA4] – per rappresentare, tra l'altro
 - elementi infrastrutturali per lo scambio di messaggi – come *Message* e *Message Channel*
 - tipi di canali per messaggi – come *Point-to-point Channel*, *Publish-Subscribe Channel* o *Invalid Message Channel*
 - tipi di messaggi – come *Document Message* o *Request-Reply*
 - routing di messaggi – come *Splitter* o *Aggregator*
 - trasformazioni di messaggi – come *Content Enricher* o *Content Filter*
 - endpoint per lo scambio di messaggi – come *Messaging Gateway* o *Channel Adapter*
 - gestione e monitoraggio del sistema – come *Control Bus* o *Process Manager*



Discussione

- Nell'esempio relativo al sistema WGRUS è possibile osservare l'applicazione di alcuni pattern architetturali fondamentali
 - *Messaging* – come pattern per l'integrazione di applicazioni
 - *Pipes and Filters* – i sistemi basati sullo scambio di messaggi sono spesso organizzati mediante questo stile architetturale
 - il processo di gestione degli ordini deve trasformare un flusso di messaggi in ingresso (ordini) in flussi di messaggi in uscita (spedizioni, fatture)
 - la trasformazione è stata decomposta in una sequenza di passi successivi, guidati da un opportuno modello di dominio
 - *Domain Model* – il modello di dominio utilizzato in questo caso è un modello delle attività, che descrive le attività da svolgere nel processo di gestione degli ordini, insieme al loro ordine
 - l'architettura del sistema integrato per la gestione degli ordini ha proprio la forma del diagramma delle attività da svolgere