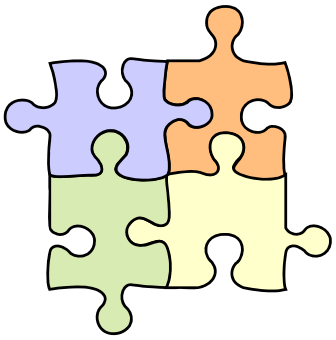
Luca Cabibbo

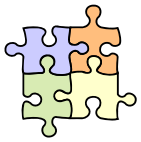


Architetture Software

Messaging (e integrazione di applicazioni)

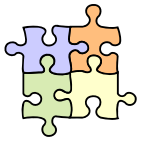
Dispensa PA 3

ottobre 2008



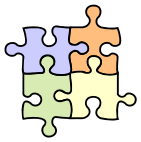
- Fonti

- [POSA4] Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing
- [Hohpe&Woolf 2004] Enterprise Integration Patterns – *http://www.enterpriseintegrationpatterns.com/*



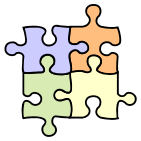
- Obiettivi e argomenti

- Obiettivi
  - conoscere lo stile architetturale del messaging
  - mostrare un esempio di applicazione del messaging all'integrazione di applicazioni
- Argomenti
  - introduzione
  - messaging [POSA4]
  - altri pattern per il messaging [POSA4]
  - integrazione di applicazioni
  - messaging per l'integrazione di applicazioni



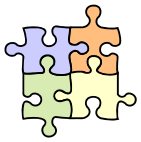
* Introduzione

- Le architetture client/server e ad oggetti distribuiti sono basate su un paradigma di interazione richiesta-risposta
  - estensione della chiamata di procedure/invocazione di metodi (meccanismo fondamentale della programmazione imperativa) ad un contesto distribuito
  - comunicazione uno-a-uno
  - basato su interfacce procedurali e una tipizzazione forte
  - accoppiamento forte dei client nei confronti dei server
- In alcuni casi è preferibile invece
  - un paradigma di interazione diverso
  - basato sullo scambio di messaggi/documenti, per quanto possibile auto-descrittivi
  - comunicazione molti-a-uno o uno-a-molti
  - accoppiamento debole tra le parti coinvolte



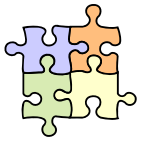
Middleware message-oriented

- Middleware message-oriented (MOM)
  - famiglia di middleware basata sullo scambio asincrono di messaggi – e non su protocolli sincroni di richiesta/risposta
  - sostengono un accoppiamento debole tra componenti
  - possono offrire elevata flessibilità ed affidabilità
  - numerose implementazioni, sia “centralizzate” (ad es., JMS in Java EE) che “distribuite” (ad es., Tibco)
  - questa modalità di interazione
    - è disponibile anche nelle tecnologie a componenti
    - può essere utilizzata nell’integrazione di applicazioni
    - è un ingrediente essenziale nelle architetture orientate ai servizi



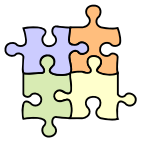
Messaging e Publisher-Subscriber

- [POSA4] distingue due stili architetturali fondamentali per la comunicazione asincrona
  - messaging
    - comunicazione basata sullo scambio di messaggi
    - accoppiamento debole tra produttori e consumatori di messaggi
    - comunicazione molti-a-uno
  - publisher-subscriber
    - comunicazione basata sulla notifica di eventi
    - accoppiamento ancora più debole
    - comunicazione uno-a-molti
  - ci concentriamo soprattutto su messaging – è possibile considerare publisher-subscriber una sua variante



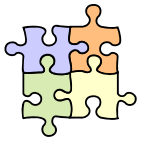
Messaging e strumenti di middleware

- Strumenti di middleware e modalità di comunicazione
  - gli strumenti MOM offrono normalmente entrambe le modalità di comunicazione asincrona
  - le tecnologie a componenti offrono sia meccanismi di comunicazione sincroni (basati su interfacce procedurali) che asincroni (basati sullo scambio di messaggi/documenti/eventi)
  - anche la tecnologia dei Web Services consente sia la comunicazione sincrona che quella asincrona



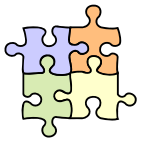
* Messaging [POSA4]

- Il pattern architetturale **Messaging**
  - una possibile infrastruttura di comunicazione – per integrare componenti sviluppati indipendentemente in un sistema coerente
  - comunicazione è basata sullo scambio asincrono di messaggi (o documenti) tra i vari componenti
    - non sulla base di invocazioni remote
    - la ricezione di un messaggio da parte di un componente scatena l'esecuzione di un'operazione per gestire il messaggio ricevuto
- L'applicazione di Messaging richiede normalmente anche l'uso di ulteriori pattern (più specifici), alcuni dei quali sono descritti nel seguito



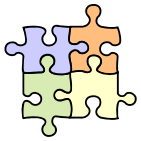
Messaging

- Contesto
  - integrazione di componenti (o servizi) sviluppati indipendentemente
- Problema
  - un sistema composto da componenti (o servizi) sviluppati indipendentemente
  - questi componenti devono essere integrati – per formare un sistema coerente
    - Enterprise Application Integration – EAI
  - questi componenti devono interagire in modo affidabile
  - l'accoppiamento complessivo tra i componenti deve rimanere basso
    - per essere indipendenti, i componenti non sono (e non devono essere) a conoscenza l'uno dell'altro



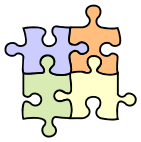
Messaging

- Soluzione
  - collega i componenti (o servizi) mediante un *bus per messaggi* che gli consente di scambiarsi *messaggi* in modo *asincrono*
  - codifica i messaggi in modo che mittente e destinatario possano comunicare in modo affidabile e senza dover conoscere staticamente tutte le informazioni sui tipi di dati
    - i messaggi
      - incapsulano richieste e strutture di dati
      - sono spesso auto-descrittivi – contengono sia dati (valori) che meta-dati, che descrivono l'organizzazione ed il significato dei dati
- Il messaging è sostenuto da strumenti di middlware opportuni
  - *Message-Oriented Middleware* (*MOM*)



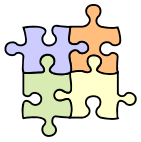
Messaging

- Modalità d'interazione in un'architettura di messaging
  - la comunicazione viene di solito iniziata dal componente (*produttore*) che produce il messaggio
  - in genere, il produttore non specifica l'identità del componente (*consumatore*) che consumerà il messaggio
    - piuttosto, il produttore invierà il suo messaggio ad un *canale* (o *destinazione*) intermedio
  - un consumatore leggerà messaggi da questi canali intermedi
    - la lettura del messaggio scatenerà l'esecuzione di un'operazione opportuna da parte del consumatore, per gestire il messaggio
    - è possibile che la gestione del messaggio preveda un messaggio di risposta al produttore – ma questa non è la norma
  - invio e ricezione dei messaggi avvengono in modo asincrono

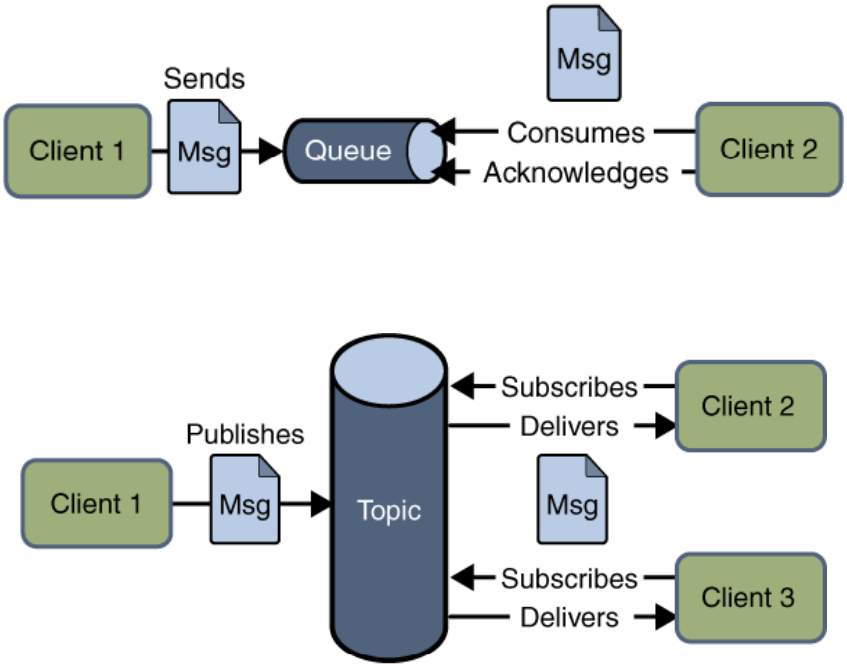


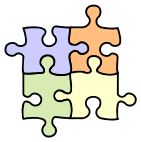
Messaging e Publisher-Subscriber

- Due varianti principali per la comunicazione asincrona basata sullo scambio di messaggi
  - sulla base di due tipi principali di canali/destinazioni
  - *Messaging*, basato su *code*
    - ciascun messaggio è consumato da uno ed un solo consumatore – è un canale di comunicazione *molti-a-uno*
  - *Publisher-Subscriber*, basato su *topic* (argomenti)
    - ciascun messaggio (evento) può essere consumato da più consumatori, registrati presso il canale – è un canale publisher-suscriber, *uno-a-molti* – i canali sono generalmente "tematici", ovvero ciascuno è legato ad un argomento – possibile un'organizzazione gerarchica degli argomenti

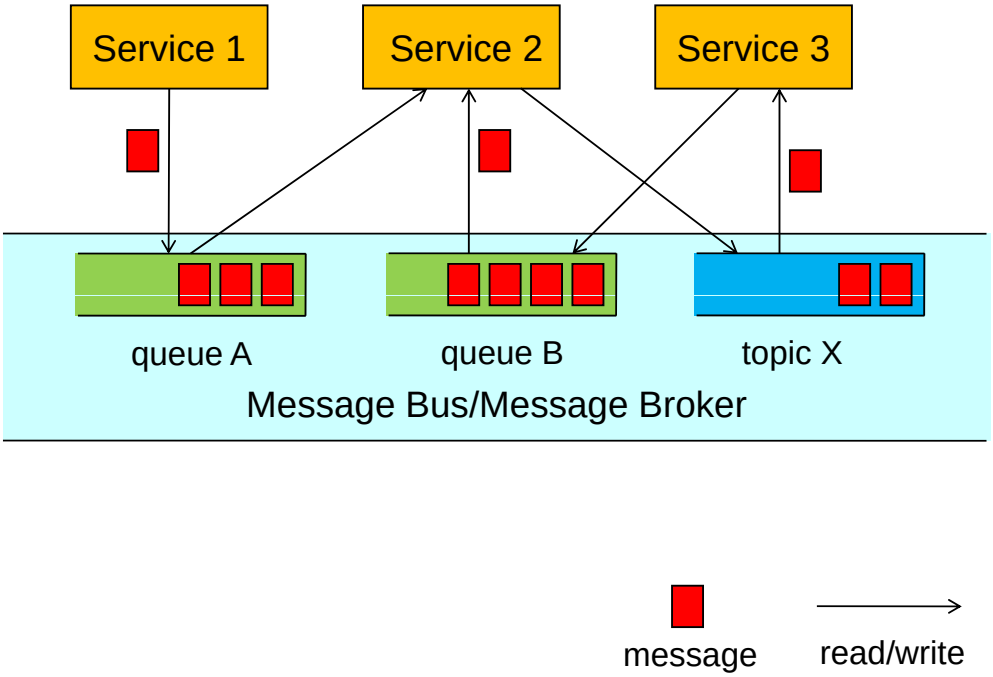


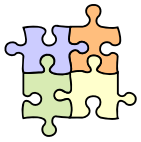
Code e argomenti





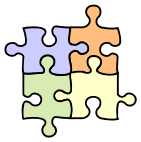
Messaging





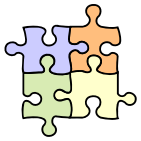
Messaging e accoppiamento

- Il messaging sostiene un *accoppiamento debole* tra componenti
  - produttori e consumatori, per comunicare, devono essere d'accordo
    - sul formato dei messaggi scambiati
    - sul canale usato per lo scambio dei messaggi
  - produttori e consumatori non devono conoscersi ulteriormente
    - non devono conoscere l'uno l'identità dell'altro
    - non devono conoscere l'uno l'interfaccia (in senso procedurale) dell'altro
    - non devono essere attivi in modo sincrono
  - l'accoppiamento tra componenti può essere "astratto e minimale"



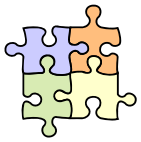
Messaging

- Conseguenze
  - ☺ manutenibilità – è possibile l'aggiunta/rimozione/sostituzione di componenti
  - ☺ affidabilità – possibile la consegna affidabile (transazionale) di messaggi
  - ☺ affidabilità – possibilità di effettuare il broadcast di guasti
  - ☹ prestazioni – overhead dovuto alla gestione delle destinazioni (code ed argomenti) e degli eventi, nonché alla necessità di codificare/decodificare messaggi
  - ☹ un componente che genera eventi non sa se i suoi eventi verranno effettivamente gestiti – oppure ci potrebbero essere conflitti se un evento viene gestito da più componenti



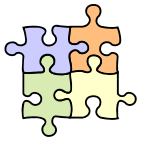
* Altri pattern per il Messaging [POSA4]

- L'applicazione di Messaging richiede normalmente anche l'uso di ulteriori pattern (più specifici)
  - la comunicazione avviene mediante lo scambio di messaggi
  - la comunicazione avviene tramite canali per messaggi
  - componenti pre-esistenti possono essere collegati ai canali mediante componenti adattatori
  - possibile avere componenti aggiuntivi – ad esempio
    - componenti che si occupano della trasformazione di messaggi (filtri)
    - componenti che si occupano del routing di messaggi
  - publisher-subscriber è una variante di messaging



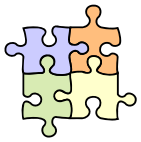
- Message [POSA4]

- Problema
  - come è possibile connettere due applicazioni/componenti per consentire lo scambio di pezzi di informazioni – ma anche l'invocazione di servizi?
- Soluzione
  - encapsula i dati scambiati (o la richiesta di invocazione) in un *message* (messaggio)
  - il messaggio contiene
    - un header – specifica metadati circa le informazioni trasmesse (ad es., origine, destinazione, dimensione, scadenza, ...)
    - un body (o payload) – contiene le informazioni effettive



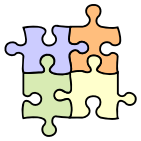
Message

- Conseguenze
  - sostiene un accoppiamento debole
    - conoscere il formato dei messaggi accettati da un destinatario è una forma di accoppiamento più debole che non conoscere l'interfaccia procedurale del destinatario
  - flessibilità
    - se il formato dei messaggi inviati viene "esteso" rispetto a quello inizialmente previsto, un consumatore potrebbe essere in grado di erogare un servizio soddisfacente considerando solo i dati che comprende ed ignorando quelli che non comprende
  - overhead nella codifica/decodifica di messaggi



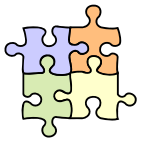
- Message Channel [POSA4]

- Problema
  - i messaggi contengono solo i dati che devono essere scambiati tra client e servizi
  - per sostenere un accoppiamento debole, client e servizi non dovrebbero sapere chi è interessato a quali messaggi
  - è necessario un meccanismo per connettere client e servizi, consentendo lo scambio di messaggi
- Soluzione
  - connetti i client e i servizi che devono collaborare usando un *message channel* (canale per messaggi) che gli consenta di scambiare messaggi
  - quando un client deve comunicare un messaggio, lo scrive nel canale dei messaggi
  - i servizi interessati al messaggio lo possono prelevare ed elaborare



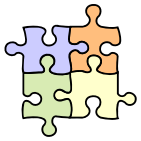
Message Channel

- Discussione
  - possibili/comuni diversi tipi di canali di messaggi specializzati
- Conseguenze
  - sostiene un accoppiamento debole
    - conoscere una destinazione intermedia condivisa con un destinatario è una forma di accoppiamento più debole che non conoscere l'identità del destinatario
  - possibile una gestione affidabile (ad esempio, transazionale) dei messaggi e della loro consegna
  - la gestione di un message channel richiede memoria, risorse di rete ed eventualmente anche memorizzazione persistente



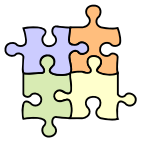
- Message Endpoint [POSA4]

- Problema
  - si vogliono far comunicare, mediante lo scambio di messaggi, applicazioni o componenti autonomi (ad esempio, pre-esistenti), in cui tale modalità di comunicazione non era prevista
  - per quanto possibile, non si vogliono modificare tali applicazioni o componenti
  - è tuttavia necessario abilitarli all'invio e/o alla ricezione di messaggi



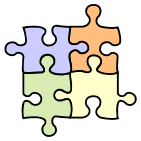
Message Endpoint

- Soluzione
  - connetti i client e i servizi che devono interagire all'infrastruttura di messaging mediante dei *message endpoint* specializzati che gli consentano di scambiare messaggi
  - quando un client deve comunicare dei dati, questi dati vengono passati al/intercettati dal message endpoint che gli è associato – che converte i dati in un messaggio e scrive il messaggio in un message channel
  - il messaggio viene ricevuto da un altro message endpoint – che estrae i dati dal messaggio e li passa al servizio consumatore



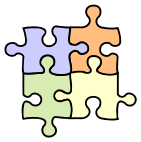
Message Endpoint

- Discussione
  - alcuni esempi di modalità di funzionamento di un message endpoint
    - il client e/o il servizio sono parte di un'applicazione client/server – il message endpoint si sostituisce al client e/o al server, catturando le rispettive richieste o risposte e trasmettendole sotto forma di messaggi
    - in un'applicazione per basi di dati – vengono definiti dei trigger che catturano particolari cambiamenti nella base di dati (ad es., è stato memorizzato un nuovo ordine) e si attivano per generare e trasmettere un messaggio
  - un message endpoint incapsula l'accesso al middleware per il messaging – client e servizi non sono accoppiati al middleware utilizzato



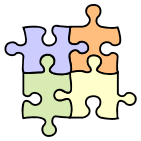
- Message Translator [POSA4]

- Problema
  - per sostenere un accoppiamento debole, non è possibile ipotizzare che i servizi che ricevono i messaggi comprendano il formato dei messaggi utilizzato dai client che generano messaggi
  - è spesso necessario trasformare i messaggi dal formato utilizzato dai client al formato compreso dai servizi



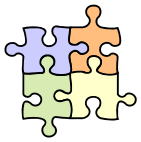
Message Translator

- Soluzione
  - introduci dei *message translator* (traduttori di messaggi) tra client e servizi, in grado di convertire messaggi da un formato all'altro
  - un client invia un messaggio nel formato che preferisce
  - il message translator garantisce che il servizio riceva il messaggio nel formato a lui preferito
  - è possibile definire message translator che realizzano una traduzione bidirezionale tra formati di messaggi



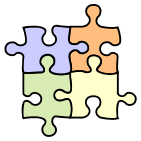
Message Translator

- Discussione
  - sostiene un accoppiamento debole
  - ci sono strumenti dedicati alla traduzione di messaggi (tipicamente XML) tra formati diversi



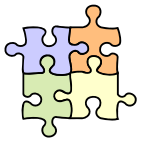
- Message Router [POSA4]

- Problema
  - i messaggi scambiati tra client e servizi devono essere instradati nell'infrastruttura di messaging
  - client e servizi (e nemmeno canali e traduttori) non dovrebbero avere conoscenza del cammino di instradamento da scegliere
  - è tuttavia necessario scegliere un percorso per la propagazione dei messaggi
- Soluzione
  - introduci dei *message router* che consumano messaggi da un canale e li re-inseriscono in un altro canale, sulla base di alcune condizioni (ad esempio, sull'header o sul body)
  - un message router connette un insieme di canali per messaggi in una rete di canali per messaggi – muovendo i messaggi verso il ricevitore più opportuno



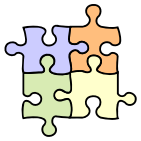
- Publisher-Subscriber [POSA4]

- Problema
  - i componenti di un gruppo di applicazioni distribuite sono debolmente accoppiati tra di loro, ed operano in modo largamente indipendente
  - è necessario un meccanismo di notifica – ad esempio, per propagare informazioni ad alcuni o a tutti i componenti
  - queste notifiche sono relative ad eventi che potrebbero avere effetto sull'elaborazione svolta dai singoli componenti
- Soluzione
  - definisci un'infrastruttura per propagare notifiche che consenta a *publisher* di diffondere eventi che potrebbero interessare altri, e a *subscriber* di essere notificati di questi eventi quando tali informazioni sono pubblicate
  - usa degli opportuni *canali publish-subscribe*



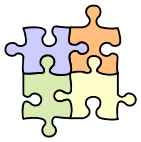
- Tecnologie per il Messaging

- Per le tecnologie per il messaging, vedi anche
  - dispensa su Messaging



* Integrazione di applicazioni

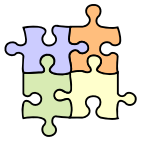
- Le applicazioni “interessanti” vivono raramente in isolamento
  - spesso devono comunicare tra di loro, per scambiarsi dati o servizi
  - questo solleva il problema dell’integrazione di applicazioni (EAI)
- Nel corso del tempo, sono stati introdotti ed utilizzati in pratica diversi approcci per l’integrazione di applicazioni
  - trasferimento di file
  - base di dati condivisa
  - invocazione di procedure remote
  - messaging



* Messaging per l’integrazione di applicazioni

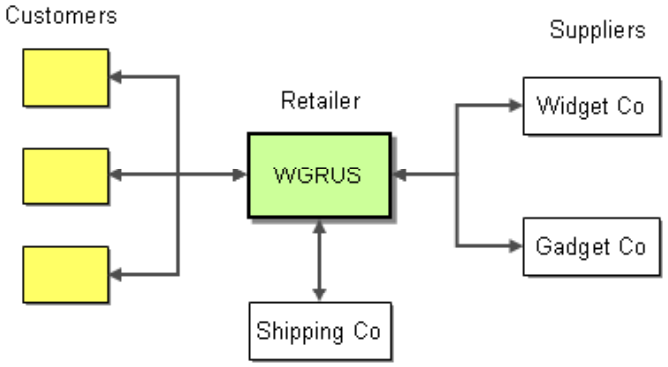
- Vedi

  http://www.enterpriseintegrationpatterns.com/Chapter1.html
- Il messaging è considerata una tecnologia fondamentale nell’integrazione di applicazioni pre-esistenti
  - l’integrazione avviene realizzando un’infrastruttura di comunicazione tra le applicazioni pre-esistenti, basata appunto sul messaging
  - il messaging viene (talvolta) preferito ad altre tecnologie perché richiede un accoppiamento basso tra i componenti ed offre una maggior flessibilità
    - alcuni vantaggi – accoppiamento debole, asincronia, consegna “immediata” (appena possibile), affidabilità, formati personalizzati, ...

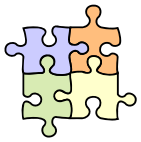


- Studio di caso: Widgets & Gadgets 'R Us

- Widgets & Gadgets 'R Us è un rivenditore che acquista e rivende "widgets and gadgets"
  - nato dalla fusione di due aziende pre-esistenti

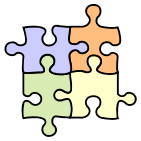


  - il sistema *WGRUS* deve integrare alcuni componenti pre-esistenti
    - i sistemi pre-esistenti di Widget Co e Gadget Co



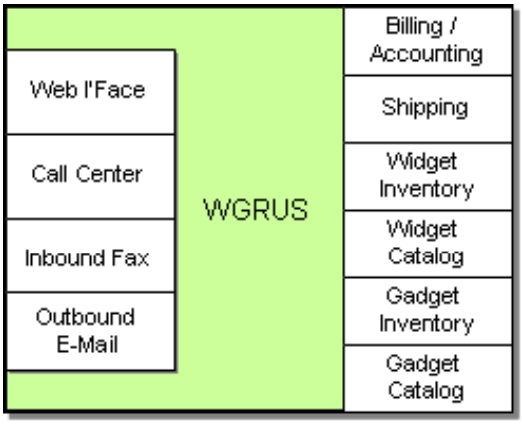
WGRUS

- Funzionalità di WGRUS
  - inserimento ordini
  - elaborazione ordini
  - verifica stato dell'ordine
  - gestione clienti
  - gestione catalogo prodotti
  - ....
- Consideriamo (parzialmente) solo la gestione degli ordini (inserimento, elaborazione, verifica stato)

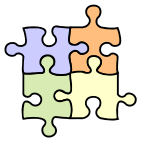


WGRUS come problema di integrazione

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

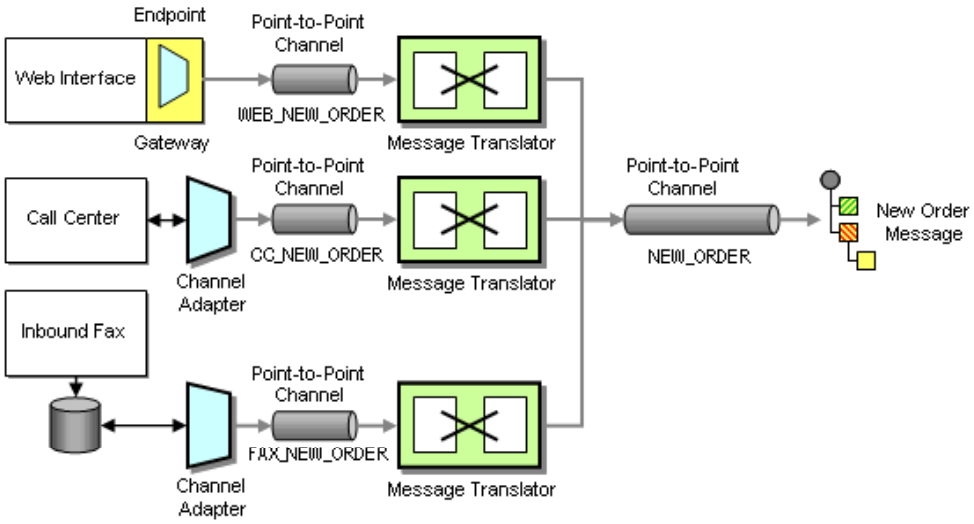


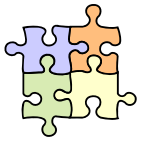
  - a sinistra, sono mostrati i canali di interazione con i clienti
  - a destra, i componenti applicativi pre-esistenti da riusare



- 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 ed omogeneo) per tutti gli ordini



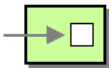


Pattern per l'EAI (1)

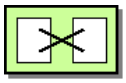
- Alcuni pattern per l'Enterprise Application Integration
  - *Message*
    - un messaggio (un tipo/flusso di messaggi)

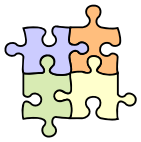
  - *Message* (*Point-to-Point*) *Channel*
    - un canale per lo scambio di messaggi

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



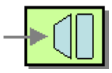
  - *Message Translator*
    - una trasformazione che cambia il formato di un messaggio



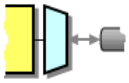


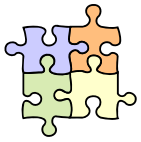
Pattern per l'EAI (2)

- Esistono vari tipi di *Message Endpoint* – tra cui
  - *Messaging Gateway*
    - endpoint che incapsula l'accesso al sistema di messaging, fornendo un'interfaccia con i metodi specifici del dominio applicativo



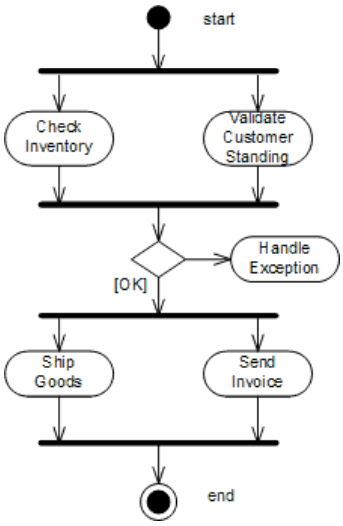
  - *Channel Adapter*
    - endpoint che realizza una connessione tra un'applicazione (anche pre-esistente) e il sistema di messaging

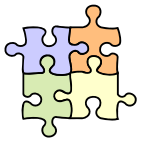




- Elaborazione di ordini

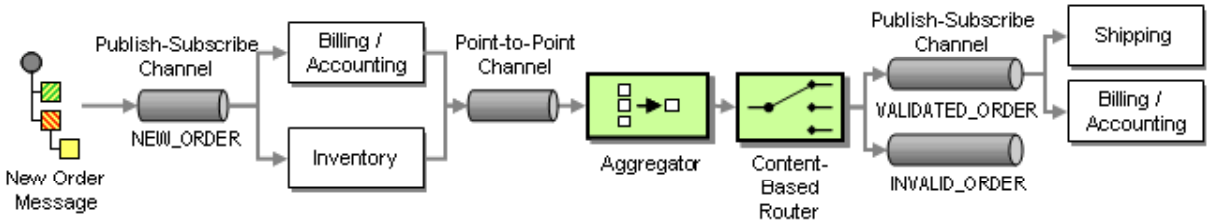
- Ora abbiamo un flusso consistente di ordini – l'elaborazione di un ordine richiede
  - di verificare lo stato del cliente – nessun debito in sospeso
  - di verificare l'inventario – disponibilità degli articoli ordinati
  - se tutto ok, si può procedere





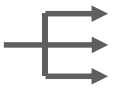
Elaborazione di ordini

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

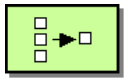


Pattern per l'EAI (3)

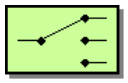
- *Publish-Subscribe Channel*
  - un topic



- *Aggregator*
  - combina il contenuto di messaggi diversi ma correlati

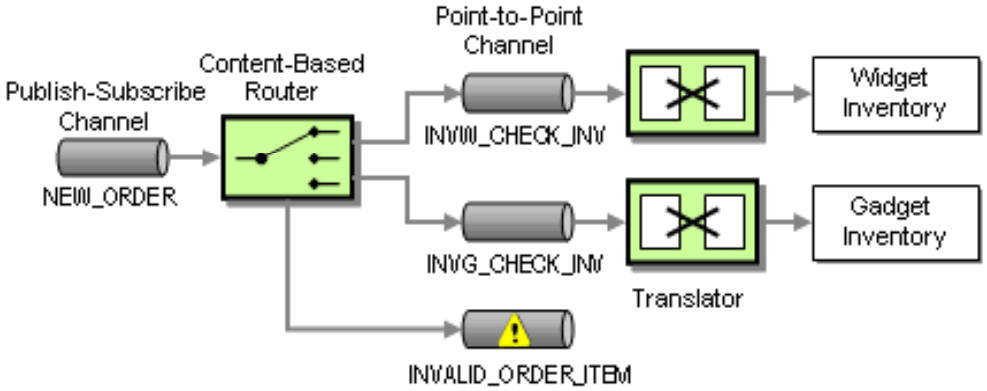


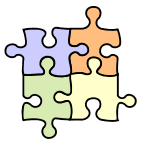
- *Content-Based Router*
  - gira un messaggio ad un'opportuna destinazione, sulla base del contenuto del messaggio



- Controllo dell'inventario

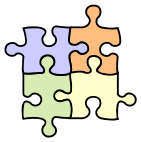
- In realtà, ci sono due sistemi/funzionalità per il controllo dell'inventario
  - una per i Widget ed una per i Gadget
  - ciascuna richiesta va instradata al sistema giusto
    - nota: il primo carattere del codice del prodotto è G o W





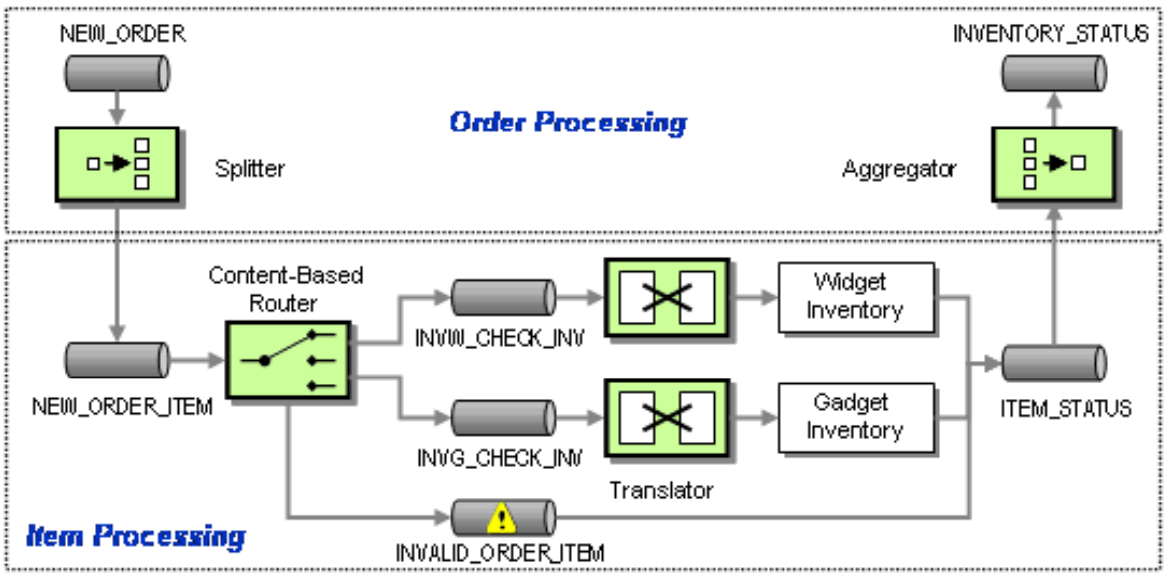
Pattern per l'EAI (4)

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

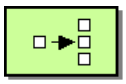


- Ordini con più righe d'ordine

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



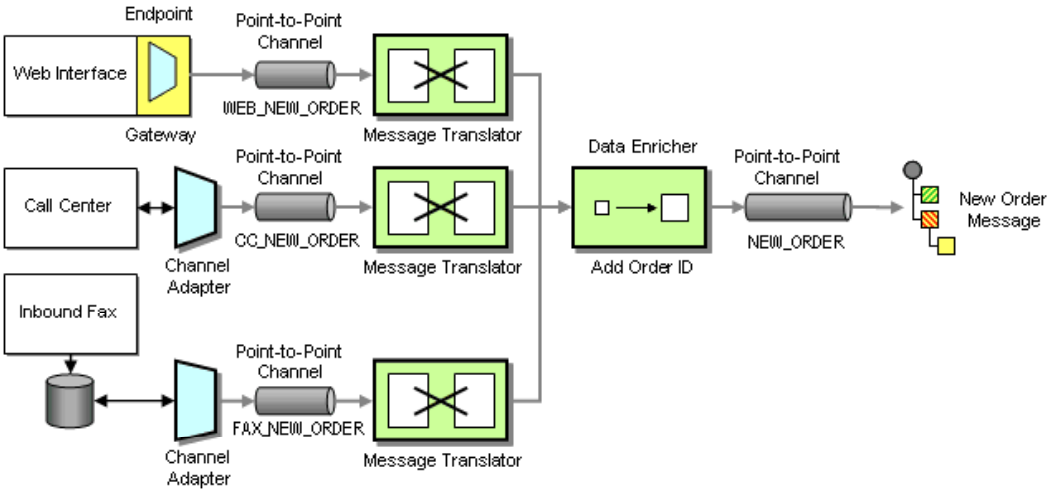
Pattern per l'EAI (5)

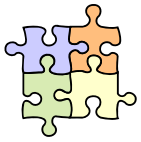


- *Splitter*
  - decompone un messaggio in un insieme di messaggi, ciascuno dei quali può richiedere una diversa elaborazione

- Identificatore d'ordine

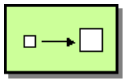
- Messaggi elaborati separatamente possono essere ricombinati mediante un Aggregator sulla base di un opportune informazioni di correlazione
  - ad es., un identificatore d'ordine
  - ma è necessario aggiungere un identificatore a ciascun ordine

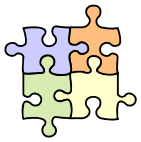




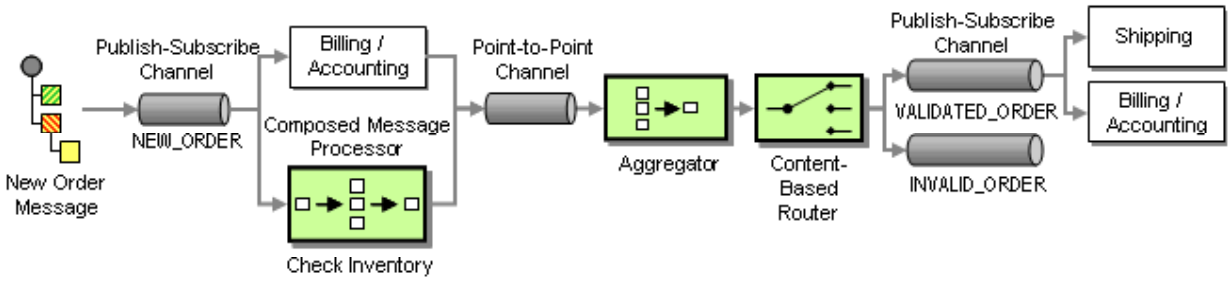
Pattern per l'EAI (6)

- *Content Enricher*
  - aggiunge informazioni ad un messaggio

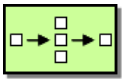




- Gestione degli ordini – rivista



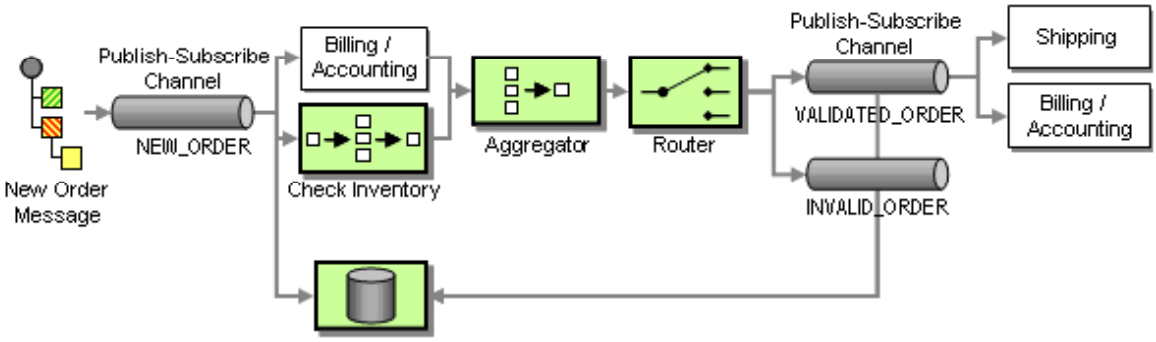
Pattern per l'EAI (7)

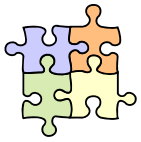


- *Composed Message Processor*
  - mantiene il flusso di messaggi complessivo, anche se i diversi messaggi richiedono elaborazioni diverse

- Verificare lo stato di un ordine

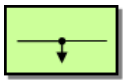
- L'elaborazione di un ordine richiede lo svolgimento di varie attività
  - come consentire ad un utente di verificare lo stato di un suo ordine? è stata effettuata la spedizione? è in attesa di prodotti? bloccato perché il cliente ha debiti in sospeso?
  - è possibile rispondere conoscendo l'"ultimo"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



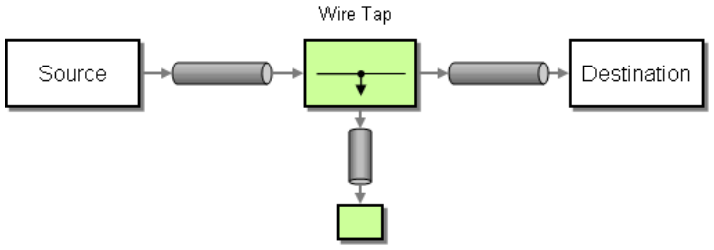


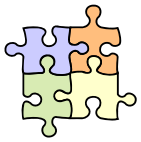
Pattern per l'EAI (8)

- *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*



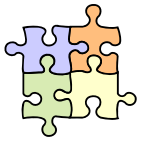
- *Wire Tap*
  - per duplicare su più canali messaggi i inviati su un canale





- Discussione

- L'esempio WGRUS mostra come usare il messaging per l'integrazione di applicazioni
  - i componenti pre-esistenti non sono connessi direttamente
  - piuttosto, l'integrazione è basata su costruzione, consumo, trasformazione, splitting, aggregazione e routing di messaggi, anche con riferimento ad un certo numero di canali di comunicazione
  - componenti pre-esistenti sono collegati al sistema di messaging mediante adattatori – componenti nuovi mediante degli endpoint, che comunque incapsulano l'accesso al sistema di messaging
- Il messaging è anche adeguato per lo sviluppo di nuove applicazioni



Discussione

- Oltre a quelli visti, [Hohpe&Woolf 2004] presenta diversi altri pattern per il messaging – alcuni dei quali ripresi da [POSA4] – per rappresentare, tra l'altro
  - elementi dei sistemi di messaging – ad es., *Message* o *Message Channel*
  - canali di messaging – ad es., *Point-to-point Channel* o *Guaranteed Delivery*
  - tipi di messaggi – ad es., *Document Message* o *Request-Reply*
  - routing di messaggi – ad es., *Splitter* o *Aggregator*
  - trasformazioni di messaggi – ad es., *Content Enricher* o *Content Filter*
  - estremità per lo scambio di messaggi – ad es., *Messaging Gateway*
  - gestione e monitoraggio del sistema – ad es., *Control Bus* o *Process Manager*