PCD Module 5.1

Ingegneria dei sistemi distribuiti: integrazione aziendale

Integrazione (delle applicazioni)

L'integrazione è una chiave moderna dello sviluppo software. Interessa applicazioni raramente vive in isolazione. Tutte le soluzioni di integrazione devono avere a che fare con alcune sfide fondamentali:

- **le reti sono inaffidabili**: la programmazione distribuita deve essere preparata ad avere a che fare con un insieme più largo di problemi possibili.
- **le reti sono lente**: ordine di magnitudo più lenta del fare chiamate a metodi locali
- **due applicazioni qualsiasi sono diverse**: linguaggi, OS, versioni, ecc.
- **il cambiamento è inevitabile**: avere a che fare con cambiamenti nelle applicazioni che sono integrate minimizzando le dipendenze (**loose coupling**)

Approcci principali

- **Trasferimento file**: un'app scrive i file che un'altra legge
- **Database condiviso**: app multiple condividono lo stesso schema del database, posizionato in un db fisico singolo.
- **Remote Procedure Call (RPC)**: un'app espone alcune delle sue funzionalità a cui le app possono avere accesso, come procedure remote. La comunicazione avviene in tempo reale e in modo sincrono
- **Messaging**: un'app pubblica un messaggio in un canale comune, altre app possono leggere tale messaggio dal canale in un secondo tempo. Le app devono mettersi d'accordo sul canale e sul formato. La comunicazione è asincrona.

La necessità dell'integrazione

Aziende del mondo reale: centinaia, migliaia di applicazioni sono costruite in mdoo personalizzato, acquistate da terzi o parte di sistemi obsoleti ed operanti su tanti sistemi operativi diversi.

Ragioni:

- scrivere applicazioni aziendali è difficile, creare una singola app enorme per tutta l'azienda è impossibile
- diffusione di business funzionali tra più applicazioni forniscono il business con la flessibilità di selezionare il "migliore" per ogni aspetto

Sfide dell'integrazione

L'integrazione aziendale non è un compito facile. Sfide:

- richiede uno spostamento significativo nelle politiche aziendali, stabilendo una comuniczione non solo tra più sistemi informatici ma anche tra BU e dipartimenti IT
- avere a che fare col poco controllo sull'integrazione che tipicamente hanno gli sviluppatori su app a cui partecipano
- adottare standard, non solo di sintassi ma di semantica
- deployment più complesso, monitoraggio dei problemi

Il mondo selvaggio dell'integrazione

La nozione di integrazione è veramente vasta. Alcuni tipi di integrazione:

- Portali di informazioni
- Repliche dei dati
- Funzioni business condivise
- Architetture orientate ai servizi
- Processi business distribuiti
- Integrazione B2B

Portali di informazione

Alcune aziende devono accedere a più di un sistema per rispondere a una specifica domanda o per eseguire certe funzioni business (es. verificare stato di un'ordine). I portali di informazioni **aggregano le informazioni da più sorgenti in una singola visualizzazione**, per evitare che gli utenti debbano accedere a più sistemi informativi.

Repliche dei dati

Molte aziende richiedono l'accesso agli stessi dati (es. indirizzi dei clienti per sistema customer care + accounting + logistica). Molti di questi sistemi hanno nei loro archivi i dati relativi ai clienti. Quando un utente cambia un'informazione, **tutti i sistemi devono cambiare la loro copia**. Questo può essere fatto implementando una strategia di integrazione basata sulla replica dei dati.

Funzioni business condivise

Per gli stessi motivi che portano molte aziende ad archiviare dati ridondanti, tendono anche a ridondare le funzionalità (es. controllare se un SSN è valido). Ha senso esporre queste funzioni come **funzioni business condivise implementate una volta** e disponibili come servizio per gli altri sistemi.

Architetture orientate ai servizi (SOA)

Le funzioni business condivise sono a volte chiamate anche **servizi**: una funzione ben definita universalmente disponibile e che risponde alle richieste da parte di un cliente del servizio.

Dal momento in cui un'azienda assembla una collezione di servizi, gestirli può essere critico:

- **directory dei servizi**
- **directory delle interfacce e descrizione dei contratti**

=> **Servizi di ricerca dei servizi e negoziazione**. Le SOA offusca la linea tra integrazione e applicazioni distribuite – fornendo strumenti per sfruttare servizi facilmente come chiamate di metodo.

Processi business distribuiti

Un driver di integrazione è che una singola transazione business può essere diffusa tra molti sistemi diversi. In molti casi tutte le funzioni rilevanti sono incorporate all'interno delle applicazioni esistenti. Ciò che manca è la coordinazione tra queste applicazioni. È possibile aggiungere un **componente di gestione dei processi business** che gestisca l'esecuzione di una funzione su diversi sistemi multipli esistenti.

Il confine tra SOA e business distribuito può essere sfocato: potremmo esporre tutte le funzioni business rilevanti come servizi e quindi codificare i processi business all'interno come applicazioni che accedono ai servizi tramite SOA.

Integrazione B2B

In molti casi le funzioni business non sono nella stessa/singola azienda, ma possono estendersi su multipli partner aziendali o fornitori. In questo caso, comunicazione tramite Internet o altre reti fa emergere nuove problematiche relative al protocollo di trasporto e alla sicurezza. Inoltre è molto importante che il formato dei dati sia standardizzato.

Accoppiamento (Coupling)

Accoppiamento debole (loose)

Serve a ridurre i presupposti/ipotesi che due parti (componenti, app, servizi, programmi, utenti..) fanno tra loro stesse quando scambiano informazioni.

Più ipotesi = più efficienza, ma meno tolleranza ai cambiamenti, interruzioni, meno flessibilità.

Accoppiamento forte (tight): esempio delle invocazioni locali ai metodi

- Basate su molti presupposti tra la routine chiamante e chiamata:

    - entrambi i metodi devono essere eseguiti sugli stessi processi
    - devono essere scritti negli stessi linguaggi (o VM)
    - i metodi chiamanti devono passare l'esatto numero di parametri attesi usando i tipi di dati accordati
    - la chiamata è immediata: i metodi chiamati iniziano a processare immediatamente dopo che il metodo chiamante fa la chiamata
    - i metodi chiamanti ripristineranno il processo solo quando il metodo chiamato completa
    - la comunicazione è immediata e istantanea

- Tutti questi presupposti rendono molto facile il lavoro di scrivere applicazioni strutturate bene che dividono le funzionalità in metodi individuali che possono essere chiamati da altri metodi. Il risultato è un grande numero di piccoli metodi che permettono flessibilità e riuso.
- Diversi approcci integrativi hanno aiutato a creare comunicazioni remote semplicemente impacchettando uno scambio di dati remoto con la stessa semantica dei metodi locali (RPC).
    - la semantica delle chiamate ai metodi sincroni sono molto familiari agli sviluppatori, perciò perchè non usarla?
    - utilizzando la stessa sintassi e semantica per entrambe le chiamate a metodi locali e remoti ci dovrebbe permettere di differire a tempo di deploy la decisione riguardo quali componenti dovrebbero essere eseguiti localmente e quali remotamente.

E quindi?

Il problema

Il problema è che **le chiamate remote invalidano molti dei presupposti su cui sono basate le chiamate locali**. Come risultato, astrarre la comunicazione remota in una semantica semplice di una chiamata a metodo può confondere e ingannare.

Esempio: una chiamata tramite la rete tende ad essere molto più lenta di una chiamata locale. Il metodo quindi deve aspettare? E se la rete si interrompe? Quanto dovremmo aspettare?

Diventa chiaro che l'integrazione remota porta a galla tanti problemi che un metodo locale non avra mai a che fare.

Un esempio di integrazione di applicazione aziendale

- Online banking
- Permette di depositare soldi da altre banche
- Per fare ciò, il frontend ha bisogno di essere integrato con il backend finanziario.

```
String hostname = "finance.bank.com"
int port = 80
IpHostEntry hostInfo = Dns.GetHostByName(hostName);
IpAddress address = hostInfo.AddressList[0];
IPEndPoint endpoint = new IPEndPoint(address, port)
Socket socket = new Socket(address.AddressFamily,
                           SocketType.Stream, ProtocolType.Tcp);
socket.Connect(endpoint);
byte[] amount = BitConverter.GetBytes(1000);
byte[] name = Encoding.ASCII.GetBytes("joe");
int byteSent = socket.Send(amount);
byteSent += socket.Send(name);
socket.Close();
```

Problemi e presupposti

La soluzione minimale fa tali presupposti:

- **tecnologia della piattaforma**
  - indipendenza debole dalla piattaforma - funziona solo con flussi di byte
  - rappresentazione eterogenea di numeri, oggetti, codifiche
- **locazione**
  - se muovessimo la funzione su un computer diverso su dominio diverso? Se le macchine fallissero e avessimo bisogno di altre macchine? Dovremmo cambiare il codice. Dal momento che utilizziamo funzioni remote, diventa un incubo molto velocemente. Serve trovare un modo per rendere la comunicazione indipendente da una specifica macchina della rete
- **tempo**
  - Tutti i componenti devono essere disponibili allo stesso tempo
  - TCP-IP è orientato alla connessione
  - Se qualcuna delle parti coinvolte nella comunicazione non fosse disponibile in tal momento, il dato non può essere inviato
- **formato dei dati**
  - La lista dei parametri e i tipi devono combaciare
  - Se volessimo inserire un terzo parametro, avremmo bisogno di modificare sia il mittente che il destinatario

Rimuovere le dipendenze

- **dipendenti dalla piattaforma**: utilizzando formati dati standard (XML, JSON, ecc...)
- **relative alla locazione**: invece che inviare informazioni a una macchina, le si mandano ad un **canale** indirizzabile. Un canale è un indirizzo logico su cui mittente e destinatario devono mettersi d'accordo senza conoscere la relativa identità.
- **relative al tempo**: si accodano richieste inviate finchè la rete e il sistema ricevente non è pronto. Accodare le richieste dentro al canale richiede che i dati vengano spezzati in messaggi autocontenuti affinchè il canale conosca quanti dati deve bufferare ed inviare in qualsiasi momento.
- **sul formato dati**: permettendo trasformazioni del formato dati

Interazione poco accoppiata

I meccanismi come i formati dati comuni, la comunicazione asincrona su canali con code e i trasformatori aiutano una soluzione fortemente accoppiata a farla diventare poco accoppiata.

Svantaggio principale: **complessità addizionale**. Soluzioni con codice più lungo, modelli di programmazione più complessi che rendono la progettazione, la costruzione e il debugging più complesso.

Elementi base

- **Canale di comunicazione** per muovere le informazioni da un'app all'altra
- **Messaggi** con dati spezzettati, con un significato già preaccordato
- **Traduzione** per supportare i formati dati interni delle varie applicazioni
- **Instradamento** per muovere i dati verso un determinato posto e indirizzo
- **Funzioni di gestione del sistema** per monitorare il flusso dei dati, assicurarsi che tutte le app e i componenti siano funzionanti, riportando condizioni di errore.