**CAPITOLO 11: MESSAGING**

*In questo capitolo sono illustrate le scelte intraprese per la realizzazione di una versione dell’applicazione che utilizza il paradigma di comunicazione asincrona basato su messaggi, code e topic.*

**11.1 Premessa**

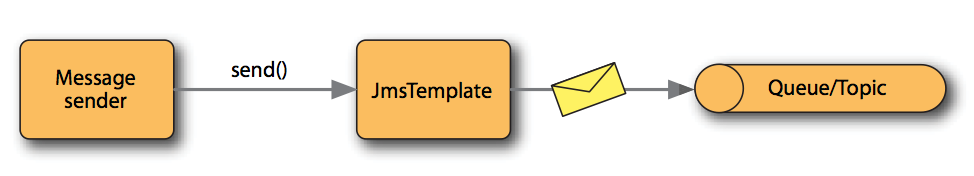
Molto spesso vi è l’esigenza di integrare componenti o servizi sviluppati indipendentemente. Tali componenti devono essere integrati per formare un sistema coerente ed interagire in modo affidabile mantenendo, però, basso l’accoppiamento. Per ottemperare a tali obiettivi, è stata prevista una versione del sistema in cui è presente lo scambio asincrono di messaggi tra componenti mediante un bus per messaggi.

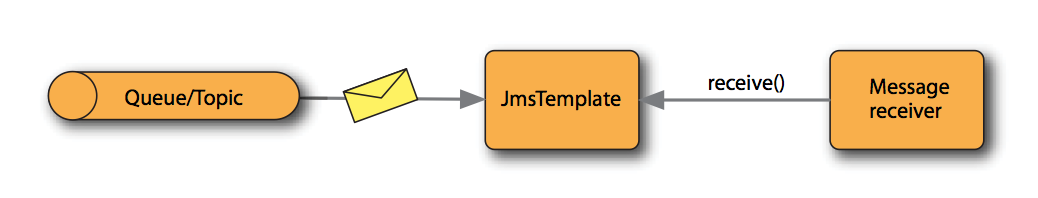
**11.2 Messaging**

Il messaging è una tecnologia di comunicazione per sistemi distribuiti in cui i componenti o le applicazioni comunicano scambiandosi messaggi in una relazione peer-to-peer. Più nello specifico un componente nel ruolo di produttore può inviare un messaggio a un altro componente che riveste il ruolo di consumatore. Tali messaggi sono scambiati mediante un canale di comunicazione intermedio. Esistono due tipi di canali di comunicazione: le code ed i topic o argomenti. Le code sono un canale di comunicazione “a uno” intendendo che un messaggio inviato a tale destinazione intermedia sarà consumato da uno e un solo consumatore. Nei topic invece il messaggio viene ricevuto da tutti i consumatori registrato presso la destinazione intermedia. Per tale motivo gli argomenti sono un canale di comunicazione “a molti”. Tra le caratteristiche più importanti del messaging c’è il fatto che la comunicazione viene iniziata dal componente produttore del messaggio, è indiretta ed asincrona.

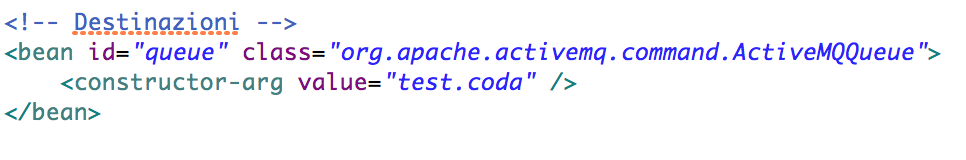
**11.3 Spring JMS**

Spring Framework rende più semplice lo sviluppo di applicazioni che necessitano di usare JMS. Per aiutare lo sviluppatore è stato creato un apposito modulo chiamato Spring JMS. Lo scopo di tale frammento dell’ecosistema Spring è quello di evitare di semplificare il lavoro del programmatore in fase di configurazione delle destinazioni intermedie ed evitare di riscrivere sempre le stesse porzioni di codice. Per ricevere o inviare messaggi è possibile utilizzare la classe JmsTemplate. Tale classe permette di tralasciare tutte le istruzioni che altrimenti sarebbero necessarie per effettuare la connessione a un provider JMS per pubblicare o ricevere messaggi.





Tutte le informazioni relative al provider che si intende usare per la gestione delle destinazioni intermedie cosi come il loro nome è definito in un file di configurazione.



Il frammento appena inserito è necessario per definire una coda di nome “test.coda”.

**11.4 car2go-jms**

Per mostrare l’uso di JMS è stato realizzato un semplice caso d’uso. Come spiegato nei capitoli procedenti, un utente che intende noleggiare una vettura, dopo averla prenotata e prima di mettersi alla guida, deve indicare, tra le altre cose, se la macchina presenta delle anomalie nella carrozzeria. Se l’utente segnala un qualche tipo di problema, l’amministratore potrà richiedere di riparare la macchina. Il caso d’uso realizzato va a completare ed ampliare tale funzionalità. Se l’utente segnala che lo stato esterno della macchina è danneggiato (selezionando la voce “non sufficiente” dal menù a tendina dalla pagina per effettuare la guida) la macchina invierà ad una coda il tipo di guasto che la affligge. Più in dettaglio è stata creato un elenco di guasti ed il componente che gestisce le macchine, automaticamente in maniera casuale, ne selezionerà uno e lo invierà in una coda. Il componente consumatore che è registrato presso la coda processerà tali messaggi salvandoli nel database. In questo modo quando l’amministratore vorrà far riparare le vetture, oltre all’indicazione delle auto che necessitano di essere riparate, avrà le informazione sui malfunzionamenti che sono presenti in tale vettura. Per realizzare tale funzionalità sono quindi stati usati due tipi di componenti (oltre alla coda dove giungono i vari guasti). E’ stato realizzato un semplice elemento sender, il cui scopo è quello di inviare i messaggi ad una specifica coda. I messaggi sono della forma “nomeGuasto,idMacchina”. Il ricevitore è un consumatore asincrono. Ogni qualvolta arriverà un messaggio alla coda, verrà richiamato dal container il metodo onMessage che ricaverà i dati necessari da rendere persistenti sul database.