LABORATORIO DI INGEGNERIA DEI SISTEMI SOFTWARE

Introduction

Un caso di studio con cui iniziare ad affrontare l'analisi e la progettazione di sistemi distribuiti è un sistema formato da a N Produttori che inviano informazione a 1 Consumatore.

Requirements

Costruire un sistema software distribuito costituio da N (con N >= 1) Produttori che inviano informazione a 1 Consumatore, il quale deve elaborare tale informazione.
a dislocazione dei componenti sui nodi di elaborazione può essere:

- OneNode: tutti i componenti operano nello stesso nodo;
- TwoNodes: gli N Produttori operano in uno stesso nodo, mentre il Consumatore opera in un diverso nodo;
- ManyNodes: il Consumatore opera in suo proprio nodo, mentre i Produttori operano su K nodi diversi (con K maggiore di 1, e al più uguale a N).

Requirement analysis

- Produttore e Consumatore rappresentano gli unici due tipi di entità interagenti necessari all'interno del sistema.
- I requisiti non stabiliscono come l'informazione debba essere trasmessa e ricevuta.
- Le modalità di scambio di informazioni fra Produttori e Consumatore non sono stabilite: in particolare, non si precisa se il Produttore debba ricevere una risposta dal Consumatore.

Problem analysis

Architettura logica

Stando all'analisi dei requisiti, l'architettura dovrà necessariamente prevedere una modalità di invio di messaggi dai Produttori al Consumatore. Sulla base di questo, l'architettura logica del sistema può essere organizzata in diversi modi, ad esempio:

- **Modello Client-Server**: l'architettura prevede diversi client che effettuano richieste verso un server. In questo caso, i client sono rappresentati dai Produttori, mentre il server è dato dal Consumatore. Stando ai requisiti, l'interazione può essere asincrona non bloccante (le richieste dei client possono essere "fire & forget"). In questo modello, i Produttori devono conoscere il Consumatore, mentre non è vero il viceversa.
- **Publisher-Subscriber (Pub/Sub)**: In questo caso, è presente un gestore della comunicazione (broker), a cui i publisher (i Produttori) e il subscriber (il Consumatore) sono iscritti; quest'ultimo riceverà i messaggi dal broker, che li avrà ricevuti precedentemente dai publisher. In questo modello, i Produttori e il Consumatore conoscono il broker.
- **Modello ad Attori (Message-Driven)**: in questo caso, Produttori e Consumatore sono rappresentati da Attori, entità dotate di comportamento autonomo, capaci di ricevere e trasmettere messaggi. Gli Attori sono inseriti all'interno di un contesto, che ne consente l'interazione con l'esterno.

In ogni caso, per comunicare, le due parti dovranno concordare un coerente formato dei messaggi.

Ulteriori considerazioni

Come già specificato, è necessario che si preveda una modalità di invio delle richieste dai Produttori al Consumatore, mentre non è necessario il viceversa. Qualora non si intenda sviluppare una modalità di risposta al Produttore (Fire & Forget), quest'ultimo non avrebbe modo di verificare la corretta ricezione della richiesta.

Test plans

I piani di test prevedono la verifica di una richiesta, proveniente da uno dei nodi Produttore per il nodo Consumatore, e sarà necessario verificare:

- il corretto invio dei messaggi da parte dei Produttori.
- la corretta ricezione dei messaggi da parte del Consumatore.
- il corretto formato dei messaggi.

Qualora si intenda sviluppare una modalità di risposta al Produttore, diventa necessario prevedere le analoghe verifiche anche per tale messaggio.

Project

La scelta di progettazione ricade sul **Modello ad Attori** accennato nella sezione precedente. Gli Attori sono entità attive, inserite all'interno di un contesto che si occupa di gestirne l'interazione con l'esterno, e mantiene una mappa di tutti gli Attori al suo interno. Gli Attori potranno inviare messaggi ad altri Attori, inserendoli nella coda del destinatario, oppure possono elaborare i messaggi prelevandoli dalla propria coda. Si decide di procedere implementando Produttori e Consumatori mediante l'utilizzo del **Observer Pattern**; in particolare, per abbattere il costo di progetto, la progettazione avviene sulla base delle interfacce *IActor24* e *IContext24*, e della classe *ActorBasic24* già fornite. Vediamo le componenti fondamentali del sistema, sulle quali baseremo la struttura di Produttori e Consumatori:

- Interfaccia **Observable**: rappresenta il contratto che tutte le entità osservabili debbono rispettare. In particolare, dichiara i metodi per l'aggiunta e la rimozione di un'entità osservatrice, nonché il metodo di *update* utilizzato per l'aggiornamento degli Observer.
- Classe Astratta **ObservableActor** [*extends ActorBasic24, implements Observable*]: rappresenta l'implementazione base di un'entità osservabile nel contesto degli Attori.
  In particolare, implementa i metodi per l'aggiunta e la rimozione di un'entità osservatrice, e del metodo di aggiornamento degli Observer; non si occupa di implementare la primitiva applicativa di gestione dei messaggi.

Sulla base di queste componenti, vediamo la struttura di Produttori e Consumatori:

- **Produttore**: Classe **ObservableProducer** [*extends ObservableActor*]. La classe rappresenta un Attore come entità osservabile atto all'invio di messaggi a un Consumatore, e implementa la primitiva applicativa per l'invio di messaggi.
  Ad ogni invio, si occuperà di inviare un *update* a tutti gli Observer nella sua lista.
- **Consumatore**: Classe **ObservableConsumer** [*extends ObservableActor*]. La classe rappresenta un Attore come entità osservabile atto alla ricezione di messaggi inviati da un Produttore. Implementa la primitiva applicativa per la gestione di messaggi, gestendo le richieste in modo opportuno, attraverso l'invio di una risposta.

Ad ogni risposta, anch'esso si occuperà di inviare un *update* a tutti gli Observer nella sua lista.

- In questa prima implementazione, i Produttori dovranno conoscere il nome dei Consumatori per poter recapitare il messaggio nella coda dell'Attore corretto.
- Come nei casi analizzati nelle precedenti lezioni, si prevedono due modalità di invio delle richieste: Request e Forward.
  - Per un messaggio Request, il Produttore si aspetterà di ricevere una risposta da parte del Consumatore: si prevede pertanto l'invio di un messaggio di ACK (acknowledgement) da parte del Consumatore per comunicare la ricezione della richiesta al Produttore.
  - Per un messaggio Forward, il Produttore non si aspetta una risposta, e non è prevista quindi nessuna ulteriore gestione della comunicazione in tale scenario (Fire & Forget).

Testing

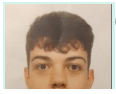
Il testing del sistema è stato effettuato utilizzando JUnit. Per poter verificare la correttezza dell'invio di messaggi, delle relative risposte, e degli aggiornamenti inviati, si utilizza la gestione di un sistema di logging attraverso la seguente entità:

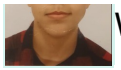
- Classe **ObserverLogger** [*extends ActorBasic24*]: si tratta di un Attore, registrato come Observer presso Produttore e Consumatore, che si occuperà di ascoltare l'invio degli aggiornamenti relativi all'invio di messaggi e risposte da parte di entrambe le parti, e di scriverle a coppie in un file di log *LogProdCons.txt*.

Il programma di test andrà a eseguire molteplici scambi di messaggi fra Produttore e Consumatore e, sfruttando il Logger, andrà a verificare la correttezza del formato dei messaggi, il corretto invio delle richieste e la corrispondenza con le relative risposte leggendo i messaggi scambiati direttamente dal file di log.

Deployment

Maintenance

By Tommaso Sgreccia - Email Istituzionale:  GIT repo: https://github.com/

tommaso.sgreccia@studio.unibo.it  WheelyMcBones/ISS2024