

LABORATORIO DI INGEGNERIA DEI SISTEMI SOFTWARE

Introduction

L'oggetto di studio è un sistema formato da N Produttori che inviano informazione a 1 Consumatore

Requirements

Costruire un sistema software distribuito costituito da N ($N \geq 1$) Produttori che inviano informazione a 1 Consumatore, il quale deve elaborare tale informazione. La 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 $N \geq K > 1$)

Requirement analysis

1. I requisiti non esprimono quale protocollo di comunicazione specifico sfruttano consumatore e produttori
2. I requisiti non esprimono quale linguaggio specifico o tecnologia specifica utilizzare
3. Teoricamente, consumatori e produttori potrebbero anche sfruttare linguaggi diversi
4. Produttore e Consumatore sono enti computazionali attivi e autonomi

Problem analysis

Interazione logica

I requisiti non specificano se la comunicazione tra produttori e consumatore debba essere sincrona oppure asincrona. Se i producer dovessero aver bisogno di un feedback ad ogni invio da parte del consumer prima di continuare, allora sarebbe più opportuno sfruttare un protocollo sincrono (come HTTP), altrimenti si potrebbe valutare un protocollo asincrono

Architettura logica

Si potrebbe pensare di sfruttare un'architettura di tipo client-server, in cui i Producer sono i

client, e il Consumer il server. Questo obbligherebbe i producer a conoscere l'indirizzo del Consumer, ma non il Consumer a conoscere gli indirizzi dei Producer (a meno che non sia importante distinguere chi sia un Producer "lecito" e chi no). Si potrebbe pensare di sfruttare un'architettura di tipo publish-subscribe, in cui i client consegnano in una cassetta postale un messaggio e non conoscono però chi potrebbe essere il mittente, ma solo dove consegnare il messaggio. Si potrebbe anche pensare di utilizzare una comunicazione ad eventi, che permetta ai client di emettere un evento senza però sapere chi potrebbe riceverlo.

Test plans

Per poter predisporre un piano di test per questo tipo di applicazione si potrebbe pensare di far mandare un messaggio da uno o più Producer verso il Consumer, e verificare che questo li abbia ricevuti correttamente, e che li sappia interpretare nel modo corretto

Project

Alternativamente all'implementazione in project0 si potrebbe pensare di sviluppare una soluzione che si basi sul concetto di actor: queste entità sono implementate con una classe che consente di inviare dei messaggi, di elaborare quelli ricevuti (eventualmente anche da parte di alieni). Questo permette di realizzare un'implementazione basata su concetti di più alto livello, e quindi di concentrarsi maggiormente sulla business logic rispetto a come effettivamente viene implementata a basso livello. Viene fornita anche un'implementazione di actor che sia "observable", ovvero che permetta ad altri attori di registrarsi ad una lista apposita, e che ogni volta che lo desidera può inviare loro dei messaggi di dispatch, ad esempio per aggiornarli. L'actor observable è stato pensato come implementazione di un actor di base, in quanto può essere visto come specializzazione del concetto di actor di base, che invece ha soltanto lo scopo di inviare e ricevere messaggi. Inoltre, si è deciso di sfruttare il pattern observer: in questo modo gli actor observable contengono una lista di attori che li osservano un metodo che consente l'aggiunta di attori in questa lista, uno per la rimozione di attori da questa lista, e un metodo per notificarli tramite dispatch. In questo modo si riuscirebbe a rendere più pulito il piano di testing: a differenza di quanto fatto in precedenza, in cui occorre che il consumer registrasse in un file di log i messaggi ricevuti/inviati, qui basterà avere uno o più attori osservatori a cui verrà delegato il compito di trascrivere su appositi file di log i messaggi ricevuti/inviati tramite i messaggi di update.

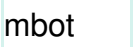
Testing

```
//Creazione dei contesti ctx1 = new ActorContext24("ctxprodcons", "localhost", 8223);  
//creazione degli attori prod = new ProducerAsObservableActor("prod1", ctx1); consumer = new  
ConsumerAsObservableActor("consumer", ctx1 ); obsLogger = new ObsLogger("observer1",  
ctx1, "obsloggerLog.txt"); prod.registerObserver(obsLogger);  
consumer.registerObserver(obsLogger); //test di updateResource try {  
prod.updateResource("res1"); CommUtils.delay(500); testLogFile("res1"); } catch (Exception e) {  
fail("testRequest " + e.getMessage()); }
```

Deployment

Maintenance

By studentName email: gabriele.daga@studio.unibo.it,

 GIT repo: <https://github.com/dagus01-lab/issLab2024>