Proiectul Nr.1 (8 ore)

Tema:

Agent de mesagerie. Invocarea la distanță.

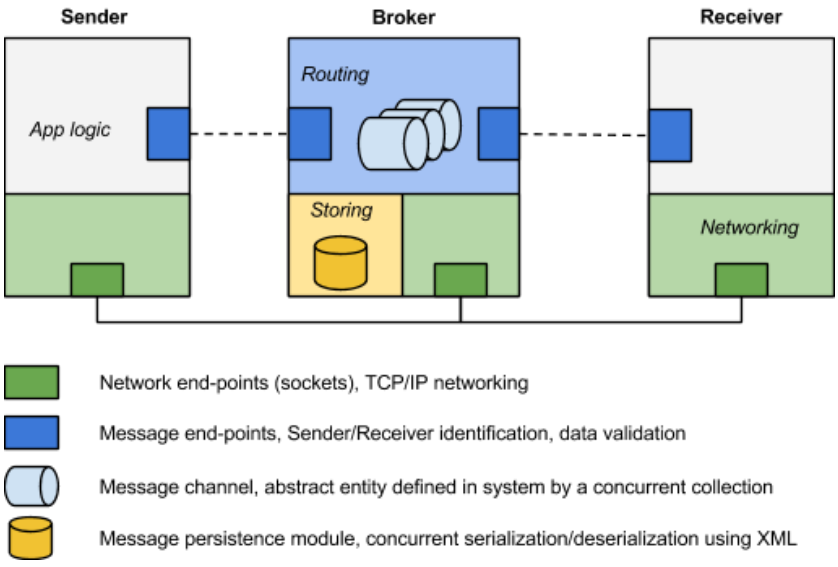
Scopul lucrării:

Integrarea bazată pe agenți de mesaje care ar permite o comunicare asincronă dintre componentele distribuite ale unui sistem. Studierea mecanismului de invocare la distanță.

Obiectivele lucrării:

Realizarea sistemului utilizînd „Socket-uri” (Partea 1) și realizarea sistemului utilizînd invocările la distanță, “RPC-uri” (Partea 2).

Partea 1



1. Definirea protocolului de lucru al agentului de mesaje [1]
    1. Formatul (tipul) mesajelor de transmis: XML/JSON
    2. Numărului de canale unidirecționale (variabil/fix, dependent de tipul mesajelor, etc.)
    3. Structura comunicației asigurată de agent (unul-la-unu)
    4. Politici de livrare pentru diverse cazuri definite de logica de lucru al agentului (mesaje invalide, căderea agentului, etc.)

Datele între elemente vor fi transmise în format XML/JSON, deci sender-ul va fi cel care va serializa mesajul. Deserializarea poate avea loc atît pe partea brokerului cît și pe partea de receiver, mesajul nu trebuie sa fie alterat în timpul transmiterii. Numărul de canale de comunicare poate fi setat în momentul creării socketului (argumentați alegerea voastră). Fiecare receiver primește mesaj de un anumit tip, brokerul trebuie să efectueze rederecționarea corectă în dependență de tipul mesajului (tipul mesajului poate fi inclus chiar în corpul mesajului și brokerul va fi nevoit să parseze mesajul ca să afle tipul lui). Nu uitați să tratați excepțiile în caz de transmitere a unor mesaje invalide pentru a evita căderea agentului.

2. Elaborarea nivelului abstract de comunicare (rețea) necesară elementelor pentru primirea/transmiterea mesajelor de către emițător-agent-receptor;
    1. Protocolul de transport se alege în dependență de obiectivele protocolului de lucru
    2. Tratarea concurentă a cererilor.

Alegeți protocolul de transport care după părerea voastră ar fi mai eficient (argumentați alegerea TCP/UDP). Protocolul la fel poate fi indicat la setarea socketului. Cererile trebuie să fie tratate concurent. Deci asta ar presupune ca canalul de I/O să nu fie blocat. Poate fi utilizată metoda **thread per request**. Fiecare cerere să fie prelucrată de un fir de execuție separat care nu ar bloca canalul de comunicare. La broker putem avea un set de work joburi pornite (fire de execuție care execută careva instrucțiuni la un interval specific de timp. Work joburile mai sunt numite și Cron Joburi.) Aceste work joburi vor prelucra concurent mesajele stocate în storage la broker și le vor transmite la receiveri.

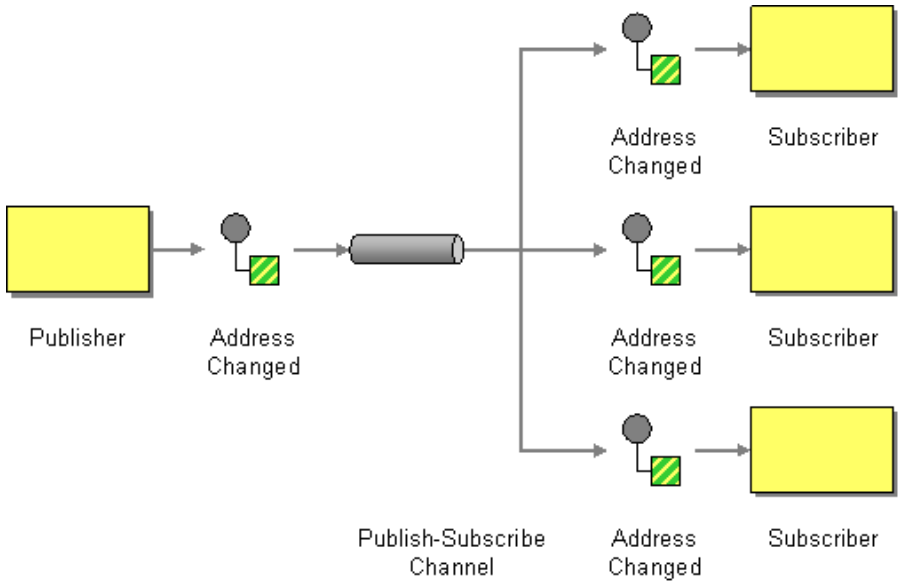
3. Elaborarea elementelor ce asigură păstrarea mesajelor primite
    1. Metoda transientă: mesajele vor fi stocate în colecții concurente de date specifice limbajului selectat.
    2. Metoda persistentă: mesajele vor fi serializate/deserializate utilizînd metode de procesare asincronă sau concurentă.
    3. Elaborarea nivelului abstract de rutare a mesajelor.

Dacă limbajul utilizat deține colecții thread-safe, atunci se recomandă utilizarea lor. Dacă nu – va fi nevoie de implementat un mecanism concurent propriu (utilizînd lock-uri, etc.)

Implementarea șablonului Publisher / Subscriber

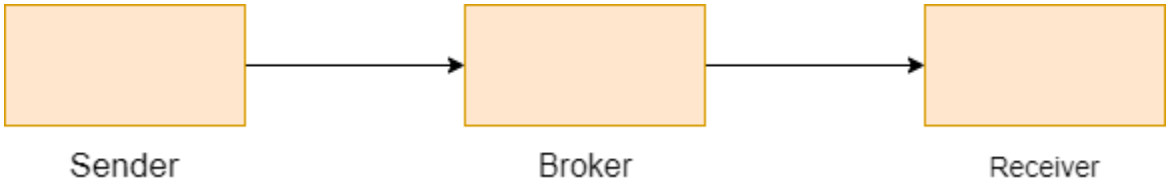
În timp ce message pooling este relativ simplu de pus în aplicare, ar putea pune o mulțime de presiuni de rețea asupra brokerului de mesaje în cazul în care există o mulțime de consumatori. În mod normal, brokerii de mesaje oferă o interfață bazată pe evenimente prin implementarea modelului Publisher / Subscriber. În acest fel, producătorii publică în continuare mesaje la o anumită coadă, totuși consumatorii „se abonează" la mesaje noi de la anumite cozi. Modul de implementare a mecanismului de rutare, trebuie să decideți. Implementarea de rutare se poate baza pe cozi în mod explicit sau poate introduce o abstractizare diferită care poate fi utilizată pentru rutare (de exemplu, subiectul mesajului).

Subiectul mesajului: Subiectul mesajului este un identificator logic al unui canal de comunicare între producători și consumatori prin intermediul brokerului de mesaje. : Rolul subiectului este destul de asemănător cu o coadă de mesaje unică - care grupează mesaje pentru a le putea direcționa. : Cu toate acestea, subiectele permit să se rezume la implementare, făcând mecanismul de rutare mai transparent atât pentru producători cât și pentru consumatori.



Pentru a obține o notă minimă este necesar de efectuat:

Un sistem simplifcat alcătuit dintr-un **sender, broker și receiver**. Brokerul va prelua mesajul de la sender și îl va transmite Receiverului. Brokerul cunoaște adresa Receiverului din start, astfel nu este necesar de implementat mecansimul de subscripție la Broker și nu este necesar de implementat un storage pentru mesaje, ele vor fi redirecționate de Broker imediat după ce au fost primite. Brokerul are doar obiectivul de a redirecționa mesajul. Va fi un plus dacă cererile vor fi prelucrate concurent.



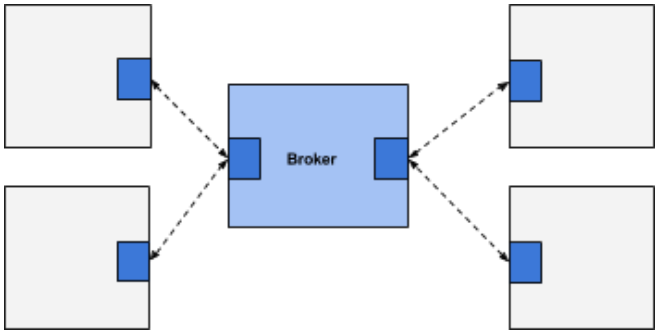
Partea 2

Dacă în cadrul primei părți a fost realizată arhitectura sistemului, au fost decuplate componentele și s-a implementat totul la un nivel mai jos (utilizind Socketuri), păi la a doua parte - se folosesc RPC-uri, ceea ce mărește nivelul de abstracție și permite focusarea pe alte lucruri decît pe modul cum are loc în detalii comunicarea între componente. Aici se va face cunoștință cu noi forme de prezentare a datelor (gRPC - proto, Apache Thrift - thrift, etc.) în dependență de RPC-ul ales. Deci în principiu - arhitectura sistemului rămîne aceeași, dar implementarea diferă. Astfel, toate punctele enumerate în partea 1 trebuiesc respectate și în partea 2, însă cu unele modificări:

1. Utilizarea unui RPC în loc de Socketuri (gRPC, Apache Thrift, etc.) (vezi referințele)
2. Reprezentarea datelor: proto/avro/thrift/graphql.
3. Nu uităm de validarea datelor și procesarea concurentă.

Considerații generale

Agentul de mesaje (message broker - eng.) este o componentă fizică care gestionează comunicarea dintre componetele unei aplicații distribuite. Avantajul utilizării acestei tehnici constă în decuplarea receptorului de transmițătorul mesajelor. Prin urmare o aplicație participantă transmite mesaje doar agentului, indicînd un nume logic al receptorului.



Agentul poate expune diverse interfețe aplicațiilor în colaborare și poate transfera mesajele între acestea, ne impunînd o interfață comună tuturor participanților întru asigurarea interacțiunii. Responsabilitățile și colaborările esențiale ale unui broker de mesaje sunt prezentate în tabelul de mai jos.

Responsabilități	**Colaborări**
Primirea mesajelor	Expiditori: aplicații (componente) ce trimit mesaje agentului
Determinarea destinatarilor și efectuarea rutării	Receptori: aplicații (componente) ce primesc mesaje de la broker
Tratarea diferențelor dintre interfețe	

Transmiterea mesajelor

Decizia de a utiliza brokerul de mesaje pentru integrarea aplicațiilor balansează între flexibilitatea primită prin decuplarea participanților și efortul de a menține brokerul [2]:

1. Beneficii
    1. Reduce cuplarea - transmițătorii comunică doar cu brokerul, astfel o potențială grupare a mai multor receptori sub un nume logic comun poate deveni transparentă transmițătorilor;
    2. Mărește integrabilitatea - aplicațiile care comunică cu brokerul nu trebuie să aibă aceiași interfață, astfel brokerul poate deveni o punte dintre aplicații cu diferite nivele de securitate și calitate a serviciilor QoS;
    3. Mărește evolutivitatea - brokerul protejează componentele de modificările individuale ale aplicațiilor integrate, deseori oferînd capacități de configurare dinamică;
2. Constrîngeri
    1. Crește complexitatea - brokerul comunicînd cu toți participanții trebuie să implementeze multiple interfețe (protocoale) și în perspectiva performanței utilizează multithreadingul;
    2. Crește efortul pentru mentenanță - toți participanții trebuie să fie înregsitrați la brocker și se cere un mecanism de identificare a acestora;
    3. Reduce disponibilitatea - o singură componentă care intermediază comunicarea este singurul punct de eșec (single point of failure - eng.), căderea acestuia implică

blocarea activității întregului sistem; această problemă se remediază prin dublarea brokerului și sincronizarea stărilor agentului primar și secundar;

4. Reduce performanța - agentul de mesaje adaugă un pas adăugător, care implică cheltuieli suplimentare (overhead - eng.).

Recomandări:

Utilizarea colecțiilor **Thread Safe** pentru a implementa unitatea de stocare a mesajelor în broker (storage) pentru varianta transientă.

Referințe:

1) About Message Brokers: https://www.ibm.com/cloud/learn/message-brokers

2) Exemplu de client/server bazat pe socketuri in Java: https://www.baeldung.com/a-guide-to-java-sockets

3) Colectii Thread Safe in C#: https://www.tutorialspoint.com/Thread-Safe-Concurrent-Collection-in-Chash

4) JSON vs protobuf: https://auth0.com/blog/beating-json-performance-with-protobuf/

5) Thrift vs Avro vs Protobuf: https://stackoverflow.com/questions/40968303/thrift-avro-protocolbuffers-are-they-all-dead

6) Apache Thrift: https://thrift.apache.org/?fbclid=IwAR37GVZMBDfPar-d_OjivuPgZvlTlmVoxSiA5sCn0hUyl-VNw263h82rj0E

7) Introducere gRPC: https://grpc.io/docs/what-is-grpc/introduction/