LABORATORIO DI INGEGNERIA DEI SISTEMI SOFTWARE

Introduction

Requirements

Costruire un sistema software distribuito costituio da N (N>=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 (1<\K<=N).

Requirement analysis

Produttore: Entità che invia informazioni.
Consumatore: Entità che riceve informazioni.

Il metodo di comunicazione non è stato dfinito dai requisiti.
Per questo sprint verrà considerato come protocollo di comunicazione TCP e comunicazione diretta tra Produttore e Consumatore.

Il contenuto dell'informazione non è stato definito dai requisiti.
Per trasmettere informazioni verrà usato il formato dei messaggi definito dalla libreria in IApplMessage.

Produttori e Consumatori devono poter essere eseguiti sia sulla stessa macchina che in macchine differenti.

Problem analysis

L'utilizzo del protocollo TCP implica come minimo la conoscenza da parte di produttori del consumatore, che potrebbe essere un problema di sicurezza

Non è specificato dei requisiti in che modo il consumatore processa i messaggi, per questo sprint si adotterà una strategia FIFO ma un'implementazione diversa basata sul filtraggio dei messaggi per tipo di richiesta potrebbe essere rilevante per la gestione del carico computazionale.

La comunicazione verrà fatta asincrona per avere più flessibilità nella gestione dei messaggi.
Per mantenere sicurezza sulla ricezione del messaggio da parte dell'consumatore quest'ultimo dovrà inviare un messaggio simil-ack al termine della gestione del messaggio al rispettivo produttore.

La scelta di usare il protocollo TCP per la comunicazione implica che il Consumatore deve necessariamente essere online prima dell'invio dei messaggi dei Produttori, altrimenti i messaggi verranno persi.

La logica applicativa dei Produttori e dei Consumatori dovrebbe essere il più debbolmente accoppiati possibile dalla parte di messaggistica, considerando anche che il sistema di messaggi potrebbe diventare anchesso distrubuito in un futuro a seconda dell'implementazione e dalla scalabilità del sistema.

Test plans

Il testing deve poter essere svolto in maniera automatizzata.

Il disaccoppiamento delle singole parti del sistema semplifica il testing delle componenti.

Project

Il progetto sarà suddiviso in due parti fondamentali: il Produttore ed il Consumatore.
Verrà poi utilizzato il main come bootstrap del sistema.

Il Produttore è un POJO che invia messaggi al Consumatore secondo la specifica della libreria unibo.basicomm23.Interaction e sfruttando questo costrutto stesso.

Il Produttore sfrutta un oggetto di tipo Interaction per inviare un messaggio in modalità fire and forget, ma attende comunque una risposta di conferma dal consumatore per garantire l'arrivo del messaggio a destinazione.

In seguito vengono mostrate le funzionalità principali del Producer.

```
public class ProducerTCP {
    private String hostIp;
    private int hostPort;
    private int id;
    private String consumer;
    private Interaction interaction;

    public ProducerTCP() {
        hostIp="127.0.0.1";
        hostPort=8011;
        id=1;
        consumer="servicemath";
        ConnectionFactory conFac=new ConnectionFactory();
        interaction=conFac.createClientSupport(ProtocolType.tcp,hostIp, Ir
    }

    public void doJob() {
        try {
            this.getInteraction().forward(BasicMsgUtil.buildRequest("Produ
            IApplMessage msg=this.getInteraction().receive();
            System.out.println("Producer "+ this.getId()+" msg received: "
            this.getInteraction().close();
        }catch(Exception e) {
            //messaggio errore
        }
    }
}
```

Il Consumatore è suddiviso in due classi, una per la ricezione delle comunicazioni ed una per la gestione della logica per la gestione del singolo messaggio, disaccoppiando la logica applicativa da quella di supporto alla messaggistica

La classe ConsumerTCP è la classe che funge da server tcp per la ricezione di comunicazioni da parte dei Producer sfruttando ServerSocket, poi passerà l'esecuzione alla parte di logica e riprendere l'attesa di nuove comunicazioni.

In seguito viene mostrato il funzionamento di ConsumerTCP:

```
public class ConsumerTCP extends Thread{
    private ServerSocket server;
    private int port;
    private int id;

    public ConsumerTCP() {
        try {
            port=8011;
            server=new ServerSocket(port);
            id=1;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void run() {
        while(true){
            try {
                System.out.println("Waiting for the client request");
                //creating socket and waiting for client connection
                Socket socket = server.accept();

                //gestione messaggio conesecuzione relativa funzione
                ConsumerLogic logic=new ConsumerLogic(socket,id);
                logic.run();

                //aumento msgid dopo ogni comunicazione
                id++;

            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

Il ConsumetTCP gestisce i messaggi in maniera FIFO, ed anchesso seguendo le direttive della libreria unibo.basiccomm23.

ConsumerLogic gestisce la comunicazione singola tra consumer ed un producer che fa inviato un messaggio
Una volta ricevuta la Socket verrà incapsulata all'interno di una Interaction per gestire la comunicazione ad un livello di astrazione maggiore.

ConsumerLogic estende la classe Thread di java per essere eseguita in parallelo a ConsumerTCP per non bloccare la ricezione di nuove connessioni

In seguito vengono mostrate le funzionalità principali del ConsumerLogic.

```
public class ConsumerLogic extends Thread{
    private Socket socket;
    private Interaction inter;
    private int id;

    private ConsumerLogic(Socket soc,int i) {
        try {
            this.socket=soc;
            this.inter=new TcpConnection(this.socket);
            this.id=i;
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void run() {
        IApplMessage msg;
        try {
            //estrazione messaggio
            msg = this.getInter().receive();
            System.out.println(msg);

            //risposta simil-ack
            if (msg.msgContent().equals("consume")) {
                this.getInter().forward(BasicMsgUtil.buildRequest(msg.msgF
            }
            else {
                this.getInter().forward(BasicMsgUtil.buildRequest(msg.msgF
            }

            //chiusura connessione
            this.getInter().close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

In seguito viene riportato il main come esempio si bootstrap del sistema:

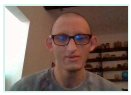
```
public class BootStrap{

    public static void main(String[] args){
        ProducerTCP prod=new ProducerTCP();
        prod.run();
        ConsumerTCP cons=new ConsumerTCP();
        cons.doJob();
    }
}
```

Testing

Deployment

Maintenance

By Arasi Stefano matr. 0001103134 email: stefano.arasi@studio.unibo.it,  GIT repo: https://github.com/ArasiStefanoISS/ISS (https://github.com/ArasiStefanoISS/ISS)