Distributore di eventi

Si consideri un sistema distribuito costituito produttori e consumatori. Ciascun produttore esegue ininterrottamente le seguenti azioni: genera un evento[1] e lo distribuisce ai consumatori. Ciascun consumatore esegue ininterrottamente le seguenti azioni: riceve un evento e lo consuma.

L'interazione tra produttori ed i consumatori avviene indirettamente attraverso un *distributore di eventi*. Non appena un produttore ha prodotto un evento, lo *pubblica* presso il distributore che lo distribuisce ai consumatori in modalità *push*[2]. In accordo a questa modalità, un consumatore notifica al distributore di eventi la propria presenza e questi gli invia gli eventi pubblicati. Un consumatore che ad un certo istante non è presente prima o poi si presenta.

Un produttore può pubblicare un evento in modo che esso sia distribuito a tutti i consumatori ovvero ai soli consumatori presenti. Un evento da consegnare a tutti i consumatori deve essere *bufferizzato* fintanto che tutti i consumatori lo hanno ricevuto.

Progettare e realizzare tutti i moduli del servizio di distribuzione degli eventi utilizzando il linguaggio Java, con particolare riferimento alla tecnologia RMI[3]. A tal proposito si suggerisce di considerare le seguenti interfacce[4].

```
public interface Produzione extends Remote {
   void publishAll(String n, Event e) throws RemoteException;
   void publishPresent(String n, Event e) throws RemoteException;
}
```

Il metodo `publishAll` permette ad un produttore di nome (testuale) `n` di pubblicare un evento `e` per tutti i consumatori. Il metodo `publishPresent` permette ad un produttore di nome (testuale) `n` di pubblicare un evento `e` per tutti i consumatori presenti.

```
public interface Consumo extends Remote {
   void present(String n, EventBox eb) throws RemoteException;
}
```

Il metodo `present` permette ad un consumatore di nome (testuale) `n` di notificare al distributore di eventi la propria presenza specificando la *event box* in cui depositare gli eventi.

```
public interface EventBox extends Remote {
   void deliver(Event e) throws RemoteException;
}
```

Il metodo `deliver` permette di depositare l'evento `e` nella event box.

```
public interface Event extends Serializable {
```

---

[1] La classe `Event`, definita di seguito per realizzare un evento, non ha nulla a che vedere con l'omonima `java.awt.Event`.

[2] Modalità *push* (letteralmente "spingere") significa che è il server ad inoltrare l'evento ai client. Al contrario, nella modalità *pull* ("tirare") è il client a richiedere l'evento.

[3] É richiesto il caricamento remoto delle classi, adoperando *Apache Tomcat©* come web server.

[4] Non é richiesta la realizzazione di interfacce grafiche utente.

```
    Object consume();
}
```

Il metodo `consume` non prende alcun argomento e ritorna un `Object` come valore di ritorno.

Ai fini della valutazione del progetto, il candidato dovrà prepararsi ad eseguire in sede di esame un test funzionale dell'intera applicazione, distribuita su tre host: uno per il produttore, uno per il consumatore ed uno per il distributore di eventi. Il caricamento remoto delle classi deve essere supportato.

Inoltre, il candidato dovrà produrre una documentazione di 10-15 pagine, costituita dalla descrizione degli aspetti funzionali (con qualche foto dell'interfaccia), architetturali (con qualche diagramma UML), ed implementativi (con tutto il codice Java opportunamente commentato e documentato con javadoc, comprensivo di eventuali script di compilazione/distribuzione).