

GOSSIP: servizio di scambio istantaneo/differito di messaggi in WAN

Autore: Orlando Leombruni
Matricola 475727

Indice

1	Panoramica	2
2	Architettura del sistema	2
2.1	Componente <i>registry</i>	2
2.2	Componente <i>proxy</i>	3
2.3	Componente <i>user agent</i>	3
3	Descrizione dei thread attivati	4
3.1	Thread attivati dal componente <i>registry</i>	4
3.2	Thread attivati dal componente <i>proxy</i>	5
3.3	Thread attivati dal componente <i>user agent</i>	5
4	Strutture dati utilizzate	5
5	Descrizione delle classi implementate	7
5.1	Struttura del codice sorgente	7
5.2	Package <i>common</i>	8
5.3	Package <i>proxy</i>	9
5.4	Package <i>registry</i>	9
5.5	Package <i>useragent</i>	10
6	Istruzioni per l'esecuzione	10
	Appendici	12
A	Cambiare la posizione della <i>registry</i>	12
B	Comandi dello <i>user agent</i>	12

1 Panoramica

GOSSIP è un servizio che permette agli utenti registrati di scambiarsi messaggi, sia istantanei (se sia mittente che destinatario sono online nello stesso momento) sia differiti. L'applicativo è costituito da tre parti: la *registry*, il componente “centrale” deputato ad offrire servizi di registrazione, gestione delle liste di amici e sincronizzazione delle strutture dati dell'intero sistema; il *proxy*, che riceve i messaggi diretti agli utenti offline; lo *user agent*, ossia il programma che gli utenti usano per accedere al servizio. Ognuno di questi tre componenti può essere dislocato su una diversa macchina di una WAN.

2 Architettura del sistema

Un sistema GOSSIP prevede un'unica *registry* centrale, un certo numero arbitrario di *proxy* e uno *user agent* per ogni utente che vuole usufruire del servizio. Nel caso in cui nessun proxy sia collegato, il sistema continua a funzionare ma è possibile solo l'utilizzo in modalità *messaging istantaneo*. Le funzionalità di scambio differito di messaggi sono ripristinate automaticamente non appena si collega un nuovo proxy.

2.1 Componente *registry*

Identificata dalle classi del package omonimo, la *registry* è l'unica delle tre componenti ad avere una locazione “fissa”: essa deve essere infatti avviata sulla macchina **router** del laboratorio virtuale, con indirizzo IP **172.241.0.1** e porte 49555 - 49556. Per eseguire la *registry* su macchina e/o porta diverse, è necessario modificare il codice sorgente (si veda l'appendice A). La *registry* implementa le interfacce necessarie per l'invocazione di metodi remoti da parte delle altre due componenti ed esporta gli oggetti RMI sulle porte 49555 (servizi per gli *user agent*) e 49556 (servizi per i *proxy*); rimane poi in attesa, stampando su **stdout** messaggi di stato relativi ai servizi erogati. Mediante il comando **shutdown**, il gestore della *registry* ne interrompe l'esecuzione in modo sicuro, provocando anche la disconnessione di eventuali proxy e utenti collegati. Il comando **poll** è invece utile per avere una panoramica degli utenti registrati e dei proxy collegati.

La *registry* gestisce anche la comunicazione *user agent* \longleftrightarrow *user agent* e *user agent* \longleftrightarrow *proxy*: essa provvede infatti a tenere aggiornati gli indirizzi nelle liste di uscita, sostituendo l'indirizzo di un utente che va offline con quello del proxy a cui esso è assegnato. L'assegnamento degli utenti ai proxy tiene conto del carico di questi ultimi; viene scelto sempre il proxy meno affollato (che gestisce meno utenti). All'atto del login, poi, la *registry* fornisce all'utente l'indirizzo del proxy a cui è stato assegnato, delegando allo *user agent* il compito di recuperare i messaggi ricevuti mentre si era offline.

Nel realizzare il componente *registry* si è scelto di utilizzare RMI per la comunicazione con i proxy, oltre che per la comunicazione con gli *user agent* come richiesto dalla consegna; tale scelta è motivata dall'osservazione che l'interazione che avviene tra registry e proxy è del tutto simile a quella che avviene tra registry e UA, ossia improntata su eventi brevi e con limitato scambio di

informazioni, e dunque adatta al paradigma di chiamata di procedura remota implementato da RMI.

2.2 Componente *proxy*

Il componente *proxy* può essere dislocato su una qualsiasi macchina del laboratorio virtuale; la gestione da parte della registry è dinamica e permette di avere un numero arbitrario di proxy sulla rete, anche zero.

Ogni proxy collegato al servizio GOSSIP è deputato alla ricezione di messaggi destinati a utenti offline e alla consegna degli stessi quando questi ultimi tornano online. Il componente *proxy* gestisce tre tipi di connessioni: una TCP, sulla porta passata come argomento dell'invocazione; una UDP, sulla porta immediatamente successiva, ed infine una RMI, sulla stessa porta di UDP. Tramite una socket TCP, il *proxy* attende la richiesta di un utente di fornirgli la lista dei messaggi ricevuti mentre era offline; rimanendo in ascolto sulla porta UDP, invece, esso riceve messaggi destinati agli utenti a suo carico. Si è scelto UDP, di comune accordo con il docente, in quanto questo protocollo ben si adatta alla comunicazione di un messaggio da user agent a proxy. Mediante RMI, infine, il *proxy* riceve dalla registry gli username degli utenti da gestire oppure interi database di messaggi provenienti da altri proxy; il sistema, infatti, è dotato di una funzione di preservazione dei messaggi, che alla disconnessione di un proxy provvede ad affidare il suo database ad un suo pari nel sistema. Anche il componente *proxy* è dotato del comando `poll`, per vedere il contenuto attuale del database, e del comando `shutdown`, per la terminazione sicura del proxy e l'avvio della routine di affidamento del database.

2.3 Componente *user agent*

Ogni utente che vuole usufruire del servizio GOSSIP deve eseguire il componente *user agent* sulla propria macchina. Uno *user agent* mantiene in locale le liste di entrata (*allow list*) e di uscita (*friend list*), oltre ad una cache di indirizzi relativi agli utenti nella propria lista di uscita; in questo modo si riduce il carico di lavoro che la registry deve compiere, dato che si elimina la necessità di contattare quest'ultima per operazioni banali come la presenza nelle liste o il reperimento dell'indirizzo del destinatario. La registry comunica mediante una callback RMI gli eventi di interesse per lo user agent; quest'ultimo comunica con i suoi pari (o con i proxy che ne fanno le veci) mediante UDP, e al login recupera i messaggi dai proxy con una connessione TCP. La porta utilizzata da UDP e RMI è passata come argomento, mentre la connessione TCP usa una porta effimera.

L'interfaccia utente dello *user agent* è un prompt dei comandi. La lista dei comandi disponibili e dei loro effetti è riportata nell'appendice B (e disponibile nel programma con il comando `help`).

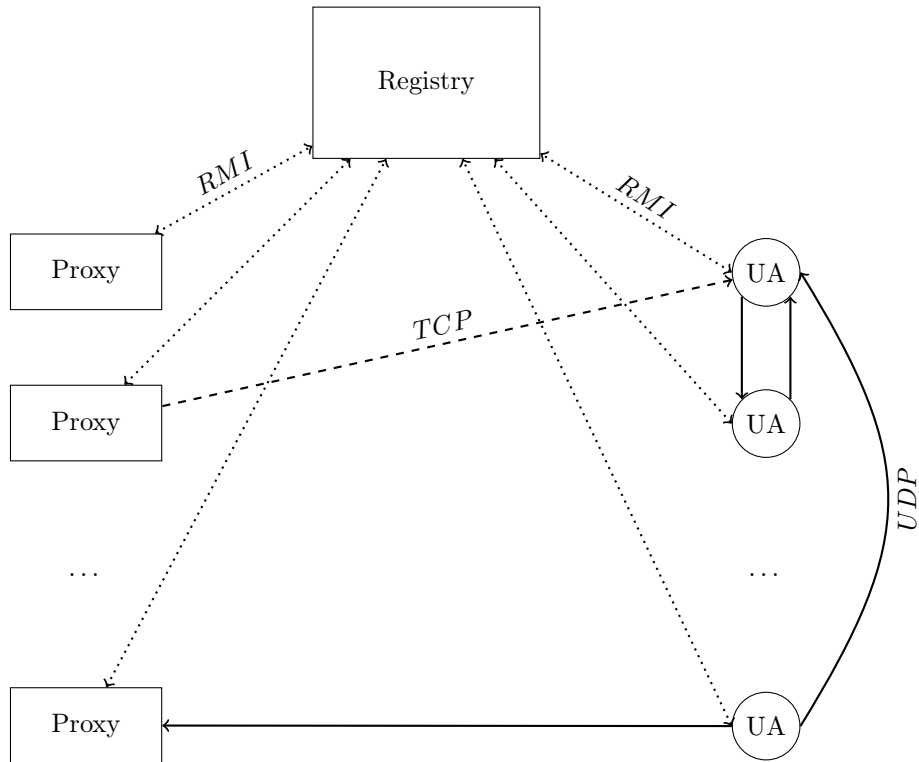


Figura 1: Esempio di sistema GOSSIP in esecuzione, con evidenziate le varie tipologie di collegamenti tra componenti. Le linee puntate sono collegamenti RMI, quelle tratteggiate TCP, quelle continue UDP.

3 Descrizione dei thread attivati

L'implementazione fornita di GOSSIP fa ampio uso di thread e, ove necessario, di tecniche di thread pooling per fornire le funzionalità richieste.

3.1 Thread attivati dal componente *registry*

Nella registry viene creato un thread per l'inizializzazione degli oggetti remoti e il loro *binding* al servizio RMI; il thread principale rimane attivo per ricevere comandi da parte del gestore. Il thread RMI viene inserito all'interno di un pool di dimensione fissa 1; ciò è stato fatto per poterne attendere la terminazione mediante il metodo `awaitTermination` della classe `ExecutorService`.

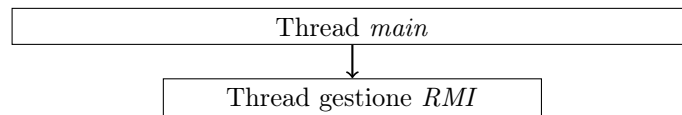


Figura 2: Schema dei thread attivati dal componente *registry*

3.2 Thread attivati dal componente *proxy*

Il thread principale del componente *proxy* crea un pool di 2 thread, uno deputato alla ricezione di datagrammi UDP (ricezione messaggi) ed uno dedicato ad accettare connessioni TCP (invio lista messaggi offline); a sua volta, quest'ultimo crea un thread per ogni connessione, inserendoli in un pool *cached*. Il thread principale passa poi ad attivare la callback RMI (senza attivare nuovi thread) e ad accettare comandi dal prompt.

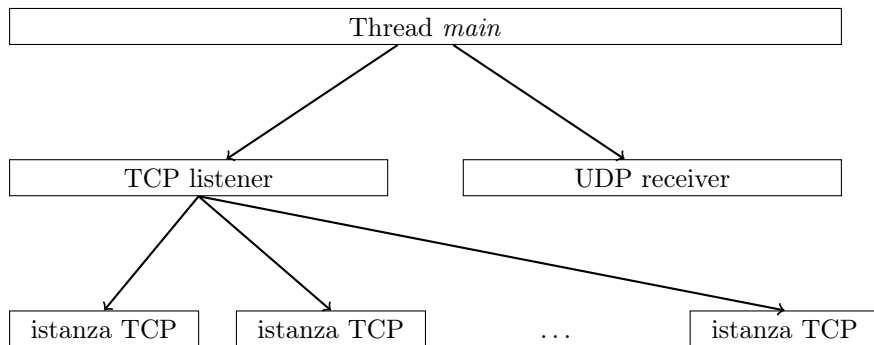


Figura 3: Schema dei thread attivati dal componente *proxy*

3.3 Thread attivati dal componente *user agent*

Il componente *user agent* attiva un thread ricevitore di datagrammi UDP, per i messaggi provenienti da altri utenti. Dopodiché, dopo aver impostato la connessione RMI con la registry ed aver registrato la propria callback, il thread principale passa a gestire i comandi dell'utente: tutte le successive operazioni, compreso l'avvio della connessione TCP con il proxy per il reperimento dei messaggi offline, vengono eseguite dal thread principale. Nello *user agent* non sono utilizzate tecniche di thread pooling.

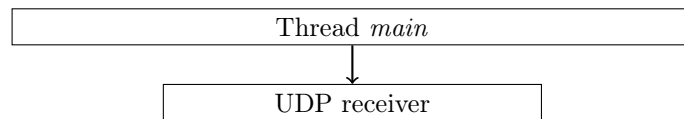


Figura 4: Schema dei thread attivati dal componente *user agent*

4 Strutture dati utilizzate

Nel realizzare il sistema GOSSIP particolare attenzione è stata riposta nell'utilizzo di strutture dati adeguate, per complessità e caratteristiche di *thread safety*, scelte tra quelle presenti nel JAVA COLLECTIONS FRAMEWORK oppure implementate per l'occasione. In particolare, nella *registry*:

- dato che non possono esistere due utenti con lo stesso nome, si è deciso di utilizzare una struttura dati di tipo insieme; gli utenti sono in-




seriti in un **TreeSet** di tipo *synchronized*. Ciò garantisce una bassa complessità ($O(\log n)$) nelle operazioni comuni di inserimento e verifica di appartenenza, oltre che il corretto funzionamento in presenza di concorrenza;

- gli utenti sono rappresentati come oggetti di una classe **User**, al cui interno sono mantenute tre liste (entrata, uscita e una terza lista composta da utenti che hanno l'utente in questione nella lista di uscita). Anche queste tre liste sono rappresentate da **TreeSet** *synchronized*;
- i proxy sono mantenuti in una struttura dati apposita, chiamata **ProxyPool**, mediante un **Vector**; anche in questo caso, si ha una complessità bassa per le operazioni frequenti (tempo costante ammortizzato per la **add**, tempo costante per la **get**) e un comportamento predicibile in caso di operazioni concorrenti;
 - come gli utenti, anche i proxy sono rappresentati come oggetti di una classe apposita (**Proxy**). La lista degli utenti assegnati ad un proxy è anch'essa un **Vector**.

L'unica struttura dati di rilievo nel componente *proxy* è un dizionario, implementato con una **HashMap** *synchronized*, che associa ad ogni utente gestito dal proxy una lista di messaggi in formato JSON (come definito nella libreria esterna JSONSIMPLE). L'implementazione a dizionario garantisce un reperimento in tempo costante della lista di messaggi di un utente, e permette di “fondere” due database di messaggi (operazione necessaria per la preservazione dei messaggi quando un proxy si disconnette). Anche la cache di indirizzi nel componente *user agent* è gestita con un dizionario, mentre le liste di entrata e di uscita sono **Vector**.

5 Descrizione delle classi implementate

5.1 Struttura del codice sorgente

-  src
 -  common
 - *  Address.java
 - *  Message.java
 - *  ProxyException.java
 - *  ProxyOps.java
 - *  RegistryOperations.java
 - *  RegistryProxyOps.java
 - *  UAOperations.java
 - *  User.java
 - *  UserException.java
 -  proxy
 - *  ProxyMain.java
 - *  ProxyRMI.java
 - *  ProxyTCPDispatcher.java
 - *  ProxyTCPInstance.java
 - *  ProxyUDPReceiver.java
 -  registry
 - *  Proxy.java
 - *  ProxyPool.java
 - *  ProxyServices.java
 - *  RegistryMain.java
 - *  RegistryRMI.java
 - *  UserServices.java
 -  useragent
 - *  UserAgent.java
 - *  UserListener.java
 - *  UserRMI.java

5.2 Package *common*

5.2.1 Address

Un oggetto di tipo **Address** incapsula una coppia (indirizzo IP, porta) per l'utilizzo da parte delle altre classi. L'indirizzo IP è mantenuto sia in formato stringa che in formato **InetAddress**.

5.2.2 Message

Rappresenta un singolo messaggio: mantiene al suo interno mittente, destinatario e messaggio vero e proprio in formato stringa, con la possibilità di convertirlo facilmente in un oggetto/stringa JSON. La classe espone anche un metodo statico per ottenere un oggetto di tipo **Message** a partire da un oggetto JSON.

5.2.3 ProxyException

È un'eccezione *checked* lanciata quando si incontra un problema che riguarda un proxy (ad esempio, quando nessun proxy è disposto ad accettare l'affidamento del database di un altro proxy che si sta disconnettendo).

5.2.4 ProxyOps

Interfaccia RMI delle operazioni fornite dal proxy alla registry (callback).

5.2.5 RegistryOperations

Interfaccia RMI delle operazioni fornite dalla registry agli user agent.

5.2.6 RegistryProxyOps

Interfaccia RMI delle operazioni fornite dalla registry ai proxy.

5.2.7 UAOperations

Interfaccia RMI delle operazioni fornite dallo user agent alla registry (callback).

5.2.8 User

È una struttura dati che incapsula tutte le informazioni relative ad un utente: nome, liste, indirizzo, porta, stato (online/offline) e callback. Implementa l'interfaccia **Comparable** ed è dunque utilizzabile come tipo di oggetto in un **SortedSet**.

5.2.9 UserException

È un'eccezione *checked* lanciata quando si incontra un problema che riguarda un utente (ad esempio, quando si prova a registrare un utente con lo stesso nome di uno già presente nel sistema).

5.3 Package *proxy*

5.3.1 ProxyMain

Classe che contiene il thread principale del componente *proxy*: esso si occupa di creare il database del proxy, di reperire l'indirizzo IP della macchina (ciclando fra le interfacce di rete fino a trovarne una non di loopback), di far partire i thread *TCP listener* e *UDP receiver* e di creare l'oggetto remoto che verrà esportato come callback RMI.

5.3.2 ProxyRMI

Implementa l'interfaccia **ProxyOps** e dunque i metodi invocabili via RMI dalla registry: assegnamento di un utente, del database di un altro proxy oppure chiusura del servizio (quando la registry si disconnette).

5.3.3 ProxyTCPDispatcher

In questa classe è definito il comportamento del thread TCP listener del proxy: esso rimane in attesa dell'apertura di una connessione TCP da parte di un utente, dopodiché crea un'istanza del thread di connessione vera e propria.

5.3.4 ProxyTCPInstance

Implementa il thread che si occupa di gestire la comunicazione con un utente che chiede i propri messaggi offline.

5.3.5 ProxyUDPReceiver

Classe che contiene il thread che si occupa di ricevere messaggi (codificati in JSON) via UDP da parte degli utenti.

5.4 Package *registry*

5.4.1 Proxy

Rappresentazione di un'istanza del componente *proxy* interna alla registry. Contiene indirizzo, ID, callback e utenti gestiti dal proxy.

5.4.2 ProxyPool

Classe ausiliaria per la gestione dei proxy da parte della registry. Mantiene la lista dei proxy attualmente collegati e si occupa di assegnare e revocare utenti dai proxy.

5.4.3 ProxyServices

Implementa l'interfaccia **RegistryProxyOps**, ossia i metodi invocabili via RMI dai proxy sulla registry: collegamento e disconnessione dal sistema.

5.4.4 RegistryMain

Thread principale del componente *registry*: si occupa di creare il thread che gestisce le comunicazioni RMI e di ricevere comandi dal prompt.

5.4.5 RegistryRMI

Classe che contiene il thread che si occupa della creazione ed esportazione degli oggetti remoti RMI, sia verso i proxy che verso gli user agent.

5.4.6 UserServices

Implementa l'interfaccia **RegistryOperations**, ossia i metodi invocabili via RMI dagli user agent sulla registry: registrazione, login, logout e gestione delle liste di entrata/uscita.

5.5 Package *useragent*

5.5.1 UserAgent

Thread principale del componente *user agent*. Fa partire il thread UDP receiver, crea ed esporta l'oggetto remoto per la callback RMI e cura le interazioni con l'utente.

5.5.2 UserListener

Classe che contiene il thread per la ricezione di messaggi via UDP da parte di altri utenti.

5.5.3 UserRMI

Implementa l'interfaccia **UAOperations**, ossia i metodi invocabili via RMI dalla registry: messaggi di stato ed aggiornamento delle liste di entrata/uscita.

6 Istruzioni per l'esecuzione

Gli eseguibili inclusi nella consegna sono tre: uno per la registry, **registry.jar**; uno per i proxy, **proxy.jar**; uno per gli utenti, **gossip.jar**.

Come già accennato precedentemente, la componente *registry* deve essere obbligatoriamente dislocata sulla macchina **router** del laboratorio virtuale; le altre due componenti, infatti, riferiscono ad essa mediante indirizzo IP e porta fissati. Una volta portato l'eseguibile sulla macchina **router**, con il comando

```
java -jar registry.jar
```

si avvia la registry.

Per quanto riguarda il componente *proxy*, esso è collocabile su qualsiasi macchina della rete: è stato testato con successo sia sulla macchina ospite sia su entrambe le macchine del laboratorio virtuale. È possibile attivare un'istanza del proxy con il comando

```
java -jar proxy.jar numeroporta
```

dove **numeroporta** indica naturalmente il numero di porta da utilizzare. Bisogna prestare attenzione al fatto che il proxy usa due porte diverse, **numeroporta** e **numeroporta + 1**; si deve dunque evitare di utilizzare porte immediatamente

successive a quelle già utilizzate, ad esempio quando si fanno partire più istanze del proxy sulla stessa macchina.

Infine, anche il componente *user agent* può essere collocato su una macchina qualsiasi: si invoca con il comando

```
java -jar gossip.jar numeroporta
```

e, al contrario del proxy, utilizza solo la porta indicata. Può capitare, specialmente se viene eseguito su sistemi Windows, che lo user agent usi un indirizzo IP relativo ad una rete diversa da quella del laboratorio virtuale, con conseguente impossibilità di collegarsi alla registry; è possibile risolvere questo problema scrivendo il comando **settings** all'interno del programma e fornendo manualmente l'indirizzo IP corretto.

Il progetto è stato realizzato con Eclipse, versioni Kepler e Luna, e testato su macchine ospiti con OS X 10.10 Yosemite e Windows 8.1.

Appendici

A Cambiare la posizione della *registry*

Come già spiegato, la posizione della registry è fissa: deve essere eseguita sulla macchina **router** del laboratorio virtuale, con indirizzo IP 172.241.0.1 e porte 49555 - 49556. Se si desidera cambiare la posizione della registry, bisogna effettuare le seguenti modifiche al codice sorgente:

- nel file `RegistryMain.java`, sostituire la riga 18:

```
RegistryRMI reg = new RegistryRMI(InetAddress.getLocalHost().getHostAddress(), pp);
```


con l'altro costruttore della classe `RegistryRMI`:

```
RegistryRMI reg = new RegistryRMI(porta, indirizzo, pp);
```

- nei file `ProxyMain.java`, righe 37-38, e `UserAgent.java`, righe 33-34, sostituire:

```
private static final String registryRMI =  
    "rmi://172.241.0.1:49555/GossipRegistryProxyService";
```


con

```
private static final String registryRMI =  
    "rmi://indirizzo:porta/GossipRegistryProxyService";
```

B Comandi dello *user agent*

L'interfaccia utente dello user agent è simile ad un prompt dei comandi. Se l'utente non ha ancora eseguito il login, il prompt sarà della forma

```
>$
```

Una volta loggato, il prompt diverrà

```
nomeutente>$
```

I comandi disponibili all'utente sono:

- `register <username>`
Registra l'utente *username* al servizio
- `login <username>`
Accede al servizio con il nome *username*
- `logout <username>`
Disconnette *username* dal servizio (possibile solo se *username* è lo stesso con cui si è effettuato il login)
- `send <username>`
Invia un messaggio all'utente *username*. Il messaggio va composto dopo aver inviato il comando

- **allow <username>**
 Aggiunge *username* alla propria lista di entrata (utenti abilitati ad inviare messaggi)
- **disallow <username>**
 Rimuove *username* dalla propria lista di entrata
- **friend <username>**
 Aggiunge *username* alla propria lista di uscita (utenti a cui è possibile inviare messaggi); ciò è possibile solo se si è presenti nella lista di entrata di *username*
- **unfriend <username>**
 Rimuove *username* dalla propria lista di uscita
- **status**
 Visualizza informazioni sulla connessione e sulle proprie liste di entrata/uscita
- **settings**
 Permette all'utente di modificare l'indirizzo IP con il quale tenta di collegarsi al servizio, e di riprovare la connessione alla registry
- **help**
 Visualizza un aiuto in linea
- **exit**
 Disconnette l'utente dal servizio, se è connesso, e chiude lo *user agent*