Relazione del progetto: GOSSIP.

Andrea Valenti – mat. 490124

- ## Architettura di sistema.

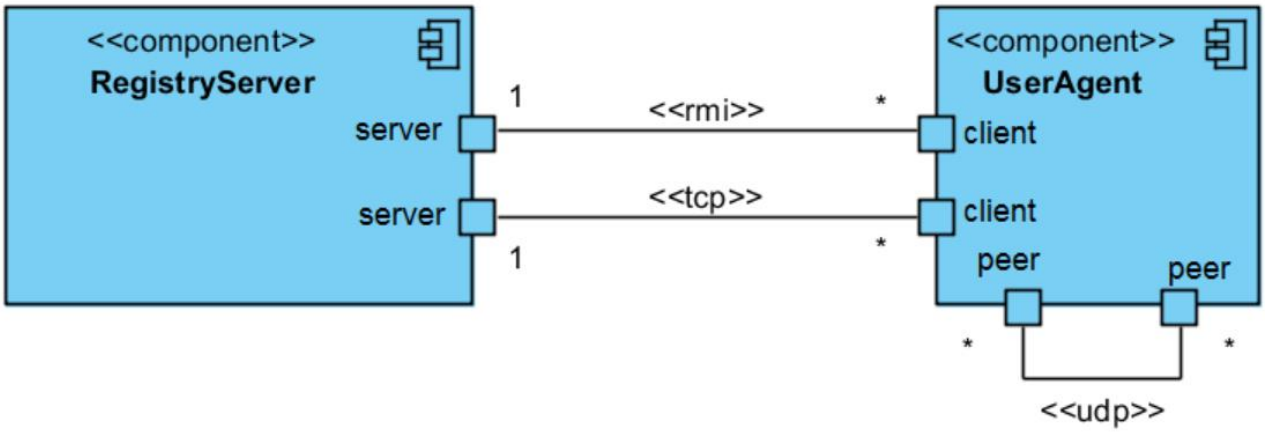

Figura 1 – Diagramma UML dell'architettura generale di sistema.

GOSSIP è composto da due unità principali: il *RegistryServer* e lo *UserAgent* (Fig. 1).

Il *RegistryServer* ha il compito di memorizzare lo stato complessivo del sistema, ascoltando segnalazioni dei cambiamenti di stato dei client mediante TCP. Quando un client segnala il cambiamento del proprio stato (da online a offline o viceversa), il *RegistryServer* si occupa di aggiornare il proprio stato interno in accordo con la segnalazione e ad invocare opportune callback RMI agli *UserAgent* interessati dal cambiamento. Inoltre, il *RegistryServer* gestisce un registry RMI, mediante il quale mette a disposizione metodi remoti che gli *UserAgent* possono invocare per comunicare col server.

Lo *UserAgent* implementa la chat vera e propria, gestendo l'interfaccia grafica e l'interazione con l'utente. Oltre alle comunicazioni TCP e RMI precedentemente menzionate, lo *UserAgent* implementa il protocollo peer to peer su UDP per lo scambio di messaggi tra utenti.

- Schemi interni dei componenti.

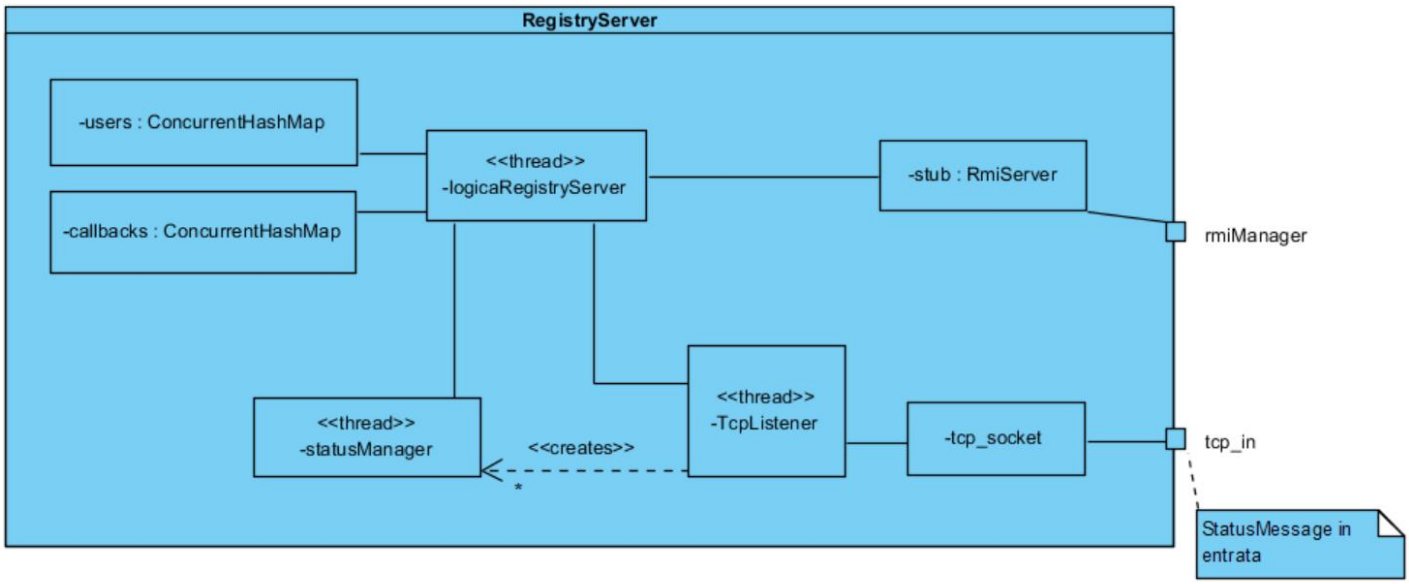

Figura 2 – Diagramma UML del componente RegistryServer.

I componenti principali del *RegistryServer* sono mostrati in Fig. 2: i campi *users* e *callbacks* hanno il compito di memorizzare informazioni riguardo agli utenti correntemente registrati a GOSSIP. È stato scelto di separare la memorizzazione delle callback da quella degli utenti per sfruttare meglio il parallelismo: spesso infatti i thread del *RegistryServer* devono accedere alle due strutture dati in modo indipendente. L'utilizzo di una tabella hash thread-safe come *ConcurrentHashMap* permette l'accesso concorrente di più thread senza l'onere di gestire esplicitamente la sincronizzazione degli accessi.

La connessione TCP è gestita dal thread *TcpListener*. Ha il compito principale di mettersi in attesa di connessioni dei client. Per ogni nuova connessione, TcpListener crea un nuovo thread di tipo *StatusManager*, passandogli i riferimenti a *users* e *callbacks*. *StatusManager* ha il compito di ascoltare ogni nuovo messaggio in arrivo sulla connessione, processarlo (aggiornando lo stato del *RegistryServer*) ed invocare le callback dei client interessati.

Dal momento che esiste uno *StatusManager* per ogni *UserAgent* connesso con il server, e che lo *StatusManager* rimane in esecuzione finché la connessione TCP non viene chiusa (i.e. finché lo *UserAgent* all'altra estremità della connessione non viene chiuso), si è scelto di non utilizzare tecniche di thread pooling. È stato infatti ritenuto che i vantaggi di queste tecniche non sarebbero stati sfruttati appieno in questo contesto (gli *StatusManager* in generale hanno un ciclo di vita molto lungo, e le operazioni da svolgere dopo la ricezione di un messaggio sono relativamente semplici e veloci da eseguire).

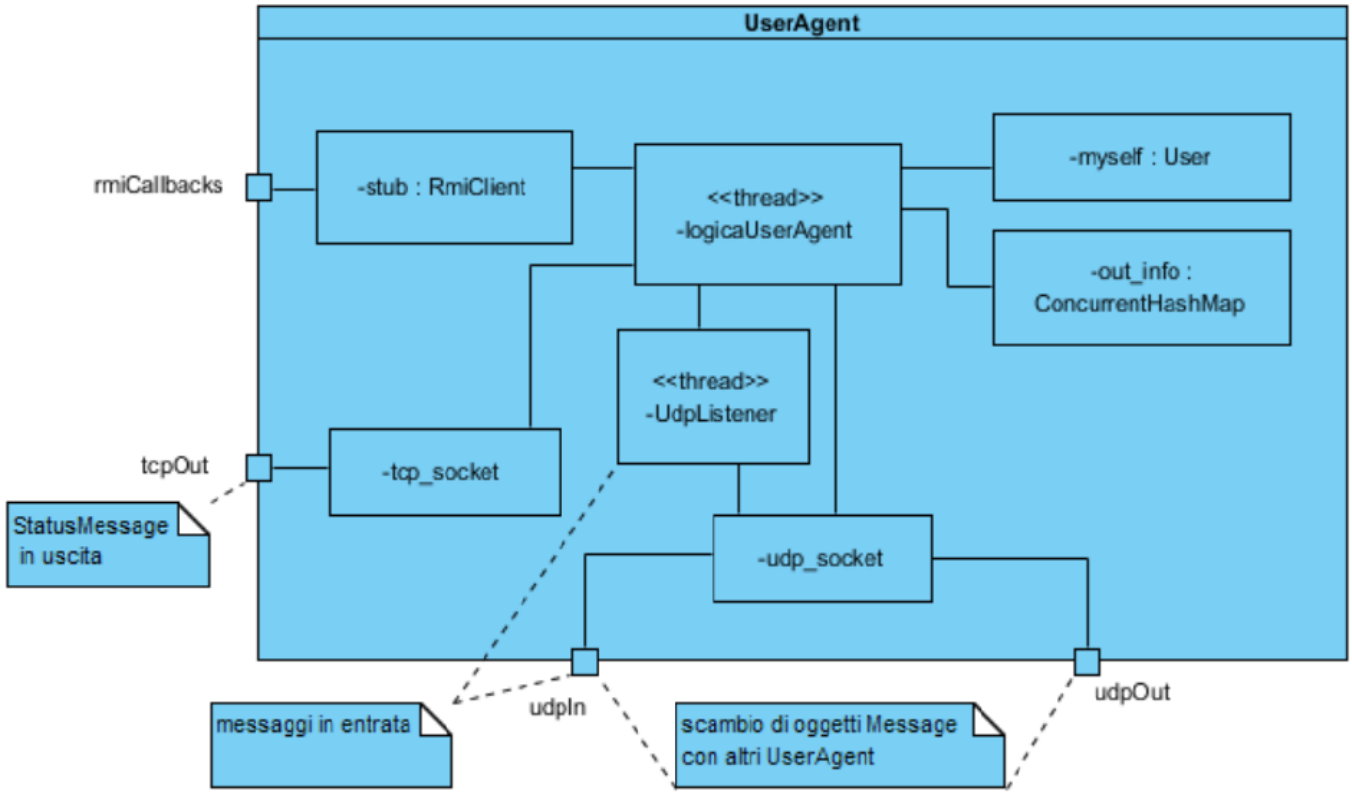

Figure 3- Diagramma UML del componente UserAgent

Fig. 3 mostra i component dello *UserAgent*: *myself* memorizza lo stato dell'utente che sta utilizzando lo *UserAgent*, *out_info* contiene informazioni relative agli altri utenti per l'invio dei pacchetti UDP, *udp_socket* e *tcp_socket* si occupano delle comunicazioni UDP e TCP. Per chiarezza di rappresentazione, nel diagramma UML sono stati omessi i componenti dell'interfaccia grafica ed i thread invocati automaticamente dalla JVM per la gestione degli eventi grafici.

Mentre l'invio di messaggi UDP in uscita è gestito dal gestore di eventi grafici, l'ascolto di messaggi in ingresso è affidato al thread *UdpListener*, che ha il compito di ricevere ogni messaggio in arrivo sul socket e visualizzarlo. Dal momento che la text area è l'unico oggetto condiviso tra questi thread, è bastato aggiungere dei blocchi synchronized attorno al codice di gestione della text area per garantire l'accesso in mutua esclusione.

• Descrizione delle classi.

Per una descrizione dettagliata delle classi e dei metodi, si rimanda alla relativa documentazione Javadoc, nella stessa cartella di questa relazione.

• Come eseguire i file eseguibili.

Gli eseguibili sono forniti sotto forma di file .jar.

Per eseguire GOSSIP in modalità server:

```
java –jar GOSSIP.jar server
```

Per eseguire GOSSIP in modalità client:

```
java –jar GOSSIP.jar client <ip server>
```

dove al posto di <ip server> va inserito l'indirizzo ip dell'host in cui si trova il server di GOSSIP.

Prima di attivare un client deve esserci almeno un server già in esecuzione. Due server non possono essere eseguiti sullo stesso host.