UNIVERSITÀ DEGLI STUDI DI MILANO,
DIPARTIMENTO DI INFORMATICA
LAUREA TRIENNALE IN COMUNICAZIONE DIGITALE
CORSO DI RETI DI CALCOLATORI
ANNO ACCADEMICO 2011/2012

Progetto Gennaio 2013: Social Chat Internazionale

Dicembre 2012

1 DESCRIZIONE DEL PROBLEMA

Lo scopo del progetto è creare un'applicazione che permetta la gestione di una social chat internazionale (SCI) impiegando il modello client-server. L'idea di social chat internazionale è mostrata in figura 1.1. Come si può osservare, nel modello di SCI esistono due livelli. Il primo livello è dato da una rete sociale tra le persone iscritte al servizio (cerchi e collegamenti arancioni), mentre un secondo livello è definito dalle chat-rooms a cui ogni utente può liberamente iscriversi (rettangoli e link blu).

I servizi che un server di tipo SCI mette a disposizione sono essenzialmente la possibilità di chattare SOLO con gli amici della mia rete sociale e la disponibilità di chattare con appartenenti alle chat-rooms a cui si è iscritti. L'utente A quindi, sfruttando la chat tra amici potrà chattare solo con B e C, mentre l'utente D può chattare solo con B sfruttando la chat-amici e chattare contemporaneamente con E ed F usando la chat-room. La chat-amici crea un canale di comunicazione tra solo due elementi della social network, mentre una chat-room crea un canale di tipo multicast tra gli utenti iscritti. Ciò significa che un messaggio inviato da A alla chat-room 'Chat Room 1' viene diffuso contemporaneamente a tutti gli iscritti alla chat-room.

L'internazionalità di SCI è data dal fatto che ogni utente specifica la lingua con cui comunicare nella chat. Il carattere innovativo di un SCI è dato dal servizio di traduzione immediata

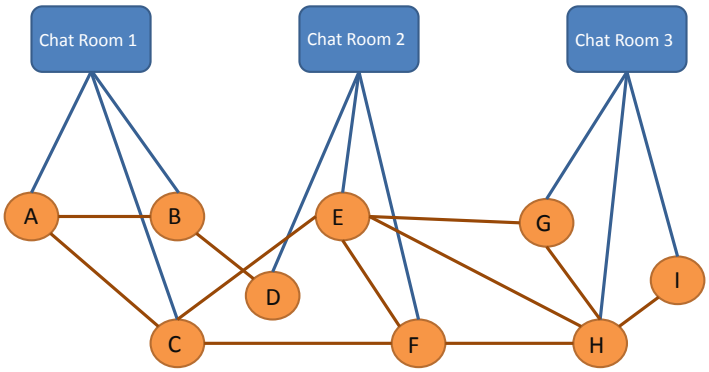

Figura 1.1: Esempio di social chat internazionale

dei messaggi tra utenti che non parlano la stessa lingua. Per esempio se A parla in italiano e B comunica in lituano, A scriverà un messaggio in italiano a B che riceverà il messaggio opportunamente tradotto in lituano. A sua volta B scriverà solo in lituano e A riceverà i messaggi di B tradotti. Attraverso la traduzione istantanea ad A sembrerà di chattare con madrelingua italiani, ma in realtà sta comunicando con utenti di lingue diverse.

2 Specifiche tecniche

2.1 Server

Il cuore dell'applicazione SCI è il server. Il server gestisce la rete sociale tra gli iscritti al servizio e tutte le funzionalità inerenti la chat. Dal momento che il server deve gestire la comunicazione istantanea tra più client, esso deve essere **obbligatoriamente** multithread, cioè deve poter gestire più richieste di servizio contemporaneamente.

Per un'opportuna interazione tra client e server, il secondo riconosce alcuni determinati comandi che seguono una sintassi ben specificata.

2.1.1 Operazioni Social network

Le operazioni riguardanti la gestione della social network sono la registrazione dell'utente, il suo login, la creazione di una relazione di amicizia, e la ricerca di amici. Le risposte inviate dal server seguono il formato JSON.

- `register:<nome>-<cognome>-<lingua>`. Il client richiede che l'utente con nome e cognome specificati venga registrato nella social network. Il server aggiunge un nodo

con nome e cognome specificati. Se esiste già un utente con quel nome e cognome, il server comunica che non è possibile inserire l'utente. Il comando non è case-sensitive, quindi Mario Rossi e mario rossi sono la stessa persona. Il terzo parametro `lingua` può assumere i valori ISO 639 reperibili, per esempio, nella seconda colonna della tabella all'indirizzo `http://www.loc.gov/standards/iso639-2/php/code_list.php` Il JSON di risposta è nel formato:

```
{"reply" : "<Messaggio>"}
```

dove Messaggio è un messaggio che comunica l'avvenuta o meno registrazione a SCI. Di seguito un esempio di possibile interazione:

```
register:Mario-Rossi-ca
Registrazione avvenuta con successo.
```

In questo caso viene creato un utente Mario Rossi di lingua catalano.

- `login:<nome>-<cognome>` L'utente richiede una login al servizio. Se l'utente è registrato, viene loggato e il server comunica l'avvenuta identificazione, altrimenti comunica che l'utente non è registrato alla social network. Il JSON di risposta è nel formato:

  ```
  {"reply" : "<Messaggio>"}
  ```

  Una volta loggato l'utente può eseguire uno dei seguenti comandi:

- `search:<nome>-<cognome>` L'utente richiede di ricercare un amico. Se esiste un utente con nome e cognome specificati, il server comunica che l'utente cercato è presente nella network. La risposta segue il formato:

  ```
  {"reply" : "<Messaggio>"}
  ```

  Per esempio, supponendo Pino Lino registrato e Ugo Sugo assente, si otterranno i seguenti risultati:

  ```
  search:Pino-lino
  Pino Lino trovato
  search:ugo-sugo
  Ugo Sugo non trovato
  ```

- `friend:<nome>-<cognome>` Mediante questa comando l'utente richiede che venga creata una relazione di amicizia con l'utente specificato. A differenza di altri social network non si deve aspettare che l'utente accetti l'amicizia. Il server, quindi, costruirà un link nel grafo tra l'utente che esegue il comando e quello specificato come argomento del comando. Nel caso l'utente specificato non esista, il server non crea alcun arco e comunica l'errore all'utente. Anche per questo comando, la risposta del server è un JSON:

```
{"reply" : "<Messaggio>"}
```

- `listfriend` Questo comando restituisce una lista degli amici dell'utente. Il JSON restituito dal server sarà del tipo:

  ```
  {"friendlist" : ["Nome Cognome Lingua",...,"Nome Cognome Lingua"]}
  ```

  dove lingua segue il formato ISO 639.

N.B. Si assuma che in Nome e Cognome **non compare mai** il simbolo '-'.
N.B. 2: Le parole tra < e > rappresentano dei tipi non il contenuto effettivo, sono dei 'segnaposto'.

2.1.2 Operazioni Chat-Rooms

Il server riconosce alcuni comandi per la creazione e la gestione di una chat-room. Una chat-room può essere creata solo da un utente ed è definita univocamente da una serie di caratteri alfanumerici. Di conseguenza non possono esistere due chat-rooms con lo stesso identificativo. I comandi riconosciuti sono:

- `create:<chatId>-<argomento>`. Questo comando crea una chat-room con identificativo chatId. `argomento` è opzionale. Nel caso una chat-room con chatId specificato esista già, il server comunica l'errore all'utente, nel formato:

  ```
  {"reply" : "<Messaggio>"}
  ```

- `addme:<chatId>`. Il comando addme aggiunge l'utente alla chat-room specificata dal chatId. Nel caso l'utente sia già iscritto alla chat-room il server comunica la già avvenuta iscrizione. Il formato del messaggio è:

  ```
  {"reply" : "<Messaggio>"}
  ```

- `chatlist`. Questo comando restituisce tutte le chat-rooms disponibili, comprese quelle a cui l'utente è iscritto. La risposta, sempre in formato JSON contiene una lista di tutte le chat-rooms. Per ogni entry vengono riportati la chatId, l'argomento, se definito, e un flag di tipo 'Y' o 'N' che indica se l'utente è iscritto o meno alla chat-room. In questo caso il formato è più complesso:

  ```
  {"chatlist": [
  { "chatId": "Id1", "arg":"<argomento>" , "flag":"N" },
  { "chatId": "Id2", "arg":"<argomento>" , "flag":"Y" },
  { "chatId": "Id3", "arg":"<argomento>" , "flag":"N" },
  { "chatId": "Id4", "arg":"<argomento>" , "flag":"Y" }]
  }
  ```

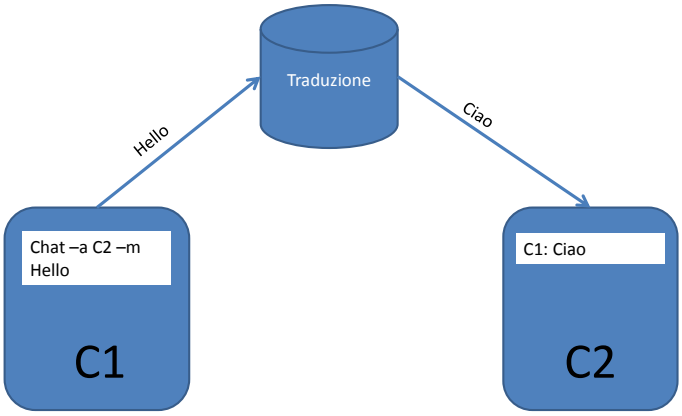

Figura 2.1: Esempio di funzionamento della chat-amici

2.1.3 CHAT-AMICI E CHAT-ROOMS

Le funzionalità sono attive solo tra utenti che sono attualmente loggati all'applicazione SCI e quindi online, non è previsto alcun invio differito dei messaggi. Il recapito dei messaggi avviene in modo **istantaneo** e **asincrono**.Ciò significa che il messaggio ricevuto viene visualizzato subito e non sono necessarie particolari interazioni dell'utente con l'applicazione per consentire la visualizzazione del messaggio. Per esempio l'utente non deve digitare alcun comando del tipo 'scarica messaggi' oppure eseguire un comando riconosciuto e ricevere come risposta anche un messaggio che gli è stato inviato.

Il servizio di chat avviene sempre mediante il server SCI, ciò significa che il server agisce da intermediario tra i client interessati a chattare. Un esempio è mostrato in figura 2.1.

Il server riconosce i seguenti comandi:

- `chatfriend:<nome>-<cognome>-<messaggio>`. Mediante chatfriend l'utente che invoca il comando chiede che il messaggio specificato venga inviato all'amico specificato da Nome e Cognome. Posso emergere errori se l'utente non è mio amico, se l'utente non esiste oppure se l'utente non è al momento online. In tutti i casi (errore o esito positivo) il server risponde al mittente con un messaggio del tipo

  ```
  {"reply" : "<Messaggio>"}
  ```

  In caso di successo messaggio contiene il testo inviato, mentre in caso di errore contiene un messaggio comunicante il tipo di errore occorso. Il destinatario del messaggio riceverà dal server un messaggio di tipo JSON nella forma:

  ```
  {"sender" : "<Nome Cognome>",
  ```

```
"text" : "<Messaggio>"
}
```

dove messaggio contiene la traduzione del messaggio nella lingua del destinatario.

- `chatroom:<idChat>-<messaggio>`. Mediante chatroom l'utente che invoca il comando chiede che il messaggio specificato venga inviato a tutti gli utenti (amici e non) iscritti alla chatroom con identificativo idChat. Possono emergere errori se la chatroom non esiste o se nessun utente iscritto alla chat-room è online (ad esclusione dell'utente che invoca il comando). In tutti i casi (errore o esito positivo) il server risponde al mittente con un messaggio del tipo

  ```
  {"reply" : "<Messaggio>", "chatId" : "<chatId>"}
  ```

  In caso di successo messaggio contiene il testo inviato, mentre in caso di errore contiene un messaggio comunicante il tipo di errore occorso. Gli iscritti alla chat-room riceveranno dal server un messaggio del tipo:

  ```
  {"sender" : "<Nome Cognome>",
  "chat-room" : "<chatId>" ,
  "text" : "<Messaggio>"
  }
  ```

  che indica che l'utente Nome e Cognome ha inviato un messaggio a tutta la chat-room chatId con il contenuto opportunamente tradotto secondo la lingua del destinatario

2.1.4 TRADUZIONE

Per quanto riguarda la traduzione dei messaggi, il server si può avvalere di alcune REST API messe a disposizione da alcuni providers. In particolare viene consigliato il servizio MyMemoryTranslated.net che offre un semplice servizio di traduzione free. La documentazione è reperibile all'indirizzo `http://mymemory.translated.net/doc/spec.php`.

2.2 CLIENT

Il client deve interagire con il server e con l'utente, di conseguenza oltre a inviare i comandi con la giusta sintassi, deve fornire una user interface testuale usabile. Non vengono posti particolari vincoli sulla struttura del client. Lo studente dovrà analizzare le criticità imposte dalla doppia interazione server/utente e valutare quali approcci intraprendere. L'unico vincolo imposto è dato dal formato dei messaggi che il server riconosce. Per testare la validità dei messaggi viene fornita una classe TestMessaggi.class (server di test) che risponde con una stringa in formato JSON del tipo:

```
{"reply" : "<Messaggio>"}
```

La classe riceve come argomento opzionale una parametro che specifica la porta su cui il server si mette in ascolto. La porta di default è la 7000.

3 CONSEGNA

Il progetto è **individuale** e verranno presi provvedimenti nel caso ci sia il sospetto di una eventuale copiatura. Il linguaggio da utilizzare per lo sviluppo è Java. Non ci sono vincoli sulle librerie da utilizzare per lo svolgimento, tuttavia é consigliato l'uso della libreria Graphstream per quanto riguarda la parte inerente al grafo. Il codice del progetto e i relativi jar delle librerie utilizzate devono essere inviati al docente in una directory compressa (zip o tar.gz) denominata 'progettoGennaio'. La mail deve avere come oggetto 'ProgettoGennaioReti' e contenere nel corpo nome, cognome e numero di matricola dello studente. Il materiale deve essere consegnato entro le 23:59:59 del 5 gennaio 2013. Valgono tassativamente le seguenti regole:

- Il codice che non compila non verrà considerato
- I progetti consegnati dopo la data di consegna non saranno ritenuti validi

Per essere considerato sufficiente e quindi ammissibile per la discussione, il progetto deve implementare correttamente **tutte** le funzioni precedentemente illustrate. Principali criteri di valutazione del codice:

- Gestione di possibili errori nell'input da parte dell'utente
- Modularità del codice e rispetto del paradigma ad oggetti
- Gestione oculata delle eccezioni
- Uso appropriato dei commenti
- Gestione e risoluzione di eventuali problematiche legate alla concorrenza sia lato client sia lato server

Per chiarimenti, dubbi potete contattare il docente all'indirizzo reti@dsi.unimi.it, specificando come oggetto obbligatorio 'ProgettoRetiGennaio'. Eventuali modifiche al testo del progetto saranno comunicate sulla pagina web del corso.