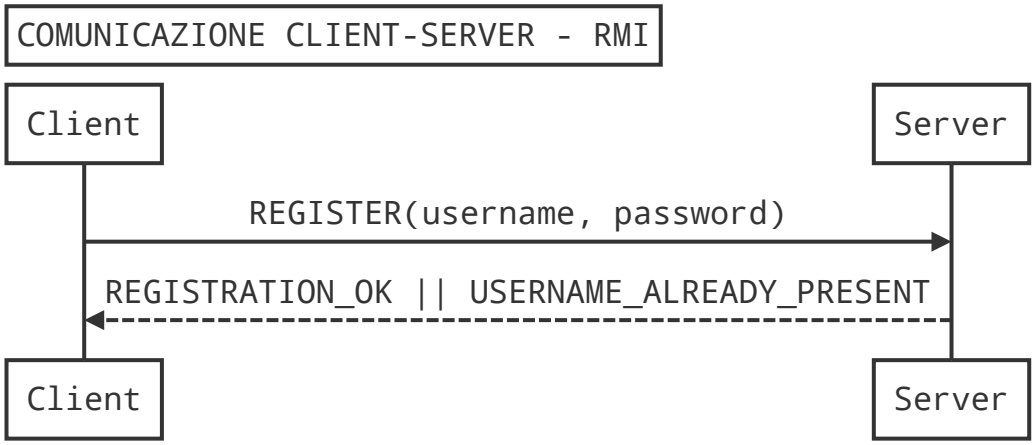
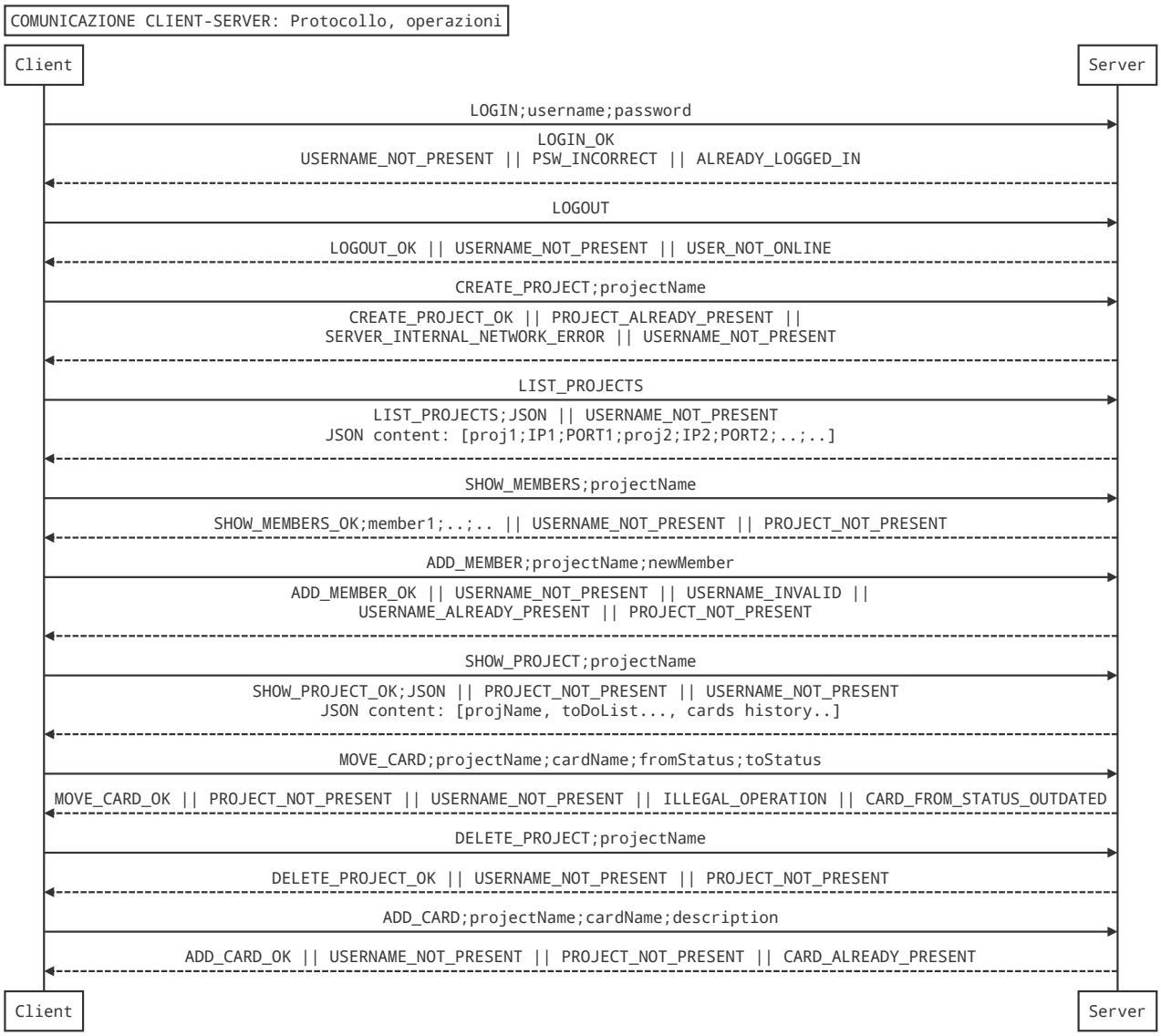
PROGETTO WORTH

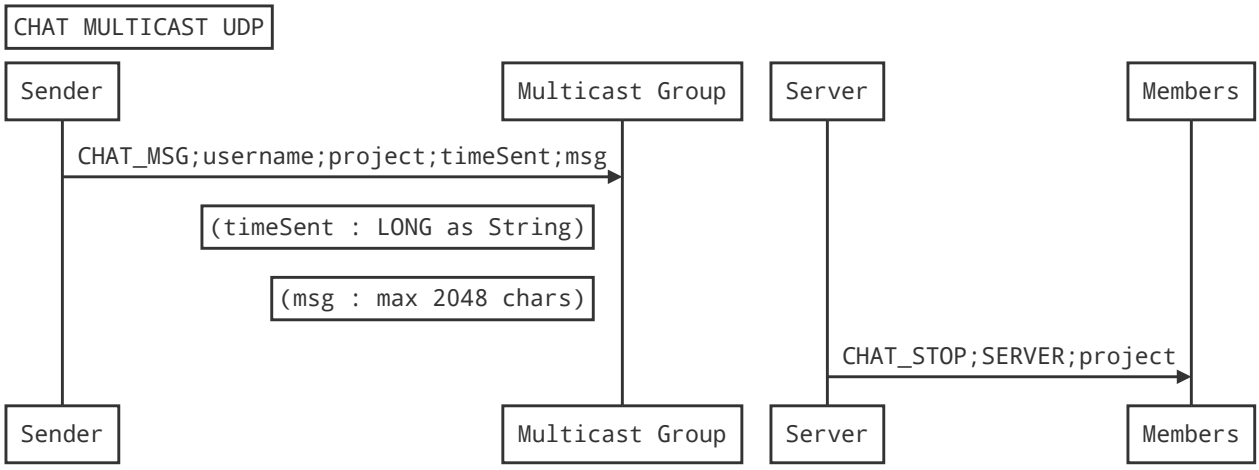
Indice

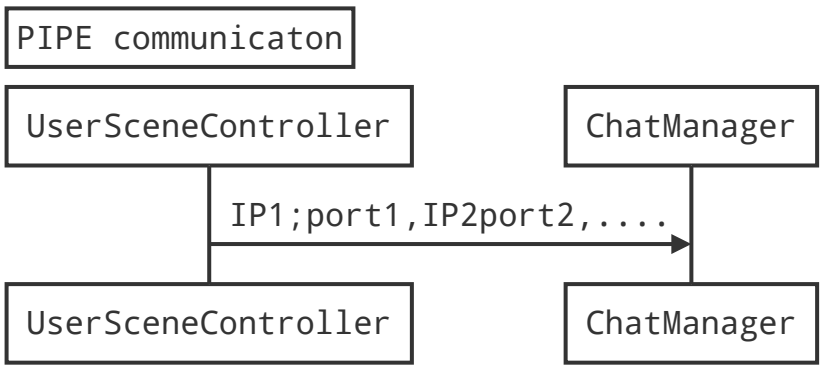
1. Struttura classi

uso `ConcurrentHashMap` per le registrazioni in quanto tutte le operazioni sono garantite essere *thread-safe*.
Non vi sono sovrapposizioni fra inserimenti e rimozioni in quanto un utente non può registrarsi ed effettuare il login contemporaneamente, pertanto reputo tale scelta la più efficiente garantendo l'accesso concorrente alla struttura dati.
Di conseguenza non ho bisogno di gestire la concorrenza server-side riguardo l'RMI.




COMUNICAZIONE CLIENT-SERVER: Protocollo, operazioni
Client
Server
LOGIN;username;password
LOGIN_OK
USERNAME_NOT_PRESENT || PSW_INCORRECT || ALREADY_LOGGED_IN
LOGOUT
LOGOUT_OK || USERNAME_NOT_PRESENT || USER_NOT_ONLINE
CREATE_PROJECT;projectName
CREATE_PROJECT_OK || PROJECT_ALREADY_PRESENT ||
SERVER_INTERNAL_NETWORK_ERROR || USERNAME_NOT_PRESENT
LIST_PROJECTS
LIST_PROJECTS;JSON || USERNAME_NOT_PRESENT
JSON content: [proj1;IP1;PORT1;proj2;IP2;PORT2;..;..]
SHOW_MEMBERS;projectName
SHOW_MEMBERS_OK;member1;..;.. || USERNAME_NOT_PRESENT || PROJECT_NOT_PRESENT
ADD_MEMBER;projectName;newMember
ADD_MEMBER_OK || USERNAME_NOT_PRESENT || USERNAME_INVALID ||
USERNAME_ALREADY_PRESENT || PROJECT_NOT_PRESENT
SHOW_PROJECT;projectName
SHOW_PROJECT_OK;JSON || PROJECT_NOT_PRESENT || USERNAME_NOT_PRESENT
JSON content: [projName, toDoList..., cards history..]
MOVE_CARD;projectName;cardName;fromStatus;toStatus
MOVE_CARD_OK || PROJECT_NOT_PRESENT || USERNAME_NOT_PRESENT || ILLEGAL_OPERATION || CARD_FROM_STATUS_OUTDATED
DELETE_PROJECT;projectName
DELETE_PROJECT_OK || USERNAME_NOT_PRESENT || PROJECT_NOT_PRESENT
ADD_CARD;projectName;cardName;description
ADD_CARD_OK || USERNAME_NOT_PRESENT || PROJECT_NOT_PRESENT || CARD_ALREADY_PRESENT
Client
Server


CHAT MULTICAST UDP
Sender
Multicast Group
Server
Members
CHAT_MSG;username;project;timeSent;msg
(timeSent : LONG as String)
(msg : max 2048 chars)
CHAT_STOP;SERVER;project
Sender
Multicast Group
Server
Members


PIPE communicaton
UserSceneController
ChatManager
IP1;port1,IP2port2,....
UserSceneController
ChatManager

2. Ipotesi

- un utente può loggarsi su una sola connessione (no login multipli)
- stateful
- limite superiore numero progetti dato dal numero di IP multicast locali
- scalabilità multihread + multiplexed

3. Scelte

quando la finestra della GUI del client viene chiusa viene una mandata una richiesta di **EXIT** che informa il server di chiudere quella connessione TCP in quanto non verranno effettuati altri login.

Se il client aveva effettuato l'accesso in aggiunta viene inviato una richiesta di **LOGOUT**.

Comunicazione TCP stateful, lo stato di utente loggato è registrato durante la comunicazione, infatti non si ha bisogno di passare nuovamente il proprio username.

``

? é importante che threadChatManager e ClientWT usino dbHandler che non è sincronizzato

- Java DB Developer's Guide

Multi-Connection
From an application, using
multiple connections to a
Derby database and issuing
requests against those
connections on multiple
threads.

«If thread
A does database work that is not transactionally related to thread B, assign them to
different Connections»

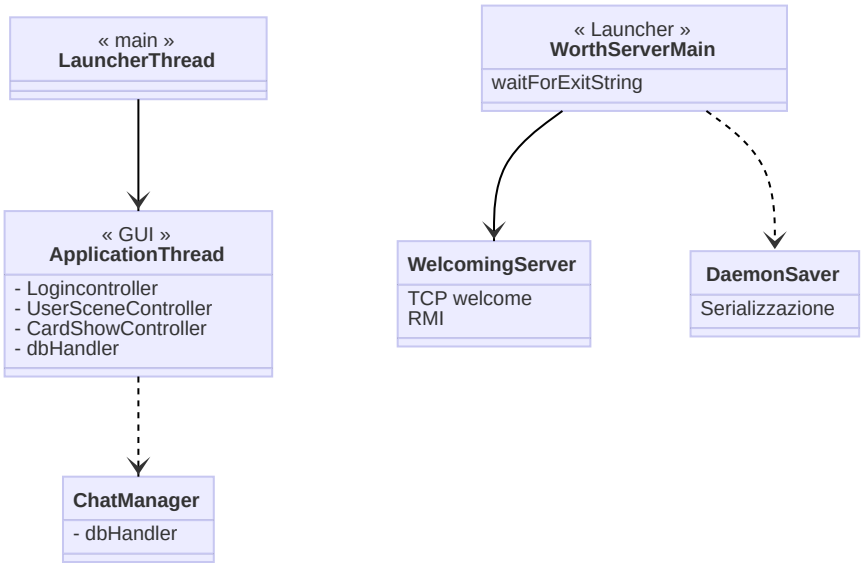
Per l'aggiornamento automatico della Chat:

- nel `ClientLogic`, al momento dell'istanziamento di un'istanza di `DbHandler`, setto la variabile statica `currentChatMsgList` passando una `ObservableList <ChatMsgObservable>` .

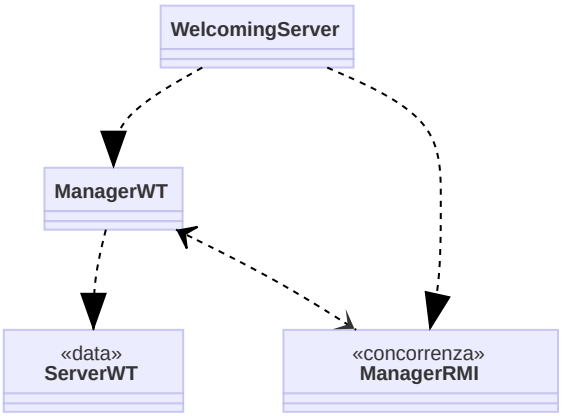
CONCORRENZA

Server: utenti con concurrentHashMap e ordine delle operazioni, prima users poi usersOnline

3.1. Threads


« main »
LauncherThread
« GUI »
ApplicationThread
- Logincontroller
- UserSceneController
- CardShowController
- dbHandler
ChatManager
- dbHandler
« Launcher »
WorthServerMain
waitForExitString
WelcomingServer
TCP welcome
RMI
DaemonSaver
Serializzazione

3.2. Oggetti Server



4. Comandi

- mvn -Pserver, aspettare che parta
- mvn -Pclient

settare JAVA_HOME e consiglio JAVA_TOOL_OPTIONS='file.encoding="UTF-8"'

La guida ufficiale per scaricare ed installare Maven è disponibile qui.