Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia Informática
Sistemas Distribuídos e Criptografia
4ºano
11/12/2015

Paradigmas de Sistemas Distribuídos

TP1 - Chat Server

(Quasar & ZeroMQ)

Trabalho realizado por:
João Pedro Costa Ferreira Dias **pg**30466
Paulo Ricardo Cunha Correia Araújo **a**58925
Simão Pedro Carmo Pinto Dias **a**61006

Índice

Arquitetura

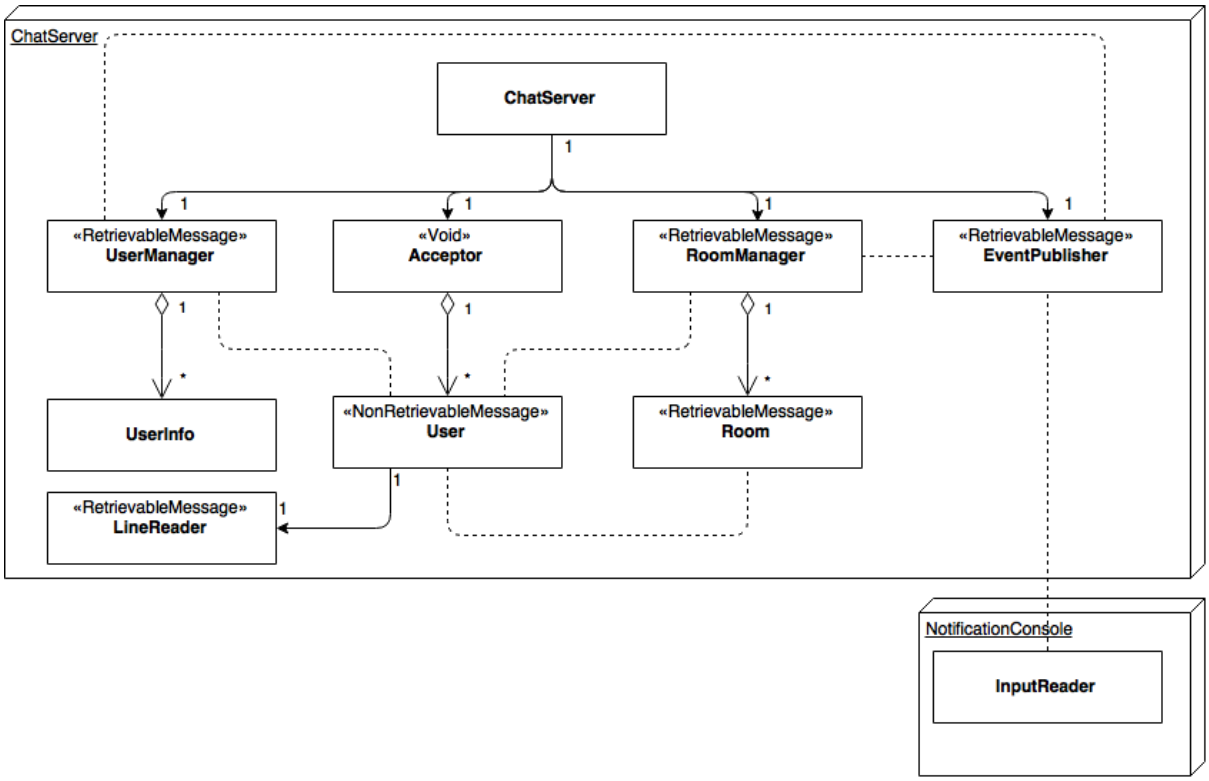

Figura 1: Arquitetura da aplicação.

O projeto é composto por duas aplicações: um servidor com o qual os utilizadores e administradores interagem através de uma ferramenta de comunicação baseada em texto *(telnet/netcat)* com o serviço de "chat rooms", e um cliente referente à consola de notificações, onde é possível subscrever a eventos relevantes.

Chat Server

O utilizador pode definir várias portas correspondentes a diferentes servidores de mensagens no inicio da aplicação. Cada uma destas portas corresponderá a uma instância de **ChatServer,** onde são definidas as classes **UserManager**, **Acceptor**, **RoomManager** e **EventPublisher**, assim como as instâncias de **Supervisor**. A classe **Acceptor** contêm a lógica responsável pela recepção de conexões de utilizadores, instanciando actores da classe **User** responsáveis pela sessão e atribuindo a cada um uma instância de **FiberSocketChannel**. É também no **Acceptor** que estas instâncias são adicionadas ao **UserSupervisor** (definido mais à frente no relatório).

User

A classe **User** contém toda a lógica associada à sessão, possuindo informação do utilizador, das salas de chat às quais está associado e dos managers de salas e de utilizadores. A classe **User** possui um limite de mensagens na **MailBox** de 100 mensagens, efetuando DROP das que chegarem quando é atingido o limite. É importante mencionar que um utilizador pode estar em várias salas simultaneamente (i.e. receber mensagens de diferentes salas), mas pode apenas escrever para uma de estas (como se estivesse com várias conversas abertas de Facebook, mas apenas com uma selecionada). Cada instância de **User** possui associada uma instância de actor **LineReader**, responsável por processar as mensagens enviadas pelo utilizador através do socket, e redirecionar uma mensagem processada para o **User**, onde está presente toda a lógica de processamento de comandos. As instâncias de **User** possuem as funcionalidades de utilizador normal e administrador, sendo que no momento de login o **UserManager** indica ao **User** que tipo de privilégios este possui.

User Manager

O actor **UserManager** possui a informação referente apenas aos utilizadores (username, password, estado de autenticação, privilégios de administrador, etc), abstraindo-os da informação das salas, que é gerida por um actor **RoomManager**. O **UserManager** possui informação sobre todos os utilizadores autenticados e registados, e troca mensagens com o **User** para executar funcionalidades como dar registo de um novo utilizador ou enviar uma mensagem privada a uma lista de utilizadores. O **UserManager** é assim responsável pelo envio de mensagens privadas (ao invés dos **Rooms**, responsáveis pelo envio de mensagens de grupo). Dado que este utilizador recebe em média mais mensagens que um dado **User,** o tamanho da sua **MailBox** é de 1000 mensagens.

Room Manager

O actor **RoomManager** possui informação referente às salas públicas e privadas, e aos utilizadores associados às salas privadas (apenas o **username**), pelo que o nível de abstração à informação dos utilizadores continua consistente. Este actor troca mensagens com instâncias de **User** e é responsável pela lógica de criação, remoção e gestão de salas, sendo estas adicionadas/removidas do **RoomSupervisor** simultaneamente. Uma das funcionalidades mais importantes da aplicação é a possibilidade da criação de salas privadas apenas disponíveis a um grupo de utilizadores definidos. Este utilizador possui uma **MailBox** da mesma proporção que o **UserManager**, devido a ser provavelmente o tipo de actor com maior atividade de recepção mensagens.

Room

Cada sala é um actor (instância) **Room**, que possui a informação referente aos utilizadores que estão nessa sala. Cada sala, similarmente aos utilizadores, é identificada por um nome, sendo o **RoomManager** responsável por mapear o nome das salas à referência de cada uma. Cada vez que o administrador remove uma sala, essa sala é responsável por notificar todos os seus utilizadores que vai deixar de existir. Após todos os utilizadores estarem notificados, o fiber responsável pelo actor **Room** termina a execução.

Consola de notificações

Para a obtenção de eventos a ocorrer no sistema, foi criado um cliente independente que abre uma conexão, através de sockets **ZeroMQ**, com o ator responsável pela publicação de eventos a decorrer no servidor principal.
Os eventos considerados são eventos diretamente relacionados com utilizadores e/ou salas (login/logout, utilizador entrou/saiu da sala, novo utilizador/sala entre outros).
O ator responsável pela publicação de eventos, como supracitado, é uma instância de **EventPublisher.** Sempre que um evento acontece, o(s) ator(es) responsáveis pela lógica desses eventos, enviam uma mensagem para o **EventPublisher** com o tipo apropriado, o qual é encarregado de anunciar estes eventos para o próprio servidor assim como para o socket **ZeroMQ** para que qualquer cliente que esteja subscrito a estes possa também receber as notificações.
O publisher, utiliza uma porta diferente do servidor, incrementando a porta deste último por 1, tornando o tráfego entre Servidor↔End User e Servidor↔Notification Console mais independente um do outro e eliminando conflitos de *'bind'* a portas que já estão em uso pela aplicação.
Tendo em conta esta informação, na aplicação **Notification Console**, o utilizador pode explicitar inicialmente a porta utilizada pelo **EventPublisher** do servidor de mensagens que se pretende supervisionar. A execução da **Notification Console** representa uma instância de **Main** onde é definida a classe **InputReader.** A classe **Main** contém a lógica responsável pela conexão ao socket **ZeroMQ** e posterior receção e apresentação de mensagens do mesmo.
A classe **InputReader,** uma thread inicializada na **Main**, é a principal responsável pela interação com o utilizador, recebe e processa diferentes tipos de comandos e executa as ações correspondentes. Esta classe é também responsável por:

- Manter a lista de subscrições atuais do utilizador
- Manter a lista de tópicos fornecidos pelo servidor para o intuito da aplicação

Mensagens

Os atores trocam mensagens de diversos tipos entre si, conforme as necessidades de informação que cada uma deva conter:

- **RetrievableMessage**
  - Contém os campos "type", "object" e "sender".
  - Utilizada quando existe a necessidade de responder a quem enviou.
  - Maioritariamente usada para enviar informação para os managers.
- **UserDataMessage**
  - Contém os campos "username", "userdata" e sender.
  - Utilizada quando existe a necessidade de enviar informação sobre um dado utilizador.
  - Usada para registo e login de utilizadores.

Supervisores

De forma a controlar e recuperar as falhas dos actores, foram criadas instâncias da classe **Supervisor** pertencente à *framework* **Quasar**. As instâncias criadas são:

- **UserSupervisor**
  - Responsável pelo controlo de execução dos actores da classe User
- **RoomSuperVisor**
  - Responsável pelo controlo de execução dos actores da classe Room
- **ManagerSupervisor**
  - Responsável pelo controlo de execução dos actores das classes UserManager e RoomManager.

Os supervisores seguem uma filosofia de recuperação de **ONE_FOR_ONE,** sendo que apenas recuperarão um actor no caso de este tiver acabado por causas não naturais. Efetuam um máximo de 10 tentativas de recuperação do actor em intervalos de 1 segundo.

Funcionalidades

Nesta secção são apresentadas as funcionalidades de ambas as partes do projeto, i.e., os comandos através dos quais o utilizador é capaz de interagir com o sistema.

Chat Server

Esta aplicação fornece os seguintes comandos (argumentos em sublinhado):

Comando	Descrição
!changeroom NomeSala	Modificar a sala na qual o cliente está a escrever para NomeSala
!enterroom <NomesSalas>	Adicionar o utilizador atual às salas da lista NomesSalas
!leaveroom <NomesSalas>	Remover o utilizador atual da salas da lista NomesSalas
!login username password	Login do utilizador username com a respetiva password
!logout	Logout do utilizador autenticado atual
!register username password	Registar o utilizador username e fazer login
!listmyrooms	Listar as salas em que o utilizador se encontra
!createprivateroom NomeSala <NomeUsers>	Criar sala privada cujo nome é NomeSala e cujos utilizadores são da lista NomeUsers
!createroom <NomesSalas>	Criar salas públicas com os nomes da lista NomesSalas
!removeroom <NomesSalas>	Remover as salas da lista NomesSalas
!listrooms	Listar todas as salas disponíveis
!listusers	Listar todos os utilizadores online
!listroomusers	Listar todos os utilizadores online na sala atual
!sendpm @<NomesUsers> msg	Enviar mensagem privada para utilizadores com username da lista NomesUsers com o texto msg

Tabela 1 - Comandos disponíveis para interagir com o Chat Server

Consola de notificações

Esta aplicação fornece os seguintes comandos (Prefixo^+ expressão regular sobre Prefixos explícitos em baixo):

Comando	Descrição
commads	Exibe a lista de comandos existente
subscribe Prefixo^+	Subscreve ao(s) tópico(s) de prefixo especificado(s)
Unsubscribe Prefixo^+	Remove subscrição do(s) tópico(s) de prefixo especificado(s)
topics	Exibe a lista de tópicos fornecidos pelo servidor
status	Exibe a lista de subscrições atuais

Tabela 2 - Comandos disponíveis na Consola de notificações

O **ZeroMQ** fornece a noção de tópicos para as mensagens, um prefixo que identifique diferentes conteúdos para que os utilizadores possam filtrar (subscrever) mensagens de interesse. Assim os tópicos fornecidos pelo servidor usam os seguintes prefixos para subscrições:

Prefixo	Eventos que a consola exibe
@<username>	Todos relacionados especificamente com o utilizador username.
@login	Relacionados com login e logout.
@user	Relacionados com criação e remoção de utilizadores.
@private_room_management	Relacionados com criação e remoção de quartos privados.
@public_room_management	Relacionados com criação e remoção de quartos públicos.
@room_management	Relacionados com criação e remoção de quartos.
@private_room_users	Relacionados com utilizadores a entrarem em salas privadas.
@public_room_users	Relacionados com utilizadores a entrarem em salas públicas.
@room_users	Relacionados com utilizadores a entrarem em qualquer tipo de sala.
@all	Todos acima mencionados excepto @<username> por razões óbvias.

Tabela 3 - Prefixos para os tópicos fornecidos pelo servidor

Os utilizadores desta consola podem a qualquer momento consultar a lista de tópicos fornecidos pelo servidor, subscrever ou remover subscrição a qualquer tipo de tópico fornecido e consultar os tópicos aos quais se encontra atualmente subscrito.

Fluxo de mensagens

Nesta secção são apresentados diagramas de sequência representativos dos comandos mais generalizados, isto é, o conjunto de comandos que demonstra na sua completude as capacidades do sistema e a troca de mensagens efetuada entre cada componente.

Chat Server – Criar salas

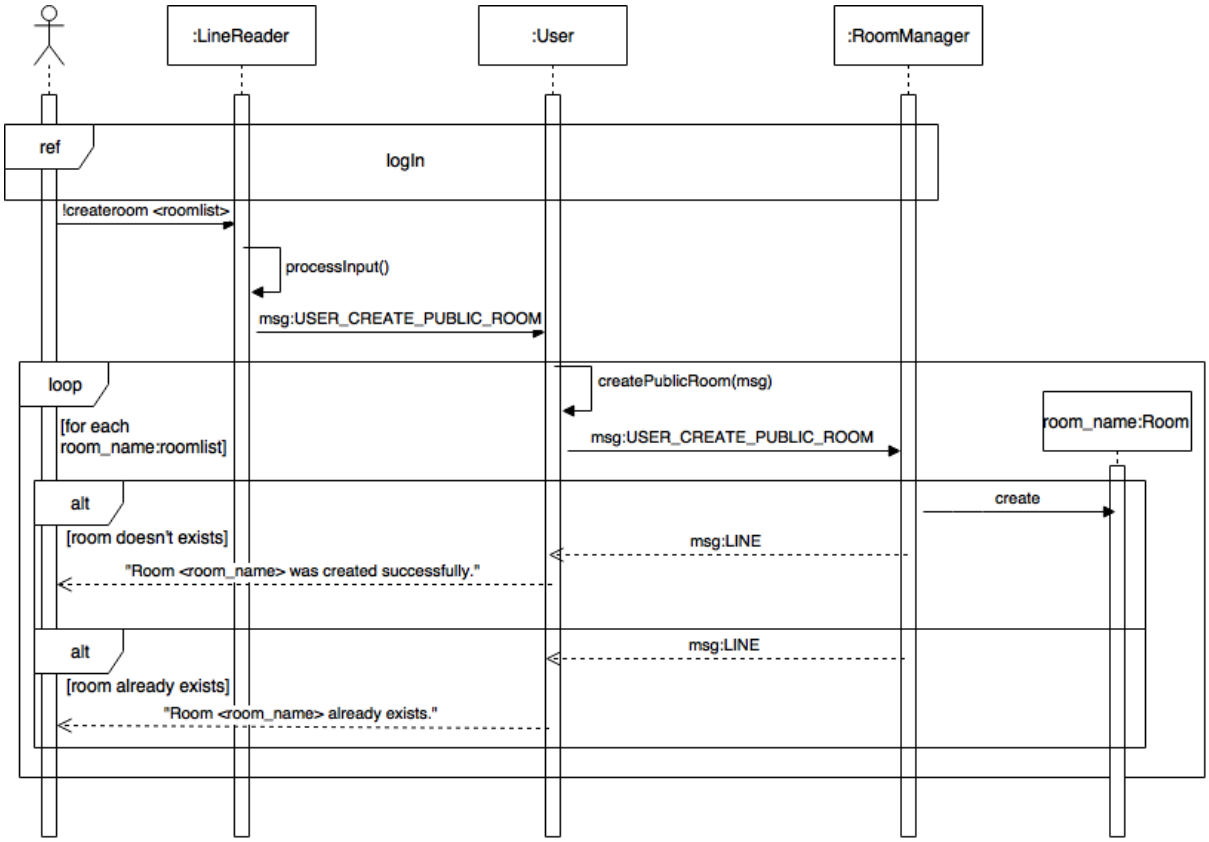

Figura 2: Fluxo de mensagens na criação de salas (Room).

De forma a poder criar salas, um dado utilizador deve estar registado e autenticado, sendo que é possível este ser um utilizador normal ou um administrador. Para cada nome passado como argumento no comando **!createroom** será criada uma instância de actor **Room**, através de uma mensagem enviada do **User** para o **RoomManager,** este último responsável por efetuar a instanciação. Após a tentativa de criação de sala, para cada sala passada como argumento o utilizador receberá uma mensagem informativa do sucesso ou insucesso da acção.

Chat Server – Listar salas

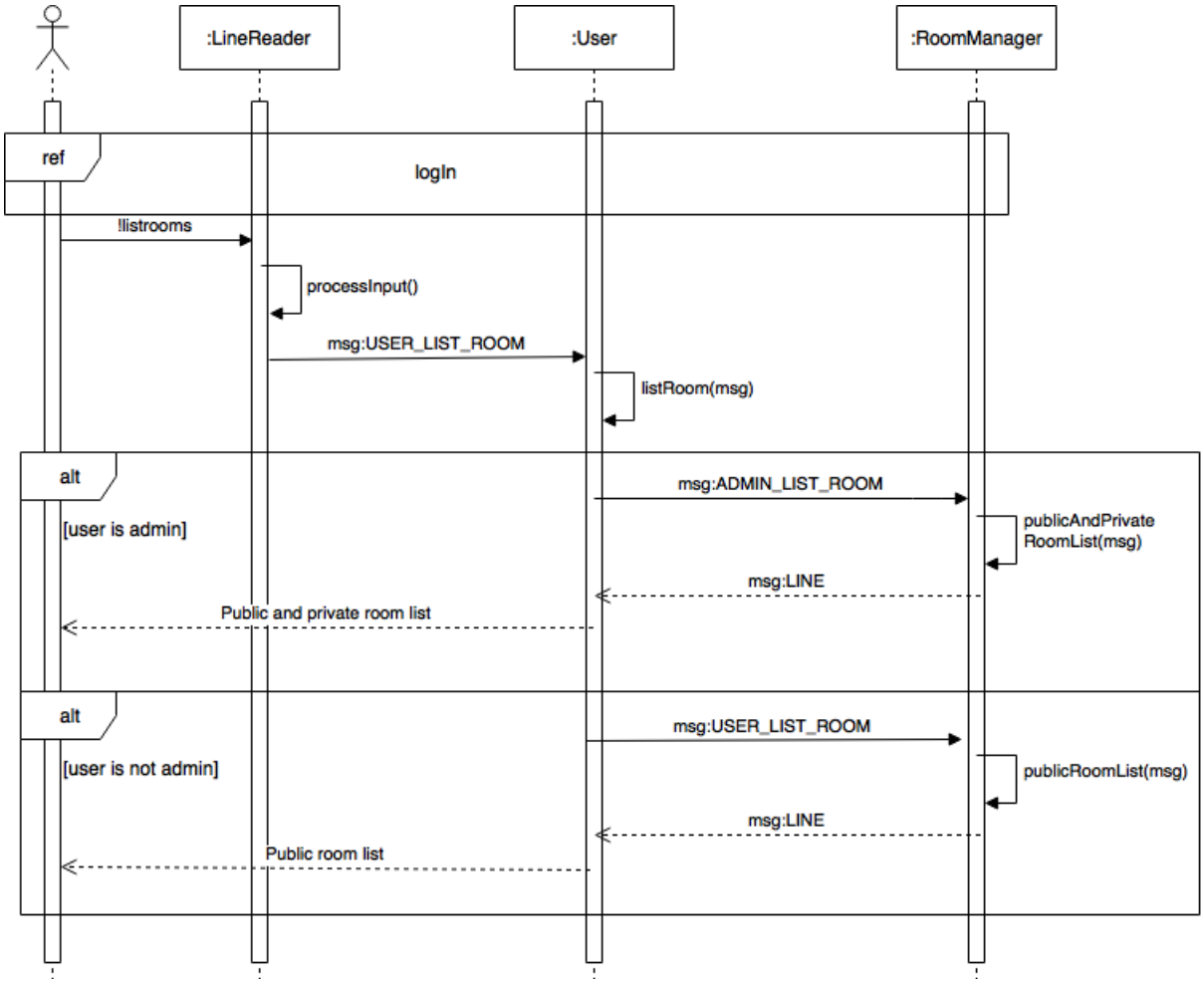

Figura 3: Fluxo de mensagens na listagem de salas (Room).

De forma a poder receber uma lista das salas disponíveis, um dado utilizador deve estar registado e autenticado. O **RoomManager** contém toda a informação acerca das salas privadas e públicas, sendo que para um utilizador normal apresenta a informação sobre todas as salas públicas disponíveis até ao momento, e todas as salas privadas às quais o utilizador possui acesso. Se o utilizador possui privilégios de administrador, ser-lhe-ão apresentadas todas as salas públicas e privadas disponíveis.

Chat Server – Efetuar registo de utilizador

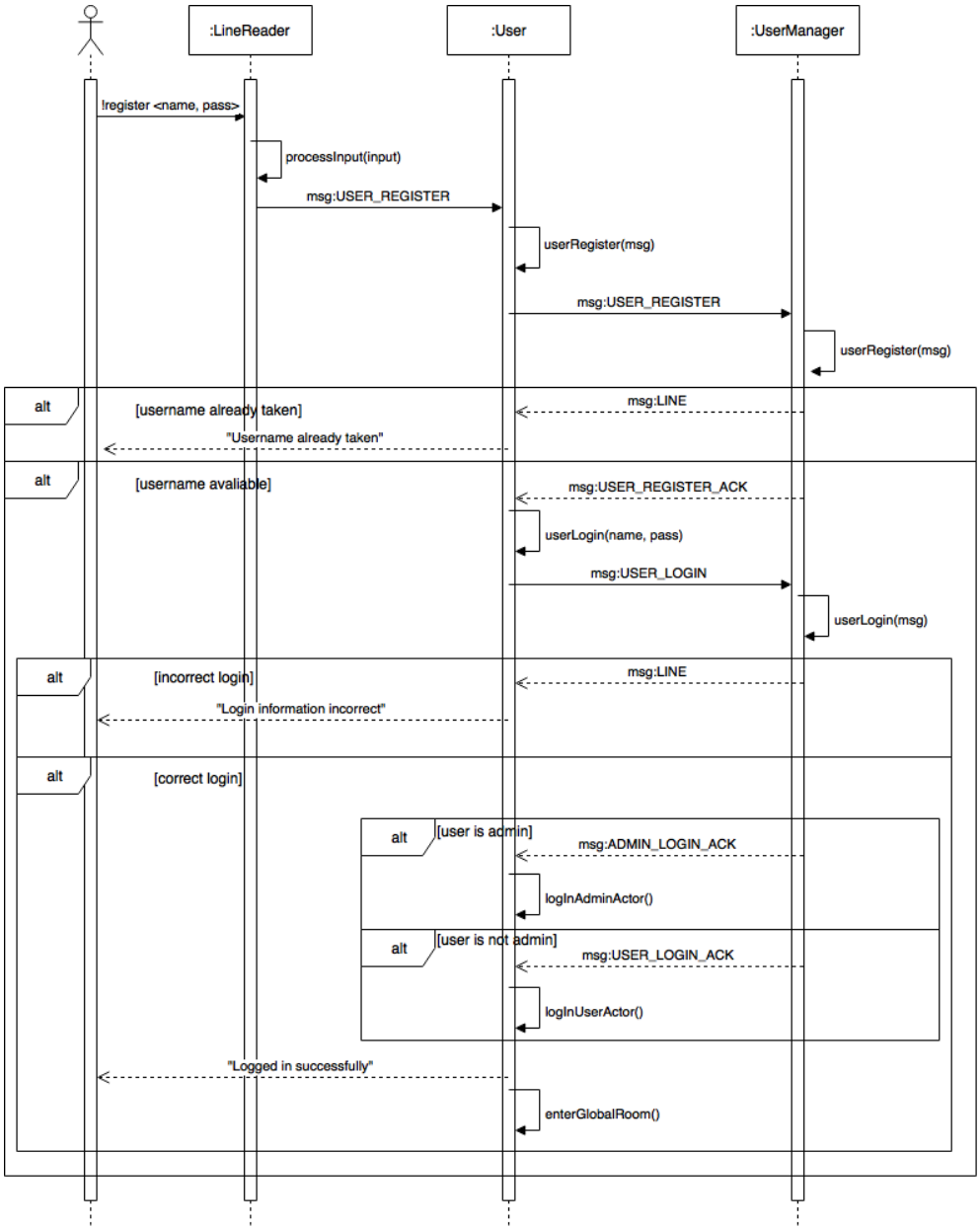

Figura 4: Fluxo de mensagens no registo de utilizadores.

Um dado utilizador deve estar registado e autenticado de forma a poder interagir com a aplicação, sendo que não pode registar-se nem autenticar-se se já tiver uma sessão autenticada. O registo é efetuado através do envio de uma mensagem com as credenciais do utilizador para o **UserManager**. Caso as credenciais sejam válidas, o **User** recebe um "ack" de registo com sucesso, procedendo ao envio de uma mensagem de login automaticamente. Um processo análogo é efetuado para o login, sendo que em caso de sucesso, o **User** recebe um novo "ack" e envia uma nova mensagem para o **RoomManager** de forma a entrar na sala de chat geral.

Chat Server – Enviar mensagem privada

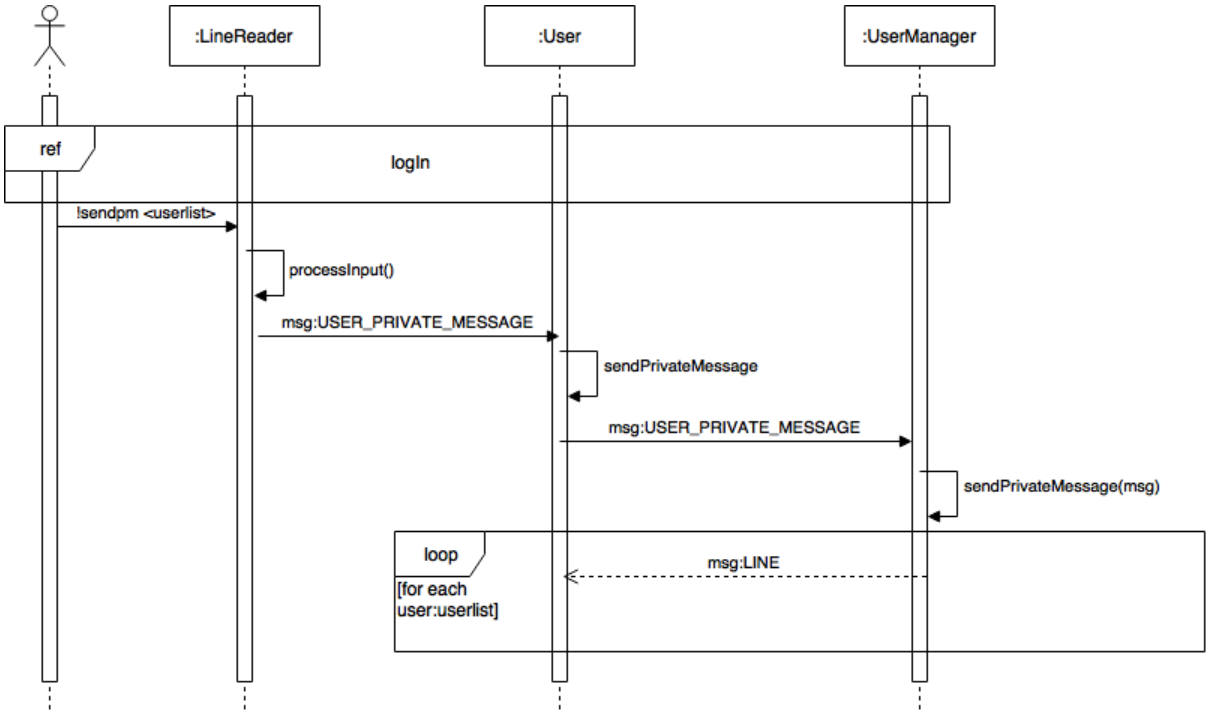

Figura 5: Fluxo de mensagens no envio de mensagens privadas.

A lógica do envio de mensagens privadas pertence ao **UserManager,** dado que este possui toda a informação acerca dos utilizadores que se encontram autenticados. O utilizador requer o envio de uma dada mensagem de texto para uma lista de outros utilizadores, e esta mensagem é enviada pelo **User** para o **UserManager**, que iterativamente a distribui pela lista de utilizadores referenciados na mensagem. As mensagens de texto enviadas para os utilizadores possuem sempre o tipo "*LINE*".

Chat Server – Entrar em sala

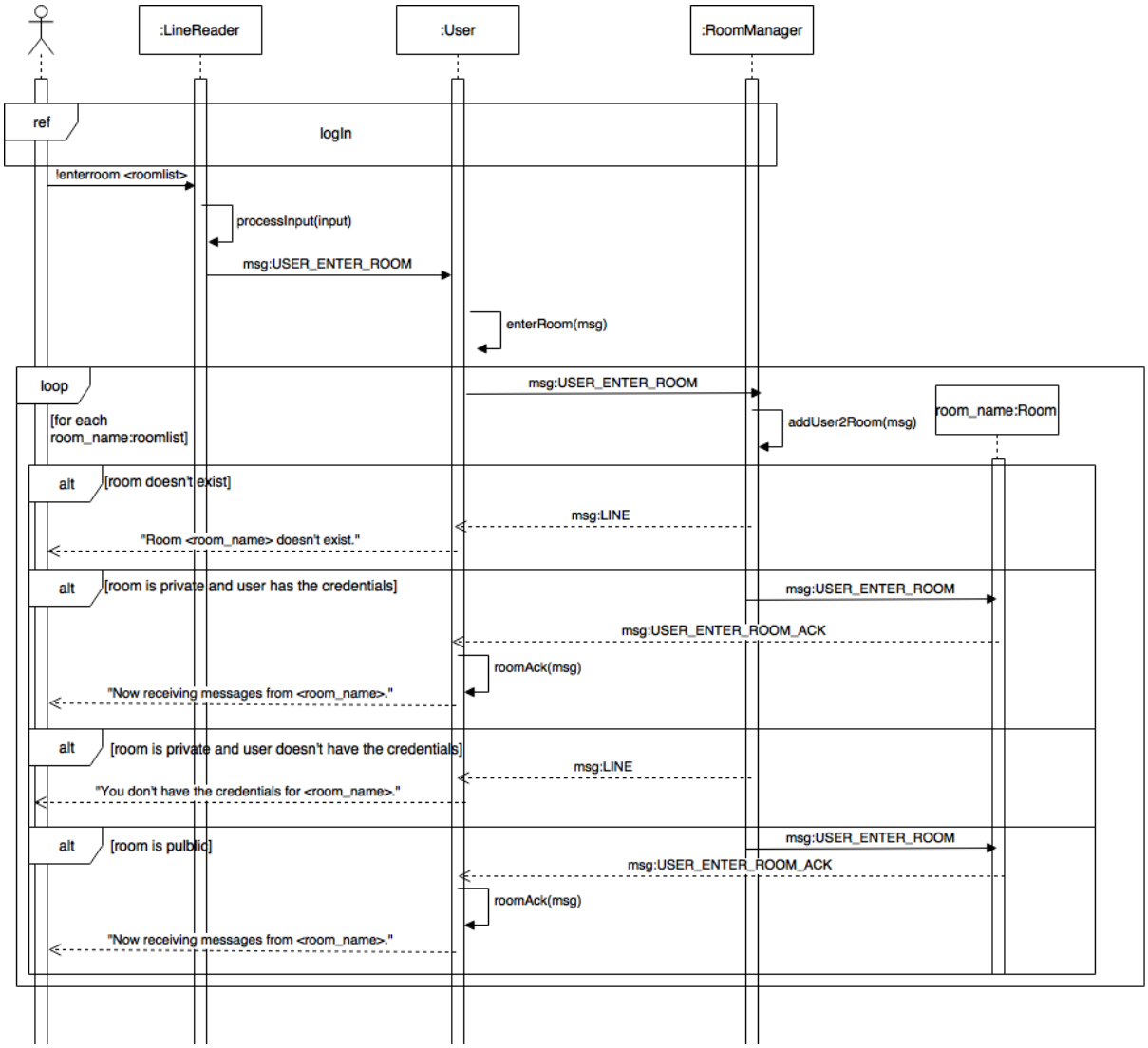

Figura 6: Fluxo de mensagens na entrada de salas (Room).

Um aspecto fulcral na lógica da aplicação é o sentido de um utilizador poder "ouvir" de várias salas ao mesmo tempo, mas apenas pode escrever para uma de cada vez, sendo que para receber as mensagens de uma sala, deve entrar nela. Cada sala (**Room**) possui a informação referencial dos utilizadores presentes nesta. Para um dado utilizador ser capaz de entrar na sala, é enviada uma mensagem ao **RoomManager** de forma a averiguar se a sala existe e se é possível aceder a ela. Caso isto se verifique, o **RoomManager** envia a informação do utilizador para a dada instância de **Room**, sendo esta responsável por apresentar-se ao utilizador de forma a este poder associar referencialmente a sua localização (através do envio de um "ack" de entrada na sala). No caso de a sala não existir ou o utilizador não possuir as credenciais para aceder a esta, o **RoomManager** envia uma mensagem de texto ao **User** informativa da sua situação.

Consola de notificações – Subscrever a evento

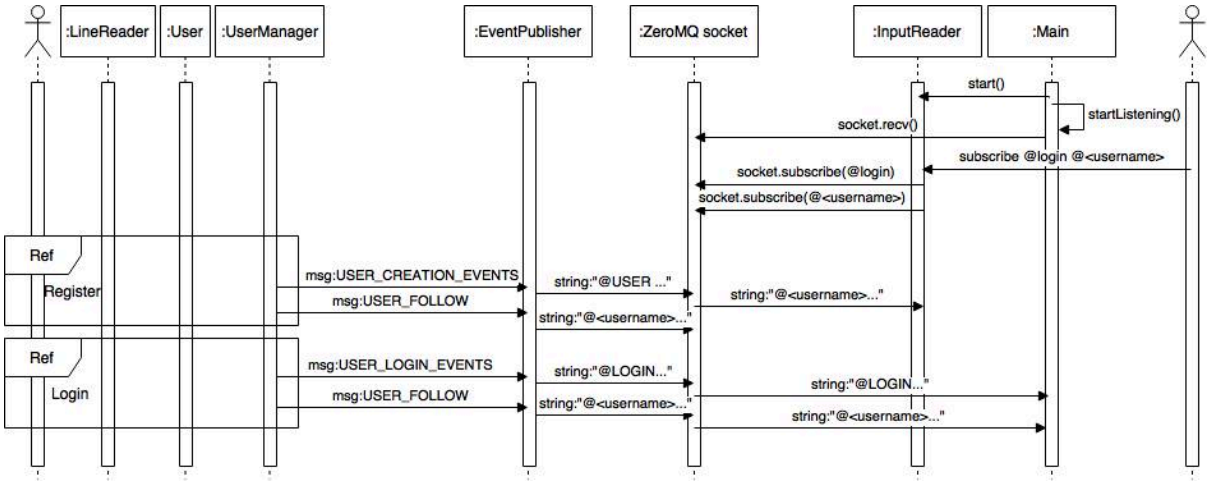

Figura 7: Fluxo de mensagens na subscrição de eventos.

Neste exemplo temos 2 intervenientes: o **end user** (lado esquerdo) e **notification console** (lado direito) que executam em paralelo. A **Notification Console** ao executar conecta-se ao socket **ZeroMQ** do qual fica à escuta e inicializa uma thread do tipo **InputReader** responsável por satisfazer os pedidos do utilizador.
Neste exemplo, esta thread recebe um comando do cliente da **Notification Console** "subscribe @login @<username>", o **InputReader** trata o comando e envia o pedido de subscrição de eventos de login e de todos os eventos relacionados com o utilizador "username" para o socket **ZeroMQ**.
No lado do End User este regista-se corretamente e como tal é automaticamente feito o login. Esta ação leva a 4 eventos distintos que são enviados para o socket ZeroMQ:

- Criação de um utilizador (USER_CREATION_EVENTS), tópico @USER
- Criação de um utilizador específico (USER_FOLLOW), tópico @<username>
- Login de um utilizador (USER_LOGIN_EVENTS), tópico @LOGIN
- Login de um utilizador específico (USER_FOLLOW), tópico @<username>

Como anteriormente o utilizador da notification console fez subscrição aos tópicos @LOGIN e @<username>, está à escuta destes no socket ZeroMQ, e assim recebe 3 diferentes notificações de eventos (não 4 porque não está subscrito aos eventos de tópico @USER):

```
1. @LOGIN [13:30:6]> username has logged in.
2. @<USERNAME> [13:30:6]> has logged in.
3. @<USERNAME> [13:30:6]> joined the public room global_room.
```

Todos os eventos são tratados desta mesma forma:

- Consola de notificações coloca-se à escuta de eventos
- Cliente consola de notificações subscreve a tópicos de interesse
- Eventos acontecem no sistema do servidor
- Servidor publica todos os esses eventos
- Cliente da consola de notificações recebe eventos correspondentes aos tópicos de interesse

Pensamentos Finais

Finalmente é relevante mencionar as decisões que levaram a desenvolver a aplicação seguindo a arquitetura apresentada e melhorias que podem ser realizadas sobre esta.
Relativamente à aplicação que funciona como servidor de mensagens, esta segue uma arquitetura muito "hierárquica", sendo que componentes maiores são sucessivamente responsáveis por componentes menores, aumentando a abstração de dados entre as diversas partes. Este foi um ponto importante que a lógica de contenção de dados dos actores nos influenciou a seguir, e levou-nos a focar mais na troca de mensagens entre actores como forma de obter informação do que na contenção da informação em si. O desenvolvimento permitiu-nos explorar as funcionalidades de supervisores fornecidas pela framework **Quasar**, que apesar de simples, revelou-se ainda "*buggy*" e num estado de desenvolvimento muito precoce. **Quasar** fornece também actores do tipo **Event Source** que infelizmente não tivemos a oportunidade de implementar dado que a funcionalidade fornecida por estes era já abordada por **ZeroMQ**.
Creemos que a lógica de leitura e escrita de mensagens poderia ter sido abordada de forma diferente entre **User** e **Managers**, com a criação de workers que intermediassem a operação. Isto simplificaria os dados contidos nas mensagens enviadas e possivelmente facilitaria o processamento destas. Seria também interessante implementar esta aplicação seguindo um padrão de "proxy server", no qual cada **Room** e **User** seria responsável pela sua informação, e existiria um servidor que estabeleceria a conexão entre estes dois tipos de componentes.
Apesar de que a ideia inicial ao desenvolver a consola de notificações era incorporá-la na aplicação de servidor de chat, esta ideia tornou-se um desafio estruturalmente demasiado complexo, dado que esta seria efetuada obrigatoriamente recorrendo às funcionalidades oferecidas por **ZeroMQ**. A gestão de portas com as quais as sockets se associariam era demasiado contraintuitiva, e creemos que desenvolver um cliente à parte foi uma decisão acertada.
A consola de notificações poderia ser melhorada de modo a obter os tópicos disponíveis em cada servidor através dos próprios e não '*hardcoded'* em si; e ser capaz de supervisionar diferentes servidores numa só execução. Para esta última implementação seria necessário alterar os tópicos das mensagens enviadas de modo a identificar unicamente os servidores de origem e deste modo também identificar unicamente os eventos para não haver conflitos de salas ou utilizadores com o mesmo nome em servidores diferentes.
Em forma de conclusão, pensamos ter desenvolvido uma aplicação escalável e funcional que cumpre os requisitos propostos (e mais), e na qual integramos com sucesso duas tecnologias diferentes.