UNI
FR

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

SALONS DE DISCUSSION DISTRIBUES - ERLANG

Systèmes Concurrents et Distribués

18 DECEMBRE 2017

JULIEN CLEMENT, JOHAN JOBIN
Université de Fribourg

Table des matières

I. Description du problème

Ce projet repose sur l'implémentation distribuée de salons de discussion et de clients – à la fois des bots et des humains – en langage Erlang. Le défi principal est de garantir un fonctionnement stable du système s'appuyant sur l'exclusion mutuelle sans que cela n'impacte les utilisateurs. En effet, le nombre de requêtes peut rapidement devenir important car les clients ont la possibilité de changer de salon de discussion et d'interagir avec les autres clients en envoyant et en recevant des messages. De ce fait, il est essentiel que toutes les requêtes soient correctement traitées, et ce, en gardant un ordre cohérent d'arrivée des messages dans les salles de discussion.

Afin de vérifier empiriquement l'efficacité de la solution décrite dans ce projet, le système offre la possibilité de définir le nombre de salons de discussion et de bots qui interagiront lors du démarrage du serveur. Le nombre de salons de discussion est limité au nombre d'ordinateurs disponibles sur teDA[1] afin qu'ils puissent tous fonctionner d'une façon parfaitement distribuée. Cela permet notamment de prendre en compte certains facteurs qui influent sur le comportement du système comme la vitesse de propagation des messages. En revanche, il n'existe aucune restriction concernant le nombre de bots, de sorte à réellement pouvoir mettre le système dans une situation ardue. Pour ce faire, les bots fonctionnent sur des ordinateurs différents tant que cela est possible, puis dans différents processus sur un unique ordinateur lorsqu'il n'y en a plus d'autres à disposition sur le réseau teDA.

En outre, il est également possible de se connecter manuellement depuis son propre ordinateur en utilisant le client Telnet, de s'ajouter dans un salon de discussion et de discuter avec d'autres clients ou avec les bots.

II. State-of-the-art

L'idée de salons de discussion existe depuis très longtemps sur internet. Les services les plus connus exploitent le protocole IRC (Internet Relay Chat) développé à la fin des années 80. On peut notamment citer IRCnet ou EFnet. En se penchant plus particulièrement sur les plateformes de chat utilisant le langage Erlang dans leur

[1] teDA : "test and evaluation for distributed algorithms". Réseau de machines du CDC-Lab de l'Université de Fribourg sur lequel sont déployés à distance les programmes Erlang.

implémentation, on remarque que des services comme WhatsApp sont implémentés dans ce langage. Il est possible d'établir un parallèle entre les idées avancées dans ce projet et ces services en ligne : chaque discussion entre 2 personnes (privée) ou entre N personnes (groupe) peut être vue comme un salon de discussion. Erlang est apprécié pour ses qualités de parallélisme et de communication. En revanche, certains désavantages propres à Erlang ont poussé de grandes entreprises comme Facebook à passer d'Erlang vers d'autres langages pour leur système de chat.

III. Solution

1. Architecture

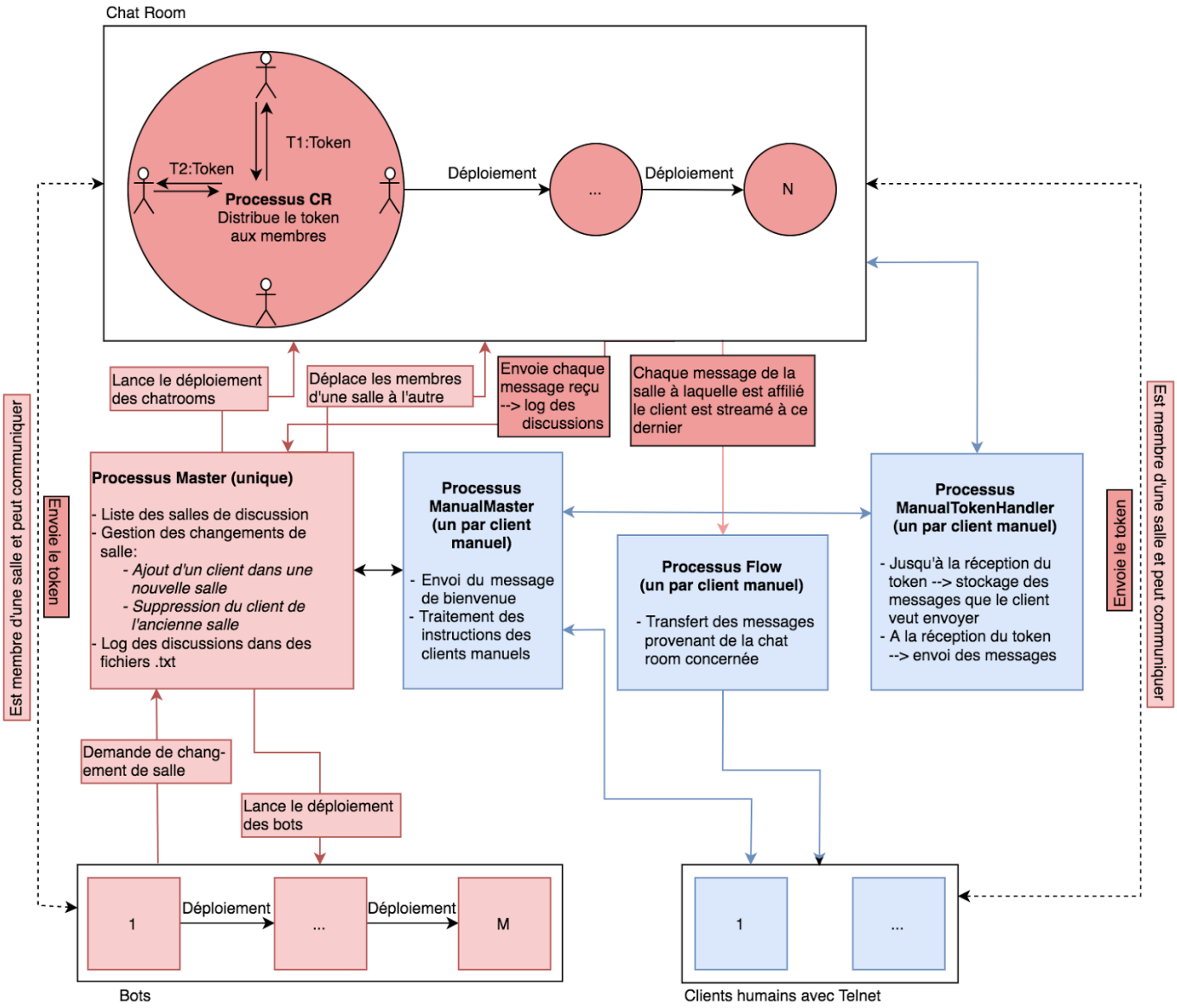

Figure 1 : Schéma de l'architecture

2. Détails d'implémentation

Salles de chat et bots : déploiement

L'utilisateur a le choix du nombre de chat rooms et du nombre de bots qu'il souhaite faire interagir. La première étape consiste donc à déployer les chat rooms. Afin de pouvoir tester le système de manière réelle sous des contraintes typiques liées à la programmation distribuée (comme le temps nécessaire à l'envoi de messages), les chats rooms se déploient impérativement sur des ordinateurs différents. Ainsi, le nombre de chat rooms doit être strictement plus petit que le nombre d'ordinateurs disponibles. Le processus principal déploie la première salle de discussion avant de lui envoyer un message pour l'informer qu'elle doit procéder au déploiement de la prochaine salle. Cette phase est donc une réaction en chaîne qui se termine par l'envoi de la liste des PIDs des salles de chat au processus principal, confirmant le bon déroulement de l'opération.

Une fois cette étape terminée, le processus maître peut lancer le déploiement des bots. Comme les bots ne communiquent pas directement ensemble, il n'existe pas de restriction quant à leur nombre. Dans le meilleur des cas, ils sont tous déployés sur des ordinateurs différents. Lorsque tous les ordinateurs disponibles ont été utilisés, le reste des bots est déployé sur le même ordinateur dans des processus différents. La liste des PIDs de ceux-ci est également retournée au processus principal afin de lui indiquer la fin de la phase de déploiement.

Clients humains : déploiement

Une fois la partie automatique en fonction, le processus principal crée un deuxième processus - ManualMaster - qui se charge d'attendre des connexions entrantes sur un port prédéfini (8070). Les communications s'effectueront à l'aide du protocole TCP, garantissant l'intégrité des données grâce au système d'accusé de réception du protocole.

Lorsqu'un client se connecte au serveur, le processus chargé d'attendre une connexion envoie un message de bienvenue et attend les instructions provenant du client.

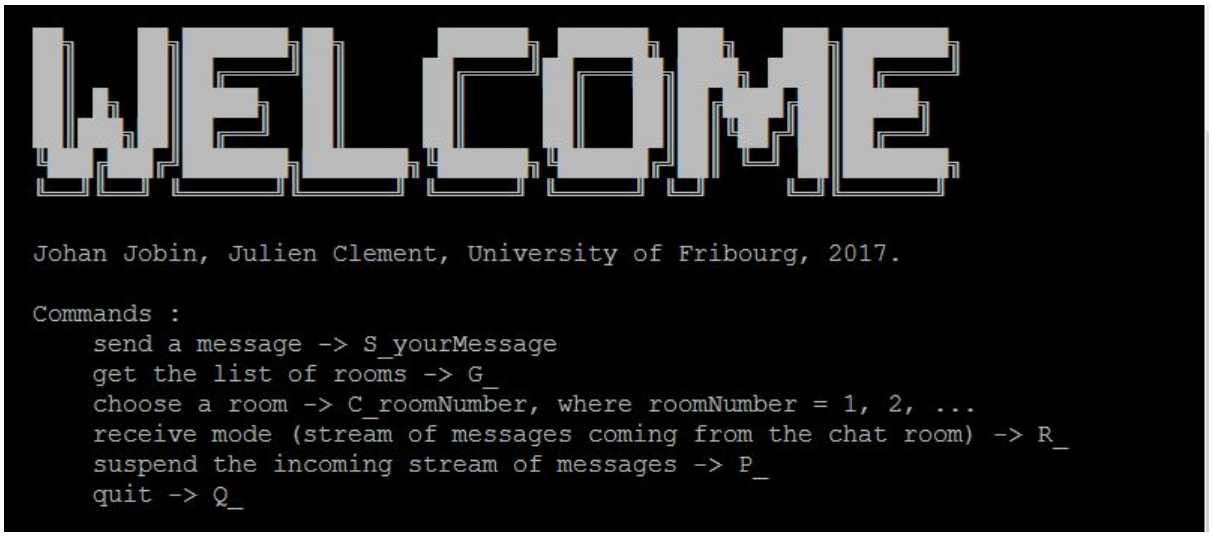

Figure 2 : Message d'accueil envoyé par le serveur

En addition à ce processus, un autre processus parallèle appelé “Flow” est généré pour chaque client. Son rôle sera de lui redistribuer les messages arrivant dans la chat room de sorte à ce qu’il suive la discussion en direct. Finalement, un troisième processus “ManualTokenHandler” est encore nécessaire pour agir comme queue de messages à envoyer, qui se vide lorsque le client reçoit le token.

Processus master

Le processus master est le cerveau du système. Son rôle principal est de coordonner les opérations afin que les changements de salle se déroulent sans encombre. En effet, lorsqu’un bot ou un client désire entrer dans une nouvelle salle de discussion, il lui fournit une liste de toutes les salles disponibles. Le client fait son choix avant d’en informer le master qui s’occupera d’avertir l’ancienne salle de chat du départ du client ainsi que de l’arrivée de celui-ci dans la nouvelle.

Il sert également à tenir un journal des discussions et des entrées/sorties pour chaque salle de chat. Lorsqu’une salle reçoit un message, elle l’envoie au processus master avec plusieurs informations comme la date et l’heure d’envoi ainsi que l’expéditeur du message. De ce fait, il est possible de vérifier que tout fonctionne correctement en suivant les faits et gestes des clients (voir section « Logs » p.7).

Salles de chat : fonctionnement

En plus de permettre au processus master d’ajouter et de retirer un membre, les salles de chat s’occupent de donner la parole à chacun des membres à l’aide d’un token. Le

fonctionnement détaillé est décrit à la page 6. Comme mentionné dans le paragraphe précédent, elles envoient également chaque nouveau message reçu au processus master de sorte à avoir un historique des conversations et des mouvements des clients.

Bots : fonctionnement

A sa création, et une fois le token reçu, un bot n'a d'autre choix que de sélectionner aléatoirement une salle de discussion parmi celles disponibles. A partir de moment-là, il peut effectuer deux opérations à intervalle aléatoire lorsqu'il reçoit le token : envoyer un message dans la salle de chat (probabilité de 90%) ou changer de salle (probabilité de 10%). Cela permet de créer un scénario correspondant à une situation réelle où tous les clients seraient humains. Pour l'envoi de messages, le bot peut piocher une phrase aléatoirement parmi les phrases prédéfinies dans le fichier "phrases.txt". Ce fichier, se trouvant dans "teda_light/apps/chat", est modifiable à la convenance de chacun, à condition d'entourer la phrase de guillemets et d'ajouter un point après les deuxièmes guillemets.

ManualMaster, Flow et ManualTokenHandler : fonctionnement

Il existe une instance de ces trois processus pour chaque nouveau client humain. Une fois les étapes expliquées au point "Déploiement : clients humains" terminées, le processus "ManualMaster" se charge de gérer les requêtes des clients selon les règles suivantes :

Commande	Description	Exemple
G_	Demande au "master" de lui renvoyer la liste de toutes les salles de discussion disponibles.	G_
C_*Nombre*	Permet au client humain de sélectionner la salle de discussion dans laquelle il aimerait rentrer en indiquant simplement son numéro ($\in$ [1, Nb. de chat rooms]).	C_3
S_*Message*	Lorsque le client est affilié à une salle de discussion, il peut envoyer des messages dans la conversation à l'aide de cette commande.	S_Salut
R_	Mode permettant de recevoir et d'afficher le flux de la discussion instantanément. De plus, lorsque la commande est appelée après avoir appelé P_, tous les messages qui ont été mis en attente pendant ce laps de temps sont affichés.	R_
P_	Mode qui met la réception du stream de la discussion en pause. Il est ainsi plus facile de taper de nouvelles commandes.	P_
Q_	Quitte le serveur et ferme les processus liés au client concerné.	Q_

Figure 3 : Requêtes du client humain

Le client a la possibilité d'effectuer ces requêtes à tout moment. Elles seront toutes traitées instantanément, à l'exception de *S_Message* qui mettra l'envoi du message en attente tant que le client n'a pas reçu le token de la part de la salle de chat. Cette fonction est gérée par le processus ManualTokenHandler qui permet de stocker dans une queue tous les messages envoyés lorsque le client humain n'est pas en possession du token.

De son côté, Flow fonctionne de manière unidirectionnelle. Son rôle est simplement de transmettre tous les messages arrivant dans la salle de chat vers son client humain via le protocole TCP, dans le but de permettre au client de suivre la discussion.

3. Problème principal : exclusion mutuelle

Système de token

Un système de token a été implémenté afin de garantir l'exclusion mutuelle au sein des salles de discussion et la cohérence d'ordre d'arrivée des messages. Le token est implémenté comme un entier compris entre 1 et le nombre de membres de la salle de discussion, correspondant à sa place dans la liste de membres (le premier arrivé recevra "1", le deuxième "2", etc.). Il représente ainsi le droit d'effectuer une action et fait des allers-retours entre la salle de discussion et ses membres. Une fois que le token a été passé à tous les membres, il est réinitialisé à un 1 puis est à nouveau distribué entre les membres pour un nouveau tour. Ce sont les salles de discussion qui se chargent de distribuer le token à tour de rôle à chaque membre car chacune d'elles possède une liste contenant leur PIDs. Il est nécessaire que le token soit renvoyé par le bot/client qui l'a reçu afin qu'il puisse être redistribué au prochain.

Bots/Clients et token

Un bot attend d'être en possession du token pour décider s'il souhaite envoyer un message dans le salon de discussion ou changer de salon.
Un client humain peut quant à lui prendre la décision d'envoyer un message ou de changer de salle à tout moment. Dans la première situation, son message est mis en attente et sera envoyé à la salle de chat aussitôt qu'il aura reçu le token. Dans la

deuxième situation, la requête est traitée immédiatement. Dans tous les cas, le token est renvoyé au salon qui le lui a donné.

Logs

Un log de discussion est créé sur l'ordinateur affecté au processus master pour chaque salle de chat. Ces fichiers se trouvent dans le dossier "mpe/erl/apps/chat" sur l'ordinateur distant et suivent la nomenclature suivante : "Chatroom<PID>". En les consultant, il est possible de suivre les mouvements des bots/clients et les messages envoyés dans la salle de chat (avec la date et l'heure d'envoi ainsi que le PID du bot/client).

```
------- <6471.40.0> enters Room


------- <6471.38.0> enters Room


------- <6471.41.0> enters Room


------- <6469.38.0> enters Room
{{2017,12,17},{17,0,35}}, <6471.40.0> : "I am counting my calories, yet I really want dessert."
{{2017,12,17},{17,0,39}}, <6471.38.0> : "She was too short to see over the fence."


------ <6471.39.0> enters Room ----- From Room <6468.38.0>
{{2017,12,17},{17,0,45}}, <6471.41.0> : "The clock within this blog and the clock on my laptop are 1 hour different from each other."


------ <6470.38.0> enters Room ----- From Room <6468.38.0>
{{2017,12,17},{17,0,49}}, <6469.38.0> : "Wow does that work?"
{{2017,12,17},{17,0,54}}, <6471.39.0> : "I love eating toasted cheese and tuna sandwiches."
{{2017,12,17},{17,1,0}}, <6470.38.0> : "She was too short to see over the fence."
{{2017,12,17},{17,1,2}}, <6471.40.0> : "There were white out conditions in the town, subsequently, the roads were impassable."
{{2017,12,17},{17,1,8}}, <6471.38.0> : "He turned in the research paper on Friday, otherwise, he would have not passed the class."
{{2017,12,17},{17,1,14}}, <6471.41.0> : "I currently have 4 windows open up and I do not know why."
{{2017,12,17},{17,1,21}}, <6469.38.0> : "The old apple revels in its authority."
```

Figure 4 : Extrait d'un log d'un salon de discussion

IV. Evaluation de la solution

Afin de pouvoir tester le système de token, les scénarios de test ont été les suivants :

1. Lancement du serveur avec plusieurs bots au sein du même salon de discussion, dans le but de vérifier l'ordre de la prise de parole des bots et d'observer que le token était bel et bien réinitialisé lorsque tous les membres du salon avaient déjà parlé une fois.
2. Lancement du serveur avec plusieurs bots et plusieurs salons pour contrôler le fonctionnement du changement de salle. En effet, lorsqu'un bot change de salle, il doit renvoyer le token à la salle et être supprimé de la liste de ses membres.

De plus, il doit être ajouté à la liste des membres de la nouvelle salle en dernière position pour pouvoir recevoir le token et communiquer dans celle-ci.

3. Dans les scénarios précédents, les discussions étaient complètement simulées avec des bots. Par conséquent, ce troisième test consistait à connecter des clients humains avec Telnet et à les placer au sein des discussions afin de vérifier les éléments suivants :
   a. La connexion entre le serveur et le client humain à l'aide du protocole TCP.
   b. Le bon fonctionnement des commandes G_, C_, S_, P_, R_, Q_ (voir la figure 3).
   c. Pour le système de token, il ne doit il y avoir aucune différence entre un client humain et un bot : tous les clients doivent être traités de la même manière.
   d. Le stream en direct de la discussion grâce au processus "Flow".
   e. Les messages du client humain sont placés dans une queue si le message est envoyé lorsque celui-ci n'est pas en possession du token et ils sont envoyés dès l'instant où il le reçoit.

V. Discussion des résultats

Voici les résultats des différents tests présentés ci-dessus :

1. Les bots parlent les uns après les autres et le token est bien réinitialisé à chaque tour. Ainsi, la discussion garde un ordre cohérent.
2. Les changements de salle de discussion, peu importe quand ils surviennent, sont complètement transparents : le token continue à se passer entre les différents membres du salon.
3. Clients humains :
   a. La connexion TCP fonctionne, le menu principal est bien envoyé à quiconque essaie de se connecter au serveur.
   b. Toutes les commandes fonctionnent
   c. Lorsqu'un client manuel est inséré dans une salle de discussion, le processus ManualTokenHandler gère correctement le renvoi du token et des messages sortants en attente.
   d. Le processus Flow renvoie bien la discussion instantanément lorsque l'on est en mode "R_" et la garde dans un buffer lorsque le mode "P_" est choisi.

Lorsque l'on passe de "P_" à "R_" tout le buffer est envoyé au client humain, ce qui lui permet de voir l'entièreté de la conversation.

e. Lorsque le client humain envoie un message, il n'est envoyé que lorsqu'il reçoit le token. Cela est également valable lorsque l'on envoie plusieurs messages à la suite, sans avoir reçu le token.

VI. Bibliographie

State-of-the-art

IRC Networks - Top 100. 2017. [Consulté le 13.12.2017]. URL: http://irc.netsplit.de/networks/top100.php

METZ, Cade, 2015. WHY WHATSAPP ONLY NEEDS 50 ENGINEERS FOR ITS 900M USERS. *Wired.com*. 09.15.15. [Consulté le 10.12.17]. URL : https://www.wired.com/2015/09/whatsapp-serves-900-million-users-50-engineers/

MAURER, Ben. 2014. When did Facebook switch away from using Erlang for Facebook Chat? What was the reason for it? What did they replace it with?. 24.02.2014. [Consulté le 13.12.17]. URL: https://www.quora.com/When-did-Facebook-switch-away-from-using-Erlang-for-Facebook-Chat-What-was-the-reason-for-it-What-did-they-replace-it-with/answer/Ben-Maurer

Code

FARMER, Jesse. 2008. Network programming in Erlang. *20bits.com*. 02.05.2008. [Consulté le 24.11.17]. URL : http://20bits.com/article/network-programming-in-erlang

Buckets of Sockets. *Learnyousomeerlang.com.* [Consulté le 24.11.17]. URL : http://learnyousomeerlang.com/buckets-of-sockets

VII. Appendices

Utilisation

Pour déployer le programme sur teDA, il suffit de lancer le script “run_d.sh” se trouvant dans le dossier “Code” de l’archive. Afin que ce script fonctionne, il est impératif que le dossier “teda_light” se trouvant dans “Code” ne change pas de place ou de nom.

L’appel à la fonction nécessaire pour lancer le code suit le format suivant : “chat:start(R,C)”, où R représente le nom de chat rooms et C le nombre de clients. Les fichiers de log seront créés sur l’ordinateur spécifié sur cette même ligne, qui contient également un nom d’utilisateur à modifier.

```
#!/bin/sh

echo "cd ./teda_light/apps/chat"
cd ./teda_light/apps/chat
echo "../../scripts/ping.sh chat hosts.conf"
../../scripts/ping.sh chat hosts.conf
echo "../../scripts/depl_app.sh chat make"
../../scripts/depl_app.sh chat make
echo "../../scripts/depl_enodes.sh chat hosts_alive.conf"
../../scripts/depl_enodes.sh chat hosts_alive.conf
echo "../../scripts/run.sh chat 'chat:start(2,6)' hosts_alive.conf diufmac31.unifr.ch clemenju"
../../scripts/run.sh chat "chat:start(2,6)" hosts_alive.conf diufmac31.unifr.ch clemenju
```

Figure 5 : Script de déploiement. Exemple avec 2 salles de discussion et 6 bots.

Démonstration

Afin de limiter la taille de l’archive, une vidéo de démonstration est disponible en se rendant à l’adresse suivante : https://www.youtube.com/watch?v=TFUPdcj_4AU

Code

L’intégralité du code source est disponible dans le fichier « chat.erl » présent dans le dossier « Code » de l’archive.